**React**

**Introduction to React**

**|---> JavaScript Modules**

**|---> React Elements vs DOM Elements**

**|---> Setting up a Fresh React Project using npm**

**|---> Webpack, CSS Loaders**

**|---> JSX & Babel compiler**

**JavaScript Modules**

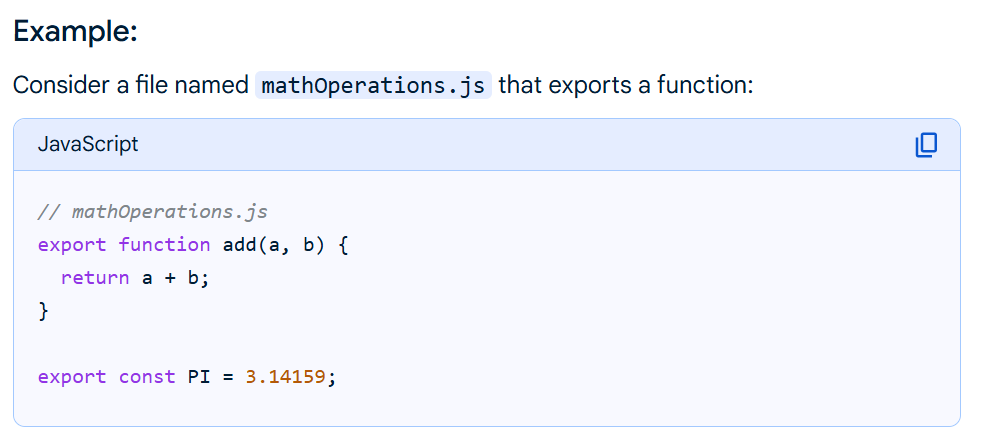
* JavaScript modules are a feature that allows developers to organize and structure code into reusable, self-contained units.
* This approach helps in managing complexity, improving code readability, and promoting reusability in larger applications.

**Import and Export:**

Modules use import and export statements to share functionality between different files.

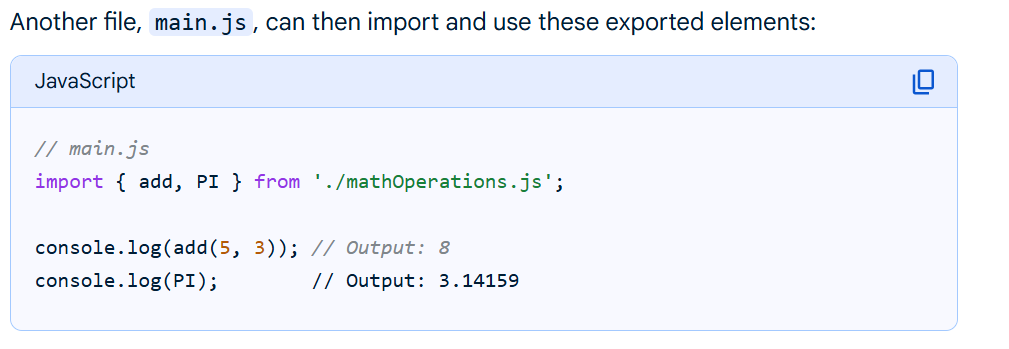
* + export: Used to make variables, functions, classes, or objects available for use in other modules.
  + import: Used to bring exported elements from other modules into the current module.

Lets see it with a example how JavaScript Modules work in the below I mentioned it with the example:



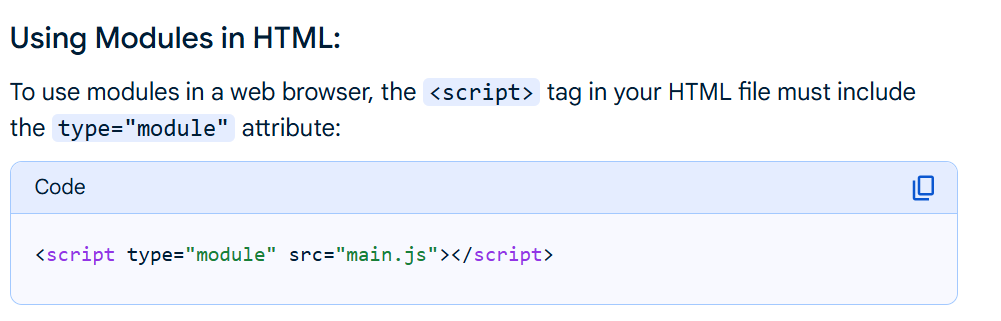
Here above we created a file called **mathOperations.js** where we wrote the functionalities of **“addition operation”** and stored **PI value**.

Now we want want use the **functionalities** and the value that are stored in the **mathOperations.js** in another JavaScript files you can simply do it **by importing** them into your current JavaScript file.



Here we created a **main.js** file where we are **importing the functionalities of the add and PI** into our **main.js** file.

And when are you using other files function in the HTML page you should include your **working file** as **source** and set **type** as **“module”**.



