# Exercice 1: Your first React application

- Generate a new React applicaton using `create-react-app`.
- Start your application (look for a script called `start` in the file `package.json`).
- Open the `index.js` file and look for the method `ReactDOM.render`.
- Replace the existing `ReactDOM.render` by the following code:

```
const reactElement = React.createElement('div', null, 'Hello world')
const domElement = document.getElementById('root')

ReactDOM.render(reactElement, domElement)
```

The first line creates a new react element with a div (first parameter), no attribute on it (second parameter) and the text 'Hello world' in it (third parameter).

The second line selects an element with the id `root` on our html page.

The last line calls ReactDOM.render to display our react element in our html page, more precisely in our element with the id `root`.

- Make sure you understand the description of those three lines. They are the very basic of every React application.
- To finish, verify your application displays 'Hello world'.

# Exercice 2: JSX

In a real React application, calling `React.createElement` everytime can become quite annoying. To avoid that, React created JSX: a HTML like syntax which is compiled to `React.createElement` calls when you build your application.

- Start your app
- Open the `index.js` file and look for the method `React.createElement`.
- Replace the line with `React.createElement` by the following code:

```
const reactElement = <div>Hello world</div>
```

`<div>Hello world</div>` and `React.createElement('div', null, 'Hello world')` both creates the exact same result in your react app: a div with the string `Hello world` inside. JSX is only a more HTML-like way to create react elements.

- Make sure your app still displays `Hello world`

# Exercice 3: my first React component

In a React application, one way to create a component is to create a function which returns react elements (using `React.createElement` or JSX).

- Create a new file called `MyHello.js` in the `src` folder of your app.
- Inside this file, create a new function `MyHello` (be carefull of the uppercase on the first caracter, it allows React to known your function is a React component).
- Make sure this function returns `<div>Hello world</div>`.

At this point, you defined your first React component. But you're not using it, you only defined it. You can use a React component like any html elements in JSX.

- Go to your `index.js` file
- Replace the line with `const reactElement =` by the following code:

```
const reactElement = <MyHello />
```

At this point, you should have an error saying something like `MyHello is not defined`. You're telling React to create the component `MyHello` (by create it means 'calls the function MyHello'). But your component `MyHello` is defined in `MyHello.js` and here you are in `index.js`. To have access to your component in this file, you have to import it. And to import it, it should first be exported.

- Go back to `MyHello.js`.
- In front of `function MyHello`, add `export default`

By adding `export default`, you're saying 'my file `MyHello.js` has one default export, and it is my function `MyHello`'. Remember: a javascript file can only have one default export.

- Go back to `index.js`
- On the top of your file, add the following code:

```
import MyHello from './MyHello'
```

By doing so, you're saying 'I import what's exported by default from the file `MyHello.js`. This file is located in the same folder as my current file. I stock what I import in a variable called `MyHello`'.

- Make sure your app still displays `Hello world`

# Exercice 4: a component in a component

We've seen we can use JSX to display HTML elements in our app through React components. We can actually use React components inside other React components.

- Create a new file called `MyApp.js` in the `src` folder.

- Inside this file, create a new function `MyApp` and export it by default.
- Import the React component `MyHello` (the same way you did in your `index.js` file)
- Add the following code to your function:

```
return (
  <MyHello />
)
```

This way, you're telling React 'My composant MyApp uses my other component MyHello, which in turns create a div with the string "Hello world" inside of it'.

- Go to the `index.js` file
- Replace the line with `const reactElement =` by the following code:

```
const reactElement = <MyApp />
```

- After this modification, you should have an error in your file, fix it
- Make sure your app still displays `Hello world`

If you've reached this point and understand what you just did in this exercice (using a react component inside another React component), congratulations. You just did what developpers do most of the time in real world React applications !

## Exercice 5: my first props

The word `props` in React designates properties (data basically) you pass to your React components so they can do things with them. In a React component defined with a function like we did, props are always the first parameter of our function. By default, it's an empty object `{}` .

- Go to the `MyHello.js` file
- Add a parameter to your function `MyHello` called `props` .
- Use `console.log` inside your function to log the value of `props` . What do you see ?

At this point, even though we log the props of the `MyHello` component, it shows an empty object, because there is nothing inside of it. To actually pass props to a component, we use a similar syntax to define attributes in html elements.

- Go to the `MyApp.js` file
- Replace the content of your function `MyApp` by the following code:

```
return (
  <MyHello name="world" />
)
```

Here we're saying 'When `MyApp` is created, create `MyHello` component and pass in its props a variable named `name` with the string value `world`'.

- Go back to the `MyHello.js` file
- What do you see in the props now ?

In JSX, to display the value of a variable, you can use the following syntax:

```
{myVar}
```

It is called **interpolation**. `{myVAr}` will be replaced by the value of the variable `myVar`, whatever there is inside of it.

- Instead of displaying `Hello world`, make sure the `MyHello` component displays `Hello` and whatever there is inside of `props.name`.
- Make sure your app still displays `Hello world`

## Exercice 6: my first dynamic props

For now, the props `name` we're passing to `MyHello` is still always the same string `world`. It's possible to change the value of what we pass to a component.

- Go to the file `MyApp.js`
- Add the following JSX to the return of your function:

```
<button type="button">Riri</button>
```

At this point, you should have an error. One constraint of React components is to always return only one root element. Here your template looks something like

```
<button type="button">Riri/</button>
<MyHello name="world" />
```

which makes two root elements.

- In the return of your `MyApp` function, add a `div` which encapsulates `button` and `MyHello`.

To listen to an event with the JSX syntax, we use a similar syntax than with classical html, but with `camelCase`.

```
<button onClick={() => doSomething()}>
```

Notice how the syntax with `{` `}` to define a handler function. This function will be called automatically every time the `click` event will be emitted from the button. Warning: you must always pass a function as an event handler. If instead I wrote

```
<button onClick={doSomething}>
```

Then the function `doSomething` would have been called when my component would have been created.

- Add the following at the top of the `MyApp.js` file

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

Note you may already have something like this

```
import React from 'react'
```

in your file, in this case update the line like this:

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

It says: 'import whatever is exported by default from the package `react` and put it on a variable called React. And also import the useState function from the same package'.

- Add the following code to your `MyApp` component (leave the return of the function as it is)

```
const [name, setName] = useState('world')
```

We will see it in details later, for now you can just remember it creates a variables called `name` (with a default value to 'world') and a function called `setName` which will allow us to modify `name`.

- Listen to the event click of your button and add the following handler:

```
() => setName('Riri')
```

- Modify the property `name` you pass to `MyHello`: instead of passing the string `world`, pass it the variable `name` you just created (the syntax is similar to how you listen to an event).
- Click on your button, what do you see ?

If you reached this point, by default your app should still displays `Hello world`. But when you click on the `Riri` button, you app displays `Hello Riri`. Do you understand why ? React actually detects that the props `name` you pass to your component `MyHello` changes when you click on your `Riri` button. Then, automatically it updates the display of your `MyHello` component to make sure what it displays is the latest version of your date. This process is called **re-render**.