React Fundamentals

tsevdos.me / @tsevdos

Agenda

All the content can be found here: https://github.com/codehub-learn/react-bootcamp.

- what is react
- core principles
- JSX
- components

Rules

Feel free to interrupt me for:

- questions
- relevant comments

What is React

React is a library for building user interfaces

- virtual DOM
- JSX
- event handling
- performance

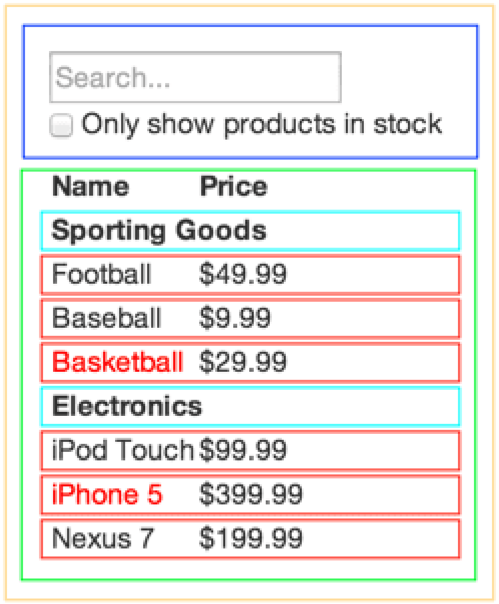
Core principles

- composition
- declarative
- unidirectional dataflow
- explicit mutations

Composition

- divide and conquer
- hide complexity
- comes from functional programming

Composition



Search...

Only show products in stock

Name	Price
Sporting Goods	
Football	$49.99
Baseball	$9.99
Basketball	$29.99
Electronics	
iPod Touch	$99.99
iPhone 5	$399.99
Nexus 7	$199.99

Composition

```
<Widget>
  <SearchForm />
  <Results>
    <Header />
    <SportsTable />
    <ElectronicsTable />
  </Results>
</Widget>
```

Composition

twitter.com example

- how UI is going to look
- state

Avatar sample code

```
function getProfilePhoto(username) {
  return "https://twitter.com/photos/" + username;
}

function getProfileLink(username) {
  return "https://twitter.com/" + username;
}

function getAvatar(username) {
  return {
    photo: getProfilePhoto(username),
    link: getProfileLink(username)
  }
}
```

Avatar (React code)

```
function ProfilePhoto(props) {
  return <img src={"https://twitter.com/photos/" + props.usern
}

function ProfileLink(props) {
  return (
    <a href={"https://twitter.com/" + props.username}>
      { props.username }
    </a>
  );
}

function Avatar(props) {
  return (
    <div>
```

Imperative and Declarative

- imperative programming is a programming paradigm that uses statements that change a program’s state
- declarative programming is a programming paradigm that expresses the logic of a computation without describing its control flow

Imperative

```
// Imperative (How)
var numbers = [1, 2, 3, 4, 5];
var total = 0;

for (var i = 0; i < numbers.length; i++) {
  total += numbers[i];
}
```

Declarative

```
// Declarative (What)
var numbers = [1, 2, 3, 4, 5];
var total = numbers.reduce(function(total, item){
  return total + item;
}, 0);
```

JavaScript built in methods

- map
- reduce
- filter

Declarative

- reduce side effects and mutability
- more clear / readable code
- less errors / bugs