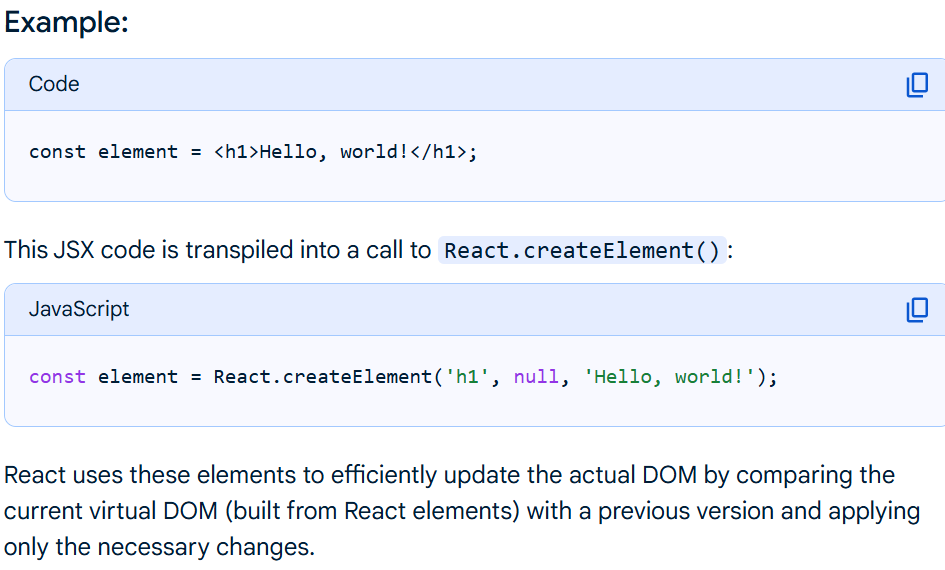
**Benefits of Using Modules:**

* **Improved Code Organization:** Keeps related code together in separate files.
* **Enhanced Reusability:** Allows sharing code across different parts of an application or even between projects.
* **Easier Maintenance:** Smaller, focused modules are simpler to debug and update.
* **Reduced Global Scope Pollution:** Variables and functions are not automatically added to the global scope, minimizing conflicts.

**React Elements vs DOM Elements**

**React Elements:**

* React elements are the fundamental **building blocks** of a React application's user interface.
* They are **plain JavaScript objects** that describe what you want to see on the screen.
* Unlike browser **DOM elements**, React elements are **lightweight** and **immutable**.



Here in the above JSX code the **createElement()** takes **three arguments** or more where :

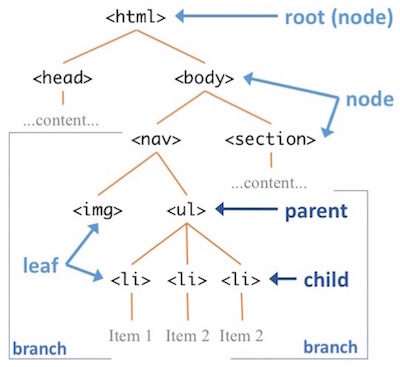
**First Argument**: it takes the **type of the element** here it is “h1”

**Second Argument**: it takes the **properties of the element** like {id : “title”, …} here in the above example we don’t have any arguments so we wrote “null”

**Third Argument**: From here onwards it consider them as **its children** of its element and also after this argument everything it consider as a children of it.

**DOM Elements:**

* DOM elements are the **individual components** or nodes that make up the Document Object Model (DOM) of a web page.
* When a browser loads an HTML document, it creates a **DOM tree**, where each HTML tag, text content, and even comments are represented as a node in this tree structure.
* These nodes are what are referred to as DOM elements.



You can access the **HTML elements** by using DOM by there **Class name** or **ID name**.

In the HTML DOM, the **Element object** represents an HTML element, like **P, DIV, A, TABLE,** or **any other HTML element**.

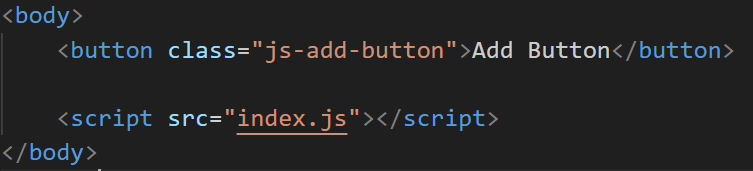
We have so many **methods** to access the HTML elements inside our JavaScript file.

To Explore more methods [Refer this site](https://www.w3schools.com/jsref/dom_obj_all.asp).

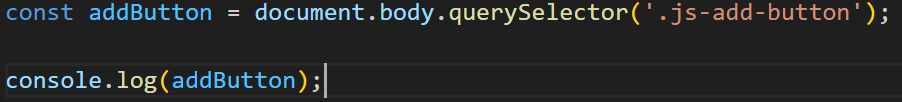
In that one method is **querySelector()** where you can access the particular **Class Element** and also you can store it in the **variable** also.

I will show you it with an Example see the below example:

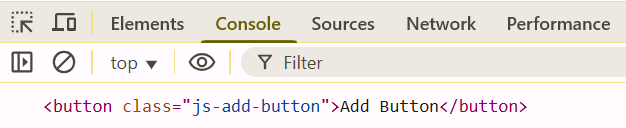
1.This is the HTML page and we can access this class by using querySelector() method:



2.In JavaScript file we can access it like shown in the below image:



3.You can check it by console.log() weather we are accessing that element or not as shown in the below image:



The distinction between a DOM element and a React element is fundamental to understanding how React applications render user interfaces.

**DOM element vs React Element:**

**DOM Element:**

* A DOM element is an actual HTML element that exists in the browser's Document Object Model.
* It represents a visible, interactive part of a web page and is directly manipulated by the browser.
* When you inspect a web page using developer tools, you are looking at the live DOM elements.

**React Element:**

* A React element is a lightweight, plain JavaScript object that describes what you want to see on the screen.
* It is a virtual representation of a DOM element, not the actual DOM element itself.
* React elements are immutable, meaning once created, their properties (children, attributes) cannot be changed.

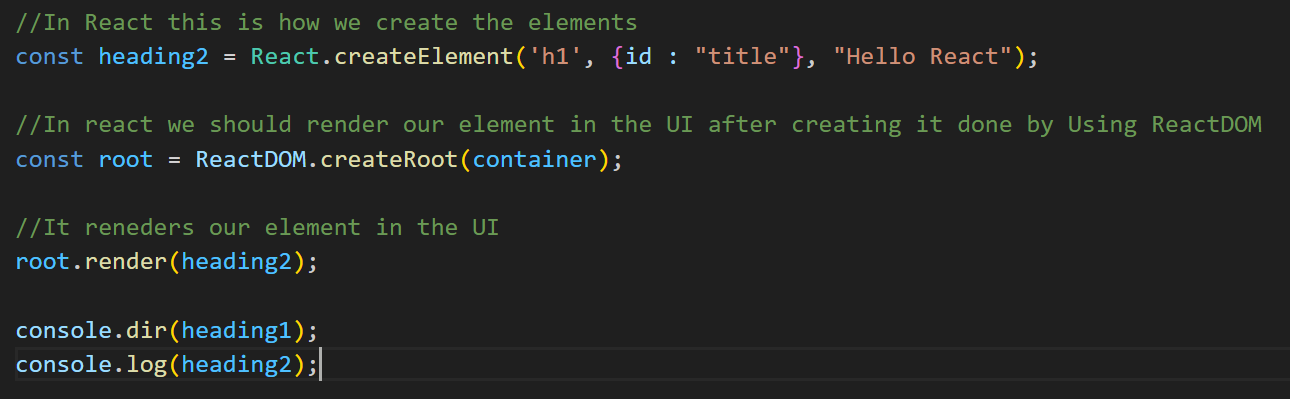
In **DOM** elements we create the or generate the elements **manually** in the JavaScript file where the react elements are all written in the **JavaScript code** only.

