## Part 1:

Let's write some scripts in package.json file to run

Like we run

>npx parcel index.html

Now we can run it with help of NPM scripts.

Now go to package.json and in the scripts key write these commands

scripts": {

"test": "jest",

"start": "parcel index.html",

"build": "parcel build index.html"

},

Now after witing this no need to write

>npx parcel index.html

>npx parcel build index.html

We can use

> npm run start

It will run the scripts which we have written in the script section.

So we can customise any script with the help of package.json file.

Npm run start is equal to npx parcel index.html

Npm run build is equal to npx parcel build index.html

Also, we can call it short like

NPM start directly.

In this example

>npx parcel index.html

>npm run start

> npm start

The 3 statements are going to do the same functionality.

But this is only going t work for start.

Start is keyword reserver by npm

But for other scripts, you have to call like

> npm run \_\_\_\_ (script name)

## Part 2

HTML tags are dom elements

Like we have elements in react also.

The React elements are ultimately are objects.

When we render the React elements then it will become dom elements.

## Part 3.

As we know we have created the element with help of react.

react.createElement

reactDOM.createRoot/

But the structure is not good in look and feel not developer-friendly

So Facebook created something called JSX.

React and JSX are different. (JSX is not part of react).

We can write react code without help of jsx code but that will be very complicated to understand.

Jsx makes our life easy.

What is JSX?

Previously we have separate page for html and js

If we have created some functionality on click of button we have to write it separately

Jsx mixed them (as both are interlinked so jsx mixed them)

We can write mixed code of js and html in jsx.

Const heading = <h2>This is Heading in JSX</h2>;

Jsx is not HTML inside javascript.

Jsx and html are different

**Jsx is HTML LIKE syntax** or XML like syntax.

Lets see

HTML

<h1 id=”parent” >Hello World</h1>

Javascript

const parent = React.createElement(

'div',

{

id: "parent"

},

"Hello World"

);

JSX

const heading = <h1 id='parent'>Hello World</h1>

These 3 are different in syntax but have exactly same output.

When you print console.log js parent and jsc heading it will have same object.

That process jsx converts the jsx system to js code before use.

So now next onwards we are not going to use createElement method

We will use JSX syntax.

ES6 is ECMA script

Which do not understand jsx it only understands the javascript.

When we directly his the jsx code in js it will show error to use.

Js engine can not read jsx code

So we need to convert the jsx to js before use.

Parcel is doing the same.

This code is transpiled before it goes to the js engine so that the browser can understand it.

Parcel transfer jsx code to js

But does parcel doing it alone? No, Parcel is a manager

This functionality is performed by Bebel.

Bebel is npm package. (its Javascript compiler)

So ultimately what is happening here

When we write code in jsx

It converts code into react code

Then react code convert into object

Then that object will be rendered and create HTML code

Which we can see on the browser.

Let test html like code written in jsx is really a html code or something else

HTML

<h1 id='parent' class='heading'>Hello world</h1>

JSX

const heading = <h1 id='parent' className='heading'>Hello World</h1>

If you write class in jsx it will show Error.

Now when you transpile the code and check on console you will get again

HTML code.

<h1 id='parent' class='heading'>Hello world</h1>

It clearly shows that code written inside the JSX is not HTML its html like code.

We do not put hyphen ( - ) in jax

We have to use camel case structure to write the attributes like data-name it will be dataName

Assignment

List out all the tags in jsx which is different from html but has same functionality

Example class will become className in jsx. Find others.

Now

About single line and multiple line jsx code

If you are writing single line of code then its perfectly valid syntex

Const heading = <h1>Hello world</h1>;

But then you will write multi line statements

Then you have to wrap the code in the braces. ()

Const heading = ( <h1>

Hello world

</h1>);

Why to use ()?

As we know babel is converting it to js code,

So Babel should know where is the code starts and where is ending.

Extension in VS Code.

