Laying the foundation

1. **what is the another way to start the development build instead of writing this `npx parcel index.html` command again & again?**

   - Yes, We can create a `script` that will build our project instead of writing this command again and again.
   - In `package.json` , in the script section, we can create a different types of scripts like :
     - *start a project in a dev mode,*
     - *build our production ready app,*

       example :

       ```
       "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 start
   ```

   or

   ```
   npm run start
   ```

   For Production Build :

   ```
   npm run build
   ```

   > we can't write `npm build` as a shortcut. so always write `npm run build`

   > `npm start` only is the shortcut of `npm run start`

2. **NPM start :**

   - `npm start` is equivalent to `npm run start`

    - `npm run start` behind the scenes, execute the package `parcel` with `index.html` because we have configured this inside our `package.json`.

Introducing JSX :

- JSX is looks like HTML or XML syntax. JSX stands for JavaScript XML. It's a syntax extension for JavaScript.
- JSX is not HTML, inside JavaScript.
- JSX is different than HTML.
- JSX is a convention where we kind of merge these *HTML* and *JavaScript* together.
- JSX is not a part of React. React apps can be built even without JSX but the code will become very hard to read.
- JavaScript engine cannot understand JSX as it only understands ECMAScript.

Using Pure React :

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

Using JSX :

```
const jsx_heading = <h1>Namaste React using JSX.</h1>
```

- when we `console.log()` heading & jsx_heading, it gives same object.

Introducing Babel

1. **Is JSX a valid JavaScript?**

   The answer is yes and no.

   > Javascript is a code that JavaScript engine can understand.

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

2. **If the browser can’t understand JSX how is it still working?**

   - This is because of `Parcel`. Parcel is doing the job behind the scenes.

   - Before the code gets to JavaSript 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'. Babel 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

```
JSX (transpiled by Babel) ⇒ React.createElement ⇒ ReactElement ⇒ JavaSript
Object ⇒ HTML Element(render)
```

3. **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.

CamelCase example : `tabIndex, htmlFor` etc.

4. **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>
)
```

React component

- Everything is a component in React,

1. **What is components?**

- We can say that we develop a webpage in React, will be made up of pieces called `Components`.
- Like, Header, Footer, Card, Button, Heading, Title, Search Bar, all of them are components.
- A component represents a part of the user interface.
- Usually, the root component is named as the APP component.
- All components come together to make an entire Application.

   - Components are reusable and also it can be nested inside other components with different properties information.
   - Component placed in JavaScript file. for ex: app component in `App.js`

2. **There are two types of components:**

   - Class Based Component - Old Way of writing code

     > Class Based component uses javascript classes

     - Class components are basically ES6 Classes.
     - It maintains a private state, and uses that state to describe UI. It must contain a render() method which returns html.

   - Functional Component - New Way of writing code

     > Functional Component uses javascript functions to create a components

3. **What is React Functional component?**

   - It is just a JavaScript Function that returns some piece of JSX code or a react element.

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

     All are the same

     ```
     const HeadingComponent1 = () => (
         <h1>Namaste</h1>
     )

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

     for single-line code :

     ```
     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()}`

4. **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>
    )
    ```

5. **How to use JavaScript code inside JSX?**

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

    Example :

    ```
    const number = 10000;

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

6. **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>
    )
```

7. **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.

8. **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)