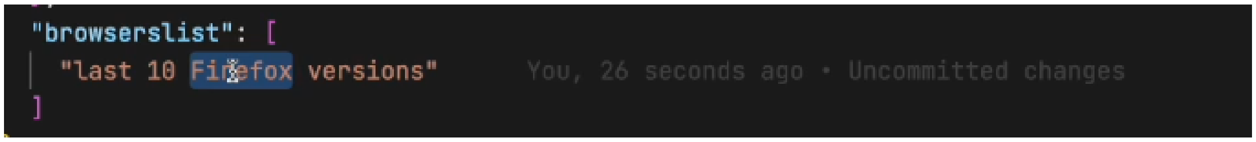
# 4. LAYING THE FOUNDATION



```
"browserslist": [
  "last 10 Firefox versions"      You, 26 seconds ago • Uncommitted changes
]
```

this does not means that this only work for firexfox

this means it will definately work on firefox else browser mein bhi run kr sakta hai

but why it wont works on some browsers the reason is that some browser dont know es6, or some browser dont know what is map function

so in that case **pollyfill** is used

A "polyfill" (sometimes spelled "pollyfill" or "polyfiller") is a term used in web development to describe a piece of code or a script that provides modern functionality to older web browsers that lack support for certain features or APIs. It essentially "fills in" the gaps or missing functionality in older browsers, enabling them to work with newer web technologies.

for example if map function wont work than we can create own map function so that it work

pollyfill is also created by parcel so that our webpage also work on oldder browser

Yes, Parcel, a popular web application bundler, provides built-in support for polyfills. When configuring your project with Parcel, you can specify which browsers you want to support, and Parcel will automatically include the necessary polyfills based on your target browser list.

**Babel is primarily responsible for providing polyfills, not Parcel.**

parcel mein babel hota hai isliye pollyfill supported by parcel we dont need to do polyfill babel does for us itself so that our app work properly in old browser also

Babel is a widely used JavaScript compiler and transpiler that allows developers to write modern JavaScript code while ensuring compatibility with older browsers and environments. It is a toolchain that transforms code written in the latest ECMAScript (ES) standards into backward-compatible versions that can run in older environments.

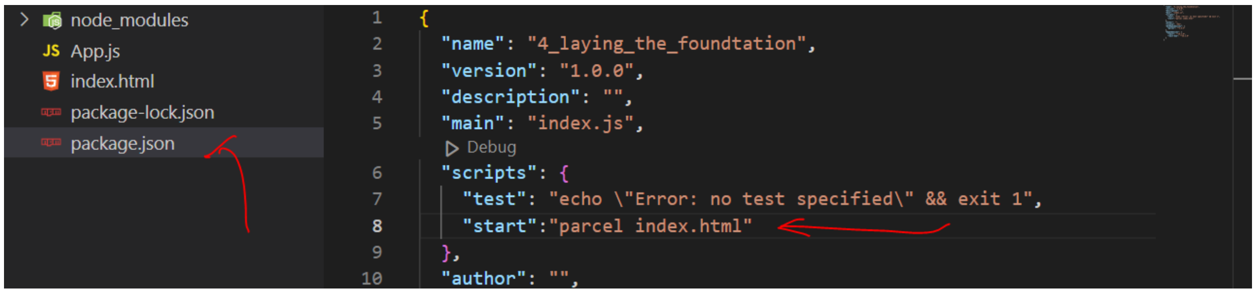
Babel is often integrated into build systems, such as bundlers like Parcel, webpack, or Rollup, to automatically process JavaScript code during the build process. I

in large projects webpack is used that is alternate to parcel

react create app use the webpack

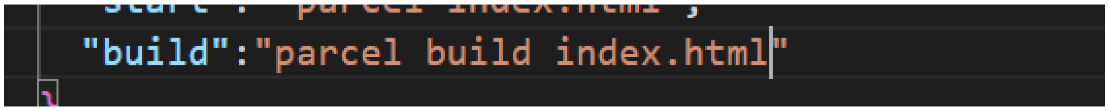
now if we need to create a local server again or build again than we dont use comands again and again we use below methods

In the context of a `package.json` file, the `"scripts"` field is used to define a collection of scripts that can be executed via the command line using a package manager like npm or Yarn. These scripts provide a convenient way to automate various development tasks or define custom commands for your project.



```
{
  "name": "4_laying_the_foundtation",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1",
    "start":"parcel index.html"
  },
  "author": "",
```

now how to run that command using this command ⇒ **npm run start** simiarly we can write script for build