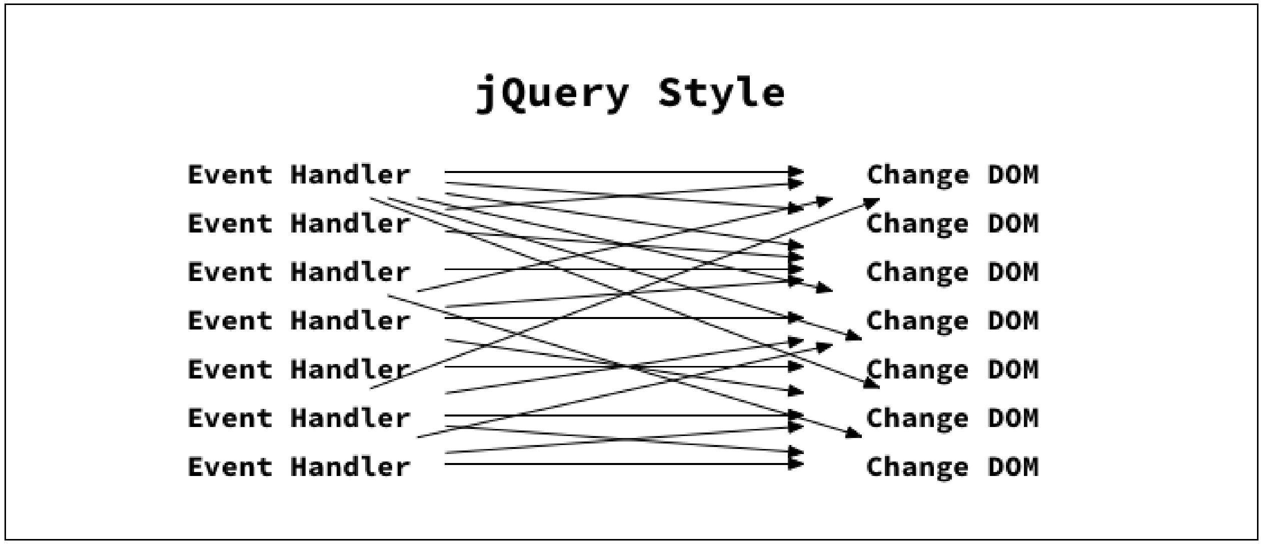
React is declarative

```
$("#btn").click(function(){
  $(this).toggleClass("active");
  if( $(this).text() === "Active" ) {
    $(this).text("Inactive")
  } else {
    $(this).text("Active")
  }
})
```

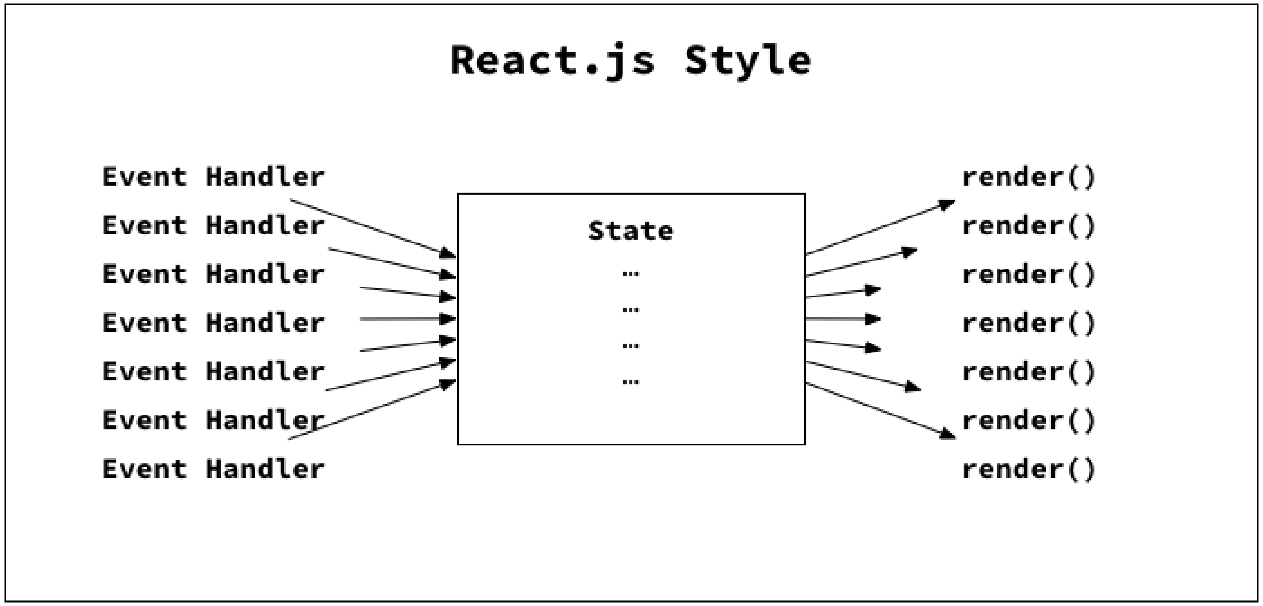
```
<Button onclick="handleClick" isActive={isActive} />

setIsActive(!isActive);
```

Unidirectional dataflow



jQuery Style
Event Handler
Event Handler
Event Handler
Event Handler
Event Handler
Event Handler
Event Handler
Change DOM
Change DOM
Change DOM
Change DOM
Change DOM
Change DOM
Change DOM



React.js Style
Event Handler
Event Handler
Event Handler
Event Handler
Event Handler
Event Handler
Event Handler
State
…
…
…
…
render()
render()
render()
render()
render()
render()
render()

