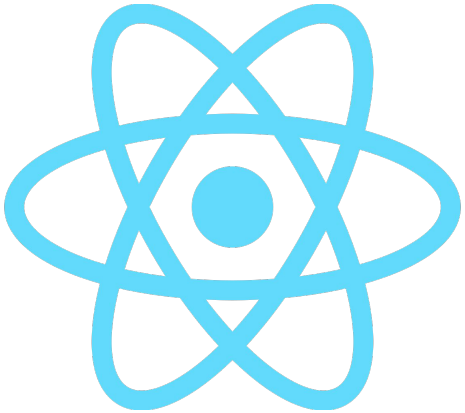
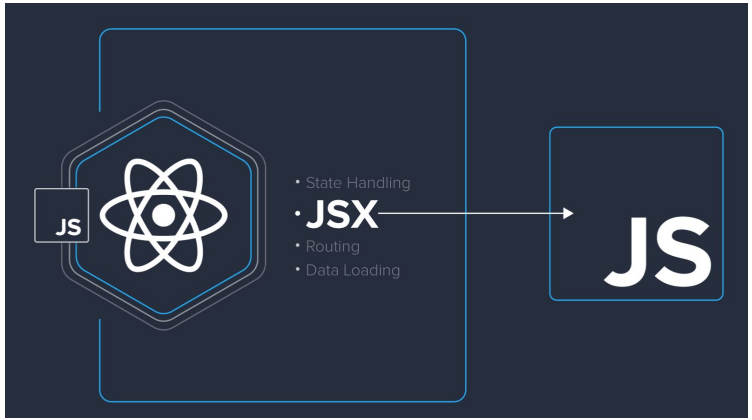


Тема занятия:

React: JSX, props, children, state

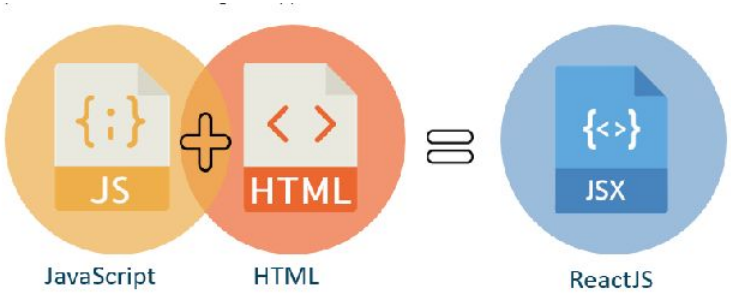
JSX



Рассмотрим объявление переменной:

```
const element = <h1>Привет, мир!</h1>;
```

Этот странный тег — ни строка, ни фрагмент HTML.



Это JSX — расширение языка JavaScript. Его рекомендуем использовать его, когда требуется объяснить React, как должен выглядеть UI. JSX напоминает язык шаблонов, наделённый силой JavaScript.

JSX создаёт «элементы» React

Встраивание переменных в JSX

В JSX мы можем не просто создавать элементы, но и делать их гибкими, с помощью встраивания переменных.

В приведённом ниже примере мы объявляем переменную с именем name, а затем используем ее внутри JSX, обернув ее в фигурные скобки:

```
const name = 'Josh Perez';
const element = <h1>Привет, {name}</h1>;
```

Встраивание выражений в JSX

JSX допускает использование любых корректных JavaScript-выражений внутри фигурных скобок. Например, 2 + 2, user.firstName и formatName(user) являются допустимыми выражениями.

```
function formatName(user) {
  return user.firstName + ' ' + user.lastName;
}

const user = {
  firstName: 'Марья',
  lastName: 'Моревна'
};

const element = (
  <h1>
    Здравствуй, {formatName(user)}!
  </h1>
);
```

JSX это тоже выражение

После компиляции каждое JSX-выражение становится обычным вызовом JavaScript-функции, результат которого — объект JavaScript.