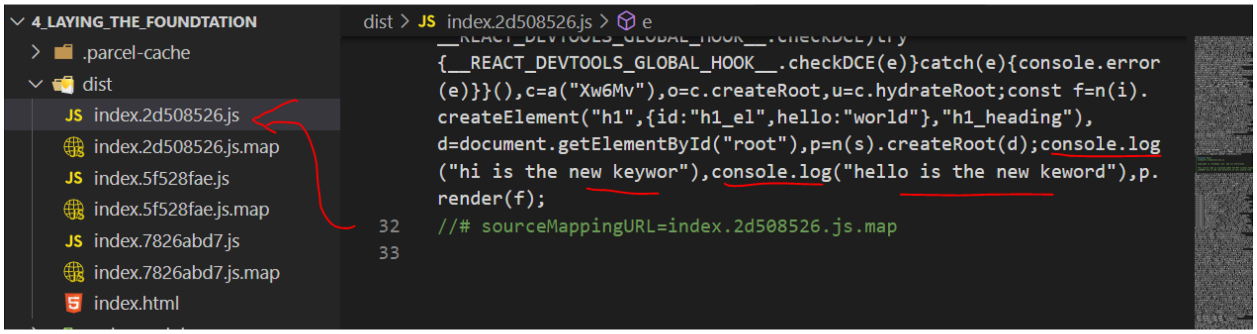
```
"build":"parcel build index.html"
```

how to run this above script⇒ **npm run build**

we can assume that **npx== npm run**

we can also do **npm start** this also work for only start command not for build

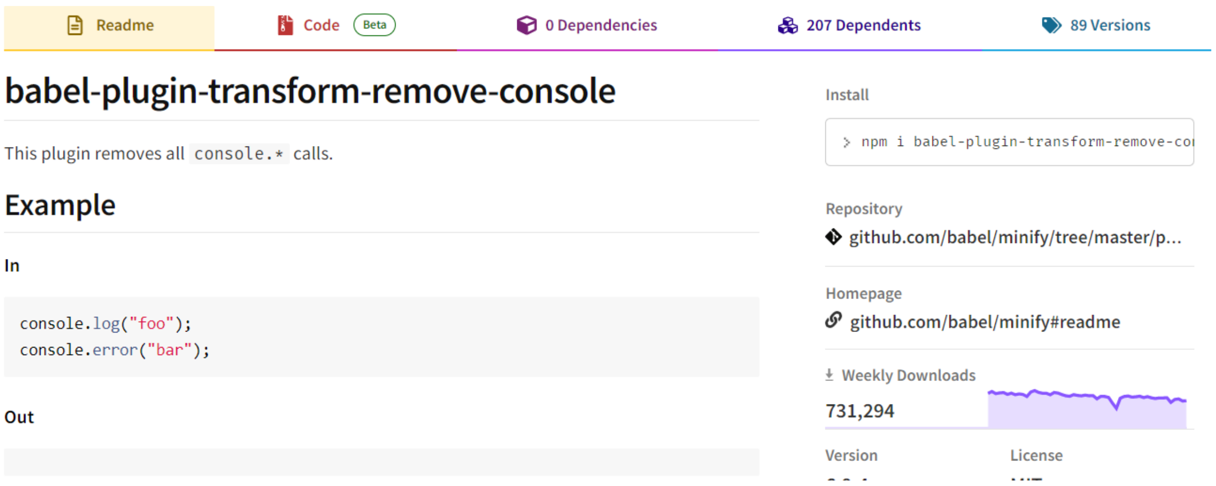
parcel will not remove **console.log();** automatically  we need to configure the project with plugin in order to remove the **console.log();**



```
4_LAYING_THE_FOUNDTATION
  > .parcel-cache
  v dist
      index.2d508526.js
      index.2d508526.js.map
      index.5f528fae.js
      index.5f528fae.js.map
      index.7826abd7.js
      index.7826abd7.js.map
      index.html
```

```
dist > index.2d508526.js > e
{__REACT_DEVTOOLS_GLOBAL_HOOK__.checkDCE(e)}catch(e){console.error
(e)}}(),c=a("Xw6Mv"),o=c.createRoot,u=c.hydrateRoot;const f=n(i).
createElement("h1",{id:"h1_el",hello:"world"},"h1_heading"),
d=document.getElementById("root"),p=n(s).createRoot(d);console.log
("hi is the new keywor"),console.log("hello is the new keword"),p.
render(f);
//# sourceMappingURL=index.2d508526.js.map
```

we need to install this plugin in order to remove the console



Readme | Code Beta | 0 Dependencies | 207 Dependents | 89 Versions

## babel-plugin-transform-remove-console

This plugin removes all `console.*` calls.

