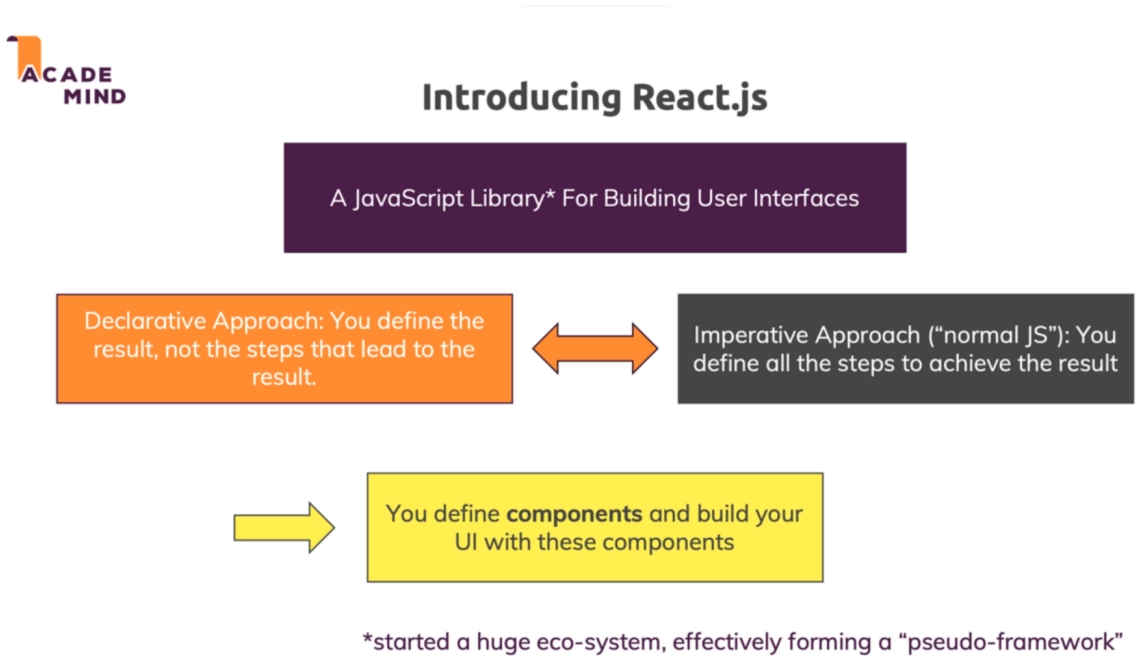
React Refresher Notes

By: Shashwat Agrawal

- ## React in a Nutshell



- ## Setting a react project

❖ npx create-react-app myApp
❖ cd myApp
❖ npm start

- ## Understanding JSX

  - React uses a special syntax where one can write HTML code in JavaScript files.

- Thus, Javascript XML Code. just creating normal functions in javascript and using them as components to render objects on web applications.
- JSX - JSX stands for JavaScript XML. JSX allows us to write HTML in React. JSX makes it easier to write and add HTML in React.
- Eg.
  return <h1 title="This works">This is react!</h1> is same as
  return React.createElement('h1', { 'title': 'This works' }, 'This is react')
  So instead of writing js functions in js files we write HTML code to make it easier.

Components in ReactJS

- A react component can be defined in two ways:
  - A functional component - where a normal js function is there which returns a html code.
  -
```
const App = () => {
  return <h1 title="This works!">Hi, <span>this</span> is ReactJS!</h1>;
};
```
  - Class-based components - which is a class that extends to render html code.
```
class App extends React.Component {
  render() {
    return <h1 title="This works!">Hi, <span>this</span> is ReactJS!</h1>;
  }
}
```
- Components are used in order to reuse them so that less code is required.
- We use multiple components and use components inside other components by importing and exporting them. But do we do so?
- CSS files no matter where they are imported are applied globally in React app i.e. all JSX files.

Working with Multiple Components

- Why do we make multiple components in React?
  - It helps in splitting your app into smaller pieces.
  - It helps in structuring code and being more specific and focused on a particular section.
  - This eventually is a better approach when working and managing on a big project.

