React counter app

Introduction to the project

Welcome to the first project in the course.
This project is perfect if you are totally new to React. I will go in details with every single thing we do, so that in the next projects you'll be up to speed with the basics.

What we'll build

In this first project we'll build a very simple example of a counter.
We are going to have a simple web page with 4 buttons, and a place where we show the count.
The count starts at zero, and the buttons we'll add will increment the count by 1, 10, 100, 1000 depending on which button is pressed.

We're going to associate one of those values to a button, and we will show it in the button text.

Bootstrap a React app using create-react-app

I'm going to use CodeSandbox in my tutorial. It's a nice service that allows you to work with React directly in the browser.

It's the same as using `create-react-app` locally. You can choose to develop locally, or use CodeSandbox.

The choice is yours.

If you choose to work locally, just start a new React app by using the command `npx create-react-app counter-react`, then go into that folder and run `npm run start`.

Here is the link to create a new CodeSandbox project: https://codesandbox.io/s. You just visit the page, click **React** and start building things right away. There are many other options, and CodeSandbox is an awesome tool for all things JavaScript.

Start with the button component

The React application created by `create-react-app` has a single component, App. CodeSandbox keeps it in `src/index.js`.

As I mentioned one of the main building blocks of the application is a button.

We're going to have 4 of them, so it makes perfect sense to separate that, and move it to its own component:

```
const Button = () => {


}
```

The component will render a button:

```
const Button = () => {
  return <button>...</button>
}
```

and inside of this button we must show the number this button is going to increment our count of.

We'll pass this value as a prop:

```
const Button = props => {
  return <button>+{props.increment}</button>
}
```

Notice how I changed the function signature from `const Button = () => {}` to `const Button = props => {}`. This is because if I don't pass any parameter, I must add `()` but when I pass one single parameter I can omit the parentheses.

Now add `import React from 'react'` on top, and `export default Button` at the bottom, and save this to `src/components/Button.js`. We need to import React because we use JSX to render our output, and the export is a way to make Button available to components that import this file (later on, App):

```
import React from 'react'

const Button = props => {
  return <button>+{props.increment}</button>
}

export default Button
```

Our Button component is now ready to be put inside the App component output, and thus show on the page.

Show the interface

This is the App component at this point, stored in the file `src/index.js`:

```
import React from "react"
import ReactDOM from "react-dom"

import "./styles.css"

function App() {
  return (
    <div className="App">
      <h1>Hello CodeSandbox</h1>
      <h2>Start editing to see some magic happen!</h2>
    </div>
  )
}

const rootElement = document.getElementById("root")
ReactDOM.render(<App />, rootElement)
```

> *A note on semicolons: I don't use semicolons, I think the resulting code is cleaner, and they don't really make any difference in 99.9% of the cases. You can still use them if you prefer.*

Let's remove all that's inside the `div` rendered by App.

```
import React from "react"
import ReactDOM from "react-dom"

import "./styles.css"

function App() {
  return (
    <div className="App">
    </div>
  )
}

const rootElement = document.getElementById("root")
ReactDOM.render(<App />, rootElement)
```

We now import the Button component from the Button file:

```
import Button from './components/Button'
```

and we can now use Button inside our App component, and thus show on the page:

```
function App() {
  return (
    <div className="App">
      <Button />
    </div>
  )
}
```

We pass the `increment` prop to it, so it can show that value inside the button text:

```
function App() {
  return (
    <div className="App">
      <Button increment={1} />
    </div>
  )
}
```

You should now see a button showing up in the page, with a `+1` text on it.
Let's add 3 more buttons:

```
function App() {
  return (
    <div className="App">
      <Button increment={1} />
      <Button increment={10} />
      <Button increment={100} />
      <Button increment={1000} />
    </div>
  )
}
```

and finally we add a counter, which our buttons will increment when clicked.
We store the counter result in the `count` variable, which I declare as a `let` (see here for more info on let if you are unfamiliar with it).

We then print this variable in the JSX:

```
function App() {
  let count = 0

```

```
  return (
    <div className="App">
      <Button increment={1} />
      <Button increment={10} />
      <Button increment={100} />
      <Button increment={1000} />
      <span>{count}</span>
    </div>
  )
}
```

Incrementing the count

Great! Let's now add the functionality that lets us change the count by clicking the buttons, by adding a `onClickFunction` prop. We pass that to the Button component, and we use an `onClick` event handler that intercepts automatically the clicks made on the button, and it calls the `handleClick` function:

```
const Button = ({ increment, onClickFunction }) => {
  const handleClick = () => {
    onClickFunction(increment)
  }
  return <button onClick={handleClick}>+{increment}</button>
}
```

When `handleClick` runs, it calls the `onClickFunction` prop.
Here's App with the new `onClickFunction` prop defined on the Button components:

```
function App() {
  let count = 0

  const incrementCount = increment => {
    //TODO
  }

  return (
```

```
    <div className="App">
      <Button increment={1} onClickFunction={incrementCount} />
      <Button increment={10} onClickFunction={incrementCount} />
      <Button increment={100} onClickFunction={incrementCount} />
      <Button increment={1000} onClickFunction={incrementCount} />
      <span>{count}</span>
    </div>
  )
}
```

Here, every Button element has 2 props: `increment` and `onClickFunction`. We create 4 different buttons, with 4 increment values: 1, 10 100, 1000.

When the button in the Button component is clicked, the `incrementCount` function is called.

This function must increment the local count. How can we do so? We can use hooks (read an intro to hooks on https://flaviocopes.com/react-hooks/).

We import `useState` from React, and we call:

```
const [count, setCount] = useState(0)
```

when we need to use define our count. Notice how I removed the `count` `let` variable now. We substituted it with a React-managed state.

In the `incrementCount` function, we call `setCount()`, incrementing the count value by the increment we are passed.

Why we didn't just update the count value, why do we need to call `setCount()`? Because React depends on this convention to manage the rendering (and re-rendering) of components. Instead of watching all variables and spend time and resource trying to "spy" on values and do something when they change, it tell us to just use the `set*` function, and let it handle the rest.

`useState()` initializes the count variable at 0 and provides us the `setCount()` method to update its value.

We use both in the `incrementCount()` method implementation, which calls `setCount()` updating the value to the existing value of `count`, plus the increment passed by each Button component.

Here's the full code:

```
import React, { useState } from 'react'
```

```
import ReactDOM from 'react-dom'
import Button from './components/Button'

import './styles.css'

function App() {
  const [count, setCount] = useState(0)

  const incrementCount = increment => {
    setCount(count + increment)
  }

  return (
    <div className="App">
      <Button increment={1} onClickFunction={incrementCount} />
      <Button increment={10} onClickFunction={incrementCount} />
      <Button increment={100} onClickFunction={incrementCount} />
      <Button increment={1000} onClickFunction={incrementCount} />
      <span>{count}</span>
    </div>
  )
}

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