### Example

**In**

```
console.log("foo");
console.error("bar");
```

**Out**

Install

```
> npm i babel-plugin-transform-remove-co
```

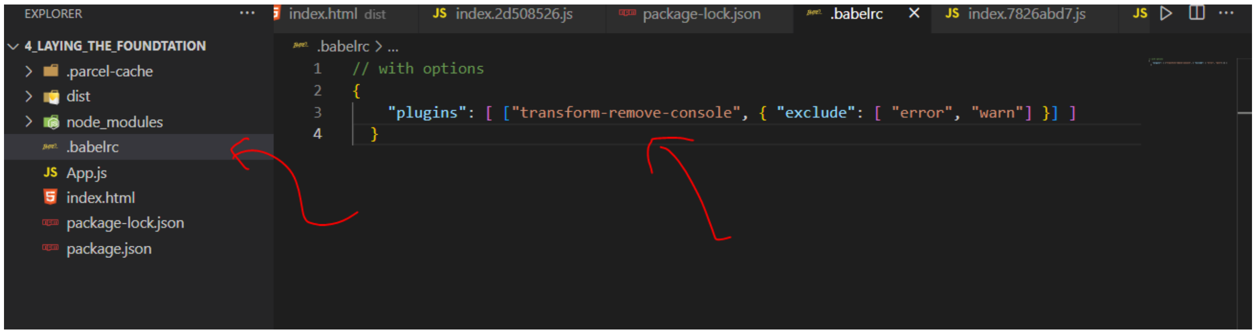
Repository
github.com/babel/minify/tree/master/p...

Homepage
github.com/babel/minify#readme

Weekly Downloads
731,294

Version | License

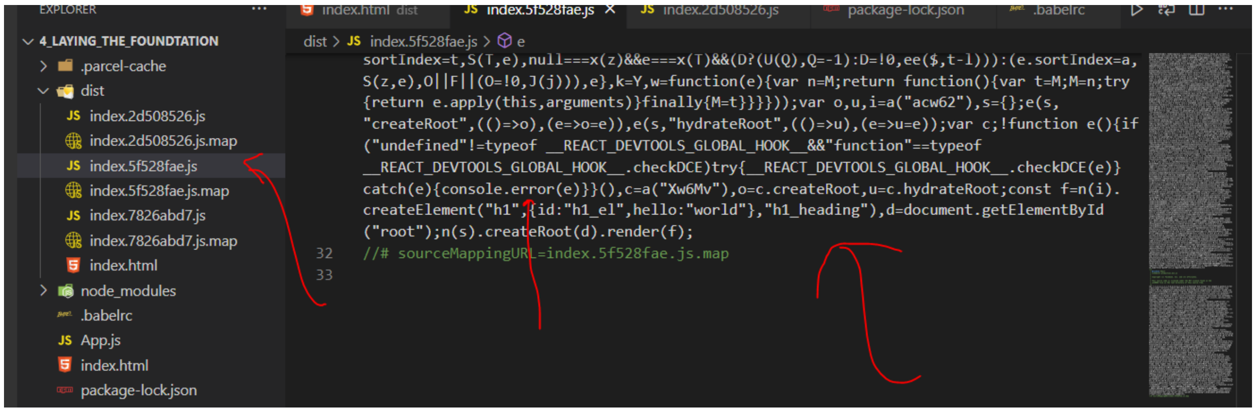
first install this plugin as dev dependenciy and than create a **.babelrc** file



```
// with options
{
    "plugins": [ ["transform-remove-console", { "exclude": [ "error", "warn"] }] ]
 }
```

now lets build again

before build it is recommened to delete dist folder than start the build because agar delete na kiya multiple js files bnti hai har build par



