React JS

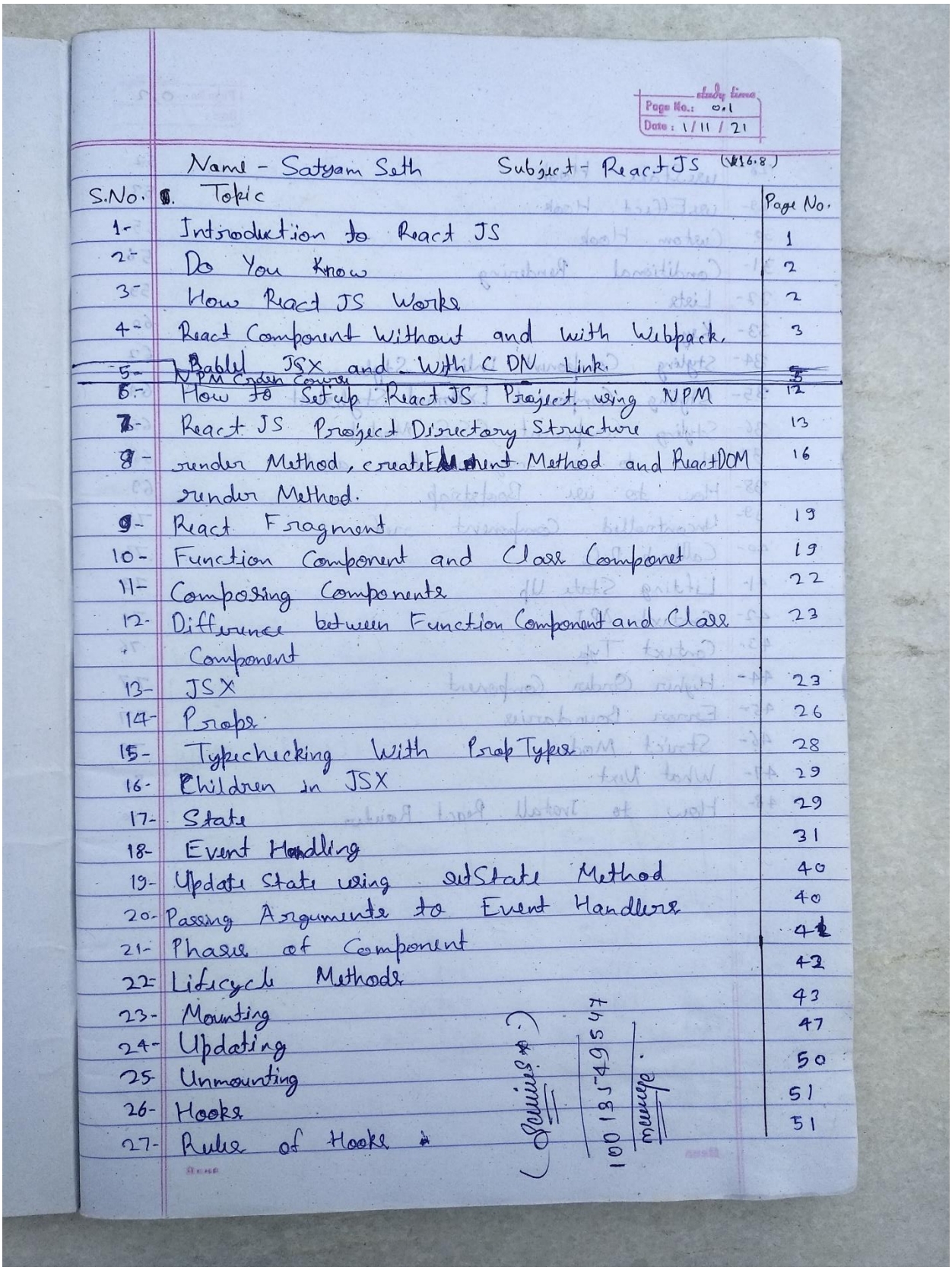
GEEKYSHOWS YOUTUBE CHANNEL LEARNING NOTES

Source Code - https://github.com/satyam-seth-learnings/reactjs_learning/tree/master/Geekyshows

YouTube Link - https://youtube.com/playlist?list=PLbGui_ZYuhignjLLXTJWkRJKN-SgAqClL

SATYAM SETH

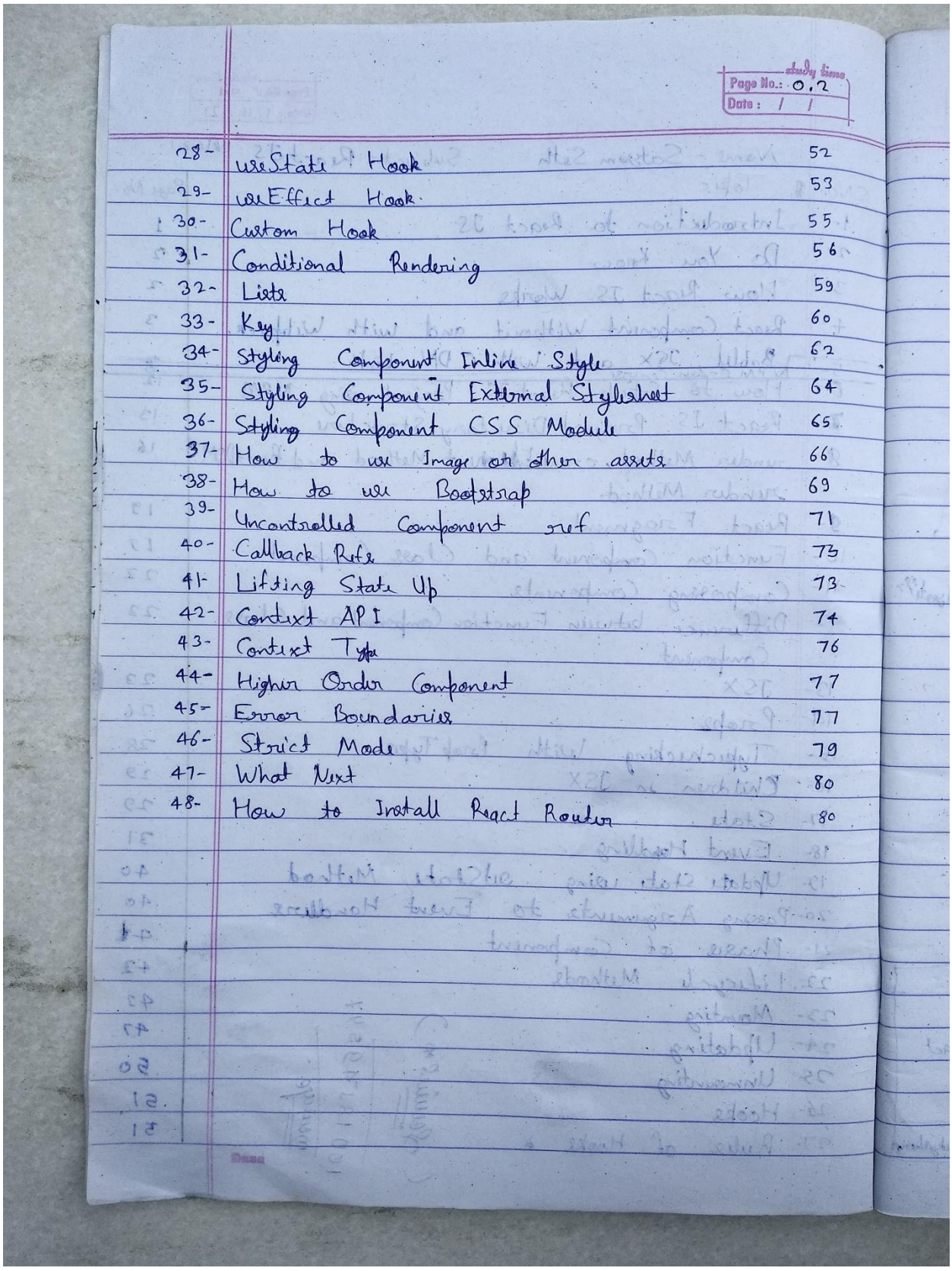
04-09-2022



study time
Page No.: 0.1
Date: 1/11/21

Name - Satyam Seth Subject - React JS (#16.8)

(Genius) 100 18 49 47 message



Page No.: 1
Date: 2/11/21

What is ReactJS?

- React JS is a JavaScript Library for building front end application or user interface (UI).
- React JS allows us to create reusable UI components.
- It is created by Facebook.

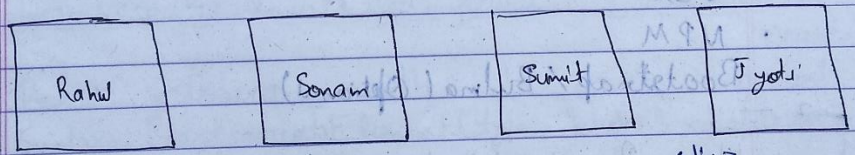
Component:- Components are the building blocks of any React apps.

Advantages of ReactJS -

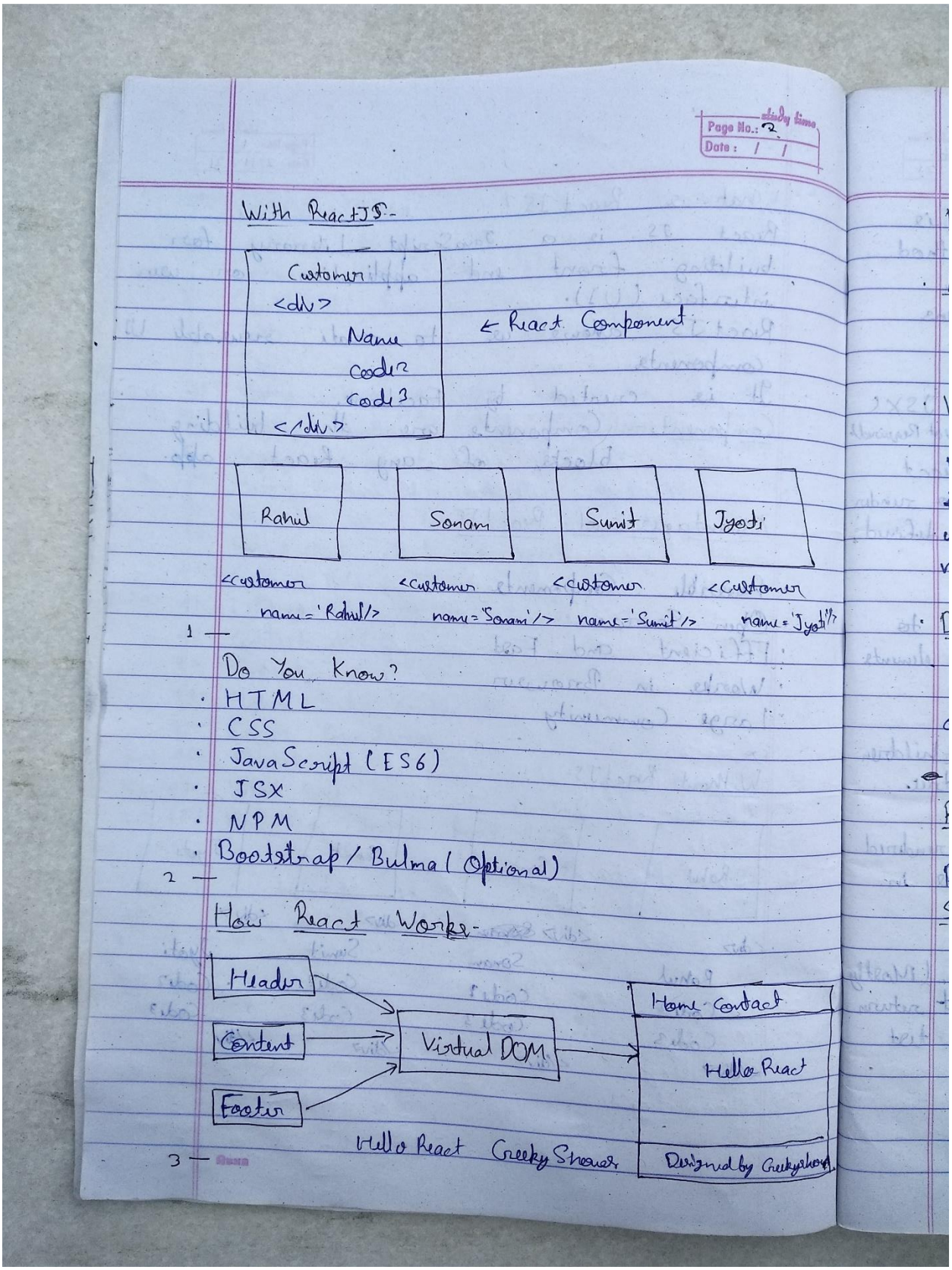
- Reusable Components
- Open Source
- Efficient and Fast
- Works in Browser
- Large Community

Ex

Without ReactJS -

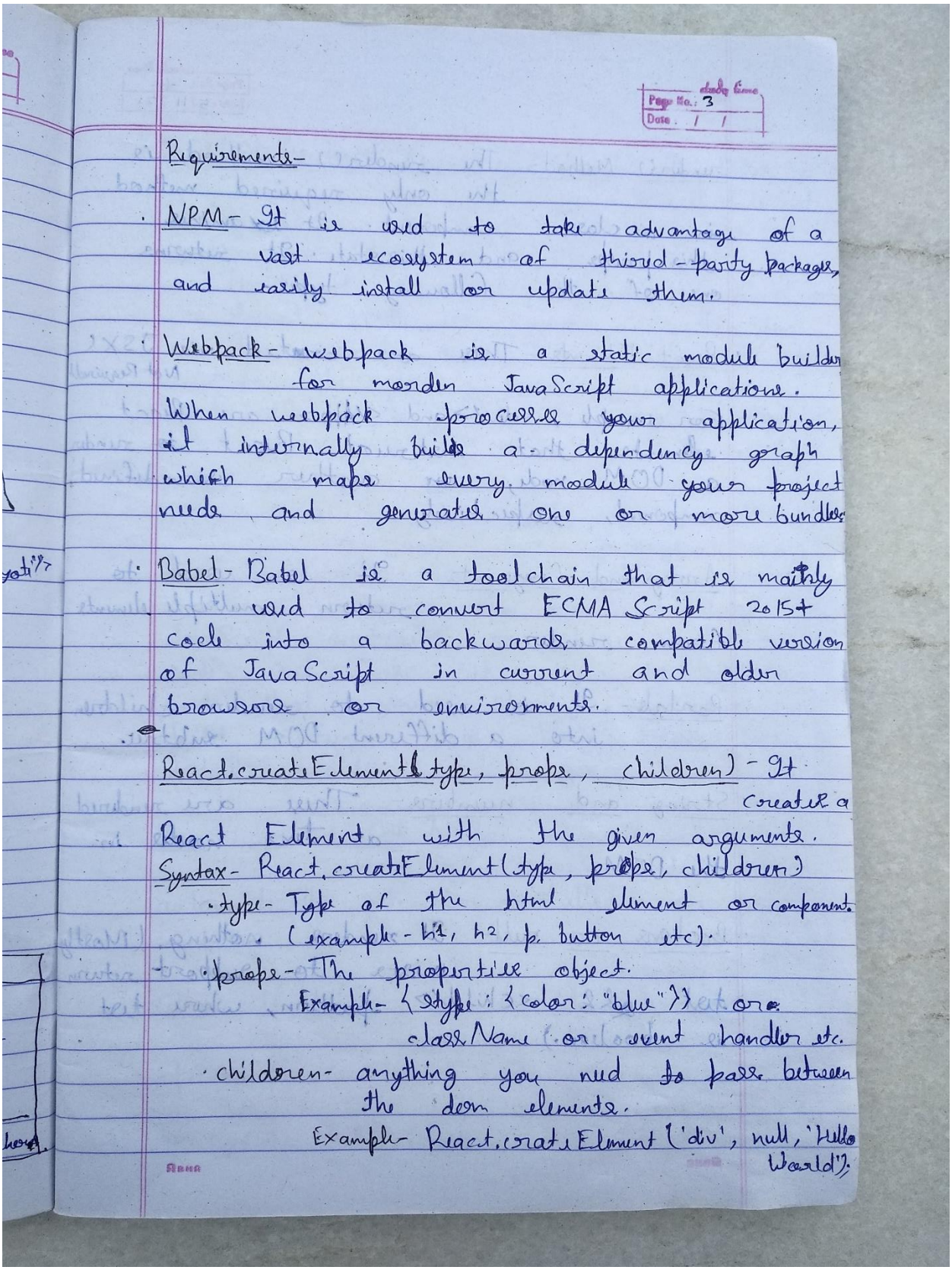


Rahul Sonam Sumit Jyoti
Code2 Code2 Code3 Code2
Code3 Code3 Code3


Page No.: 2
Date : / /
With ReactJS-
Customer

Name
code2
code3

← React Component
Rahul
Sonam
Sumit
Jyoti
<customer name='Rahul'/>
<customer name='Sonam'/>
<customer name='Sumit'/>
<Customer name='Jyoti'/>
1 —
Do You Know?
· HTML
· CSS
· JavaScript (ES6)
· JSX
· NPM
· Bootstrap / Bulma (Optional)
2 —
How React Works-
Header
Content
Footer
Virtual DOM
Home Contact
Hello React
Designed by Geekyshows
Hello React Geeky Shows
3 —



Page No. 3
Date / /

Requirements-

- NPM - It is used to take advantage of a vast ecosystem of third-party packages, and easily install or update them.

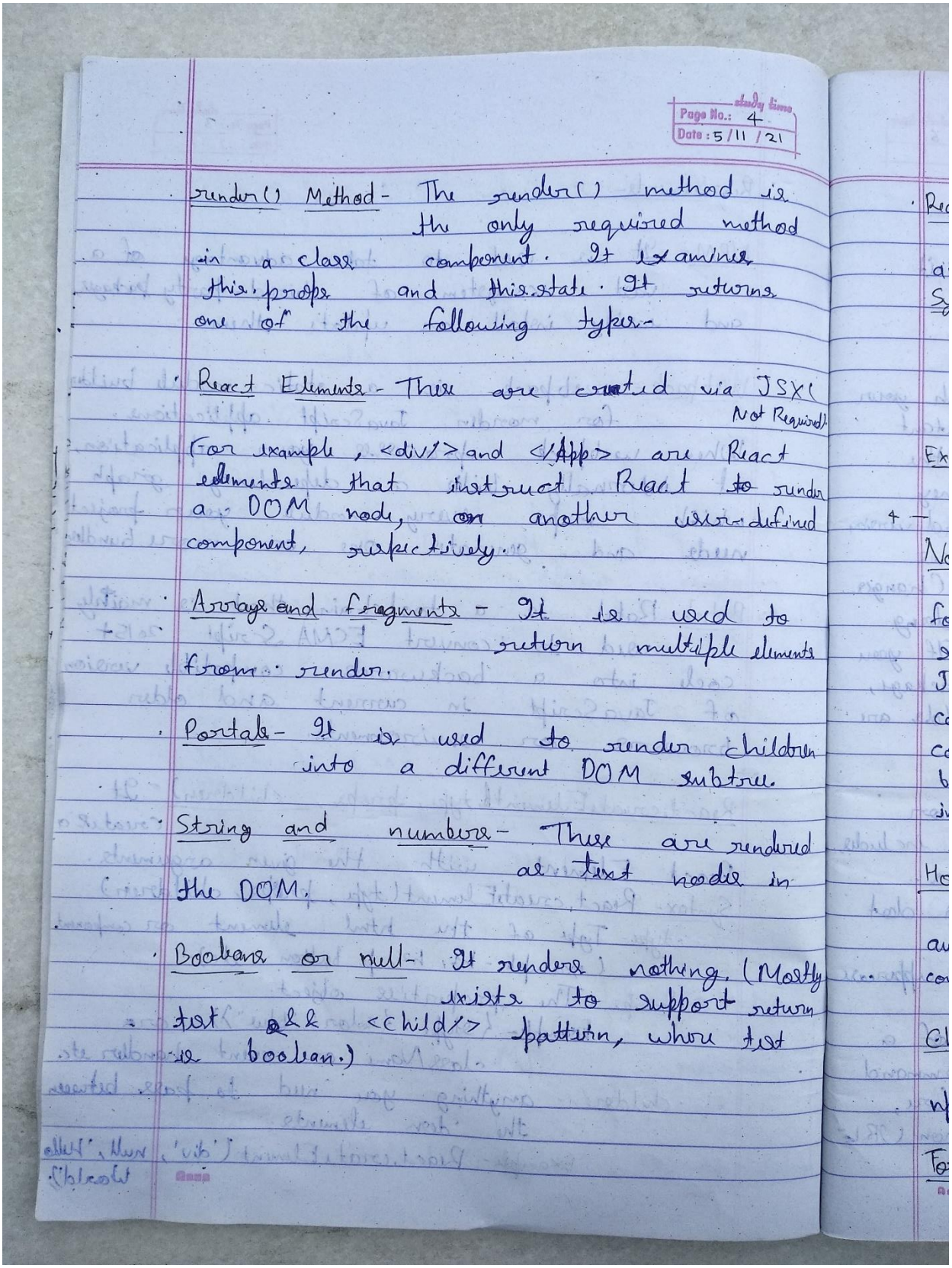
- Webpack - webpack is a static module builder for modern JavaScript applications. When webpack processes your application, it internally builds a dependency graph which maps every module your project needs and generates one or more bundles.

- Babel - Babel is a toolchain that is mainly used to convert ECMA Script 2015+ code into a backwards compatible version of JavaScript in current and older browsers or environments.

React.createElement(type, props, children) - It creates a React Element with the given arguments.

Syntax - React.createElement(type, props, children)

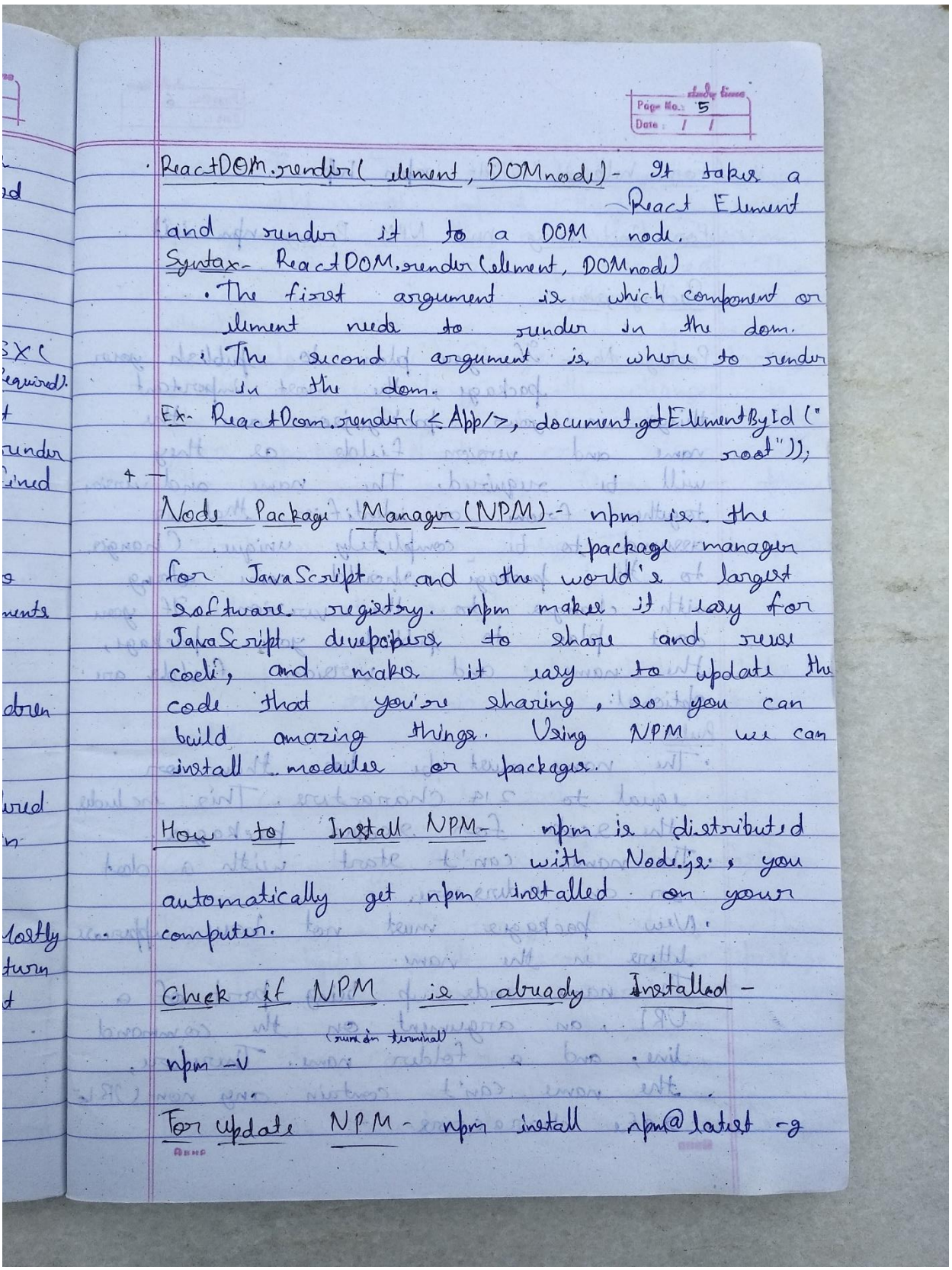
- type - Type of the html element or component. (example - h1, h2, p, button etc).
- props - The properties object.
  Example - {style: {color: "blue"}} or className or event handler etc.
- children - anything you need to pass between the dom elements.
  Example - React.createElement('div', null, 'Hello World');



Page No.: 4
Date: 5/11/21

render() Method - The render() method is the only required method in a class component. It examines this.props and this.state. It returns one of the following types -

- React Elements - These are created via JSX (Not Required). For example, and <App/> are React elements that instruct React to render a DOM node, or another user-defined component, respectively.
- Arrays and fragments - It is used to return multiple elements from render.
- Portals - It is used to render children into a different DOM subtree.
- String and numbers - These are rendered as text nodes in the DOM.
- Boolean or null - It renders nothing. (Mostly exists to support return test && <child/> pattern, where test is boolean.)



Page No.: 5
Date: / /

- ReactDOM.render(element, DOMnode) - It takes a React Element and render it to a DOM node.

Syntax - ReactDOM.render(element, DOMnode)

- The first argument is which component or element needs to render in the dom.
- The second argument is where to render in the dom.

Ex. ReactDom.render(<App/>, document.getElementById("root"));

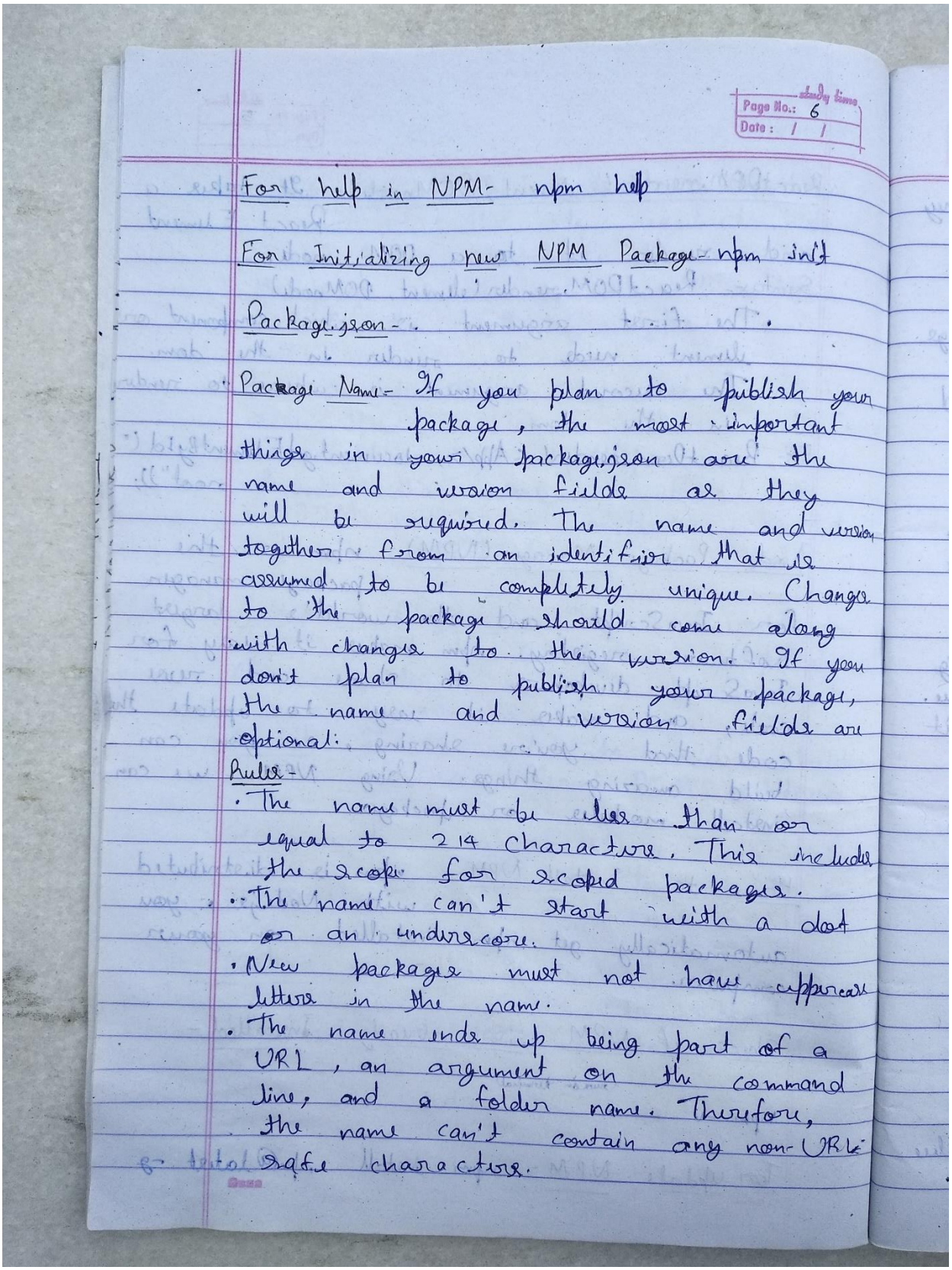
Node Package Manager (NPM) - npm is the package manager for JavaScript and the world's largest software registry. npm makes it easy for JavaScript developers to share and reuse code, and makes it easy to update the code that you're sharing, so you can build amazing things. Using NPM we can install modules or packages.

