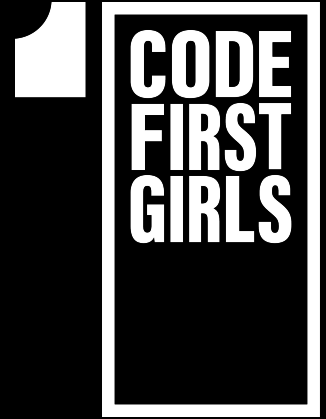


# **WELCOME TO CFG** **YOUR INTRODUCTION** **TO JAVASCRIPT**



**TECH SHOULDN'T JUST BE A BOYS CLUB.**

# COURSE JOURNEY

MODULE 7: JAVASCRIPT

INTRO  
JAVASCRIPT

**MODULE 01**

CONDITIONS  
& LOGIC

**MODULE 02**

THE DOM

**MODULE 03**

INTRO  
REACT

**MODULE 04**

REACT  
COMPONENTS

**MODULE 05**

STYLING  
COMPONENTS

**MODULE 06**

STATES &  
EVENTS



**MODULE 07**

PROJECT  
PRESENTATION

**MODULE 08**

**Behaviour & Events**

**What is State?**

**Component State**

**Hooks**

# Walkthrough: What is behaviour?

## ✉ And how does it relates to components?

- It effectively is what we learnt during our JavaScript course - we simply write a set of instructions that the component must follow depending on the event
- For example, we can create a **onClick** function and assign it to the button component
- Once assigned, we can tell the component that every time it's clicked, it must follow whatever is within **onClick**
- A primary difference to JS is that these 'behavioural' functions are **normally stored within the component file as well** - effectively **centralising** a lot of code together (the component file is already **.js**!)

# YOUR INSTRUCTOR WILL EXPLAIN THE VARIOUS PARTS AND THEIR PURPOSE

```
const Button = ({ message = "" }) => {  
  const clicked = () => {  
    console.log('You clicked me!')  
  }  
  
  return (  
    <button className="button" onClick={clicked}>  
      <h2 className='button__text'  
        {message}  
      </h2>  
    </button>  
  );  
}
```

# Exercise: Adding a onClick for React



*Time to do it on your end (if not done)*

- Similar to the example before, ensure that your button outputs some sort of a message when pressed!
- Learning from the previous slide, you'll need to add the correct property/s and design a correct mechanism



You have approx. **8 minutes** for this (depending on your instructor's discretion + current time!). Google when you can!

# EXAMPLE OUTPUT

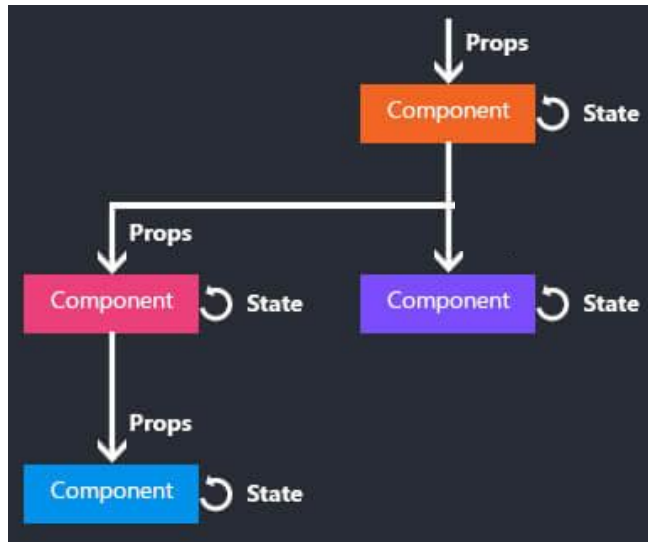


# Introducing State

✂ *One of React's most coolest parts!*

- Components have a paired element to them called **the state** - this is a object that contains information and attributes about the component
- This state can change over time (e.g. through modification, perhaps we increment a value that's stored inside it) - when this occurs, **the component re-renders**
- Consider it like the components information bank - I can store the components text, attributes and anything inside its state. The bonus is that when these critical values update, **so does the component**
- **Props** and **state** are very similar - they're objects that store information (which can influence the output of a render)! The difference is that **props get passed to the component** (similar to function parameters) whereas **state is managed within the component** (similar to variables declared within a function).

# EXAMPLE




# Demonstrating the need for State

## ✂ Showcasing its need

- Let's say that I want to create a button whose value increments per click
- The issue is that everytime I click, the variable increments **but this change isn't reflected in the button.** Why?
- Consider carefully as to why the button doesn't show the change - **the answer is why State is important in React**

```
function IncrementButton() {  
  let idx = 0;  
  
  function whenButtonPressed() {  
    idx++  
    console.log("Value of idx is now: " + idx)  
  }  
  
  return (  
    <>  
      <button  
        className="incrementButton"  
        onClick = {whenButtonPressed}  
      >  
        {"The counter currently is: " + idx}  
      </button>  
    </>  
  );  
}  
  
export default IncrementButton;
```

## # CODE AND RENDERED EXAMPLE

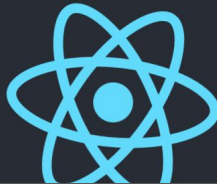


Value of idx is now: 18
Value of idx is now: 19
Value of idx is now: 20
Value of idx is now: 21
Value of idx is now: 22
Value of idx is now: 23
Value of idx is now: 24
Value of idx is now: 25
Value of idx is now: 26
Value of idx is now: 27

of book reading are fairly self-explanatory

but Dune message. They're great!

