Передовые инженерные школы

МИНОБРНАУКИ РОССИИ

УНИВЕРСИТЕТ ИННОПОЛИС

Занятие 16

Props и State

План занятия

ПЕРЕДОВАЯ
ИНЖЕНЕРНАЯ ШКОЛА
УНИВЕРСИТЕТА ИННОПОЛИС

1. Props
2. State

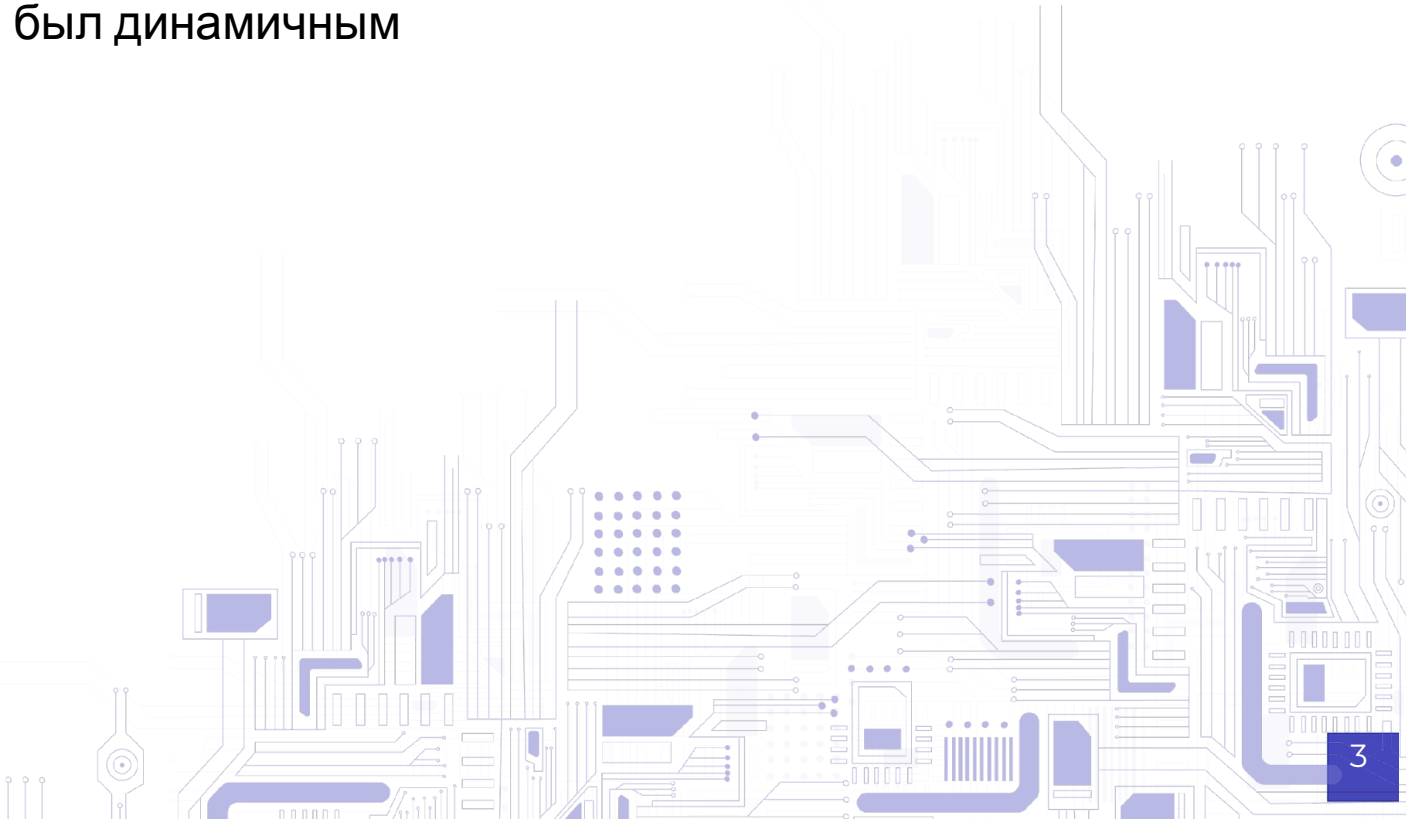
Props

ПЕРЕДОВАЯ ИНЖЕНЕРНАЯ ШКОЛА
УНИВЕРСИТЕТА ИННОПОЛИС

Поговорим о важной концепции React – Props, чтобы поддерживать динамичность потока данных в вашем приложении

Используем Props в React для передачи данных от одного компонента к другому (от родительского компонента к дочерним компонентам)

Props - это просто более короткий способ обозначения свойств
Они нужны, чтобы поток данных в приложении был динамичным



Props

Начнем с компонета

```
function App() {
  return (
    <div className="App">

    </div>
  )
}

export default App
```

Props

ПЕРЕДОВАЯ ИНЖЕНЕРНАЯ ШКОЛА
УНИВЕРСИТЕТА ИННОПОЛИС

Создадим другой компонент с именем Tool.js
Без props код будет выглядеть так:

Файл Tool.js

```
function Tool() {
    return (
      <div>
        <h1>My name is Ihechikara.</h1>
        <p>My favorite design tool is Figma.</p>
      </div>
    );
}

export default Tool
```

Props

Импортируем этот компонент в App компонент. То есть:

Файл App.js

```
import Tool from "./Tool"

function App() {
  return (
    <div className="App">
      <Tool/>
    </div>
  )
}

export default App
```

Props

ПЕРЕДОВАЯ ИНЖЕНЕРНАЯ ШКОЛА
УНИВЕРСИТЕТА ИННОПОЛИС

Как использовать props без деструктурирования

Чтобы использовать props, нужно передать его props в качестве аргумента в функцию
Это похоже на передачу аргументов в обычные функции JavaScript

Вот пример:

Файл Tool.js

```
function Tool(props) {
  const name = props.name;
  const tool = props.tool;
    return (
      <div>
        <h1>My name is {name}.</h1>
        <p>My favorite design tool is {tool}.</p>
      </div>
    );
}

export default Tool
```

Props

ПЕРЕДОВАЯ ИНЖЕНЕРНАЯ ШКОЛА
УНИВЕРСИТЕТА ИННОПОЛИС

Что произошло выше, шаг за шагом.....

