React.1

▼ What is React?

React is a JavaScript library created and (mostly) maintained by Facebook.

React is a JavaScript library for building User Interfaces.

▼ Define React

React is a JavaScript library that is only responsible for the view layer of an application.

This means it is only responsible for rendering a User Interface and updating the UI whenever it changes.

▼ Is React a framework?

No, it's a library

▼ What is the difference between a library and a framework?

The difference between a library and a framework is that a library only helps you in one aspect whereas a framework helps you in many aspects.

- React is a library because it only takes care of your UI.
- Angular, on the other hand, is a framework because it handles much more than the UI (It handles Dependency Injection, CSS encapsulation, etc.)

▼ Every JavaScript file is a...

standalone module, meaning variables, functions, and imports in one file/module do not affect other files/modules

▼ What is a bare import?

an import without a file path

e.g. import React from "react";

▼ What does a code bundler do?

When running a code bundler such as webpack or Parcel (or create-react-app which uses Webpack), then bare imports are resolved from your node_modules folder.

It will look for

```
node_modules/react/
```

to figure out how to import that package.

▼ What is the current React version?

17.0.2

▼ Do you need to import React for version 17?

No

▼ What is the current JavaScript or ECMAScript version?

ES11

JavaScript 2020

▼ What is the size of a React import?

approximately 12.5KB.

▼ What is document.createElement(tagName)?

a Web API provided by browsers that let you programmatically create an HTML element

```
const element = document.createElement("h2");

// this creates: <h2></h2>
// which you can then insert in the DOM:
document.body.appendChild(element);
```

▼ What does console.dir(element) show?

a method provided by browsers to list all the properties of a certain object

```
console.dir(lesson) // given element with variable lesson

div#lesson
accessKey: ""
align: ""
ariaAtomic: null
ariaAutoComplete: null
ariaBusy: null
ariaChecked: null
...
```

▼ How can you change the style of an element with element.style?

```
element.style = "color: red; background-color: blue";
```

▼ How can you change the class of an element with element.className?

```
element.className = "name-of-class";

element.className = "container center"; // multiple classes
```

▼ Why don't use element.classList.add("name-of-class") with React?

in React you will need to avoid changing the DOM directly (for example with classList.add) and instead **define the elements that you'd like to render**

▼ In React you avoid changing the DOM directly and instead define what?

the elements that you'd like to render

▼ document.createElement vs. React.createElement

document.createElement returns a DOM element (for example a div or an h1)

React.createElement returns **an object that represents the DOM element**

▼ How to React.createElement(type, options, children)?

```
const element = React.createElement("h1");
//returns an object similar to this one:
{
  type: 'h1',
  props: {}
}
```

▼ Why does React.createElement return an object rather than the DOM element?

because React operates a Virtual DOM

▼ What is a virtual DOM?

a representation of the UI is kept in memory and synced with the DOM

this is a performance optimization

▼ How to change the class/style of React.createElement?

```
React.createElement("h1", {className: "center", style: "color: red"})
```

▼ How to write text for a React.createElement?

Notice how we're saying that we don't want to set a className or style (or other options) so we pass {} as the second parameter (you could also pass null).

```
React.createElement("h1", {}, "Hello World")
```

▼ What is the 3rd parameter for React.createElement?

children (so it also accepts other Elements for later on)

▼ What is a React element?

an Element is a representation of what the smallest piece of a User Interface will look like

▼ What is the glue between React and the DOM?

ReactDOM

▼ What is ReactDOM?

the library that receives instructions from React and efficiently updates the DOM based on that Virtual DOM

▼ Why does the virtual DOM exist?

to figure out which parts of the UI need to be updated and then batch these changes together

▼ ReactDOM binds the idea of React to what?

a web browser (example: Firefox, Chrome, Safari, Edge, etc.)

▼ What is React Native?

the glue between React and native apps

▼ What is reconciliation?

React is creating the virtual representation of the UI in memory, and then ReactDOM receives that and syncs the UI to the actual DOM.

▼ How to install React and ReactDOM?

npm install react react-dom

▼ How to import React?

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

▼ How to import ReactDOM?