Из этого следует, что JSX можно использовать внутри инструкций **if** и циклов **for**, присваивать переменным, передавать функции в качестве аргумента и возвращать из функции.

```
function getGreeting(user) {
  if (user) {
    return <h1>Здравствуй, {formatName(user)}!</h1>;
  }
  return <h1>Здравствуй, незнакомец.</h1>;
}
```

Встроенный оператор if-else с тернарным оператором

Другой метод встроенной условной отрисовки элементов — использование условного оператора в JavaScript

```
условие ? true : false.
```

```
  return (
    <div>
      Пользователь <b>{isLoggedIn ? 'в данный момент' : 'не'}</b> авторизован.
    </div>
  );
}
```

Установка атрибутов с помощью JSX

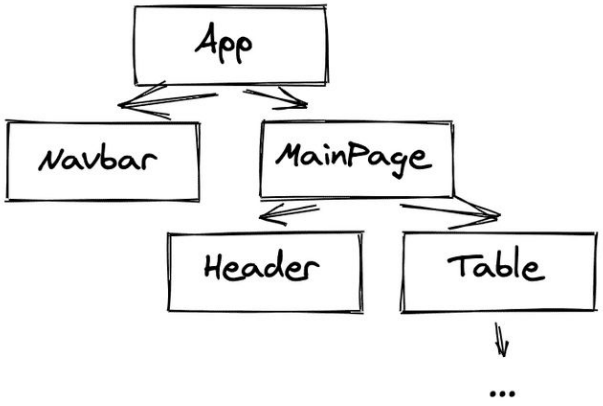
Вы можете использовать кавычки для указания строковых литералов в качестве атрибутов:

```
const element = <div tabIndex="0"></div>;
```

Вы также можете использовать фигурные скобки для вставки JavaScript-выражения в атрибут:

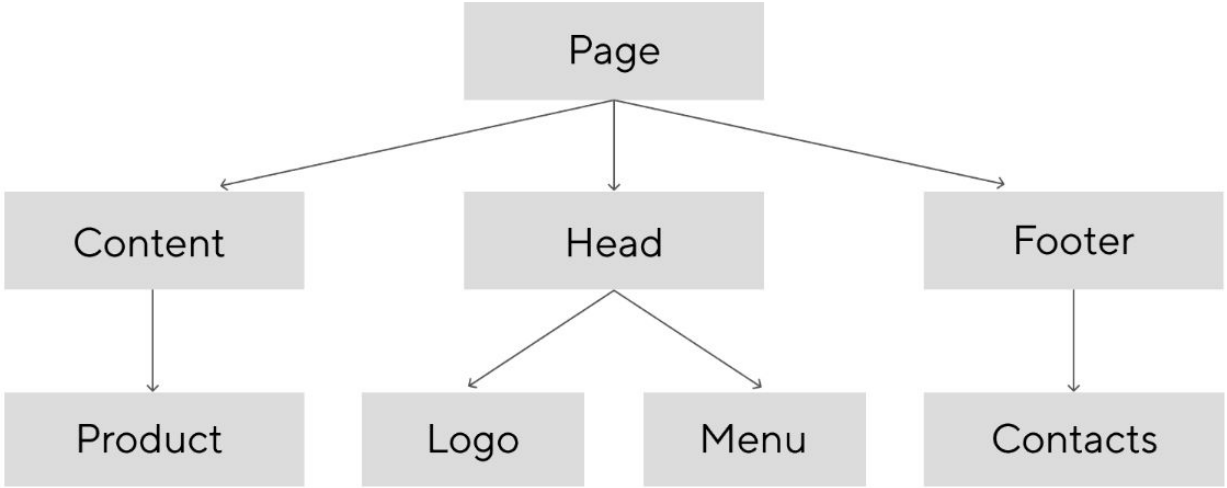
```
const element = <img src={user.avatarUrl}></img>;
```

React-компонент



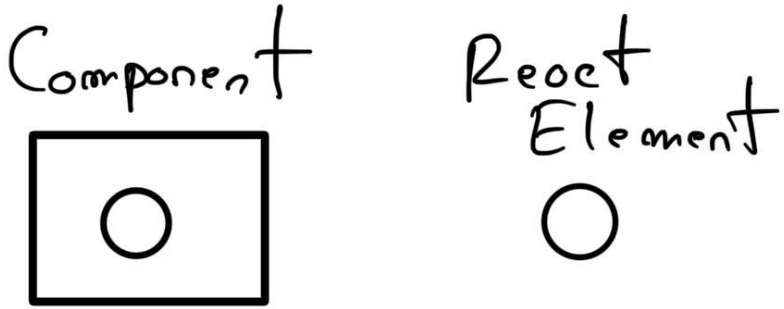
Приложения React состоят из компонентов.