The counter currently is: 0



Value of idx is now: 13
Value of idx is now: 14
Value of idx is now: 15
Value of idx is now: 16
Value of idx is now: 17
Value of idx is now: 18
Value of idx is now: 19
Value of idx is now: 20
Value of idx is now: 21
Value of idx is now: 22
Value of idx is now: 23
Value of idx is now: 24
Value of idx is now: 25
Value of idx is now: 26
Value of idx is now: 27
Value of idx is now: 28
Value of idx is now: 29

# Walkthrough: State within class components

✂ *Actually using state properly now!*

- Now that we have our class component, we can start using state properly
- Take a look at the right-hand side code; what does A, B or C mean? Before your instructor explains, take a moment to consider what each part is and why we need something like **setState**
- Consider points like - why is it bad to change the state directly? Why can the state only be changed through **setState** ideally?

```
constructor(props) {  
  super(props);  
  
  this.state = {valueKey: actualValue};  
  
  this.whenButtonPressed = this.whenButtonPressed.bind(this);  
}  
  
whenButtonPressed() {  
  
  console.log(this.state.counterValue)  
  
  this.setState({  
    valueKey: newValue  
  });  
}
```

A

B

C



# Walkthrough: State in functional components

Class components have a lot of **functionality out the box** but can be harder to read and understand.

Thankfully there's a way of adding this same functionality (and more) to functional components using Hooks!



```
import React, { useState } from 'react';  
  
const Button = () => {  
  
  const [counterValue, setCounterValue] = useState(initValue);  
  
  const triggeredEvent = () => {  
  
    console.log(counterValue)  
  
    setCounterValue(newValue) }  
  }
```

A

B

C

## Exercise 2: Adding state to our button

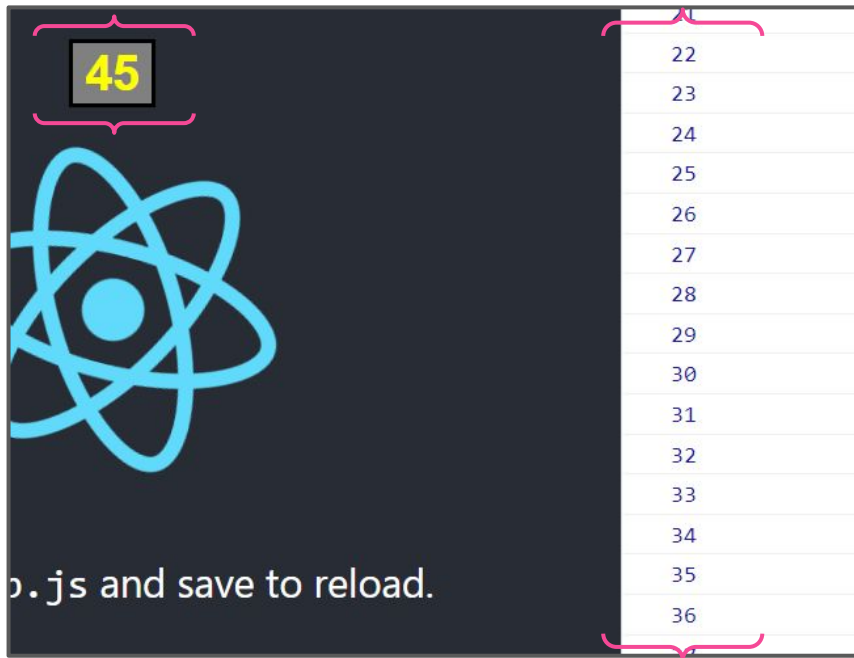
**Now we want to ensure that our button displays a counter that increments per click!**

- Modifying your previous component button earlier, make sure it now displays a counter that increments every time its clicked
- You'll need to use state for this - using a normal variable will not work as you need the component to re-render every time (in order to display the new increment value)
- The appearance of the button can be anything - the only critical part is that it's value increments per click

You have approx. **15 minutes** for this (depending on your instructor's discretion + current time!). Google when you can!



## # EXAMPLE OUTPUT



# COMMON HOOKS

React comes with a bunch of useful hooks for all sorts of purposes. The two main ones you'll come across are:

## **useState**

used to give functional components state functionality

## **useEffect**

used to re-render functional components based on an array of values

There are many other useful hooks but you'll use these far less often (it's just handy to know they exist)

Other common hooks:

**useMemo, useCallback, useRef**

As well as React's own hooks there are also 3rd party hooks you can import and use as well as the ability to write your own hooks! This may not sound useful on the face of it but it has its uses!

# DO I NEED STATE?

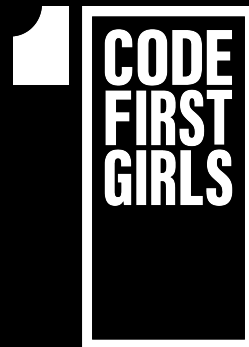
So we now know two main methods of using data in components:

- **State** is data stored in each individual component
- **Props** are data passed into each individual component

**WARNING:** I guarantee you will get tripped up by props/state being out of sync or not being set/stored correctly in a specific place! For this reason use state sparingly in favour of props



# SUMMARY



- Components can have data assigned to them through States - these are just like variables, except changes to them trigger component re-renders (allowing us to update it dynamically)
- Components can also be written in a class format - this is necessary if we want to use state within the component out the box
- Functional components need a little more to enable state, this is completed with Hooks... more specifically the useState hook.
- State allows data to be used and updated inside the component and can be utilised for things like events where you want a user interaction to change the behaviour.

# HOMEWORK

## + Homework Task

Add state and events to your app. Think about the parts of your site that could utilise it and add it at the relevant level (higher in the app hierarchy if it's shared state across multiple components otherwise as low down as possible).

**THANK YOU**  
**HAVE A GREAT**  
**WEEK!**

