React.js theory fun

What is react?

React is a JavaScript library which makes the development of the front end of web apps incredibly efficient. It relies on the mashing up of HTML tags and JavaScript. It is a declarative and component-based JavaScript library which uses functions to render HTML on the page. React makes heavy use of classes and functions, so try to gain a thorough understanding of how they work.

Declarative

Focus of the language is on specifying what will be displayed, but with something like imperative (C) you tell the computer what to do to display that thing. A lot of the control statements are the same between React and C.

Example: changing a like button to be a different colour

Imperative

```
if ( user.likes () ) {
     if ( hasBlue () ) {
          removeBlue ();
          addGrey ();
     } else {
          removeGrey ();
          addBlue();
     }
}
```

Declarative

```
if ( this.state.liked ) {
     return <blueLike />;
} else {
     Return <greyLike />;
}
```

Obviously in the context of UI and design, a declarative approach seems easier and makes more sense.

This could also be understood as going into a restaurant and ordering a meal. You ask the waiter for the seafood pasta, and it is delivered to you sometime later. You do not tell the chef how to cook the pasta or what to use for it. React does all the heavy lifting.

Component Based

A component is a piece of reusable code which takes some parameters and returns a React element which describes how a section of the UI should look. This ties in with being declarative, you do not specify how it should be rendered just what should be rendered.

There are huge advantages to being component based.

- One is that you can easily replace a component/fix the code of a particular piece of the UI
    - No need to search through many lines and replace them
    - If you do not like a component it is easy to rewrite

    - Easy to isolate issues with your code
- Secondly, it is reusable
    - For example, if you want to display a bunch of profiles after searching a database, you simply need to call the component and pass in parameters as opposed to writing out the HTML for each one individually
- Makes source code cleaner and easier to understand
- Means if you make a component for one project you can then go and use that for another project and not have to rewrite any code
    - These are component libraries made by people and make development very quick
    - Material.io has many components which are well made and easy to use, allows you to focus on bigger picture stuff and just putting the pieces together

How does it work?

In very simplified terms, into a component you pass props (properties), then do something with the properties and return a react element. That react . The JSX is compiled into JavaScript. and rendered on the screen, then re-rendered each time there is a change in state to each component.

React uses the compiler Babel.

Overview of react

Main parts of react

React is based around JS which means there are very weird things in it like assigning a function to a constant or being dynamically typed. However with this comes many benefits

Components

Everything in React is a component. A component is a piece of reusable code which defines how one element of the UI should look. Combining these then creates the website. *Well how small should a component be ?* Try and draw a diagram/mock-up of your site then anything you can draw a box around should be its own component. While it seems tedious it means that everything will be more uniform and makes dynamic websites easier to work with and responsive.

There are two main ways for us to define a component, through an ES6 (the second version of JavaScript) JS class or a function definition. A function will take in one object of props and return a JSX expression. These are the only requirements for a function component. Typically with a function definition, the component will be stateless. This just means that it wont be very responsive. However, there are ways to give state to a functional component using hooks, we won't go through that today but this does.

On the other hand we could use a class to define a component. This is a slightly stranger concept but allows us more flexibility and more dynamic and reactive components. Classes allow for state unlike functions.

Props

Props are the arbitrary values that a function first receives when created/rendered. A function cannot update it's props and can only pass values down the 'tree' ensuring a one way data flow from parent to child. Once data is updated, the parent will pass new props to the child and it will be re rendered.

State

State, oh state, everything has a state, sort of. State is similar to props passed into functions, however it is private and controlled purely by that component. React makes updating state very easy, using the function

```
this.setState()
```

and passing in an object (in JSON form) as the parameter. Anything with state has access to this function. This is pretty much only class defined components. Updating state will cause the component to be rendered again. This can be used with text input, as the user types the value displayed in the field is updated to reflect what they are typing then the value is compared in some sort of control sequence.

