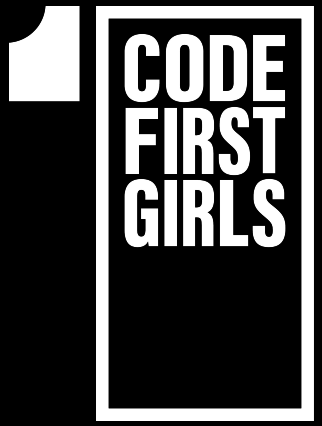
WELCOME TO CFG
YOUR INTRODUCTION TO JAVASCRIPT



TECH SHOULDN'T JUST BE A BOYS CLUB.

COURSE JOURNEY

MODULE 01	MODULE 02	MODULE 03	MODULE 04	MODULE 05	MODULE 06	MODULE 07	MODULE 08
INTRO JAVASCRIPT	CONDITIONS & LOGIC	THE DOM	INTRO REACT	REACT COMPONENTS	STYLING COMPONENTS	STATES & EVENTS	PROJECT PRESENTATION

06 JAVASCRIPT

Adding a little style

CSS Styling

Classnames

Alternative methods to styling

CSS

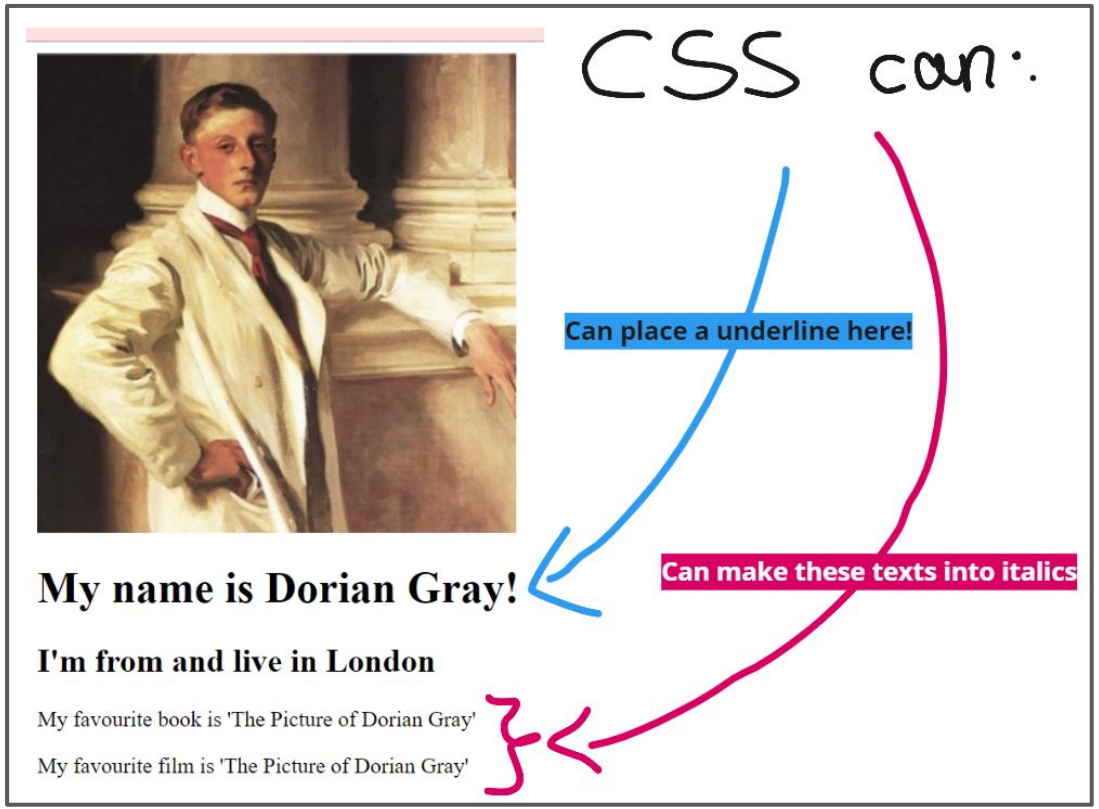
CODE FIRST GIRLS

Introducing CSS

Finally, some style

- Cascading Style Sheet (CSS) is a stylesheet language - it describes how HTML should be displayed (to the browser).
- For example, it can point to a specific element (e.g. the paragraph tag) and say that it should have **bold text** - from that point on, the <p> tag will be displayed as bold!
- Consider HTML (and JSX) as the **bland body you create** (e.g. we should have two eyes!), whereas **CSS is the attributes you set** (e.g. those eyes should be the colour brown!)

