

the Master Course

{C0DENATION}

Introduction to **React**.

{CODENATION}

Learning Objectives

To understand what React is and why we would use it.

To be able to create your own components and understand what props are.

React.js

What is React?

A Javascript library for building user interfaces.

React.js

What is React?

Using React we build a user interface with **components**, which we can easily reuse anywhere in our application.

React.js

By using these independent, reusable & isolated components our code is super easy to manage and keep up to date.

React.js

React uses a special syntax called **JSX (although this is not compulsory!)**

Using a compiler we can make Javascript **look like HTML! We use JSX to create our own custom HTML tags. **Magic.****

React.js


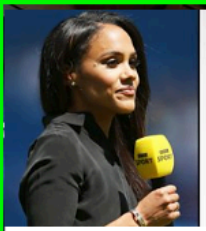
Well not quite magic. At the end of the day, this JSX HTML-looking code, is just converted into standard Javascript.

React.js

Let's have a look at some webpages which use react, and how they split the UI into components.


← → ↻ 🔒 https://www.bbc.co.uk 🔍 ☆ 📺 B

NEW Women's World Cup 2019 >



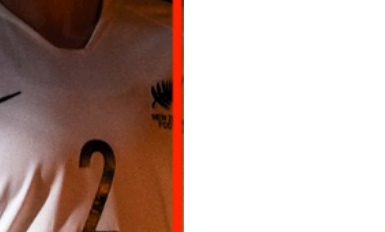
Stars predict which team is heading for glory

WOMEN'S FOOTBALL



Is the Women's World Cup a turning point for France?


WOMEN'S FOOTBALL



Can Scotland win the Women's World Cup?


WOMEN'S FOOTBALL

Real-life stories >



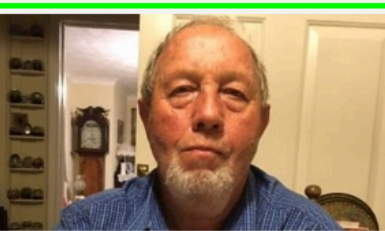
The double agent who hoodwinked Hitler

BBC RADIO 5 LIVE



21-year-old claims to be youngest to visit every country

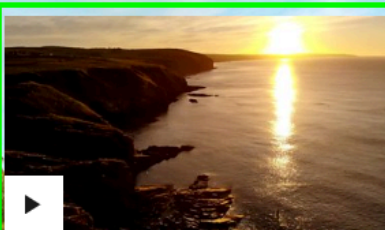
NEWSROUND



'My name stopped me buying drinks in my teens'


ENGLAND

Springwatch 2019 >



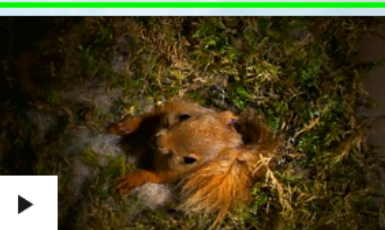
Phenomenal beauty of UK coastline is 'so vulnerable'

BBC TWO



LIVE Get close to nature on the #Springwatch wild cams

BBC TWO



A fluffy red squirrel tucks itself into a cosy blanket

BBC TWO

React.js

Why use react?

We could just hard code everything using HTML and JS, but think how much we would be repeating ourselves!

React.js

Why use react?

Working with the actual DOM directly can become difficult with complex UI's or larger applications.

React.js

Why use react?

React is efficient, fast and makes dynamically updating elements much easier.

React.js

It's all about the components!

So this is where we're going to start.





React.js

What is a component?

In simple terms, it's either a javascript **function** or **class** which returns a piece of the user interface.

React.js

Our components are rendered by React to represent HTML elements (JSX), but these elements are really just Javascript objects!



React.js

Remember we can build our components in isolation and then put them all together.



