Laying the Foundation! (Namaste-React)

? Please make sure to follow along with the whole ***"Namaste React"*** series, starting from Episode-1 and continuing through each subsequent episode. The notes are designed to provide detailed explanations of each concept along with examples to ensure thorough understanding. Each episode builds upon the knowledge gained from the previous ones, so starting from the beginning will give you a comprehensive understanding of React development.

? I've got a quick tip for you. To get the most out of these notes, it's a good idea to watch **Episode-2** first. Understanding what ***"Akshay"*** shares in the video will make these notes way easier to understand.

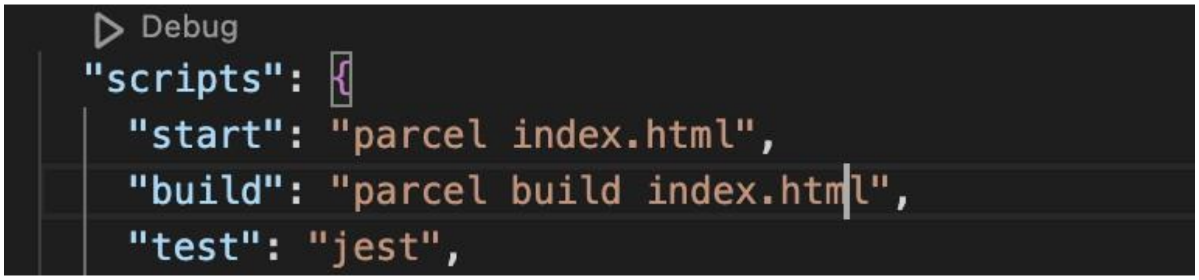
> **So far, here's what we've learned in the previous episode**
>
> - We learned that npm is anything but not node package manager and what is npx.
> - We included node-modules and React in our project.
> - We got to know the difference between package.json and package-lock.json. We also explored the concept of bundlers.
> - We learned how to start our app.
> - Don't forget "Parcel is a Beast".

.

Part-1

Q) What is another way of starting the build of the project?

- We will be creating scripts instead of using "*npx parcel index.html*". We can create different scripts for starting our project in Development and Production.
- In `package.json`, in the script section write the following command.



```
"scripts": {
  "start": "parcel index.html",
  "build": "parcel build index.html",
  "test": "jest",
```

- To run these scripts, enter the following commands in the terminal,

To start:

```
npm run start
```

or

```
npm start
```

For Production Build:

```
npm run build
```

If you're not sure how to start the project in a new company then find these scripts in `package.json` and use them.

Part-2

Revision of previous Episodes

Part-3

Introducing JSX.

Before we begin, we have to remove the existing React Code from `App.js` where we used *React.createElement()* for displaying content on the webpage but its syntax is very bad. It’s not developerfriendly, and very hard to read. To solve this problem Facebook developers built JSX.

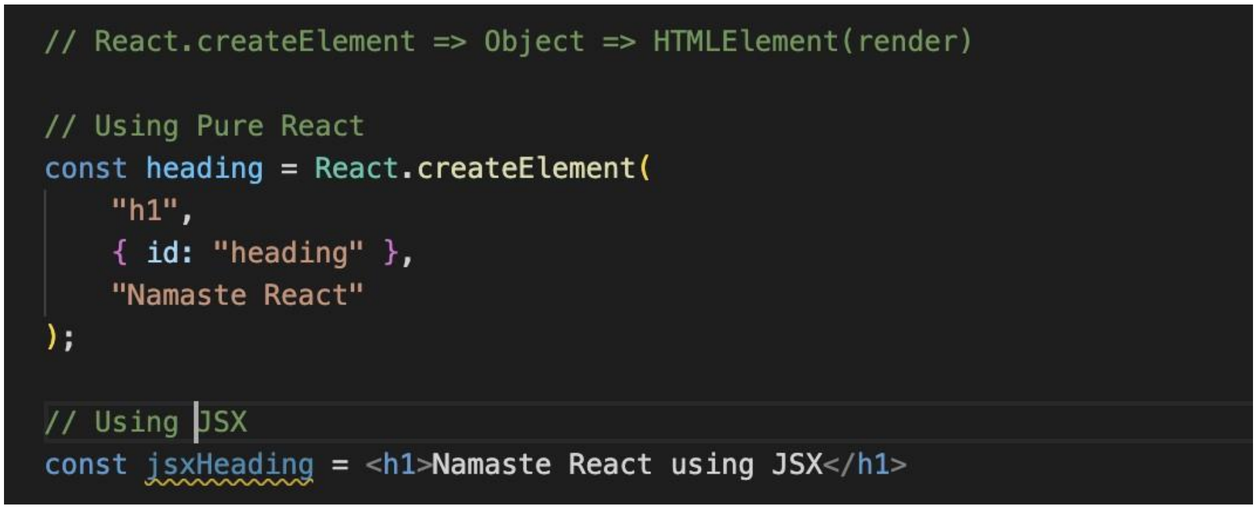
JSX makes developer life easy as we no longer have to write our code using *React.createElement()*

NOTE: We write code for both Machines and Humans but first for Human understanding as it is read by a lot of developers

Q) What is JSX?

JSX is HTML-like or XML-like syntax. JSX stands for JavaScript XML. It's a syntax extension for JavaScript.