Шаг 1 - Передайте props в качестве аргумента
В первой строке кода: function Tool(*props*){}
Это автоматически позволяет использовать props в компоненте приложения React

Шаг 2 - Объявить переменные props

```
const name = props.name;
const tool = props.tool;
```

Эти переменные отличаются от обычных переменных, потому что данные в них имеют отношение к props

Передать их непосредственно в шаблон следующим образом:

```
<h1> My name is {props.name} </h1>
```

Props

ПЕРЕДОВАЯ ИНЖЕНЕРНАЯ ШКОЛА
УНИВЕРСИТЕТА ИННОПОЛИС

Шаг 3 - Используйте переменные в шаблоне JSX

Когда объявили переменные, можно продолжить и разместить их там, где нужно для вывода или вычисление

```
return (
      <div>
        <h1>My name is {name}.</h1>
        <p>My favorite design tool is {tool}.</p>
      </div>
    );
```

Props

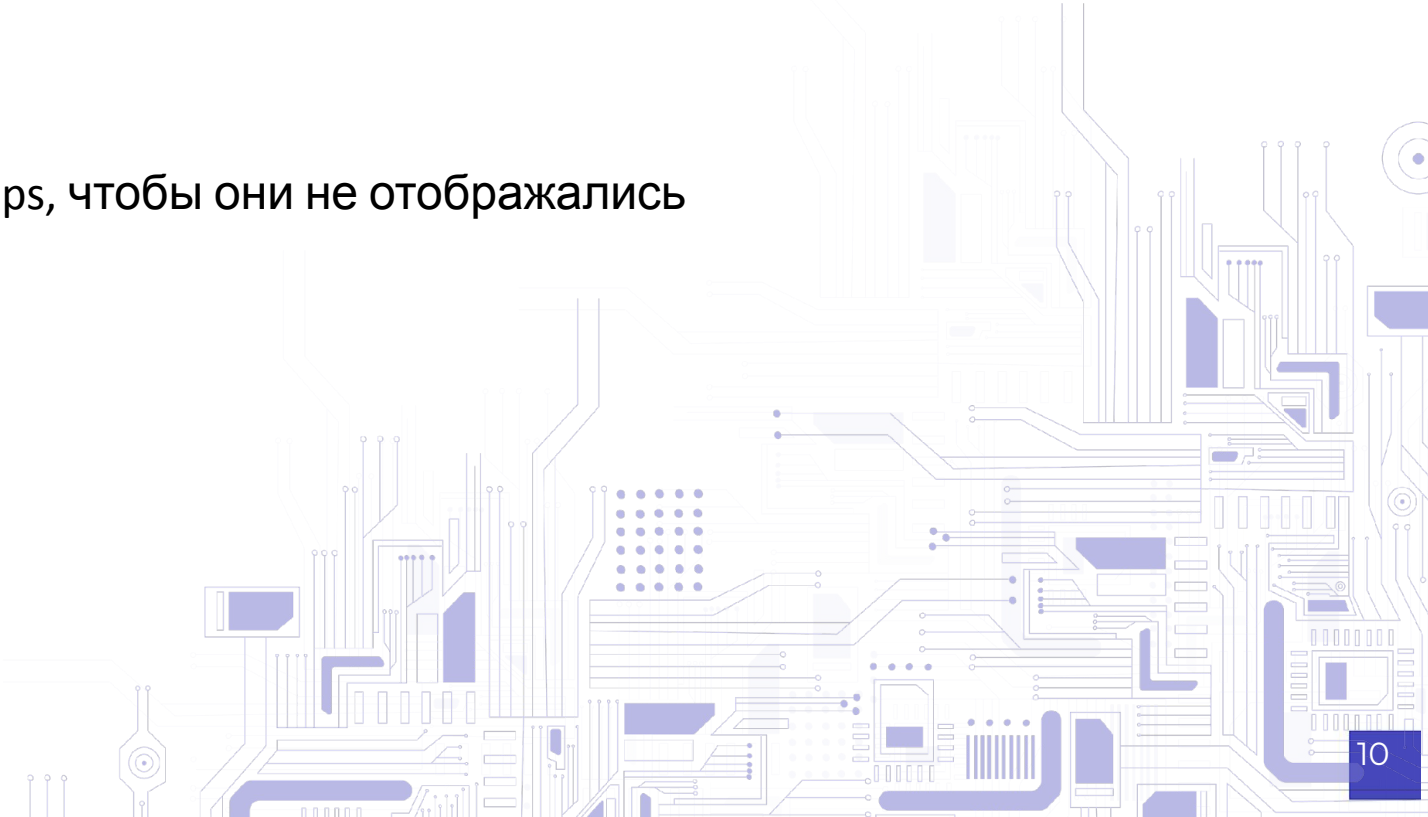
Шаг 4 - Передайте данные в props в App компоненте

Шаг - передать в props данные

Tool Компонент уже импортировали, и в данный момент он отображается в браузере:

```
My name is .
My favorite design tool is .
```

Можно создать данные по умолчанию для props, чтобы они не отображались пустыми при объявлении



Props

ПЕРЕДОВАЯ ИНЖЕНЕРНАЯ ШКОЛА
УНИВЕРСИТЕТА ИННОПОЛИС

Передаете данные как атрибуты
Это выглядит следующим образом:

```
import Tool from "./Tool"

function App() {
  return (
    <div className="App">
      <Tool name="Ihechikara" tool="Figma"/>
    </div>
  )
}

export default App
```

Props

Теперь двигаемся
от `<Tool/>`
до `<Tool name="Ihechikara" tool="Figma"/>`

отображается в браузере:

```
My name is Ihechikara.
My favorite design tool is Figma.
```

Props

Как использовать деструкцию ES6

Деструктурировать - означает демонтировать структуру чего-либо

В JavaScript эта структура может быть массивом, объектом или даже строкой, где свойства, составляющие структуру, будут использоваться для создания новой идентичной структуры (свойства могут быть изменены)

До появления ES6 именно так можно было извлекать некоторые данные в JavaScript:

```
var scores = [500, 400, 300];

var x = scores[0],
    y = scores[1],
    z = scores[2];

    console.log(x,y,z); // 500 400 300
```

Props

Но в ES6, используя деструктурирование, мы можем сделать это:

```
let scores = [500, 400, 300];
let [x, y, z] = scores;
console.log(x,y,z); //500 400 300
```

Переменные x, y и z будут наследовать значения в массиве scores в том порядке, в котором они отображаются, so x = 500, y = 400 и z = 300

В ситуации, когда все значения в массиве были унаследованы, любое другое значение, оставшееся без родительского значения, будет возвращено как неопределенное.
То есть:

```
let scores = [500, 400, 300];
let [x, y, z, w] = scores;
console.log(x,y,z,w); //500 400 300 undefined
```

Props

Вот пример использования объектов:

```
let scores = {
    pass: 70,
    avg: 50,
    fail: 30
};

let { pass, avg, fail} = scores;

console.log(pass, avg, fail); // 70 50 30
```

Процесс тот же, что и при деструктурировании массивов.