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

import {render} from "react-dom"; // render method
```

▼ Where should elements be rendered?

within a root element aka the container

```
<div id="root"></div>

import {render} from "react-dom";

const root = document.querySelector("#root");
render(React.createElement("p", {}, "Hello World"), root);
```

The render(element, root) method expects the element to render as a first argument and the root as the second one.

▼ Apps built with React usually have how many root elements?

One. The whole application is rendered inside that root element.

▼ The root element is completely managed by what?

ReactDOM

▼ What does JSX stand for?

JavaScript XML (eXtensible Markup Language)

▼ The JSX syntax looks similar to HTML but it is not what?

HTML

▼ JSX syntax

```
import React from "react";

// this code will be translated
const title = <h1>Hello World</h1>

// into this
const title = React.createElement("h1", {}, "Hello World");
```

▼ the JSX that you write is being converted to what?

React.createElement

▼ JSX requires what?

React. It creates JSX

you need to import React in every file that uses JSX or else it won't work

▼ You need to convert JSX with babel to what?

JavaScript

▼ JSX with attributes. Note className not class

```
const title = <h1 id="brand-title" className="primary-color">Supermarket</h1>;
```

▼ KEY: Since JSX is being converted into a React.createElement(...) that returns an object, you can essentially think of a JSX element as if it was what?

an object

As a reminder, this object is a representation of a piece of your UI that React maintains in its Virtual DOM. This lets it efficiently know when it updates and re-render it using ReactDOM.

```
// given:
<h1 className="title">Supermarket</h1>

// we can think of it as an object like this:
{
  type: 'h1',
  props: {
    className: "title",
    children: "Supermarket"
  }
}
```

▼ What is an expression?

any valid JavaScript code that resolves to a value

▼ How to use expression in JSX?

wrap in curly brackets {}

you can have variables and function calls inside expressions

```
const title = <h1>You have {2 + 3} notifications</h1>;
```

▼ How to do attribute values in JSX?

strings OR expressions

```
<input tabIndex={2} disabled={true} className="textbox" />
```

▼ How to do dynamic attribute values in JSX?

use {}

```
<li id={`item-${count}`}></li>

// multiple classes
<button className={`${clickable} ${active}`}>Click me</button>;
```

▼ Why do you need to wrap JSX elements in a return(); ?

Because if you don't use the (), then JavaScript will automatically add a ; immediately after the return keyword before the JSX elements

```
const getList = () => {
    return (
        <ul>
            <li>First Item</li>
            <li>Second Item</li>
        </ul>
    );
}
```

▼ What are self-closing tags?

HTML tags that cannot contain children thus they do not need a closing tag

▼ What is <br /> or <br> ?

line break

▼ What is <hr /> or <hr> ?

horizontal ruler aka thin grey line

▼ Why can you only return 1 JSX element at a time?

every Element is an object and you cannot return 2 or more objects next to each other

▼ What is React.Fragment?

it allows you to return multiple JSX elements within a fragment

The fragment is an internal representation in React that lets you wrap multiple elements and will not be rendered (unlike a div).

Notice how the opening fragment <> and the closing fragment </> wrap the elements.

```
function getHeroBanner() {
    return (
        <>
            <h1>Grocery delivered to your door</h1>
            <h2>Free delivery</h2>
            <p>Get started now!</p>
        </>
    );
}
```

▼ Fragment vs. Div?

You can choose to return a <div> however the <div> will end up being rendered in your final HTML whereas a fragment will disappear.

▼ What is a React Component?

A function that returns a React Element which describes how a section of the User Interface should look like.

It is a template or a blueprint that lets you describe a section of the UI, e.g. footer.

▼ Why a Component?

The Component allows you to split your UI into independent, reusable pieces.

This lets you think about each piece of your UI in isolation which makes it easier to debug.

▼ React function Component syntax

We call this a function component because the Component is defined as a function.

The first character has to be in upper case for it to be considered a Component e.g. UpperCamelCase for the component name.

```
import React from "react";

function Footer() {
    return (
        <div>
            <h3>Company name</h3>
            <p>All rights reserved</p>
        </div>
    );
}

...
render(<Footer></Footer>, document.querySelector("#root"));
```

▼ How can React tell the difference between components and DOM tags?

Components are uppercase e.g. Footer

DOM tags such as the HTML footer element are lowercase.

▼ How to accidentally create an infinite loop when creating a React component?