EXAMPLE



Exercise: Add content!

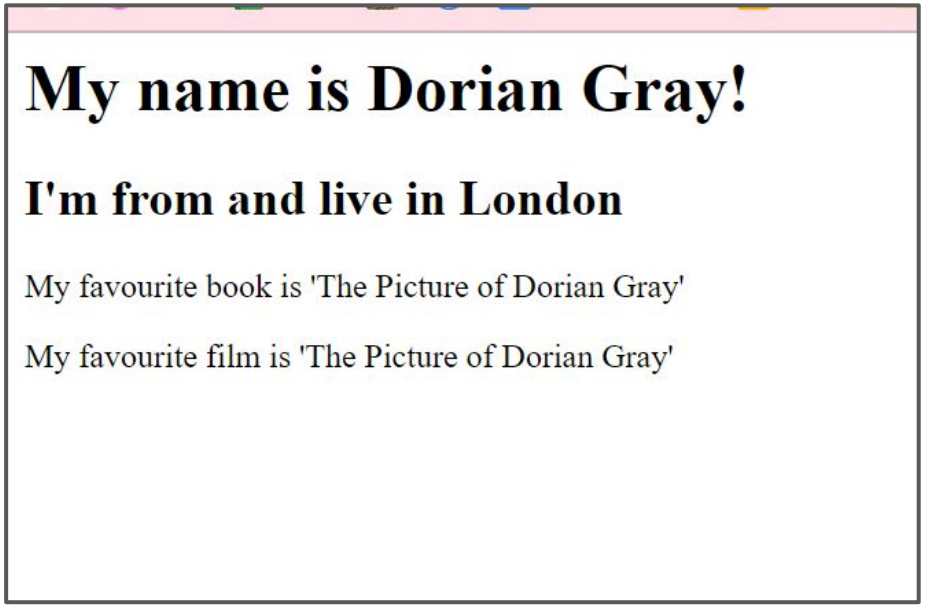
Before we can add style we things to style!

- **Using the following tags....**
- *<h1> (or h2, h3, ..., h5, h6)*
- *<p>*

- **.....add the following information to your website:**
- Your name
- Which city you're from in the world
- Your favourite book
- Your favourite film

- **Use a combination of tags: you must have at least one of headings + paragraph tags**
- **Remember - you can google throughout! (Hint:** Use Mozilla documentation, it's incredibly thorough**)**

EXAMPLE OUTPUT



Part 1 Walkthrough: CSS How-to

Linking CSS to our app!

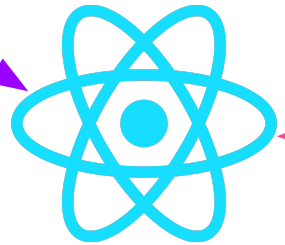
- Create a new file, in the same location as your Button.jsx component, called '**Button.css**' (convention is to call it the same as your component with '**.css**'!)
- CSS needs to be first linked; add this line to your JSX file, after your other imports. This lets React know what CSS file to utilise.
- At this point, the JSX and CSS files are interlinked together! Make sure CSS is in the same location though so we don't have issues with the wrong file addressing

EXAMPLE

```
import React from 'react';
import './Button.css';

const Button = () => {
 return (
   <button type="button">Click Me</button>
 );
}

export default Button;
```