React.js

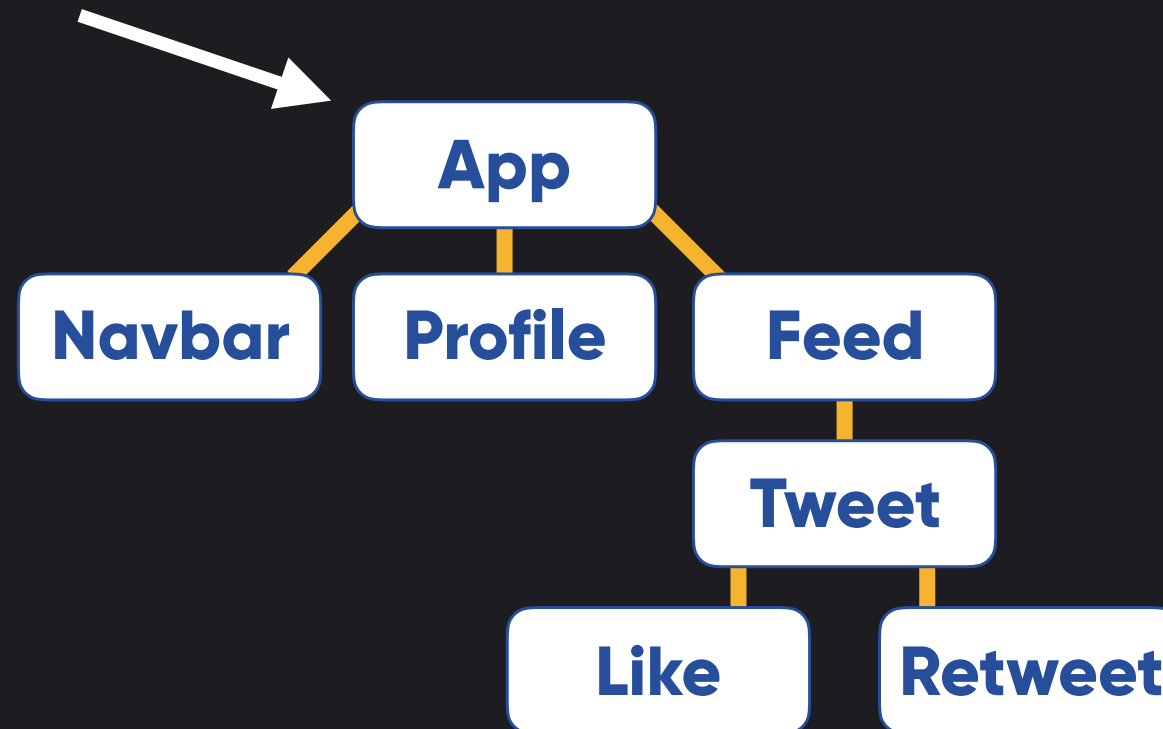
Our components form a tree structure or hierarchy, with one main (root) component.





