To run the project, execute this in the terminal: `npx parcel index.html`. This will create a development build and host it locally on port 1234.

```
"scripts": {

  "start": "parcel index.html", - used for development
  "build": "parcel build index.html", - used for production
  "test": "jest"
}
```

Yes, npx is used to execute the npm package `parcel`, with the source file given as `index.html`.To simplify our workflow, we can create a script and use that script to run the project.This script should be written in `package.json`.

> That's correct! When you join a company and need to start a project, you can go to the `package.json` file and find the script.
>
> To run the script, you can use the command `npm run <script-name>`.
>
> For starting the project, you can use `npm start`. This is equivalent to running `npm run start`.
>
> If you want to build the project, you can use `npm run build`.
>
> These commands help standardize the development process across different environments and teams. 😊

Part 2

> Absolutely, let's start from scratch.
>
> In `app.js`, we don't have any React code in `index.html`.
>
> React elements are kind of equivalent to DOM elements. Suppose you want to display an `h1` tag with the text "Hi Abhi" in the browser.

You can do this by creating a React element:

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

This basically creates an object. When we render this element onto the DOM, it becomes an HTML element.

How do we render it?

First, you have to create a root inside React:

```
const root = document.getElementById("root");
```

Now, suppose you want to render your heading:

```
const heading = React.createElement("h1",{id:"heading"},"namaste React")

const root = ReactDOM.createRoot(document.getElementById("root"));

root.render(heading);
```

This will render the heading “Namaste React” onto the DOM. 😊

But this is not a good way. The syntax is very bad.

To help react developer community , facebook developers created JSX.

React is different jsx is different. React can be written without using JSX. But JSX will make building react easy.

```
// jsx is not html in javascript. this is html like syntax but not html

const jsxHeading = <h1 id="heading">namaste React by Abhishek</h1>  // this piece code becomes react element(object)
```

```
const root = ReactDOM.createRoot(document.getElementById("root"));


root.render(jsxHeading);
```

You’re absolutely correct! The JavaScript engine doesn’t understand JSX syntax. If you try to run JSX code in the browser console, it will throw an error.

The reason your code is executing successfully is because of Parcel. Parcel downloads Babel, which is present in the node modules. Babel helps convert JSX to JavaScript so that the JavaScript engine can understand it.

Here’s the process:

- You write JSX code, like const jsxHeading = <h1 id="heading">Namaste React by Abhishek</h1>.
- Babel converts this JSX code into React.createElement calls, which JavaScript can understand.
- These React.createElement calls create JavaScript objects.
- When these objects are rendered (using something like ReactDOM.render()), they become HTML elements that are displayed on the webpage.
-

Babel is a transpiler, a type of compiler that can convert one source code into another source code. It’s used to make your JavaScript code compatible with older browsers that may not support the latest JavaScript features.

In JSX, we write in camelCase. This is because JSX is closer to JavaScript than to HTML, and in JavaScript, camelCase is the convention for writing variable names and function names.

If you’re writing multiple lines of JSX, you should wrap your code in parentheses. This helps with readability and avoids potential issues with automatic semicolon insertion by the JavaScript engine.

Part 3

A React Component is a building block of a React application. As you've mentioned, everything in React is indeed a component. There are two types of components:

1. **Class-based components**: This is the older way of defining components in React. However, it's not commonly used nowadays.
2. **Functional components**: This is the latest and most commonly used way of defining components in React.

A **React Functional Component** is just a normal JavaScript function that starts with a capital letter and returns JSX.

Here's the definition: A function that returns JSX code or a React element is known as a functional component.

Here's an example:

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

If there's just one line, you can write it like this:

```
const HeadingComponent = () => <h1 className="heading">Namaste React by Abhishek</h1>;
```

These components help in building reusable and maintainable code in React applications.

```
  // Import React and ReactDOM (make sure to include these in your HTML file)
import React from 'react';
import ReactDOM from 'react-dom';


// Define a functional component called HeadingComponent
const HeadingComponent = () => {
  return <h1 className="heading">Namaste React by Abhishek</h1>;
}
```

```
// Create a root element using ReactDOM
const root = ReactDOM.createRoot(document.getElementById("root"));


// Render the HeadingComponent inside the root element
root.render(<HeadingComponent />);
root.render(HeadingComponent);  -- this will throw error. You cant render this in normall way. Put it inside react fragments. < />. It's the syntax that babel enderstands.
```

Component composition.

Putting one component inside another.

```
const TitleComponent = () => <h1 className="title">This is a title</h1>;
const HeadingComponent = () => {
  return (
    <div>
      <TitleComponent/>
      <h1 className="heading">Namaste React by Abhishek</h1>
    </div>
  );
}
const root = ReactDOM.createRoot(document.getElementById("root"))
root.render(<HeadingComponent/>)
```

Part 5----------

Yes, you're correct. In JSX, you can run any JavaScript code inside curly braces {}.

You can put a React element inside a React component, and you can also nest React elements inside other React elements. Here's an example:

```
const TitleComponent = () => <h1 className="title">This is a title</h1>;

const elementR = (
  <h1>
    This is a React element. We know Babel converts this, so it's like a piece of JavaScript code only.
    So put this in another element with curly braces.
  </h1>
);

const HeadingComponent = () => {
  return (
    <div>
      {elementR}
      <TitleComponent/>
      <h1 className="heading">Namaste React by Abhishek</h1>
    </div>
  );
}
```

In this code, `elementR` and `TitleComponent` are both React elements that are being used inside the `HeadingComponent` component. This is one of the powerful features of React that allows for component composition, leading to more reusable and maintainable code.