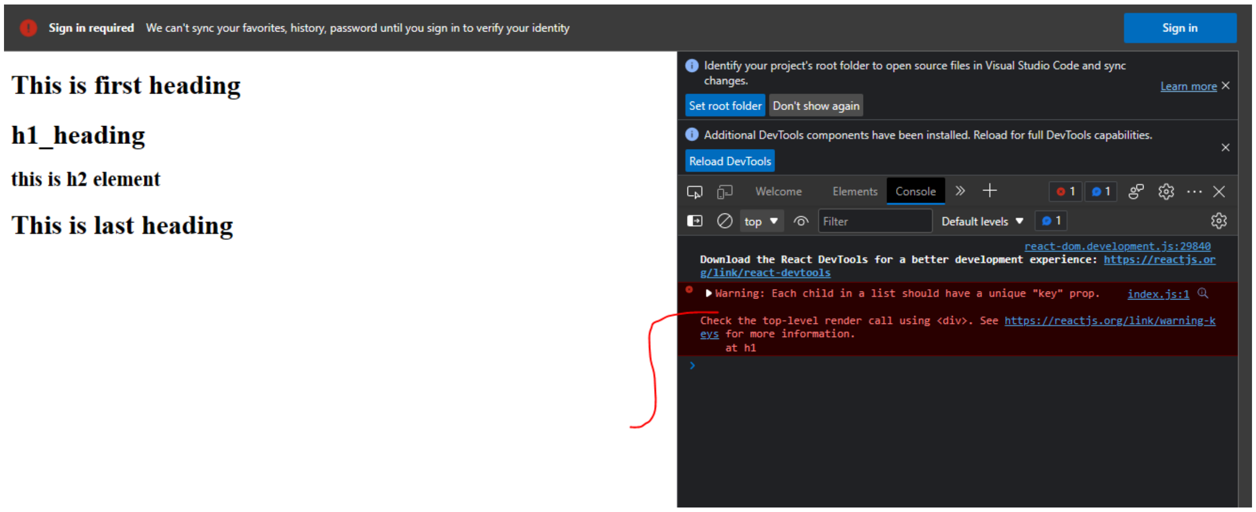
```
sortIndex=t,S(T,e),null===x(z)&&e===x(T)&&(D?(U(Q),Q=-1):D=!0,ee($,t-1))):(e.sortIndex=a,
S(z,e),O||F||(O=!0,J(j))),e},k=Y,w=function(e){var n=M;return function(){var t=M;M=n;try
{return e.apply(this,arguments)}finally{M=t}}}});var o,u,i=a("acw62"),s={};e(s,
"createRoot",(()=>o),(e=>o=e)),e(s,"hydrateRoot",(()=>u),(e=>u=e));var c;!function e(){if
("undefined"!=typeof __REACT_DEVTOOLS_GLOBAL_HOOK__&&"function"==typeof
__REACT_DEVTOOLS_GLOBAL_HOOK__.checkDCE)try{__REACT_DEVTOOLS_GLOBAL_HOOK__.checkDCE(e)}
catch(e){console.error(e)}}(),c=a("Xw6Mv"),o=c.createRoot,u=c.hydrateRoot;const f=n(i).
createElement("h1",{id:"h1_el",hello:"world"},"h1_heading"),d=document.getElementById
("root");n(s).createRoot(d).render(f);
//# sourceMappingURL=index.5f528fae.js.map
```

now we can see there is no console of message only console of error comes here

when we pass children or sibling to div like in below code

```
import React from "react";
import ReactDOM from "react-dom/client";
const heading=React.createElement("h1",{
    id:"h1_el",
    hello:"world"
},"h1_heading");

const heading2=React.createElement("h2",null,"this is h2 element");

const div=React.createElement("div",null,[heading,heading2]); // passsing childrens as siblings
const rootElement = document.getElementById('root');
const root = ReactDOM.createRoot(rootElement);

console.log("hi is the new keywor");
console.log("hello is the new keword");
root.render(div);
```

this belows error comes


Sign in required We can't sync your favorites, history, password until you sign in to verify your identity
Sign in
Identify your project's root folder to open source files in Visual Studio Code and sync changes.
Learn more
Set root folder
Don't show again
Additional DevTools components have been installed. Reload for full DevTools capabilities.
Reload DevTools
Welcome
Elements
Console
top
Filter
Default levels

This is first heading

h1_heading

this is h2 element

This is last heading

```
react-dom.development.js:29840
Download the React DevTools for a better development experience: https://reactjs.or
g/link/react-devtools
▶Warning: Each child in a list should have a unique "key" prop.    index.js:1

Check the top-level render call using <div>. See https://reactjs.org/link/warning-k
eys for more information.
    at h1
```

so inorder to resolve this error we need to pass keys which uniquely identitfy these

```
import React from "react";
import ReactDOM from "react-dom/client";
const heading=React.createElement("h1",{
    id:"h1_el",
    hello:"world",
    key:"heading_1"
},"h1_heading");

const heading2=React.createElement("h2",{key:"heading_2"},"this is h2 element");

const div=React.createElement("div",null,[heading,heading2]);
const rootElement = document.getElementById('root');
const root = ReactDOM.createRoot(rootElement);

console.log("hi is the new keywor");
console.log("hello is the new keword");
root.render(div);
```

updating something in the DOMtree is called render

In React, the `key` prop is used to uniquely identify and differentiate between sibling elements in a collection of items rendered in a list or iterable. The `key` prop is required when rendering arrays of elements in React components, especially when the list is dynamic and may change over time.