Simply React acts a bridge between **Actual DOM** and **UI**.

Here React is a **light weight** because it takes the **less storage** when you compared it with the Actual DOM.

For Example, you print heading element in javascript and React and open console to check if you see the actual js heading gives you lot of properties which takes more space(storage) then the React elements which makes the React a light weight.

After create the element in the react you can not use or render directly in UI that’s where we use the **react-dom** and a **createRoot(“root”)** function.It renders our element in to the UI.

****

**Setting Up A Fresh React Project Using npm**

1.For to set up a react project we need to install the **Node.js software from chrome** and **npm(in our project terminal)** in our project.

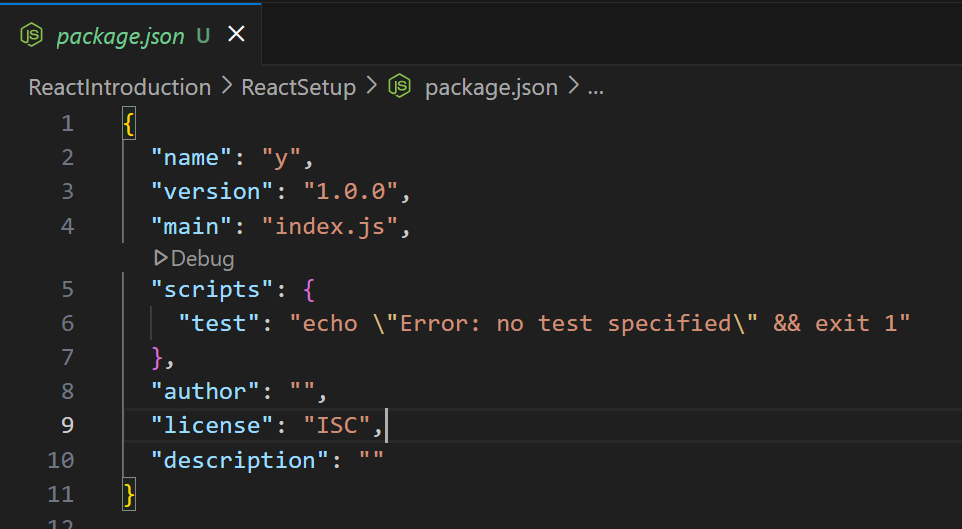
2.To install “npm” you can simply type **npm init** in your project folder in the terminal.

3.It will create a **package.json** file where it contains all the **details** and **dependencies** of you project in a **JSON** model (In simple words it will track your entire project what are installing and what versions you are using).

4.**JSON** means **JavaScript Object Notation** it will be store the data in the manner of objects in a **key : value** pair as a **Strings.**



5.When you run this command it will create a **package.json** file as shown in below image:



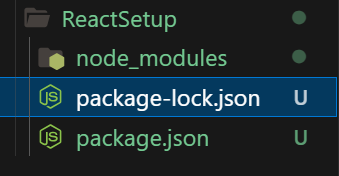
6.Now we should install the **react** and **react-dom** into our project to use the react in our project and we have command for it like **npm install react react-dom** as shown in below image:

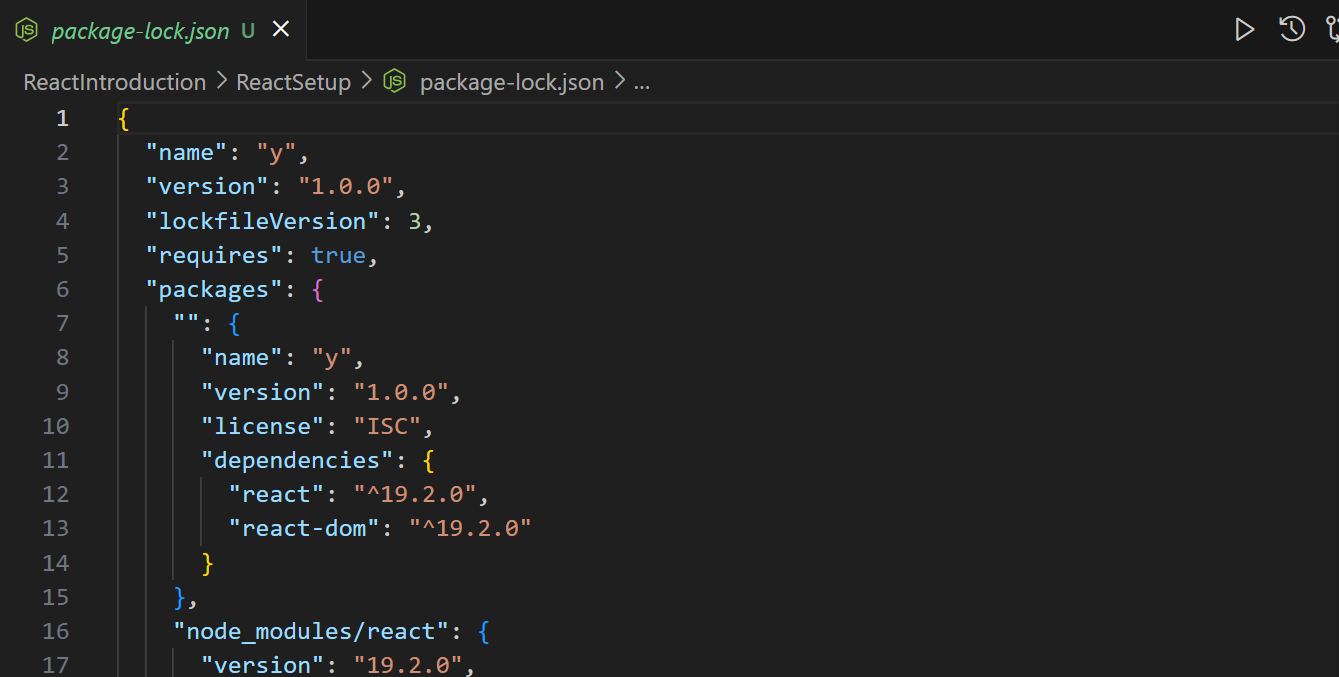


7.For example, in your future you may build large projects at that time we need to install all modules into our project you can simply use the **npm install command**.