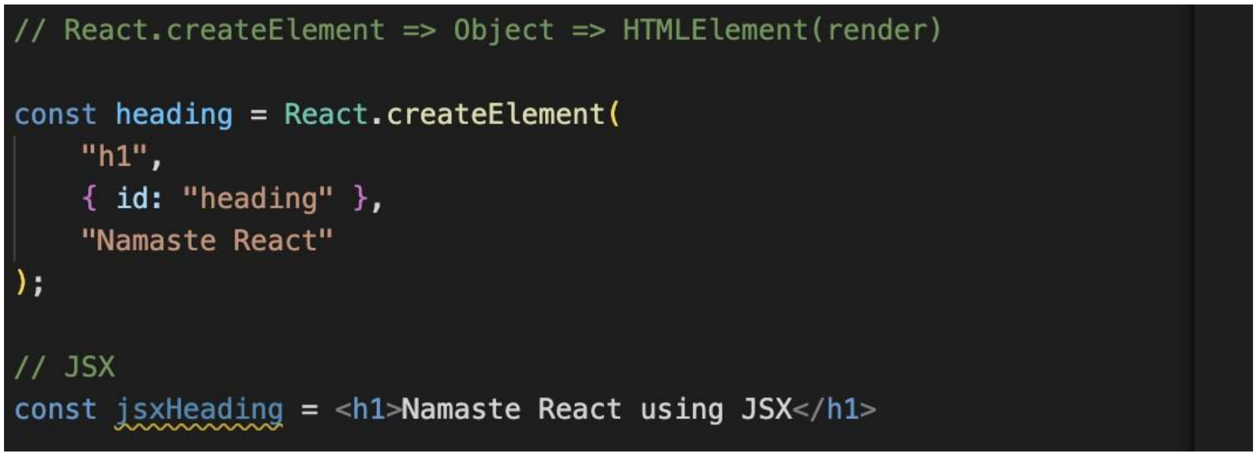
- It is not a part of React. React apps can be built even without JSX but the code will become very hard to read.
- It is not HTML inside JavaScript.
- JavaScript engine cannot understand JSX as it only understands ECMAScript



```
// React.createElement => Object => HTMLElement(render)

// Using Pure React
const heading = React.createElement(
    "h1",
    { id: "heading" },
    "Namaste React"
);

// Using JSX
const jsxHeading = <h1>Namaste React using JSX</h1>
```



```
// React.createElement => Object => HTMLElement(render)

const heading = React.createElement(
    "h1",
    { id: "heading" },
    "Namaste React"
);

// JSX
const jsxHeading = <h1>Namaste React using JSX</h1>
```

When we log *heading* and *jsxHeading*, it gives the same object.

From this point, we will not be using *React.createElement()*

Introducing Babel

Q) Is JSX a valid JavaScript?

The answer is yes and no.

- JSX is not a valid JavaScript syntax as it's not pure HTML or pure JavaScript for a browser to understand. JS does not have built-in JSX. The JS engine does not understand JSX because the JS engine understands ECMAScript or ES6+ code

Q) If the browser can't understand JSX how is it still working?
This is because of Parcel because "*Parcel is a Beast*".
Before the code gets to JS Engine it is sent to Parcel and Transpiled there. Then after transpilation, the browser gets the code that it can understand.
Transpilation ⍰ Converting the code in such a format that the browsers can understand.
Parcel is like a manager who gives the responsibility of transpilation to a package called Babel.
Babel is a package that is a compiler/transpiler of JavaScript that is already present inside 'node-modules'. It takes JSX and converts it into the code that browsers understand, as soon as we write it and save the file. It is not created by Facebook. Learn more about Babel on babeljs.io
JSX (transpiled by Babel) ⍰ React.createElement ⍰ ReactElement ⍰ JS Object ⍰ HTML Element(render)

Q) What is the difference between HTML and JSX?

JSX is not HTML. It's HTML-like syntax.

- HTML uses 'class' property whereas JSX uses 'className' property
- HTML can use hypens in property names whereas JSX uses camelCase syntax.

Single Line and Multi Line JSX Code

Single line code:

```
const jsxHeading = <h1>Namaste React</h1>
```

Multi-line code:

If writing JSX in multiple lines then using '()' parenthesis is mandatory. To tell Babel from where JSX is starting and ending.

```
const jsxHeading = (
  <div>
   <h1>Namaste React</h1>
  </div>
)
```

NOTE:

1) Use "Prettier - Code Formatter" VS Code Extension to make your code look beautiful with proper formatting
2) Use "ES lint" VS Code Extension for linting
3) Use "Better Comments" VS Code Extension to beautify your comments

Code all of these things discussed until now for better understanding.

Part-4

Introducing React Components

Everything inside React is a component.

Q) What are Components?

There are 2 types of components:

1.Class-based Components - Old way of writing code, used rarely in industry

2.Functional Components - New way of writing code, most commonly used

Q) What is a React Functional Components?

It is just a JavaScript Function that returns some JSX or a react element.

Always name React Functional Component with Capital Letters otherwise you will confuse it with normal function

We can write all theses ways-

```
// All are the same for single-line code
const HeadingComponent1 = () => (
  <h1>Namaste</h1>
)

const HeadingComponent2 = () => {
 return <h1>Namaste</h1>
}

const HeadingComponent3 = () => <h1>Namaste</h1>
```

To render a functional component, we call them '<Heading1 />'. This is the syntax that Babel understands. You

can also call them using these ways,

'<Title></Title>' or

'{Title ()}'

Components Composition A

component inside a component.