Here are the main reasons why `key` props are necessary in React:

1. **Element identification**: The `key` prop helps React identify each element in a list. It allows React to efficiently track changes, update, and reorder elements in the list when the underlying data changes. Without a `key` prop, React may have difficulty distinguishing between elements and may need to re-render and reconcile the entire list instead of updating only the necessary parts.

**https://www.youtube.com/watch?v=J_S97E8xjcA**

see this video in order to understand the need of keys

```
```

keys will always be unique for each element

**React.createElement** gives us the object which is then converted into the html and this html element put into the DOM using reactDOM

NOW LETS CREATE THIS BELOW HTML USING REACT

```
<div>
<h1>LOGO</h1>
<ul>
    <li>About us</li>
    <li>Contact us</li>
</ul>

    </div>
```

```
const div2=React.createElement("div",null,[
  React.createElement("h1",null,"LOGO") ,
  React.createElement("ul",null,[
    React.createElement("li",null,"about us"),
    React.createElement("li",null,"contact us")

  ])
])






const root = ReactDOM.createRoot(rootElement);

console.log("hi is the new keywor");
console.log("hello is the new keword");
root.render(div2);
```

so it is very painful in creating in react it is messy

so there is another way to create it

```
import {createElement as ce} from "react";
import ReactDOM from "react-dom/client";

/*
const heading=ce("h1",{
    id:"h1_el",
    hello:"world",
    key:"heading_1"
},"h1_heading");

const heading2=ce("h2",{key:"heading_2"},"this is h2 element");

const div=ce("div",null,[heading,heading2]);
*/
const rootElement = document.getElementById('root');







//------------------

const div2=ce("div",null,[
  ce("h1",null,"LOGO") ,
  ce("ul",null,[
    ce("li",null,"about us"),
    ce("li",null,"contact us")

  ])
])





const root = ReactDOM.createRoot(rootElement);

console.log("hi is the new keywor");
console.log("hello is the new keword");
root.render(div2);
```

but it also looks messy

so then JSX created

JSX (JavaScript XML) is an extension syntax for JavaScript that allows you to write HTML-like code directly within your JavaScript code in React applications. It provides a way to describe the structure and content of components in a more declarative and intuitive manner.

JSX is html inside javascript ⇒ true/false ans is false;

jsx is html like code inside javascript ⇒ this is true

JSX ⇒ javscript XML (not this fullform mentioned in docs)

**JSX is an extension of the JavaScript language based on ES6.**

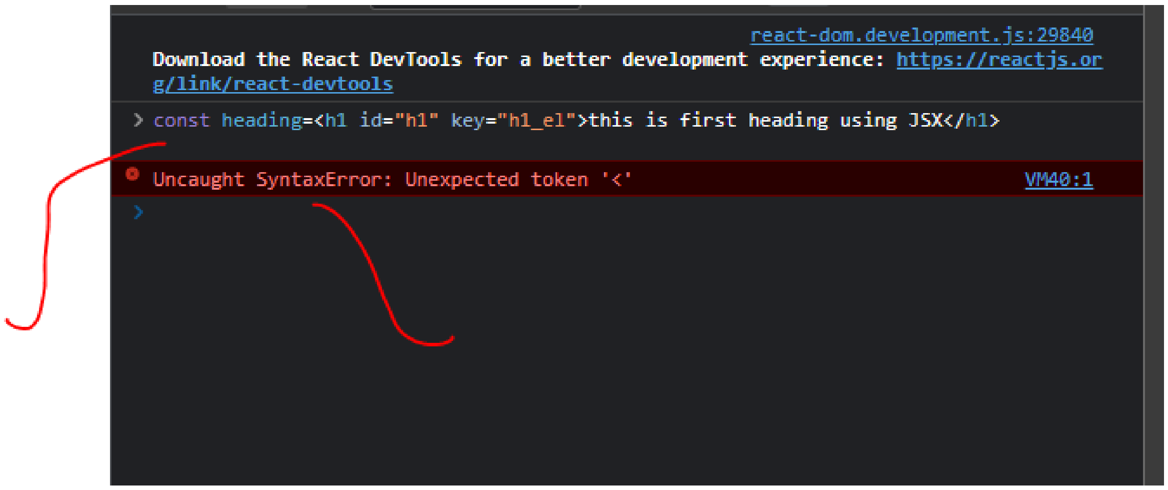
example⇒ **const heading=<h1 id="h1" key="h1_el">this is first heading using JSX</h1>**