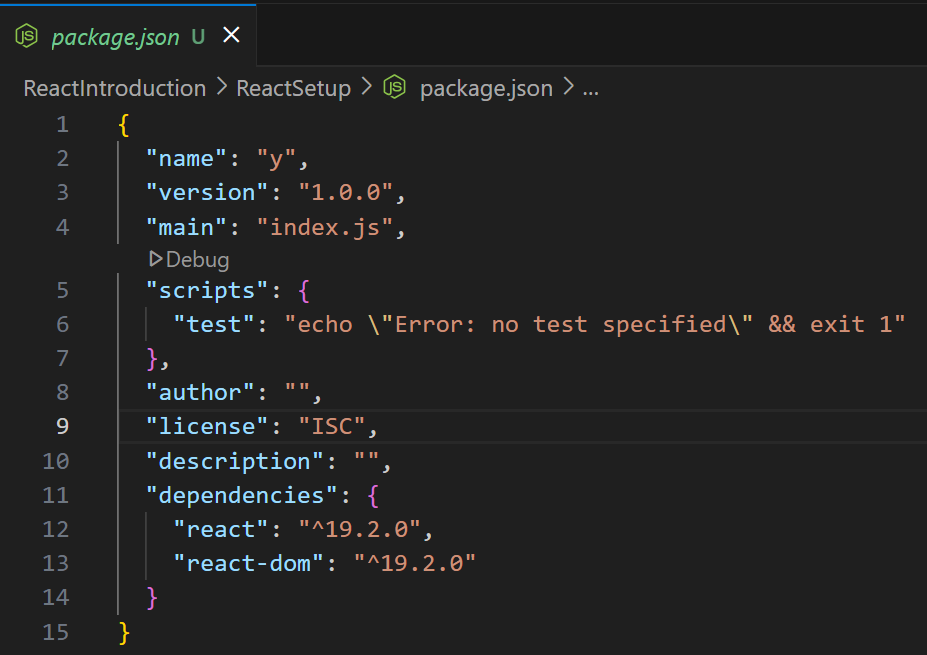


8.Upon installing **npm react and react-dom** it will create a one folder called **node modules** and another file called **package-lock.json**.





9.And also you can also see some changes are being made in the **package.json** file like adding dependencies what we installed.



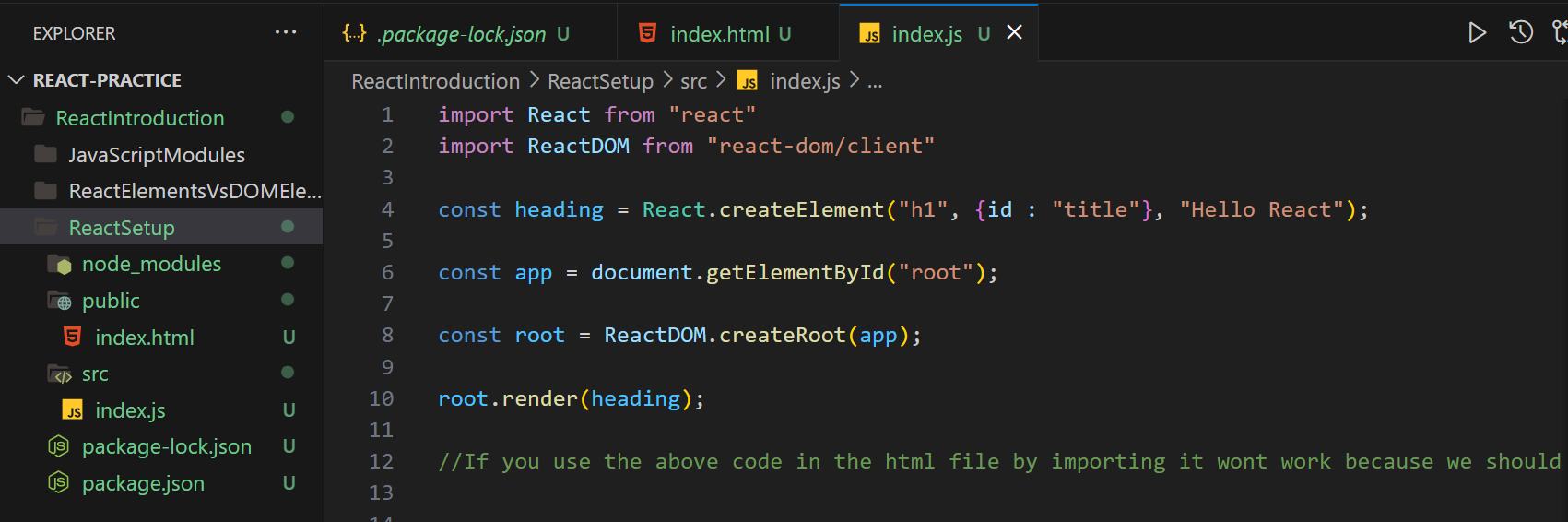
10.**package-lock.json** - it locks the dependencies that we are using in the project.

11.Suppose if you delete the **node\_modules** folder and you want to install it again you can run **npm install** it makes easy for npm to understand what should it download because they are already in our **package-lock.json**. That is the use of package-lock.json.

12.**public folder** – use to store our static files inside this folder.

13.**src folder** – here we store all our code related files and folders here.