Компонент — это часть пользовательского интерфейса, которая имеет свою собственную логику и внешний вид. Компонент может быть маленьким, как кнопка, или большим, как целая страница.



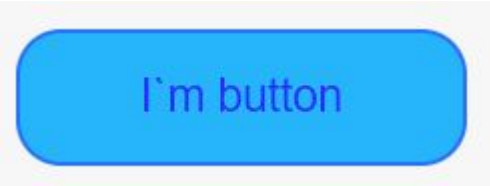
Раньше мы сталкивались только с элементами React, представляющие DOM-теги:

Однако элементы также могут быть пользовательскими компонентами



Компоненты React — это функции JavaScript, которые возвращают разметку:

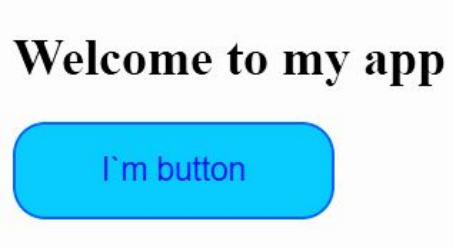
```
1  function MyButton() {
2      return <button>I'm a button</button>;
3  }
```



Обратите внимание, что **<MyButton />** начинается с заглавной буквы. Так вы узнаете, что это компонент React.

Компоненты, которые мы создаём могут быть вложены в другие теги или компоненты

```
1  export default function MyApp() {
2      return (
3          <div>
4              <h1>Welcome to my app</h1>
5              <MyButton />
6          </div>
7      );
8  }
```

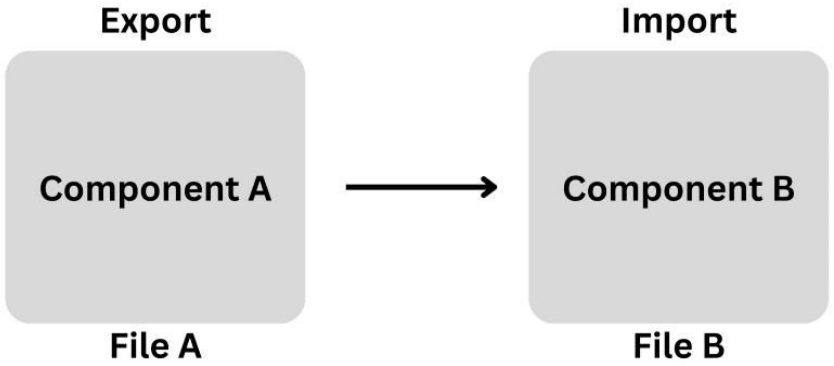


Импорт и экспорт

Для использования функционала и инструментов, предоставляемые определенными библиотеками, мы должны импортировать их к себе в файл

```
import React from 'react';
import moment from 'moment';

const MyComponent = () => {
  const currentDate = moment().format('MMMM Do YYYY, h:mm:ss a');

  return (
    <div>
      <h1>Hello! I am a React component.</h1>
      <p>Current date and time: {currentDate}</p>
    </div>
  );
};

export default MyComponent;
```

Магия компонентов заключается в возможности их повторного использования, часто имеет смысл начать разделять их на разные файлы. Это позволяет легко сканировать файлы и повторно использовать компоненты в большем количестве мест.



Экспорт по умолчанию

Экспорт компонента по умолчанию. В таком случае его можно будет использовать с любым именем при импорте.

```
// User.js
import React from 'react';

const UserProfile = () => {
  return <h1>User Profile</h1>;
};

export default UserProfile;
```

Импорт компонента экспортированного по умолчанию

Мы добавляем компонент UserProfile из предыдущего примера, в файл Dashboard, но под другим именем

```
// Dashboard.js
import React from 'react';
import CustomUserProfile from './User'; // Импортируем с другим именем

const Dashboard = () => {
  return (
    <div>
      <h2>Dashboard</h2>
      <CustomUserProfile />
    </div>
  );
};

export default Dashboard;
```

Именованный экспорт

Экспортируем компоненты по имени для возможности импорта нескольких компонентов из одного файла.