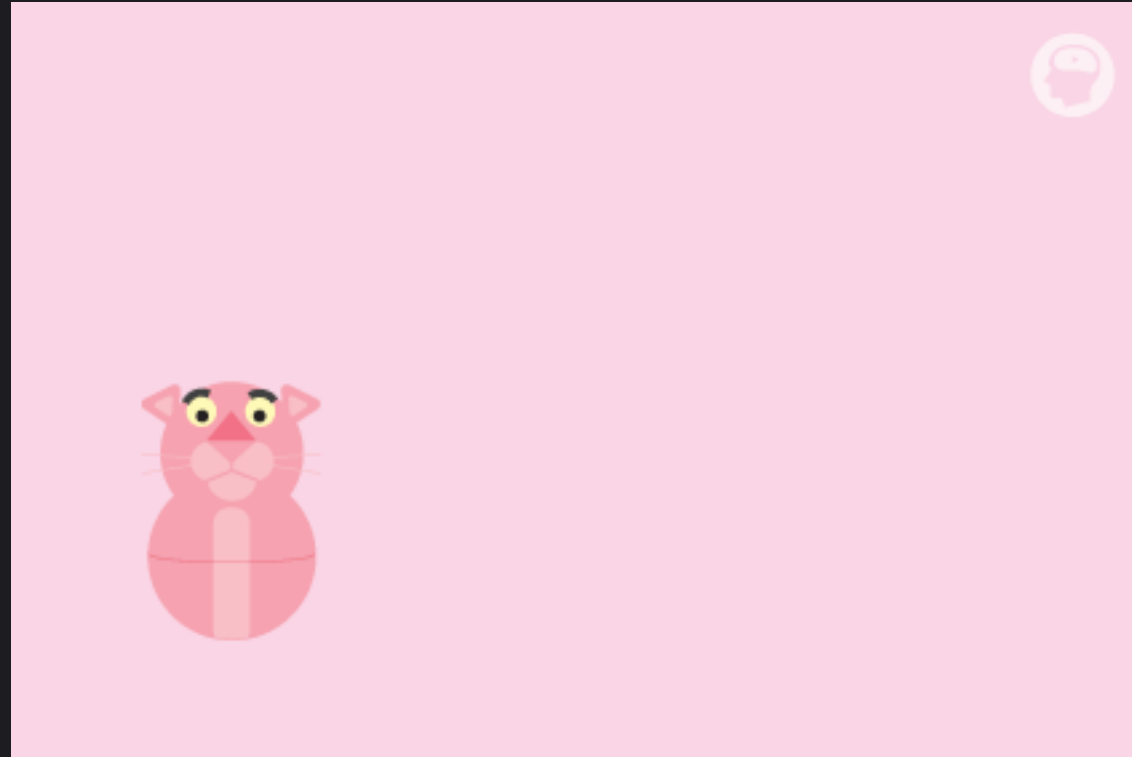
Component tree

Root component





Think of it like this:



React.js

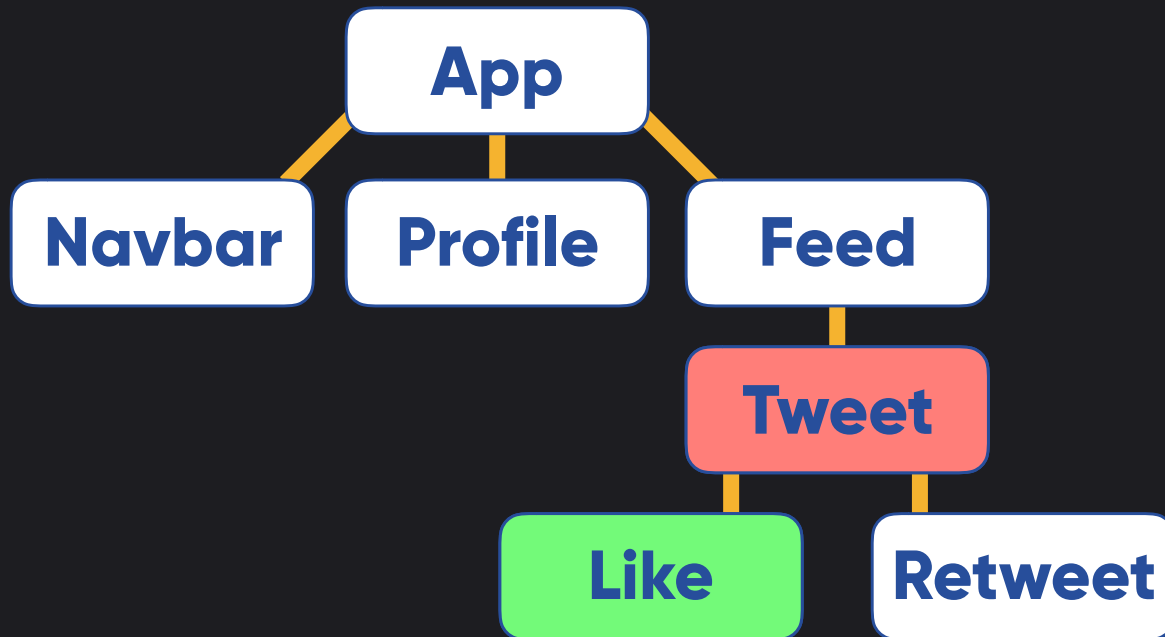
React creates a virtual DOM, which is a lightweight representation of the actual DOM, stored in memory.