> PRETTIER - CODE FORMATTER

> Bracket pair colorization toggler

> ESLint

> Better Comments

## Part 4: React Components :

There are 2 types of components

> Class-based components (old way of wiring code )

> Functional components ( new ways of writing code)

What is the React functional component?

Its a normal js function which returns some JSX.

> Always give a name with capitals.

const HeadingComponent = () =>{

return <h1>Hello World</h1>

}

Or if we have single line function we can write it without return andbrasis.

const HeadingComponent = () => <h1>Hello World</h1>;

Both above code is the same.

// React element

const heading = (

<h1 className='head' tabIndex={5}>Hello World</h1>

);

// react component

const HeadingComponent = () =>{

return <h1 className='head' tabIndex={5}>Hello World</h1>;

}

We render react elements.

Can we render react component? If yes how?

root.render (heading); // correct

root.render(HeadingComponent) // wrong

There is a way to render component.

Root . render ( <HeadingComponent /> ); // correct syntax.

So when we pass <HeadingComponent /> to babel then it will understand there is a component with the given name.

import React from 'react';

import ReactDOM from 'react-dom/client';

// const heading = React.createElement('h1', {"id":"heading"}, "Hello world with React!");

const heading = (

<h1 className='head'>Hello World, React Element</h1>

)

const HeadingComponent = () =>(

<div id='container'>

<h1 className='heading'>Hello World, React Component</h1>

</div>

)

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

root.render(<HeadingComponent />);

We can use components inside the component and that terms called

COMPONENT COMPOSITION.

import React from 'react';

import ReactDOM from 'react-dom/client';

const Title = (

<h1 className='head'>Hello World,t</h1>

)

const HeadingComponent = () =>(

<div id='container'>

<Title />

<h2 className='heading'>Example of Component compositions</h2>

</div>

)

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

root.render(<HeadingComponent />);

## Part 5:

Can we use normal function in place of fat arrow function?

YES

arrow

const HeadingComponent = () =>(

<div id='container'>

<Title />

<h2 className='heading'>Example of Component compositions</h2>

</div>

)

Function

const HeadingComponent = function(){

return (<div id='container'>

<Title />

<h2 className='heading'>Example of Component compositions</h2>

</div>);

}

Both are same.

Now

In the jsx how to use js?

Magic:

import React from 'react';

import ReactDOM from 'react-dom/client';

// component

const Title = () => (

<h1 className='head'>Hello World</h1>

)

// element

const title = (

<h1 className='head'>Hello World</h1>

)

const HeadingComponent = function(){

return (<div id='container'>

{title}

<h2 className='heading'>Example of Component compositions</h2>

</div>);

}

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

root.render(<HeadingComponent />);

In the JSX syntax, we can write any piece of js code just under the curly braces , check the above example

We use the react element in the react component

Like:

<h1>{heading}</h1>

Const number = 1000;

<h2>{number}</h2>

<h1>{ console.log() } </h2> // Amazing

We can perform any code in the { } .

We can put elements inside the elements.

import React from 'react';

import ReactDOM from 'react-dom/client';

// component

const Title = () => (

<h1 className='head'>Hello World</h1>

)

const elem = <span>Yes. </span>;

// element

const title = (

<h1 className='head'>Hello World , {elem}</h1>

)

const HeadingComponent = function(){

return (<div id='container'>

{title}

<h2 className='heading'>Example of Component compositions</h2>

</div>);

}

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

root.render(<HeadingComponent />);

We can place anything anywhere, it will work fine, just maintain the proper order accessibility and scope of elements.

We can see we are executing elements directly

Suppose we are getting some malicious data in api like some scripts

Then it will be directly executed in the code..

This is called cross-site scripting.

If I can get access to js on your browser. I can steal any information.

**JSX is take care of this injection attacks**

**Jsx escapes the code.**

**It sanitises the data.**

**It prevents cross-site scripting.**

<Title /> This is actually calling the function

We can also write like

{Title()} both are same

<h1> <Title /> </h1>

This will be converted into

<h1> {Title() } </h1>

<Title />

<Title > </Title>

{Title()}

All the above statements are the same.