**Day 2**

Create a new project with *create-react-app*, as the start project for the following exercises

* Remove the icon from the project (logo.svg) and all references to the icon (in App.js)
* Remove the style from the header in App.js, and use this code as the start code for the following exercises.

**Functional versus Class Components**

Read/skim [this article](https://reactjs.org/docs/components-and-props.html) (the section Functional and Class Components)

Whenever you don't use state or life cycle methods (we will come back to both later) you can write your Components as either a Functional or a Class Component. Actually functional components are always preferred when possible.

a) Rewrite the App.js into a functional component, instead of the Class Component generated by create-react-app.

*Note the exercise below demonstrates a simple way to have "many" small code examples in a single create-react-app generated project. Use this strategy for all the "smaller" exercises in this set*

**Composing Components**

**a**) In the src folder, add a new file App2.js, and add an import for React in the first line (as in App.js)

**b**) Copy the Welcome and the App component (and only those) from [this example](https://reactjs.org/docs/components-and-props.html#composing-components) into this file:

*Make sure you understand why the parenthesis is needed after the return statement in the App component, and not in the Welcome Component.*

**c**) Open index.js and comment out this line, and the corresponding import statement for App.

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

Add this line instead, and import App2:

ReactDOM.render(<App2 />, document.getElementById('root'));

Verify that you can see the three Welcome Components rendered.

*Obviously there are ways to navigate between pages (views) in react. For now however, use this simple strategy to hold your different examples, in a single project.*

**d)** Answer the following questions (if you skip this part, you have wasted your time)

* What is a functional component?
* What is a Class Component
* What is the idea with props
* Provide a simple example in how you write a Component that accepts props
* Provide a simple example (could be a line from the exercise above) that demonstrates how you pass props into a Component

**Adding Local State and Lifecycle Methods to a class**

**a**) Add a new file Clock.js to the *src* folder and add the code from this [example](https://reactjs.org/docs/state-and-lifecycle.html#adding-lifecycle-methods-to-a-class) (only the Clock class, remember to export it, and remember to include React)

**b**) Test the code, using the simple strategy given in the previous exercise

**c**) Right now the Example sleeps a hardcoded second between each update. Change this, so you can provide a sleep-time as a props value to the Clock Component.

**d**) We also don't like the hardcoded message "Hello World". Add the necessary changes so you can pass in any text, via props to the Clock Component  (Like: "Check our cool React driven timer ;-)"

**f**) Answer the following questions (if you skip this part, you have wasted your time)

* Would it be possible to rewrite the Clock component into a functional component (provide arguments for your answer)?
  + No, it would not
* How do you set new values for state: In the constructor, and all other places?
  + Use this.setState
* How is it possible to "tell" React that you want the UI to be updated (re-rendered)?
  + SetState
* What is the difference(s) between state and props
  + State is for change, props is constant
* How do you pass in prop values to a Component
  + As a parameter in a class or in a constructor
* What is the purpose of React Components Life Cycle Methods?
  + Performance enhancing
  + Things happends

**Lifecycle method Continued**

This exercise continues with the previous example, however before you start, you should [skim this article](https://reactjs.org/docs/react-component.html) about lifecycle methods.

The article introduces the lifecycle methods for *Mounting*, *Updating* and *Unmounting* components.

**a)** The Clock Component already includes four lifecycle methods (which?). In each of those methods add a console.log statement like: console.log("I am the componentDidMount");

**b**) (yellow) Add an override for each of the missing lifecycle methods (like componentWillMount), with a message as sketched above.

Note: while you do a and b make sure to have your browsers Developer Console open so you can correct any mistakes you might make.

**c**) Clear the log in your Developer Console, and observe the messages printed (change the sleep time to a larger value to make it easier to read)

**d**) Provide a small explanation for each of the outputs, the purpose of the override, and when you observed it to print (do this only for part a, if your are “green”)

**Events and ES6 arrow Functions**

Before you start you should [skim this article](https://reactjs.org/docs/handling-events.html) about Reacts way of handling events.

**a**) Add a new file Toogle.js into the src folder, and paste in the Toogle class from this [example](https://reactjs.org/docs/handling-events.html) (remember to: import React,  only include the class, and also export the Class )

**b**) Test the example, similar to how you did with the previous examples

**c)** In the constructor comment out this line, and explain the result:

this.handleClick = this.handleClick.bind(this);

***Info:***

*JavaScript includes the "this" reference, spelled like in Java, but there are serious differences between the two. This can be tricky in several situations, especially if you expect JavaScript-this to behave like in Java.*

*ES6 (es2015) has tried to clean up the this-behavior inside classes, where you can get a behavior (almost) similar to Java if you use ES6 arrow functions.*

**d)** Still with the line commented out as you did in c), rewrite the *handleClick* function into an arrow function as sketched below.

handleClick = () => {

   ...

}

**e)** Verify that the example works again

***Info:***

*These are the two ways you can (and must) use to deal with this for event handlers (****bind*** *and* ***arrow*** *functions). You can use whatever you prefer, but you need to know both strategies, since they are both used "out there"*

**f**) Answer the following questions (if you skip this part, you have wasted your time)

* What is the purpose of this line in the constructor: this.handleClick = this.handleClick.bind(this);
* How can we disable the default behavior of an event handler (for example prevent a submit handler to submit?)
* What is the benefit(s) you get from using arrow-functions in a ES6 class?

**Using what we have learned up until now**

**a**) Create a new project with create-react-app, as the start project for this exercise. Clean up the generated start code, similar to what we have done up until now.

**b**) Create a new JavaScript file and inside this file,  create a class that extends React.Component (Remember to import React and to export the class similar to the previous examples)

**c**) Add some content into the render() method and make the application load the component and render the content

**d**) Create an image folder in the 'public' folder and put some images inside

**e**) In your new class component add an array of references to the images

Hint: ['image/img1', 'image/img2', ...]

**f**) Let your component loop through the array and display all the images in its render() method

**g**) (yellow) Style your component to give each image a visible border and a padding of 2px.

**h**) (yellow) Change the Component so that it will display only a single random image from the collection.

**i**) (yellow) Add a button + event handling, so that clicking it will change to a new random image on display

**j**) (yellow/red) Inspired by the way you used the setInterval method in the Clock Component Exercise, change the code to automatically display a new random image after a given amount of time (say 6 seconds)

**k**) (red) Create an ImageHolder component as a functional stateless component, that can have both an image and some text related to the image

Hint: use props to send information about the image source and the text to the sub component.

**l**) (red)  Use ImageHolder component inside the image gallery component to show the image plus the text of each image in the gallery

**Was that it, I want more ;-)**

Great to hear. Continue with the egghead-video tutorial or just Google to find additional material.

Or even better, start to play around with what you have learned, and invent your own small cases. This is how you become a true React Expert :-)