How to Install NPM - npm is distributed with Node.js, you automatically get npm installed on your computer.

Chek if NPM is already Installed - (run in terminal)

npm -v

For update NPM - npm install npm@latest -g



Page No.: 6
Date: / /

For help in NPM- npm help

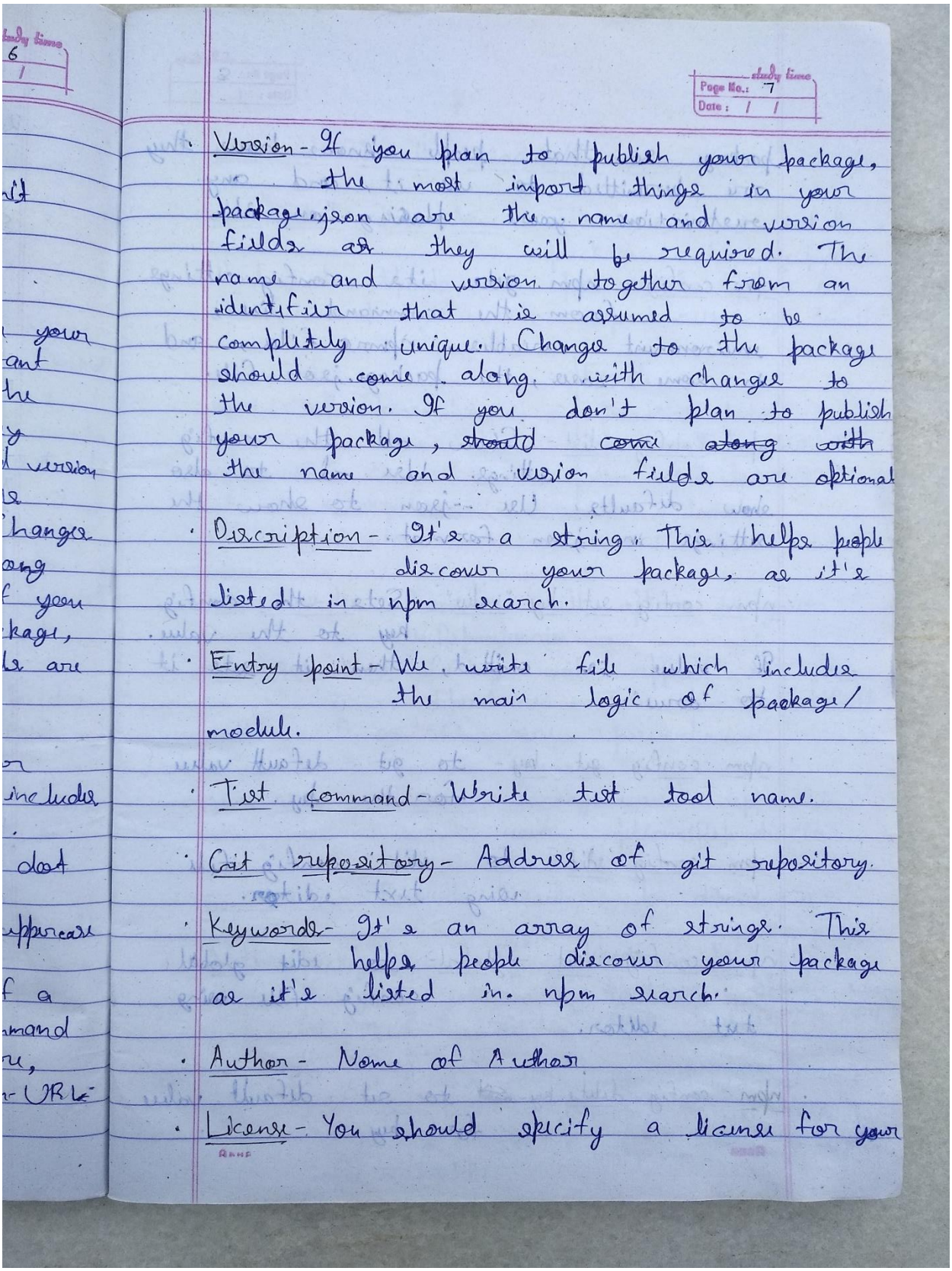
For Initializing new NPM Package- npm init

Package.json-

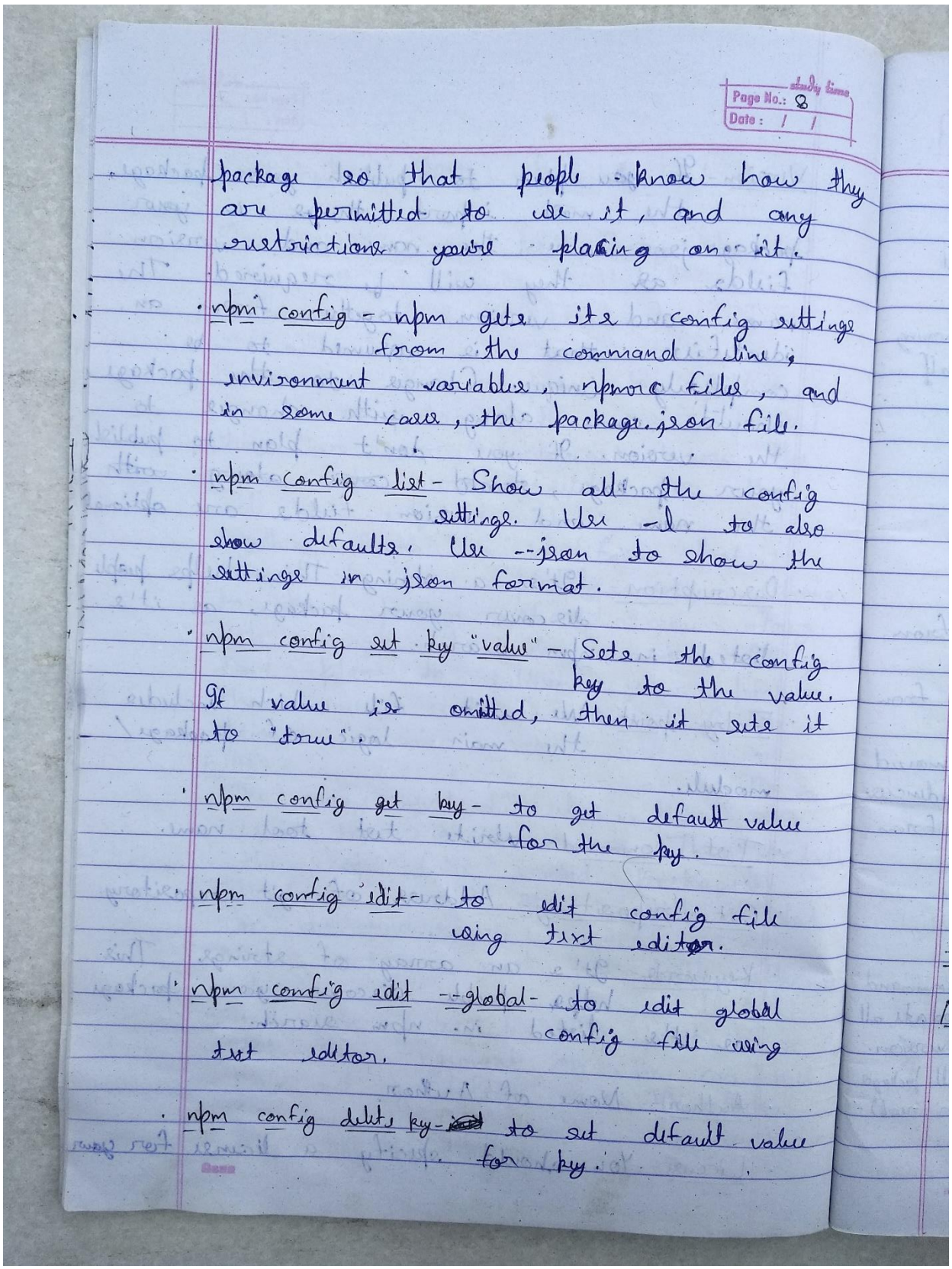
Package Name- If you plan to publish your package, the most important things in your package.json are the name and version fields as they will be required. The name and version together form an identifier that is assumed to be completely unique. Changes to the package should come along with changes to the version. If you don't plan to publish your package, the name and version fields are optional.

Rules-

- The name must be less than or equal to 214 characters. This includes the scope for scoped packages.
- The name can't start with a dot or an underscore.
- New packages must not have uppercase letters in the name.
- The name ends up being part of a URL, an argument on the command line, and a folder name. Therefore, the name can't contain any non-URL-safe characters.

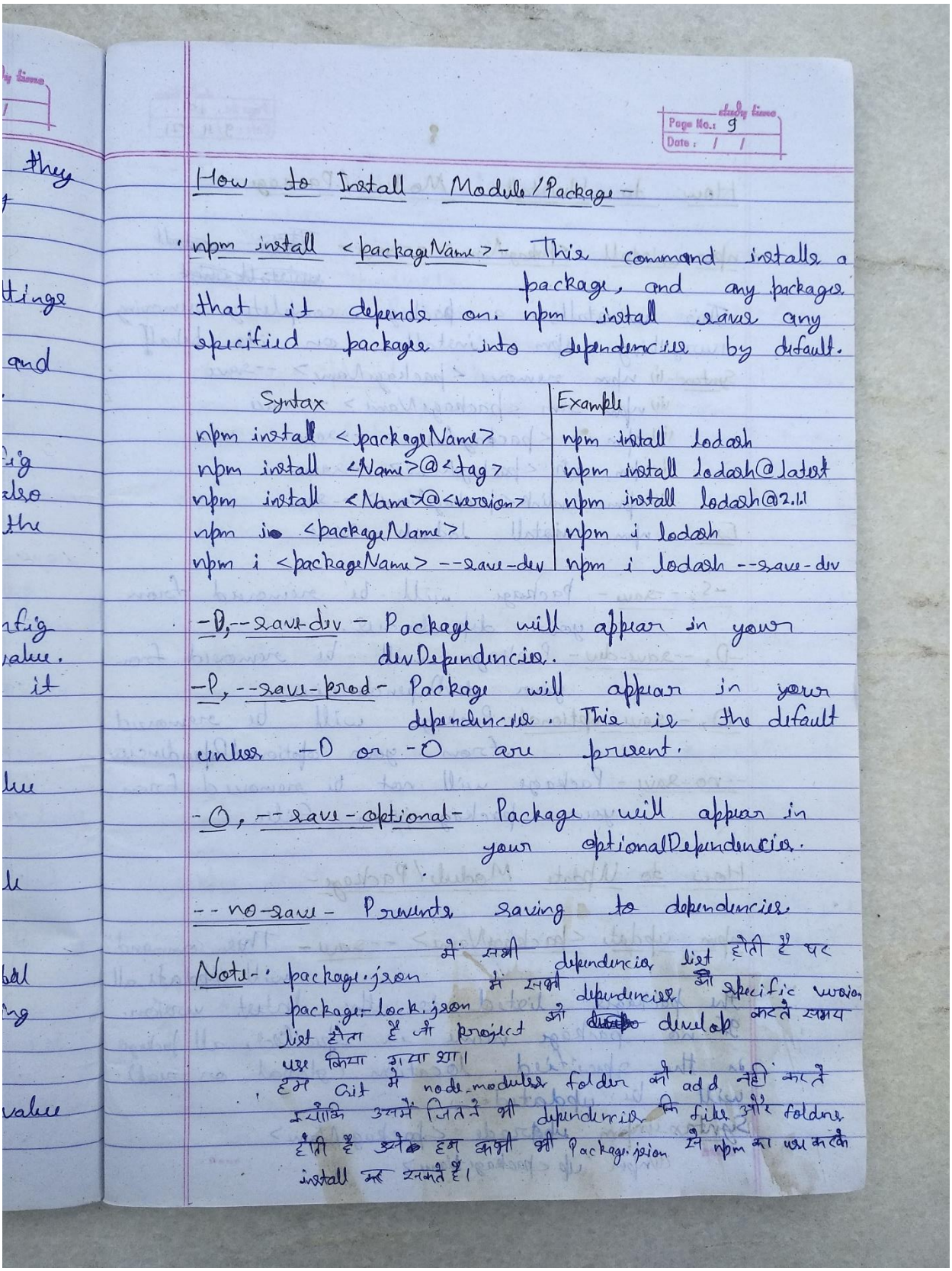


- Version - If you plan to publish your package, the most import things in your package.json are the name and version fields as they will be required. The name and version together from an identifier that is assumed to be completely unique. Changes to the package should come along with changes to the version. If you don't plan to publish your package, ~~should come along with~~ the name and version fields are optional.
- Description - It's a string. This helps people discover your package, as it's listed in npm search.
- Entry point - We write file which includes the main logic of package/ module.
- Test command - Write test tool name.
- Git repository - Address of git repository.
- Keywords - It's an array of strings. This helps people discover your package as it's listed in npm search.
- Author - Name of Author
- License - You should specify a license for your



package so that people know how they are permitted to use it, and any restrictions you're placing on it.

- npm config - npm gets its config settings from the command line, environment variables, npmrc files, and in some cases, the package.json file.
- npm config list - Show all the config settings. Use -l to also show defaults. Use --json to show the settings in json format.
- npm config set key "value" - Sets the config key to the value. If value is omitted, then it sets it to "true".
- npm config get key - to get default value for the key.
- npm config edit - to edit config file using text editor.
- npm config edit --global - to edit global config file using text editor.
- npm config delete key - to set default value for key.



Page No.: 9
Date: / /

How to Install Module / Package -

- npm install <packageName> - This command installs a package, and any packages that it depends on. npm install saves any specified packages into dependencies by default.

Syntax	Example
npm install <packageName>	npm install lodash
npm install <Name>@<tag>	npm install lodash@latest
npm install <Name>@<version>	npm install lodash@2.1.1
npm i <packageName>	npm i lodash
npm i <packageName> --save-dev	npm i lodash --save-dev

-D, --save-dev - Package will appear in your devDependencies.

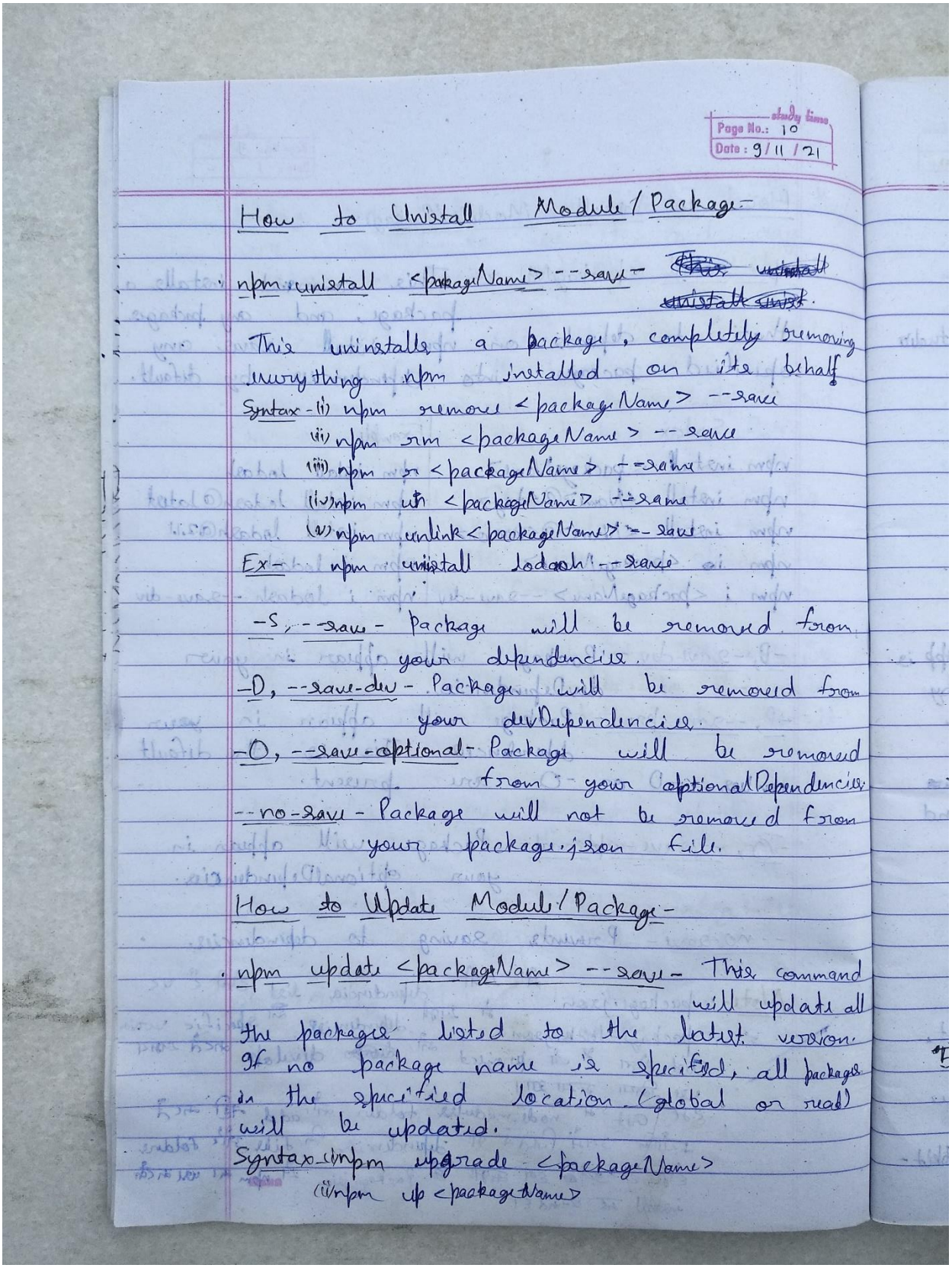
-P, --save-prod - Package will appear in your dependencies. This is the default unless -D or -O are present.

-O, --save-optional - Package will appear in your optionalDependencies.

--no-save - Prevents saving to dependencies.

Note - • package.json में सभी dependencies list होती है पर package-lock.json में सभी dependencies की specific version list होता है जो project को develop करते समय use किया गया था।

• हम Git में node_modules folder को add नहीं करते क्योंकि उसमें जितने भी dependencies की files और folders होती है उन्हें हम सभी को Package.json से npm का use करके install कर सकते है।



Page No.: 10
Date: 9/11/21

How to Uninstall Module / Package -

npm uninstall <packageName> --save - ~~uninstall uninstall shrist~~.

This uninstalls a package, completely removing everything npm installed on its behalf.

Syntax - (i) npm remove <packageName> --save
(ii) npm rm <packageName> --save
(iii) npm r <packageName> --save
(iv) npm un <packageName> --save
(v) npm unlink <packageName> --save

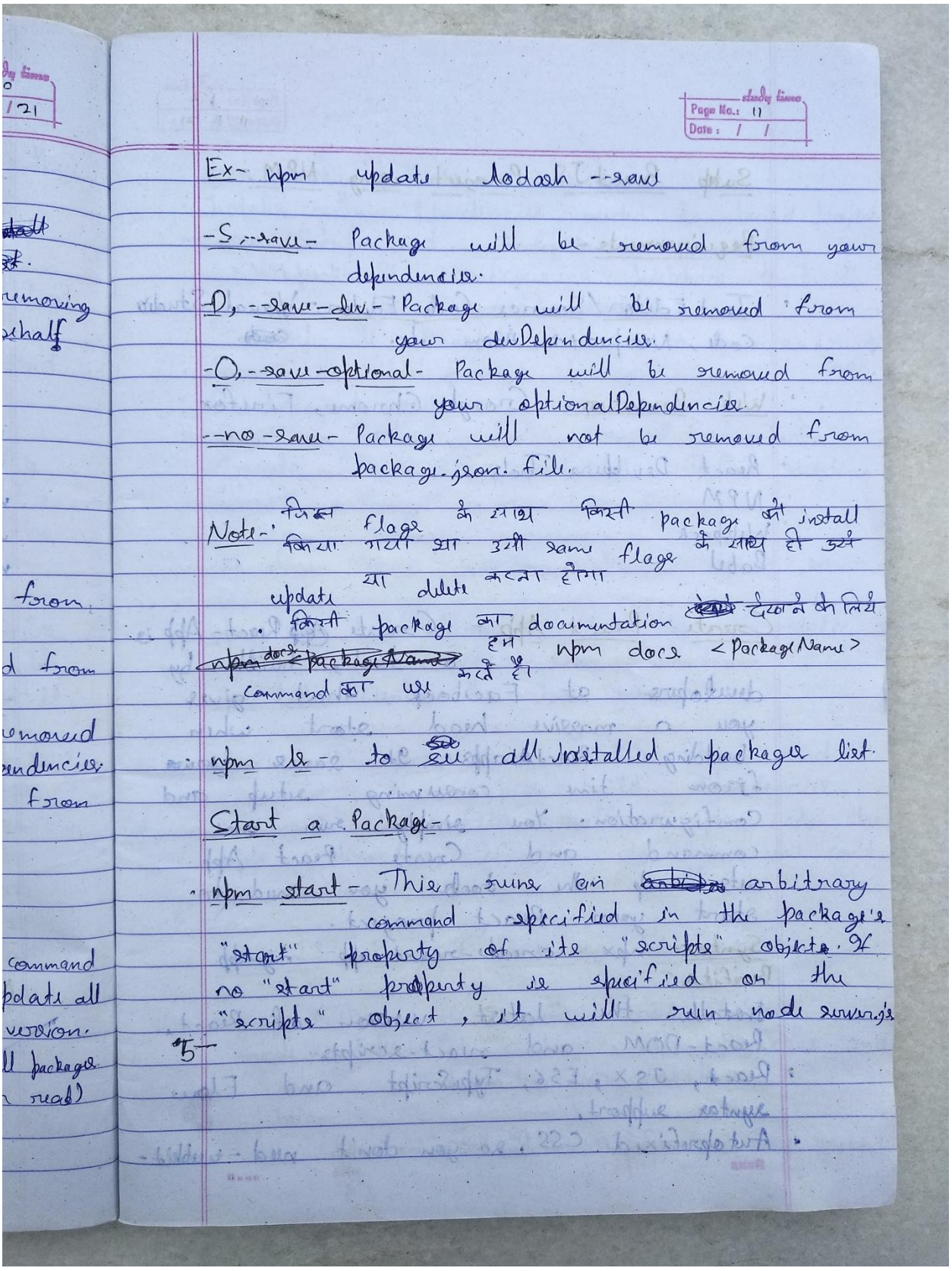
Ex - npm uninstall lodash --save

-S, --save - Package will be removed from your dependencies.
-D, --save-dev - Package will be removed from your devDependencies.
-O, --save-optional - Package will be removed from your optionalDependencies.
--no-save - Package will not be removed from your package.json file.

How to Update Module / Package -

npm update <packageName> --save - This command will update all the packages listed to the latest version. If no package name is specified, all packages in the specified location (global or root) will be updated.

Syntax - (i) npm upgrade <packageName>
(ii) npm up <packageName>



Ex- npm update lodash --save

-S, --save - Package will be removed from your dependencies.

-D, --save-dev - Package will be removed from your devDependencies.

-O, --save-optional - Package will be removed from your optionalDependencies.

--no-save - Package will not be removed from package.json file.

Note - जिस flag के साथ किसी package को install किया गया था उसी same flag के साथ ही उसे update या delete करना होगा।

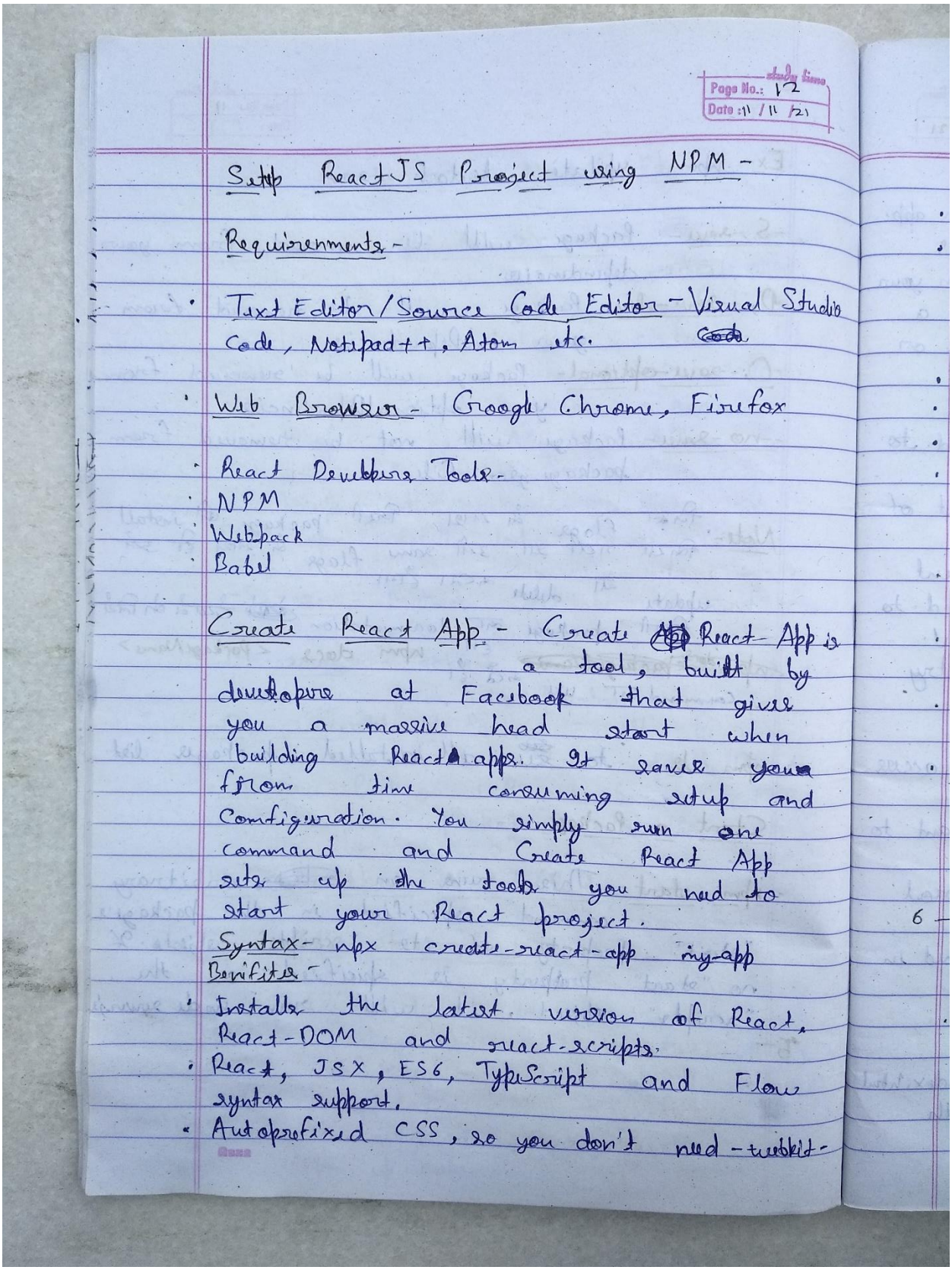
- किसी package का documentation देखने के लिये ~~npm docs <packageName>~~ हम npm docs <PackageName> command का use करते है।

- npm ls - to see all installed packages list.

Start a Package -

- npm start - This runs an arbitrary command specified in the package's "start" property of its "scripts" object. If no "start" property is specified on the "scripts" object, it will run node server.js

5-



Page No.: 12
Date: 11/11/21

Setup ReactJS Project using NPM -

Requirements -

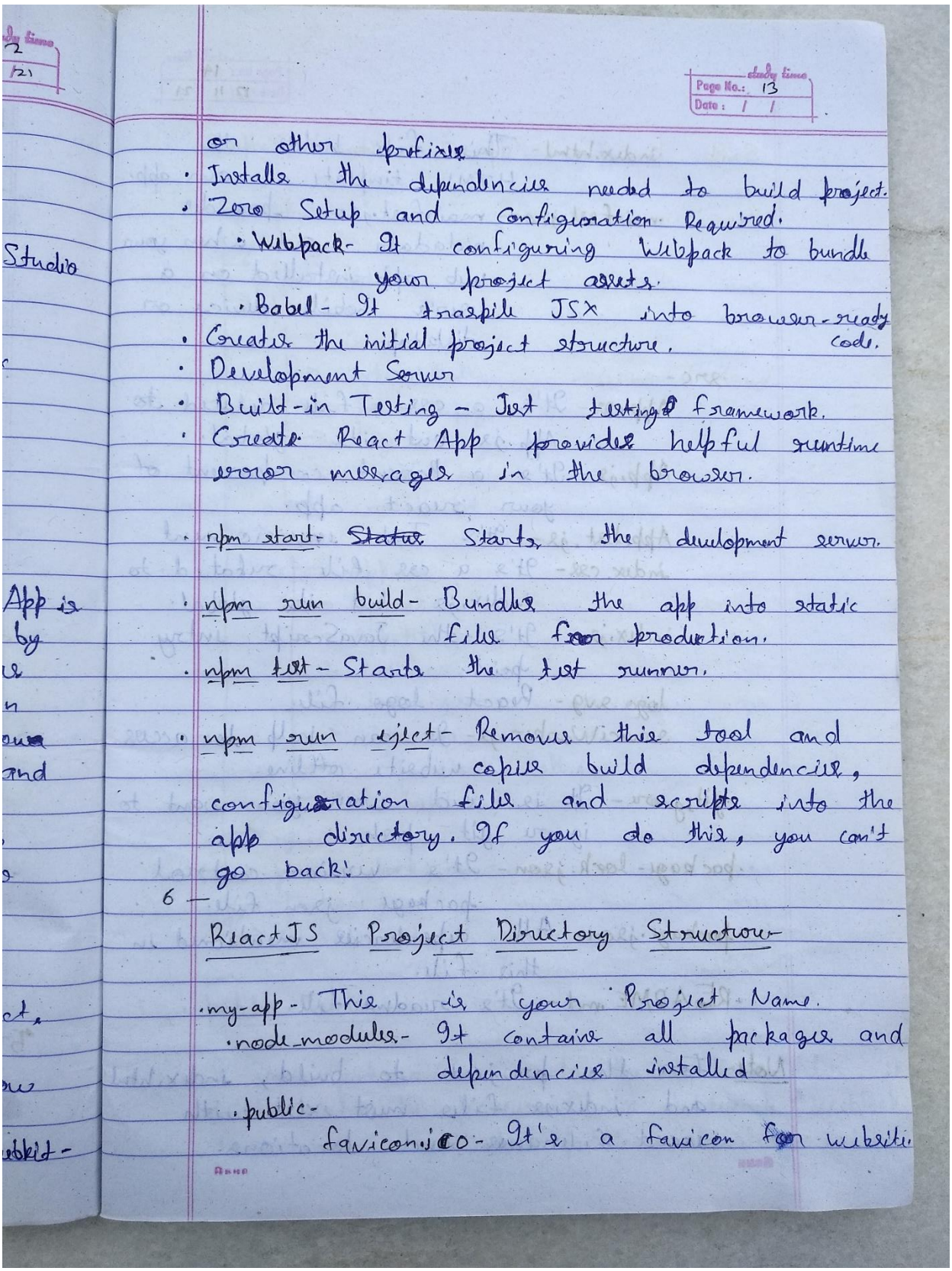
- Text Editor / Source Code Editor - Visual Studio Code, Notepad++, Atom etc.
- Web Browser - Google Chrome, Firefox
- React Developers Tools -
- NPM
- Webpack
- Babel

Create React App - Create React App is a tool, built by developers at Facebook that gives you a massive head start when building React apps. It saves you from time consuming setup and configuration. You simply run one command and Create React App sets up the tools you need to start your React project.

