React.2

▼ What does spread operator do?

it creates a new copy of an array with the same values

```
const numbers = [1, 2, 3];
const grades = [...numbers];
console.log(grades); // [1, 2, 3] (shallow copy)
```

▼ What is a shallow copy?

a copy of the items inside an array (but down to 1 level)

▼ How can you use spread syntax to concatenate arrays?

```
const winners = ["Jane", "Bob"];
const losers = ["Ronald", "Kevin"];

const players = [...winners, ...losers];
console.log(players); // ['Jane', 'Bob', 'Ronald', 'Kevin']
```

▼ How do we add an item to an array in an immutable way?

We don't use .push() because it will mutate original array.

We have to make a shallow copy and then insert the new item in a new array.

```
const numbers = [1, 2, 3];
const result = [...numbers, 4];
console.log(result); //[1, 2 ,3 ,4]
```

▼ How to update array items in an immutable way?

you can use the .map method to return a copy of the array while modifying one or more items

```
const grades = [10, 20, 18, 14];
// change 18 to 17
const updatedGrades = grades.map(grade => {
    if (grade === 18){
        return 17;
    }
    // in all other cases, keep it as it was
    return grade;
});
console.log(updatedGrades); //[10, 20, 17, 14]
```

▼ Why is slice() immutable and splice() is not?

splice() method returns the removed item(s) in an array

slice() method returns the selected element(s) in an array, as a new array object

▼ How to remove array items in an immutable way?

you can use the .slice method which returns the selected element(s) in an array, as a new array object

you can also use the .filter method which will return a subset of the original array based on a condition

```
const grades = [10, 8, 9, 4, 16];

// remove the first grade
// think of it as: get all grades except the first one
const subset1 = grades.slice(1); //start from position 1
console.log(subset1); // [8, 9, 4, 16]

// remove the last 2 grades
// think of it as: get all grades except the last 2
// so start from 0 and stop after 5 - 2 = 3 items
const subset2 = grades.slice(0, grades.length - 2);
console.log(subset2); // [10, 8, 9]
```

```
const grades = [10, 8, 9, 4, 16];

// return all grades >= 10
const subset1 = grades.filter(grade => grade >= 10);
console.log(subset1); // [10, 16]

// remove the 2nd grade
const subset2 = grades.filter(grade => grade !== 8);
console.log(subset2); // [10, 9, 4, 16]
```

▼ How can .map can be used **inside** JSX to loop through arrays?

Every time you have a map in JSX, you need to provide a key or else you will get a warning.

React needs to be able to know what item to update in a list without re-rendering the whole list for every update.

The key should be a unique representation of the single item inside the map.

```
import React from "react";

function Grades(){
    const grades = [8, 18, 10, 7, 14];

    return <ul>{
        grades.map((grade, index) => <li key={index}>{grade}</li>)
    }</ul>;
}
```

▼ Why do we need keys in React?

For example, given a list React needs to be able to know which to update thus it requires a unique key so that it is able to only update that item without having to remove all the items and render them again.

▼ What does a key allow React to do efficiently?

update the DOM with the least amount of operations

▼ How to add a key/value to object immutably?

```
const data = {
    id: 1,
    name: "Sam"
}

// immutable
const newObj = {...data, age: 18}
console.log(newObj); // {id: 1, name: "Sam", age: 18}
```

▼ How to replace the value of an existing key immutably?

we need to create a new copy of that object with { ...data } and then merge it with the new same key but a different value

It's important to note that when you want to replace, the new values should be after the copy of the old object in order to override old value.

```
const data = {
    id: 1,
    age: 19
}

// immutable
const newObj = {...data, age: 20};
console.log(newObj); // {id: 1, age: 20}
console.log(data); // original object did not change {id: 1, age: 19}
```

▼ How to to create a new copy of an existing object?

{ ...obj }

▼ What does new Date() return?

an instance of the Date object that gives us the current date & time

e.g. "Tue Feb 18 2020 16:34:15 GMT..."

▼ How to immutably remove a key/value pair from an object?

The reason why this works is because const {year, ...rest} = obj is destructuring the value of the key year from obj.

So we end up with rest an immutable copy of obj excluding the year!

```
const obj = {
    id: 1,
    title: "Harry potter",
    year: 2017,
    rating: 4.5
}

// immutable
const {year, ...rest} = obj;
console.log(rest); // { id: 1, title: "Harry potter", rating: 4.5}
```

▼ How to loop through an object in JSX?