Explicit mutations

```
setName("John");
```

Rendering elements

- React.createElement
- JSX
- virtual DOM

DOM scripting: document.createElement

```
<html>
<head></head>
<body>
  <div id="app"></div>
  <script type="text/javascript">
    const rootElement = document.getElementById('app');
    const element = document.createElement('div');
    element.textContent = 'Hello World';
    element.className = 'container';
    rootElement.appendChild(element);
  </script>
</body>
</html>
```

React.createElement

```
<html>
<head>
  <script src="https://unpkg.com/react@16/umd/react.developmen
  <script src="https://unpkg.com/react-dom@16/umd/react-dom.de
</head>
<body>
  <div id="app"></div>
  <script type="text/javascript">
    const rootElement = document.getElementById('app');
    const element = React.createElement(
      'div',
      { className: 'container' },
      'Hello World'
    );
    ReactDOM.render(element, rootElement);
```

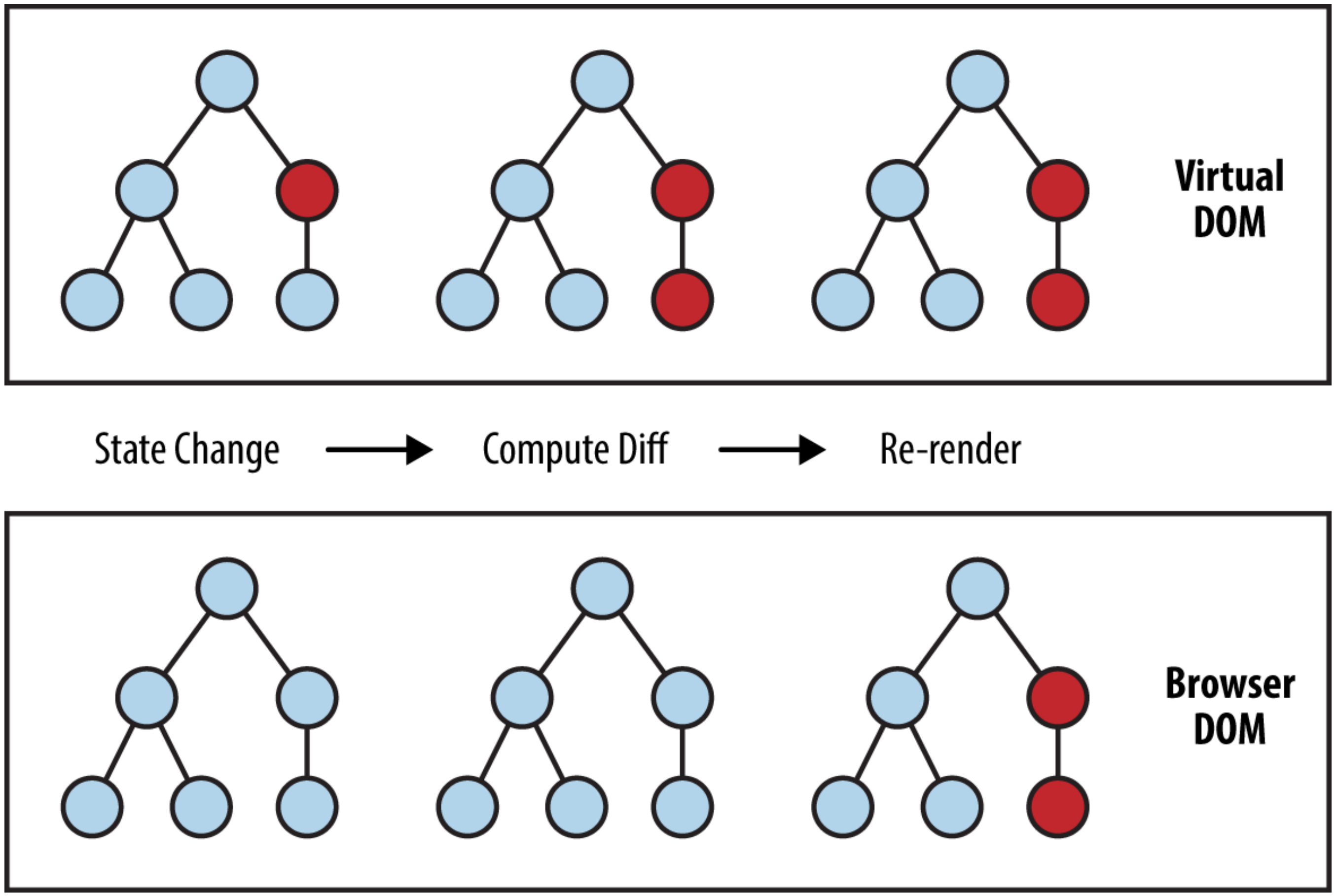
React.createElement

```
React.createElement(
  type,
  [props],
  [...children]
)
```