14.if you create some files inside the src folder you should import react and react-dom right you don’t give the complete path you just import them like as shown in below image:



15.Now let us know about the npx:

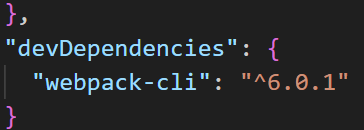
* npx means **"Node Package eXecute"** and is a tool that runs JavaScript packages from the npm registry without needing to install them globally.
* It's bundled with npm version 5.2.0 and above, making it a convenient way to use packages for one-off tasks or to test new versions, like running npx create-react-app to create a new React project.

**Webpack:**

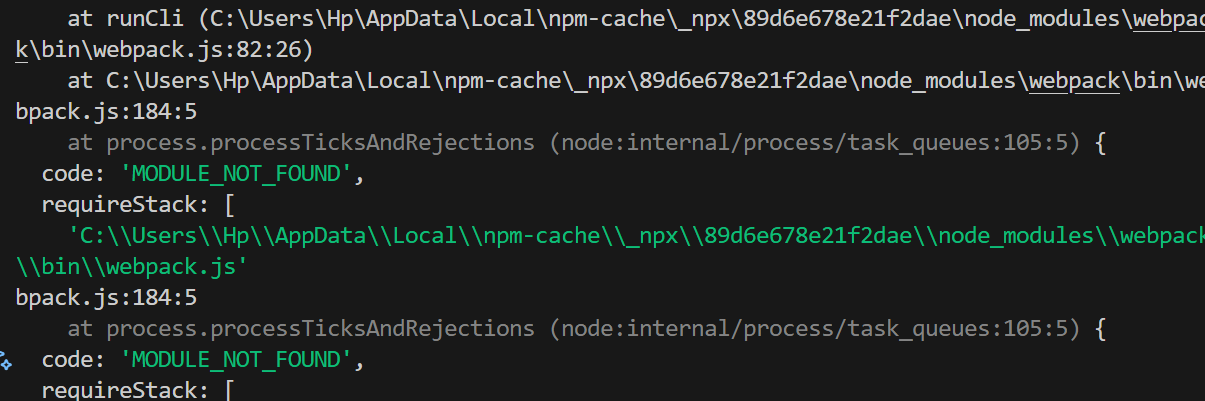
For example, we install the **npx webpack** bundler here in our project



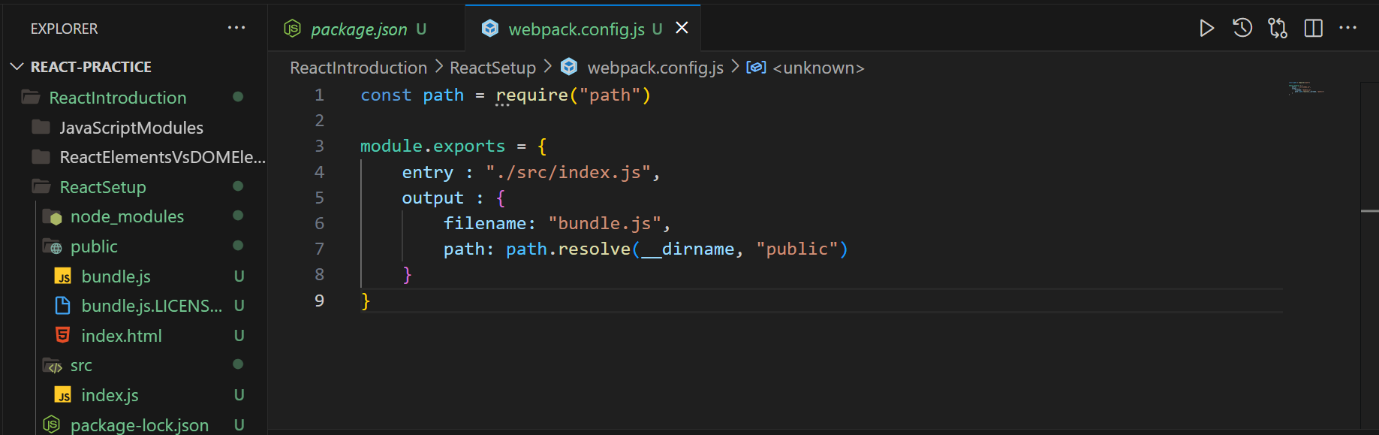
i.It will add the dependency into our **package.json** like shown in the below:



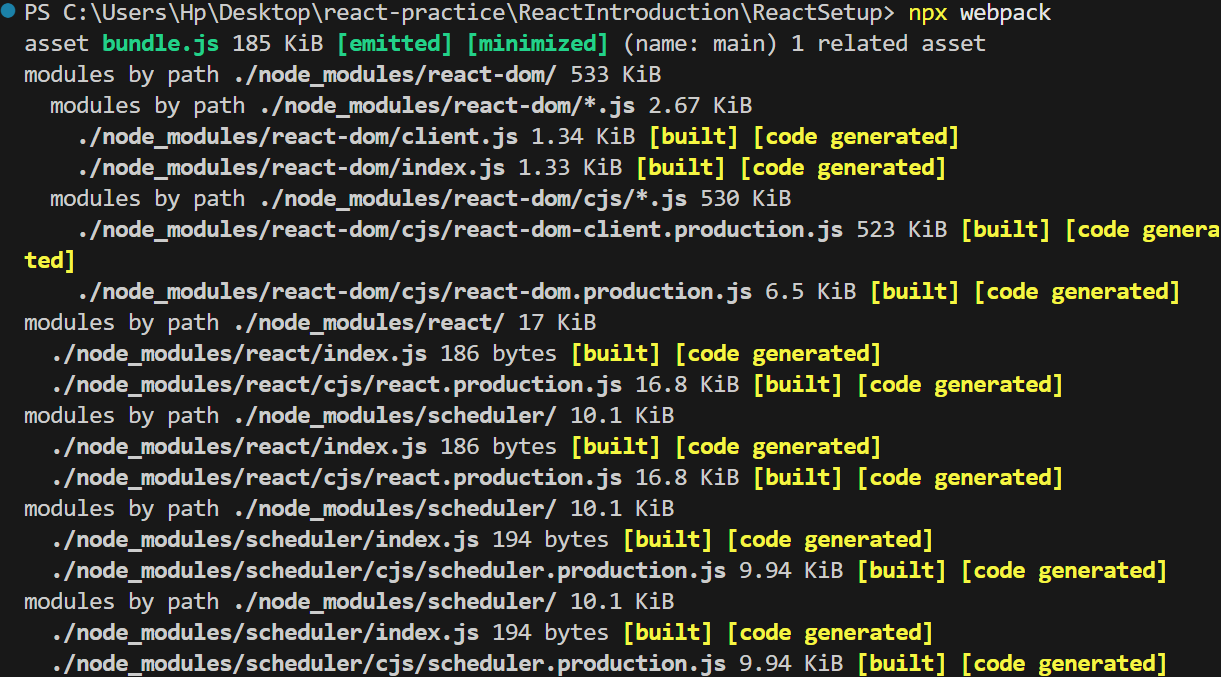
ii.But after the installation you will encounter with a problem as shown in below image:

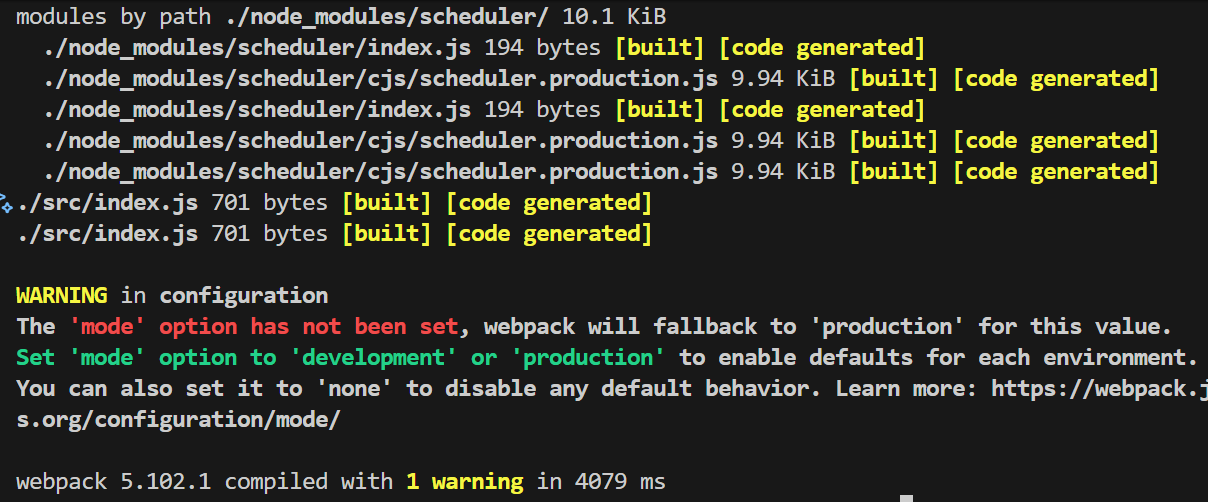


iii.To solve that error create **webpack.config.js file** and write the path to the webpack in it as shown in below(For every bundle installation we should give a starting point to it like telling it to install them here):



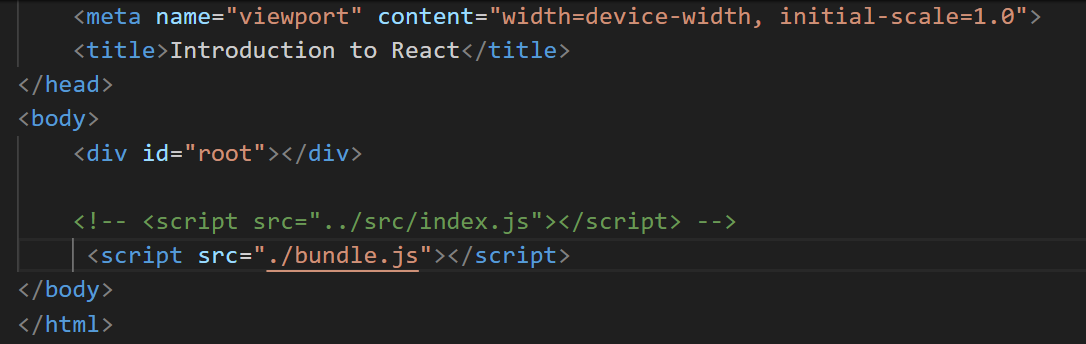
iv.Now again run the command **npx webpack** you will not get the error again because we gave a path to it and it will create a bundle.js file in the given path folder name.



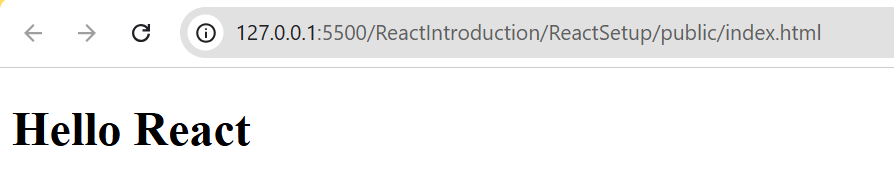


v.It will **bundle** every **single line of code** that we wrote in the point 14 it will go line by line and bundle it in bundle.js file

vi.Now we will link the **bundle.js file** into our HTML page now it will give us the correct ui with what we typed in the js files.



vii.Earlier we used index.js in the script tag before bundling it did not rendered the “Hello React” but after bundling using webpack it is working fine by using the bundle.js file.