The Object.entries() method returns an **array** of a given object's own enumerable string-keyed property **[key, value]** pairs

```
import React from "react";

function App() {
    const settings = {
        title: "Blog",
        theme: "dark"
    }

    return <ul>{
        Object.entries(settings).map(item => {
            return <li key={item[0]}>{item[0]} with value {item[1]}</li>
        })
    }</ul>;
}

/*
<ul>
    <li key="title">title with value Blog</li>
    <li key="theme">theme with value dark</li>
</ul>
*/
```

▼ How to add a default value on a input in JSX?

```
<input type="text" name="address" defaultValue="Amsterdam" />
```

▼ How to add a **read only value** on a input in JSX?

```
<input type="text" name="address" value="Amsterdam" />
```

▼ Event handler for an input JSX?

onChange attribute

event.target refers to the element (in this example the <input />)

because it's an input, you read what's written inside of it by accessing the .value property

```
import React from "react";

function handleAddressChange(event) {
    console.log(event.target.value);
}

<input type="text" name="address" onChange={handleAddressChange} />
```

▼ What is a controlled component?

when you keep track of an input's value as state and update it whenever it changes

▼ How to create a controlled component?

1. We start by creating a **state** variable to store the value
2. This state will have a default value of an empty string `""` or another default value
3. We set the value of the input to that **state** variable
4. We update the state every time it changes

```
import React, {useState} from "react";

function App() {
    const [address, setAddress] = useState("");

    return <input type="text" value={address} onChange={event => setAddress(event.target.value)} />;
}
```

▼ Controlled select component (example)

```
import React, {useState} from "react";

function App() {
    const [country, setCountry] = useState("");

    return <select value={country} onChange={e => setCountry(e.target.value)}>
        <option>Country</option>
        <option value="netherlands">Netherlands</option>
        <option value="belgium">Belgium</option>
        <option value="france">France</option>
    </select>
}
```

▼ Controlled textarea component (example)

```
import React, {useState} from "react";

function App() {
    const [comment, setComment] = useState("");

    return <textarea value={comment} onChange={e => setComment(e.target.value)} />
}
```

▼ React submit a form (example)

event.preventDefault() : called on the event when submitting the form to prevent a browser reload/refresh

```
import React from "react";

function App(){

    function handleFormSubmit(event) {
        event.preventDefault();
    }

    return <form onSubmit={handleFormSubmit}>
        <input type="text" name="name" />
        <input type="submit" value="Add" />
    </form>;
}
```

▼ Why event.preventDefault() when submit form?

it's called on the event when submitting the form to prevent a browser reload/refresh

▼ What is accessibility?

the design and creation of web applications that can be used by everyone

it's a practice that promotes inclusion because everyone (people with all abilities) will be able to use your website

▼ What is a11y?

You may often see accessibility shortened as a11y, which means it's a word that starts with a, ends with y, and has 11 characters in between (ccessibilit).

▼ Why is it important to **add a label element to every input** (except buttons), textarea and select in your form?

Accessibility

For: 1) Mouse users & users with motor impairment 2) Visually impaired users use screen-readers

▼ A <label /> needs an htmlFor attribute (React) for what?

to point to the ID of the element

```
<form>
    <label htmlFor="login-email">Email: </label>
    <input type="email" id="login-email" placeholder="alex@email.com" />

    <label htmlFor="login-password">Password: </label>
    <input type="password" id="login-password" placeholder="Password" />

    <input type="submit" />
</form>
```

▼ Shallow vs Deep copies of arrays of objects

A shallow copy means that it creates a new array, but the objects are still referring to the old ones.

A deep copy means that the new array contains new objects so that changes to those new objects will not affect the old ones in the original array.

▼ What is the solution for copies of arrays of objects in React?

Even though the spread operator creates a shallow copy, it is creating **a new copy of the array which is enough to notify React that the state has changed**.

or you could use a library like immutable-js

```
const users = [{
   id: 1,
   name: "Alex"
}];

// create a (shallow) copy
```

```
const usersCopy = [...users];

// are they the same? If `false` then React thinks
// that the state has changed
console.log(users === usersCopy); //false
```

▼ Example of iterating and deleting items by ID

```
<ul className="store-front">
    {products.map(product => <li key={product.id}>
        <Product details={product} />
        <button className="btn-outline btn-delete" onClick={() => handleDeleteButton(product.id)}>Delete</button>
    </li>)}
</ul>
```

▼ What is Create React App (or CRA)?

an officially supported way of creating single-page React Apps

▼ What is a single page app?

A single page app is a website built with a single index.html whereby JavaScript is responsible for re-writing the content based on the URL.

▼ Official documentation for Create React App

https://create-react-app.dev/

▼ Why use create-react-app?

Using create-react-app allows you to get started with building a React app without having to worry about setting up a build system (such as Webpack).

▼ What is Webpack?

Webpack is used to allow you to import libraries that you will need inside your project (among other things).

▼ Command for creating a react app with create-react-app

npx create-react-app name-of-your-app

▼ What is npx?

npx is a command that is available with NodeJS and it allows you to run and execute a certain type of node modules (specifically: package binaries)

▼ What are scripts?