Syntax - npx create-react-app my-app

Benifits -

- Installs the latest version of React, React-DOM and react-scripts.
- React, JSX, ES6, TypeScript and Flow syntax support.
- Autoprefixed CSS, so you don't need -webkit-



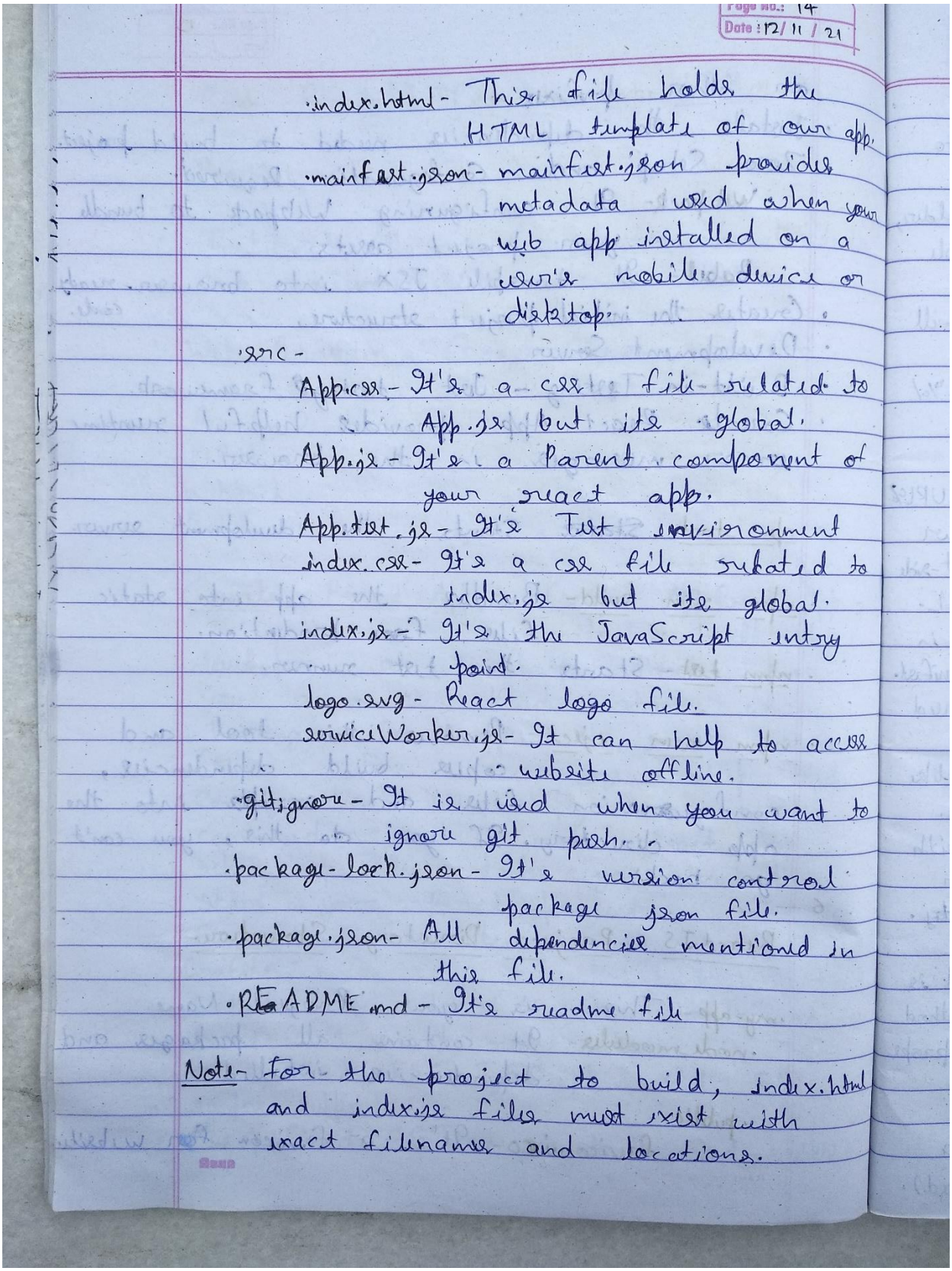
or other prefixes
- Installs the dependencies needed to build project.
- Zero Setup and Configuration Required.
  - Webpack- It configuring Webpack to bundle your project assets.
  - Babel- It transpile JSX into browser-ready code.
- Creates the initial project structure.
- Development Server
- Built-in Testing - Jest testing framework.
- Create React App provides helpful runtime error messages in the browser.

- npm start- ~~Status~~ Starts the development server.
- npm run build- Bundles the app into static file for production.
- npm test - Starts the test runner.
- npm run eject - Removes this tool and copies build dependencies, configuration file and scripts into the app directory. If you do this, you can't go back!

6 —

ReactJS Project Directory Structure

- my-app - This is your Project Name.
  - node_modules - It contains all packages and dependencies installed
  - public -
    - favicon.ico - It's a favicon for website

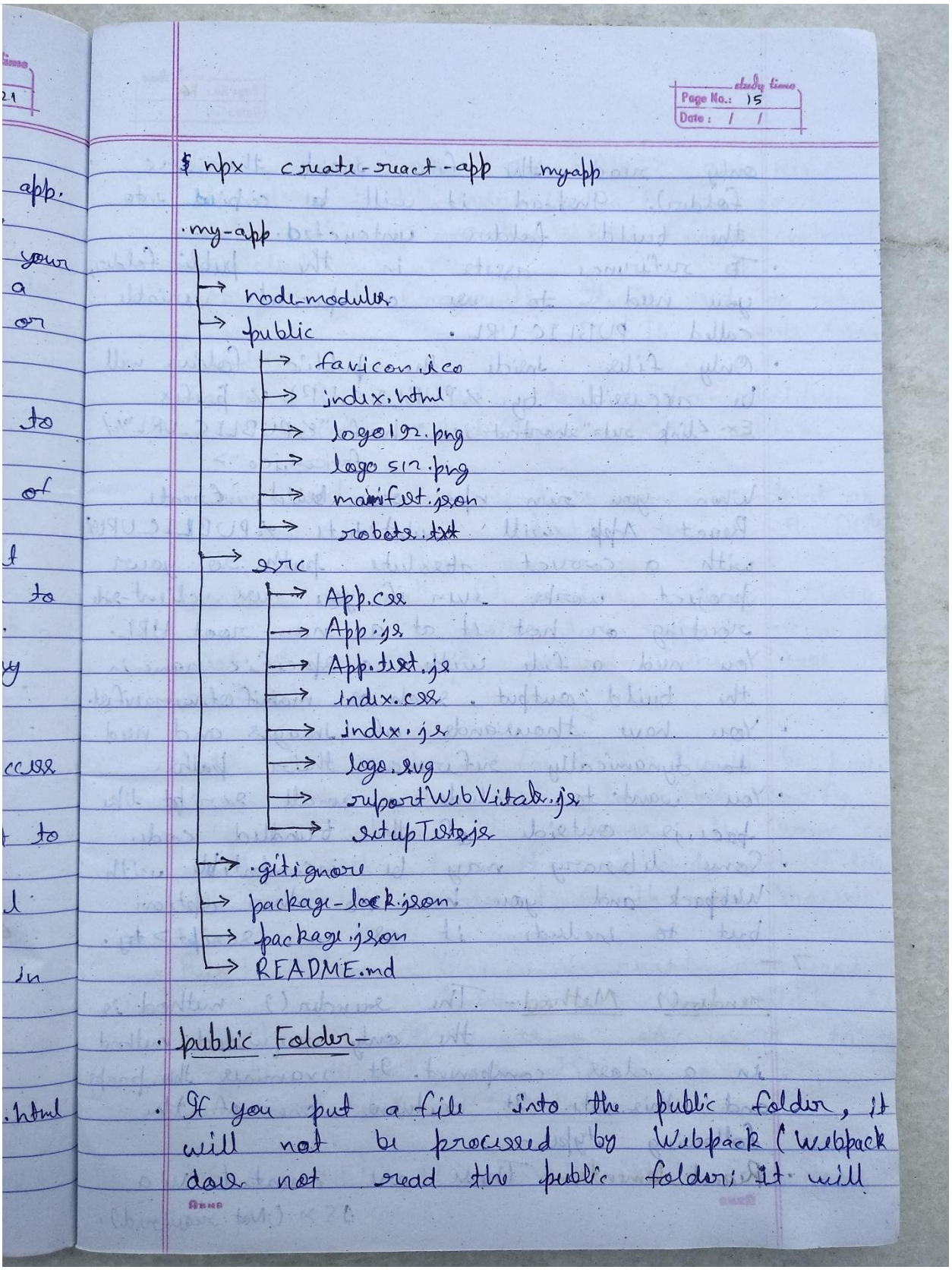


Page No.: 14
Date: 12/11/21

- index.html - This file holds the HTML template of our app.
- manifest.json - manifest.json provides metadata used when your web app installed on a user's mobile device or desktop.

- src -
  - App.css - It's a css file related to App.js but its global.
  - App.js - It's a Parent component of your react app.
  - App.test.js - It's Test environment
  - index.css - It's a css file related to index.js but its global.
  - index.js - It's the JavaScript entry point.
  - logo.svg - React logo file.
  - serviceWorker.js - It can help to access website offline.
- .gitignore - It is used when you want to ignore git push.
- package-lock.json - It's version control package json file.
- package.json - All dependencies mentioned in this file.
- README.md - It's readme file

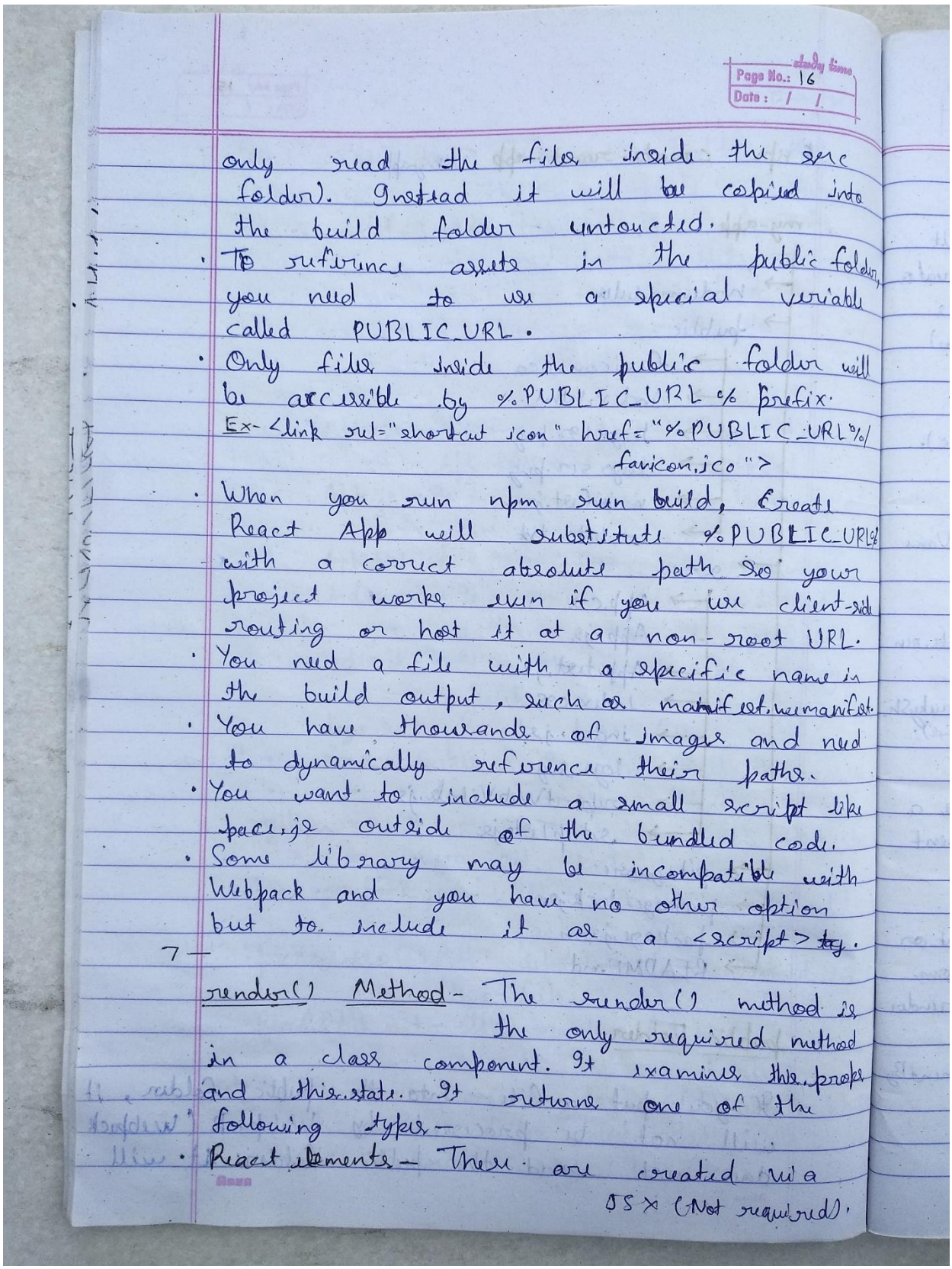
Note - For the project to build, index.html and index.js files must exist with exact filenames and locations.



$ npx create-react-app myapp

- my-app
  - node_modules
  - public
    - favicon.ico
    - index.html
    - logo192.png
    - logo512.png
    - manifest.json
    - robots.txt
  - src
    - App.css
    - App.js
    - App.test.js
    - index.css
    - index.js
    - logo.svg
    - reportWebVitals.js
    - setupTests.js
  - .gitignore
  - package-lock.json
  - package.json
  - README.md

- **public Folder-**
- If you put a file into the public folder, it will not be processed by Webpack (Webpack does not read the public folder; it will



only read the files inside the src folder). Instead it will be copied into the build folder untouched.

- To reference assets in the public folder, you need to use a special variable called PUBLIC_URL.
- Only files inside the public folder will be accessible by %PUBLIC_URL% prefix.

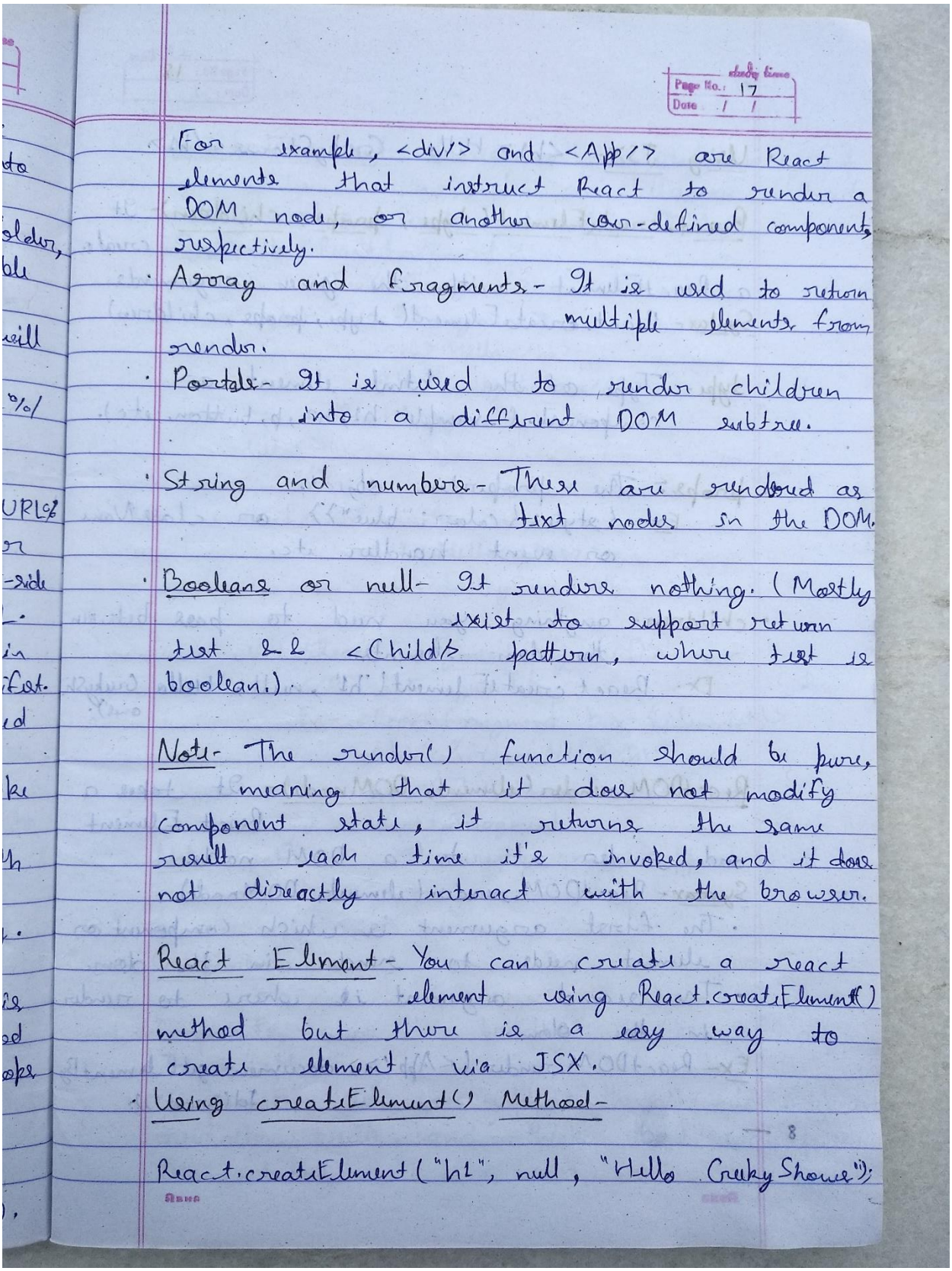
Ex- <link rel="shortcut icon" href="%PUBLIC_URL%/favicon.ico">

- When you run npm run build, Create React App will substitute %PUBLIC_URL% with a correct absolute path so your project works even if you use client-side routing or host it at a non-root URL.
- You need a file with a specific name in the build output, such as manifest.webmanifest.
- You have thousands of images and need to dynamically reference their paths.
- You want to include a small script like pace.js outside of the bundled code.
- Some library may be incompatible with Webpack and you have no other option but to include it as a <script> tag.

7 —

render() Method - The render() method is the only required method in a class component. It examines this.props and this.state. It returns one of the following types —

- React elements — These are created via JSX (Not required).



For example, and <App/> are React elements that instruct React to render a DOM node, or another user-defined component, respectively.

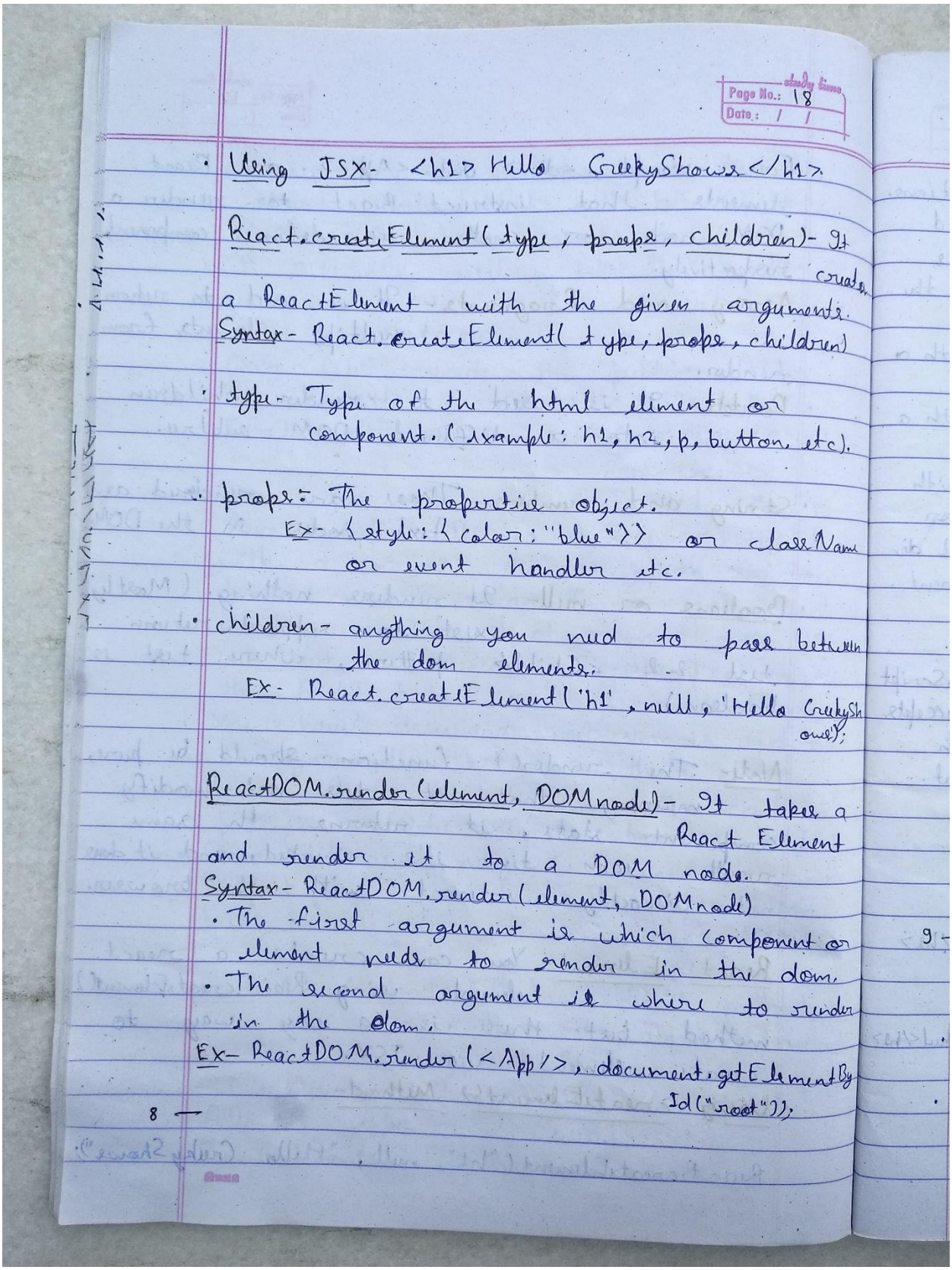
- Array and fragments - It is used to return multiple elements from render.
- Portals - It is used to render children into a different DOM subtree.
- String and numbers - These are rendered as text nodes in the DOM.
- Booleans or null - It renders nothing. (Mostly exist to support return test && <Child/> pattern, where test is boolean.)

Note - The render() function should be pure, meaning that it does not modify component state, it returns the same result each time it's invoked, and it does not directly interact with the browser.

React Element - You can create a react element using React.createElement() method but there is a easy way to create element via JSX.

- Using createElement() Method -

```
React.createElement("h1", null, "Hello Geeky Shows");
```



Page No.: 18
Date: / /

- Using JSX- <h1> Hello GeekyShows </h1>

React.createElement(type, props, children)- It creates a ReactElement with the given arguments.
Syntax- React.createElement(type, props, children)

- type- Type of the html element or component. (example: h1, h2, p, button, etc).

- props:- The properties object.
  Ex- {style: {color: "blue"}} or className or event handler etc.

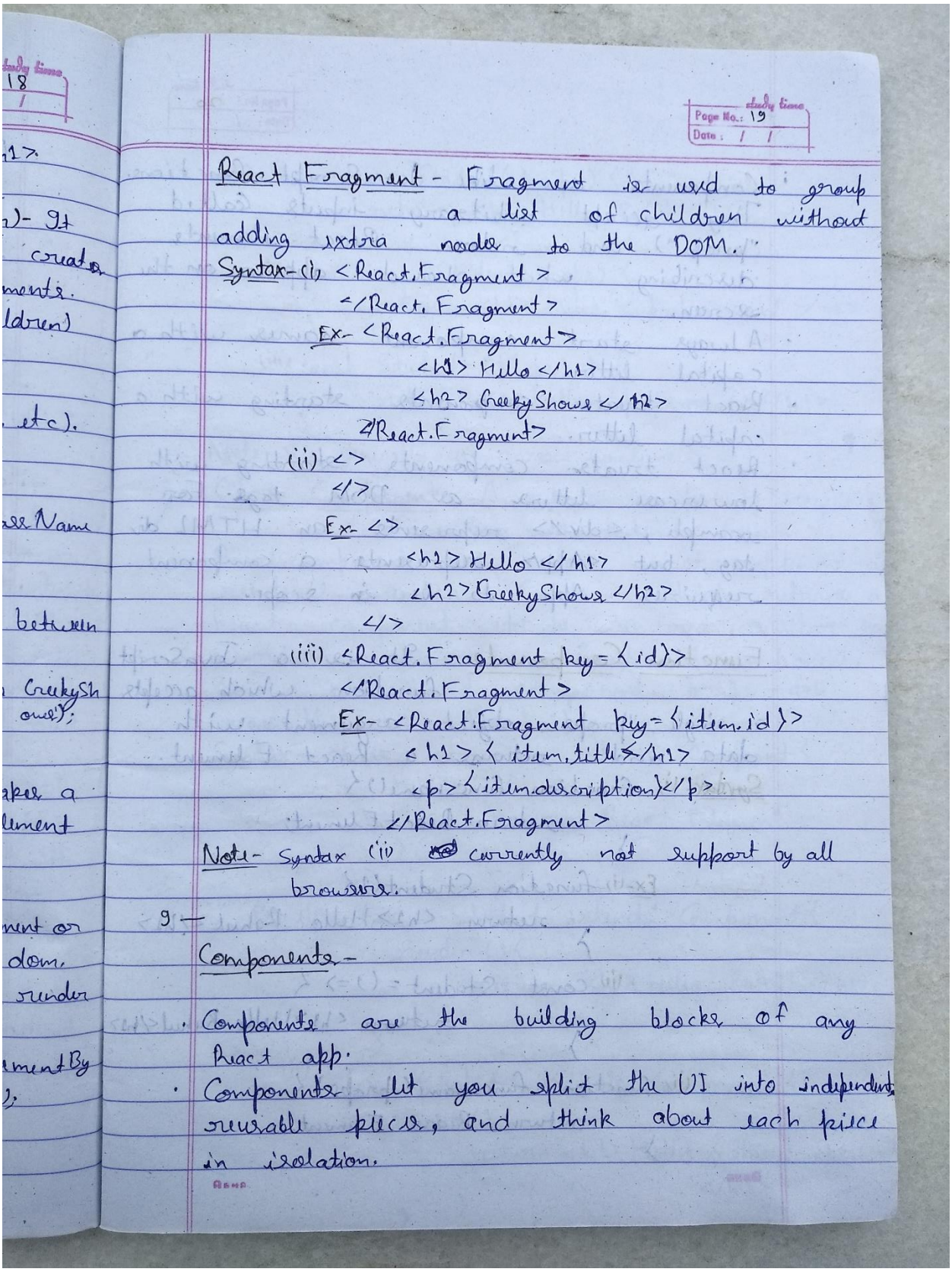
- children- anything you need to pass between the dom elements.
  Ex- React.createElement('h1', null, 'Hello GeekyShows');

ReactDOM.render(element, DOMnode)- It takes a React Element and render it to a DOM node.
Syntax- ReactDOM.render(element, DOMnode)
- The first argument is which component or element needs to render in the dom.
- The second argument is where to render in the dom.

Ex- ReactDOM.render(<App />, document.getElementById("root"));

8 —

9 —



React Fragment - Fragment is used to group a list of children without adding extra nodes to the DOM.