Virtual DOM

The virtual DOM (VDOM) is an in-memory representation of real DOM. The representation of a UI is kept in memory and synced with the “real” DOM. It’s a step that happens between the render function being called and the displaying of elements on the screen. This entire process is called reconciliation.

Virtual DOM


Virtual DOM
State Change
Compute Diff
Re-render
Browser DOM

React.createElement

```
<html>
<head>
  <script src="https://unpkg.com/react@16/umd/react.developmen
  <script src="https://unpkg.com/react-dom@16/umd/react-dom.de
</head>
<body>
  <div id="app"></div>
  <script type="text/javascript">
    const rootElement = document.getElementById('app');
    const element = React.createElement(
      "div",
      { className: "container" },
      React.createElement(
        "div",
        null,
```

JSX

```
<html>
<head>
  <script src="https://unpkg.com/react@16/umd/react.developmen
  <script src="https://unpkg.com/react-dom@16/umd/react-dom.de
  <script src="https://cdnjs.cloudflare.com/ajax/libs/babel-st
</head>
<body>
  <div id="app"></div>
  <script type="text/babel">
    const rootElement = document.getElementById('app');
    // const element = React.createElement(
    //   "div",
    //   { className: "container" },
    //   "Hello World"
    // );
```

JSX

```
<html>
<head>
  <script src="https://unpkg.com/react@16/umd/react.developmen
  <script src="https://unpkg.com/react-dom@16/umd/react-dom.de
  <script src="https://cdnjs.cloudflare.com/ajax/libs/babel-st
</head>
<body>
  <div id="app"></div>
  <script type="text/babel">
    const rootElement = document.getElementById('app');
    const element = (
      <div className="container">
        <div>Div 1</div>
        <div>
          <h2>Title</h2>
```

JSX interpolation

```
<html>
<head>
  <script src="https://unpkg.com/react@16/umd/react.developmen
  <script src="https://unpkg.com/react-dom@16/umd/react-dom.de
  <script src="https://cdnjs.cloudflare.com/ajax/libs/babel-st
</head>
<body>
  <div id="app"></div>
  <script type="text/babel">
    const rootElement = document.getElementById('app');

    const title = 'Hello World';
    const myClassName = 'container';

    const element = <div className={`${myClassName}-1`}>{title
```

Babel transpilation / compilation

- example

Components

- functional components
- props
- children
- conditional rendering

Components

A component is a function or a class which optionally accepts input and returns a React element (or null).

Still JSX (no components)

```
const element = (
  <div classname="container">
    <div>Hello World</div>
    <div>Hello World</div>
  </div>
);

ReactDOM.render(element, rootElement);
```

Still JSX (no components)

```
const myDiv = <div>Hello World</div>;
const element = (
  <div classname="container">
    {myDiv}
    {myDiv}
  </div>
);

ReactDOM.render(element, rootElement);
```

Our first functional reusable component

```
const MyDiv = (props) => {
  return <div>{props.msg}</div>
};

const element = (
  <div className="container">
    <MyDiv msg="Hello World" />
    <MyDiv msg="Welcome to Code.Hub" />
  </div>
);
```

Component rules

User-defined components must be capitalized in JSX (lower-case tag names are considered to be HTML tags)

- <mydiv /> compiles to React.createElement('mydiv') (html tag)
- <Mydiv /> compiles to React.createElement(Mydiv)

Functional component transpilation

Babel example

Components and children

```
const MyDiv = (props) => {
  return <div>{props.children}</div>
};

const element = (
  <div className="container">
    <MyDiv>Hello World</MyDiv>
    <MyDiv>
      Welcome to Code.Hub
      <MyDiv>Hi I'm a component</MyDiv>
    </MyDiv>
    <MyDiv>
      <h1>Title</h1>
      <p>Welcome</p>
    </MyDiv>
```

Children

Props.children displays whatever you include between the opening and closing tags when invoking a component.

