



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.

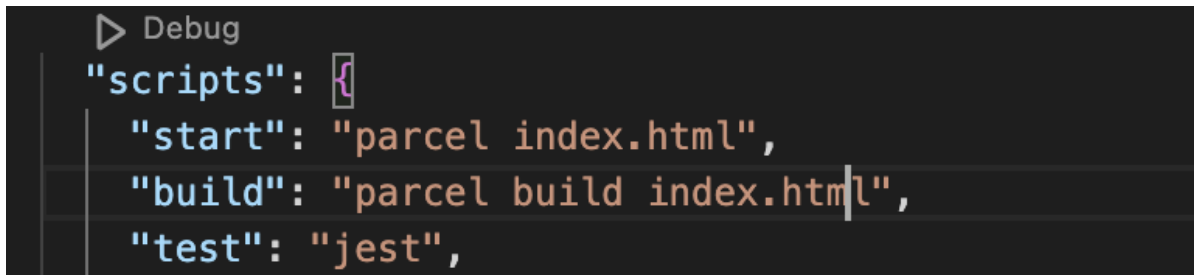
So far, here's what we've learned in the previous episode

- 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.



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

Fig 3.1

- 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 developer-friendly, 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 hyphens 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

```
// 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="container">  
    {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="container">
```

```
    {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.

Advantages of using JSX.

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)