Commands you can use in package.json

```
{
    "scripts": {
        "start": "react-scripts start",
        "build": "react-scripts build",
        "test": "react-scripts test",
        "eject": "react-scripts eject"
    }
}
```

▼ What is npm run start?

`npm run start` will start a webserver using webpack which you can access by going to **localhost:3000**

This command will use the **development** version of React

▼ What is a webserver (software)?

a web server uses HTTP (Hypertext Transfer Protocol) and other protocols to respond to client requests

▼ What is a web server (hardware)?

a web server is a computer that stores web server software and a website's component files

▼ What is npm run build?

When you're ready to deploy your project, you should run `npm run build` which will create an optimized version of your app.

It will minify all your scripts, and perform several optimizations before you deploy your app. The optimized files will be placed in a new folder called **build**.

This command will use the **production** version of React.

▼ What is npm run test?

run tests

you will have to write those tests as you develop your application

▼ What is npm run eject?

allows you to unhide configuration files so that you can edit them

Ejecting the configuration is permanent (means you cannot undo) and you will end up having to maintain your own Webpack configuration.

▼ Do not maintain your own Webpack configuration unless you're at a large company

This might make sense for some large companies, but if you're a solo developer or a small team, in most scenarios it's not recommended.

In case you're looking to use a certain feature that is not available out of the box, start by checking if it's included in the **documentation**.

If you still can't find it, then check if there's an existing **template** that solves the same issue.

▼ What should you put in the src/ folder?

contains the <App /> component and is where you should put all your scripts & styles

▼ What should you put in the public/ folder?

contains the index.html and is where you should put all your images, fonts (if any)

▼ What should be imported in index.js?

The App component is exported and then imported in the index.js

The index.js has the call to ReactDOM's render method:

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

▼ What should be in public/index.html?

▼ How to organize folder structure of small-sized projects (less than 20 components)?

```
.
└── src
    ├── products
    │   ├── Product.js
    │   ├── ProductDetails.js
    │   └── ProductQuantity.js
    ├── store
    │   ├── StoreFront.js
    │   └── Checkout.js
    └── user
        ├── User.js
        └── Profile.js
```

▼ How to organize folder structure of medium-sized projects (contains helpers and classes)?

```
.
└── src
    ├── components
    |   ├── products
    |   |   ├── Product.js
    |   |   ├── ProductDetails.js
    |   |   └── ProductQuantity.js
    |   └── store
    |       ├── StoreFront.js
    |       └── Checkout.js
    ├── helpers
    |   ├── date.js
    |   └── currency.js
    └── classes
        └── backend.js
```

▼ How to organize folder structure of large-sized projects?

Large projects will most likely have specific requirements, thus it will be up to them to decide on a folder structure.

▼ What is the starting point of a React application?

The <App /> component in src/App.js

▼ React development vs production

Development

- is not minified. It contains all the comments from the React source code and also includes development helpers such as warnings.
- react.development.js (106KB)

Production

- is minified, strips out all the comments, and does not include development helpers and warnings.
- should be used whenever you deploy your website.
- react.production.js (16KB)

▼ What is a stateless component?

it does NOT manage state internally

no useState calls.

▼ What is a stateful component?

it will manage at least 1 state variable

▼ How can a stateless component be interactive?

For example, a stateless component can contain a form as well as a textbox and a submit button. However, the state will be managed by its parent component.

▼ How to pass a function as a property to a component?

```
import React from "react";

function App() {     // App is parent component of StoreFront component

    function handleWelcome() {
        console.log("Hello World");
    }

    return <StoreFront onWelcome={handleWelcome} />;
}

function StoreFront(props) {

    if (props.onWelcome) {
        props.onWelcome();
    }
```

```
    return <div>Store renders here</div>;
}
```

▼ Naming convention for props that are functions

onSubjectEvent

onStoreOpen={handleStoreOpen}

▼ Naming conventions for functions

handleSubjectEvent

onStoreOpen={handleStoreOpen}

▼ (Example) pass a function as a property to a component - onClick

```
// index.js
import React from "react";
import {render} from "react-dom";
import Card from "./Card.js";

function App() {

    function handleCardClick() {
        console.log("Card got clicked");
    }

    return <Card onCardClick={handleCardClick} />
}

render(<App />, document.querySelector("#react-root"));
```

```
// Card.js

import React from "react";

export default function Card(props) {

    return (<div className="card">
        <button onClick={props.onCardClick}>Click me</button>
    </div>);
}
```

▼ (Example) pass a function as a property to a component - onChange

The App component is a stateful component because it manages the state.

