Parcel is a beast.

* It sets up the development environment for developing web applications.
* It makes the developer’s life easy by taking care of the configuration part, so that the developer can focus on coding rather than configuring.
* It also does HMR , which automatically updates the changes you make in the code, without having to reload the entire page.
* It also helps in faster rebuilds through caching.

Parcel is hosting the development build on the localhost:1234

So in this episode, we are trying to create a script to start our application in development mode instead of typing the npx command every time. And if it is an npm script, it has to be done in package.json.  
Cmd:

Previously we have tried to start our application in dev mode using parcel with the command:

npx parcel index.html: This command says that we are executing a npm package called parcel and the starting point of execution is index.html.

Inside the package .json scripts :

“start”: “parcel index.html”,

“build”:”parcel build index.html”

And whenever you want to start your project in dev mode you can type:

Npm run start == npm start  
and for build mode:

Npm run build

Also whenever you go to a company and want to run a project, you can go to package.json and search for the script tag to find the exact command to go with npm run \_\_\_.  
  
In the recap of how to create elements using React:

What we create an element using react.createElement(), the element is actually a JS object and only when we render this object in the DOM is when it becomes an html element.

For easy writing of code:

JSX is HTML like syntax and not html inside Js.

We create a react element using jsx syntax.

JSX allows you to combine **JavaScript** logic and **HTML-like structure** in a single, expressive syntax, making it much easier to build dynamic and interactive user interfaces, particularly in React applications.

So the code we write using jsx is not valid javascript and the js engine inside the browsers can not understand it.

Then comes parcel which has barbel as one of it’s dependencies, that transpiles the JSX into React code which is essentially JS again that the browsers can understand.

Babel is transpiler.

For ex:

JSX code :

Const heading =<h1 id=”heading”>Hello </h1>

Is converted into   
const heading = React.createElement(“h1”,{id””heading”},”hello”);  
which the browsers can understand.

To prove that JSX and HTML are different one way to show the difference in writing attributes like  
in JSX:  
<h1 className=“heading”>Head</h1>

In html it is :  
<h1 classname=”heading”>Head</h1>

Everything in React is a component.

React components are of 2 types:

Class components which uses JS classes- Old way not in use mostly

Functional components which uses JS functions- New way 99% used these days.

React Functional Components:  
It is a Js function that returns a piece of JSX code or some JSX element.

Note: When naming a react component, start with a Capital letter to avoid errors.

EG:  
 const HeadingComponent = () => {

    return <h1 id="Namaste React component!"></h1>

  }

Remember in order to render a react component it must be enclosed within angular self closing braces.

What is a component composition?

Putting or nesting one component(Title) inside another(headingcomponent) like this :  
const Title = () => (

  <h1 className="head">Namaste React!</h1>

)

//React functional component

  const HeadingComponent = () => (

    <div id="container">

    <Title/><h1 className="heading">Namaste React component using js arrow function</h1>

        </div>

  );

To put JS inside the react component use {}.

To put a react element inside a react component still use {}, because at the end of the day the variable is js.

Difference between a react component and a react element :  
 **Component** = A **JS function** that returns JSX (UI logic).

* Example: function Button() { return <button>Click me!</button>; }

 **Element** = The **result** of a component or JSX, which is **what gets rendered** into the actual DOM (UI output).

* Example: <button>Click me!</button>