4. Cross-origin :- purpose of this attribute is to share resources from one domain to another domain, basically it is used to store CROS [CROSS Origin Resource Sharing]

↳ It checks whether it is safe to allow for sharing the resources from other domain.

→ To learn more about different script types go to ~~devd~~ developer.mozill.org (MDN web DOCS)

EPISODE-3

Laying The Foundation

↳ Pollyfill

→ to make older browsers understand our new code, the new code is converted into a older code which browser can understand called pollyfill.

→ babel do this conversion automatically.

E.g:- ES6 is the newer version of JS. If I'm working on 1999 browser, my browser will not understand what is this const, newPromise etc.

So, there is a replacement code for these functionalities which is compatible with older version of browsers.

→ So, this is what happen when we write "browserslist" → our code is converted to older one.

↳ **Babel :-**

- Is a Javascript package / library used to convert code written in newer version of JS into code that can be run in older JS engines.

→ To run our app, command is :—

```
npx parcel index.html
```

↳ It means we are executing a npm package parcel & it give us the source file as index.html.

- We always don't have to write this command. Generally, we build a script inside package.json which runs this command in an easy way.

Scripts:- It is used to make npx commands more smaller or simpler for us to start our project in devlopment mode or in production mode.

package.json

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

Your own shortcut, u can name it anything.

script-name

package name

So, to run the project, I've to use:—

npm run start = npm start = npx parcel index.html

npm run build = npm parcel build index.html

npm buid ⊗ not work

→ It will only work, if "start" & "build" keywords are added into script with their respective commands i.e. parcel index.html & parcel build index.html

```
<html>
 <head>
   ⋮
  <title>
      - - - - -
        </title>
  </head>
   <body> - - - -
      - - - = = =
      - -
    </body>
   </html>
```

These are our dom elements, head, title, body.

DOM elements are nothing but the html elements.

<h1>, <footer>, <article>, <aside>, etc.

Note:- HTML elements are basically everything from start tag to end tag.

Render - means updating something in the DOM.

→ React elements are equivalent to DOM elements.

```
Const heading = React.createElement ("h1", {id:"name"}, "Hello")
```

Tag name

Object Attribute

Children of React Element

→ React elements are just JS objects.
When we render it translated into html element or dom element.

→ React dom is a JS library which basically allows react to ~~in the~~ interact with dom.

→ It is basically used to connect React with dom.

→ Used to manage dom elements.

→ It is used to render or update react elements into dom.

React.createElement() — is creating an object.

— This object is converted into HTML code and puts it upon DOM.

→ If we want to build a big HTML Structure, then using 'createElement()' is not a good solution.

So, there comes introduction of JSX.

Que: We cannot have a single cdn link/file for both react & react-dom.

Because in react native, React 3D or React Art, same react is used as a component library for adding components.

Since, react library is reusable in different places such as react native, react 3D etc.

- That's why we keep react & react dom packages seperated.
- DOM is different for mobile and desktop, that's another reason why reactdom should be a different file.

JSX

→ It is a JS syntax, which is easier to create React Elements.

→ React is different and JSX is different.

→ React elements can also be build without JSX, ~~JSX is~~

JSX makes developer life easy.

* JSX is not 'html inside JS'. It is HTML like syntax, it just looks like html & XML

```
Const heading = (
                    <h1 id = "title">
JSX                    Hello World
expression          </h1>
            );
```

Que: For whom do we write code, for humans or machines?

We write code for both, but we want our code to be understand by any other developer/human who sees our code then for machines.

- If we just want to write code for machine, we should be coding in binary (0 or 1), because machine understand binary.

→ Our JS engine does not understands ~~javascript~~ JSX. but it understands ECMA script i.e. the pure JS.

→ It understands all the versions of ES i.e. Ecmascript.

JSX — is not a valid JS. Our browser uses JS engine, cannot understand this and It will give syntax error.

Que: Then how JSX is working?

Ans- JSX Code in transpiled (i.e. Code is converted into a language which browser understand) before it reaches the JS engine.

- Transpiling is done by "parcel". But parcel is not doing it by itself (it's just like a manager)
- "Babel" (which is one package of parcel) do transpilation.

Q. What is Babel?

It is a JavaScript compiler. It converts the JSX to react Code.

i.e. JSX —Converted to (by babel)→ create.ReactElement —Converted to (by babel)→ React Element

↓

(which is nothing but a JS Object) —Through rendering→ HTML Element

REACT COMPONENT

'Everything is a component in React'

- There are two types of component in React.

(a) Class Based Component - It is the OLD WAY of writing code. { It uses Javascript classes}

(b) Functional Component - NEW WAY of writing code {It uses javascript functions to create component}

→ functional Component:

- It is just a normal JS function which returns some piece of JSX element.

Eg:-

```
Const HeaderComponent = () => {
  return <h1> Helloworld </h1>;
};
```

- for any Component, Name starts with Captial letter

→ How to render functional component?

by writing <HeaderComponent/> (in this way we render functional component)

→ There are two syntax of writing a funtional component :-

Syntax ① -

```
Const HeadingComponent = () => {
return(<h1 className = "heading">
        Namaste React
       </h1>;
)}
```

Syntax ② -

```
Const HeadingComponent = () => {
<h1 className="heading"> Namaste React
  </h1>;
}
```

> **Note:-** If we have to give attribute to "JSX" we have to use CamelCase

i.e. **In html:-**

```
<div id = "root"
      Class = "root"
Hi </div>
```

JSX:-

```
Const heading = <h1 id = "heading"
                 ClassName = "head">
                 Hello
                </h1>
```

→ **Single line $ Multiple line:-**

- Single line when we write Jsx in single line. (valid Jsx)

→ Suppose we have to write Jsx in multiple lines we have to wrap it inside paranthesis i.e [()]

- This is because babel needs to understand where JSX is starting $ ending.

→ Component Composition :

If I have to use a component inside a component. Then, it is called component composition/ Composing Components.

E.g :-

```
Const Title =()⇒{
<h1> HelloWould </h1>
}
```

Is a functional component

```
Const HeaderComponent = () => {
return- (
    <div>
    <Title/>
    <h2> Hello</h2>
    <h2> Kirti </h2>
    </div>
  );
};
```

This is Component composition

instead of this you can write { Title() }

3 ways of Component Composition

1. {Title()}
2. <Title/> → Used generally
3. <Title> <Title/>

* Whenever you write Jsx, you can write any piece of javascript code b/w paranthesis { }. It will work.

* Jsx is very secure
- Jsx makes sure your app is safe
- It does sanitization.