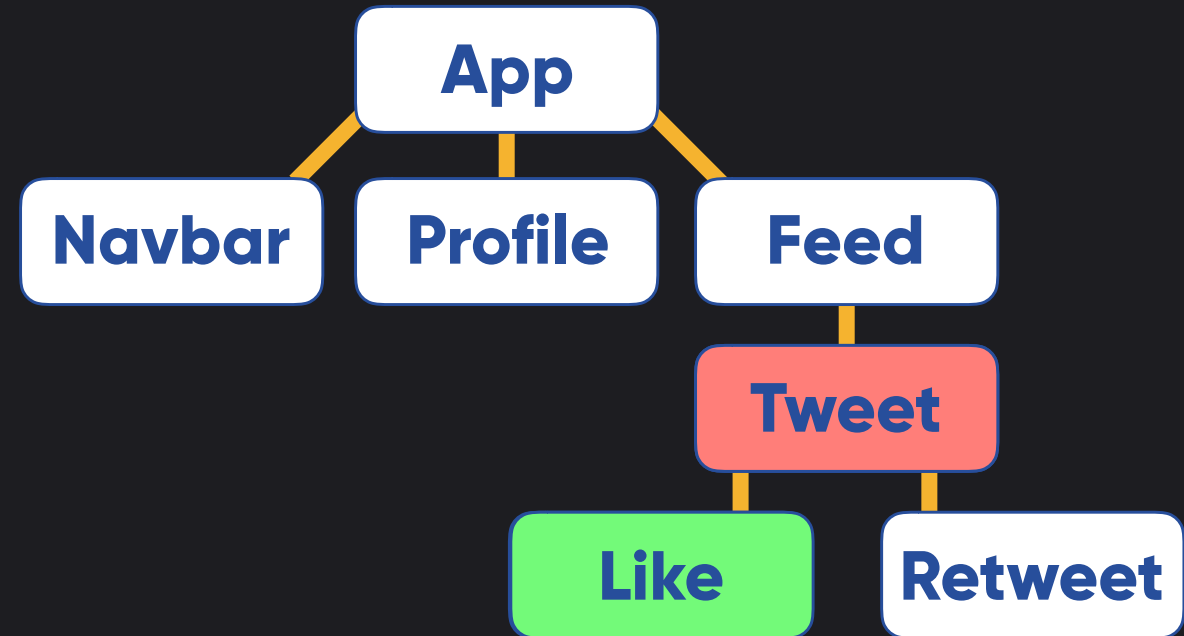


Component tree

Virtual DOM



Actual DOM



React.js

So when our app changes, **React compares the virtual DOM to the actual DOM**. If there's a difference, the actual DOM is updated to keep it in sync.



React.js

So what does this actually mean? Why is this so exciting?

React.js

**We no longer have
to work with the
DOM API in browsers.**

React.js

So no more

document.getElementById....

React.js

If we make a change to our UI, react re-renders the necessary component which updates the real DOM.

React.js

It reacts.

Get it?

React.js

A component is either a pure Javascript function, or a javascript class. Let's have a look.

```
//functional component
const Person = () => {
  return (
    <div>
      <h1>I'm a functional component</h1>
    </div>
  )
}
```

This component is a function which returns some JSX. **It looks like HTML, but it's not.** It is converted to Javascript.

```
//functional component
const Person = () => {
  return (
    <div>
      <h1>I'm a functional component</h1>
    </div>
  )
}
```

Note that the return statement is wrapped in normal brackets. This is standard in JS when our return statement is written over multiple lines.

//functional component

```
const Person = () => {  
  return (  
    <div>  
      <h1>I'm a functional component</h1>  
    </div>  
  )  
}
```

We use **capital letters** when naming our components.

//functional component

```
const Person = () => {  
  return (  
    <div>  
      <h1>I'm a functional component</h1>  
    </div>  
  )  
}
```

When returning JSX, there **must be ONE parent element**. In this case it's a div element. (Russian dolls!)