Libraries

Libraries ties in with components and pretty much everything else in React.js. It is your way of reusing code, importing components and using other people's code. If you want something in react, chances are there is a library for it. Material.io has some very nice-looking components for quick web dev. These are accessed using `import [Object] from [source]` and is done in the same way as importing components.

In order to import a component, you must give permission for that component to be exported from the file it was written in. Each component should be written in its own file and at the end of the file there should be a line saying

```
export default [componentName]
```

which gives permission for that component to be imported.

Syntax

JSX

You do not need to use JSX when coding in React, but it will certainly do you good and make things look pretty.

JSX is a mashing up of JavaScript and XML, tags and programming all in one its fun. JSX compiles into JavaScript, which means it can be used like any expression in JS. We can assign it to variables, return it from functions and use it in control flow. However, when returning JSX it must always be wrapped in one outermost tag.

However, because JSX is kind of JS, we can embed some JavaScript code into it which might come in handy. For example, we could have

```
<h1>Hello, {2 + 2}</h1>
```

which is equivalent to

```
<h1>Hello, 4</h1>.
```

All we need to do to embed JS code is to put it into curly braces. Any JS can go in there, function calls, if statements and other. This links to conditional rendering which we will cover later.

The purpose of JSX is to be 'syntactic sugar' for the `React.createElement` function. That function is useful to be aware of but JSX is much nicer and cleaner.

```
<MyButton color="blue" shadowSize={2}>
  Click Me
</MyButton>
```

Compiles to

```
React.createElement(
  MyButton,
  {color: 'blue', shadowSize: 2},
  'Click Me'
)
```

Parameters for React.createElement are (components, props, … children).
Class/style/color/shadow are all included in props and are formatted in JSON and are accessed like a dict in JS.

However, notice that the function begins with `React.` which hints at this being an object. And of course, React is an object so it must be in scope to use JSX, (even though we aren't using the function itself, our JSX is compiling into it).

Classes

Classes function pretty much the same as classes in any other language. They are a way to create a component but not always necessary. They have extra functionality but are mostly interchangeable. Classes are defined by

`class [className] extends React.Component {…}`

The class name must always begin with a capital letter with convention being camel case. The only method that MUST exist within a class is the

`render()`

method. Inside this there must be a return statement which returns a react element in JSX syntax/format.

When creating a class, we can have a method called

`constructor([props])`

which is called on the instance of class when it is first created. This method pretty much sets up the class. The parameter for this methods is the props object which is automatically passed into constructor. Again remember that props cannot be updated but we can access their values due to the one directional data flow. Constructor is where you set default state values and bind event handlers.

In constructor, we must call the function `super()`. Allows us to define methods for a class. Also, if we want to access `this.props` then we need to pass props into `super`,

i.e `super(props)`. Super calls the constructor of its parent class, and as props are in the parent class and `this` is the context the class is called from, we require this.

Event Handlers

But what are event handlers and events and why do we need to bind them ??

Often times in a web page you will have some sort of user input, whether it be a button to count or a text box to write a username in. Take an example of us having a button in our render method, which will display a button on the web page. This button will have an attribute onClick, which is our event, then it may call a method updateTally(). This method simply adds one to the total (stored in state). In order for this method to know which instance it is performing the change on; it must be bound to this class in constructor using

```
this.updateTally = this.updateTally.bind(this)
```

The reason for this is strange but it is here. Pretty much an issue of how JS uses `this`, unless explicitly specified it will assume `this` is the object in the context with which it was called.

Functions

Functions are the other way to create a component in React. This is a slightly more simple way as it takes in props and returns JSX, there is not too much to deal with in terms of state. However this slightly limits the complexity possible with a component.

To declare a functional component, begin with

```
const [ComponentName] = (props) => {…}
```

and at some point in the function return a react element.

Combining with Next

Next.js is a frontend framework for React dev which allows functionality such as server side rendering and generating static websites.

To create a next app, simply type

```
npx create-next-app
```

then follow the prompts.

Next makes it really easy to begin with react and we will use that for the practical part of today