Props

Вот еще один пример, но со строками:

```
let [user, interface] = 'UI';
console.log(user); // U
console.log(interface); // I
```

Строка была разделена на отдельные буквы, а затем присвоена переменным в массиве.

Props

ПЕРЕДОВАЯ ИНЖЕНЕРНАЯ ШКОЛА
УНИВЕРСИТЕТА ИННОПОЛИС

Как использовать деструктурирование в React.js

Существуют различные сценарии, в которых, возможно, использовать деструктурирование в React.

Распространенным из них это useState хук.

```
import { useState } from 'react';

function TestDestructuring() {
    const [grade, setGrade] = useState('A');

    return(
        <>

        </>
    )
}

export default TestDestructuring
```

Props

Выше создали постоянную переменную grade вместе с функцией setGrade, целью которой является обновление значения переменной

И устанавливаем значение grade равным 'A' с помощью деструктурирования.

Props

Как использовать props с деструктурированием

Код для этого раздела полностью совпадает с предыдущим разделом, за исключением метода объявления props

В предыдущем разделе мы объявили наш props следующим образом:

```
const name = props.name;
const tool = props.tool;
```

Но не нужно делать это с помощью деструктурирования.

Props

ПЕРЕДОВАЯ ИНЖЕНЕРНАЯ ШКОЛА
УНИВЕРСИТЕТА ИННОПОЛИС

Просто и коротко:

```
function Tool({name, tool}) {

    return (
      <div>
        <h1>My name is {name}.</h1>
        <p>My favorite design tool is {tool}.</p>
      </div>
    );
}

export default Tool
```

Props

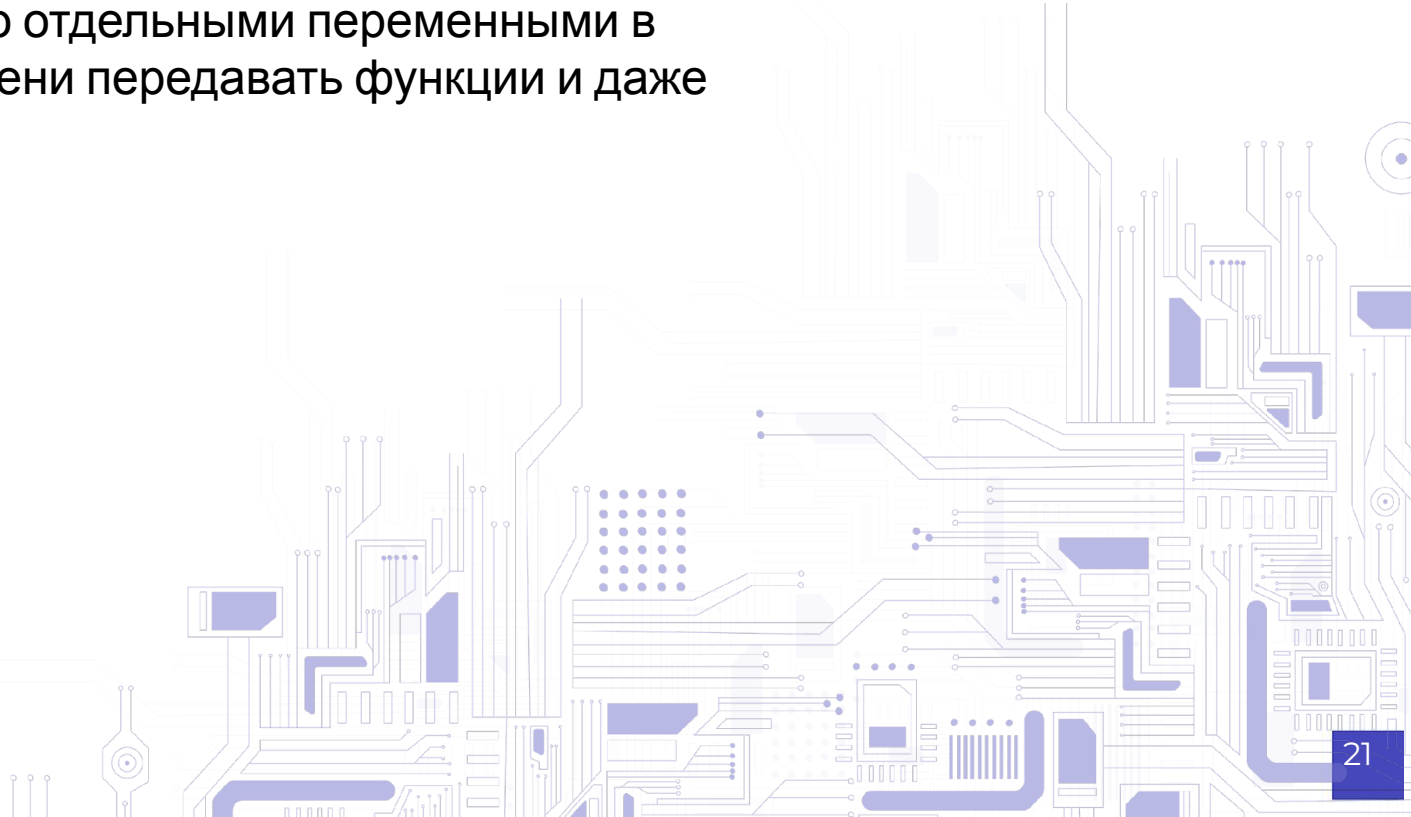
ПЕРЕДОВАЯ ИНЖЕНЕРНАЯ ШКОЛА
УНИВЕРСИТЕТА ИННОПОЛИС

Разница заключается в первой строке кода.