NameForm is a stateless component as it does NOT manage state.

```
//index.js
import React from "react";

function App() {  // stateful
    const [name, setName] = useState("");

    function handleNameChange(event) {
        setName(event.target.value);
    }

    return <div>
        <h2>Hello {name}</h2>
        <NameForm name={name} onNameChange={handleNameChange} />
    </div>
}
```

```
//NameForm.js
import React from "react";
```

```
export default function NameForm(props) {  // stateless

    return <form>
        <label htmlFor="name">Name: </label>
        <input type="text" id="name" value={props.name} onChange={props.onNameChange} />
    </form>
}
```

▼ In what direction can you share state between components?

you can lift state up to their shared closest common ancestor

▼ If you have 2 components that depend on the same state, then the state will be defined where?

in their closest parent component

▼ (Example) shared state between components

```
// common parent/ancestor component. stateful

//TodoApp.js
import React, {useState} from "react";
import TodoForm from "./TodoForm.js";
import TodoList from "./TodoList.js";

function TodoApp() {
    const [todos, setTodos] = useState([]);
    const [entry, setEntry] = useState("");

    function handleEntryChange(event) {
        setEntry(event.target.value);
    }

    function handleFormSubmit(event) {
        event.preventDefault();
        setTodos([...todos, entry]);
        setEntry("");
    }

    return <>
        <TodoForm entry={entry} onEntryChange={handleEntryChange} onFormSubmit={handleFormSubmit} />
        <TodoList todos={todos} />
    </>;
}
```

```
// stateless

//TodoForm.js
import React from "react";

function TodoForm(props) {
    return <form onSubmit={props.onFormSubmit}>
        <label htmlFor="todo">Enter To do: </label>
        <input type="text" id="todo" value={props.entry} onChange={props.onEntryChange} />
    </form>;
}
```

```
// stateless

//TodoList.js
import React from "react";

function TodoList(props) {
    return <ul>
        {props.todos.map((todo, index) => <li key={index}>{todo}</li>)}
    </ul>;
}
```

▼ Reasons for breaking up a component into smaller ones (aka abstraction)?

- Reusability

- Isolate bugs

▼ (Example) stateful and stateless components - with state shared

```
//StoreFront.js
import React, {useState} from "react";
import ProductsList from "./ProductsList.js";
import AddProductForm from "./AddProductForm.js";

export default function StoreFront() {
    const [products, setProducts] = useState([]);
    const [name, setName] = useState("");
    const [description, setDescription] = useState("");
    const [validation, setValidation] = useState("");

    function handleFormSubmit(event) {
        event.preventDefault();

        if (!name) {
            setValidation("Please enter a name");
            return ;
        }
        if (!description){
            setValidation("Please enter a description");
            return ;
        }
        setProducts([...products, {
            id: products.length + 1,
            name: name,
            description, description
        }]);
        setName("");
        setDescription("");
        setValidation("");
    }

    function handleNameChange(event) {
        setName(event.target.value);
    }

    function handleDescriptionChange(event) {
        setDescription(event.target.value);
    }

    function handleDeleteClick(id) {
        setProducts(products.filter(product => product.id !== id));
    }

    return <>
        <AddProductForm name={name} description={description} validation={validation} onNameChange={handleNameChange} onDescriptionChan
        <div>{products.length === 0 && <p>Add your first product</p>}</div>
        <ProductsList products={products} onDeleteClick={handleDeleteClick} />
    </>;
}
```

```
//ProductsList.js
import React from "react";
import Product from "./Product";

export default function ProductsList(props) {

    return <ul className="store-front">
            {props.products.map(product => <li key={product.id}>
                <Product details={product} />
                <button className="btn-outline btn-delete" onClick={() => props.onDeleteClick(product.id)}>Delete</button>
            </li>)}
        </ul>;
}
```

```
//AddProductForm.js
import React from "react";

export default function AddProductForm(props) {
    return <form onSubmit={props.onFormSubmit}>
```

```
        <div>
            <label htmlFor="product-name">Name:</label>
            <input type="text" value={props.name} onChange={props.onNameChange} id="product-name" placeholder="Enter the name" classNam
            </div>
            <div>
            <label htmlFor="product-description">Description:</label>
            <input type="text" value={props.description} onChange={props.onDescriptionChange} id="product-description" placeholder="Ent
            </div>
            <div className="form-footer">
            <div className="validation-message">{props.validation}</div>
            <input type="submit" className="btn btn-primary" value="Add product" />
            </div>
        </form>;
}
```

▼ React Dev Tools tutorial

https://react-devtools-tutorial.vercel.app/

▼ Why use React Dev Tools Profiler?

The profiler tab in React dev tools will help us **spot rendering performance issues** in our React apps.

- This tab is useful when there are a lot of components in your app, as it lets you see which components re-rendered and why.
- Often times when improving the rendering performance, you will realize that there was a component that did not need to re-render.

▼

▼

▼

▼

▼

▼

▼

▼

▼

▼

▼

▼

▼

▼

▼