```
import React from "react";

function Button() {
    // Be careful: this is an infinite loop
    return <Button></Button>;
}
```

That's because the `<Button></Button>` JSX will be converted to `React.createElement(Button)` calls which will eventually call the `Button` function. So you'd end up with a function Button that keeps calling itself over and over.

In this case, we probably meant to return `<button></button>` (lower case) which is the HTML Element `button` rather than the component `Button`.

▼ What is the best practice for the number of components per file?

1 component per file

- file: `Footer.js` for the Component `Footer`

▼ What file is the the main entry point of an app?

index.js or app.js

▼ Example Component file

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

export default function Footer() {
    return (
        <>
             <h3>Footer</h3>
             <p>All rights reserved</p>
        </>
    );
}
```

▼ Why always name export functions?

for easier debugging

▼ index.js and App Component example

```
//index.js

import React from "react";
import Button from "./Button.js";
import Link from "./Link.js";
import {render} from "react-dom";

function App() {
    return (
        <>
            <Button />
            <Button />
            <Link />
        </>
    );
}


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

▼ How to pass props or properties to a Component?

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

export default function GreetUser(props){
    return <div>Welcome {props.user}</div>;
}

// USAGE
<GreetUser user="Sam"> would render <div>Welcome Sam</div>
```

▼ What is props?

Props is an object received as the first argument of a Component

▼ What are attributes on a Component converted to?

Attributes on Components get converted into an object called Props

▼ What is props.children?

The children props represents the content within a given component.

can contain text and/or React Elements and/or React Components

note: props.children destructures last

```
import React from "react";

function HeroTitle(props) {
    return <h1 className="hero">{props.children}</h1>;
}

// USAGE
const element = <HeroTitle>Welcome!</HeroTitle> // Welcome! string is the children props
```

```
import React from "react";

function Navbar(props){
    return <div className="navbar">{props.children}</div>; // accepts React elem + comp
}

const element = <Navbar>
    <HeroTitle>Welcome!</HeroTitle>
    <div>Some content</div>
    <p>Another content</p>
</Navbar>;
```

▼ Destructuring example

```
const person = {
    first_name: "Jennifer",
    last_name: "Doe",
    age: 24
}

// You could also provide a default value for a variable, in case it
// did not exist in the object. Example:
const {first_name, last_name, status = 'single'} = person;
```

▼ Destructuring props example (common)

Note: Attributes on Components get converted into an object called Props

```
import React from "react";
import {render} from "react-dom";

function Button(props){
    const {className, children} = props; // note: props.children destructures last
    return <button className={className}>{children}</button>;
}

const root = document.querySelector("#react-root");

render(<Button className="primary">Login</Button>, root);
```

▼ Destructure props in the component function argument (common)

```
import React from "react";
```

```
function WelcomeUser({username, notifications}) {
    return <div>Welcome {username}! You've got {notifications} unread notifications.</div>;
}
```

▼ What is a UI kit?

A UI Kit is a collection of all the design elements that you will use in your app, for example:

- Buttons
- Text boxes
- Cards

▼ All React components should never modify what?

their props

```
import React from "react";

function Notifications(props) {
    //do NOT do this (this is changing the props)
    props.data.count = props.data.count - 1;
    return <h3>You have {props.data.count} unread notifications.</h3>;
}

// INSTEAD do this
function Notifications(props) {
    //this is okay because we're not changing the props
    const value = props.data.count - 1;
    return <h3>You have {value} unread notifications.</h3>;
}
```

▼ You should think of props as

read-only

Why? This allows React to quickly identify which Components need to re-render when a certain prop updates. This happens as part of the Virtual DOM.

▼ What is a pure function?

given the same input (props), the function will always have the same result (output)

▼ React expects Components to be pure so that what?

it can efficiently update the DOM only when/where necessary

▼ How to write a comment in JSX?

{/*this is a comment - the comment won't render*/}

```
import React from "react";

function Navbar() {
    return <div>{/*this is a comment - the comment won't render*/}hi</div>;
}
```

▼ In order to tell React that you would like to return nothing, you have to specifically return what?

null from the Component

```
import React from "react";

function Navbar() {
    //this is a valid Component
    return null;
}
```

▼ Conditional rendering based on props

```
import React from "react";

function Navbar(props) {
    if (!props.loggedIn) {
        return <p>Register</p>
    }
    return <p>Welcome back!</p>
}

const element1 = <Navbar loggedIn={true} />
const element2 = <Navbar loggedIn={false} />
```

▼ What is clsx library?

clsx is a tiny utility for constructing className strings conditionally

clsx({"your-class-name": booleanValue}) is the generic syntax for clsx

npm install clsx

```
// WITHOUT clsx

import React from "react";

function MyComponent(props) {
    let className = ""
    if (props.loggedIn) {
        className = "title"
    }
    return <h1 className={className}></h1>
}
```

```
// WITH clsx

import React from "react";
import clsx from "clsx";

function MyComponent(props) {
    const className = clsx({"title": props.loggedIn});
    return <h1 className={className}></h1>
}

const element = <MyComponent loggedIn={true} />;
```

```
// other ways of clsx

import clsx from "clsx";

const classes = clsx("class1", "class2");
console.log(classes); //"class1 class2"

const dynamicClass = "class1";
const classes = clsx(dynamicClass, "class2");
console.log(classes); //"class1 class2"
```