React.js

Every time we see a custom HTML tag in React, it's just a **React method in disguise.**

React.createElement()

React.js

Behind the scenes React uses a compiler called **Babel, which turns our JSX back into vanilla Javascript for us.**

React.js

Over to CodeSandbox.io

{ CODENATION }

React.js

```
ReactDOM.render(<App/>, document.getElementById('root'))
```



This is the main component that will be rendered.

React.js

The awesome thing about react, is that we can render components, inside other components!

```
const Person = () => {  
  return (  
    <div>  
      <h1>I'm a functional component</h1>  
    </div>  
  )  
}
```

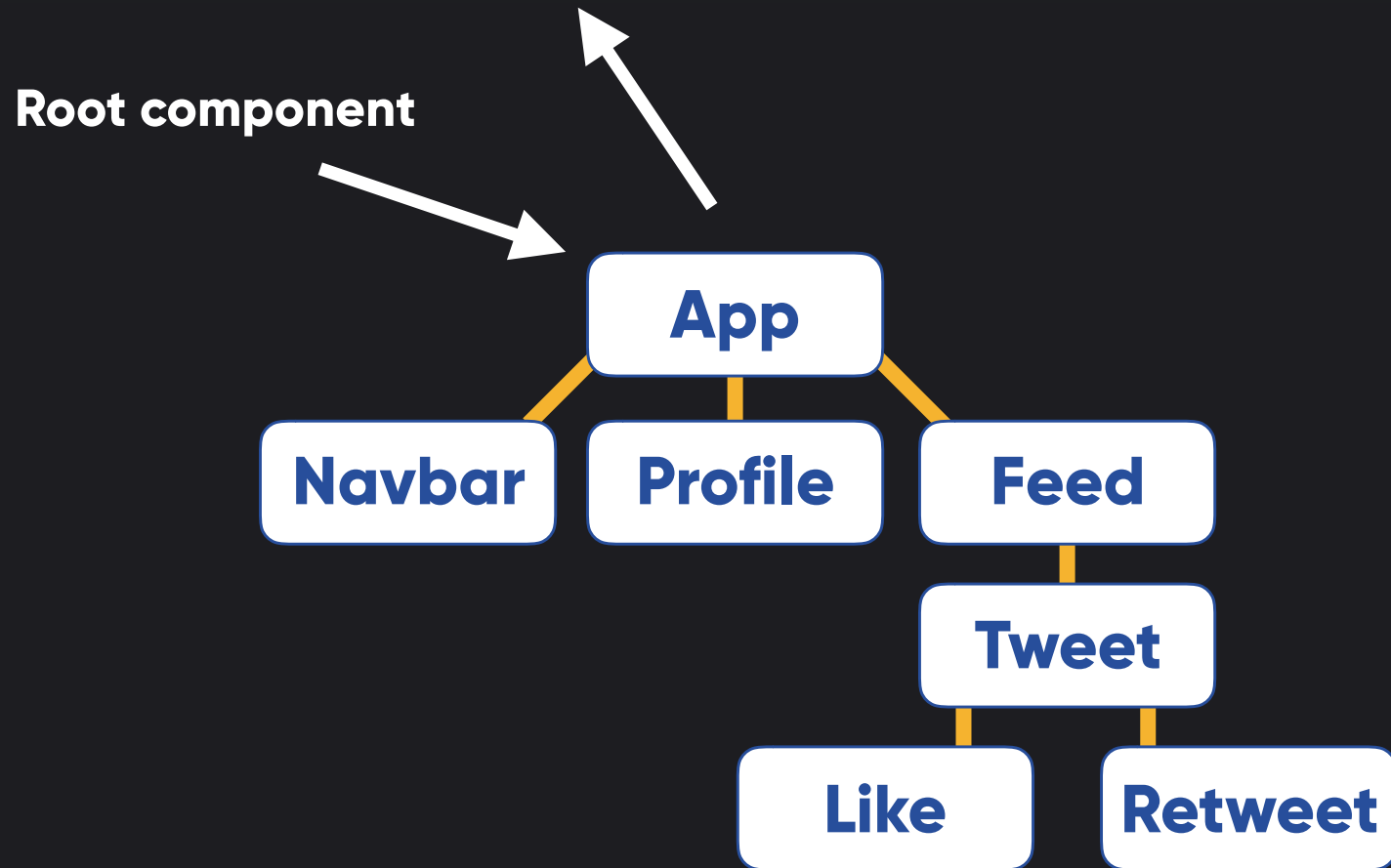
```
ReactDOM.render(<Person/>, document.getElementById('root'))
```

TASK: Try creating a few different functional components and rendering them to the root div.