Calling a component inside another component is Component Composition.

```
const Title = () => <h1>Namaste React</h1>

const HeadingComponent = () => (
 <div id="container">
  <Title />
 </div>
)
```

Code inside the 'Title' component will be used inside the 'HeadingComponent' component as the 'Title' component is called inside it. It will become something like this,

```
const HeadingComponent = () => (
 <div id="container">
   <h1>Namaste React</h1>
 </div>
)
```

Part-5

Q) How to use JavaScript code inside JSX?

Inside a React Component when '{}' parenthesis is present we can write any JavaScript expression inside it.

```
const number = 10000;

const HeadingComponent = () => (
 <div id="containter">
   {number}
   <h1>Namaste React</h1>
 </div>
)
```

Q) How to call React Element in JSX?

We can use '{}' parenthesis.

```
const elem = <span> React Element </span>

const HeadingComponent = () => (
 <div id="containter">
   {elem}
   <h1>This is Namaste React</h1>
 </div>
)
```

Q) What will happen if we call 2 elements inside each other?

If we put 2 components inside each other, then it will go into an infinite loop and the stack will overflow. It will freeze your browser, so it's not recommended to do so.

<u>**Advantages of using JSX.**</u>

1) Sanitizes the data

If someone gets access to your JS code and sends some malicious data which will then get displayed on the screen, that attack is called cross-site scripting.

It can read cookies, local storage, session storage, get cookies, get info about your device, and read data. JSx takes care of your data.

If some API passes some malicious data JSX will escape it. It prevents cross-site scripting and sanitizes the data before rendering

2) Makes code readable

JSX makes it easier to write code as we are no longer creating elements using React.createElement()

3) Makes code simple and elegant

4) Show more useful errors and warnings

5) `JSX prevents code injections (attacks)`

Self-Notes-

Chapter 03 - Laying the Foundation

How to Run the project-

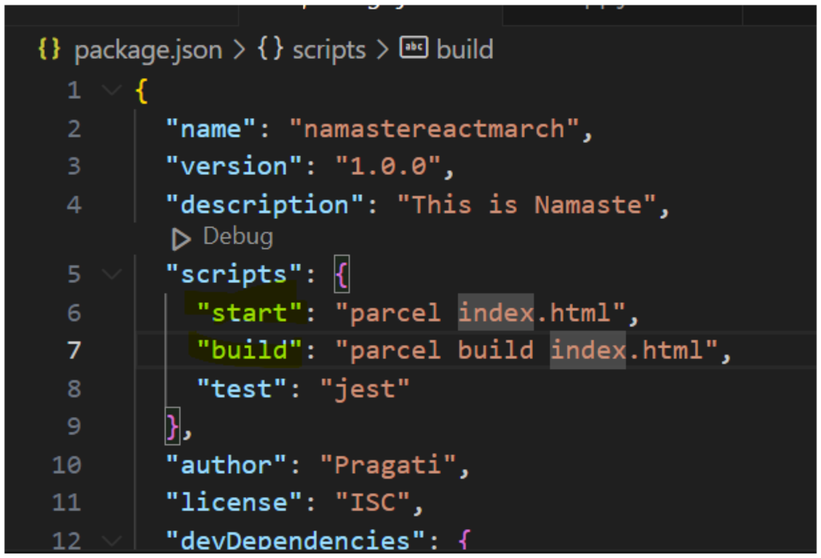
Npx parcel index.html – this will create the development built for us and it will host it up on localhost 1234

This command means that we are executing , npx means you are expecting a npm package, parcel and you give a source file as index.html and these command are diff. in development build and the production build.

To make our life easier, what we do, we create a script that will build our project instead of writing this command again and again.

Let us create the script and then we will not use npx parcel index.html to build the project, but we will then use npm scripts.

Create the script for dev and production.

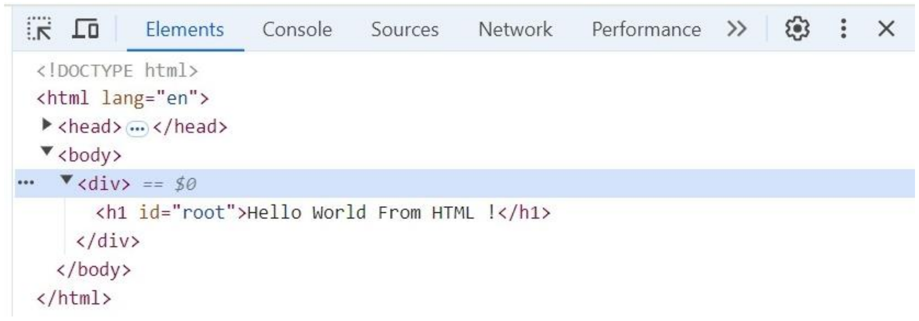


```json
{
  "name": "namastereactmarch",
  "version": "1.0.0",
  "description": "This is Namaste",
  "scripts": {
    "start": "parcel index.html",
    "build": "parcel build index.html",
    "test": "jest"
  },
  "author": "Pragati",
  "license": "ISC",
  "devDependencies": {
```

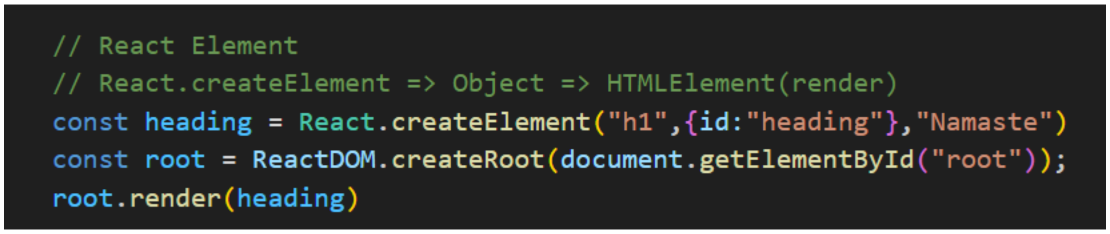
"start":" parcel index.html", // script create for dev build.
"build": "parcel build index.html", //script created building in production.

