Chapter-03-Laying-the-Foundation

▼ Babel

- Old browsers version don't support newer features of javascript (eg: ES6)
- That time code is converted to the way the older versions understands
  - Using Polyfill
  - Polyfill - creating unsupported feature using supported features

What converts our code to older code?

→ Babel https://babeljs.io/

- Babel is just a library - which takes our code as input and outputs other code (with required processing)
  - Eg: To remove console.logs from code
    - https://www.npmjs.com/package/babel-plugin-transform-remove-console
    - `npm install babel-plugin-transform-remove-console --save-dev`
    - babel.rc is a configuration file for babel → Add configuration to remove console

▼ Keys

- Always give keys to sibling. Key is something that is unique, else we get warning

```
const heading = React.createElement(
  "h1",
  {
    id: "title",
    key: "h1",
  },
  "Heading 1 for parcel"        You, 21 hours ago
);

const heading2 = React.createElement(
  "h2",
  {
    id: "title",
    key: "h2",
  },
  "Heading 2"
);
```

▼ Why do we need a key?

- React tracks uniqueness using key.→ https://legacy.reactjs.org/docs/reconciliation.html#keys
- When react is updating the dom, think of a scenario we add a sibling to start. React will have to put lot of effort to identify the diff. It may have to rerender everything. So we use keys to ease it

```
<ul>
  <li>Duke</li>
  <li>Villanova</li>
</ul>

<ul>
  <li>Connecticut</li>
  <li>Duke</li>
  <li>Villanova</li>
</ul>
```

- So we use keys to ease it. This will make react easy to just insert new element to dom

```
<ul>
  <li key="2015">Duke</li>
  <li key="2016">Villanova</li>
</ul>

<ul>
  <li key="2014">Connecticut</li>
  <li key="2015">Duke</li>
  <li key="2016">Villanova</li>
</ul>
```

How does React.createElemet work?

- gives us the javascript object

- This needs to be converted to HTML and put to DOM → ReactDOM does that

```
// React.createElement ⇒ Object ⇒ HTML(DOM)
```

When we have to build large html - would be difficult usingReact.CreateElement. Makes code messy

```
const heading = React.createElement(
  "div",
  {
    id: "title",
    key: "h2",
  },
  [
    React.createElement(
      "h1",
      {
        id: "title",
        key: "h2",
      },
      "Namaste React"
    ),
    React.createElement(
      "h1",
      {
        id: "title",
        key: "h2",
      },
      "Namaste React"
    ),
    React.createElement(
      "h1",
```

Instead we use JSX

What is JSX?

- Facebook wanted to update html using javascript in a better way (instead of document.get ..)
- React.createElement was messy

```
const heading2 = <h1>Namaste React</h1>;
```

- Is it Javascript? yes perfectly valid Javascript code
- To write in multiple line, use paranthesis

```
const heading2 = (
  <h1 id="title" key="h2">
    Namaste React
  </h1>
);
```

Is JSX html inside javascript ?

- false

JSX is html like syntax, but it is not html inside javascript.

How does javascript executes this JSX?

- If we copy to browser will it work → No
- Who understands? → Babel
    - How does Babel understand?
        - reads code line by line and parses.
- JSX uses React.createElement in background

```
// JSX ⇒ React.createElement ⇒ Object ⇒ HTML(DOM)
```

- Babel understands JSX and converts it to React.createElement

Try it out: https://babeljs.io/

So JSX is not React. Babel converts JSX to React.createElement. React comes only next

Whenever Babel sees <, starts converting it to input of React.createElement

- Babel is opensource → maybe facebook developers worked on this JSX part

Advantages of JSX:

- Developer experience
- Readability
- maintaibility

Code is written for humans, not machines. So readability is important. Machine can read binary

Flow:
Babel: JSX → React.createElement()

React: outputs javascript object

ReactDOM: Takes Javascript object as input & Updates DOM

Babels comes along with Parcel. No need to install separately

Even node_modules has package.lock.json (for transitive dependencies)

No need of importing any library to use JSX. Babel reads line by line code and understands JSX

▼ JSX

▼ How does **React.createElement()** work?

- gives us the javascript object
- This needs to be converted to HTML and put to DOM → ReactDOM does that

```
// React.createElement ⇒ Object ⇒ HTML(DOM)
```

▼ Why JSX?

