INTRODUCTION TO REACT 1.

<https://fullstackopen.com/en/part1/introduction_to_react>

<https://github.com/hiadore/fullstackopen/commits>

<https://www.youtube.com/watch?v=j942wKiXFu8&list=PL4cUxeGkcC9gZD-Tvwfod2gaISzfRiP9d&index=1>

[iamshaunjp](https://github.com/iamshaunjp)/[**Complete-React-Tutorial**](https://github.com/iamshaunjp/Complete-React-Tutorial)

<https://fullstackopen.com/en/>

<https://github.com/lirdai/-Fullstack-Part1-Exercise/blob/master/Fullstack_Part1_Exercise.txt>

[lirdai](https://github.com/lirdai)/[**-Fullstack-Part1-Exercise**](https://github.com/lirdai/-Fullstack-Part1-Exercise)Public

<https://nodejs.org/en/download/>

PS C:\Users\Seppo\Downloads\Metropolia\React> npx create-react-app part1

Need to install the following packages:

create-react-app@5.0.1

Ok to proceed? (y) y

npm WARN deprecated tar@2.2.2: This version of tar is no longer supported, and will not receive security updates. Please upgrade asap.

Creating a new React app in C:\Users\Seppo\Downloads\Metropolia\React\part1.

Installing packages. This might take a couple of minutes.

Installing react, react-dom, and react-scripts with cra-template...

added 1394 packages in 3m

210 packages are looking for funding

run `npm fund` for details

Initialized a git repository.

Installing template dependencies using npm...

added 56 packages in 10s

210 packages are looking for funding

run `npm fund` for details

Removing template package using npm...

removed 1 package, and audited 1450 packages in 4s

210 packages are looking for funding

run `npm fund` for details

6 high severity vulnerabilities

To address all issues (including breaking changes), run:

npm audit fix --force

Run `npm audit` for details.

Created git commit.

Success! Created part1 at C:\Users\Seppo\Downloads\Metropolia\React\part1

Inside that directory, you can run several commands:

npm start

Starts the development server.

npm run build

Bundles the app into static files for production.

npm test

Starts the test runner.

npm run eject

and scripts into the app directory. If you do this, you can’t go back!

cd part1

npm start

Happy hacking!

npm notice

npm notice New minor version of npm available! 8.15.0 -> 8.19.2

npm notice Changelog: https://github.com/npm/cli/releases/tag/v8.19.2

npm notice Run npm install -g npm@8.19.2 to update!

npm notice

PS C:\Users\Seppo\Downloads\Metropolia\React> npm install -g npm@8.19.2

added 1 package, and audited 212 packages in 7s

11 packages are looking for funding

run `npm fund` for details

found 0 vulnerabilities

PS C:\Users\Seppo\Downloads\Metropolia\React> npm fund

React

**Introduction to React**

We will now start getting familiar with probably the most important topic of this course, namely the [React](https://reactjs.org/)-library. Let's start off by making a simple React application as well as getting to know the core concepts of React.

The easiest way to get started by far is by using a tool called [create-react-app](https://github.com/facebook/create-react-app). It is possible (but not necessary) to install *create-react-app* on your machine if the *npm* tool that was installed along with Node has a version number of at least *5.3*.

Let's create an application called *part1* and navigate to its directory.

npx create-react-app part1

cd part1

cd C:\Users\Seppo\Downloads\Metropolia\React\part1

The application is run as follows

npm start

By default, the application runs on localhost port 3000 with the address [http://localhost:3000](http://localhost:3000/)

Your default browser should launch automatically. Open the browser console **immediately**. Also open a text editor so that you can view the code as well as the webpage at the same time on the screen:

Compiled successfully!

You can now view part1 in the browser.

Local: http://localhost:3000

On Your Network: http://192.168.8.104:3000

Note that the development build is not optimized.

To create a production build, use npm run build.

webpack compiled successfully

Compiled successfully!

