Hands-On - #1

Q1. Open/Create the LAB01 folder in VS Code and perform the following:

[a] Include the required React and ReactDOM libraries from CDN
[b] Include the Babel library for JSX also

The above libraries are included in the 'index.html' file.

[c] In the body of the HTML document, include the tag with the 'id' attribute set to 'root'

[d] Babel is a very famous transpiler that basically allows us to use JSX. However, we need to state that with the 'type' attribute as 'text/babel'

[e] Now in the 'index.js' file write you first line of React code as follows:

```
ReactDOM.render(<h1>Hello Everyone!<h1>, document.getElementById("root"))
```

Save the files and run your code.

Q2. Modify the above 'index.js' file which should help us in displaying

[a] A text as follows: Hi, my name is <your_name>! with a <p> tag.

[b] Can we use the 'querySelector()' instead of 'getElementById()'

Q3. Surprise! You probably thought you could just forget the line of code you just learned! Nope, not on my watch!

Try to write that 1-liner of React code again!

This time, see if you can figure out how to render an <ul> with 2+ s inside

Example:

React Features

* JSX
* Components
* Virtual DOM
* Simplicity
* Performance

Can we also have an header as 'React Features' along with the un-ordered list.

Think of it!

Q4. Open the LAB02 folder in VS Code. Look into the 'index.js' file.

We are trying to understand the 'Composable' feature of React.

Your task is to create your own custom React component.

The name of the component is 'MainContent' which should render the text: I'm learning React! inside an <h1> tag

It should be rendered below the 'Navbar' web-component

Q5. Open the LAB03 folder in VS Code. Look into the 'index.js' file.

We are trying to understand the difference between 'Declarative' and 'Imperative' in this case.

It is already been told - React is Declaractive.

We are trying to figure it our, if it were to be 'imperative' how we were suppose to write the code in pure Vinilla JS.

Your task is to write the imperative way of rendering the text.

Q6. Open the LAB04 folder in VS Code. Look into the 'index.js' file.

Let's figure out what is happening behind the hood when we use JSX.

To do understand that we can just, used console.log to log the information of our JSX element.

Open your console window and observe the follow:
- type, props (short for properties) className, content etc.

Can you identify what type of data is being returned by JSX.

Your task now is to write a 'page' JSX element which will help us in rendering the text 'This is JSX' with the <h1> tag and under that render the text 'We are learning JSX too...' with the <p> tag.

Console log the 'page' JSX element and record your observation.

NOTE: With JSX, the attribute is 'className' and NOT just 'class'

In JSX attributes are ALWAYS specified in CAMEL CASE.

Q7. Open the LAB05 folder and complete the task.

Q8. Create a development environment by using the 'create-react-app' by using the 'npx' Node Package Runner/Executer

[a] Give the name of your React app as 'myapp'

[b] Once the 'npx' command is completed, record your observation in the command windows.

- Make a note of how to start the development server

[c] Open your React app folder 'myapp' in VS Code and

look into the different files. Particularly - 'package.json', 'index.html', 'index.js', 'App.js' etc.

[d] Open the 'Terminal' within VS Code.

[e] Start the development server from the VS Code Terminal and observe your web-browser

It should now be displaying the 'React' log with the text 'Edit src/App.js and save to reload'

[f] Modify the 'App.js' file such that the app displays the text 'Hello from React' and save the file.

Record your observation.

NOTE: Most of the time we shall be working with the files in the 'src' directory.

Q9. There is no point in executing the 'npx create-react-app <app-name>' command again and again. This will download the modules and their dependencies and creates the dir. structure for us every time.

This is time consuming.

To make things simple for us to work with React, make a copy of the 'myapp' folder and name it as 'template'

Moreover, remove all the files under 'src' directory.

To check if this setup is working or not, just create a simple 'index.js' file in the 'src' folder and have the basic few lines of React code to display 'Hello once again!!!'

- Check if it is working by starting your development server.

NOTE: Now you should be sure that the 'src' directory files are used in the working of our React app.

We shall be making changes in this 'src' directory and continue our learning of React!

Q10. Your next challenge is to find out what happens if we try to append JSX to our div#root using .append() instead of ReactDOM

a) Create a sample page in JSX (≥ 4 elements) and save them in a variable

b) Select the div with the ID of "root" and use `.append()` to append your JSX

c) See if you can guess what will show up in the browser before running the code

d) See if you can explain what actually shows up in the browser

Observe, that React internally deals with JSON objects.

Fix the code now!

Don't forget, you're not using CDNs anymore, so there's no global "ReactDOM" variable to use anymore.

Q11. Your next challenge:

Starting from scratch, build and render the HTML for our section project.

We'll be adding styling to it later.

Hints:
* The React logo is a file in the project tree, so you can access it by using `src="./react-logo.png" in your image element

* You can also set the `width` attribute of the image element just like in HTML to 40px

The web page should look like the one shown below:

NOTE: If you are using React >= 18.0 check out where are static files stored.

Ask Google: Where are static files stored in React 18