- **Passing Data between Components (Parent to Child Component)**
  - We want to render/display data dynamically in our components rather than just hard coding it. How do we do that?
  - Props concept - we can pass data among components i.e. from A component to component B using props.
  - Earlier:

```
import React from 'react';

import './GoalList.css';

const GoalList = () => {
  return (
    <ul className="goal-list">
      <li>Finish the Course</li>
      <li>Learn all about the Course Main Topic</li>
      <li>Help other students in the Course Q&amp;A</li>
    </ul>
  );
};

export default GoalList;
```

  - Later:

```
const App = () => {
  const courseGoals = [
    {id: 'cg1', text: 'Finish the Course'},
    {id: 'cg2', text: 'Learn all about the Course Main Topic'},
    {id: 'cg3', text: 'Help other students in the Course Q&A'},
  ];

  return (
    <div className="course-goals">
      <h2>Course Goals</h2>
      <GoalList goals={courseGoals} />
    </div>
  );
};
```

```
import React from 'react';

import './GoalList.css';

const GoalList = props => {
  return (
    <ul className="goal-list">
      {props.goals.map(goal => {
        return <li key={goal.id}>{goal.text}</li>;
      })}
    </ul>
  );
};

export default GoalList;
```

- Props (short form for properties that every JSX component has by default) is an object that contains a course goals array by the name of goals.
- courseGoals is an array of objects but in components, we return HTML code thus to return that we use special syntax which is curly braces.
- Map function is used to map every object into a list item i.e. tag.

Passing Data between Components (Child to Parent Component)

- When we add a button in the form instead of using the onClick function in the button we use the onSubmit function in the form tag, this is a general practice.
- Whenever a form is created. Browser by default sends data to the server side. That's why the page reloads on submitting. We can prevent this from happening.
- In the onSubmit() function (which is an event listener) we pass a function but don't want it to execute thus we only point towards it, this passed function is called a handler. Handler is meant to handle events.
- Preventing the form data to send to server-side is also a kind of event to be prevented which is done as

```
const NewGoal = () => {
  const addGoalHandler = event => {
    event.preventDefault();

    const newGoal = {
      id: Math.random().toString(),

    };
  };

  return (
    <form className="new-goal" onSubmit={addGoalHandler}>
```

- To send data from child to parent, this is also done by passing props only. Here, instead of passing data directly, we want to collect the data.
- Thus, we pass the event handler. eventListener (eg.onAddGoal in this case) that listened to the made handler is passed to the NewGoal component.
- This handler function will push the data item fetched from the child component into the array created earlier (array => course goals)

App.js | NewGoal.js | NewGoal.css | GoalList.js

src > App.js > App

```
12    ];
13
14    const addNewGoalHandler = newGoal => {
15      courseGoals.push(newGoal);
16      console.log(courseGoals);
17    };
18
19    return (
20      <div className="course-goals">
21        <h2>Course Goals</h2>
22        <NewGoal onAddGoal={addNewGoalHandler} />
23        <GoalList goals={courseGoals} />
24      </div>
25    );
26  };
27
```

App.js | NewGoal.js | NewGoal.css | GoalList.js

src > components > NewGoal > NewGoal.js > NewGoal > addGoalHan

```
3   import './NewGoal.css';
4
5   const NewGoal = props => {
6     const addGoalHandler = event => {
7       event.preventDefault();
8
9       const newGoal = {
10        id: Math.random().toString(),
11        text: 'My new goal!'
12      };
13
14      props.onAddGoal();
15    };
16
17    return (
18      <form className="new-goal" onSubmit={addGoalHandler}>
```

- ## States Management in ReactJS
    - React does not display any data when changed, for displaying changed data we need hooks.
    - Whenever the data is updated in a component, instead of re-rendering the whole page react just render that component.
    - States are the way by which react gets to know that it's time to re-render the component, i.e. whenever the state changes, that is, stored data in it is updated, then only the component is re-rendered.
    - States are managed using hooks, which are essentially the functions for managing states. It creates snapshots and compares the snapshots under the hood.
    - **useState** is the first hook we are working with. It returns always an array of two elements, first is the data (updated data) and other one is a function to update that data.

```
App.js ×   NewGoal.js   NewGoal.css   GoalList.js   Go
src > App.js > App
7   const App = () => {
8     const [courseGoals, setCourseGoals] = useState([
9       { id: 'cg1', text: 'Finish the Course' },
10      { id: 'cg2', text: 'Learn all about the Course Main Topic'
11      { id: 'cg3', text: 'Help other students in the Course Q&A'
12    ]);
13
14    const addNewGoalHandler = newGoal => {
15      setCourseGoals(courseGoals.concat(newGoal));
16    };
17
18    return (
19      <div className="course-goals">
20        <h2>Course Goals</h2>
21        <NewGoal onAddGoal={addNewGoalHandler} />
```