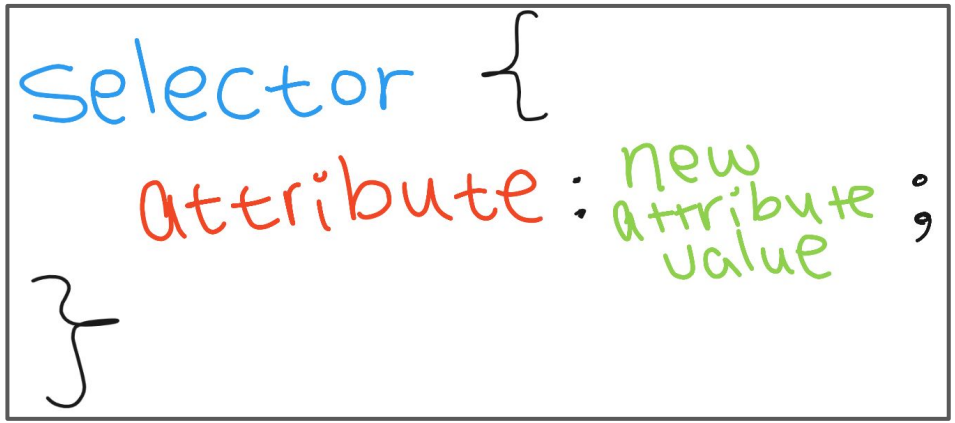
React is now linked to CSS

Part 2 Walkthrough: CSS How-to

Now to change actual parts of our website!

- CSS targets HTML element through a 'selector' - for example if we write in 'p', then it'll **target / select** all paragraph tags
- Inside the curly braces, we can describe what should be changed about our target. For example, we can say that the [attribute] should now be "red"
- For example:
  - **color: "red";**
  - Will make the selected paragraph elements have a red text instead
- Attributes have specific names - you may need to Google to find out whichever ones you need! (e.g. google 'css font italics' to find the attribute for it!)

EXAMPLE



Exercise: Apply CSS!

Knowing everything just taught, change your website!

- **Make the following changes to your website:**
  - Ensure at least one heading has red text
  - Change all paragraph tags to be in italics instead
  - Ensure that at least one heading is underlined too

*You have **up to 10 minutes** for this (depending on your instructor's discretion + time!). Make sure to google the CSS attributes you need (e.g. how to underline text! Or how to change text color to be red!). Most of these will be in the **App.css**.*

REMINDER OF SYNTAX

```
selector {
    attribute : new attribute value ;
}
```

EXAMPLE OUTPUT

Walkthrough: Specific selectors

What if you wanted to specifically target only a few elements? Or just one?

- What if, out of all the paragraph tags, we only wanted to specifically target a subset of them? Normally this is impossible since a ‘p’ would target all paragraph tags.
- However, we can use class or ID selectors to specify our range! We just set the desired HTML tags to have a class / ID as an attribute, then target it accordingly in CSS
- **Classes:** Selected via a dot notation (e.g. ‘.mySpecialParagraphs)
- **ID:** Selected via a # notation (e.g. ‘#myParagraphLine)

EXAMPLE

Exercise: Apply CSS selectively!

Start testing out how CSS can be combined and made specific!

- By this point, you'll likely have multiple HTML elements (e.g. many paragraph tags, many h1, etc - if not, add more right now!).
- **For this exercise, do the following:**
- Make **only** one HTML element have the color blue, be bold and have an underline
- Make **three** HTML elements have the color purple and be in italics (all three must share the same CSS attribute, and it can't be ID).

*You have approx. **10 minutes** for this (depending on your instructor's discretion + time!)*

EXAMPLE OUTPUT

My name is Dorian Gray!

I'm from and live in London

My favourite book is 'The Picture of Dorian Gray'

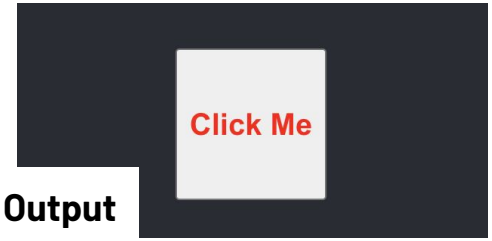
My favourite film is 'The Picture of Dorian Gray'

Classnames are King!