Now we run the application in development mode use npm run start or npm start command. If I have to build my project use npm run build.
And npm run start behind the scenes execute the package parcel with the index.html
We are doing the same things with npx or npm and now we are doing the same things with scripts.
Because we configure this inside our package.json.

React element- just like the DOM element or we can say that equivalent to the DOM elements.
DOM element are the html element -



```
<!DOCTYPE html>
<html lang="en">
  <head>…</head>
  <body>
    <div> == $0
      <h1 id="root">Hello World From HTML !</h1>
    </div>
  </body>
</html>
```



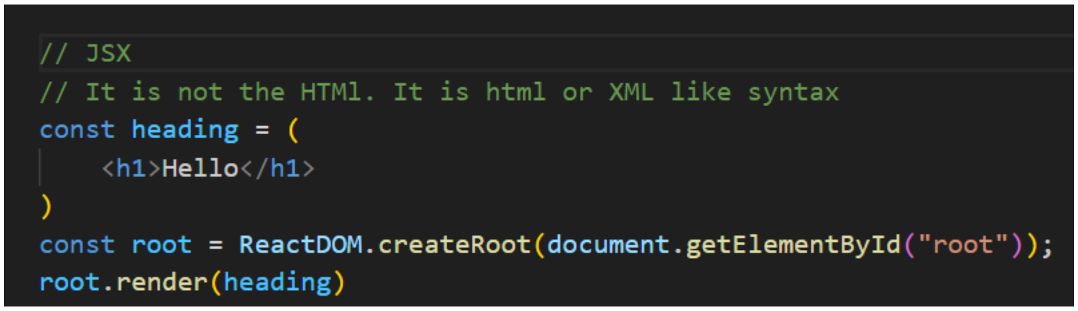
```
// React Element
// React.createElement => Object => HTMLElement(render)
const heading = React.createElement("h1",{id:"heading"},"Namaste")
const root = ReactDOM.createRoot(document.getElementById("root"));
root.render(heading)
```

React.createElement basically create object and when we do (root.render)render this element into DOM it become HTML and push it into the browser. Whatever happen in the react it will happen inside the root.

- It will be replaced whatever is written into <div id="root">
- If there something is not displaced that means there is something is not displayed on browser that means there is some problem with render.

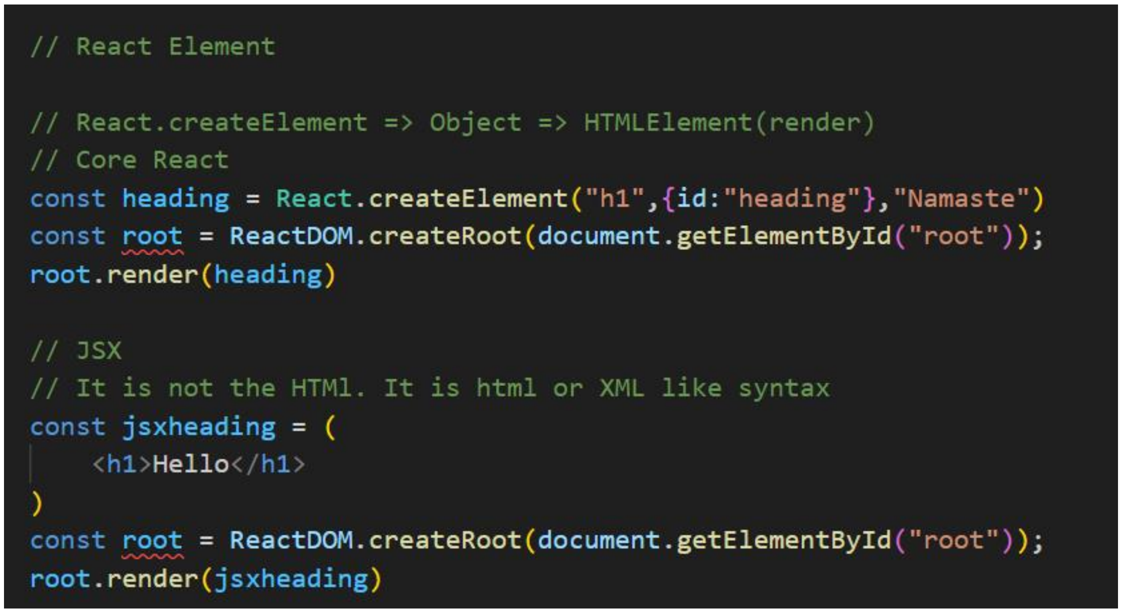
- JSX is not html inside JavaScript. It is html like syntax, jsx is just the syntax.
-Whatever we are written inside jsx it will become the react element and react elements is the object.

But it is not good way to write the coder it is not the development friendly; it is hard to read. Suppose if create the nested structure it become very clumsy. So, the Facebook developer created JSX.



```
// JSX
// It is not the HTMl. It is html or XML like syntax
const heading = (
    <h1>Hello</h1>
)
const root = ReactDOM.createRoot(document.getElementById("root"));
root.render(heading)
```

JSX is the JavaScript syntax, which is easier to create react elements.
Lot of developer think jsx is the part of react. But react is diff. and JSX is different.
We can write react without jsx. jsx makes our developers life easy.
Jsx is a convention where we kind merge these html and js to things up, we can say that we can write the html, js together and It is the HTML like syntax.
First code is written in core react and second one is written in JSX.