That also means we only need to call the `ReactDOM.render()` method once.

```
ReactDOM.render(<App/>, document.getElementById('root'))
```



React.js

Class based components are slightly different to functional components, but hopefully they will look familiar, as you've done classes in JS already.

```
//class component
```

```
class App extends React.Component {  
  render(){  
    return(  
      <div>  
        <h1>I'm a class component</h1>  
      </div>  
    )  
  }  
}
```

In React there is a class called Component.
We are using the **extends** keyword like we
did back in week 1.

```
//class component
```

```
class App extends React.Component {  
  render(){  
    return(  
      <div>  
        <h1>I'm a class component</h1>  
      </div>  
    )  
  }  
}
```

Class based components use the render() method. Remember that **classes in Javascript can have properties and methods. We'll look at this in more detail as we progress.**

```
//class component
```

```
class App extends React.Component {  
  render(){  
    return(  
      <div>  
        <h1>I'm a class component</h1>  
      </div>  
    )  
  }  
}
```

Inside the render() method we have a **return statement** like in our functional components.

```
class App extends React.Component {  
  render(){  
    return(  
      <div>  
        <h1>I'm a class component</h1>  
      </div>  
    )  
  }  
}
```

```
ReactDOM.render(<App/>, document.getElementById('root'))
```

TASK: Now try creating a few different class components and rendering them to the root div.

React.js

```
const Person = () => {  
  return (  
    <div>  
      <h1>I'm a functional component</h1>  
    </div>  
  )  
}
```

```
class App extends React.Component {  
  render(){  
    return(  
      <div>  
        <h1>I'm a class component</h1>  
        <Person />  
      </div>  
    )  
  }  
}
```



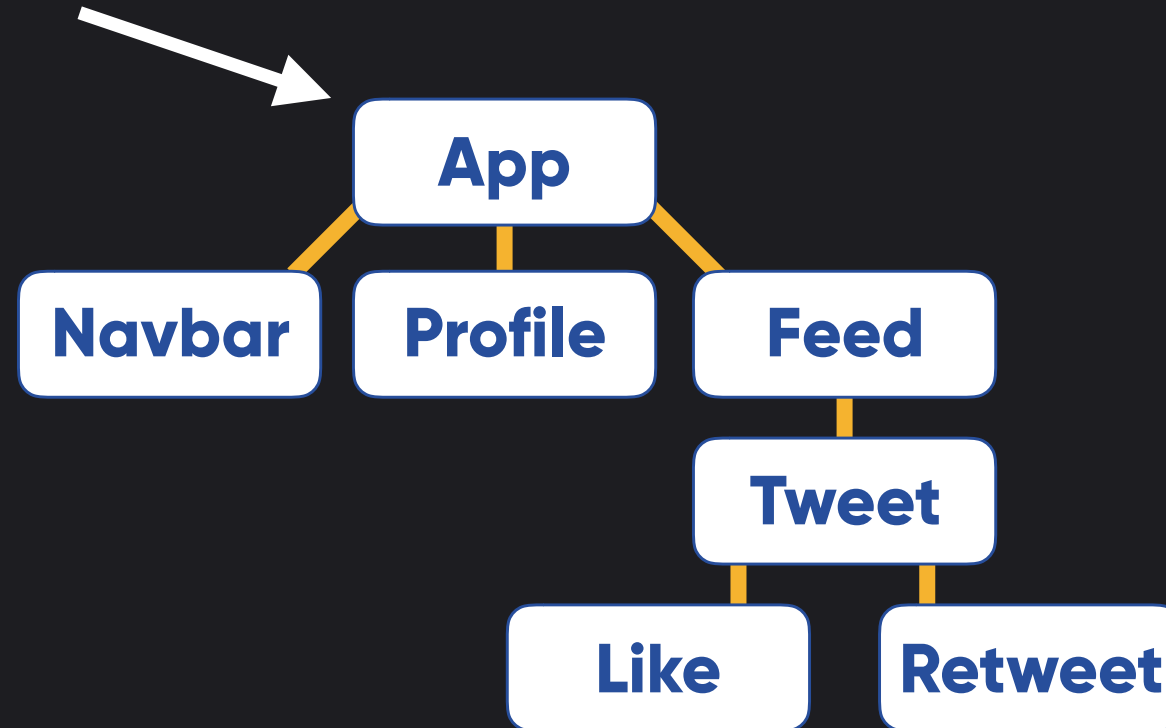
Custom HTML elements

```
ReactDOM.render(<App />, document.getElementById('root'))
```