▼ JavaScript spread operator

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

const details = {
    age: 23,
    loggedIn: true
};

const person_details = {...person, ...details};
/*
{
    id: 1,
    name: "Sam",
    age: 23,
    loggedIn: true
}
*/
```

▼ JavaScript spread operator and destructing

Say you want to create a new object that contains all the keys and values except the id, here's how you can do it with the spread operator and destructuring:

```
const {id, ...rest} = person;
console.log(id); //1
console.log(rest); //{name: "Sam", age: 23}
```

▼ This is a way to allow a Component to be customizable e.g. UI kit, using spread operator:

Benefit: easy and imagine we add a new attribute, for example: disabled={true}, then your code still works and will still apply the disabled attribute on the h1 without having to update our Component.

The most common use case of JSX Spread Attributes is when you're building a UI Kit and want to make it possible to customize the rendered Element.

```
import React from "react";

function Navbar(props) {
    const {children, ...rest} = props;
    return <h1 {...rest}>{children}</h1>;
}

<h1 tabIndex="2" className="navbar">Hello World</h1> // Hello World is children props
```

▼ What is array destructuring?

Array destructuring allows you to have a shorter syntax when reading multiple entries of an array.

```
const dimensions = [20, 5]

const [width, height] = dimensions;


const data = ["Sam", 23, {
    id: 1,
    created_at: "Jan 19"
}];

const [name, age, details] = data;
```

▼ How to destructure from a function?

```
function getUser() {
    //return the id and a function that welcomes the user
    return [15, sayHello];
}

// you can pass a function definition in the destructuring
const [id, sayHelloFunction] = getUser();
console.log(id); //15
```

▼ What is state?

State refers to any variable defined inside a Component with the intent to update later on.

▼ What happens when state changes?

When the state is updated, ReactDOM will efficiently and automatically re-render the Component to the DOM.

▼ What should be state and what should be props? (example)

Which is why **theme** is a prop. We don't plan on updating it from inside the **Stopwatch**.

Whereas the **seconds** elapsed, is a state because we plan on starting it/stopping it from inside the `<Stopwatch />` Component.

▼ What do you need to import to create a state variable?

```
import React, {useState} from "react"; // React is a default export so no {}
```

▼ Why is useState a React Hook?

This method will hook into the internals of React and notify it that we are changing the value of a state.

▼ What does useState(initial_value) return?

returns an array of 2 items:

1. the first one is the **current value of the state**
2. the second one is a function that updates the state

```
const [seconds, setSeconds] = useState(0); // use the array destructuring syntax
```

▼ Example of Component state setup

The useState should be the first thing you call inside that function.

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

```
function Stopwatch() {
    const [seconds, setSeconds] = useState(0);

    return <div>{seconds}</div>
}
```

▼ Click event on button

```
import React from "react";

function Welcome() {
    return <button onClick={() => console.log("button was clicked")}>Click me</button>
}
```

▼ How to update the state?

To update the state, you always have to use the 'set' function that you get back from useState.

```
const [seconds, setSeconds] = useState(0);
...
<button onClick={() => setSeconds(seconds + 1)}>Click to add 1</button>
```

▼ When you call the 'set' function from useState what happens?

React knows that this Component's state has changed, which means it needs to re-render, so the component function is called again

▼ What is the benefit of a 'set' function for state?

Normally with Vanilla JavaScript (JavaScript without any library), you'd have to manually update the DOM every time a variable changes and that could quickly become messy as the requirements get more complicated.

▼ What is a closure and why it is useful?

A closure is when a function contains another function.

Closures are useful because we'd like to define a new function while keeping access to the variables defined in the outer function.

▼ Closure example in React: event handlers that access component state

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

function Counter() {
    const [count, setCount] = useState(0);
    function handleIncrementClick() {
        setCount(count + 1);
    }

    return (<>
        <div>{count} times clicked</div>
        <button onClick={handleIncrementClick}>Add 1</button>
    </>);
}
```

▼ Why should you only add the onClick attribute onto button elements?

Some of your users may be using a Screen Reader to access your website, and if you add an onClick to elements other than the button, then they may not be able to click on those elements.