Вместо передачи props в качестве аргумента деструктурировали и передали переменные в качестве аргумента функции.

Все остальное остается прежним.

Обратите внимание, что не ограничены только отдельными переменными в качестве данных props – можете в равной степени передавать функции и даже данные из объектов



Props

ПЕРЕДОВАЯ ИНЖЕНЕРНАЯ ШКОЛА
УНИВЕРСИТЕТА ИННОПОЛИС

Как установить значения по умолчанию для props

Чтобы данные props не были пустыми при их создании, можно передать значение по умолчанию.
Вот как это сделать:

```
function Tool({name, tool}) {
    return (
      <div>
        <h1>My name is {name}.</h1>
        <p>My favorite design tool is {tool}.</p>
      </div>
    );

  }
  Tool.defaultProps = {
    name: "Designer",
    tool: "Adobe XD"
  }
export default Tool
```

Props

Непосредственно в конце кода перед экспортом компонента объявили значения по умолчанию для props

Чтобы сделать это, начали с названия компонента и точки, соединяющего его с defaultProps, который монтируется при создании приложения React

Теперь, куда бы ни импортировали этот компонент, эти значения будут начальными, а не пустыми

Когда передаете данные дочернему компоненту, это действие переопределяет значения по умолчанию

State

Одно из наиболее важных понятий, которое должен понимать любой современный разработчик JavaScript, - это состояние (State)

Что такое состояние?

Любой веб-сайт или приложение, которые создается с помощью простого JavaScript, уже включает state

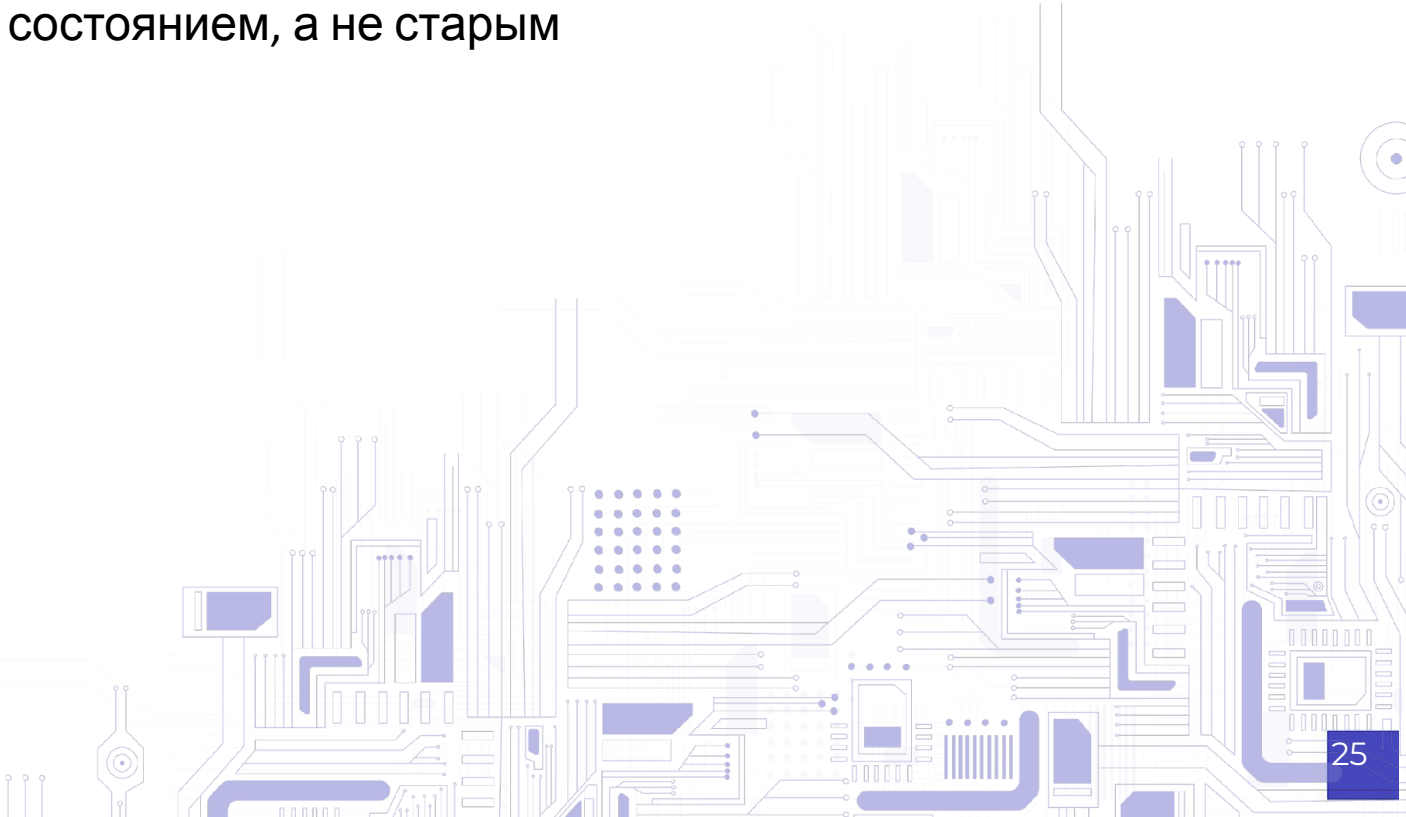
Просто не очевидно, где он находится.

State

Пользовательский интерфейс интерфейсного приложения - это представление его состояния.

Состояние - это просто моментальный снимок во времени.

Если пользователь изменяет состояние, взаимодействуя с приложением, пользовательский интерфейс впоследствии может выглядеть совершенно иначе, потому что он представлен этим новым состоянием, а не старым состоянием.



