Polyfill

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

→ babel do this conversion automatically

Eg : ES6 is the newer version of javascript if I'm working on 1999 browser, my browser will not understand what is this
const, new promise 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 version code.

Babel

transcompiler

It is a Javascript package/library used to convert code written in newer versions of JS (ECMAScript 2015, 2016, 2017 etc) into code that can be run in older JS engines.

To run our app, command is :-

npx is to execute npm modules without installing it

```
npx parcel 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.

Date.....

in package.json

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

So, to run the project, I have to use :-

`npm run start`

shortcut :-
`npm start`

"start" script execute this command.

Build command

`npx parcel build index.html`

```
"script": {
    "build": "parcel build index.html"
}
```

Note :-

`npx = npm run`

So, `npm run build` will build a project.

Date.....

Console logs are not removed automatically by parcel. You have to configure your project to remove it.

→ There is a package which helps to remove console logs :-

`babel-plugin-transform-remove-console`

→ Before installing this package create a folder called `.babelrc` for configuration

→ And include

```
{
  "plugins": [["transform-remove-console",
    { "exclude": ["error", "warn"] }]]
}
```

→ Then, build: `npm run build`

to see that all console.logs are removed.

Render — means updating something in the DOM

Reconciliation : The mechanism to diff one tree with another to determine which parts need to be changed and then update the original DOM.

→ helps to make React applications fast and efficient by minimising the amount of work that needs to be done to update the changes.

→ So you don't have to worry about what changes on every update.

Eg :-

```
<ul>
    <li> first </li>   } siblings
    <li> second </li>  }
</ul>
```

When adding an element at the end of the children : The three works well.

```
<ul>
    <li> first </li>
    <li> second </li>
    <li> third </li>
</ul>
```

• Render() function is creating a tree of React elements.

On the next state or props update, Render() fn will return a different tree of React elements.

Date.....

whenever react is updating the DOM, for eg :-

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

Now, I introduced one child over the top, then react will have to do a lot of efforts, react will have to re-render everything. That means (react will have to change the whole DOM tree.)

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

As react has to re-render everything, it will not give you good performance.
In large-scale application, it is far too expensive.

Solution - Introduction of keys

→ React supports 'key' attribute
→ When children have keys, React uses the key to match #children in the original tree with children in subsequent tree.
Thus, making tree-conversion efficient.

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

Spiral

Teacher's Sign .....

Date.....

means same type
<div
<div
<div

Thus React has to do very less work, so always use keys whenever you have multiple children. (of same type)

CreateElement

→ React createElement() is creating an object

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

nested
<div
<div
<h1>

If you want to build a big HTML structure, then using 'createElement()' is not a good solution.

So there comes introduction of JSX.

JSX

When facebook created React, the major concept behind bringing react was that we want to write a lot of HTML using JS because JS is very performant.

~~import { createElement as ce } from react~~

```
import {createElement as ce} from react
const heading = ce(
                  "h1",
                  { id: "title",
                    key: "h1"
                  },
                  "Hellow world"
);
```

use parentheses if more than one line.

⇐ Instead of writing all these

```
const heading =
<h1> Hellow world </h1>
```

This is JSX.

Date.....

* JSX is not 'HTML inside javascript'

⇒ JSX has 'HTML-like' syntax.

This is a valid javascript code :-

```
const heading = (
    JSX expression {
        <h1 id="title" key="h1">
            Hellowoned
        </h1>
    }
)
```

React keeps track of 'key'

Homework what is difference b/w HTML & JSX ?

1 major diff

	HTML	JSX
(1)	In HTML almost all tags have an opening and closing tag except probably a few like 	In JSX, however, any element can be written as a self-closing, for eg.:
(2)		
(3)	 	 </br>.
(4)	Event Listners onclick onchange	onClick onChange ? use camelcase.

Spiral

Teacher's Sign .....

Date.....

Our browser cannot understand JSX

'Babel' understands this code.

→ JSX uses React.createElement behind the scenes.

→ Babel converts JSX to react createElement().

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

(Read Babel's Documentation)

→ JSX is created to empower React

Advantages of JSX

→ Developer experience
→ Syntactical sugar
→ Readability
→ Less code
→ maintainability
→ No Repeatition

Babel comes along with parcel

Date.....

COMPONENT

'Everything is a component in Realt'

React Components

→ 2 types :-

a) functional component — NEW WAY of writing code

b) class Based component — OLD WAY

→ functional component

- is nothing but a javascript function
- is a normal JS fnc which returns some piece of react element (here JSX)

Eg :-