- freedom and composition
- almost everything can be a child (element, component and function)

Functional components

```
const Avatar = (props) => {
  return (
    <div>
      <h3>{props.username}</h3>
      <img width="100" src={props.imgUrl} />
      <p>My age is {props.age}</p>
      <p>My hobbies are:</p>
      <ul>
        { props.hobbies.map((hobbie) => <li key={hobbie}>{hobb
      </ul>
      <p>I use {props.technologies.base} as a base and {props.
      { props.isOlympiakos ? <div>Ολυμπιακάρα</div> : <div>Υπά
      <button onClick={props.handleClick}>Click me</button>
    </div>
  )
```

Valid props

- string
- number
- boolean
- array
- object
- function
- symbol

Functional components

```
const rootElement = document.getElementById("app");
const data = [
  {
    title: "Total",
    number: 10358,
    percentage: "+ 3%",
  },
  {
    title: "Direct",
    number: 8560,
    percentage: "+ 10%",
  },
  {
    title: "Referral",
    number: 1798,
```

Functional components

```
const App = () => {
  return (
    <div>
      <Widget
        title="Website traffic"
        logo="https://image.flaticon.com/icons/svg/148/148767.
        data={data}
      />
      <Widget
        title="Website errors"
        logo="https://image.flaticon.com/icons/svg/148/148836.
        data={data2}
      />
    </div>
  );
```

Functional components

```
const WidgetEntryItem = (props) => {
  return (
    <div>
      <h3>{props.title}</h3>
      <p>{props.number} &nbsp;<span>{props.percentage}</span><
    </div>
  )
}

const Widget = (props) => {
  return (
    <div>
      <img width="30" height="30" src={props.logo} />
      <h2>{ props.title }</h2>
      { props.data.map((entry) => <WidgetEntryItem key={entry.
```

Conditional rendering: If/Else

```
const User = ({ username }) => {
  if (username) {
    return <div>Hello, {username}</div>
  }

  return <div>Hi stranger!</div>
}

ReactDOM.render(<User username="tsevdos" />, rootElement);
```

Conditional rendering: Ternary operator

```
const User = ({ username }) => {
  return (
    <div>
      {
        username
          ? <span>{username}</span>
          : <span>Hi stranger!</span>
      }
    </div>
  )
}

ReactDOM.render(<User username={'tsevdos'} />, rootElement);
```

Conditional rendering: Ternary operator

```
const User = ({ username }) => {
  return (
    <div>
      {
        username
          ? <React.Fragment>{username}</React.Fragment>
          : <React.Fragment>Hi stranger!</React.Fragment>
      }
    </div>
  )
}

ReactDOM.render(<User username={'tsevdos'} />, rootElement);
```

Conditional rendering: Short-circuit operator (&&)

```
const FavoriteColorsList = ({ list }) => {
  return (
    <div>
      {
        (list.length > 0) && <div>{ list.map((color) => <span>
      }
    </div>
  )
}

ReactDOM.render(<FavoriteColorsList list={['red', 'blue']} />,
```

Conditional rendering: Element variables

```
const Header = ({ isLoggedIn }) => {
  let button;

  if (isLoggedIn) {
    button = <button>Logout</button>;
  } else {
    button = <button>Login</button>;
  }

  return <div>{button}</div>;
}

ReactDOM.render(<User isLoggedIn={true} />, rootElement);
```

Components

- functional components
- state
- hooks
- event handlers

Components

A component is a function or a class which optionally accepts input and returns a React element (or null).

Component and state

```
const LikeCount = () => {
  const [counter, setCounter] = React.useState(0);
  const handleLike = () => {
    setCounter(counter + 1);
  };

  return (
    <div>
      <div className="emoji">💓 {counter}</div>
      <button onClick={handleLike}>Like!</button>
    </div>
  );
};

ReactDOM.render(<LikeCount />, document.getElementById("app"))
```

useState hook

useState hook enqueues changes to the component state and tells React that this component and its children need to be re-rendered with the updated state. This is the primary method you use to update the user interface in response to event handlers and server responses.