State

ПЕРЕДОВАЯ ИНЖЕНЕРНАЯ ШКОЛА
УНИВЕРСИТЕТА ИННОПОЛИС

Состояние может быть разным:

- Логическое значение, которое сообщает пользовательскому интерфейсу, что диалоговый / модальный / всплывающий компонент открыт или закрыт.
- Пользовательский объект, который отражает текущего пользователя, вошедшего в систему приложения.
- Данные из удаленного API (например, объект / список пользователей), которые извлекаются в React и отображаются в вашем пользовательском интерфейсе.

Состояние - это просто еще одно причудливое слово для структуры данных JavaScript, представляющей состояние с помощью примитивов и объектов JavaScript.

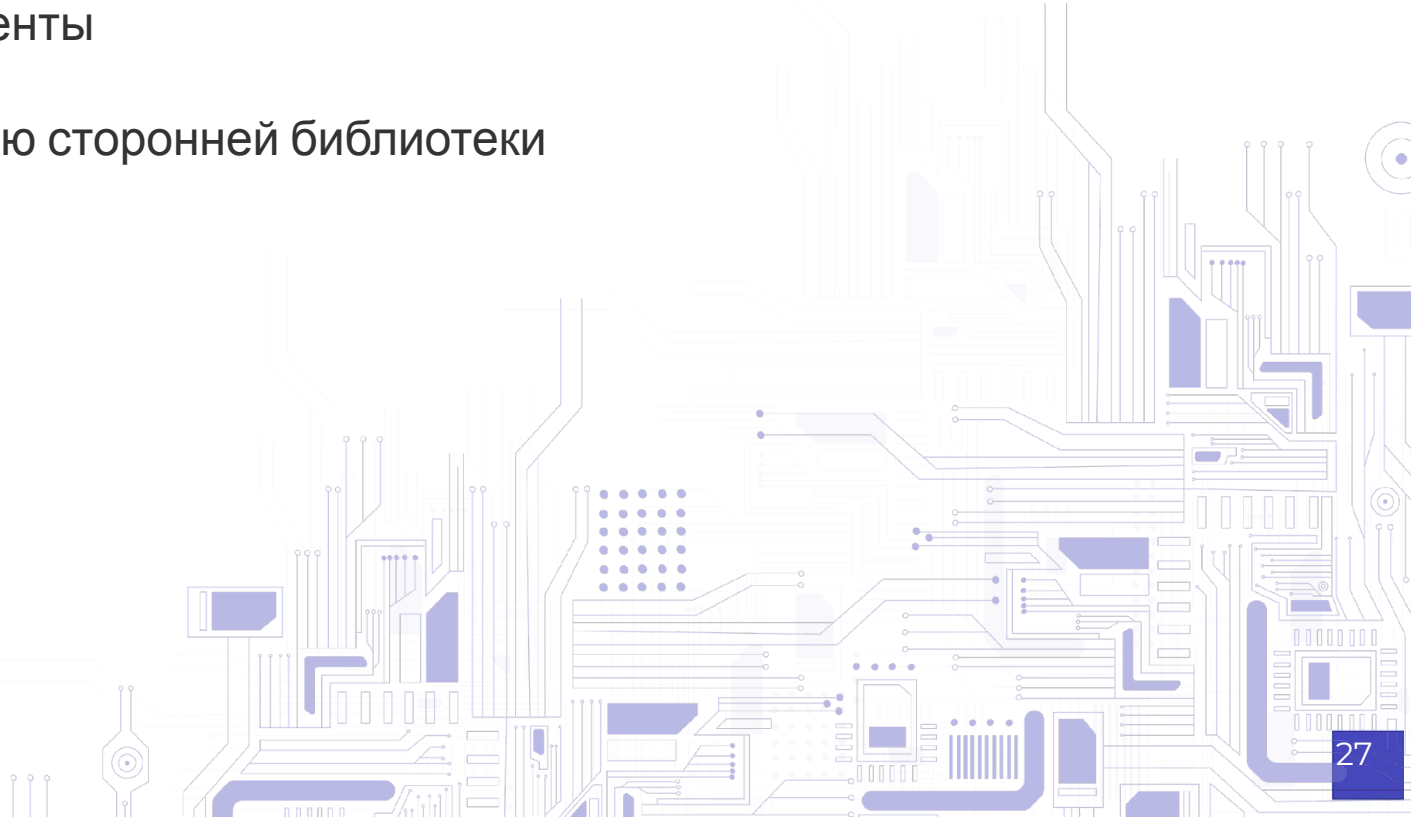
Например, простым состоянием может быть логическое значение JavaScript, тогда как более сложным состоянием пользовательского интерфейса может быть объект JavaScript

State

Локальное или глобальное состояние

Это три основные стратегии управления состоянием в React:

- Управлять состоянием в компоненте React
- Управляйте состоянием в компоненте React верхнего уровня, где оно распространяется на все дочерние компоненты
- Управляйте состоянием вне React с помощью сторонней библиотеки управления состоянием