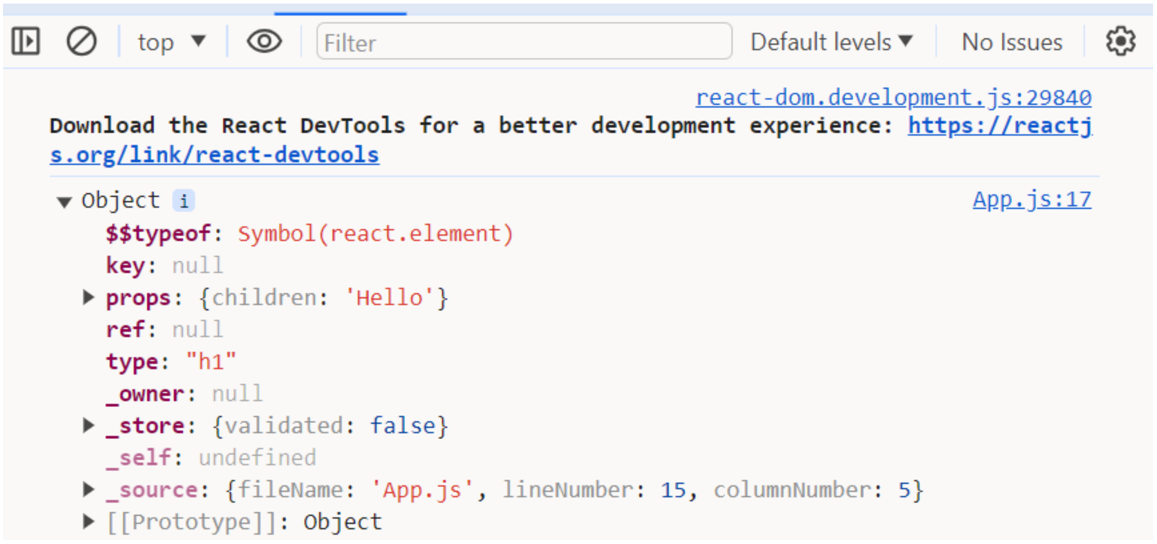


```
// React Element

// React.createElement => Object => HTMLElement(render)
// Core React
const heading = React.createElement("h1",{id:"heading"},"Namaste")
const root = ReactDOM.createRoot(document.getElementById("root"));
root.render(heading)

// JSX
// It is not the HTMl. It is html or XML like syntax
const jsxheading = (
    <h1>Hello</h1>
)
const root = ReactDOM.createRoot(document.getElementById("root"));
root.render(jsxheading)
```

When I console.log it will return the object



```
                                                    react-dom.development.js:29840
Download the React DevTools for a better development experience: https://reactj
s.org/link/react-devtools
▼ Object i                                                              App.js:17
    $$typeof: Symbol(react.element)
    key: null
  ▶ props: {children: 'Hello'}
    ref: null
    type: "h1"
    _owner: null
  ▶ _store: {validated: false}
    _self: undefined
  ▶ _source: {fileName: 'App.js', lineNumber: 15, columnNumber: 5}
  ▶ [[Prototype]]: Object
```

Que- when we write the code, we write for the machine or the humans.
We write code for both machine as well as humans. But you are writing code first for humans then for machine. When you write the code.

Can browser understand JSX, If the browser can't understand JSX how is it still working?
JavaScript engine cannot understand JSX as it only understands ECMAScript (pure JavaScript), Parcel is doing job behind the scenes, This code is developed by us even before this code goes to the JS engine, It is Transpiled before it goes to the JS engine and then JS engine receives the code that browser can understand.

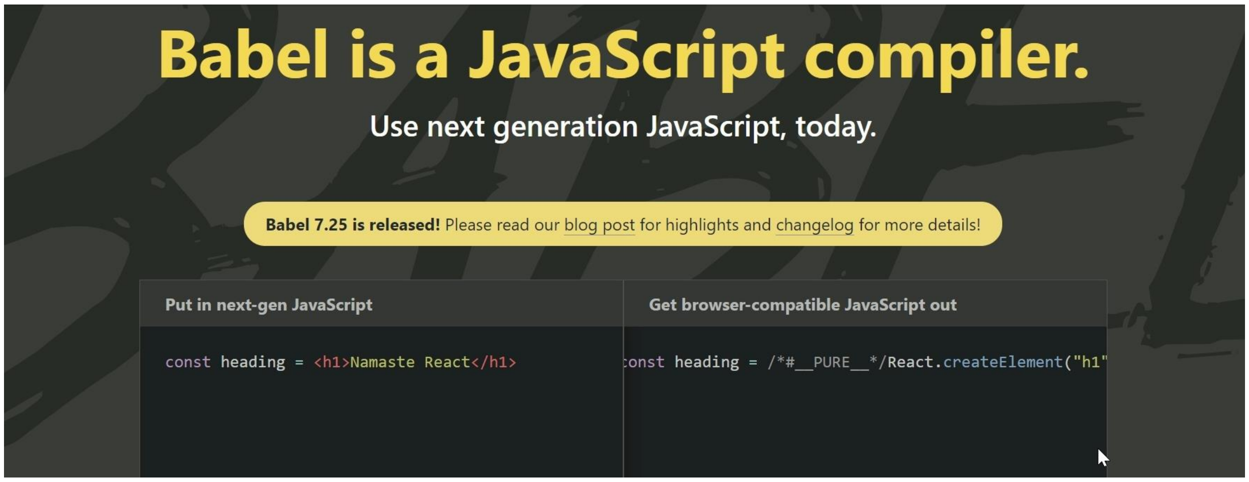
Transpiled means this code is converted to the code that browser can understand, that react can understand.