You can now view part1 in the browser.

Local: http://localhost:3000

On Your Network: http://192.168.8.104:3000

Note that the development build is not optimized.

To create a production build, use npm run build.

webpack compiled successfully

app.js

const App = () => {

return (

<div>

<p>Hello world</p>

</div>

)

}

The function defining the component may contain any kind of JavaScript code. Modify your component to be as follows and observe what happens in the console:

const App = () => {

console.log('Hello from component')

return (

<div>

<p>Hello world</p>

</div>

)

}

Download the React DevTools for a better development experience: https://reactjs.org/link/react-devtools

App.js:2 Hello from component

It is also possible to render dynamic content inside of a component.

Modify the component as follows:

const App = () => {

const now = new Date()

const a = 10

const b = 20

return (

<div>

<p>Hello world, it is {now.toString()}</p>

<p>

{a} plus {b} is {a + b}

</p>

</div>

)

}

Hello world, it is Tue Sep 27 2022 22:58:18 GMT+0300 (Itä-Euroopan kesäaika)

10 plus 20 is 30

**JSX**

It seems like React components are returning HTML markup. However, this is not the case. The layout of React components is mostly written using [JSX](https://reactjs.org/docs/introducing-jsx.html). Although JSX looks like HTML, we are actually dealing with a way to write JavaScript. Under the hood, JSX returned by React components is compiled into JavaScript.

After compiling, our application looks like this:

const App = () => {

const now = new Date()

const a = 10

const b = 20

return React.createElement(

'div',

null,

React.createElement(

'p', null, 'Hello world, it is ', now.toString()

),

React.createElement(

'p', null, a, ' plus ', b, ' is ', a + b

)

)

}

The compilation is handled by [Babel](https://babeljs.io/repl/). Projects created with create-react-app are configured to compile automatically. We will learn more about this topic in [part 7](https://fullstackopen.com/en/part7) of this course.

It is also possible to write React as "pure JavaScript" without using JSX. Although, nobody with a sound mind would actually do so.

In practice, JSX is much like HTML with the distinction that with JSX you can easily embed dynamic content by writing appropriate JavaScript within curly braces. The idea of JSX is quite similar to many templating languages, such as Thymeleaf used along with Java Spring, which are used on servers.

JSX is "[XML](https://developer.mozilla.org/en-US/docs/Web/XML/XML_introduction)-like", which means that every tag needs to be closed. For example, a newline is an empty element, which in HTML can be written as follows:

<br>

but when writing JSX, the tag needs to be closed:

<br />

**Multiple components**

Let's modify the file *App.js* as follows (NB: export at the bottom is left out in these *examples*, now and in the future. It is still needed for the code to work):

const Hello = () => {

return (

<div>

<p>Hello world</p>

</div>

)

}

const App = () => {

return (

<div>

<h1>Greetings</h1>

<Hello />

</div>

)

}

**Greetings**

Hello world

We have defined a new component *Hello* and used it inside the component *App*. Naturally, a component can be used multiple times:

const App = () => {

return (

<div>

<h1>Greetings</h1>

<Hello />

<Hello /> <Hello /> </div>

)

}

Writing components with React is easy, and by combining components, even a more complex application can be kept fairly maintainable. Indeed, a core philosophy of React is composing applications from many specialised reusable components.

Another strong convention is the idea of a *root component* called *App* at the top of the component tree of the application. Nevertheless, as we will learn in [part 6](https://fullstackopen.com/en/part6), there are situations where the component *App* is not exactly the root, but is wrapped within an appropriate utility component.

**props: passing data to components**

It is possible to pass data to components using so-called [props](https://reactjs.org/docs/components-and-props.html).

Let's modify the component *Hello* as follows:

const Hello = (props) => { return (

<div>

<p>Hello {props.name}</p> </div>

)

}

Now the function defining the component has a parameter *props*. As an argument, the parameter receives an object, which has fields corresponding to all the "props" the user of the component defines.