Syntax- (i) <React.Fragment>
</React.Fragment>

Ex- <React.Fragment>
<h1> Hello </h1>
<h2> GeekyShows </h2>
</React.Fragment>

(ii) <>
</>

Ex- <>
<h1> Hello </h1>
<h2> GeekyShows </h2>
</>

(iii) <React.Fragment key={id}>
</React.Fragment>

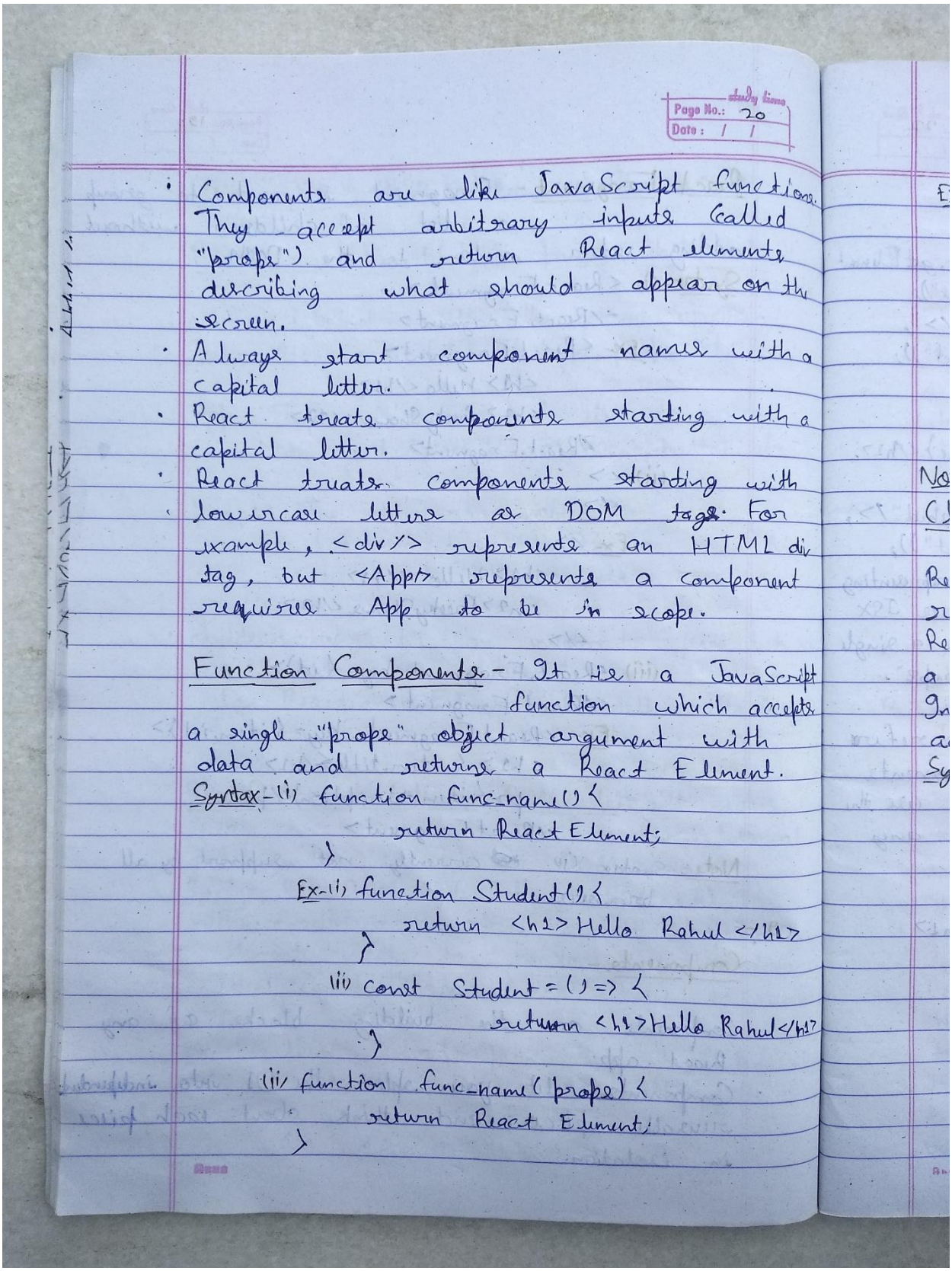
Ex- <React.Fragment key={item.id}>
<h1> {item.title} </h1>
 {item.description}
</React.Fragment>

Note- Syntax (ii) currently not support by all browsers.

9 —

Components -

- Components are the building blocks of any React app.
- Components let you split the UI into independent reusable pieces, and think about each piece in isolation.



- Components are like JavaScript functions. They accept arbitrary inputs (called "props") and return React elements describing what should appear on the screen.
- Always start component names with a capital letter.
- React treats components starting with a capital letter.
- React treats components starting with lowercase letters as DOM tags. For example, represents an HTML div tag, but <App /> represents a component and requires App to be in scope.

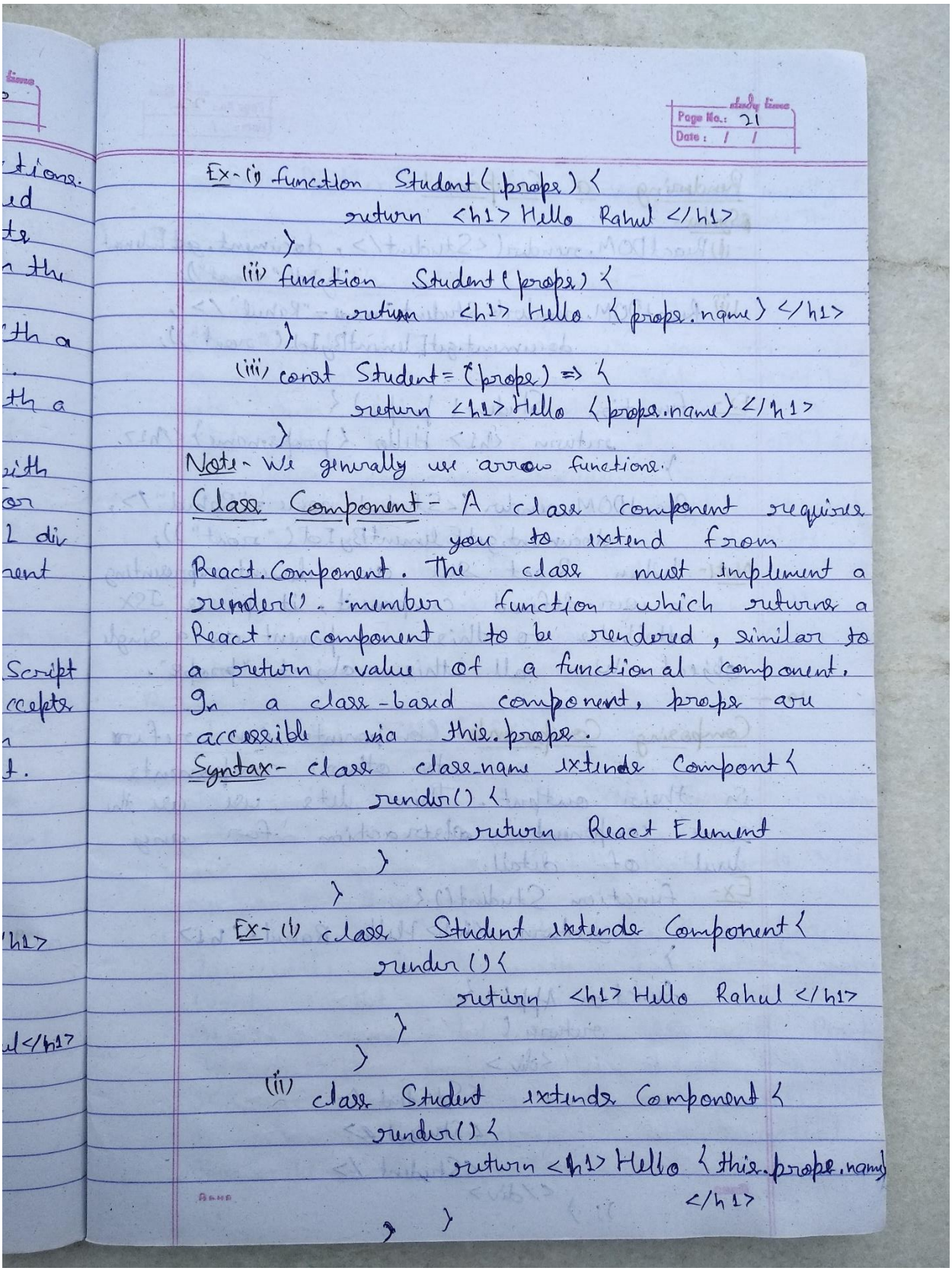
Function Components - It is a JavaScript function which accepts a single "props" object argument with data and returns a React Element.

Syntax - (i) function func_name() {
    return React Element;
}

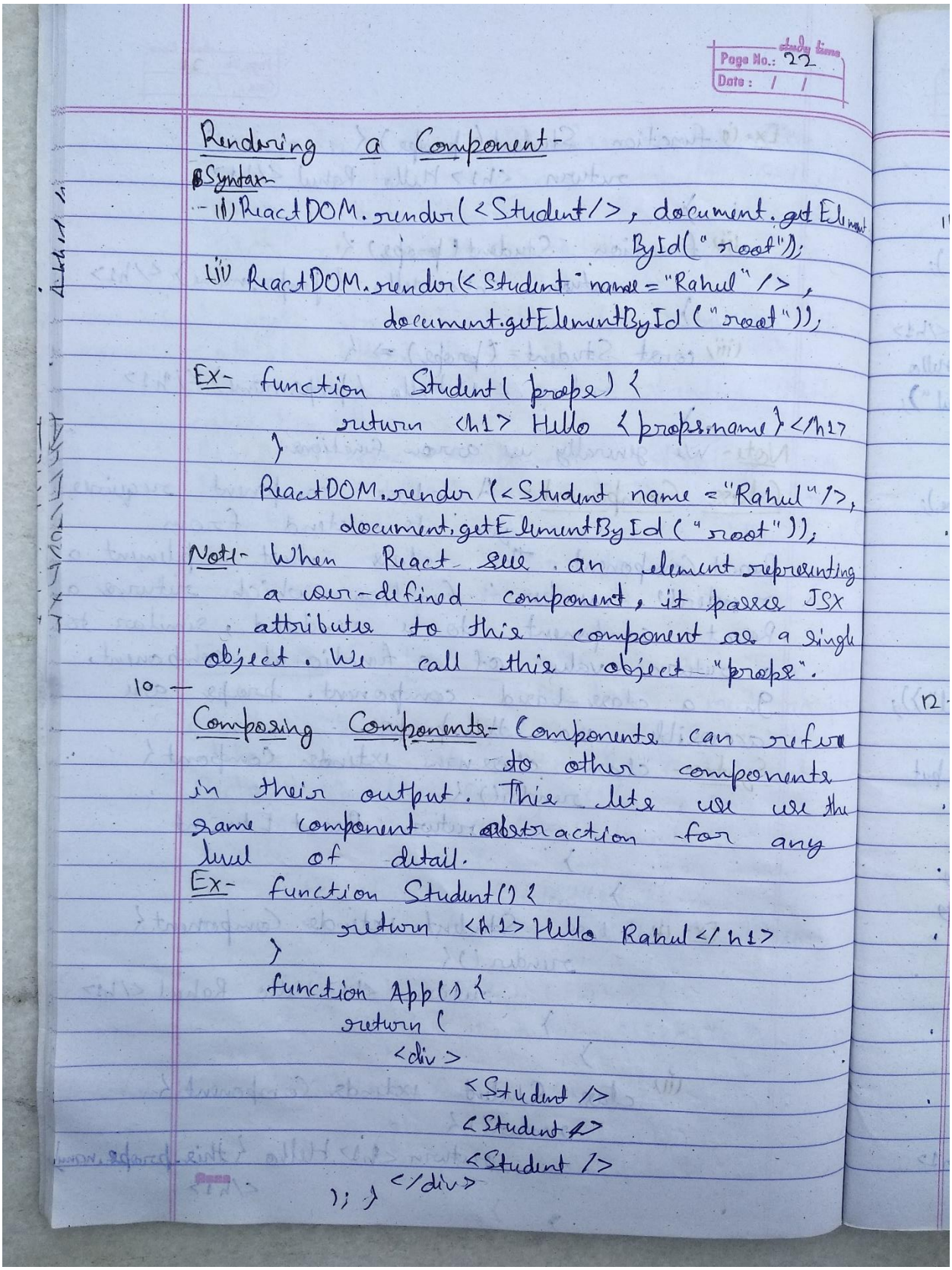
Ex - (i) function Student() {
    return <h1> Hello Rahul </h1>
}

(ii) const Student = () => {
    return <h1> Hello Rahul </h1>
}

(iii) function func_name(props) {
    return React Element;
}



Ex- (i) function Student (props) {
return <h1> Hello Rahul </h1>
}
(ii) function Student (props) {
return <h1> Hello {props.name} </h1>
}
(iii) const Student = (props) => {
return <h1> Hello {props.name} </h1>
}
Note - We generally use arrow functions.
Class Component - A class component requires you to extend from React.Component. The class must implement a render() member function which returns a React component to be rendered, similar to a return value of a functional component. In a class-based component, props are accessible via this.props.
Syntax - class class_name extends Compont {
render() {
return React Element
}
}
Ex- (i) class Student extends Component {
render() {
return <h1> Hello Rahul </h1>
}
}
(ii) class Student extends Component {
render() {
return <h1> Hello {this.props.name}
</h1>
}
}



Page No.: 22
Date: / /

Rendering a Component-

Syntax

(i) ReactDOM.render(<Student/>, document.getElementById("root"));

(ii) ReactDOM.render(<Student name="Rahul" />, document.getElementById("root"));

Ex- function Student(props) {
    return <h1> Hello {props.name} </h1>
}
ReactDOM.render(<Student name="Rahul" />, document.getElementById("root"));

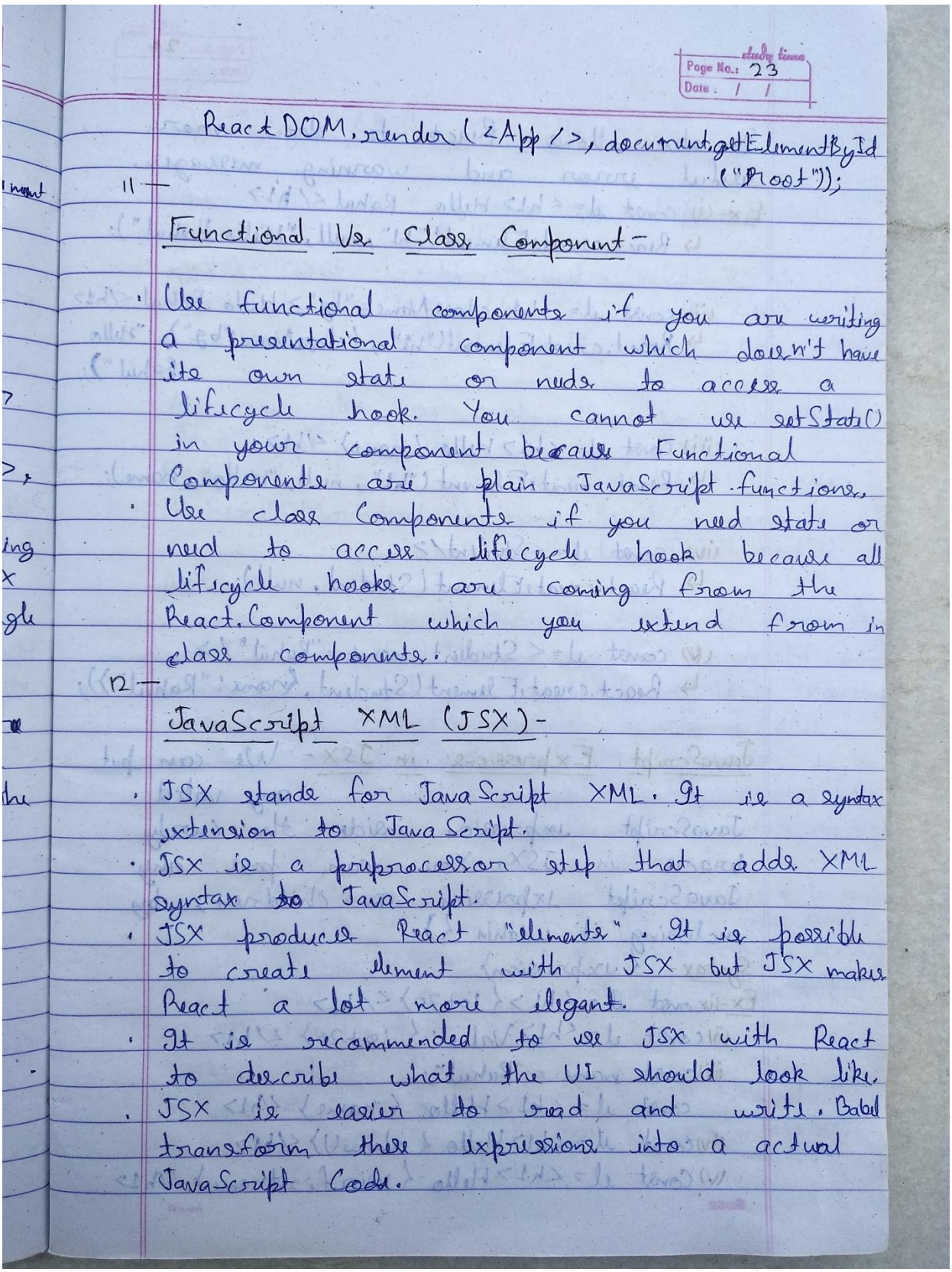
Note- When React sees an element representing a user-defined component, it passes JSX attributes to this component as a single object. We call this object "props".

Composing Components- Components can refer to other components in their output. This lets us use the same component abstraction for any level of detail.

Ex- function Student() {
    return <h1> Hello Rahul </h1>
}
function App() {
    return (

            <Student />
            <Student />
            <Student />

    );
}



ReactDOM.render(<App />, document.getElementById("Root"));

11 —

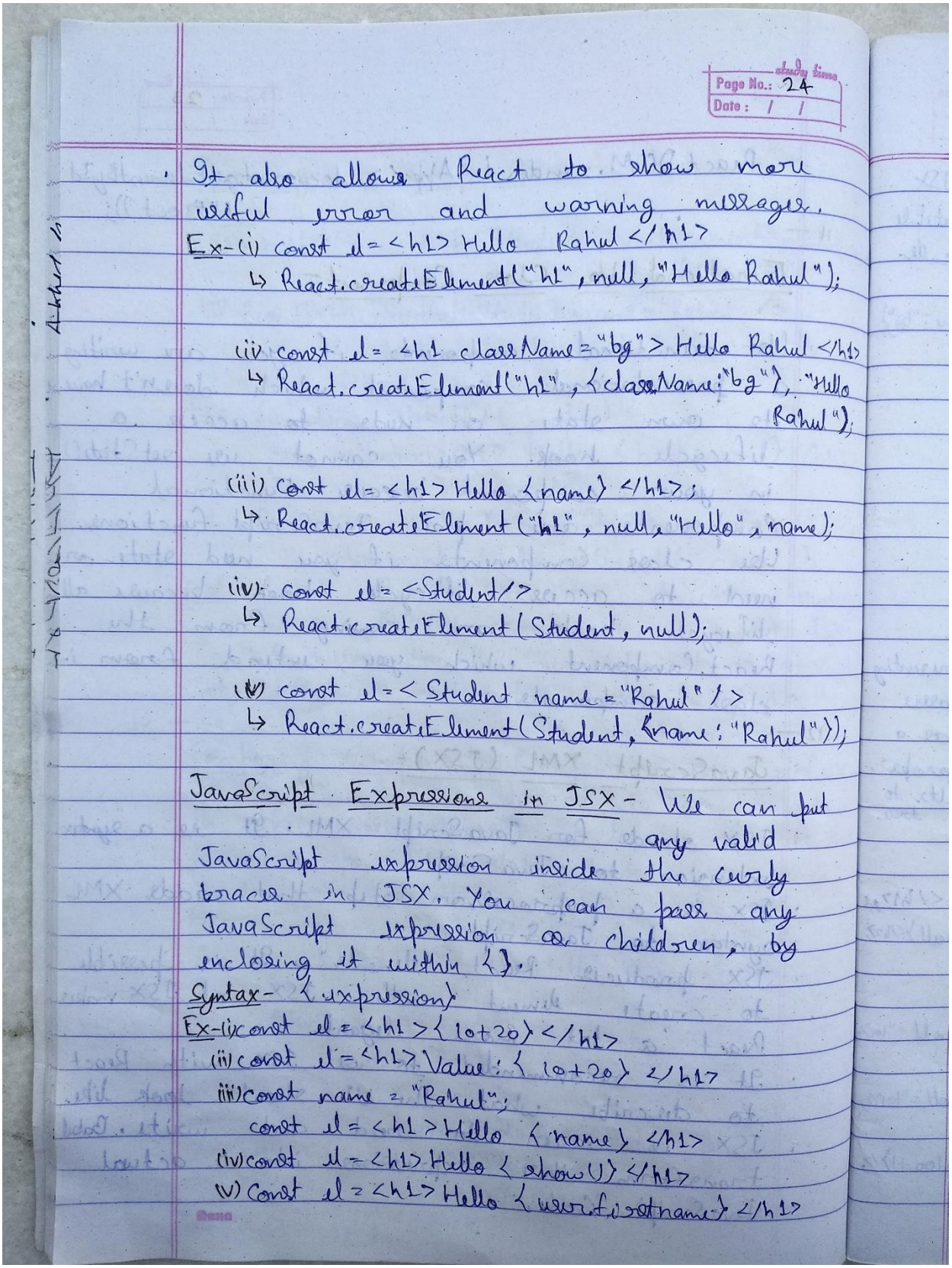
Functional Vs Class Component -

- Use functional components if you are writing a presentational component which doesn't have its own state or needs to access a lifecycle hook. You cannot use setState() in your component because Functional Components are plain JavaScript functions.
- Use class Components if you need state or need to access lifecycle hook because all lifecycle hooks are coming from the React.Component which you extend from in class components.

12 —

JavaScript XML (JSX) -

- JSX stands for JavaScript XML. It is a syntax extension to JavaScript.
- JSX is a preprocessor step that adds XML syntax to JavaScript.
- JSX produces React "elements". It is possible to create element with JSX but JSX makes React a lot more elegant.
- It is recommended to use JSX with React to describe what the UI should look like.
- JSX is easier to read and write. Babel transform these expressions into a actual JavaScript Code.



Page No.: 24
Date : / /

. It also allows React to show more useful error and warning messages.

Ex-(i) const el = <h1> Hello Rahul </h1>
↳ React.createElement("h1", null, "Hello Rahul");

(ii) const el = <h1 className="bg"> Hello Rahul </h1>
↳ React.createElement("h1", {className:"bg"}, "Hello Rahul");

(iii) Const el = <h1> Hello {name} </h1>;
↳ React.createElement("h1", null, "Hello", name);

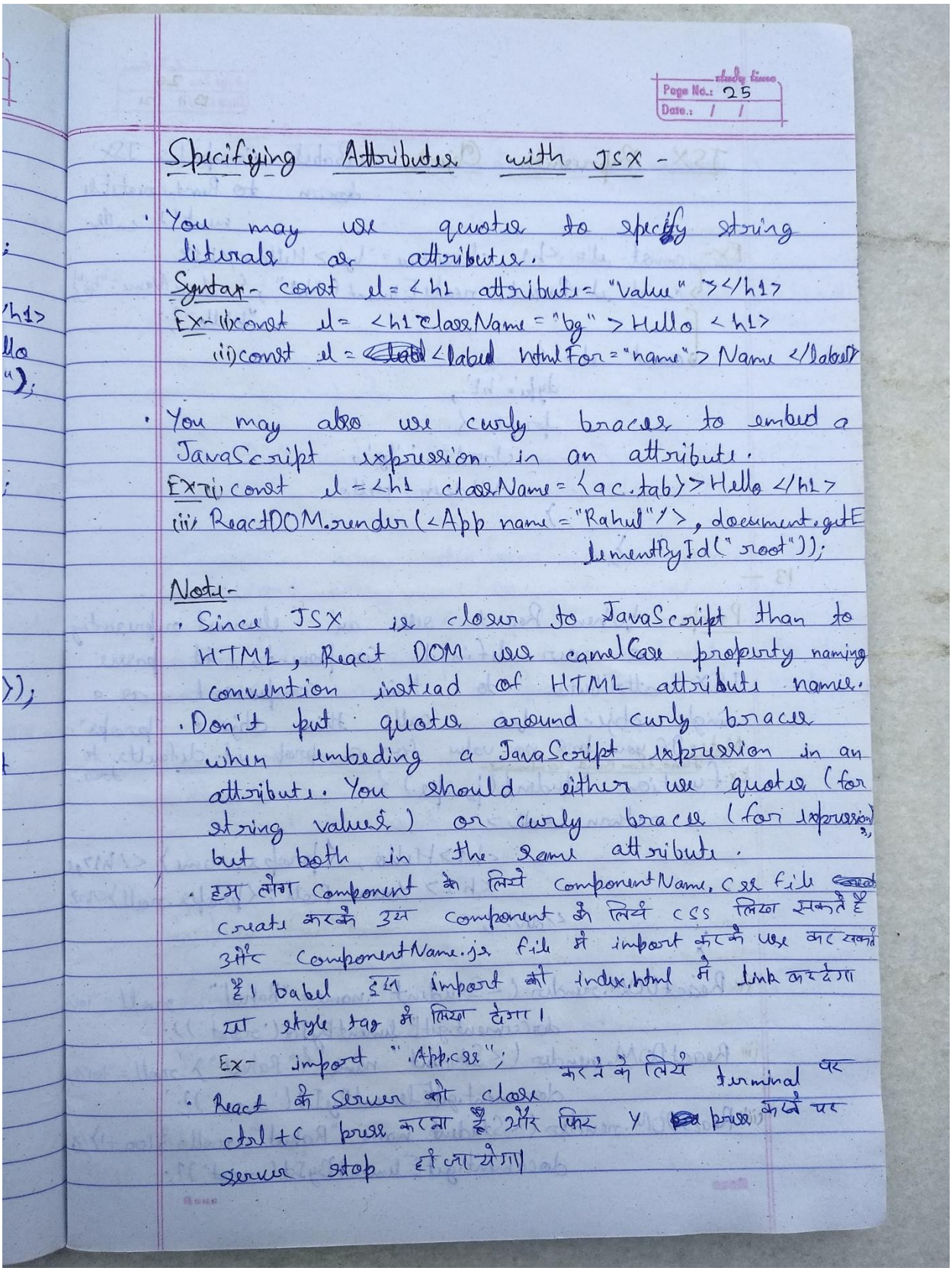
(iv) const el = <Student/>
↳ React.createElement(Student, null);

(v) const el = <Student name="Rahul" />
↳ React.createElement(Student, {name: "Rahul"});

JavaScript Expressions in JSX - We can put any valid JavaScript expression inside the curly braces in JSX. You can pass any JavaScript expression as children, by enclosing it within {}.

Syntax- {expression}

Ex-(i) const el = <h1> {10+20} </h1>
(ii) const el = <h1> Value: {10+20} </h1>
(iii) const name = "Rahul";
const el = <h1> Hello {name} </h1>
(iv) const el = <h1> Hello {show()} </h1>
(v) Const el = <h1> Hello {user.firstname} </h1>



Page No.: 25
Date: / /

Specifying Attributes with JSX -

- You may use quotes to specify string literals as attributes.

Syntax- const el = <h1 attribute = "value" ></h1>

Ex- (i) const el = <h1 className = "bg" > Hello <h1>

(ii) const el = <label htmlFor = "name" > Name </label>

- You may also use curly braces to embed a JavaScript expression in an attribute.

Ex- (i) const el = <h1 className = {ac.tab}> Hello </h1>

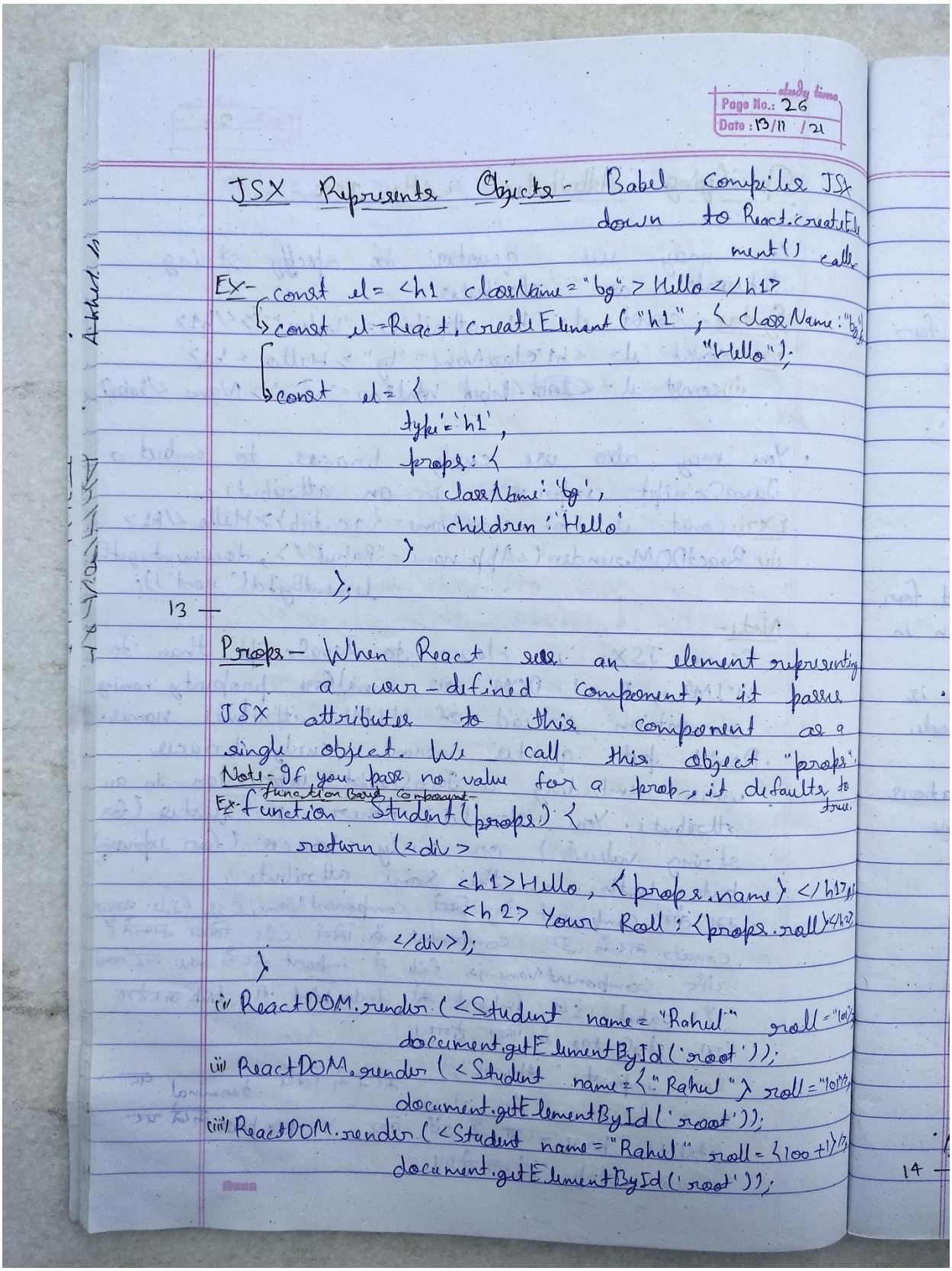
(ii) ReactDOM.render(<App name = "Rahul" />, document.getElementById("root"));

Note-

- Since JSX is closer to JavaScript than to HTML, React DOM use camelCase property naming convention instead of HTML attribute names.
- Don't put quotes around curly braces when imbeding a JavaScript expression in an attribute. You should either use quotes (for string values) or curly braces (for expression), but both in the same attribute.
- हम लोग component के लिये componentName.css file create करके उस component के लिये css लिख सकते है और componentName.js file में import करके use कर सकते है। babel इस import को index.html में link कर देगा या style tag में लिख देगा।

  Ex- import "./App.css";

- React के server को close करने के लिये terminal पर ctrl+c press करना है और फिर Y press करने पर server stop हो जायेगा।



Page No.: 26
Date: 13/11/21

JSX Represents Objects - Babel compiles JSX down to React.createElement() calls.