Basically, we are passing root dot render(root.render(parent)). root dot render understands jsx.
And who's is transpired – parcel doing this job, because parcel is the manager. parcel gives the responsibility to this transpilation to a package which is known as Babel.
Parcel install babel. Babel has converted this code quickly to a code that React will understand. Babel is transpiling the code.

Babel it takes your JSX and converts it into the code that browser understands.

How this code running –
JSX code is transpiled to react.create element and then it is converted to make js element and js object render to the HTML element then it will showing to the browser.

Babel is a transpiler and compiler, when some older browsers do not understand ES6, the newer version of JS code, Babel transpiles that ES6 code to a code that older browser understand. It basically converts one code to another.
Babel is the JS code, Js library, it is the node library, npm package, which takes a piece of code, reads that code and converts that code to something else. It converted one code to the another code.


Babel is a JavaScript compiler.
Use next generation JavaScript, today.
Babel 7.25 is released! Please read our blog post for highlights and changelog for more details!
Put in next-gen JavaScript
Get browser-compatible JavaScript out

```
const heading = <h1>Namaste React</h1>
```

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

When you write JSX in multiple lines, we have to wrap it inside parenthesis, Why?- Because babel needs to understand where is the jsx started and where is the jsx ending so it is required. But if we write in the single line no need to wrap.

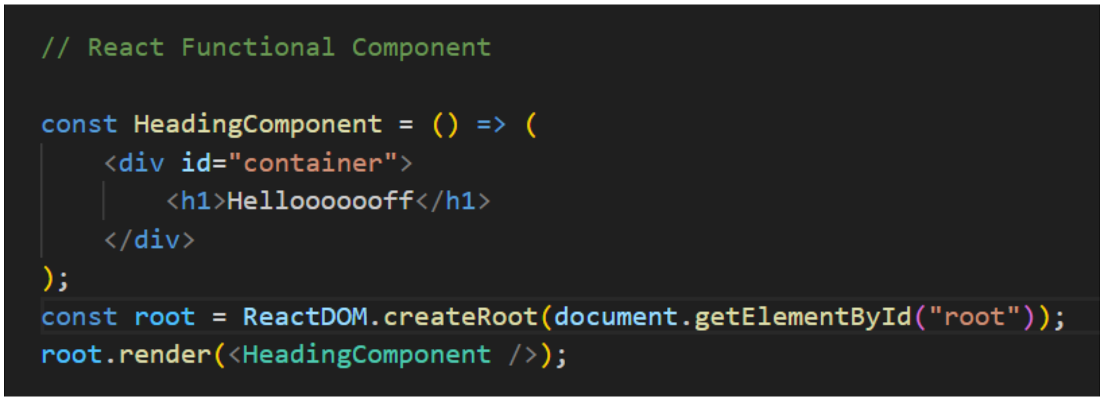
There are of 2 types of components –

Class – Old way of writing the code

Function – New way of writing the code.

- React functional component is the normal Javascript like function.
- Functional component is a function that return some piece of jsx code.

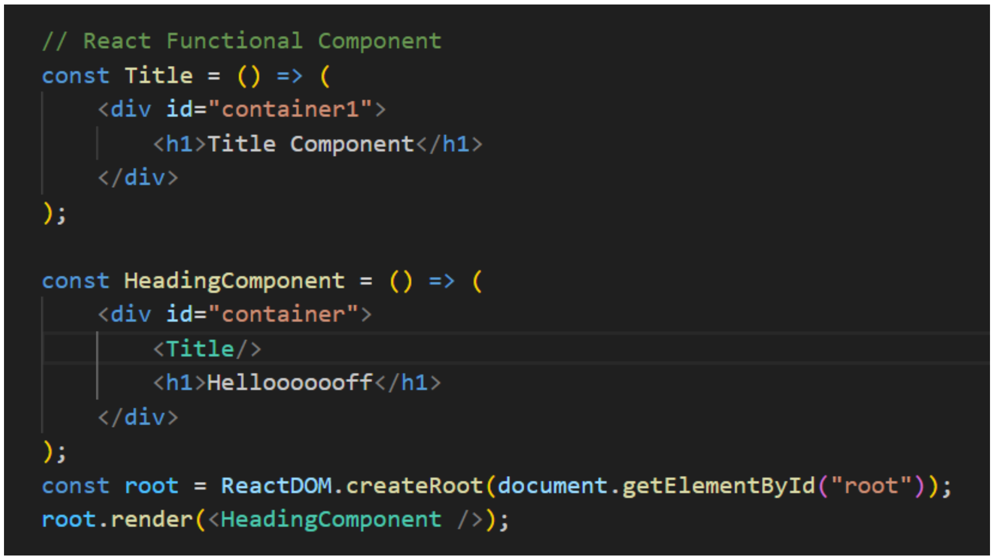
<u>Functional Component-</u>



```
// React Functional Component

const HeadingComponent = () => (
    <div id="container">
        <h1>Hellooooooff</h1>
    </div>
);
const root = ReactDOM.createRoot(document.getElementById("root"));
root.render(<HeadingComponent />);
```