Remember earlier, when we mentioned React apps have a single **root component**. Now you know how to make one. Everything else can be rendered inside it.

Root component



React.js

Task: Render a functional component 3 times inside a root class component.

React.js

Custom HTML elements can be self-closing or not.

1. **<Person />**

2. **<Person> </Person>**

```
const Person = () => {  
  return (  
    <div>  
      <h1>I'm a functional component</h1>  
    </div>  
  )  
}
```

```
class App extends React.Component {  
  render(){  
    return(  
      <div>  
        <Person />  
        <Person />  
        <Person />  
      </div>  
    )  
  }  
}
```

```
ReactDOM.render(<App />, document.getElementById('root'))
```

You should have ended up with something like this. The Person component is being rendered 3 times inside the App component.

React.js

Remember the websites we looked at which use react. They had the same components being repeated, but they had different text, or images.

Although the core component was the same, the data being passed to them was different.

React.js

**So we use the same core component
but pass different data to each one.**

Let's have a look at how we might do that.

React.js

**What do you remember
about HTML attributes?**

```
class App extends React.Component {  
  render(){  
    return(  
      <div>  
        <Person name="Dan" age = "33"/>  
        <Person name = "Ben" age = "21"/>  
        <Person name = "Stuar" age = "30-something"/>  
      </div>  
    )  
  }  
}
```

```
const Person = (props) => {  
  return (  
    <div>  
      <h1>My name is something</h1>  
    </div>  
  )  
}
```

```
ReactDOM.render(<App />, document.getElementById('root'))
```



React.js

In JSX, these HTML-like elements have attributes, but they behave a little differently.

React.js

When react renders the JSX and turns it into standard JS, it turns the attributes on our custom HTML elements into a JS object.

React.js

We refer to this object as **props.**

And the **props object is passed to our components as a function argument.**

```
class App extends React.Component {
```

```
  render(){
```

```
    return(
```

```
      <div>
```

```
        <Person name="Dan" age = "33"/>
```

```
        <Person name = "Ben" age = "21"/>
```

```
        <Person name = "Stuart" age = "30-something"/>
```

```
      </div>
```

```
    )
```

```
  }
```

```
}
```

```
const Person = (props) => {
```

```
  return (
```

```
    <div>
```

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

```
    </div>
```

```
  )
```

```
}
```

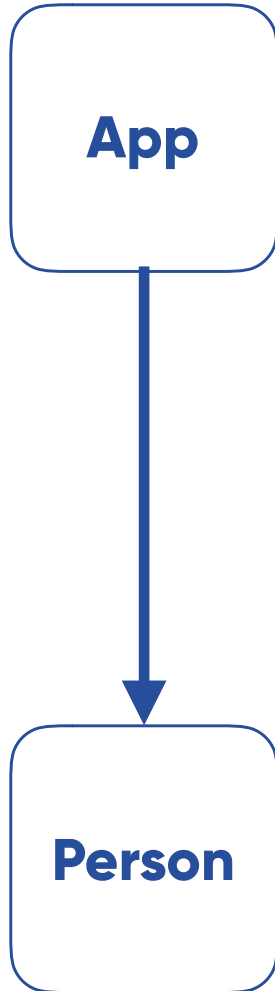
```
ReactDOM.render(<App />, document.getElementById('root'))
```

**props = {
 name: "Dan",
 age: "33"
}**

React.js

Passing props is how we pass data down the hierarchy of components.