```
const HeaderComponent = () => {

    return <h1> Hello world </h1> ;

};
```

- For any component,

  Name starts with capital letter

  (It is not mandatory, but it's a convention)

to render functional component, write :-

```
<HeaderComponent />
```

Date.....

React element

```
const heading = (
  <h1 id="title"
    key="h1">
    Hello world
  </h1>
);
```

⇓

React element is finally an object

converting it into arrow function with name functional component

functional component

```
const heading = () => {
  return (
    <h1 id="title" key="h1">
      Hello world
    </h1>);
};
```

⇓

functional component is finally a function.

The Next Amazing thing ⇓

① *
```
const Title = () => {
  return <h1> Hello world </h1>
};
```
is a functional component

*
```
const HeaderComponent = () => {
  return (
    <div>
      <Title />
      <h2> Namaste React </h2>
      <h2> Hii all </h2>
    </div>
  );
};
```

instead of this you can write { Title() }

This is component composition

This is a normal javascript function.

Teacher's Sign.....

Date.....

② * const title = (
<h1> Hello world </h1>
);

{ is a normal variable

* const HeaderComponent = () => {

return (

use Javascript in {}

you can also return without return keyword

{title}
<h2> Namaste React </h2>
<h2> Hai all </h2>

);
}

const HeaderComponent = () => (

{title}
<h2>
<h2>

);

Note:-

* whenever you write JSX, you can write any piece of javascript code between paranthesis {}. It will work.
* JSX is very secure.
JSX makes sure your app is safe.
It does sanitization.

```
const data = api.getData();
const HeaderComponent = () => {
    return (
        <div>
            {data}
            <h2> Hello world </h2>
        </div>
    );
}
```


Teacher's Sign .....

always use capital letter initials naito render naihoga.

transpiler → converts a high-level language to another high level language.

Date.....

Component Composition

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

Assignment-3

Q. What is JSX ?

Solut JSX stands for javascript XML. JSX allows us to write HTML like elements in Javascript and place them in the DOM without any createElement() and/or appendChild() method.

Example 1 using JSX :-

```
const myElement = <h1> I LOVE YOU </h1>
const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(myElement);
```

Example 2 without JSX :-

```
const myElement = React.createElement("h1", {}, "I do not use JSX");
const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(myElement);
```

Q. Superpower of JSX ?

Solut Using JSX, you can write markup inside javascript. Also JSX is very easy to maintain and debug.

Date.....

Q. Role of type attribute in script tag? what options can I use there?

Soln: The type attribute specifies the type of the script. The type attribute identifies the content between the <script> and </script> tags. It has a Default value which is "text/javascript".

type attribute can be of the following types

- text/javascript : It is the basic standard of writing javascript code inside the <script> tag.

Syntax:-
<script type="text/javascript"></script>

- text/ecmascript : this value indicate that the script is following the ECMAScript standards
- module:- This value tells the browser that the script is a module that can be imported and export other files or modules inside it.
- text/babel:- This value indicates that the script is a babel type and required babel to transpile it.
- text/typescript : As the name suggests the script is written in Typescript.

Q. {TitleComponent} vs {<TitleComponent/>} vs {<TitleComponent></TitleComponent>} in JSX.

Soln:- {TitleComponent}: This value represents a component as that is basically returning a Javascript expression or a variable as {} this can embed a javascript expression or a variable inside it.

- {<TitleComponent/>}: this value represents a component that is basically returning some JSX value. In simple terms it is a func. that is returning a JSX value.
- {<TitleComponent></TitleComponent>}: It is equivalent to {<TitleComp/>} only when it doesn't have any child component. The opening & closing tags are created to include the child components.

Spiral

Teacher's Sign .....