<u>Put one component to another component -</u>



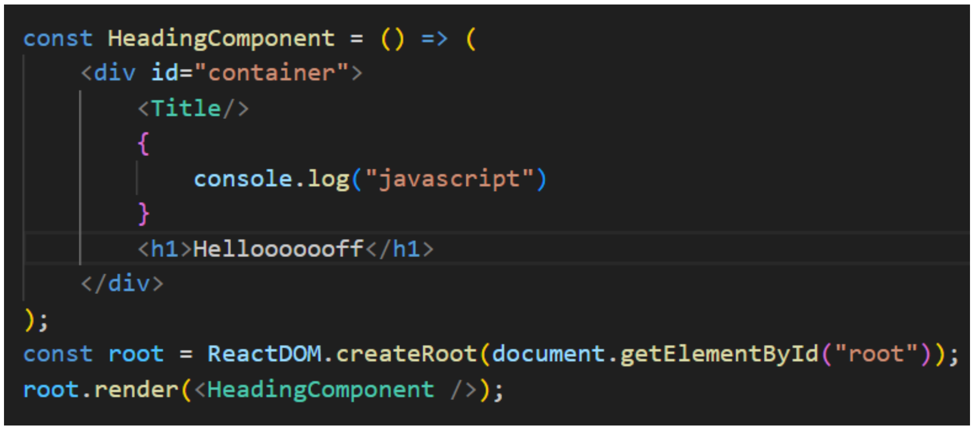
```jsx
// React Functional Component
const Title = () => (
    <div id="container1">
        <h1>Title Component</h1>
    </div>
);

const HeadingComponent = () => (
    <div id="container">
        <Title/>
        <h1>Helloooooff</h1>
    </div>
);
const root = ReactDOM.createRoot(document.getElementById("root"));
root.render(<HeadingComponent />);
```

This root.render is converting everything to html and the browser is rendering it do you.

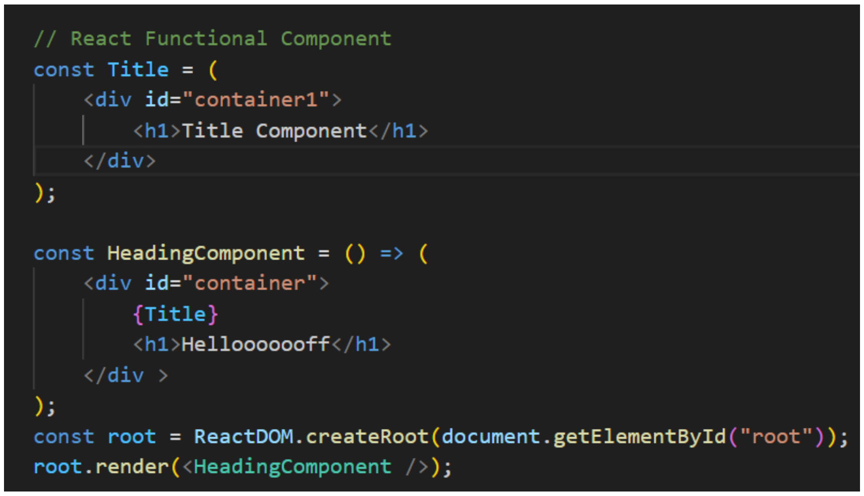
Component composition- component inside to another component.

Inside of the {} of the jsx we can write any piece of JavaScript-



```jsx
const HeadingComponent = () => (
    <div id="container">
        <Title/>
        {
            console.log("javascript")
        }
        <h1>Helloooooff</h1>
    </div>
);
const root = ReactDOM.createRoot(document.getElementById("root"));
root.render(<HeadingComponent />);
```

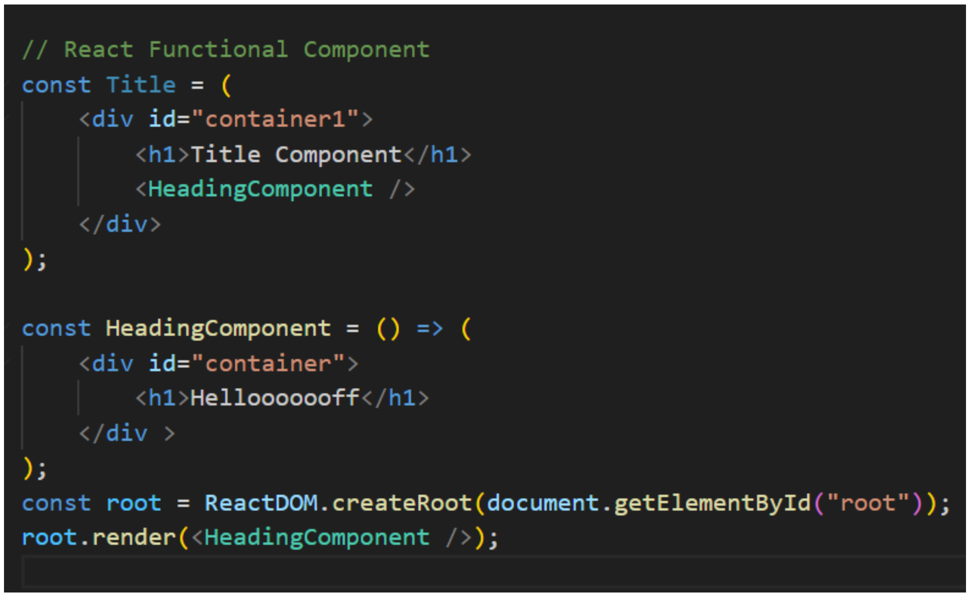
Put react element in react component-



```jsx
// React Functional Component
const Title = (
    <div id="container1">
        <h1>Title Component</h1>
    </div>
);

const HeadingComponent = () => (
    <div id="container">
        {Title}
        <h1>Helloooooff</h1>
    </div >
);
const root = ReactDOM.createRoot(document.getElementById("root"));
root.render(<HeadingComponent />);
```