State

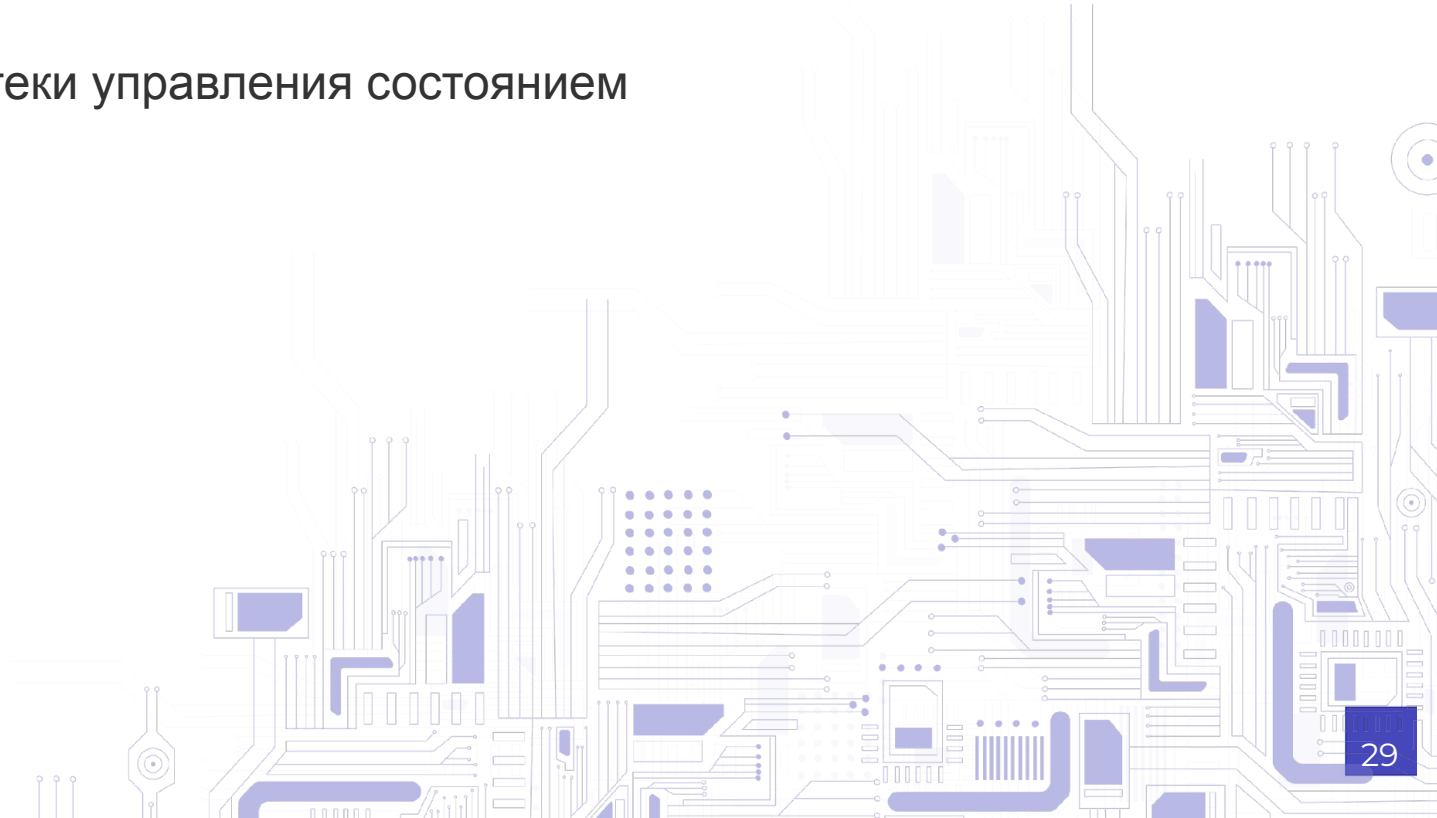
Все три стратегии соответствуют следующим типам состояний:

- локальное состояние
- глобальное состояние, но управляемое в React
- глобальное состояние, управляемое сторонней библиотекой управления состоянием

State

Кроме того, включение всех трех стратегий соответствует различным функциям или комбинациям этих функций внутри или за пределами возможностей React:

- useState и useReducer
- useState / useReducer с useContext
- Redux, MobX и различные другие библиотеки управления состоянием



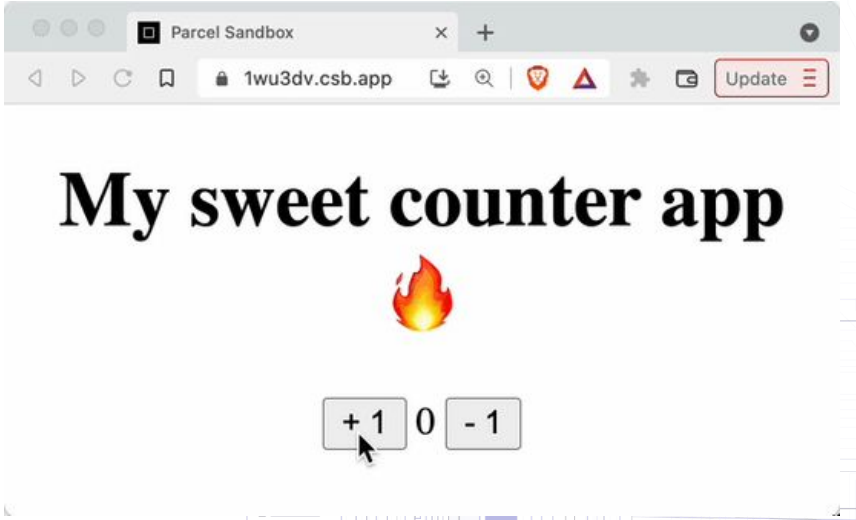
State

Вот базовый пример:

Предположим, что создаем приложение-счетчик с помощью JavaScript
Хотим, чтобы это приложение могло отображать текущее количество, а также увеличивать и уменьшать количество на единицу

Приложение будет состоять только из текущего количества, а также кнопки для увеличения количества на единицу и второй кнопки для уменьшения количества на единицу

Окончательная версия приложения:



State

Начальная верстка для приложения:

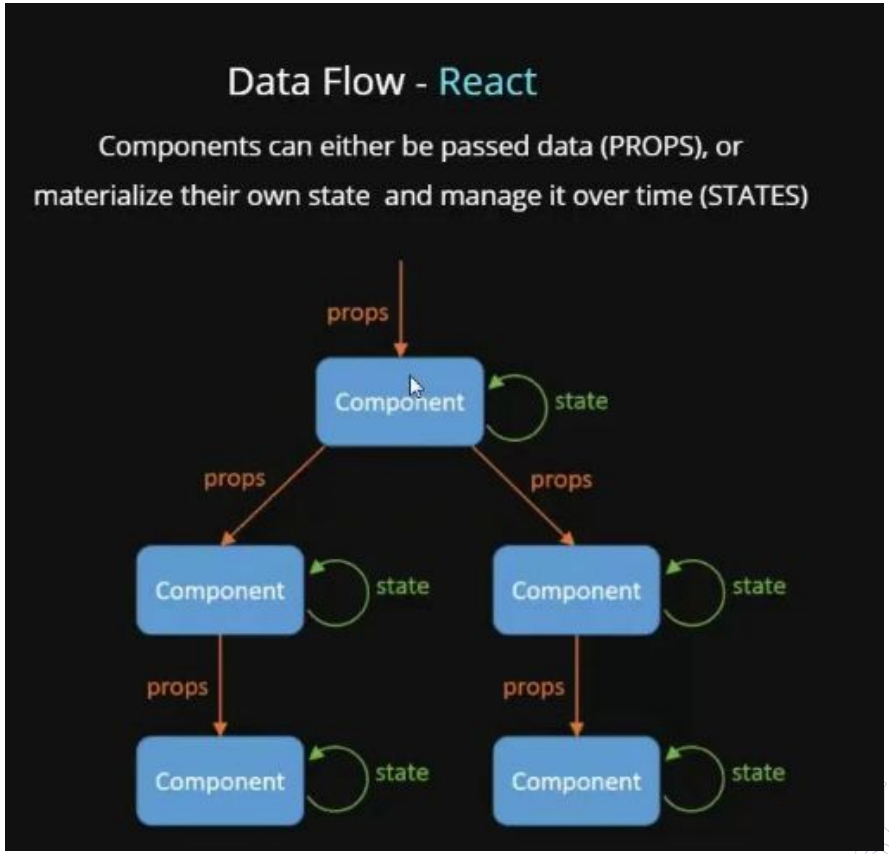
```
<!DOCTYPE html>
<html>
  <head>
    <title>Counter App</title>
    <meta charset="UTF-8" />
  </head>

  <body>
    <div>
      <button>+ 1</button>
      <span>0</span>
      <button>- 1</button>
    </div>
  </body>
</html>
```