JSX is created  by meta developers

HOMEWORK⇒ diff between JSX AND HTML

in order to give class inside JSX we use className but in react we use class

if we try to run JSX inside browser it will give error



```
                                                         react-dom.development.js:29840
Download the React DevTools for a better development experience: https://reactjs.or
g/link/react-devtools
> const heading=<h1 id="h1" key="h1_el">this is first heading using JSX</h1>
Uncaught SyntaxError: Unexpected token '<'                               VM40:1
>
```

so who understand This jsx code ⇒ babel

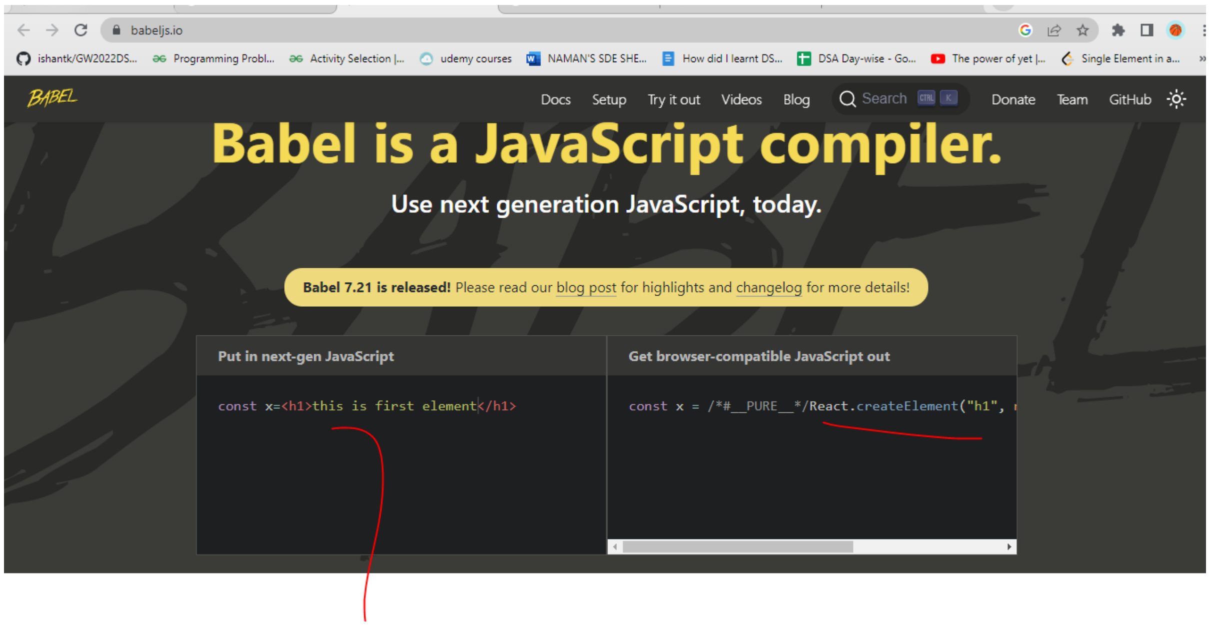
jsx first converted into  react.createElement to create element and than convert that object into html and that html eleemnt is put inside DOM

jsx converted into react.createElement using bable

babel first convert jsx code into react code and than it gives object and object than converted into html and this html is put inside browser using reactDOM

lets see proof is babel convert js code into react code

**https://babeljs.io/**



Babel is a JavaScript compiler.

Use next generation JavaScript, today.