Ex- const el = <h1 className="bg">Hello</h1>

const el = React.createElement("h1", { className: "bg" }, "Hello");

const el = {
  type: 'h1',
  props: {
    className: 'bg',
    children: 'Hello'
  }
};

13 —

Props - When React sees an element representing a user-defined component, it passes JSX attributes to this component as a single object. We call this object "props".

Note - If you pass no value for a prop, it defaults to true.

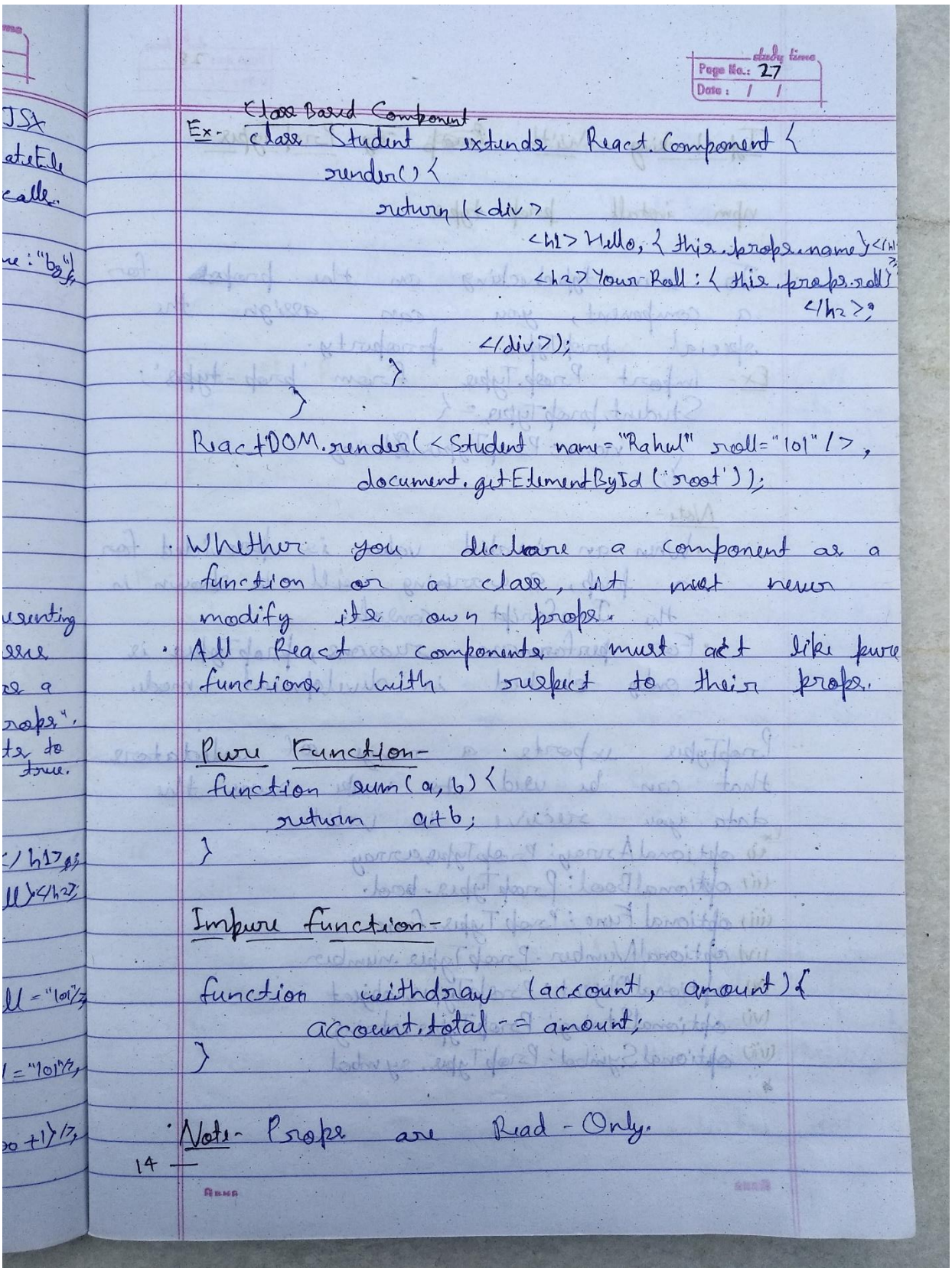
Ex- function Student(props) { (Function Based Component)
  return (
    <h1>Hello, {props.name}</h1>
    <h2>Your Roll: {props.roll}</h2>
  );
}

(i) ReactDOM.render(<Student name="Rahul" roll="101" />, document.getElementById('root'));

(ii) ReactDOM.render(<Student name={"Rahul"} roll="101" />, document.getElementById('root'));

(iii) ReactDOM.render(<Student name="Rahul" roll={100+1} />, document.getElementById('root'));

14 —



Class Based Component -

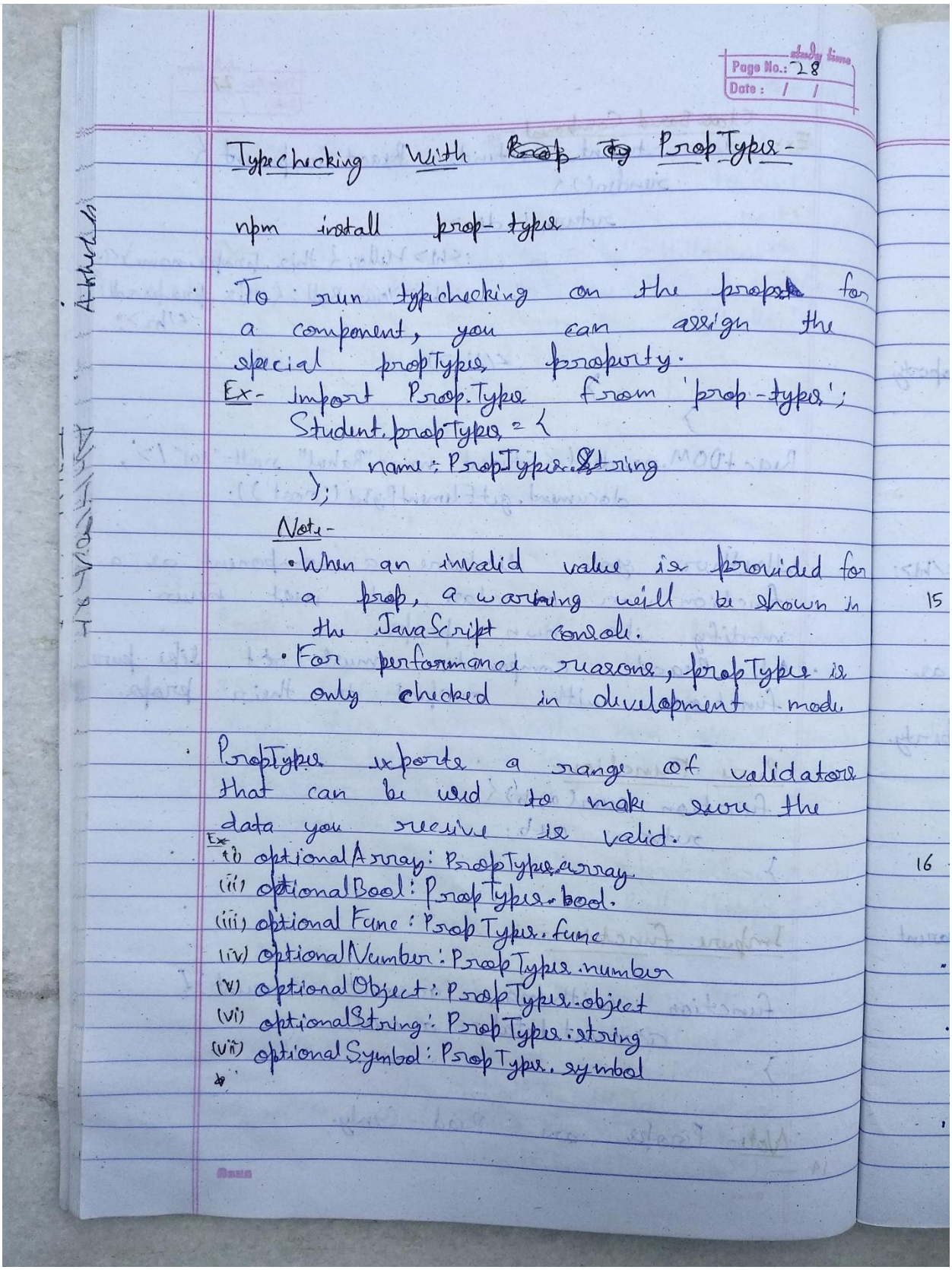
Ex.
```
class Student extends React.Component {
    render() {
        return (<div>
            <h1> Hello, { this.props.name }</h1>
            <h2> Your Roll : { this.props.roll }
            </h2>;
        </div>);
    }
}
ReactDOM.render( <Student name="Rahul" roll="101" />,
    document.getElementById('root'));
```

- Whether you declare a component as a function or a class, it must never modify its own props.
- All React components must act like pure functions with respect to their props.

Pure Function -
```
function sum(a, b) {
    return a+b;
}
```

Impure function -
```
function withdraw (account, amount) {
    account.total -= amount;
}
```

- Note - Props are Read - Only.



Page No.: 28
Date : / /

Typechecking With PropTypes -

npm install prop-types

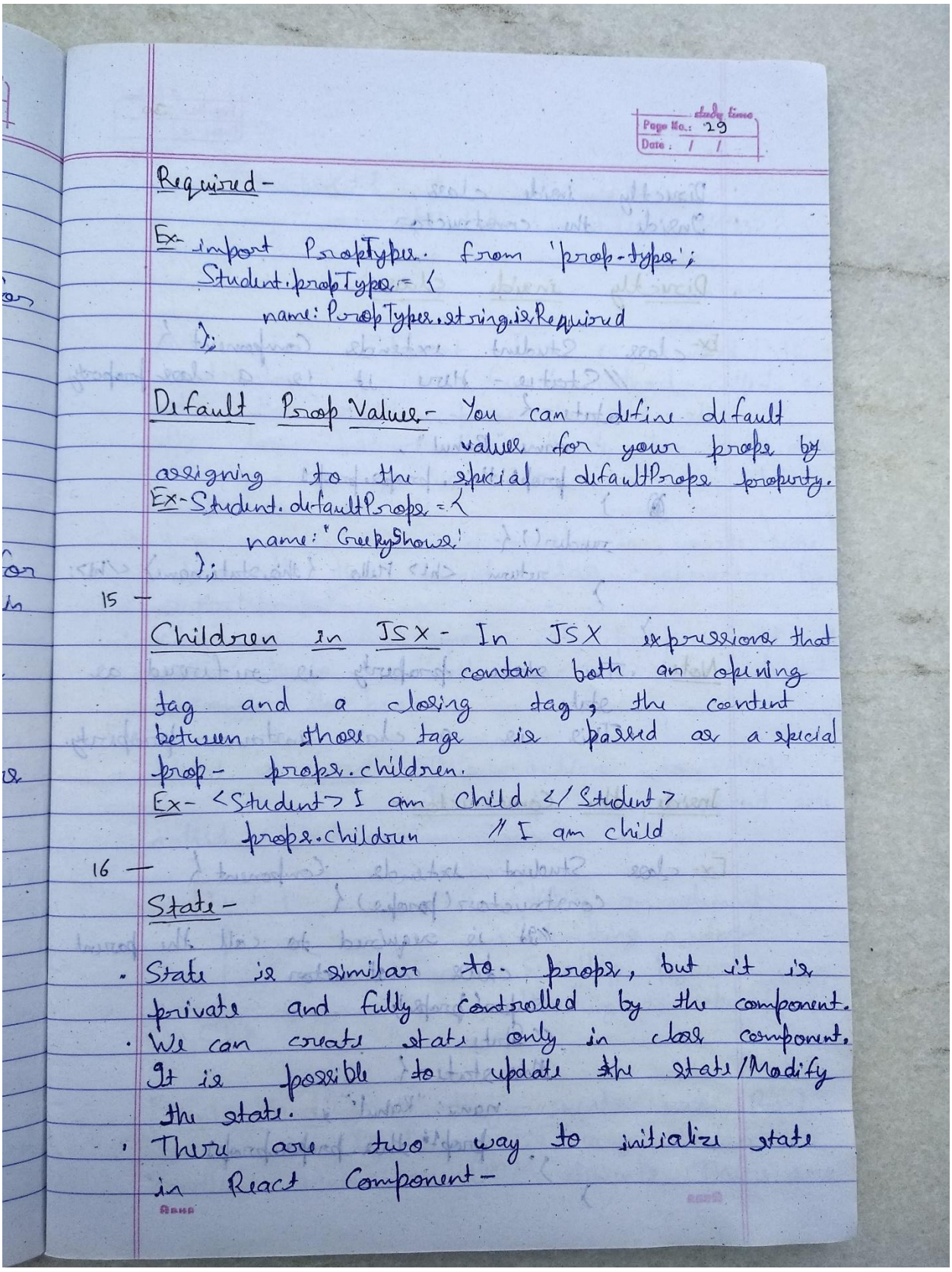
To run typechecking on the props for a component, you can assign the special propTypes property.

Ex- import PropTypes from 'prop-types';
Student.propTypes = {
name : PropTypes.String
};

Note -
- When an invalid value is provided for a prop, a warning will be shown in the JavaScript console.
- For performance reasons, propTypes is only checked in development mode.

PropTypes exports a range of validators that can be used to make sure the data you receive is valid.

Ex
(i) optionalArray : PropTypes.array
(ii) optionalBool : PropTypes.bool
(iii) optionalFunc : PropTypes.func
(iv) optionalNumber : PropTypes.number
(v) optionalObject : PropTypes.object
(vi) optionalString : PropTypes.string
(vii) optionalSymbol : PropTypes.symbol



Required -

Ex- import PropTypes. from 'prop-types';
Student.propTypes = {
name: PropTypes.string.isRequired
};

Default Prop Value - You can define default values for your props by assigning to the special defaultProps property.
Ex- Student.defaultProps = {
name: 'GeekyShows'
};

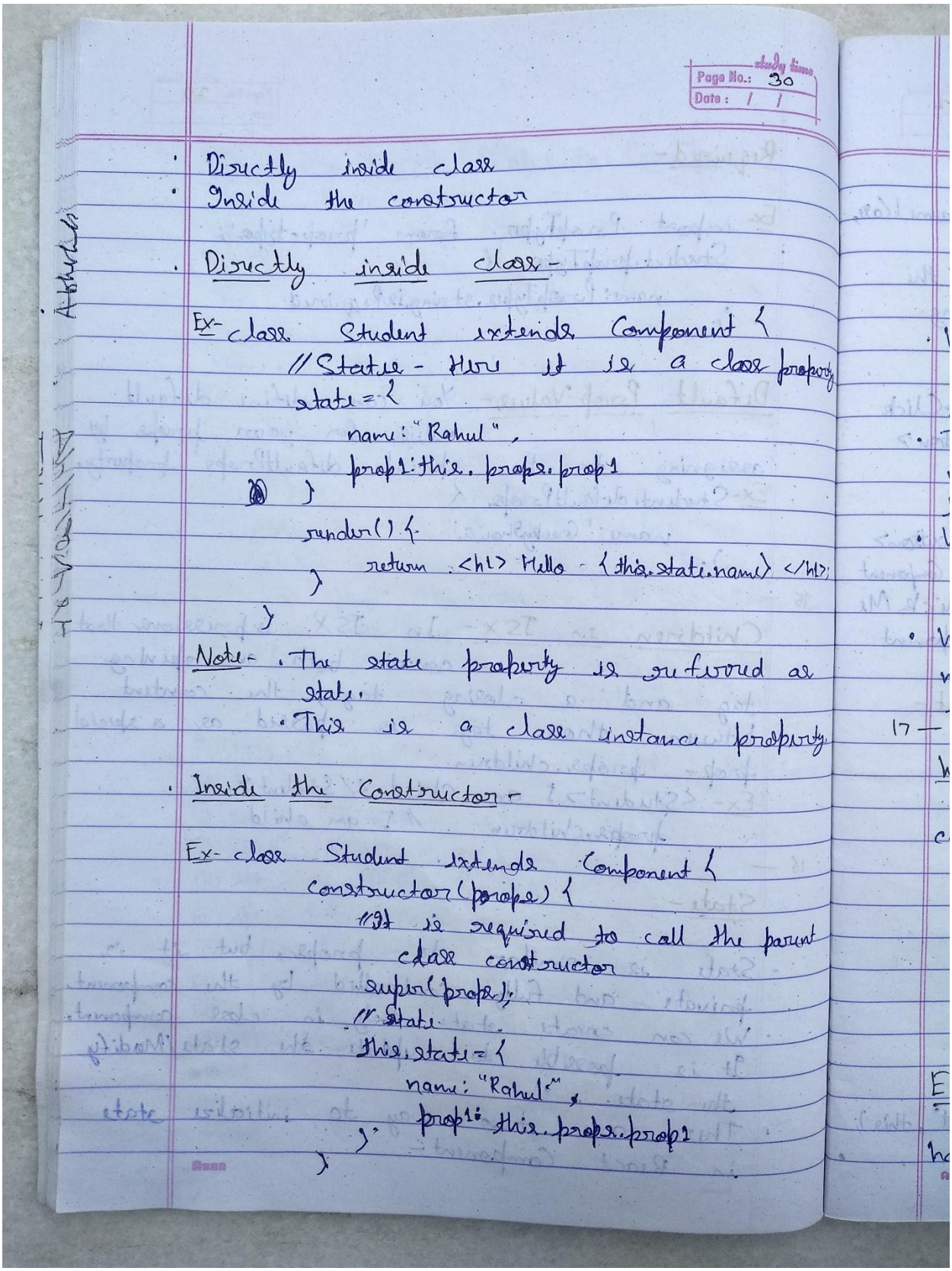
15 —

Children in JSX - In JSX expressions that contain both an opening tag and a closing tag, the content between those tags is passed as a special prop - props.children.
Ex- <Student> I am child </Student>
props.children // I am child

16 —

State -

- State is similar to props, but it is private and fully controlled by the component.
- We can create state only in class component.
- It is possible to update the state / Modify the state.
- There are two way to initialize state in React Component -



Page No.: 30
Date : / /

- Directly inside class
- Inside the constructor

- Directly inside class -

Ex- class Student extends Component {
    // State - Here it is a class property
    state = {
        name: "Rahul",
        prop1: this.props.prop1
    }
    render() {
        return <h1> Hello - {this.state.name} </h1>;
    }
}

Note - . The state property is referred as state.
. This is a class instance property

- Inside the Constructor -

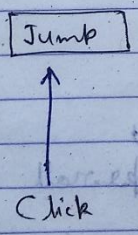
Ex- class Student extends Component {
    constructor(props) {
        // It is required to call the parent class constructor
        super(props);
        // State
        this.state = {
            name: "Rahul",
            prop1: this.props.prop1
        }
    }

```
render() {
    return <h1> Hello {this.state.name} </h1>;
  }
}
```

- When the component class is created, the constructor is the first method called, so it's the right place to add state.
- The class instance has already been created in memory, so you can use this to set properties on it.
- When we write a constructor, make sure to call the parent class' constructor by super(props).
- When you call super with props. React will make props available across the component through this.props.

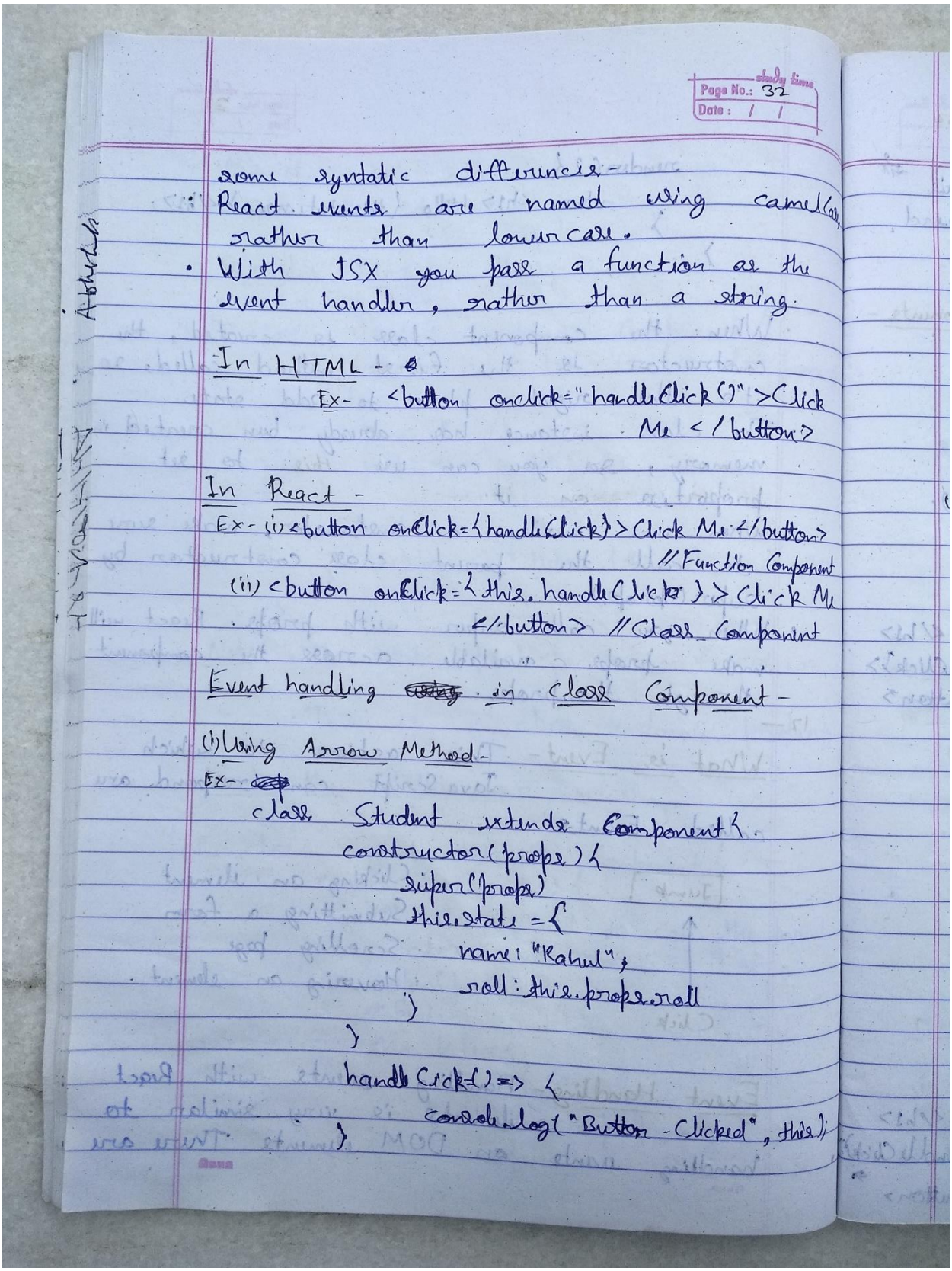
17 —

What is Event - This actions to which JavaScript can respond are called Events.



- Clicking an element
- Submitting a form
- Scrolling page
- Hovering an element

Event Handling - Handling events with React elements is very similar to handling events on DOM elements. There are



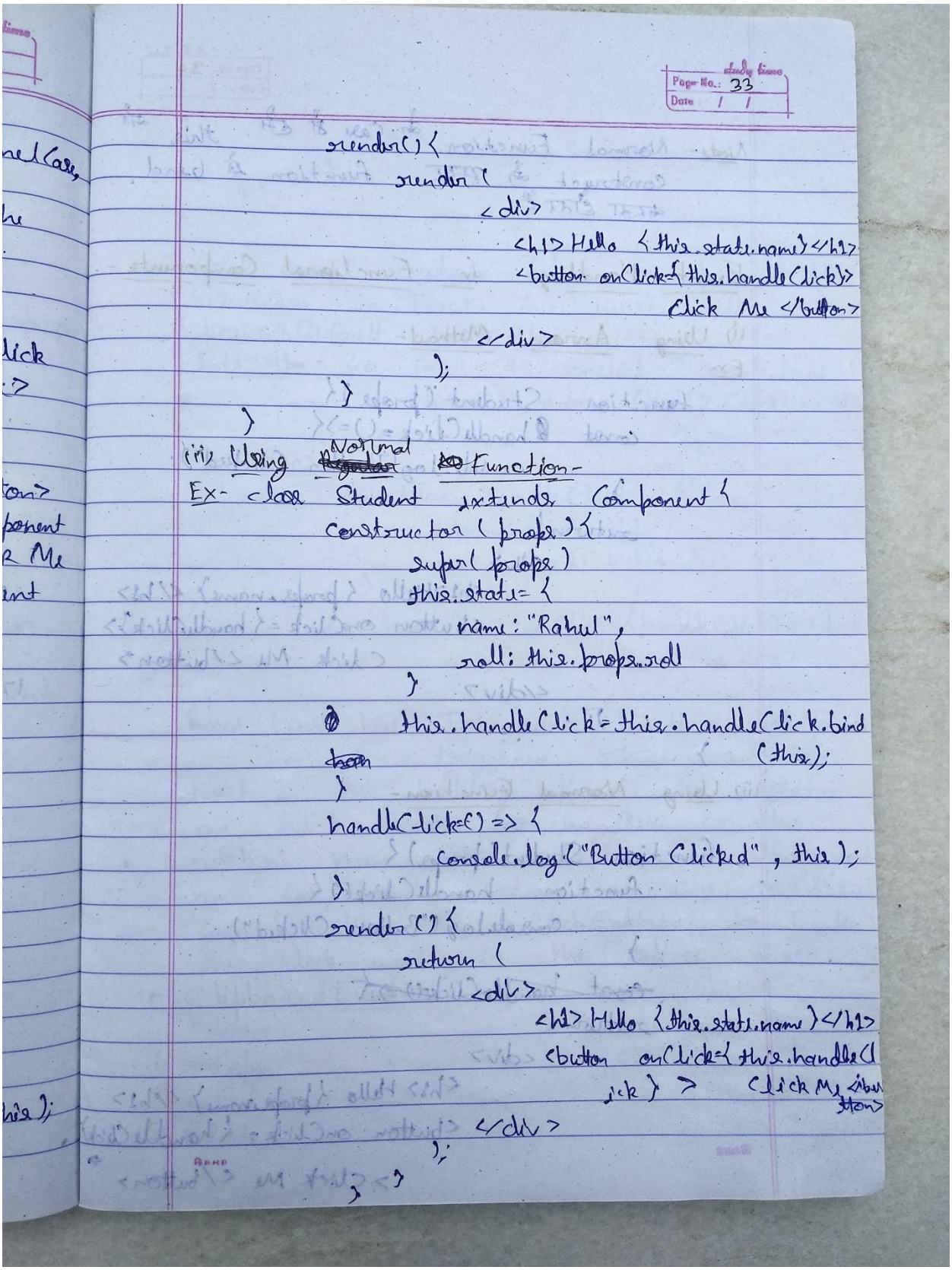
some syntatic differences -
- React events are named using camelCase, rather than lowercase.
- With JSX you pass a function as the event handler, rather than a string.