- When we have to build large html - would be difficult usingReact.CreateElement. Makes code messy. Here is the example of passing headers as child nodes of div

```
const heading = React.createElement(
  "div",
  {
    id: "title",
    key: "h2",
  },
  [
    React.createElement(
      "h1",
      {
        id: "title",
        key: "h2",
      },
      "Namaste React"
    ),
    React.createElement(
      "h1",
      {
        id: "title",
        key: "h2",
      },
      "Namaste React"
    ),
    React.createElement(
      "h1",
```

- Facebook wanted to update html using javascript in a better way (instead of document.get ..). Created JS object for html using React.createElement() was messy

▼ How to write JSX?

```
const heading2 = <h1>Namaste React</h1>;
```

- Is it Javascript? → yes perfectly valid Javascript code
- To write in multiple line, use paranthesis

```
const heading2 = (
  <h1 id="title" key="h2">
    Namaste React
  </h1>
);
```

- Is JSX html inside javascript ? → false

▼ What is JSX?

- JSX is html like syntax, but it is not html inside javascript.

▼ How does javascript executes this JSX?

- If we copy to browser will it work → No
- Who understands JSX then? → Babel
  - How does Babel understand?
    - reads code line by line and parses.
- JSX uses React.createElement in background

```
// JSX ⇒ React.createElement ⇒ Object ⇒ HTML(DOM)
```

- Babel understands JSX and converts it to React.createElement()
- Try it out: https://babeljs.io/

- So JSX is not React. Babel converts JSX to React.createElement(). React Element comes only next. Whenever Babel sees <, starts converting it to input of React.createElement(). Later this React element (i.e javascript object) can be passed to ReactDom.render() to update DOM.
- Babel is opensource → maybe facebook developers worked on this JSX part
- Flow:
  - Babel: JSX→React.createElement()
  - React: outputs javascript object
  - ReactDOM: Takes Javascript object as input & Updates DOM
- No need of importing any library to use JSX. Babel reads line by line code and understands JSX. Babels comes along with Parcel. No need to install separately

▼ Advantages of JSX

- Developer experience
- Readability
- maintaibility

▼ React Element

- Its just a javascript object, that has required information related to html dom. This object will be passed to ReactDom.render() to update actual DOM.

- Here Babel will replace it with React.createElement() to get javascript object

```
const heading2 = (
  <h1 id="title" key="h2">
    Namaste React
  </h1>
);
```

Put in next-gen JavaScript	Get browser-compatible JavaScript out

`const heading = (<h1 id="title" key="h2">`
`Namaste React`
`</h1>);`
	`const heading = /*#__PURE__*/React.createElem`
	`  id: "title",`
	`  key: "h2"`
	`}, "Namaste React");`

▼ React Components

Normal saying: Everything is a component in React

1. Functional Component → new way of writing code
2. Class based Component → old way way of writing code

This tutorial mostly functional way of wrining code as it is latest. But there will be one session full on class component

▼ Functional Component

▼ What is a functional component?

- Nothing but a Normal Javascript Function , that returns React Element. (i.e React.createElement() ⇒ Javascript object)
  - Or JSX (which Babel converts it to React Element)

▼ Conventions

- Functional component name starts with Capital letter
  - Mandatory ? → No, But normal convention
- When return is multiple lines, add ()

```
const HeaderComponent = () => {
  return (
    <div>
      <h1>Namaste React functional component</h1>
      <h2>This is a h2 tage</h2>      You, 5 seconds ago
    </div>
  );
};
```

- As we are using arrow function, no need to write return too

```
const HeaderComponent2 = () => (
  <div>
    <h1>Namaste React functional component</h1>
    <h2>This is a h2 tage</h2>
  </div>
);      You, last week • chapter-01
```

- We use tags <> when we need to invoke function or class to get react object inside react.render,

```
//async defer
root.render(<HeaderComponent />);
```

- Unlike when we are directly passing react object, no need of tags. (i.e JSX converted to → React.createElement() → React object. render() just need react object)