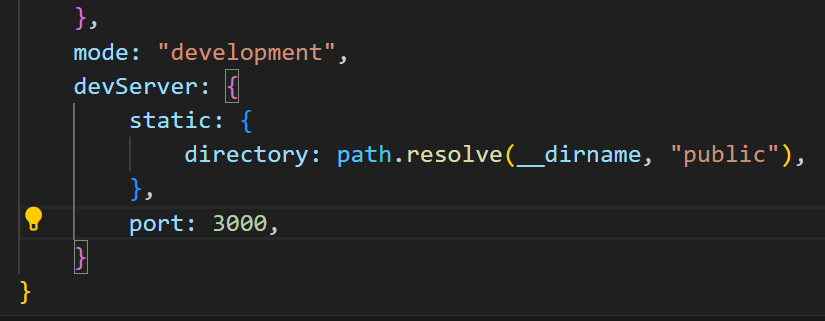


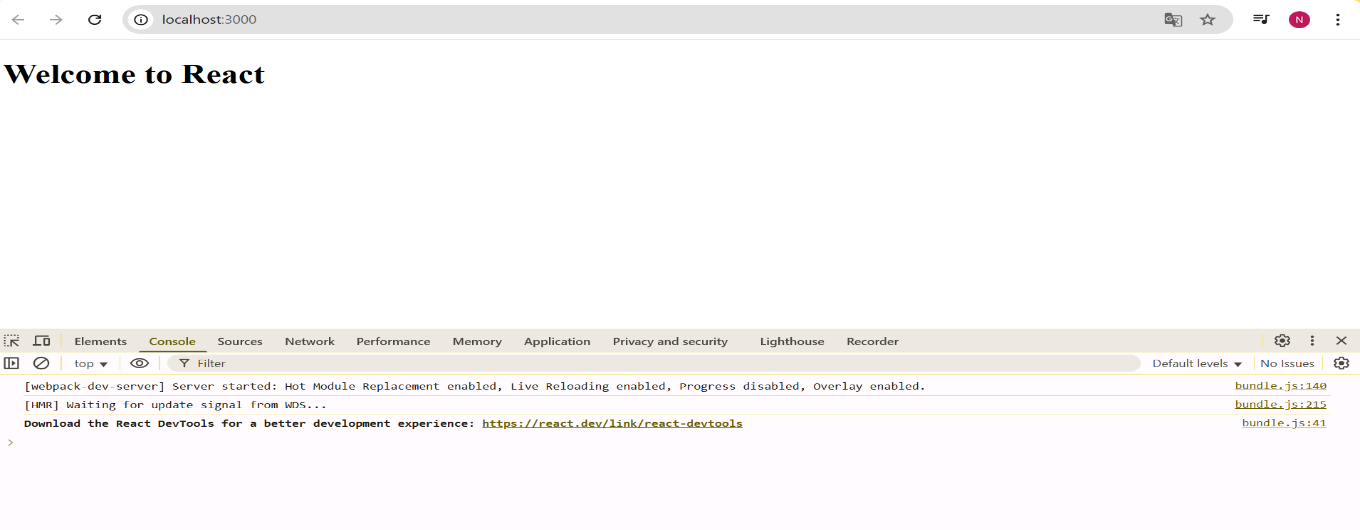
viii.This is how we render the elements into our pages.

ix.Suppose some want to use your package or you don’t want to use **npx again** and again we can simply add it to **scripts object(“scripts”: {“bundle”:”webpack”})** inside the **package.json** file and simply run **npm run bundle** in the terminal.which gives the same output.

x.We should run webpack every time when we change any thing in our code to reflect the changes in the page it is again a painful task.To overcome that we use **development server** and run it for code follow below:

run the **npx webpack server** it will setup the **localhost** to run our code and update automatically when we made the changes to it.





Now you can see it is running on development server localhost:3000 when you made changes it will automatically updates in the page. You don’t need to run npm run bundle again and again.

**CSS Loaders**

We want to **style** our elements and project at that time we use the **css loaders** or **plugins**.

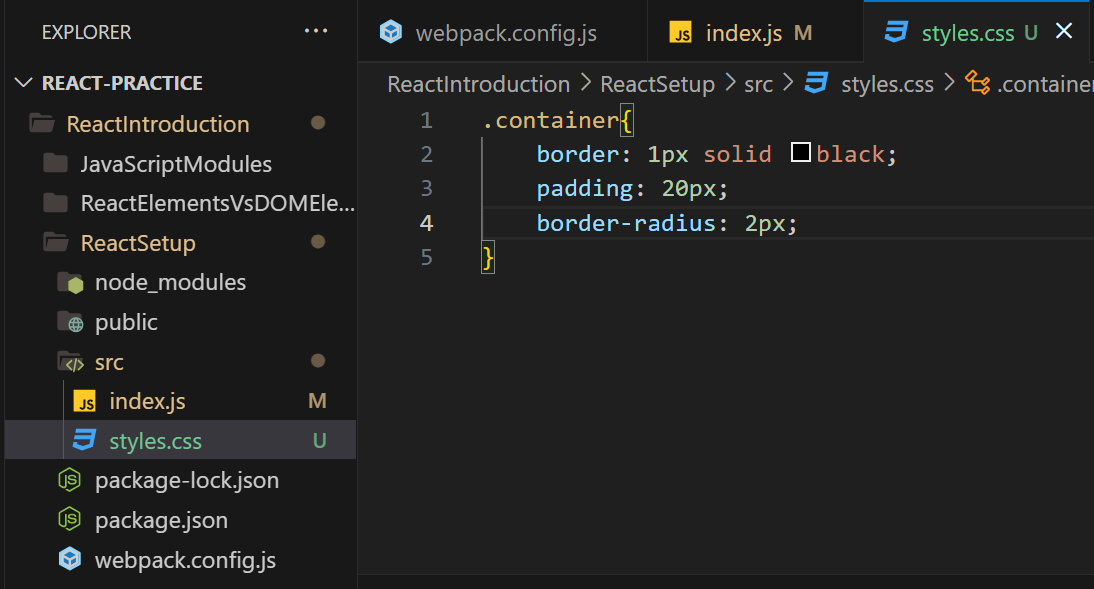
Simply it is like styling our web page.

We can give css styling in two ways like inline css and external css.

**Inline css:**



**External css:**



Now when you **import it to your project** you will get error something called can not convert this **css file into an js file to bundle** it on js files can bundle.