React.js



Data flows down the component tree through props

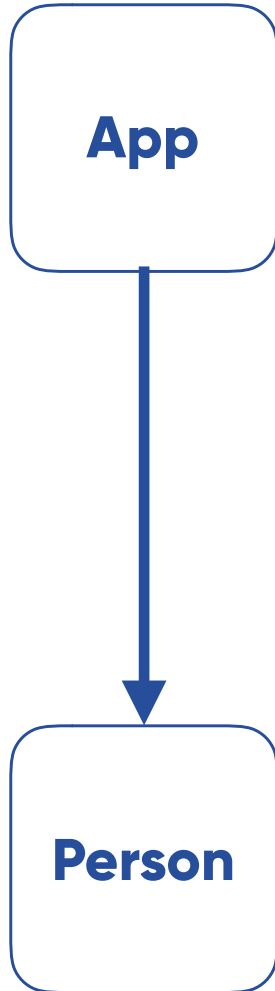
Task: Create a functional component called **Person** which returns a string "Hi my name is"

Create a main class component called **App** which renders the **Person** component.

Give the **Person** component a property called **name = "Your Name"**

Pass the props object to your functional component and use the object data inside your string, to display "Hi my name is **Your Name**"

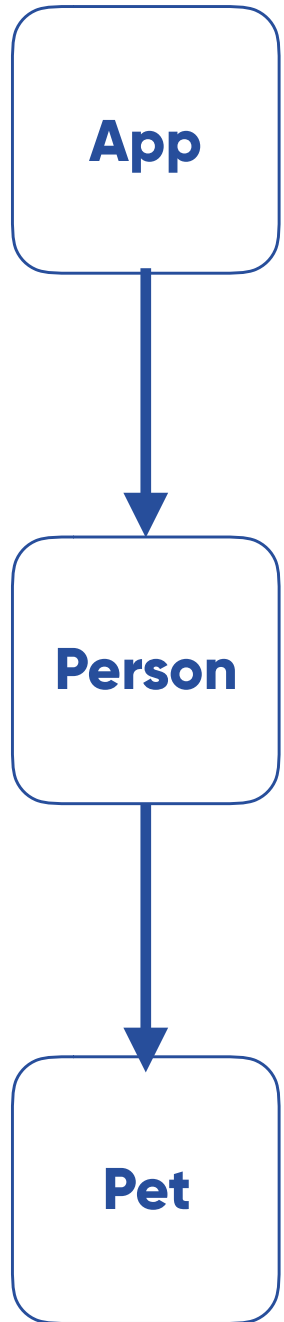
React.js



Data flows down the component tree through props

React.js

Pet's name property



**Data flows down the component hierarchy,
and we can keep passing props along.**

React.js

Task: Create another functional component called **Pet**, and make it return a **h4** tag with the text **"My pet's name is"**. Render this second functional component, inside the first functional component.

I want you to get the data (the pet's name) from the **App** component, through the first functional component, and then to the **Pet** component by passing props down the hierarchy.


```
return(  
  <div>  
    <Person name="Dan" age = "33" pet = "Polly"/>  
    <Person name = "Ben" age = "21" pet = "john"/>  
    <Person name = "Stuart" age = "30-something" pet = "sam"/>  
  </div>  
)  
}
```

```
const Person = (props) => {  
  return (  
    <div>  
      <h1>My name is {props.name}</h1>  
      <Pet petsName = {props.pet} />  
    </div>  
  )  
}  
  
const Pet = (props) => {  
  return (  
    <div>  
      <h6>My pet's name is {props.petsName}</h6>  
    </div>  
  )  
}
```



React.js

Recreate components

Task: I am going to send you a picture on slack. You need to decide how you might break the image into components, then put your components together so they match the image.



Revisiting Learning Objectives

To understand what React is and why we would use it.

To be able to create your own components and understand what props are.