In HTML -
Ex- <button onclick="handleClick()">Click Me</button>

In React -
Ex- (i) <button onClick={handleClick}>Click Me</button>
// Function Component
(ii) <button onClick={this.handleClick}>Click Me</button> // Class Component

Event handling in class Component -

(i) Using Arrow Method -
Ex-
```
class Student extends Component{
    constructor(props){
        super(props)
        this.state={
            name: "Rahul",
            roll: this.props.roll
        }
    }
    handleClick=()=>{
        console.log("Button Clicked", this);
    }
```



render() {
return (

<h1> Hello {this.state.name}</h1>
<button onClick={this.handleClick}>
Click Me </button>

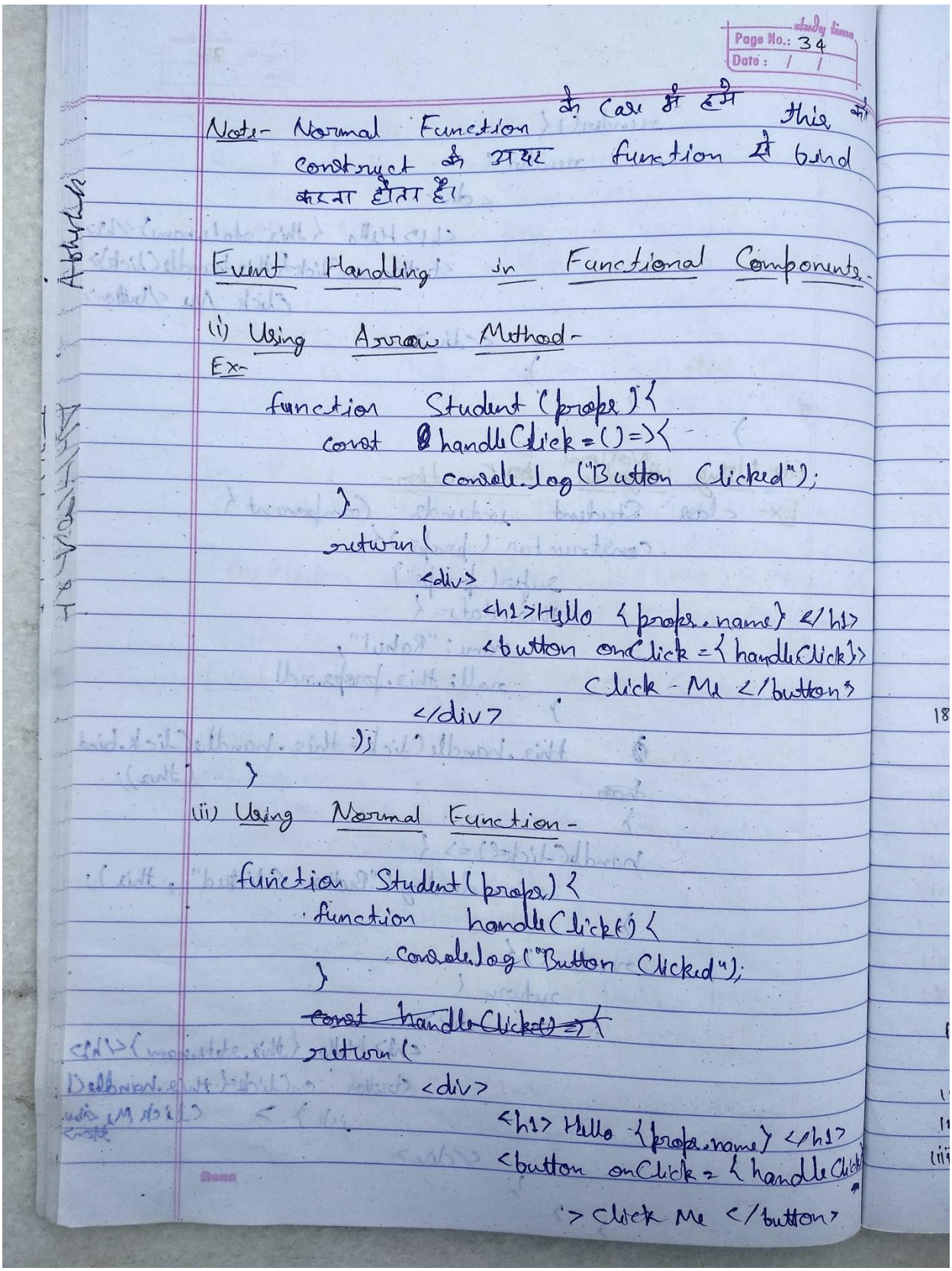
);
}
}

(iii) Using Normal Function -

Ex- class Student extends Component {
constructor (props) {
super(props)
this.state = {
name: "Rahul",
roll: this.props.roll
}
this.handleClick = this.handleClick.bind
(this);
}
handleClick=() => {
console.log("Button Clicked", this);
}
render () {
return (

<h1> Hello {this.state.name}</h1>
<button onClick={this.handleC
ick}> Click Me </but

);
}



Page No.: 34
Date: / /

Note- Normal Function के case में हमें this को Construct के अंदर function से bind करना होता है।

Event Handling in Functional Components-

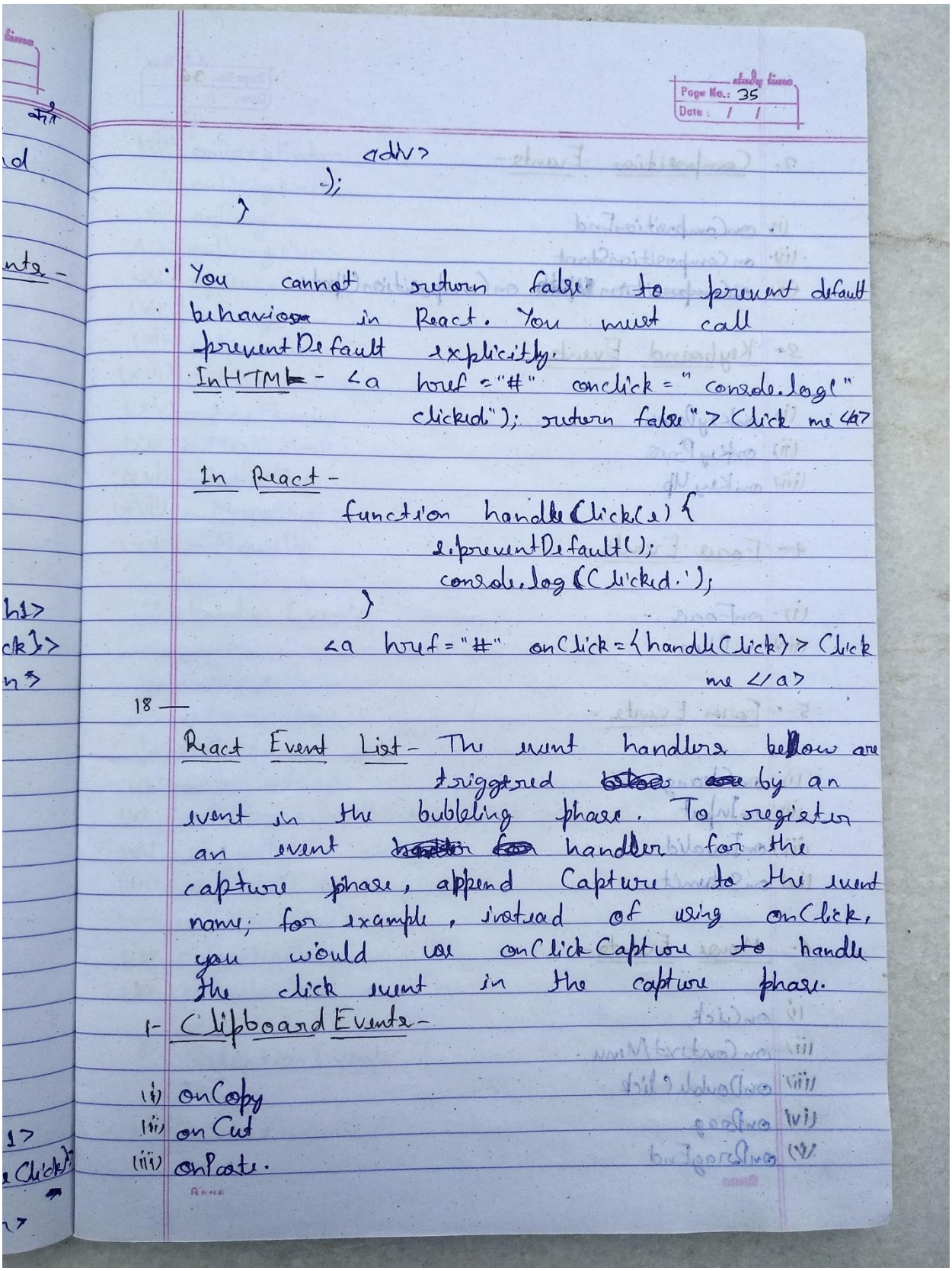
(i) Using Arrow Method-

Ex-

```
function Student (props) {
    const handleClick = () => {
        console.log("Button Clicked");
    }
    return (
        <div>
            <h1>Hello {props.name} </h1>
            <button onClick={handleClick}>
                Click Me </button>
        </div>
    );
}
```

(ii) Using Normal Function-

```
function Student(props) {
    function handleClick() {
        console.log("Button Clicked");
    }
    return (
        <div>
            <h1> Hello {props.name} </h1>
            <button onClick={handleClick}
            > Click Me </button>
```



Page No.: 35
Date: / /

```
    <div>
  );
}
```

- You cannot return false to prevent default behaviour in React. You must call preventDefault explicitly.

In HTML - `<a href="#" onclick="console.log('clicked.'); return false">Click me</a>`

In React -

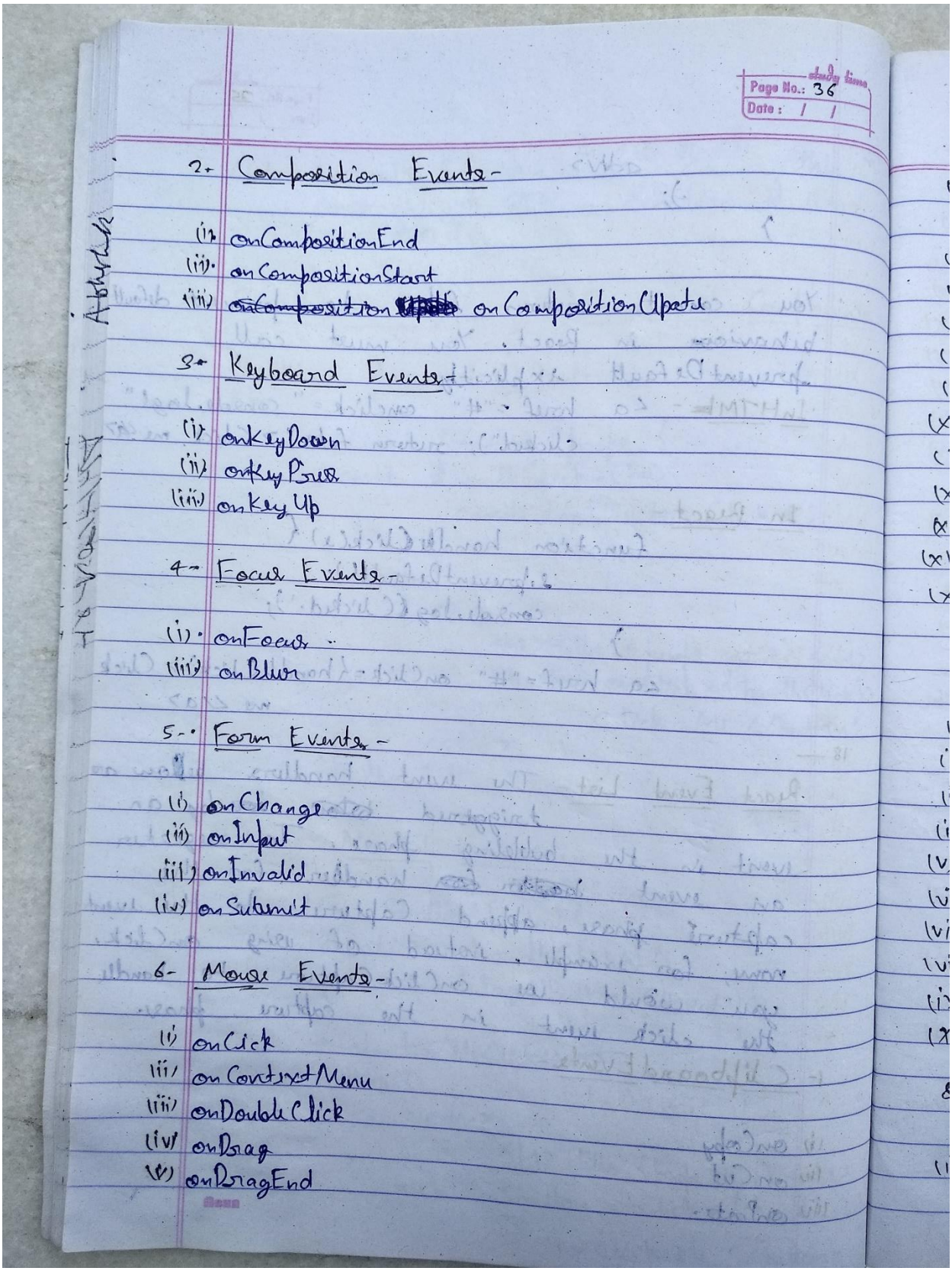
```
function handleClick(e) {
  e.preventDefault();
  console.log('Clicked.');
}
<a href="#" onClick={handleClick}>Click me</a>
```

18 —

React Event List - The event handlers below are triggered by an event in the bubbling phase. To register an event handler for the capture phase, append Capture to the event name; for example, instead of using onClick, you would use onClickCapture to handle the click event in the capture phase.

1- Clipboard Events -

(i) onCopy
(ii) onCut
(iii) onPaste.



2- Composition Events -

(i) onCompositionEnd
(ii) onCompositionStart
(iii) onCompositionUpdate

3- Keyboard Events -

(i) onKeyDown
(ii) onKeyPress
(iii) onKeyUp

4- Focus Events -

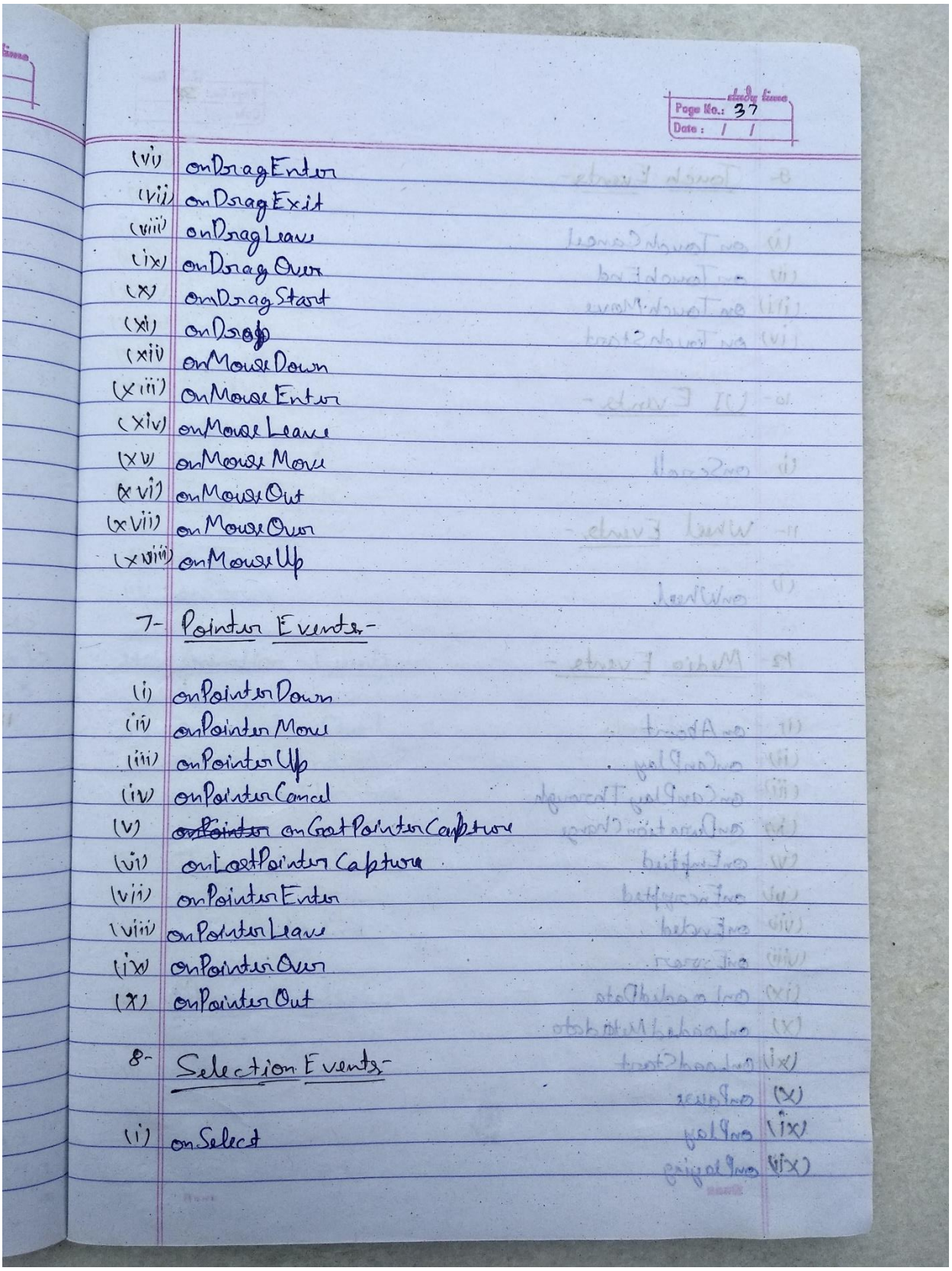
(i) onFocus
(ii) onBlur

5- Form Events -

(i) onChange
(ii) onInput
(iii) onInvalid
(iv) onSubmit

6- Mouse Events -

(i) onClick
(ii) onContextMenu
(iii) onDoubleClick
(iv) onDrag
(v) onDragEnd



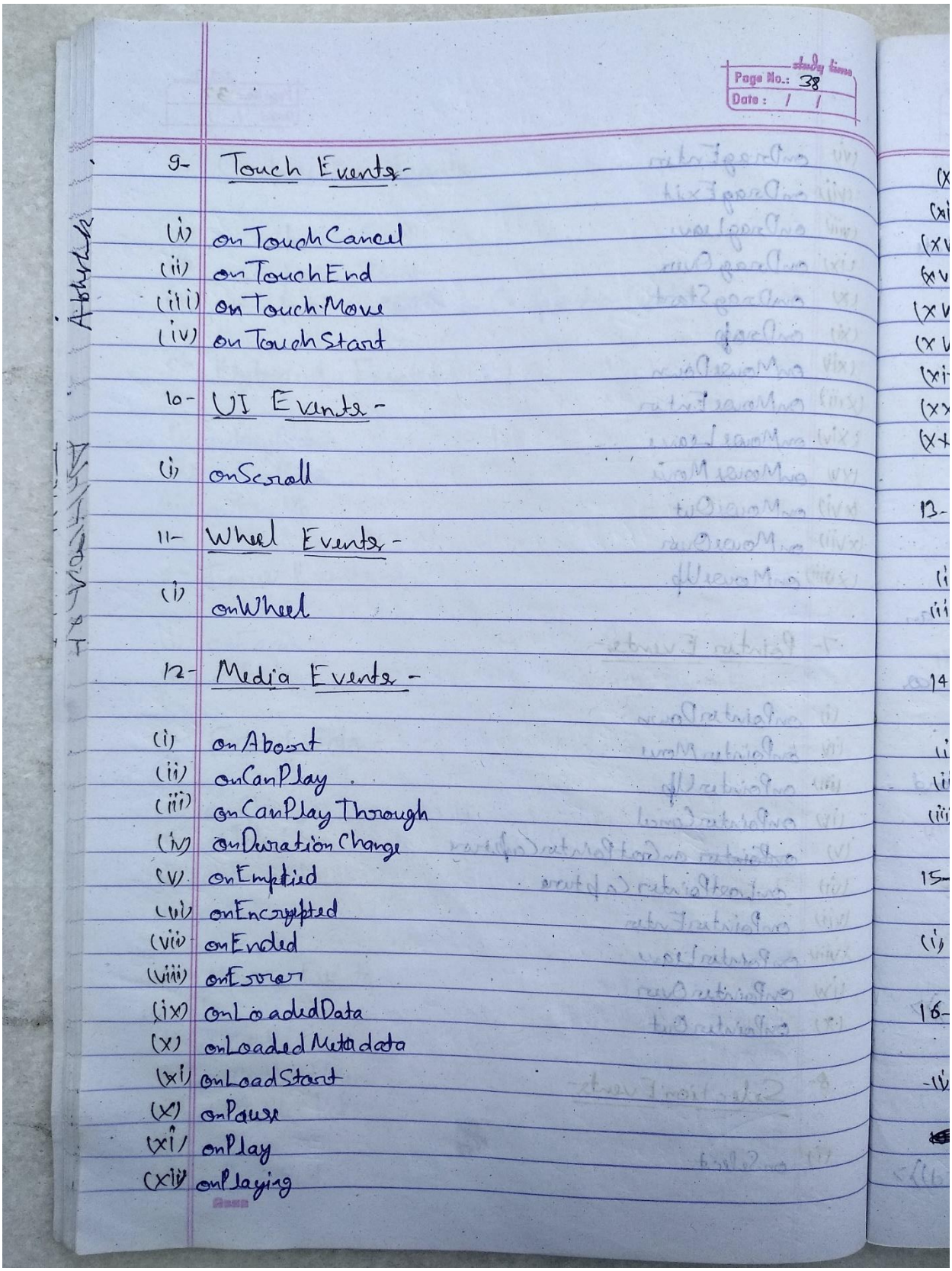
(vi) onDragEnter
(vii) onDragExit
(viii) onDragLeave
(ix) onDragOver
(x) onDragStart
(xi) onDrop
(xii) onMouseDown
(xiii) onMouseEnter
(xiv) onMouseLeave
(xv) onMouseMove
(xvi) onMouseOut
(xvii) onMouseOver
(xviii) onMouseUp

7- Pointer Events-

(i) onPointerDown
(ii) onPointerMove
(iii) onPointerUp
(iv) onPointerCancel
(v) ~~onPointer~~ onGotPointerCapture
(vi) onLostPointerCapture
(vii) onPointerEnter
(viii) onPointerLeave
(ix) onPointerOver
(x) onPointerOut

8- Selection Events-

(i) onSelect



9- Touch Events -

(i) onTouchCancel
(ii) onTouchEnd
(iii) onTouchMove
(iv) onTouchStart

10- UI Events -

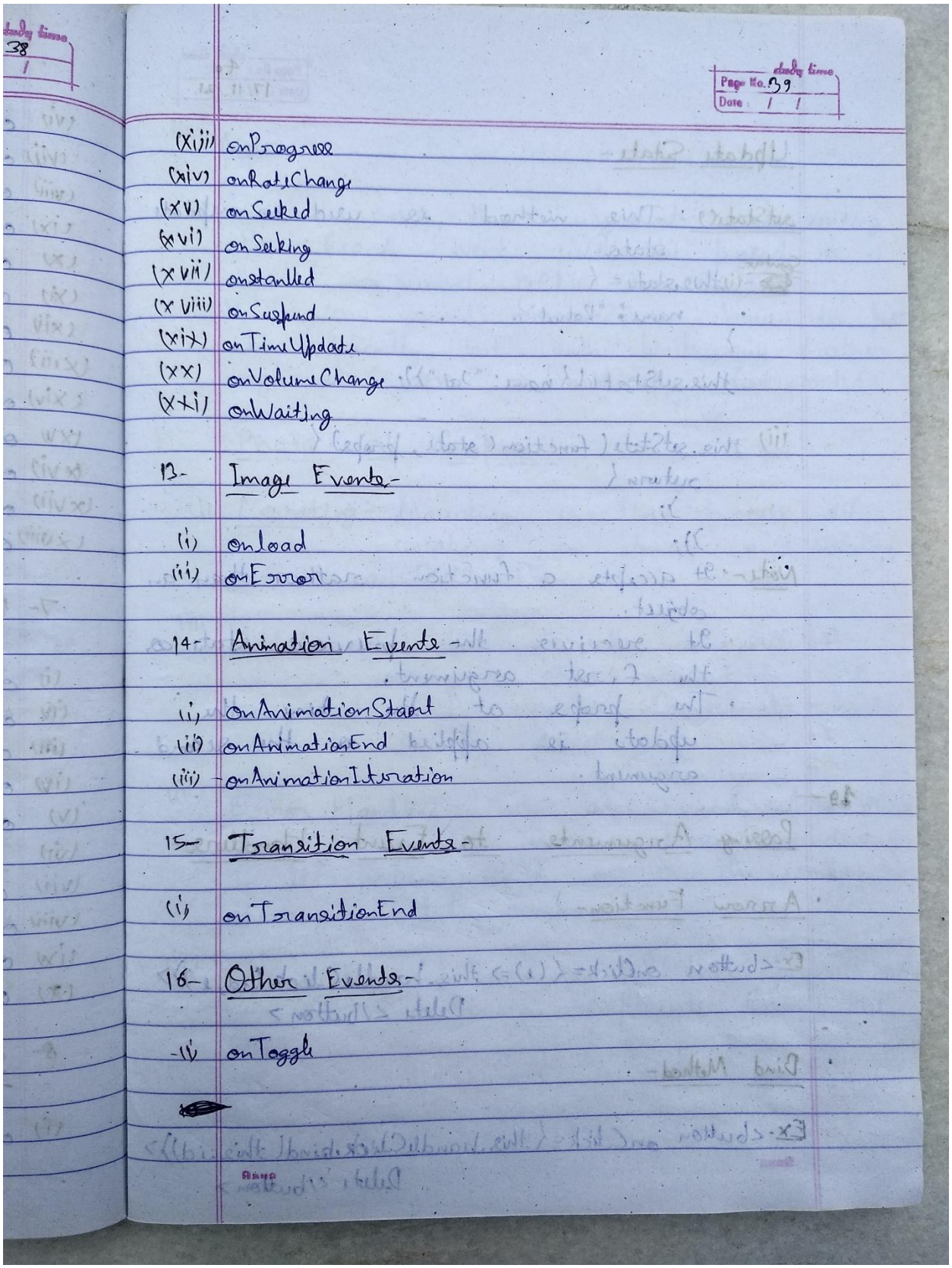
(i) onScroll

11- Wheel Events -

(i) onWheel

12- Media Events -

(i) onAbort
(ii) onCanPlay
(iii) onCanPlayThrough
(iv) onDurationChange
(v) onEmptied
(vi) onEncrypted
(vii) onEnded
(viii) onError
(ix) onLoadedData
(x) onLoadedMetadata
(xi) onLoadStart
(x) onPause
(xi) onPlay
(xii) onPlaying



(xiii) onProgress
(xiv) onRateChange
(xv) onSeeked
(xvi) onSeeking
(xvii) onStalled
(xviii) onSuspend
(xix) onTimeUpdate
(xx) onVolumeChange
(xxi) onWaiting

13- Image Events-

(i) onLoad
(ii) onError

14- Animation Events -

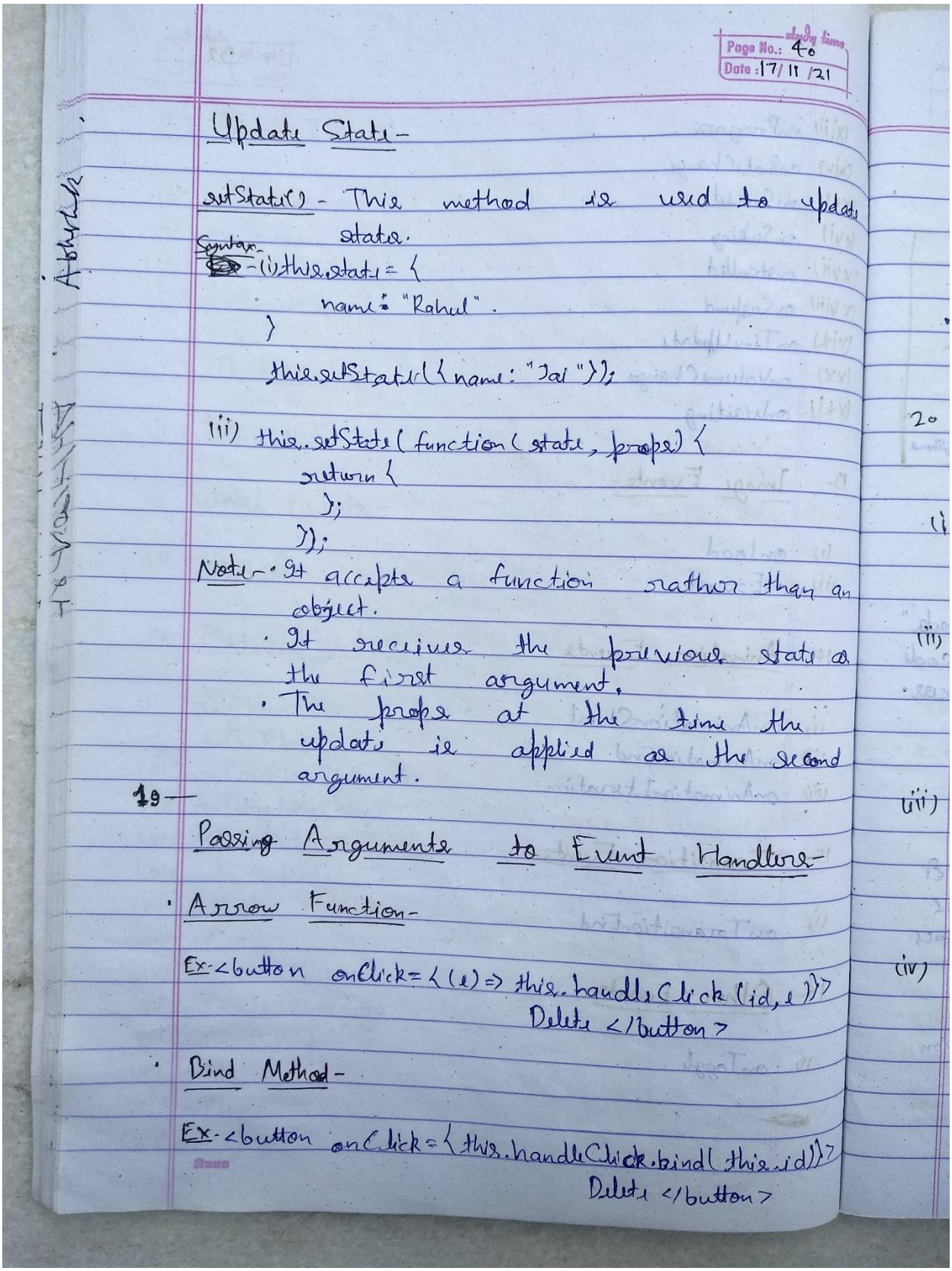
(i) onAnimationStart
(ii) onAnimationEnd
(iii) onAnimationIteration

15- Transition Events-

(i) onTransitionEnd

16- Other Events-

(i) onToggle



Page No.: 40
Date: 17/11/21

Update State -

setState() - This method is used to update state.