    - Here, why concat is used and why push was used earlier to update the array. useState wants to update data by making a new copy of the data and discarding the previous one.
    - By pushing, we are updating the existing array while by concating we are making a new array.
    - General practise or recommended approach to update the data when is it dependent on previous state is by creating a function inside the function returned

by useState, as follows:

```
const addNewGoalHandler = newGoal => {
  // setCourseGoals(courseGoals.concat(newGoal));
  setCourseGoals((prevCourseGoals) => {
    return prevCourseGoals.concat(newGoal);
  });
};
```

- ## Fetching user data in ReactJS
  - Just like every other state management, fetching of data is also done using states.
  - Two way binding is the approach we use.
  - Value attribute binds the input Text and onChange is the event listener that holds the handler. This handler will set the InputText to event.target.value which was received from the form.
  -

```
App.js   NewGoal.js ●   NewGoal.css   GoalList.js   G
src > components > NewGoal > NewGoal.js > NewGoal > addGoalHandler
5   const NewGoal = props => {
6     const [enteredText, setEnteredText] = useState('');
7
8     const addGoalHandler = event => {
9       event.preventDefault();
10
11      const newGoal = {
12        id: Math.random().toString(),
13        text: enteredText
14      };
15
16      setEnteredText('');
17
18      props.onAddGoal(newGoal);
19    };
20
```

```
return (
  <form className="new-goal" onSubmit={addGoalHandler}>
    <input type="text" value={enteredText} onChange={textChangeHandler}
    <button type="submit">Add Goal</button>
  </form>
);
};
```

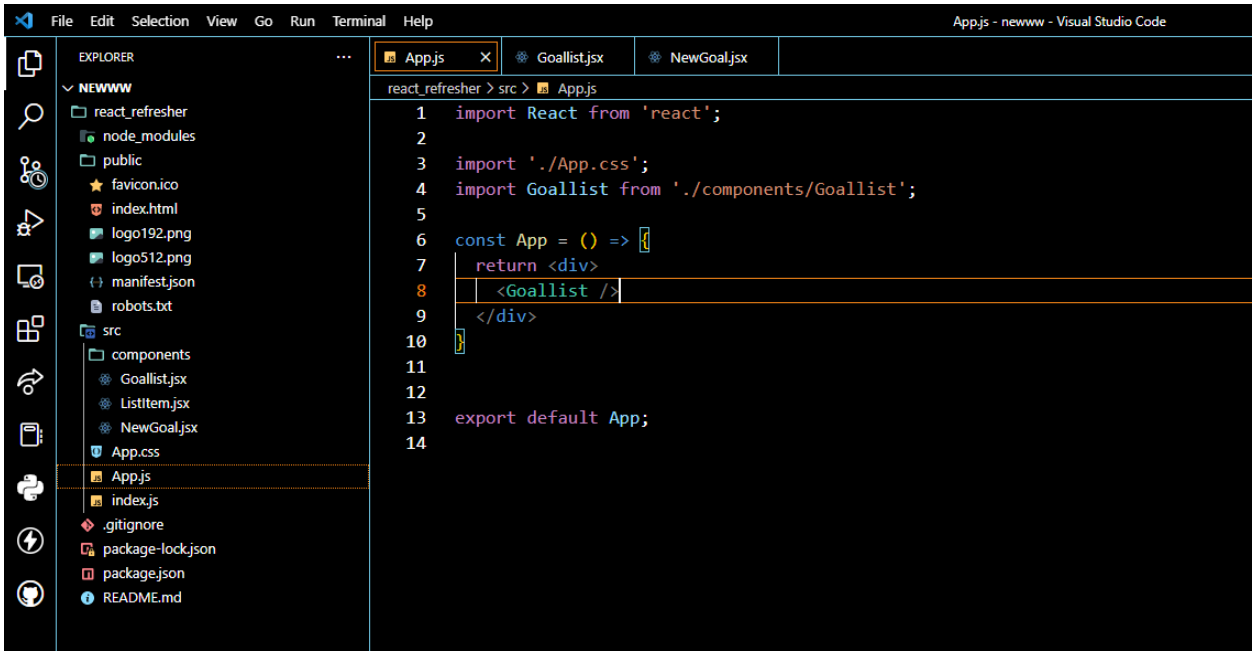
  - To clear the text from the input tag, we must update the enteredText const to an empty string in addGoalhandler.

```
App.js   NewGoal.js ●   NewGoal.css   GoalList.js
src > components > NewGoal > NewGoal.js > NewGoal > addGoalH
5   const NewGoal = props => {
6     const [enteredText, setEnteredText] = useState('');
7
8     const addGoalHandler = event => {
9       event.preventDefault();
10
11      const newGoal = {
12        id: Math.random().toString(),
13        text: enteredText
14      };
15
16      setEnteredText('');
17
18      props.onAddGoal(newGoal);
19    };
20
```