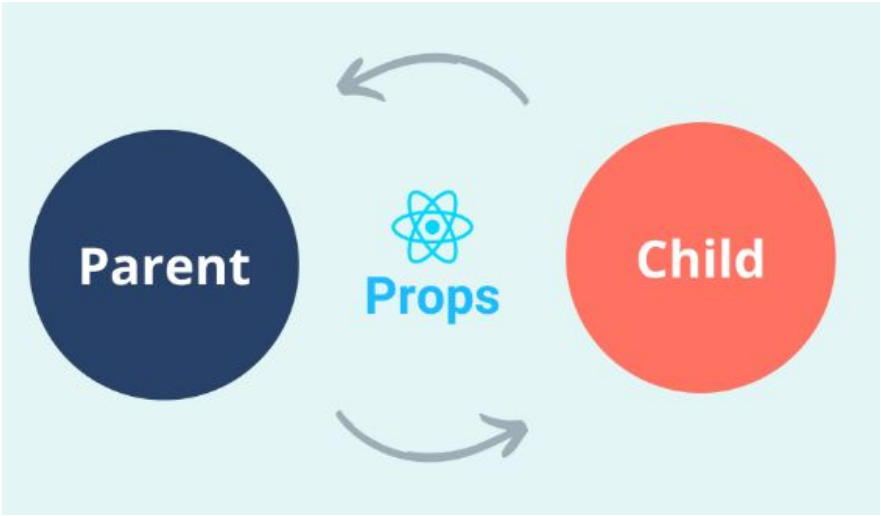
```jsx
// UserComponents.js
import React from 'react';

export const UserProfile = () => {
  return <p>User Profile</p>;
};

export const UserPosts = () => {
  return <p>Recent Posts</p>;
};
```

Именованный импорт

Импортируем именованные компоненты из файла UserComponents в UserDashboard.

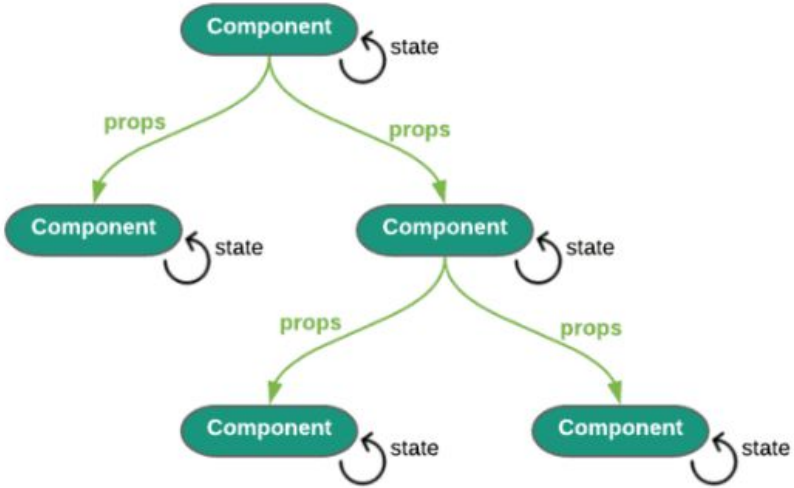
```jsx
// UserDashboard.js
import React from 'react';
import { UserProfile, UserPosts } from './UserComponents';

const UserDashboard = () => {
  return (
    <div>
      <UserProfile />
      <UserPosts />
    </div>
  );
};

export default UserDashboard;
```

Props



Когда React видит элемент, представляющий пользовательский компонент, он передаёт JSX-атрибуты этому компоненту в виде единственного объекта. Мы называем этот объект «props».

Props используются для передачи данных от родительских компонентов дочерним компонентам. Это один из основных механизмов передачи данных в React.