Syntax -

(i) this.state = {
    name: "Rahul".
}

this.setState({name: "Jai"});

(iii) this.setState(function(state, props) {
    return {
    };
});

Note -
- It accepts a function rather than an object.
- It receives the previous state as the first argument.
- The props at the time the update is applied as the second argument.

19 —

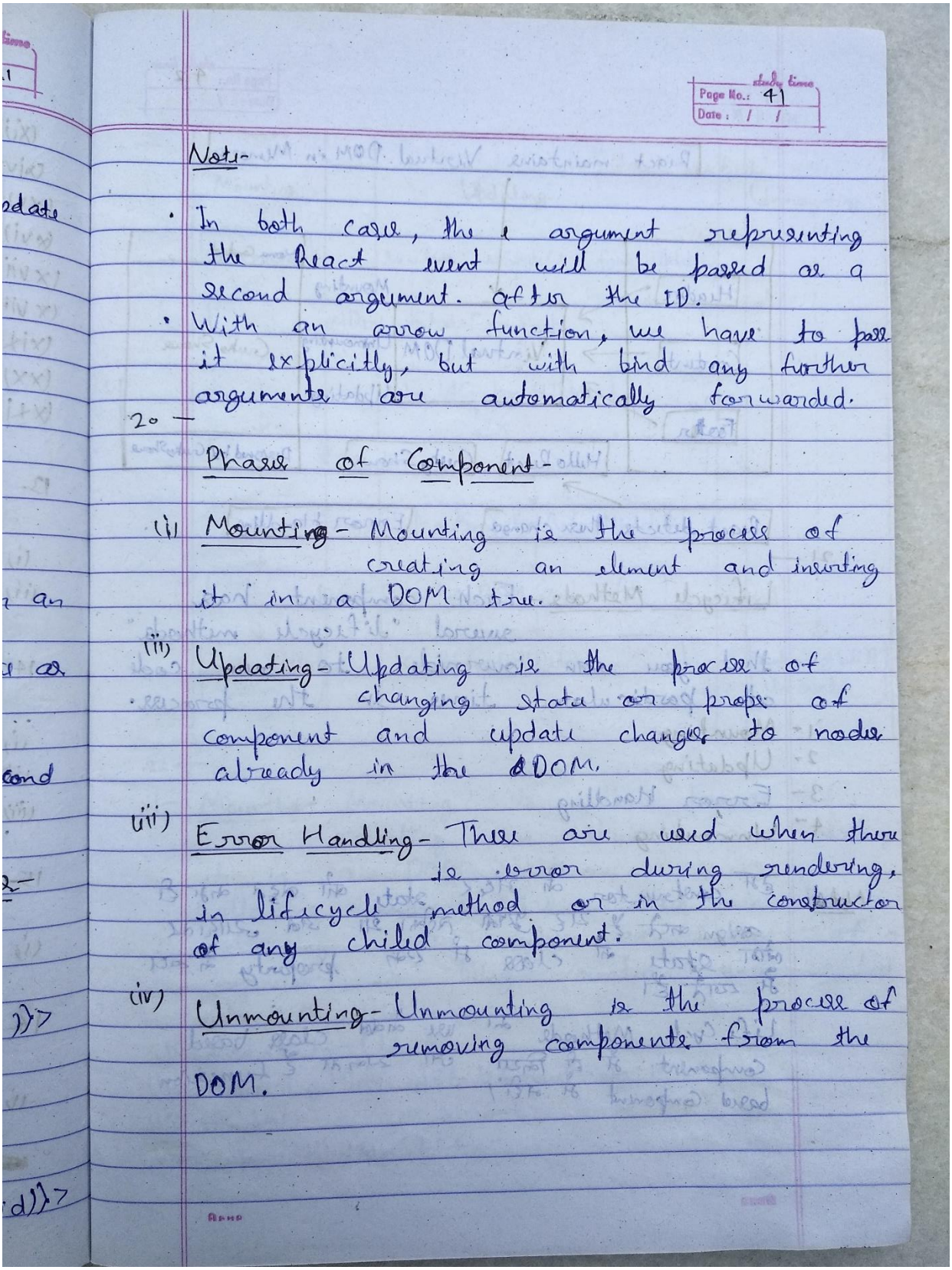
Passing Arguments to Event Handlers -

- **Arrow Function** -

Ex - <button onClick={(e) => this.handleClick(id, e)}>
Delete </button>

- **Bind Method** -

Ex - <button onClick={this.handleClick.bind(this, id)}>
Delete </button>



Page No.: 41
Date : / /

Note-

- In both case, the e argument representing the React event will be passed as a second argument. after the ID.
- With an arrow function, we have to pass it explicitly, but with bind any further arguments are automatically forwarded.

20 —

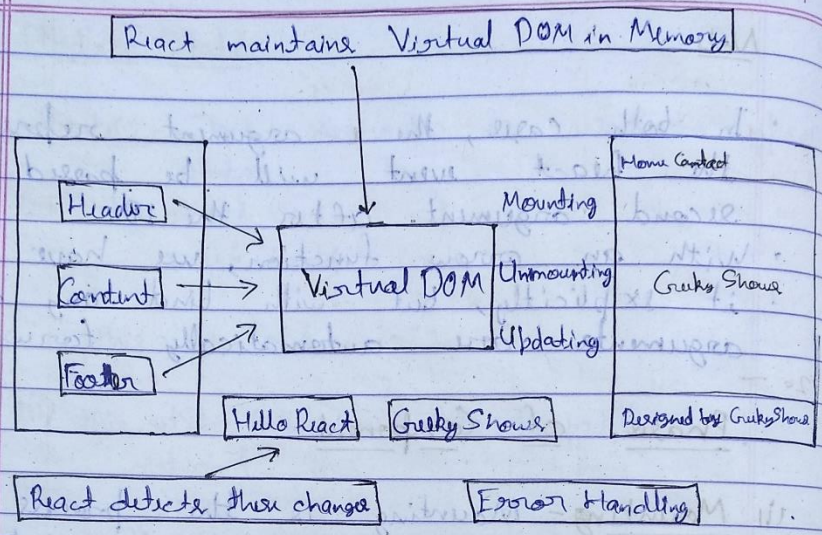
Phase of Component -

(i) Mounting - Mounting is the process of creating an element and inserting it in a DOM tree.

(ii) Updating - Updating is the process of changing state or props of component and update changes to node already in the DOM.

(iii) Error Handling - These are used when there is error during rendering, in lifecycle method or in the constructor of any child component.

(iv) Unmounting - Unmounting is the process of removing components from the DOM.



21 —

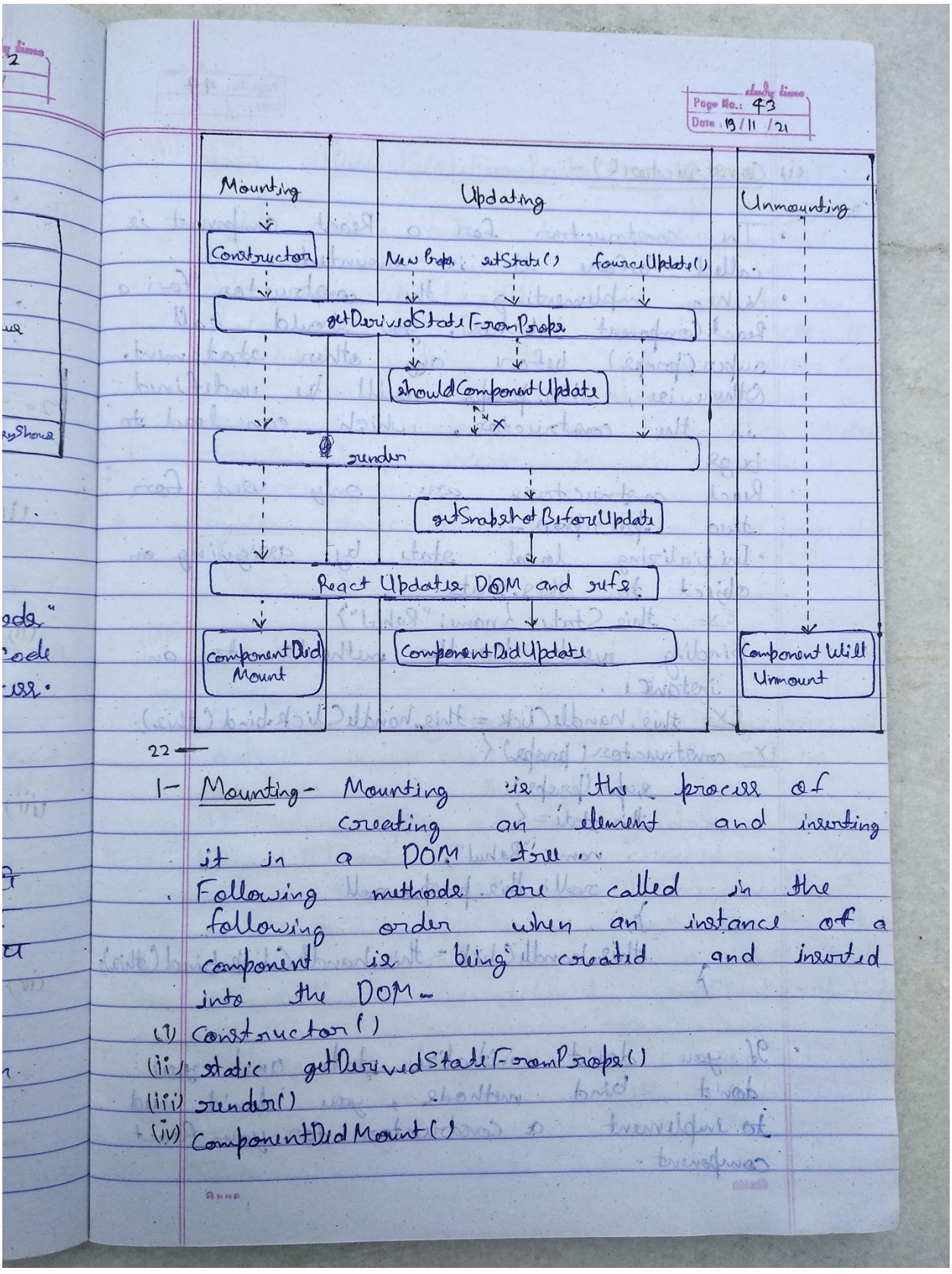
Lifecycle Methods - Each component has several "lifecycle methods" that you can override to run code at particular times in the process.

1- Mounting
2- Updating
3- Error Handling
4- Unmounting

Note - हम constructor के अंदर state को बहुत कम ही assign करते हैं, यह पुराना तरीका था अब ज्यादातर लोग state को class में एक property के रूप में रखते हैं।

Life Cycle Methods का use केवल class based component में ही किया जा सकता है function based component में नहीं।

22 —
1-
(i)
(ii)
(iii)
(iv)

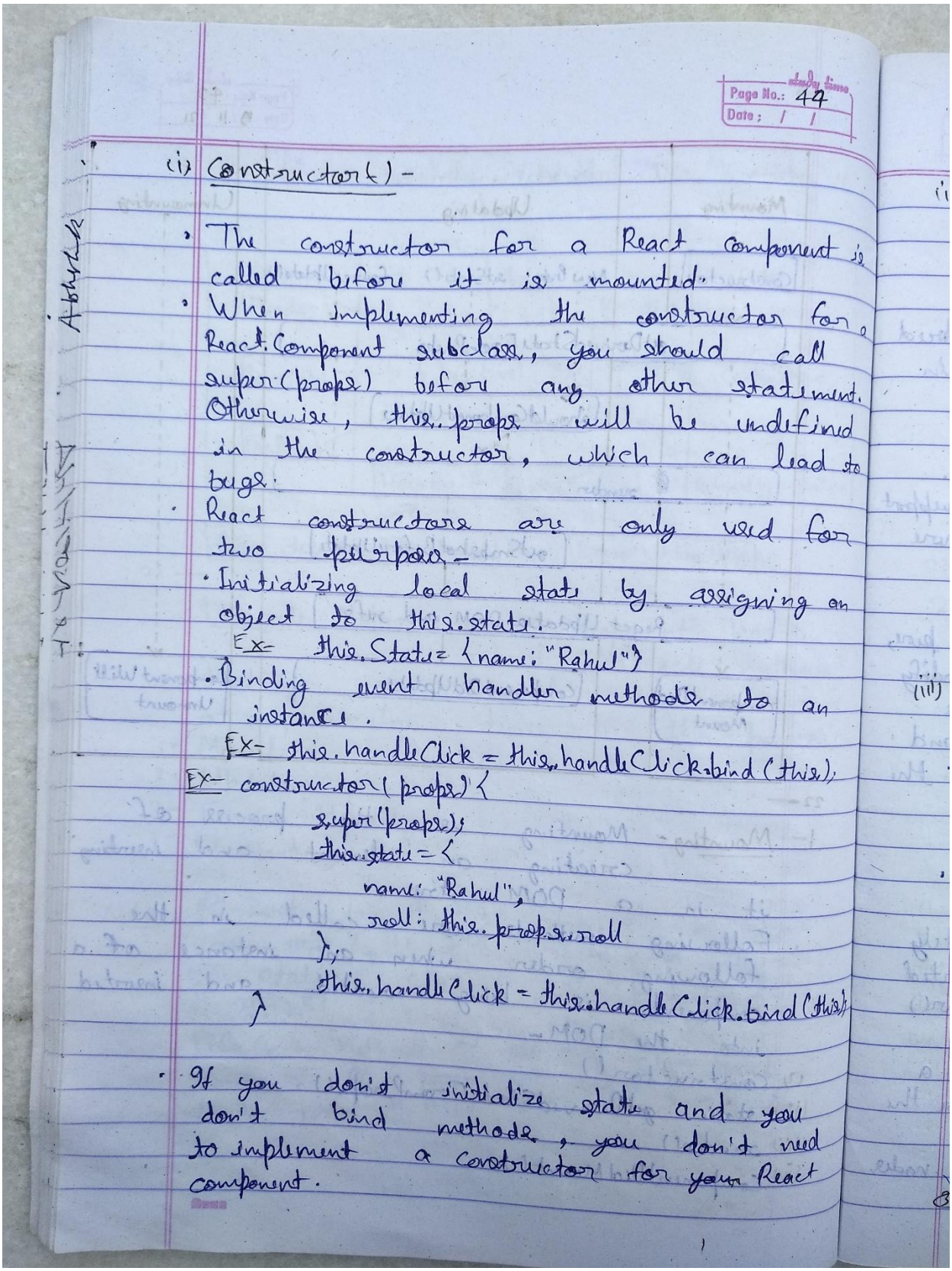


22

1- Mounting - Mounting is the process of creating an element and inserting it in a DOM tree.

Following methods are called in the following order when an instance of a component is being created and inserted into the DOM -

(i) constructor()
(ii) static getDerivedStateFromProps()
(iii) render()
(iv) componentDidMount()



(i) Constructor() -

- The constructor for a React component is called before it is mounted.
- When implementing the constructor for a React.Component subclass, you should call super(props) before any other statement. Otherwise, this.props will be undefind in the constructor, which can lead to bugs.
- React constructors are only used for two purpose -
- Initializing local state by assigning an object to this.state.

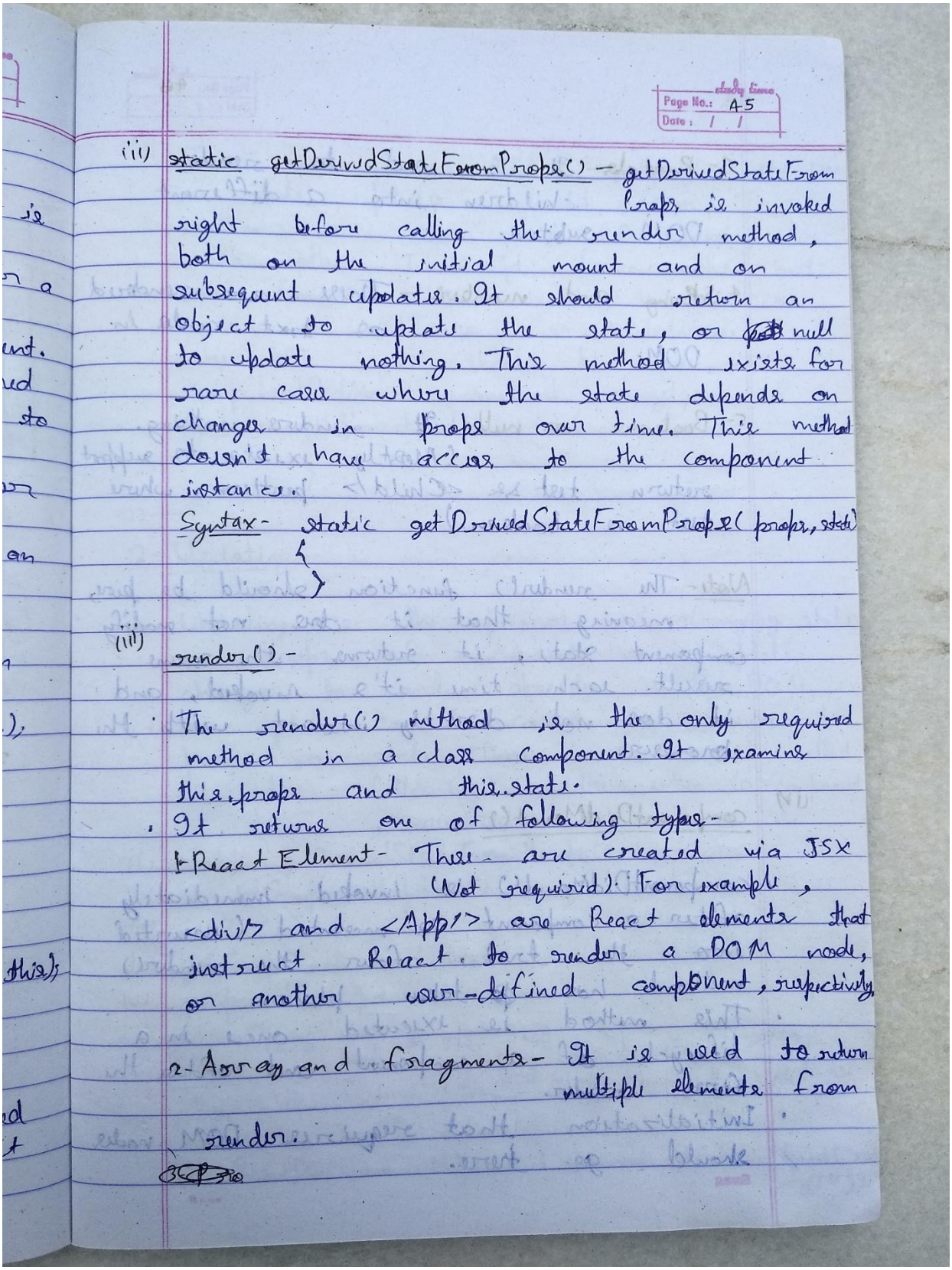
Ex- this.State = {name: "Rahul"}

- Binding event handler methods to an instance.

Ex- this.handleClick = this.handleClick.bind(this);

```
Ex- constructor(props) {
        super(props);
        this.state = {
            name: "Rahul",
            roll: this.props.roll
        };
        this.handleClick = this.handleClick.bind(this);
    }
```

- If you don't initialize state and you don't bind methods, you don't need to implement a constructor for your React component.



(iii) static getDerivedStateFromProps() – getDerivedStateFromProps is invoked right before calling the render method, both on the initial mount and on subsequent updates. It should return an object to update the state, or null to update nothing. This method exists for rare case where the state depends on changes in props over time. This method doesn't have access to the component instance.

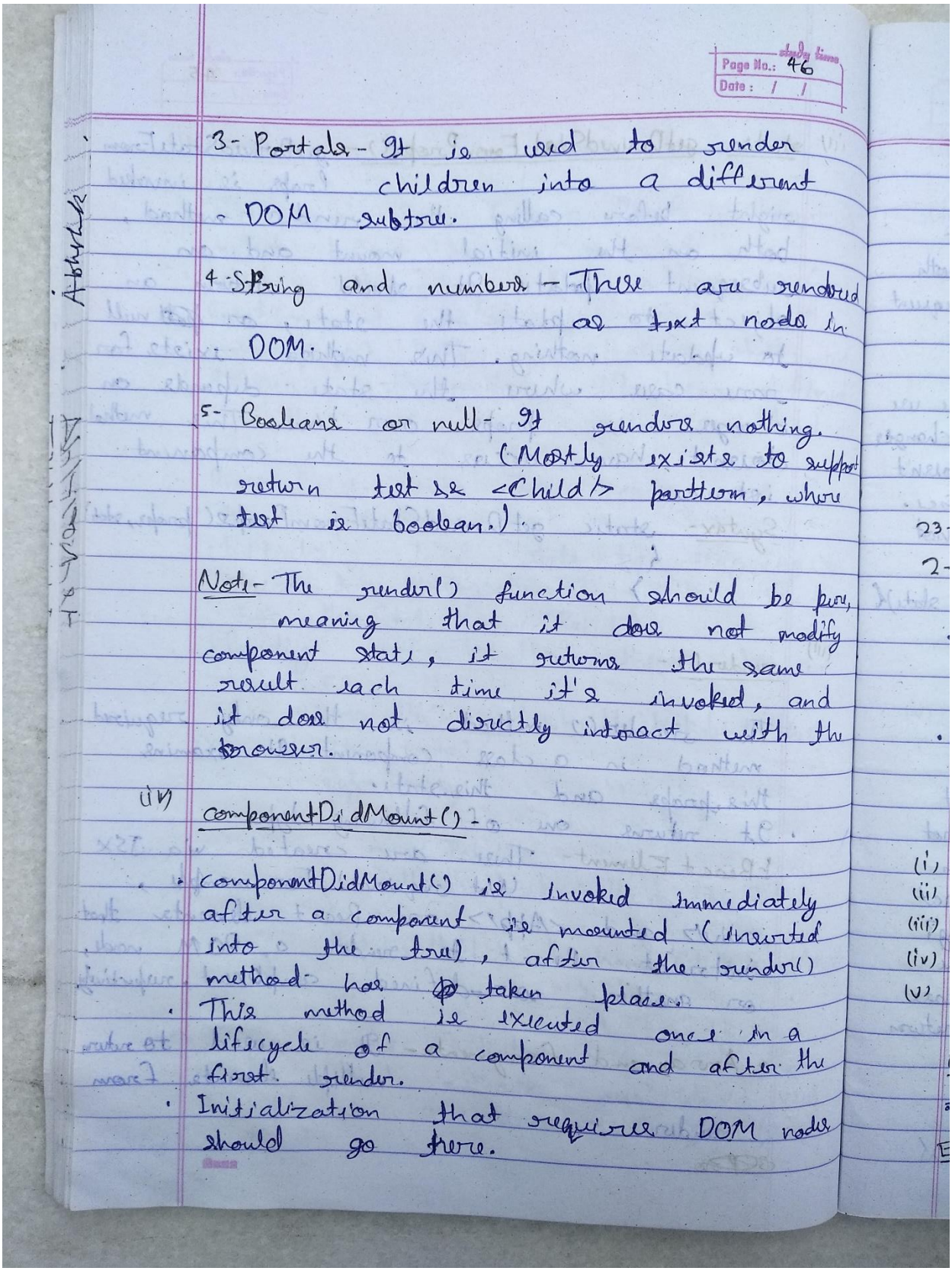
Syntax- static getDerivedStateFromProps(props, state)
{
}

(iii) render() –

- The render() method is the only required method in a class component. It examines this.props and this.state.
- It returns one of following types-

1- React Element- These are created via JSX (Not required). For example, and <App/> are React elements that instruct React to render a DOM node, or another user-defined component, respectively.

2- Array and fragments- It is used to return multiple elements from render.



Page No.: 46
Date: / /

3- Portals - It is used to render children into a different DOM subtree.

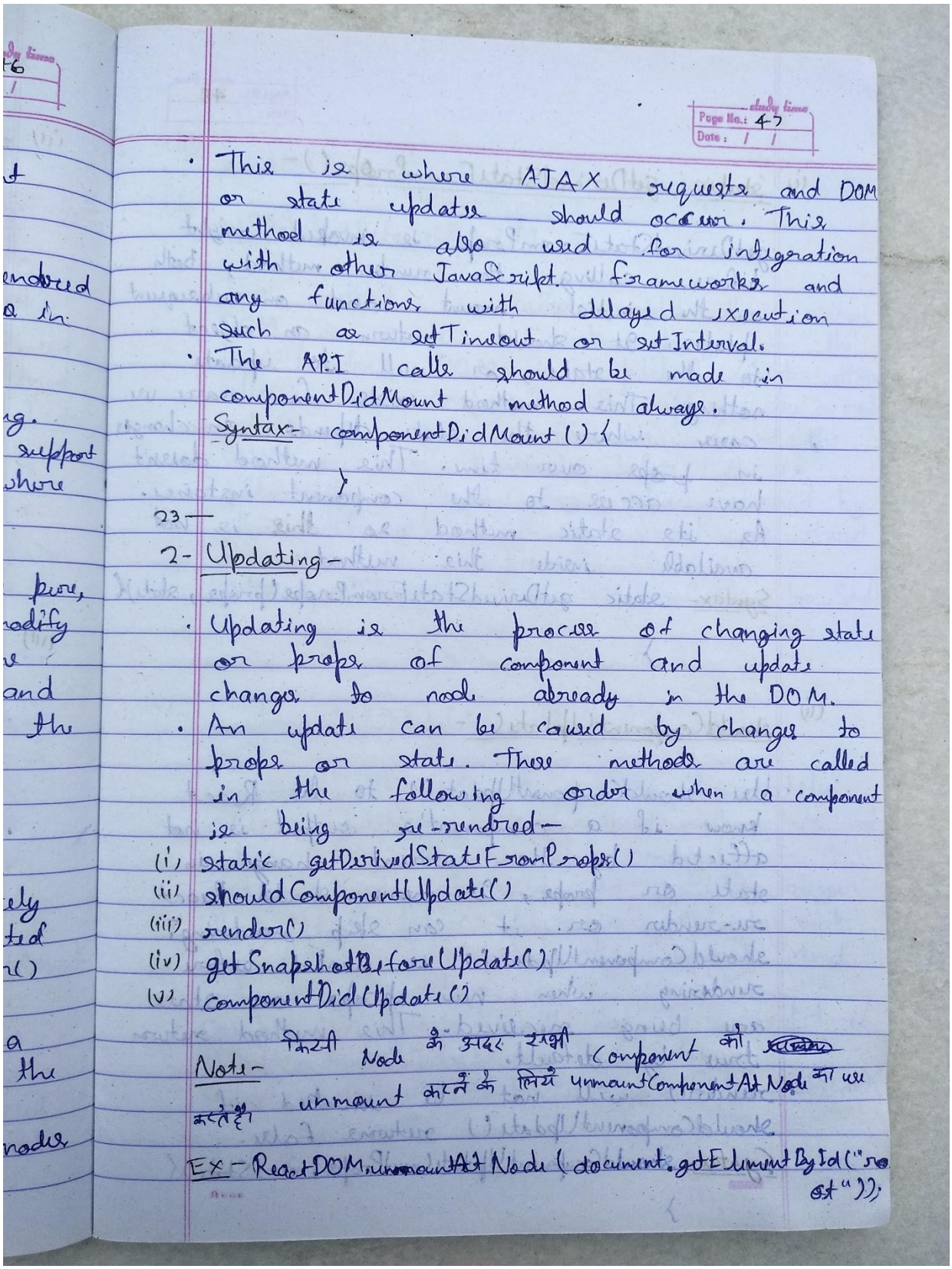
4- String and numbers - These are rendered as text node in DOM.

5- Booleans or null - It renders nothing. (Mostly exists to support return test && <Child /> pattern, where test is boolean.)

Note - The render() function should be pure, meaning that it does not modify component state, it returns the same result each time it's invoked, and it does not directly interact with the browser.

(iv) componentDidMount() -

- componentDidMount() is invoked immediately after a component is mounted (inserted into the tree), after the render() method has taken place.
- This method is executed once in a lifecycle of a component and after the first render.
- Initialization that requires DOM node should go here.



- This is where AJAX requests and DOM or state updates should occur. This method is also used for integration with other JavaScript frameworks and any functions with delayed execution such as setTimeout or setInterval.
- The API calls should be made in componentDidMount method always.