Component and state

```
const LikeCount = () => {
  const [counter, setCounter] = React.useState(0);
  const handleLike = () => {
    setCounter((counter) => counter + 1);
  };
  const handleDislike = () => {
    setCounter((counter) => counter - 1);
  };

  return (
    <div>
      <div className="emoji">💓 {counter}</div>
      <button onClick={handleLike}>Like!</button>
      <button onClick={handleDislike}>Dislike!</button>
    </div>
```

use useState hook correctly

- Only update the state with the appropriate function
- State updates may be asynchronous (React may batch multiple setState() calls into a single update for performance)

```
// Wrong
counter = 5; // this will not re-render a component

// Correct
const [counter, setCounter] = React.useState(0);
setCounter(5);

// Might cause a problem
setCounter(counter + 1);

// Correct
setCounter((counter) => counter + 1);
```

Do not mutate the state

```
const Avatar = () => {
  const [profile, setProfile] = React.useState({
    id: 10,
    user: {
      username: "tsevdos",
      name: "John Tsevdos",
      bio: "I really like React and front-end.",
    },
  });
  const changeName = () => {
    const newProfile = profile;
    newProfile.user.name = "New Name";
    setProfile(newProfile);
  };
```

Using state correctly

```
const Avatar = () => {
  const [profile, setProfile] = React.useState({
    id: 10,
    user: {
      username: "tsevdos",
      name: "John Tsevdos",
      bio: "I really like React and front-end.",
    },
  });
  const changeName = () => {
    setProfile((profile) => ({
      ...profile,
      user: {
        ...profile.user,
        name: "New Name",
```

Using state correctly

Immutable tricks for arrays and objects

```
// Arrays
// Spread Operator (ES6)
setState([...arr, "new value"]);

// Array.prototype.slice() (ES5)
const newArr = arr.slice();
newArr.push("new value");
setState(newArr);

// Objects
// Spread Operator (ES6)
setState({ ...user, name: "New Name" });

// Object.assign (ES6)
const newUser = Object.assign({}, user);
```

Using state

The state of a component can be the props of another one.

```
const Hello = ({ name }) => {
  return <h1>Hello, {name}</h1>;
};

const Form = () => {
  const [name, setName] = React.useState("");
  const handleOnChange = (e) => {
    setName(e.target.value);
  };

  return (
    <div>
      <Hello name={name} />
      <input type="text" name="name" onChange={handleOnChange}
    </div>
```

Components and events

- SyntheticEvent
- cross-browser wrapper around the browser’s native event
- it has the same interface as the browser’s native event, including stopPropagation() and preventDefault()
- you have access to the native event using event.nativeEvent

Components and events

- react events are named using camelCase, rather than lowercase
- supported events

Mini project: ToDo list

Part 1: create the presentational elements
(workshop/todo-app/00.html)

Mini project: ToDo list

Part 2: create the add todo functionality
(workshop/todo-app/01.html)

Mini project: ToDo list

Part 3: create the toggle todo functionality
(workshop/todo-app/02.html)

Mini project: ToDo list

Part 3: create the delete todo functionality
(workshop/todo-app/03.html)

Styling and CSS

- CSS classes
- in-line styles

CSS classes

```
const MyComponent = (props) => {
  return (
    <div className="columns">
      <div className="column">
        <p className="bd-notification is-primary">First column
      </div>
      <div className="column">
        <p className="bd-notification is-primary">Second colum
      </div>
      <div className="column">
        <p className="bd-notification is-primary">Third column
      </div>
      <div className="column">
        <p className="bd-notification is-primary">Fourth colum
      </div>
```

CSS classes

```
const MyComponent = (props) => {
  const columnclassName = "column";
  const paragraphClassName = "has-background-primary has-text-

  return (
    <div className="columns">
      <div className={columnclassName}>
        <p className={paragraphClassName}>First column</p>
      </div>
      <div className={columnclassName}>
        <p className={paragraphClassName}>Second column</p>
      </div>
      <div className={columnclassName}>
        <p className={paragraphClassName}>Third column</p>
      </div>
```

In-line styles

```
const firstParagraphStyle = {
  padding: "0.5em 1em",
  fontSize: "1.4em",
  background: "hsl(217, 71%, 53%)",
  color: "#fff"
};

const MyComponent = (props) => {
  const columnclassName = "column";
  const paragraphClassName = "has-background-primary has-text-

  return (
    <div className="columns">
      <div className={columnclassName}>
        <p style={firstParagraphStyle}>First column</p>
```

React and styling is a huge topic

- CSS Stylesheet
- Inline styling
- CSS Modules
- CSS-in-JS

Recap

- what is react
- core principles
- JSX
- components

Recap: Core principles

- composition
- declarative
- unidirectional dataflow
- explicit mutations

Recap: Components

- React.createElement
- JSX
- virtual DOM

Recap: Functional components

- props
- state
- children
- hooks
- conditional rendering
- event handlers

Recap: Styling and CSS

- CSS classes
- In-line styles

That's all folks

Questions / Discussions?