**Babel 7.21 is released!** Please read our blog post for highlights and changelog for more details!

Put in next-gen JavaScript

```
const x=<h1>this is first element</h1>
```

Get browser-compatible JavaScript out

```
const x = /*#__PURE__*/React.createElement("h1",
```

babel also provide another functionality also

advantages of JSX⇒ less code, readiblity etc

we don’t install babel but it comes with parcel as transitive dependency

node modules also contain package.lock.json file to maintain versions and another information of different packages inside node modules

we dont push node_modules into github but how our code will work when pull by another

person its because when we run code again node modules will created again so due to that our code will work properly

THIS BELOW IS JSX EXPRESSION

**(<h1 id="h1" key="h1_el">this is first heading using JSX</h1>);**

this below is react element

const heading=<h1 id="h1" key="h1_el">this is first heading using JSX</h1>

and we can say js object is a react element

lets understand react component

Components are independent and reusable bits of code. They serve the same purpose as JavaScript functions, but work in isolation and return HTML.

Components are of two types⇒

Functional based components ⇒ NEW METHOD

class based components ⇒ OLD METHOD

Functional Components:
Functional components are defined as JavaScript functions that return a React element/ components .

```
const Heading=()=>{
  return (<h1 id="h1" key="h1_el">this is first heading using JSX</h1>);
}
// it is good practice to use captial letter in start of functional component
```

when we want to return multiple things like div, h2 than we need to wrap inside paranthsis

```
const Heading=()=>{
  return (
    <div>
      <h1 id="h1" key="h1_el">this is first heading using JSX</h1>
      <h1 id="h1" key="h1_el_s">this is second heading using JSX</h1>
      </div>
  );
}
```

we can also skip return from this functional component means return hta sakte hai

```
import {createElement as ce} from "react";
import ReactDOM from "react-dom/client";

/*
const heading=ce("h1",{
    id:"h1_el",
    hello:"world",
    key:"heading_1"
},"h1_heading");

const heading2=ce("h2",{key:"heading_2"},"this is h2 element");

const div=ce("div",null,[heading,heading2]);
*/
const rootElement = document.getElementById('root');






//------------------
/*
const div2=ce("div",null,[
  ce("h1",null,"LOGO") ,
  ce("ul",null,[
    ce("li",null,"about us"),
    ce("li",null,"contact us")

  ])
])
*/

const Heading=()=>{
  return (
    <div>
      <h1 id="h1" key="h1_el">this is first heading using JSX</h1>
      <h1 id="h1" key="h1_el_s">this is second heading using JSX</h1>
      </div>
  );
}

console.log("heading is ",Heading," type is ",typeof(Heading));




const root = ReactDOM.createRoot(rootElement);

console.log("hi is the new keywor");
console.log("hello is the new keword");
root.render(<Heading/>);
```

## putting react element inside react component

```
import {createElement as ce} from "react";
import ReactDOM from "react-dom/client";

/*
const heading=ce("h1",{
    id:"h1_el",
    hello:"world",
    key:"heading_1"
},"h1_heading");

const heading2=ce("h2",{key:"heading_2"},"this is h2 element");

const div=ce("div",null,[heading,heading2]);
*/
const rootElement = document.getElementById('root');







//------------------
/*
const div2=ce("div",null,[
  ce("h1",null,"LOGO") ,
  ce("ul",null,[
    ce("li",null,"about us"),
    ce("li",null,"contact us")

  ])
])
*/

const hello_word=(<h1>this is hello world</h1>);

const Heading=()=>{
  return (
    <div>
      {hello_word}
      <h1 id="h1" key="h1_el">this is first heading using JSX</h1>
      <h1 id="h1" key="h1_el_s">this is second heading using JSX</h1>
      </div>
  );
}
```

```
console.log("heading is ",Heading," type is ",typeof(Heading));




const root = ReactDOM.createRoot(rootElement);

console.log("hi is the new keywor");
console.log("hello is the new keword");
root.render(<Heading/>);
```

putting one component into another

```
import {createElement as ce} from "react";
import ReactDOM from "react-dom/client";

/*
const heading=ce("h1",{
    id:"h1_el",
    hello:"world",
    key:"heading_1"
},"h1_heading");

const heading2=ce("h2",{key:"heading_2"},"this is h2 element");

const div=ce("div",null,[heading,heading2]);
*/
const rootElement = document.getElementById('root');






//------------------
/*
const div2=ce("div",null,[
  ce("h1",null,"LOGO") ,
  ce("ul",null,[
    ce("li",null,"about us"),
    ce("li",null,"contact us")

  ])
])
*/

const Hello_word=()=>{(<h1>this is hello world</h1>);}


const Heading=()=>{
  return (
    <div>
      <Hello_word/>
      <h1 id="h1" key="h1_el">this is first heading using JSX</h1>
      <h1 id="h1" key="h1_el_s">this is second heading using JSX</h1>
      </div>
  );
}

console.log("heading is ",Heading," type is ",typeof(Heading));




const root = ReactDOM.createRoot(rootElement);

console.log("hi is the new keywor");
console.log("hello is the new keword");
root.render(<Heading/>);
```