Пример использования props

Дочерний компонент

Значения для переменных в JSX мы берем из объекта props

```jsx
import React from 'react';

const ChildComponent = (props) => {
  return (
    <div>
      <p>Name: {props.name}</p>
      <p>Age: {props.age}</p>
    </div>
  );
}

export default ChildComponent;
```

Пример использования props

Родительский компонент

ParentComponent передает name и age в ChildComponent через props.

```
import React from 'react';
import ChildComponent from './ChildComponent';

const ParentComponent = () => {
  return (
    <ChildComponent name="John" age={30} />
  );
}

export default ParentComponent;
```

Несколько важных моментов о props :

1. **Props только для чтения (read-only)**: Компонент не должен изменять свои собственные props.

Несколько важных моментов о props :

2. **Деструктуризация Props:** В компонентах часто используется деструктуризация для более удобного доступа к props.

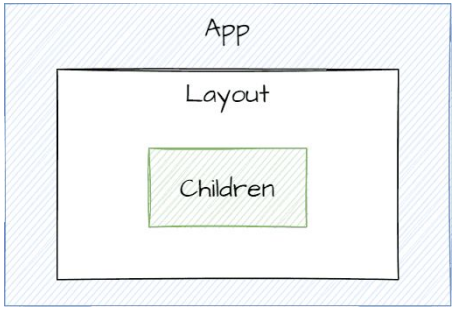
```
const ChildComponent = ({ name, age }) => {
  return (
    <div>
      <p>Name: {name}</p>
      <p>Age: {age}</p>
    </div>
  );
};
```

Несколько важных моментов о props :

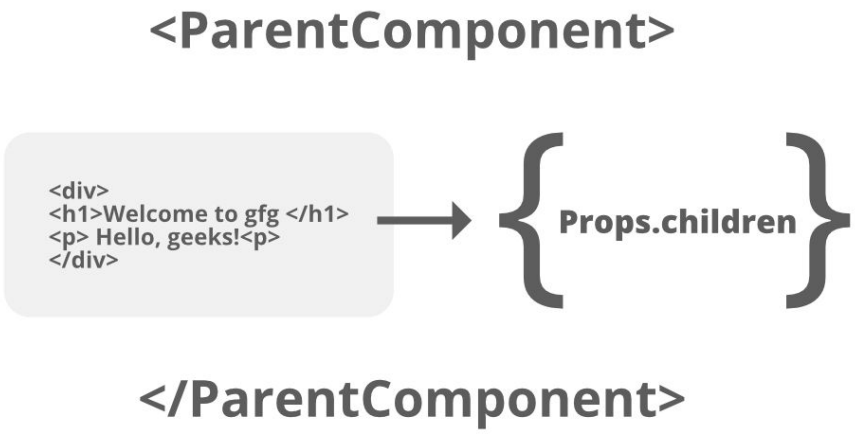
3. **Default Props:** Компонент может иметь значения по умолчанию для props, которые будут использоваться, если соответствующие значения не были переданы.

```
const ChildComponent = ({ name = 'Default Name', age = 25 }) => {
  return (
    <div>
      <p>Name: {name}</p>
      <p>Age: {age}</p>
    </div>
  );
};
```

Children



В React **children** представляет собой специальный проп, который позволяет передавать компоненту содержимое между открывающим и закрывающим тегами компонента. Это особенно полезно, когда вам нужно внедрить дочерние элементы внутрь компонента без явного передачи их как отдельных props.



Пример использования

Дочерний компонент

Добавляем специальный проп children после заголовка в компонент ChildComponent

```
const ChildComponent = (props) => {
  return (
    <div>
      <h1>Дочерний компонент</h1>
      {props.children}
    </div>
  );
}
```

Пример использования props

Родительский компонент

В родительском компоненте App, компонент <ChildComponent> записан с открывающимся и закрывающимся тегом между которыми добавлены элементы <p> и <button>

```
import ChildComponent from './ChildComponent';

const App = () => {
  return (
    <ChildComponent>
      <p>Этот текст будет дочерним элементом.</p>
      <button>Кнопка</button>
    </ChildComponent>
  );
}
```