REACT

COMPONENTS

The concept of components is not limited to just React

There are many other libraries which use components

What are components?

- Re-usable pieces of the UI
- Contain React nodes
- We build our application from multiple components
- We can organise our components so they are grouped together with their own tests and CSS styles

An example of components in use

```
<script type="text/template">
    <div>
        <NavBar />
        <main className="container">
          <Counters />
        </main>
    </div>
</script>
```

Source: counter-app

" Fun fact, there's nothing technically stopping you from writing your entire application as a single React Component. It can manage the state of your whole application and you'd have one giant render method... I am not advocating this though... Just something to think about :) " - Kent C. Dodds

Source: Prop Drilling

React has 2 type of components

Functional

```jsx
function Welcome(props) {
    return <h1>Hello, {props.name}</h1>;
}
```

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

```jsx
const Welcome = (props) => <h1>Hello, {props.name}</h1>;
```

Class based

```
class Welcome extends React.Component {
  render() {
    return <h1>Hello, {this.props.name}</h1>;
  }
}
```

```
class Welcome extends Component {
  render() {
    return <h1>Hello, {this.props.name}</h1>;
  }
}
```

Class based

```
class Clock extends Component {
  constructor(props) {
    super(props);
    this.state = { class: 'FBW6' };
  }

  render() {
    return (
      <div>
        <h1>Hello, class {this.state.class}!</h1>
      </div>
    );
  }
}
```

Modern Class based

- without the constructor

```
1  class Clock extends React.Component {
2
3    state = { class: 'FBW6' };
4
5    render() {
6      return (
7        <div>
8          <h1>Hello, class {this.state.class}!</h1>
9        </div>
10     );
11   }
12 }
```

Side note...

You may remember from vanilla JavaScript the term ***constructor***. We invoke constructors with the ***new*** keyword. All components are technically constructors (except we do not need to use ***new***, - React does this for us)

React does a lot of things for us!

When to use a **functional** or a **class based** component?

Honestly, it's not very clear

It used to be that you only used **functional** components if you didn't need component state

But since the release of React Hooks, the line is blurred - Hooks allow functional components to store state

Components and data

Components can receive data from their parent (props) or can handle their own data (state)

Components that manage their own state are known as *stateful* components

Props

Short for ***properties***

These are passed from the parent, usually as attributes from the JSX

```
function Form() {
    return <clickable text="Click here!">;
}
</clickable>
```

```
function Clickable(props) {
    return <button>{props.text}</button>;
}
```

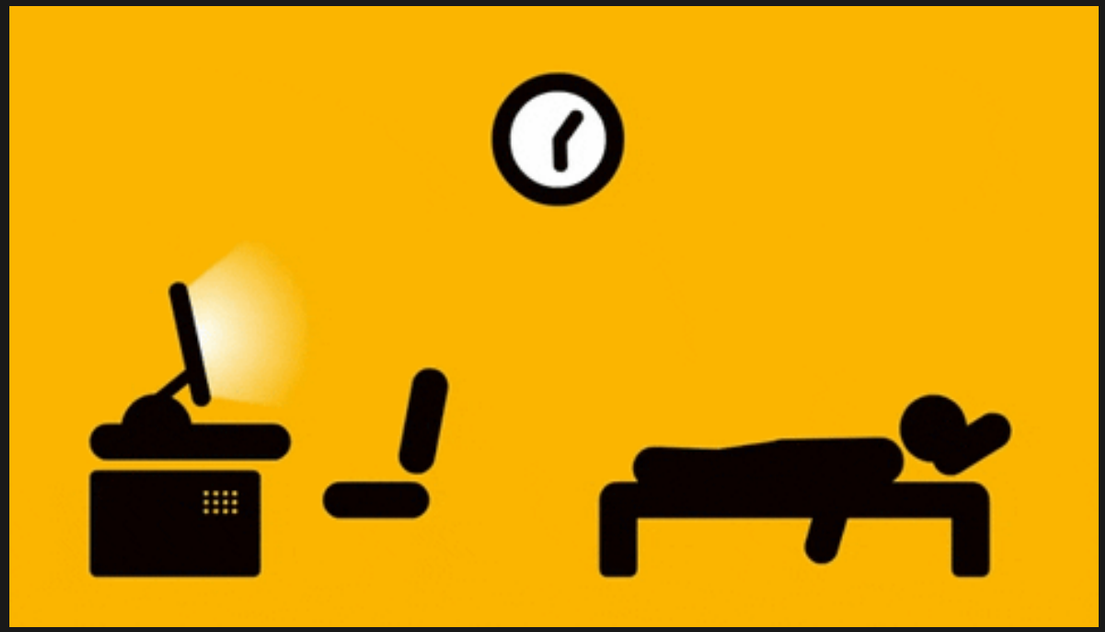
Children as props

Components have a reserved attribute called ***children*** which is used to pass JSX through to other components

```
function Header() {
return
    (<login>
        <h1>Woohoo!</h1>
    </login>);
}
```

```
function Login(props) {
    return <div>{props.children}</div>;
}
```

Lifecycle methods



Every component has a lifecycle

Lifecycle essentially refers to the ***birth***, ***life*** and ***death*** of the component

Lifecycle methods are functions which are called when certain time

Why are lifecycles important to us?

- It helps us understand what the component is doing
- It gives us control over the component at different stages in its life
- By understanding lifecycles, we can understand React Hooks better

Only with class components can we use lifecycle methods directly

For functional components, we can not use lifecycle methods, but instead (since 2018/19) we use React Hooks

Initialization

setup props and state

Mounting

componentWillMount

render

componentDidMount

Updation

props

componentWillReceiveProps

shouldComponentUpdate

true

false

componentWillUpdate

render

componentDidUpdate

states

shouldComponentUpdate

true

false

componentWillUpdate

render

componentDidUpdate

Unmounting

componentWillUnmount

We've already been using a lifecycle method

render()

As the name suggests, it handles the rendering of your component to the UI. It happens during the mounting and updating of your component.

render() should be pure with no side effects

Lifecycles related to **mounting**

birth of your component

1. componentWillMount()
2. render()
3. componentDidMount()

Loosely relates to the DOM methods ***append()*** or ***appendChild()***

Lifecycles related to component **updates**

life of your component

1. componentWillUpdate()
2. render()
3. componentDidUpdate()

Lifecycles related to **unmounting**

death of your component

1. componentWillUnmount()

Loosely relates to the DOM method ***remove()***

Lifecycles and state / props

When the data for a component changes, that component must be re-rendered

1. When you update state, you are also re-rendering the component
2. When the props change, you are also re-rendering the component

Use Cases

- starting or stopping animations
- setting or cancelling timers
- performing or cancelling network requests
- updating 3rd party UI libraries

Advanced concepts

We can *also* describe components as being:

- High level
- Low level

High level components exist higher up the hierarchy chain

- They handle more responsibilities (logic / data)

Low level components exist lower down the hierarchy chain

- They have no responsibilities
- They must receive their data from a parent, and can not affect the state of the application
- We can also refer to them as ***dumb*** components

Further reading: Smart and dumb components in React