- **Wrap Up!**

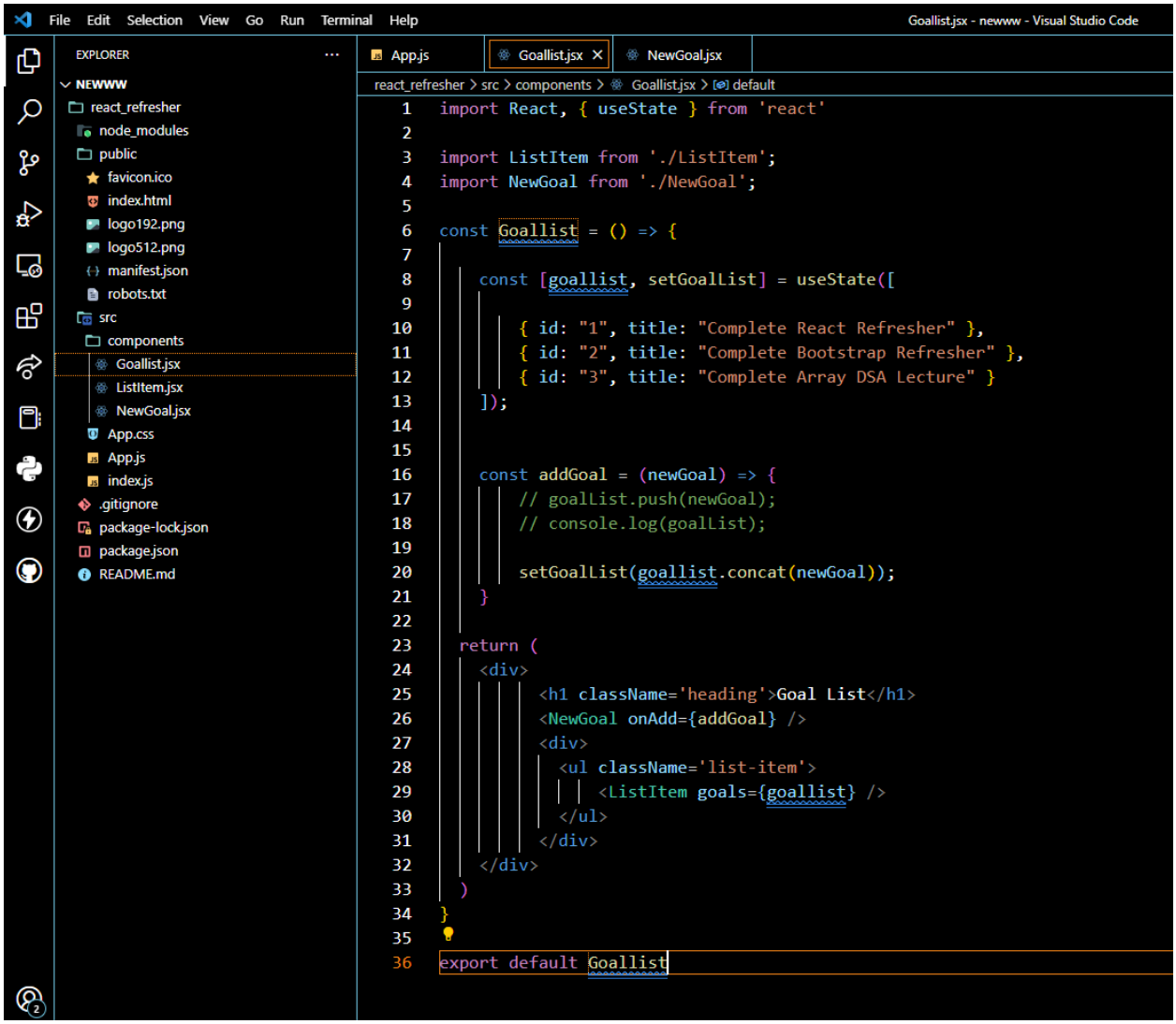
  Congrats for making it till here.🎉

1.


File Edit Selection View Go Run Terminal Help
App.js - newww - Visual Studio Code
EXPLORER
NEWWW
react_refresher
node_modules
public
favicon.ico
index.html
logo192.png
logo512.png
manifest.json
robots.txt
src
components
Goallist.jsx
ListItem.jsx
NewGoal.jsx
App.css
App.js
index.js
.gitignore
package-lock.json
package.json
README.md
App.js
Goallist.jsx
NewGoal.jsx
react_refresher > src > App.js
import React from 'react';
import ./App.css';
import Goallist from ./components/Goallist';
const App = () => {
return
<Goallist />

}
export default App;

2.


File Edit Selection View Go Run Terminal Help
Goallist.jsx - newww - Visual Studio Code
EXPLORER
NEWWW
react_refresher
node_modules
public
favicon.ico
index.html
logo192.png
logo512.png
manifest.json
robots.txt
src
components
Goallist.jsx
ListItem.jsx
NewGoal.jsx
App.css
App.js
index.js
.gitignore
package-lock.json
package.json
README.md
App.js
Goallist.jsx
NewGoal.jsx
react_refresher > src > components > Goallist.jsx > default
import React, { useState } from 'react'
import ListItem from './ListItem';
import NewGoal from './NewGoal';
const Goallist = () => {
const [goallist, setGoalList] = useState([
{ id: "1", title: "Complete React Refresher" },
{ id: "2", title: "Complete Bootstrap Refresher" },
{ id: "3", title: "Complete Array DSA Lecture" }
]);
const addGoal = (newGoal) => {
// goalList.push(newGoal);
// console.log(goalList);
setGoalList(goallist.concat(newGoal));