State

ПЕРЕДОВАЯ ИНЖЕНЕРНАЯ ШКОЛА УНИВЕРСИТЕТА ИННОПОЛИС

Состояние - это данные, которыми нужно управлять с течением времени в приложении

Состояние часто изменяется с помощью пользовательского ввода



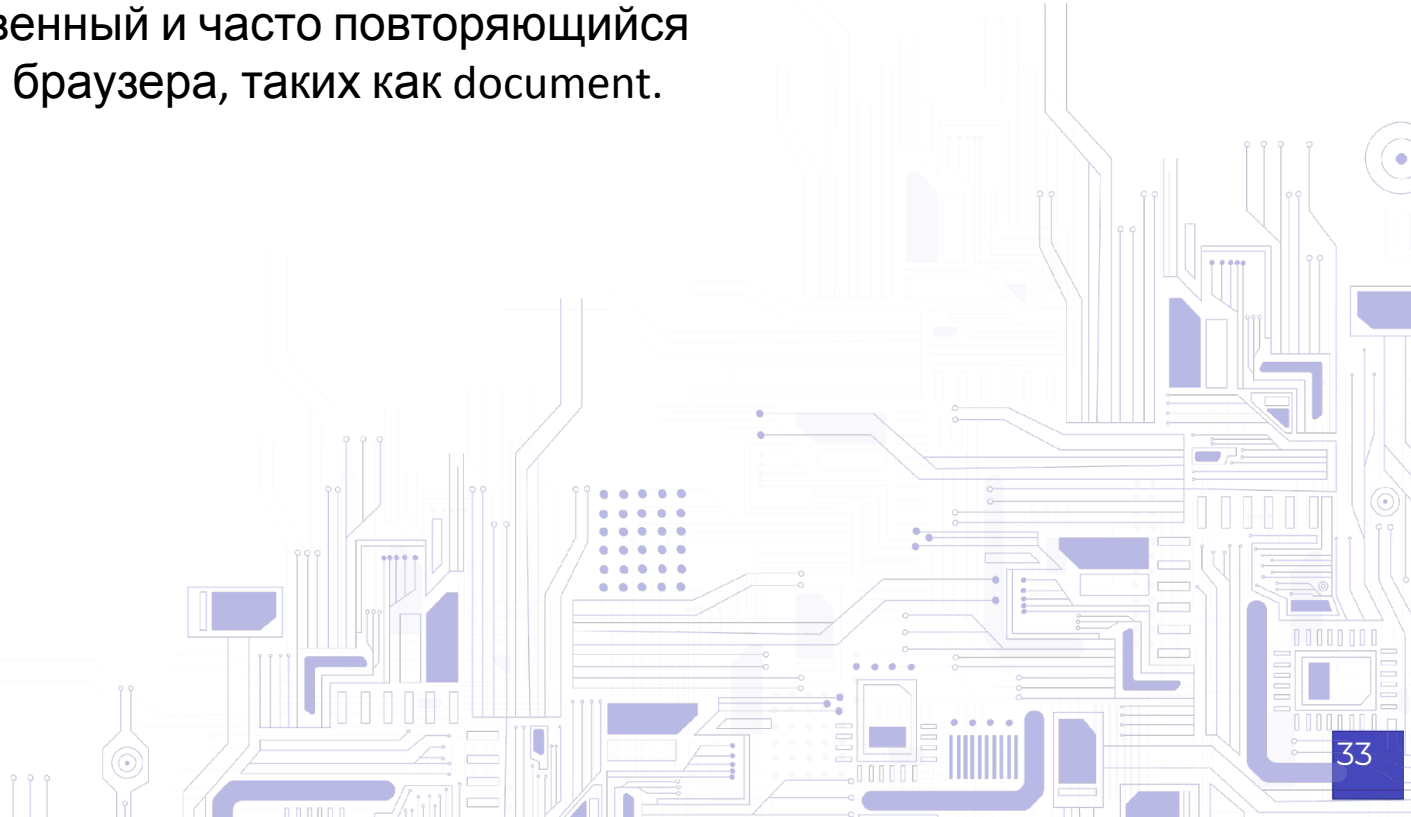
State

ПЕРЕДОВАЯ ИНЖЕНЕРНАЯ ШКОЛА УНИВЕРСИТЕТА ИННОПОЛИС

Проблемы с состоянием в обычном JavaScript

Хотя концепция state кажется простой, при использовании только простого JavaScript возникают две проблемы с ее управлением:

- Не очевидно, что это за состояние и где оно обитает
- Чтение и обновление состояния - неестественный и часто повторяющийся процесс при использовании встроенных API браузера, таких как document.



State

Как бы мы обновили наше состояние подсчета, когда наш пользователь нажимает на любую кнопку?