▼ What is onKeyDown()?

attaches the keydown event to an element

▼ What is onChange()?

attaches event to change on a select element for example

▼ What is an inline function?

a function written where it will be executed (instead of a named function with func definition elsewhere)

onClick={() ⇒ console.log('Clicked')}

▼ How to pass a reference to a function for a click event?

```
function handleLoginClick() {
    console.log("Logging in..");
};

<button onClick={handleLoginClick}>Login</button>

// this will NOT work
<button onClick={handleLoginClick()}>Login</button>
```

▼ How can props be used to conditionally change state?

```
function handleIncrementClick() {
      if (props.enabled){
          setCounter(counter + 1);
      }
  }
...
const element1 = <Counter enabled={true} />;
```

▼ Event handler naming convention?

handleSubjectEvent e.g. handleFormSubmit

▼ What can you never wrap useState in?

an if condition

▼ Conditionally render components example

```
import React from "react";
import DarkTheme from "./DarkTheme.js"
import LightTheme from "./LightTheme.js"

function App(props) {
    if (props.theme === "dark"){
        return <DarkTheme />;
    }
    return <LightTheme />;
}
```

▼ How can you store a JSX element in a variable?

```
const loginButton = <button>Login</button>;
if (props.existingUser) {
        return <div className="app-container">{loginButton}</div>;
    }
```

▼ Ternary operator with rendering components

something ? true : false

do NOT try and use the ternary operator everywhere.

You should always strive to write readable code not necessarily fewer characters.

```
import React from "react";

function App(props) {
    const loginButton = <button>Login</button>;
    const signupButton = <button>Signup</button>;

    return <div className="app-container">{props.existingUser ? loginButton : signupButton}</div>;
}
```

▼ How to make use of logical && operator to conditionally render JSX?

```
// instead of this
```

```
function Counter(props) {
    if (props.count > 99) {
        return <>
            <h1>You have {props.count} items</h1>
            <p>That is a lot of items.</p>
        </>;
    }
    return <h1>You have {props.count} items</h1>;
}
```

```
// do this

function Counter(props) {
    return <>
        <h1>You have {props.count} items</h1>
        {props.count > 99 && <p>That is a lot of items.</p>}
    </>;
}
```

▼ Conditional rendering && example

```
function Navbar(props) {
    const msg = <p>You've got {props.notifications.length} notifications</p>;
    return (
        <>
            <h2>Welcome user</h2>
            {props.notifications.length > 0 && msg}
        </>
    );
}
```

▼ State with booleans and JSX (example)

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

function App() {
    const [isFree, setIsFree] = useState(true);

    function handleUpgradeClick() {
        setIsFree(false);
    }

    if (isFree) {
        return <button onClick={handleUpgradeClick}>Upgrade</button>;
    }
    return <h2>Welcome Paid user!</h2>;
}
```

▼ How to use multiple states in the same component (example)

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

function Counter() {
    const [count, setCount] = useState(0)
    const [isLocked, setIsLocked] = useState(false);

    //...

    return null;
}
```

▼ What are 2 Hooks rules you must follow?

1) Only call Hooks from React functions

2) Only call Hooks at the Top Level of a function and never call hooks inside loops, conditions, or nested functions.

```
function App(props){
    // this will NOT work
    if (some_condition){
        const [count, useCount] = useState(0);
    }
}
```

▼ React relies on the order in which Hooks are what?

called

▼ Why are [] === [] and {} === {} both false?

they're different instances of either an Array or an Object

```
When you write [], it's the same as creating a new instance of Array.
When you write {}, it's the same as creating a new instance of Object.
```

▼ How can you create a reference to another array?

```
const first_array = [];
const second_array = first_array; // second_array now points to first_array
console.log(first_array); // []
console.log(second_array); // []

first_array.push(10);
console.log(first_array); // [10]
console.log(second_array); // [10]
```

▼ What is Immutability?

An immutable object is an object that cannot be changed.

Every update creates a new value, leaving the old one untouched.

▼ What is a deep equal comparison?

Deep equal is when you compare 2 objects based on their values.

With deep equal, [] would be equal to [].

However, JavaScript does NOT have a built-in method for deep equal, which means you will have to resort to an external library or a hack.

▼ Why does React create immutability?

The 'set' function from useState hook does a comparison.

When state === newState is false, it means that the state has changed, and React has to re-render the Component using ReactDOM.

In order for the comparison to be false, the operands must be different and not share a reference to the same object.

▼

▼

▼

▼

▼

▼

▼

▼

▼

▼

▼

▼

▼

▼

▼

▼

▼

▼

▼