}
return (

<h1 className='heading'>Goal List</h1>
<NewGoal onAdd={addGoal} />

<ul className='list-item'>
<ListItem goals={goallist} />

)
}
export default Goallist

3.

File Edit Selection View Go Run Terminal Help — NewGoal.jsx - newww - Visual Studio Code

EXPLORER

- NEWWW
  - react_refresher
    - node_modules
    - public
      - favicon.ico
      - index.html
      - logo192.png
      - logo512.png
      - manifest.json
      - robots.txt
    - src
      - components
        - Goallist.jsx
        - ListItem.jsx
        - NewGoal.jsx
      - App.css
      - App.js
      - index.js
    - .gitignore
    - package-lock.json
    - package.json
    - README.md

App.js | Goallist.jsx | NewGoal.jsx

react_refresher > src > components > NewGoal.jsx > NewGoal

```
import React, { useState } from 'react';

const NewGoal = (props) => {

    const [enteredText, setEnteredText] = useState('');

    const onSubmitHandler = (event) => {
        event.preventDefault();

        const newGoal = {
            "id": Math.random().toString(),
            "title": enteredText
        }
        // console.log(newGoal)

        setEnteredText('');

        props.onAdd(newGoal);
    }

    const onChangeHandler = (event) => {
        setEnteredText(event.target.value);
    }

    return <form className='form' onSubmit={onSubmitHandler}>
        <input type='text' placeholder='Type Your New Goal' value={enteredText} onChange={onChangeHandler} required/>
        <button>Add Goal</button>
    </form>
}

export default NewGoal;
```