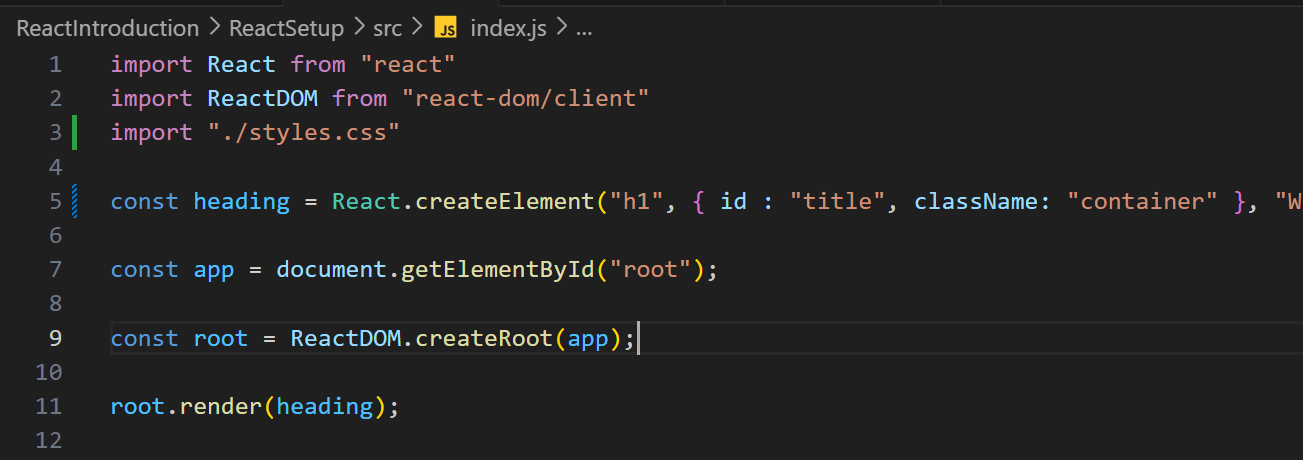
In these where **plugins** are very useful for us. Search for the webpack css loaders in the google you will get the code allows your css files to run on the webpage or bundle them.



Here it will read the css files and for that we need install the style-loader and css-loader for that **use npm I -D style-loader css-loader**.

We are using **-D** because telling it to add in the modules in the **devDependencies** also.

After restart the server again and you can use the css classes by using className in the file to use that particular style.





Like these is how we use the css loaders and like this we have many other loaders like when you are working with saas we use saas loader etc.. like that.

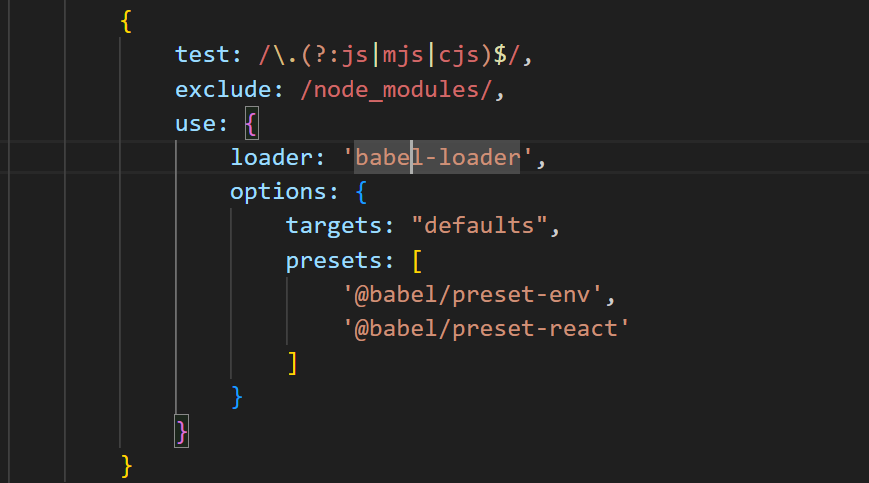
**JSX & Babel Compiler**

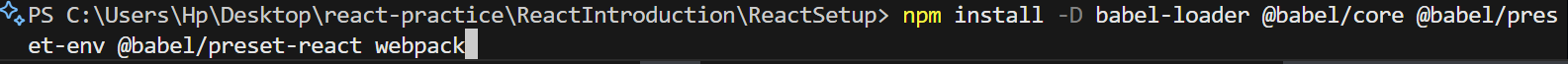
Suppose you build a application and it is available to everyone and you build the application using latest methods of a language and some one is seeing it in a older version or system so definitely it will crash because it don’t supports that version right! So in that case **babel compiler** is useful where it will convert our code **into every react version** and runs it.

In simple worlds to convert **XML** into **react elements** we use the babel compiler.

You can check it how it converts and works by searching in the google like babel online compiler.

It will make our development easy so for that again we need to integrate the babel inside the webpack configuration.





Check the site every update it changes the code slightly during installation.



You can go and check in the localhost:3000/bundle.js your element will be created automatically convert this <h1> code into react code.

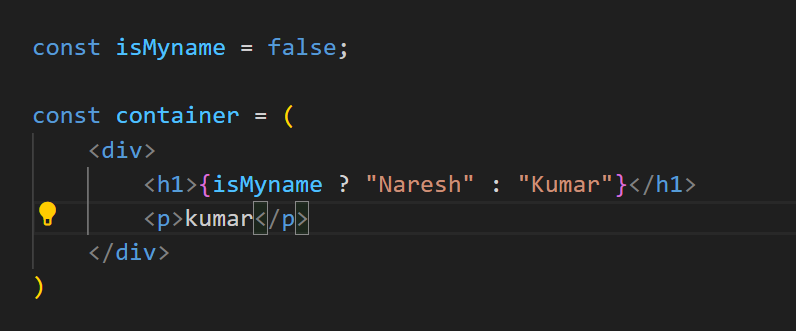
**JSX** allows to write the HTML and JS together in the react.

**JSX Rules:**

The jsx should always wrap into a **single element** you can not use multiple elements separately (in simple words it should have a single parent element inside it can have a as many as child elements).

You can also wrap your code inside a fragment <> </> like this.

You can write **expressions** or **JS variable** inside a **{ }** brackets.



**What can’t do in JSX is:**

You can’t use **null, undefined, true, false** inside a **{ }** brackets it will return empty or nothing in the page.

And also you can’t render the objects directly inside the {} it will throw you an error.