There's a general naming convention to ensure we're naming our classname's uniquely:

Block__Element—Modifier

- A **Block** "encapsulates a standalone entity that is meaningful on its own". So, every React component you write must have a Block class name on its outermost element.
- **Elements** are "parts of a Block and have no standalone meaning". So Element class names should be given to inner... elements.
- **Modifiers** are "flags on Blocks or Elements" which you can use to change the appearance of the modified item.



Output

```
.Button {
 height: 100px;
 width: 100px;
}

.Button__text {
 color: red;
}
```

Button.css

```
import React from 'react';
import "./Button.css";

const Button = () => {
 return (
   <button className="Button" type="button">
     <h2 className="Button__text">
       Click Me
     </h2>
   </button>
 );
}

export default Button;
```

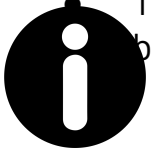
Button.jsx

Dynamic Classnames

Styles can add / hide / show / animate / resize etc. but sometimes it's useful to have these applied conditionally (as *in if x do y*).

You can do this with vanilla JavaScript but it's much easier (and common) using the package `classnames`.

- Go to your app directory in a terminal
- Run `npm install classnames`
- Now you can import this in your component
- This gives us useful utilities to build up classname strings!!



[Always read the doc :)]

```
import ClassNames from 'classnames'
import "./Button.css";

const Button = ({ message = "" }) => {
 const classnames = ClassNames(
   'Button', {
   'Button-hide': message === ""
 })
 return (
   <button className={classnames}>
     <h2 className='Button__text'>
       {message}
     </h2>
   </button>
 );
}
```

If `message` is empty `message === ""` is True

so `classnames` will be set to: `Button Button-hide` applying the styles from **both classes**.

If `message` is set to any string `message === ""` is False

so `classnames` will be set to: `Button`

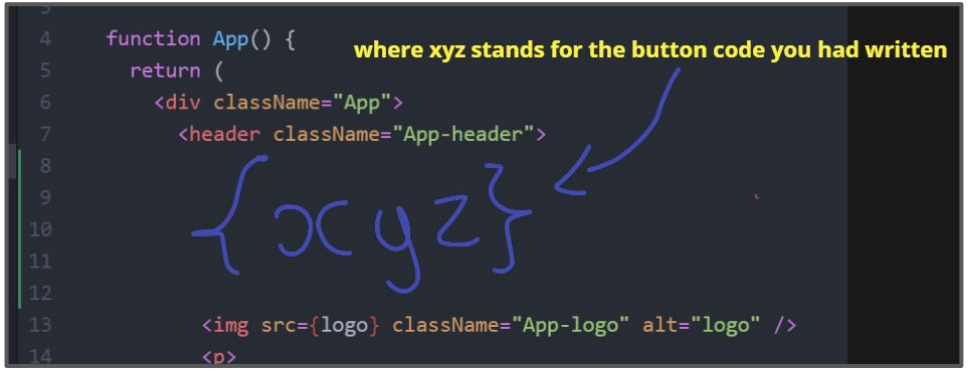
Exercise 3: Beautify your component

Make your button look shiny and nice!

Make sure your button is imported in you App.js

- Making use of a separate CSS file, apply the following effects to your new button:

1. *Text should be bolded and in a different font color (any is fine)*
2. *Button itself should be light blue*
3. *Button should have a thin border surrounding it (any thickness, any color)*
4. *Increase its font size (any is fine)*
5. *Increase the space between the actual button text and the border!*



- **Note that some CSS selectors are different this time (e.g. class keyword!).**

*You have approx. **14 minutes** for this (depending on your instructor's discretion + current time!). Google when you can!*

HOMEWORK

+ **Homework Task**

Add styles to your site, using both generic styles (in the App.css) and specific styles in your components

THANK YOU
HAVE A GREAT WEEK!