Сначала нам нужно получить ссылку на каждый элемент.
Чтобы сделать это в обычном JavaScript, обычной практикой является добавление уникального id атрибута к каждому элементу, выбор каждого элемента в JavaScript с помощью document.querySelector метода и сохранение ссылки в локальной переменной:

```
<!DOCTYPE html>
<html>
  <head>
    <title>Counter App</title>
    <meta charset="UTF-8" />
  </head>

  <body>
    <div>
      <button id="increment">+ 1</button>
      <span id="count">0</span>
      <button id="decrement">- 1</button>
    </div>

    <script>
      const increment = document.querySelector("#increment");
      const count = document.querySelector("#count");
      const decrement = document.querySelector("#decrement");
    </script>
  </body>
</html>
```

State

Есть ссылки на каждый HTML-элемент, как заставить кнопку увеличения работать?

Сначала нам нужно прослушать событие нажатия на кнопку увеличения. Затем, когда кнопка нажата, нужно получить текущее значение count из элемента с id значением "count".

Для этого погружаемся в HTML-документ с помощью document API и получаем это значение с помощью count.innerText.

innerText Значение представляет собой строку, поэтому преобразуем его в число, добавляем 1, а затем обратно записываем это значение в count.innerText

Чтобы заставить кнопку уменьшения работать, снова проделываем те же шаги. Разница лишь в том, что используем выражение Number(count.innerText - 1)

State


```html
<!DOCTYPE html>
<html>
  <head>
    <title>Counter App</title>
    <meta charset="UTF-8" />
  </head>

  <body>
    <div>
      <button id="increment">+ 1</button>
      <span id="count">0</span>
      <button id="decrement">- 1</button>
    </div>

    <script>
      const increment = document.querySelector("#increment");
      const count = document.querySelector("#count");
      const decrement = document.querySelector("#decrement");

      increment.addEventListener("click", () => {
        count.innerText = Number(count.innerText) + 1;
      });

      decrement.addEventListener("click", () => {
        count.innerText = Number(count.innerText) - 1;
      });
    </script>
  </body>
</html>
```

State

несколько шагов, которые повторяются:

- Добавить произвольный идентификатор к элементам HTML
- Запрос элемента с использованием JavaScript
- Сохраняйте ссылку на элемент в переменной
- Прослушать соответствующее событие в элементе
- Получить значение текущего состояния с помощью document API
- Записать новое значение состояния обратно на страницу с .innerText

State

Это множество низкоуровневых инструкций, которые требуются для работы программы, но они не помогают думать о базовом состоянии.

Состояние зависит от браузера. Это означает, что должны сначала "найти" состояние, а затем **императивно** обновить это значение

К счастью, React предоставляет нам гораздо более простой способ обновления state ☺️

State

Как React помогает нам управлять состоянием?

Одним из существенных преимуществ использования React является то, что он предоставляет гораздо более простые шаблоны для обновления состояния.

В отличие от обычного JavaScript, React берет на себя сложную работу по обновлению того, что видит пользователь

Все, что нужно сделать, это сообщить ему, каким состоянием управляем и каким должно быть новое значение.

Вместо состояния, живущего в браузере и вынужденного находить его каждый раз, когда нужно его прочитать или обновить, можем просто поместить его в переменную, а затем обновить значение этой переменной

После того, как это сделаем, пользователям будет показано обновление и новое значение

State

ПЕРЕДОВАЯ ИНЖЕНЕРНАЯ ШКОЛА УНИВЕРСИТЕТА ИННОПОЛИС

В этом заключается вся концепция управления состоянием в React

Вместо использования HTML-документа можем записать всю нашу верстку в компоненте React

Компонент написан идентично обычной функции JavaScript и отображает те же HTML-элементы, используя идентичный синтаксис, называемый JSX

```
export default function Counter() {
  return (
    <div>
      <button>+ 1</button>
      <span>0</span>
      <button>- 1</button>
    </div>
  );
}
```

State

Переменная может быть объявлена многими способами. Самый популярный способ управления состоянием компонента - с помощью `useState` хука.

Хук в `React` работает очень похоже на обычные функции `JavaScript`. Это означает, что можем вызвать его в верхней части нашего компонента и передать ему значение по умолчанию в качестве начального значения для приложения `counter`.

Поскольку начальное значение значения `count` равно нулю, просто вызываем хук и передаем ему значение `0`, и это значение помещается в переменную состояния

State

```jsx
import { useState } from 'react';

export default function Counter() {
  const [count] = useState(0);

  return (
    <div>
      <button>+ 1</button>
      <span>{count}</span>
      <button>- 1</button>
    </div>
  );
}
```

State

Больше нет необходимости использовать count.innerText.
Просто вывести и прочитать значение состояния с помощью count.

Возвращаемое значение из useState представляет собой массив

Когда его деструктурируем, первым деструктурируемым значением является переменная состояния. Второе значение - это функция для обновления состояния.

State

Заставить кнопку увеличения работать?

не нужно делать так:

- Нам не нужно добавлять `id` к нашим HTML-элементам
- Нам не нужно погружаться в DOM и выяснять, какая кнопка какая
- Нам не нужно прослушивать событие щелчка с `document.addEventListener`

State

Чтобы обновлять состояние при нажатии на кнопку, добавьте onClick атрибут к каждой кнопке. Это позволяет вызывать функцию при нажатии кнопки пользователем.

Для кнопки увеличения обновим состояние, передав count + 1 в setCount, а для кнопки уменьшения передадим count - 1 в setCount

State


```jsx
import { useState } from 'react';

export default function Counter() {
  const [count, setCount] = useState(0);

  function incrementCount() {
    setCount(count + 1);
  }

  function decrementCount() {
    setCount(count - 1);
  }

  return (
    <div>
      <button onClick={incrementCount}>+ 1</button>
      <span>{count}</span>
      <button onClick={decrementCount}>- 1</button>
    </div>
  );
}
```

State


```jsx
import { useState } from 'react';

export default function Counter() {
  const [count, setCount] = useState(0);

  function incrementCount() {
    setCount(count + 1);
  }

  function decrementCount() {
    setCount(count - 1);
  }

  return (
    <div>
      <button onClick={incrementCount}>+ 1</button>
      <span>{count}</span>
      <button onClick={decrementCount}>- 1</button>
    </div>
  );
}
```

Передовые инженерные школы

МИНОБРНАУКИ РОССИИ

УНИВЕРСИТЕТ ИННОПОЛИС

Спасибо за внимание