The props are defined as follows:

const App = () => {

return (

<div>

<h1>Greetings</h1>

<Hello name='George' /> <Hello name='Daisy' /> </div>

)

}

There can be an arbitrary number of props and their values can be "hard-coded" strings or the results of JavaScript expressions. If the value of the prop is achieved using JavaScript it must be wrapped with curly braces.

app.js

const Hello = (props) => { return (

<div>

<p>Hello {props.name}</p> </div>

)

}

const App = () => {

return (

<div>

<h1>Greetings</h1>

<Hello name='George' /> <Hello name='Daisy' /> </div>

)

}

export default App

# Greetings

Hello George

Hello Daisy

Let's modify the code so that the component *Hello* uses two props:

const Hello = (props) => {

return (

<div>

<p>

Hello {props.name}, you are {props.age} years old </p>

</div>

)

}

const App = () => {

const name = 'Peter' const age = 10

return (

<div>

<h1>Greetings</h1>

<Hello name='Maya' age={26 + 10} /> <Hello name={name} age={age} /> </div>

)

}

The props sent by the component *App* are the values of the variables, the result of the evaluation of the sum expression and a regular string.

# Greetings

Hello Maya, you are 36 years old

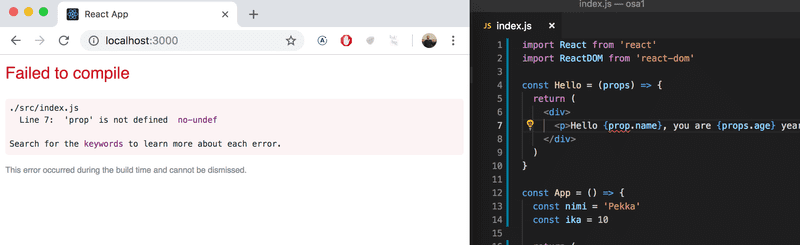
Hello Peter, you are 10 years old

The props sent by the component *App* are the values of the variables, the result of the evaluation of the sum expression and a regular string.

### Some notes

React has been configured to generate quite clear error messages. Despite this, you should, at least in the beginning, advance in **very small steps** and make sure that every change works as desired.

**The console should always be open**. If the browser reports errors, it is not advisable to continue writing more code, hoping for miracles. You should instead try to understand the cause of the error and, for example, go back to the previous working state:



It is good to remember that in React it is possible and worthwhile to write console.log() commands (which print to the console) within your code.

Also keep in mind that **React component names must be capitalized**. If you try defining a component as follows:

const footer = () => {

return (

<div>

greeting app created by <a href='https://github.com/mluukkai'>mluukkai</a>

</div>

)

}

and use it like this

const App = () => {

return (

<div>

<h1>Greetings</h1>

<Hello name='Maya' age={26 + 10} />

<footer /> </div>

)

}

the page is not going to display the content defined within the Footer component, and instead React only creates an empty [footer](https://developer.mozilla.org/en-US/docs/Web/HTML/Element/footer) element, i.e. the built-in HTML element instead of the custom React element of the same name. If you change the first letter of the component name to a capital letter, then React creates a *div*-element defined in the Footer component, which is rendered on the page.

Note that the content of a React component (usually) needs to contain **one root element**. If we, for example, try to define the component *App* without the outermost *div*-element:

