`props` and `state` are **CORE concepts** of React. Actually, only changes in `props` and/ or `state` trigger React to re-render your components and potentially update the DOM in the browser (a detailed look at how React checks whether to really touch the real DOM is provided in section 6).

### Props

`props` allow you to pass data from a parent (wrapping) component to a child (embedded) component.

### Example:

### AllPosts Component:

```
.   const posts = () => {
.       return (
.           <div>
.               <Post title="My first Post" />
.           </div>
.       );
.   }
```

Here, `title` is the custom property (`prop` ) set up on the custom `Post` component. We basically replicate the default HTML attribute behavior we already know (e.g. `<input type="text">` informs the browser about how to handle that input).

### Post Component:

```
.   const post = (props) => {
.       return (
.           <div>
.               <h1>{props.title}</h1>
.           </div>
.       );
```

```
    }
```

The `Post` component receives the `props` argument. You can of course name this argument whatever you want - it's your function definition, React doesn't care! But React will pass one argument to your component function => An object, which contains all properties you set up on `<Post ... />`.

`{props.title}` then dynamically outputs the `title` property of the `props` object - which is available since we set the `title` property inside `AllPosts` component (see above).

## State

Whilst props allow you to pass data down the component tree (and hence trigger an UI update), state is used to change the component, well, state from within. Changes to state also trigger an UI update.

## Example:

## NewPost Component:

```
class NewPost extends Component { // state can only be accessed in class-based components!
    state = {
        counter: 1
    };

    render () { // Needs to be implemented in class-based components! Needs to return some JSX!
```

```
.           return (
.               <div>{this.state.counter}</div>
.           );
.       }
.   }
```

Here, the `NewPost` component contains `state`. Only class-based components can define and use `state`. You can of course pass the `state` down to functional components, but these then can't directly edit it.

`state` simply is a property of the component class, you have to call it `state` though - the name is not optional. You can then access it via `this.state` in your class JSX code (which you return in the required `render()` method).

Whenever `state` changes (taught over the next lectures), the component will re-render and reflect the new state. The difference to `props` is, that this happens within one and the same component - you don't receive new data (`props`) from outside!