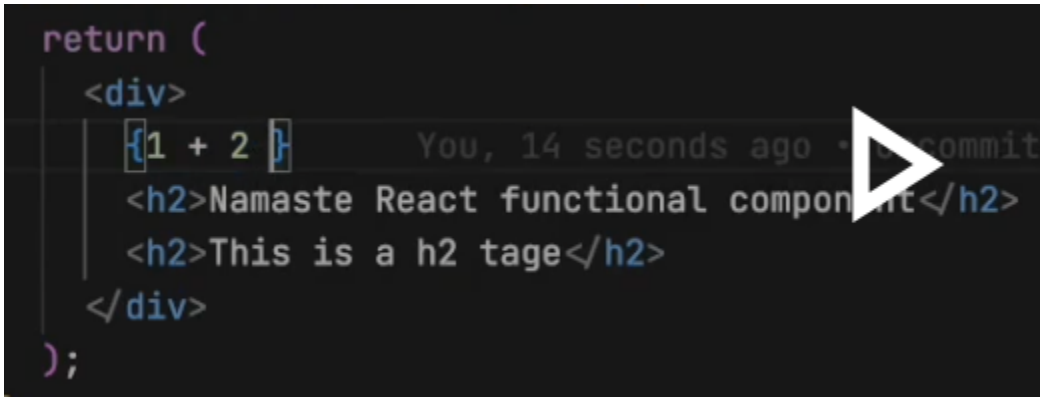
```
const heading = (      You, 3 secon
  <h1 id="title" key="h2">
    Namaste React
  </h1>
);
```

```
root.render(heading)
```

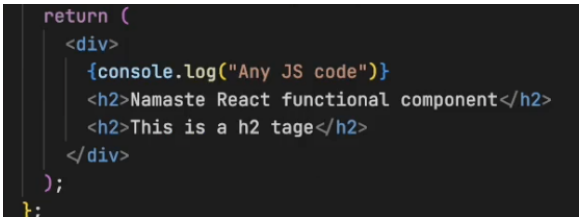
▼ JSX is Super Powerful

▼ When we are writing JSX, we can include any piece of javascript code inside { }. Eg:

- Here whatever inside { } returns, is added to div element



- This invokes console of windows, and add log in browser devtool.



▼ Calling variable and function in jsx

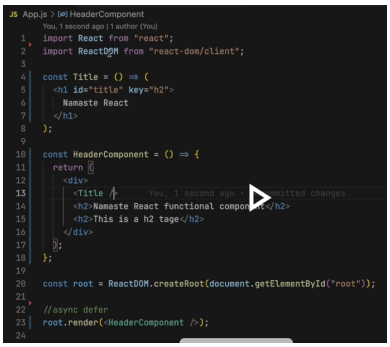
- Variable → title

```
You, 1 second ago | 1 author (You)
import React from "react";
import ReactDOM from "react-dom/client";

const title = (
  <h1 id="title" key="h2">
    Namaste React
  </h1>
);

const HeaderComponent = () => {
  return (
    <div>
      {title}        You, 1 second ago • Uncommitted changes
      <h2>Namaste React functional component</h2>
      <h2>This is a h2 tage</h2>
    </div>
  );
};

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

//async defer
root.render(<HeaderComponent />);
```

- Function → Title() . Its simple javascript. Just invoke function inside {}

```
.js > [ ] HeaderComponent
You, 6 seconds ago | 1 author (You)
import React from "react";
import ReactDOM from "react-dom/client";

const Title = () => (
  <h1 id="title" key="h2">
    Namaste React
  </h1>
);

const HeaderComponent = () => {
  return (
    <div>
      {Title()}        You, 1 second ago • Uncommitted changes
      <h2>Namaste React functional component</h2>
      <h2>This is a h2 tage</h2>
    </div>
  );
};

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

//async defer
root.render(<HeaderComponent />);
```

  - Instead we can also write in tag, if its function (I think its way of invoking function THAT BABEL UNDERSTANDS, if it returns JSX). So {Title()} is same as <Title />



▼ JSX also sanitises data we will execute, prevents xss

```
const data = api.getData();

const HeaderComponent = () => {
  return (
    <div>
      {data}        You, 2 seconds ago • Uncommitted changes
      <h2>Namaste React functional component</h2>
      <h2>This is a h2 tage</h2>
    </div>
  );
};
```

▼ What is Component Composition?

Its a jargon. Its just using Component inside component

```
JS App.js > HeaderComponent
You, 11 seconds ago | 1 author (You)
1  import ReactDOM, { createRoot } from "react-dom/client";
2
3  const Title = () => (
4    <h1 id="title" key="h2">
5      Namaste React
6    </h1>
7  );
8
9  // Composing Comopnentss
10 const HeaderComponent = () => (      You, 11 seconds ago • Uncommitte
11   <div>
12     <Title />
13     <h2>Namaste React functional component</h2>
14     <h2>This is a h2 tage</h2>
15   </div>
16 );
17
18 const root = ReactDOM.createRoot(document.getElementById("root"));
19
20 root.render(<HeaderComponent />);
21
```