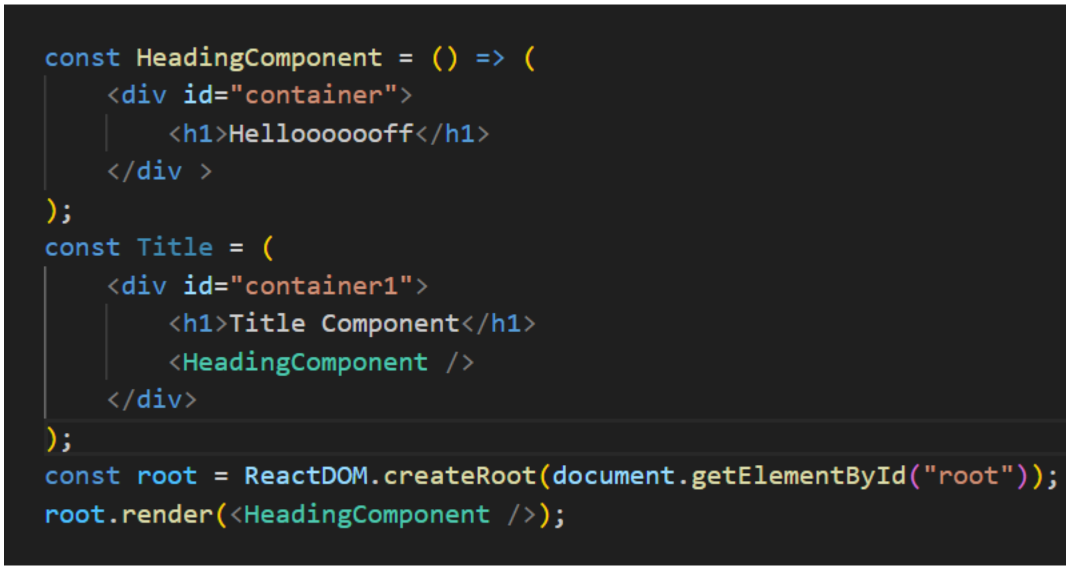
Exception-



```
// React Functional Component
const Title = (
    <div id="container1">
        <h1>Title Component</h1>
        <HeadingComponent />
    </div>
);

const HeadingComponent = () => (
    <div id="container">
        <h1>Hellooooooff</h1>
    </div >
);
const root = ReactDOM.createRoot(document.getElementById("root"));
root.render(<HeadingComponent />);
```

We will get the error, we need to change the order then we will get the output



```
const HeadingComponent = () => (
    <div id="container">
        <h1>Hellooooooff</h1>
    </div >
);
const Title = (
    <div id="container1">
        <h1>Title Component</h1>
        <HeadingComponent />
    </div>
);
const root = ReactDOM.createRoot(document.getElementById("root"));
root.render(<HeadingComponent />);
```

Suppose the data coming from Api, what if the data have some malicious data and the api send some malicious data, basically api gave some result and the api is the bad api, and attacker run some JavaScript code in the browser, this type of attacking know as cross side scripting. Attackers send some data and this data because this is javascript its execute and attacker can do the attack for you, it can steel cookies, if I execute javascript in your browser I can read you local storage, read you cookies, session storage, get the cookie and read that data and get information of your laptop. Can do lots of things if I run JavaScript in your laptop, <u>jsx takes care of these injection attacks.</u>

If these apis send some malicious data into your code jsx escape it. It basically sanitizing the data. the data wrap inside the curly braces, the browser not blindly run it. Jsx sanitize the data that come in and pass it. It prevent from cross site scripting attacks for you. This is how powerful jsx is. Don't have to take care about attackes. Feel free to use it.

Topics

- JSX
- React.createElement vs JSX
- Benefits of JSX
- Behind the Scenes of JSX
- Babel & parcel role in JSX
- Components
- Functional Components

`Composing Components`

Assignment

- What is JSX?
- Superpowers of JSX
- dRole of `type` attribute in script tag? What options can I use there?
- `{TitleComponent}` vs `{<TitleComponent/>}` vs `{<TitleComponent></TitleComponent>}` in JSX

Coding Assignment:

- Create a Nested header Element using React.createElement(h1,h2,h3 inside a div with class "title")
  - Create the same element using JSX
  - Create a functional component of the same with JSX
  - Pass attributes into the tag in JSX
  - Composition of Component(Add a component inside another)
  - `{TitleComponent}` vs `{<TitleComponent/>}` vs `{<TitleComponent></TitleComponent>}` in JSX
- Create a Header Component from scratch using Functional Components with JSX
  - Add a Logo on left
  - Add a search bar in middle
  - Add User icon on right
  - Add CSS to make it look nice

References

- Babel: https://babeljs.io/
- Attribute Type: https://developer.mozilla.org/en-US/docs/Web/HTML/Element/script#attr-type
- JS Modules: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Modules
- Babel Playground: https://babeljs.io/repl#
- React without JSX: https://reactjs.org/docs/react-without-jsx.html