const App = () => {

return (

<h1>Greetings</h1>

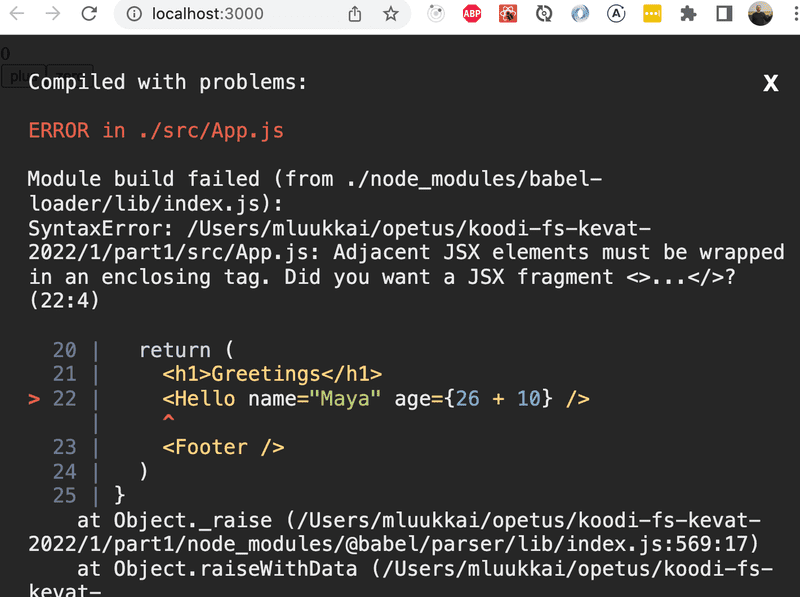
<Hello name='Maya' age={26 + 10} />

<Footer />

)

}

the result is an error message.



Using a root element is not the only working option. An *array* of components is also a valid solution:

const App = () => {

return [

<h1>Greetings</h1>,

<Hello name='Maya' age={26 + 10} />,

<Footer />

]

}

However, when defining the root component of the application this is not a particularly wise thing to do, and it makes the code look a bit ugly.

Because the root element is stipulated, we have "extra" div-elements in the DOM-tree. This can be avoided by using [fragments](https://reactjs.org/docs/fragments.html#short-syntax), i.e. by wrapping the elements to be returned by the component with an empty element:

const App = () => {

const name = 'Peter'

const age = 10

return (

<>

<h1>Greetings</h1>

<Hello name='Maya' age={26 + 10} />

<Hello name={name} age={age} />

<Footer />

</>

)

}

It now compiles successfully, and the DOM generated by React no longer contains the extra div-element.

### Exercises 1.1.-1.2.

Exercises are submitted through GitHub and by marking completed exercises in the [submission application](https://studies.cs.helsinki.fi/stats/courses/fullstackopen).

You may submit all the exercises of this course into the same repository, or use multiple repositories. If you submit exercises of different parts into the same repository, please use a sensible naming scheme for the directories.

One very functional file structure for the submission repository is as follows:

part0

part1

courseinfo

unicafe

anecdotes

part2

phonebook

countries

See this [example submission repository](https://github.com/fullstack-hy2020/example-submission-repository)!

For each part of the course there is a directory, which further branches into directories containing a series of exercises, like "unicafe" for part 1.

For each web application for a series of exercises, it is recommended to submit all files relating to that application, except for the directory *node\_modules*.

The exercises are submitted **one part at a time**. When you have submitted the exercises for a part of the course you can no longer submit undone exercises for the same part.

Note that in this part, there are more exercises besides those found below. *Do not submit your work* until you have completed all of the exercises you want to submit for the part.

#### 1.1: course information, step1

*The application that we will start working on in this exercise will be further developed in a few of the following exercises. In this and other upcoming exercise sets in this course, it is enough to only submit the final state of the application. If desired, you may also create a commit for each exercise of the series, but this is entirely optional.*

Use create-react-app to initialize a new application. Modify *index.js* to match the following

import React from 'react'

import ReactDOM from 'react-dom/client'

import App from './App'

ReactDOM.createRoot(document.getElementById('root')).render(<App />)

and *App.js* to match the following

const App = () => {

const course = 'Half Stack application development'

const part1 = 'Fundamentals of React'

const exercises1 = 10

const part2 = 'Using props to pass data'

const exercises2 = 7

const part3 = 'State of a component'

const exercises3 = 14

return (

<div>

<h1>{course}</h1>

<p>

{part1} {exercises1}

</p>

<p>

{part2} {exercises2}

</p>

<p>

{part3} {exercises3}

</p>

<p>Number of exercises {exercises1 + exercises2 + exercises3}</p>

</div>

)

}

export default App

and remove extra files (App.css, App.test.js, index.css, logo.svg, setupTests.js, reportWebVitals.js)).