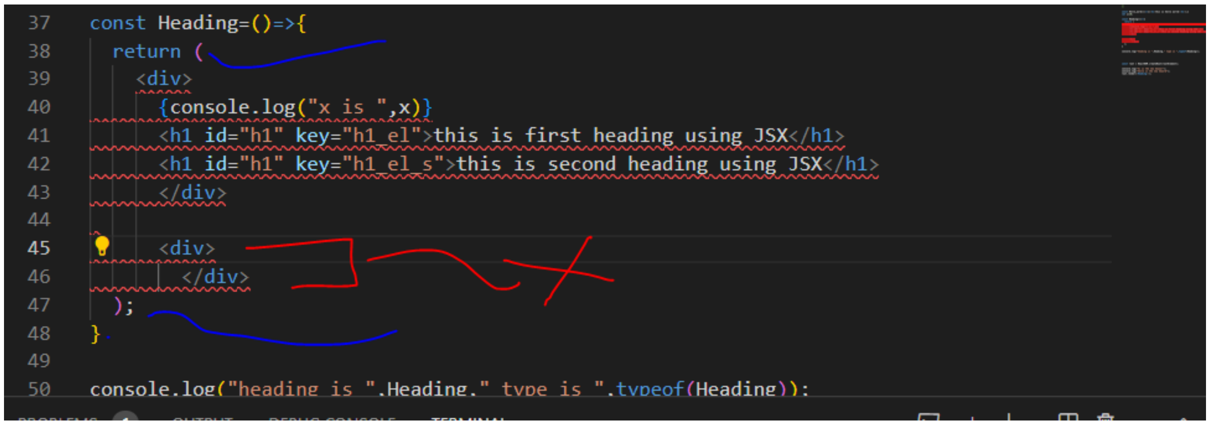
this above is called component compositon

In this approach, each component is designed to perform a specific function or display a specific piece of content, and can be combined with other components to build a more sophisticated interface.

instead of **<Hello_word/>** we can also use **{Hello_world()}**

```
import {createElement as ce} from "react";
import ReactDOM from "react-dom/client";

/*
const heading=ce("h1",{
    id:"h1_el",
    hello:"world",
    key:"heading_1"
},"h1_heading");

const heading2=ce("h2",{key:"heading_2"},"this is h2 element");

const div=ce("div",null,[heading,heading2]);
*/
const rootElement = document.getElementById('root');
```

```
//------------------
/*
const div2=ce("div",null,[
  ce("h1",null,"LOGO") ,
  ce("ul",null,[
    ce("li",null,"about us"),
    ce("li",null,"contact us")

  ])
])
*/

const Hello_word=()=>{(<h1>this is hello world</h1>);}
let x=10;

const Heading=()=>{
  return (
    <div>
      {console.log("x is ",x)}
      <h1 id="h1" key="h1_el">this is first heading using JSX</h1>
      <h1 id="h1" key="h1_el_s">this is second heading using JSX</h1>
      </div>
  );
}

console.log("heading is ",Heading," type is ",typeof(Heading));




const root = ReactDOM.createRoot(rootElement);

console.log("hi is the new keywor");
console.log("hello is the new keword");
root.render(<Heading/>);
```



```
const Heading=()=>{
  return (
    <div>
      {console.log("x is ",x)}
      <h1 id="h1" key="h1_el">this is first heading using JSX</h1>
      <h1 id="h1" key="h1_el_s">this is second heading using JSX</h1>
      </div>

      <div>
        </div>
  );
}

console.log("heading is ",Heading," type is ",typeof(Heading));
```

inside return() there is only parent element

2:35:05

import React from “react”;// we are importing the React object from react library which is present inside node modules

jsx will not work parcel is not installed becuase babel is not installed

if babel is installed and parcel is not installed than jsx will work properly

**<Title/> it is a good way to calll functinational component**

functional components behaves same as js functions

if we use createReactapp webpack comes along with and babel also come along with

why react is faster?

key is mandotaory when we have child siblings

browser will not understnad code of jsx, react it only understand html code

so thats why react code is converted to html using reactDom