Unfortunately, the entire application is in the same component. Refactor the code so that it consists of three new components: *Header*, *Content*, and *Total*. All data still resides in the *App* component, which passes the necessary data to each component using *props*. *Header* takes care of rendering the name of the course, *Content* renders the parts and their number of exercises and *Total* renders the total number of exercises.

Define the new components in file *App.js*.

The *App* component's body will approximately be as follows:

const App = () => {

// const-definitions

return (

<div>

<Header course={course} />

<Content ... />

<Total ... />

</div>

)

}

**WARNING** create-react-app automatically makes the project a git repository unless the application is created within an already existing repository. Most likely you **do not want** the project to become a repository, so run the command rm -rf .git in the root of the project.

#### 1.2: course information, step2

Refactor the *Content* component so that it does not render any names of parts or their number of exercises by itself. Instead it only renders three *Part* components of which each renders the name and number of exercises of one part.

const Content = ... {

return (

<div>

<Part .../>

<Part .../>

<Part .../>

</div>

)

}

Our application passes on information in quite a primitive way at the moment, since it is based on individual variables. This situation will improve soon.

1.1 exercise

const App = () => {

  const course = 'Half Stack application development'

  const part1 = 'Fundamentals of React'

  const exercises1 = 10

  const part2 = 'Using props to pass data'

  const exercises2 = 7

  const part3 = 'State of a component'

  const exercises3 = 14

  return (

    <div>

    <Header course = {course} />

    <Content part1 = {part1} part2 = {part2} part3 = {part3} exercises1 = {exercises1} exercises2 = {exercises2} exercises3 = {exercises3} />

    <Total exercises1 = {exercises1} exercises2 = {exercises2} exercises3 = {exercises3} />

    </div>

  )

};

const Header = (props) => {

  return (

    <div>

      <h1>{props.course}</h1>

    </div>

  )

};

const Content = (props) => {

  return (

    <div>

      <p>

        {props.part1}: {props.exercises1}

      </p>

      <p>

        {props.part2}: {props.exercises2}

      </p>

      <p>

        {props.part3}: {props.exercises3}

      </p>

    </div>

  )

};

const Total = (props) => {

  return (

    <div>

      <p>Number of exercises: {props.exercises1 + props.exercises2 + props.exercises3}</p>

    </div>

  )

};

export default App

1.2 exercise

const App = () => {

    const course = 'Half Stack application development'

    return (

    <div>

      <Header course = {course} />

      <Content />

      <Total />

    </div>

    )

};

const Header = (props) => {

  return (

    <div>

      <h1>{props.course}</h1>

    </div>

  )

};

const Total = () => {

  var num = 0

  for (var i = 0; i < PartLists.length; i++) {

    num = num + PartLists[i].exercises

  }

  return (

    <div>

      <p>Number of exercises: {num}</p>

    </div>

  )

};

const Content = () => {

  const lists = PartLists.map(

    function(list) {

      return (

        <div key={list.id}>

          <Part part={list.part} exercises={list.exercises} />

        </div>

      )

    })

  return lists

};

const Part = (props) => {

  return (

    <div>

      <p>

        {props.part}: {props.exercises}

      </p>

    </div>

  )

};

const PartLists = [

  {

    id: 1,

    part: 'Fundamentals of React',

    exercises: 10

  },

  {

    id: 2,

    part: 'Using props to pass data',

    exercises: 7

  },

  {

    id: 3,

    part: 'State of a component',

    exercises: 14

  }

];

export default App