Syntax- componentDidMount() {
}

23—

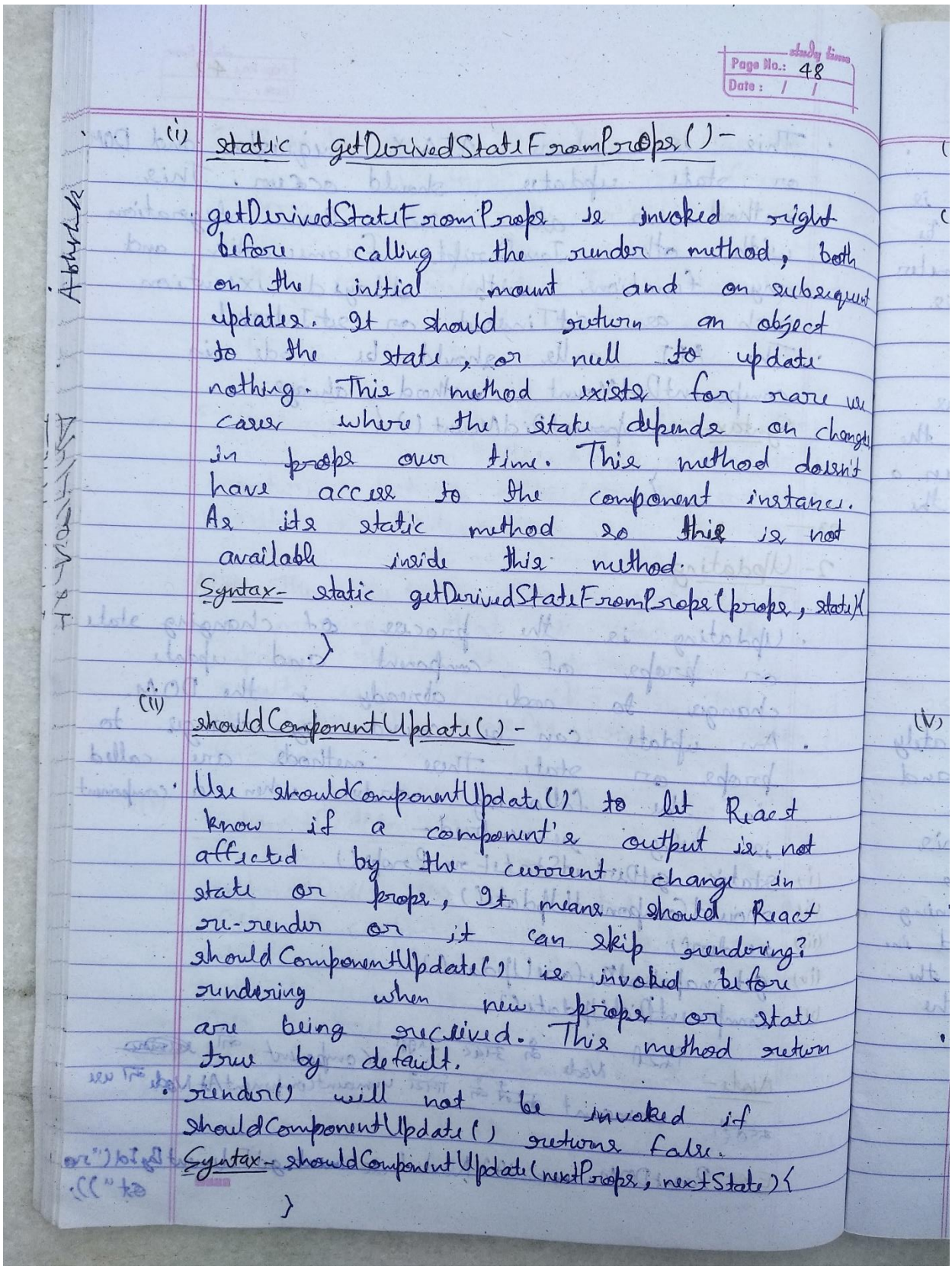
2- Updating-

- Updating is the process of changing state or props of component and update changes to node already in the DOM.
- An update can be caused by changes to props or state. These methods are called in the following order when a component is being re-rendered—

(i) static getDerivedStateFromProps()
(ii) shouldComponentUpdate()
(iii) render()
(iv) getSnapshotBeforeUpdate()
(v) componentDidUpdate()

Note- किसी Node के अंदर रखी Component को unmount करने के लिये unmountComponentAtNode का use करते है।

Ex- ReactDOM.unmountAtNode(document.getElementById("root"));



(i) static getDerivedStateFromProps()-

getDerivedStateFromProps is invoked right before calling the render method, both on the initial mount and on subsequent updates. It should return an object to the state, or null to update nothing. This method exists for rare use cases where the state depends on changes in props over time. This method doesn't have access to the component instance. As its static method so this is not available inside this method.

Syntax- static getDerivedStateFromProps(props, state){
}

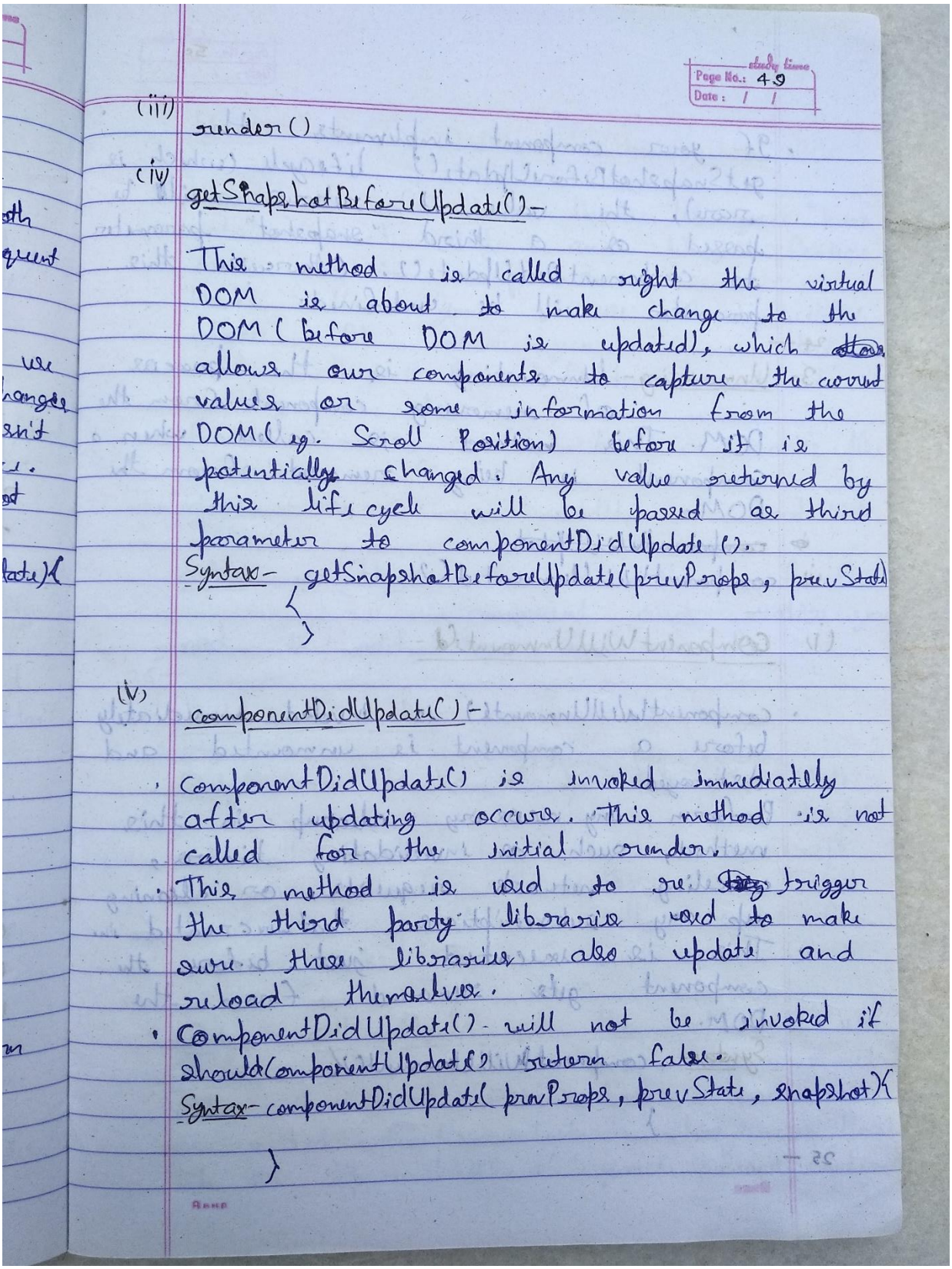
(ii) shouldComponentUpdate()-

Use shouldComponentUpdate() to let React know if a component's output is not affected by the current change in state or props, It means should React re-render or it can skip rendering? shouldComponentUpdate() is invoked before rendering when new props or state are being received. This method return true by default.

render() will not be invoked if shouldComponentUpdate() returns false.

Syntax- shouldComponentUpdate(nextProps, nextState){
}

(iv)



(iii) render()

(iv) getSnapshotBeforeUpdate()-

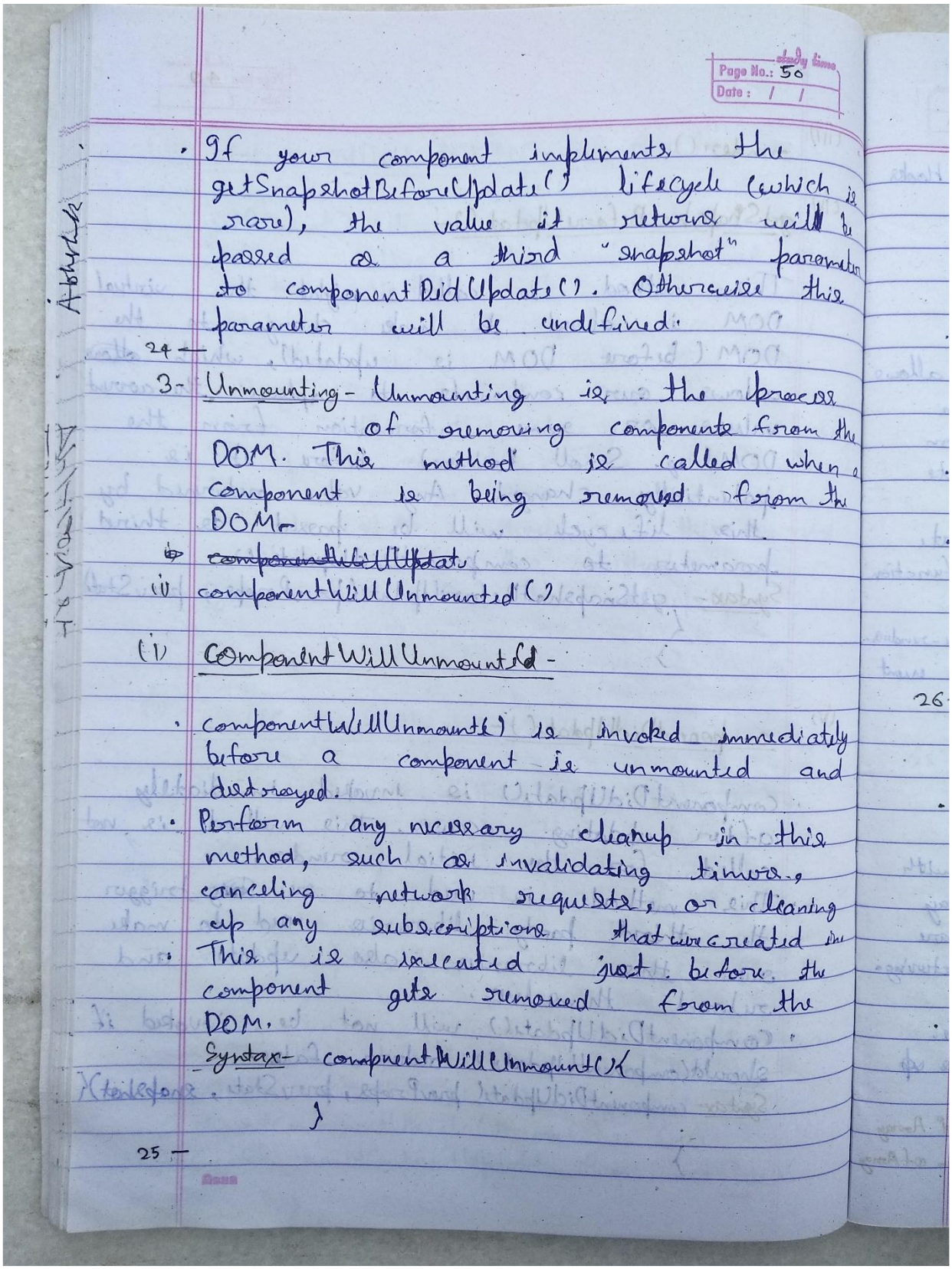
This method is called right the virtual DOM is about to make change to the DOM (before DOM is updated), which allows our components to capture the current values or some information from the DOM (eg. Scroll Position) before it is potentially changed. Any value returned by this life cycle will be passed as third parameter to componentDidUpdate().

Syntax- getSnapshotBeforeUpdate(prevProps, prevState)
{
}

(v) componentDidUpdate()-

- componentDidUpdate() is invoked immediately after updating occurs. This method is not called for the initial render.
- This method is used to trigger the third party libraries used to make sure these libraries also update and reload themselves.
- componentDidUpdate() will not be invoked if shouldComponentUpdate() return false.

Syntax- componentDidUpdate(prevProps, prevState, snapshot){
}



• If your component implements the getSnapshotBeforeUpdate() lifecycle (which is rare), the value it returns will be passed as a third "snapshot" parameter to componentDidUpdate(). Otherwise this parameter will be undefined.

24 —

3- Unmounting - Unmounting is the process of removing components from the DOM. This method is called when a component is being removed from the DOM.

(i) ~~componentWillUpdate~~

(ii) componentWillUnmounted()

(i) ComponentWillUnmounted -

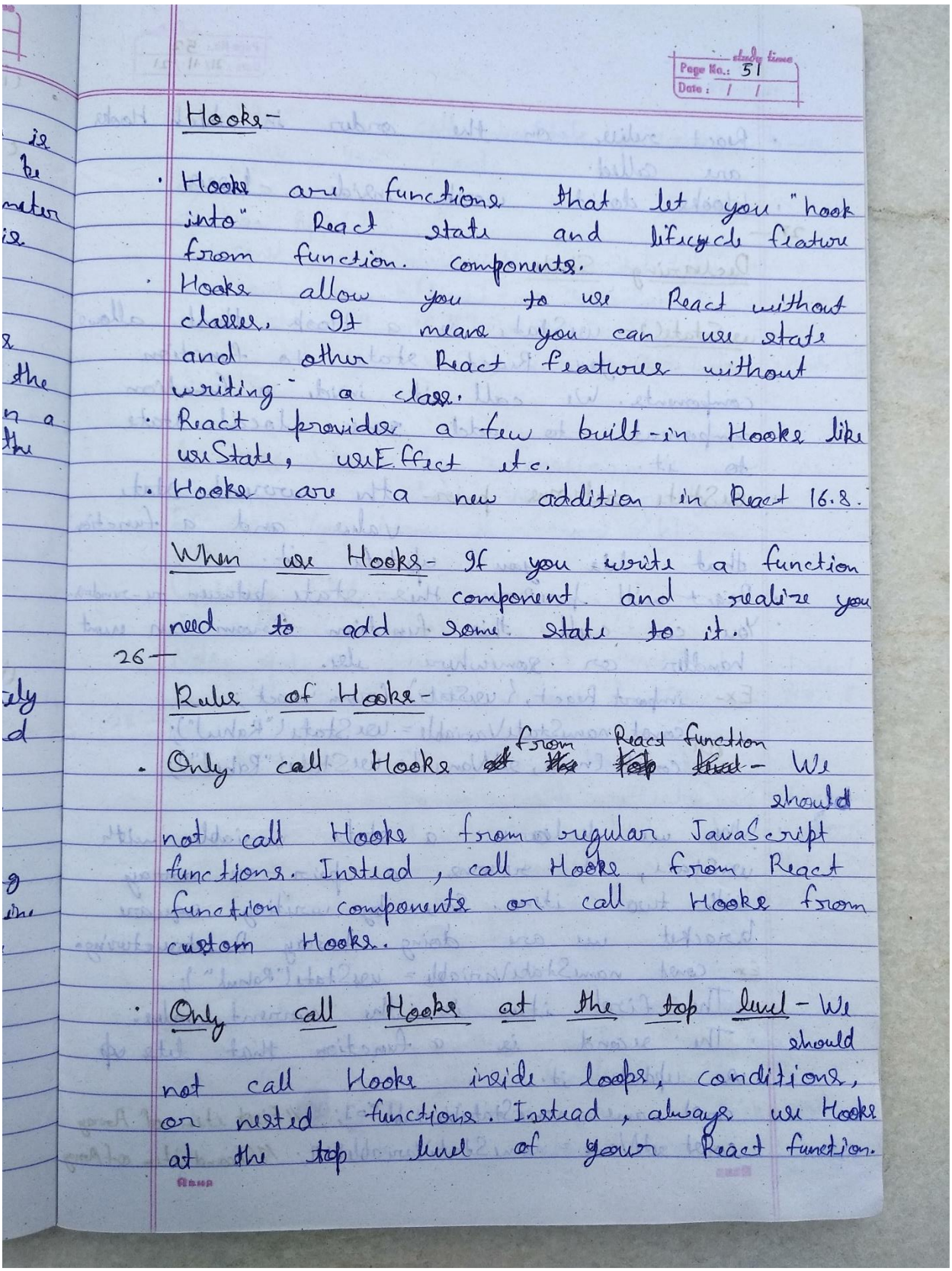
26

• componentWillUnmount() is invoked immediately before a component is unmounted and destroyed.
• Perform any necessary cleanup in this method, such as invalidating timers, canceling network requests, or cleaning up any subscriptions that were created in
• This is executed just before the component gets removed from the DOM.

Syntax- componentWillUnmount(){

}

25 —



Hooks-

- Hooks are functions that let you "hook into" React state and lifecycle feature from function components.
- Hooks allow you to use React without classes. It means you can use state and other React features without writing a class.
- React provides a few built-in Hooks like useState, useEffect etc.
- Hooks are a new addition in React 16.8.

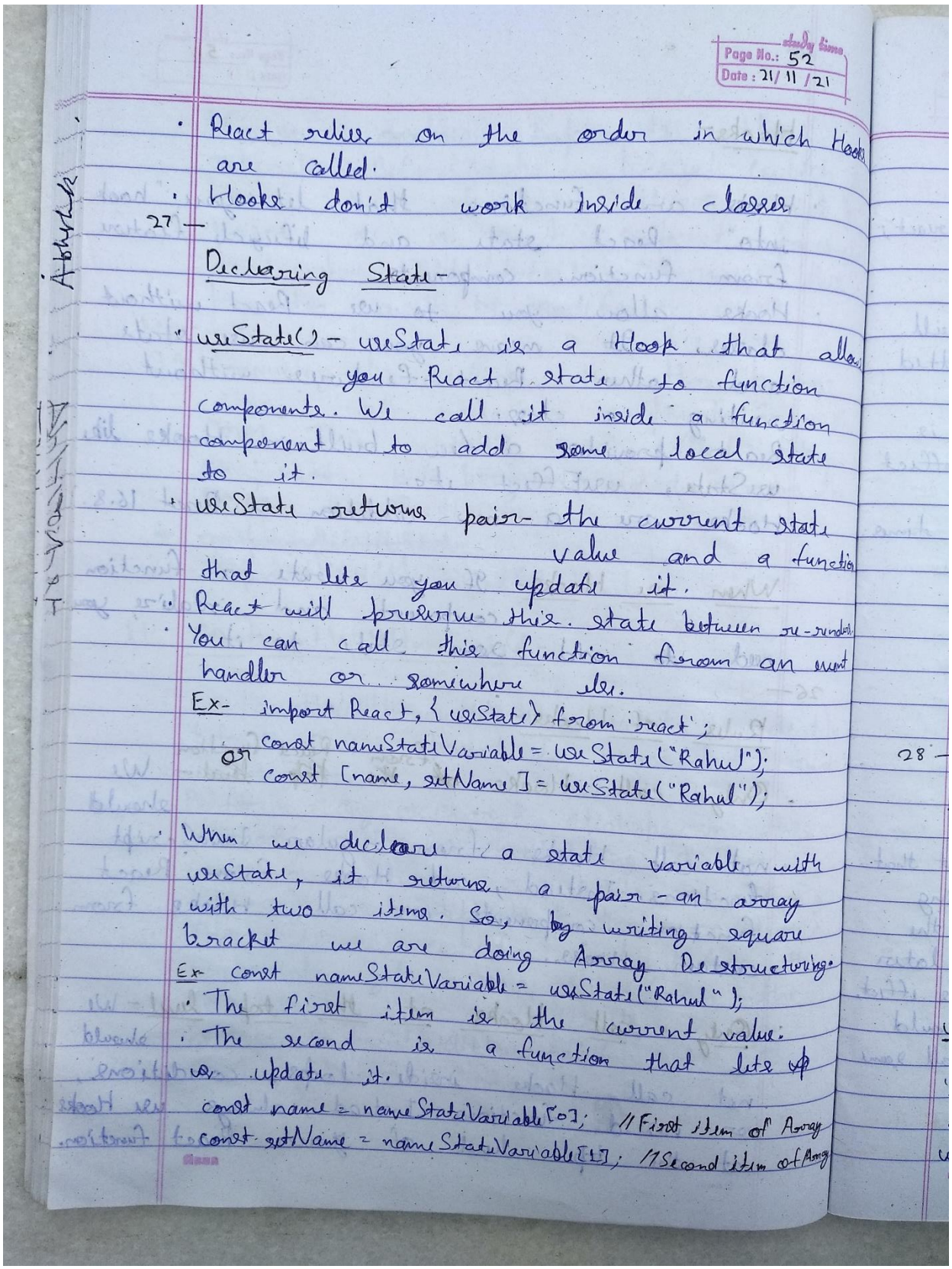
When use Hooks- If you write a function component and realize you need to add some state to it.

26—

Rules of Hooks-

- Only call Hooks ~~at the top level~~ from React function - We should not call Hooks from regular JavaScript functions. Instead, call Hooks from React function components or call Hooks from custom Hooks.

- Only call Hooks at the top level - We should not call Hooks inside loops, conditions, or nested functions. Instead, always use Hooks at the top level of your React function.



Page No.: 52
Date : 21/11/21

- React relies on the order in which Hooks are called.
- Hooks don't work inside classes.

27 —

Declaring State -

- useState() - useState is a Hook that allows you React state to function components. We call it inside a function component to add some local state to it.
- useState returns pair- the current state value and a function that lets you update it.
- React will preserve this state between re-renders.
- You can call this function from an event handler or somewhere else.

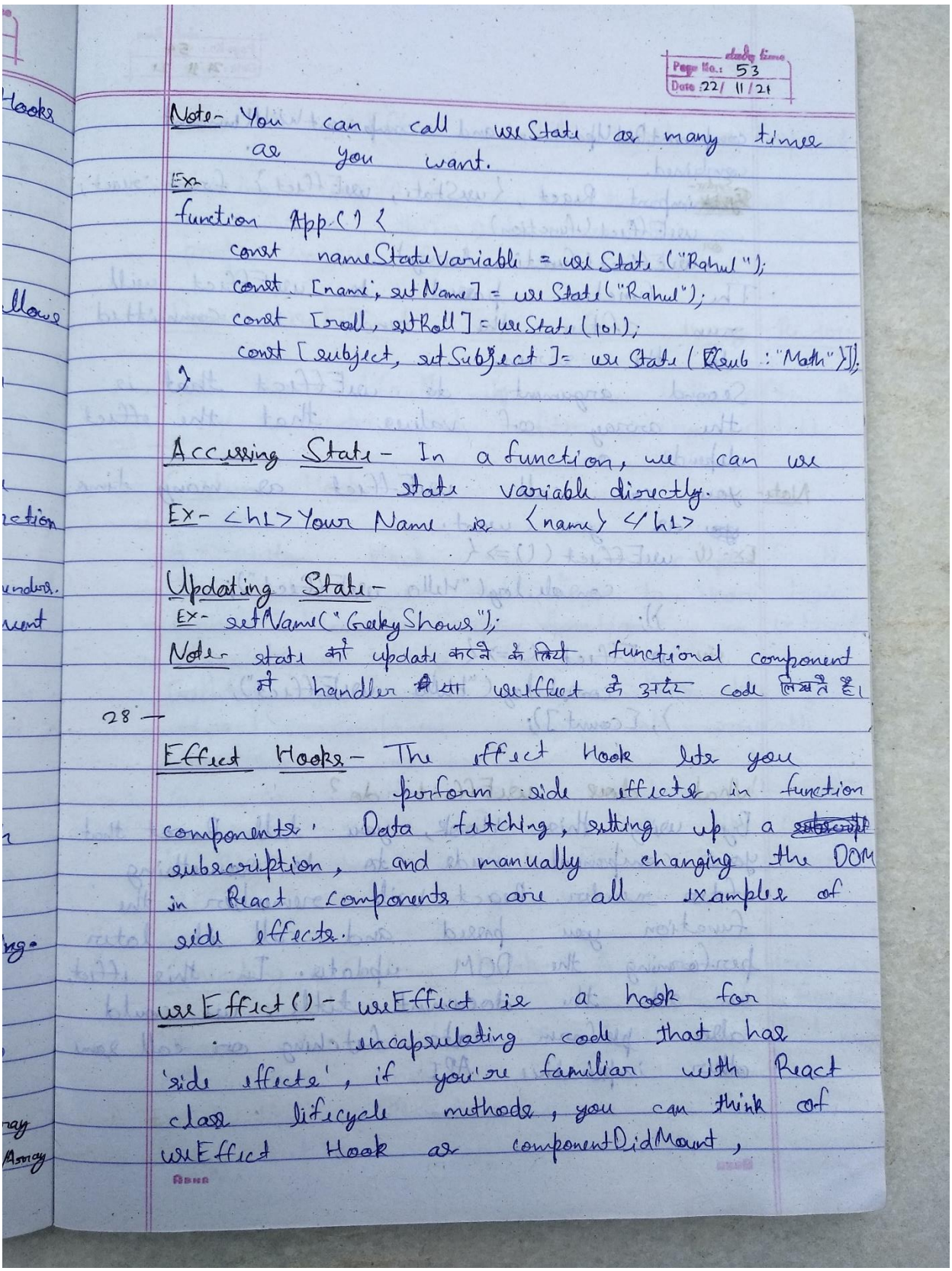
Ex- import React, { useState } from 'react';
or const nameStateVariable = useState("Rahul");
const [name, setName] = useState("Rahul");

28 —

- When we declare a state variable with useState, it returns a pair - an array with two items. So, by writing square bracket we are doing Array Destructuring.

Ex- const nameStateVariable = useState("Rahul");
- The first item is the current value.
- The second is a function that lets us update it.

const name = nameStateVariable[0]; // First item of Array
const setName = nameStateVariable[1]; // Second item of Array



Page No.: 53
Date: 22/ 11/ 21

Note- You can call useState as many times as you want.

Ex-

```
function App() {
    const nameStateVariable = useState("Rahul");
    const [name, setName] = useState("Rahul");
    const [roll, setRoll] = useState(101);
    const [subject, setSubject] = useState({sub: "Math"});
}
```

Accessing State- In a function, we can use state variable directly.

Ex- <h1> Your Name is {name} </h1>

Updating State-

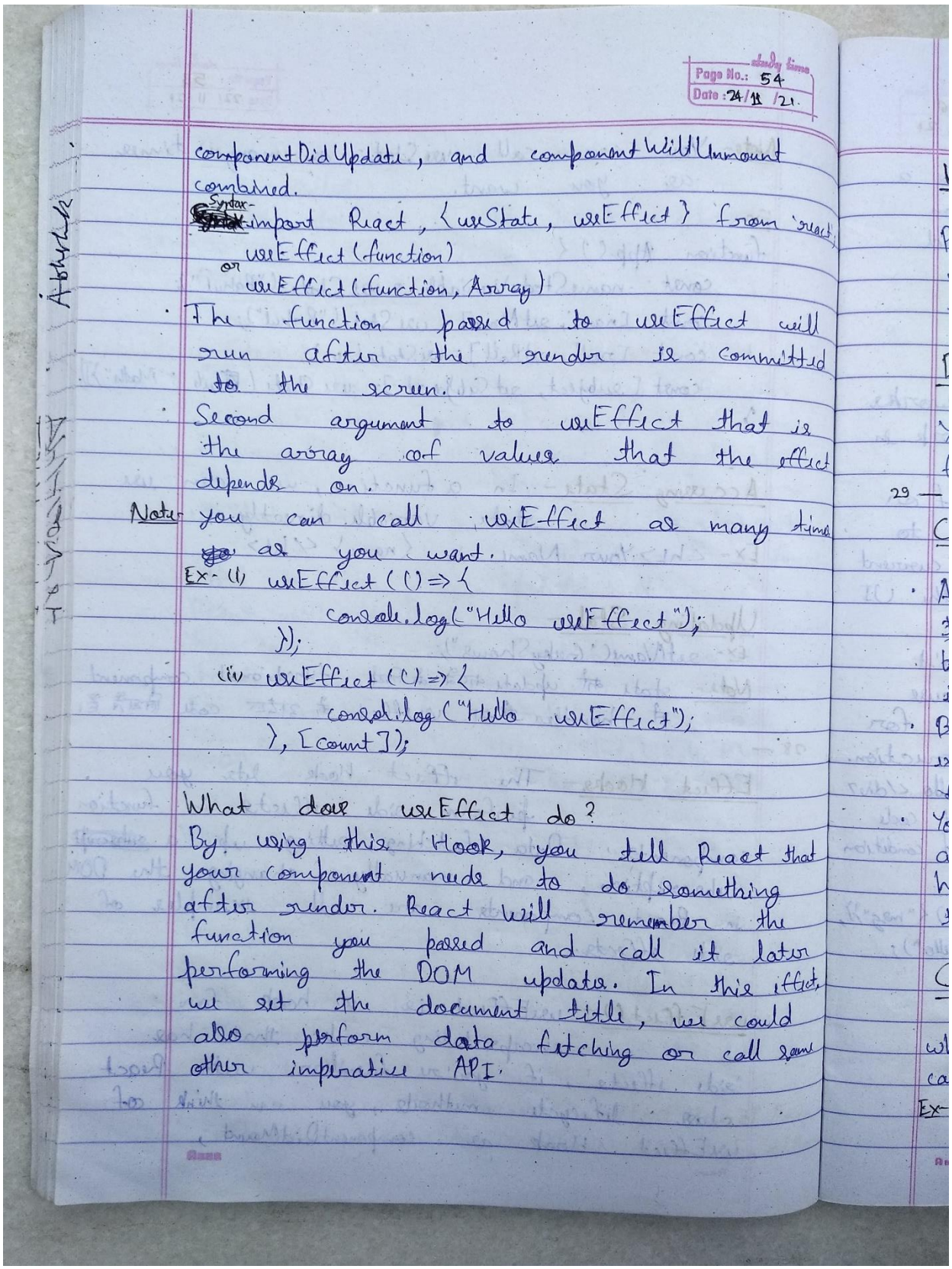
Ex- setName("Geeky Shows");

Note- state को update करने के लिए functional component में handler या useEffect के अंदर code लिखते हैं।

28 —

Effect Hooks- The Effect Hook lets you perform side effects in function components. Data fetching, setting up a subscription, and manually changing the DOM in React components are all examples of side effects.

useEffect()- useEffect is a hook for encapsulating code that has 'side effects', if you're familiar with React class lifecycle methods, you can think of useEffect Hook as componentDidMount,



componentDidUpdate, and componentWillUnmount combined.

Syntax-

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

useEffect(function)

or

useEffect(function, Array)

- The function passed to useEffect will run after the render is committed to the screen.
- Second argument to useEffect that is the array of values that the effect depends on.

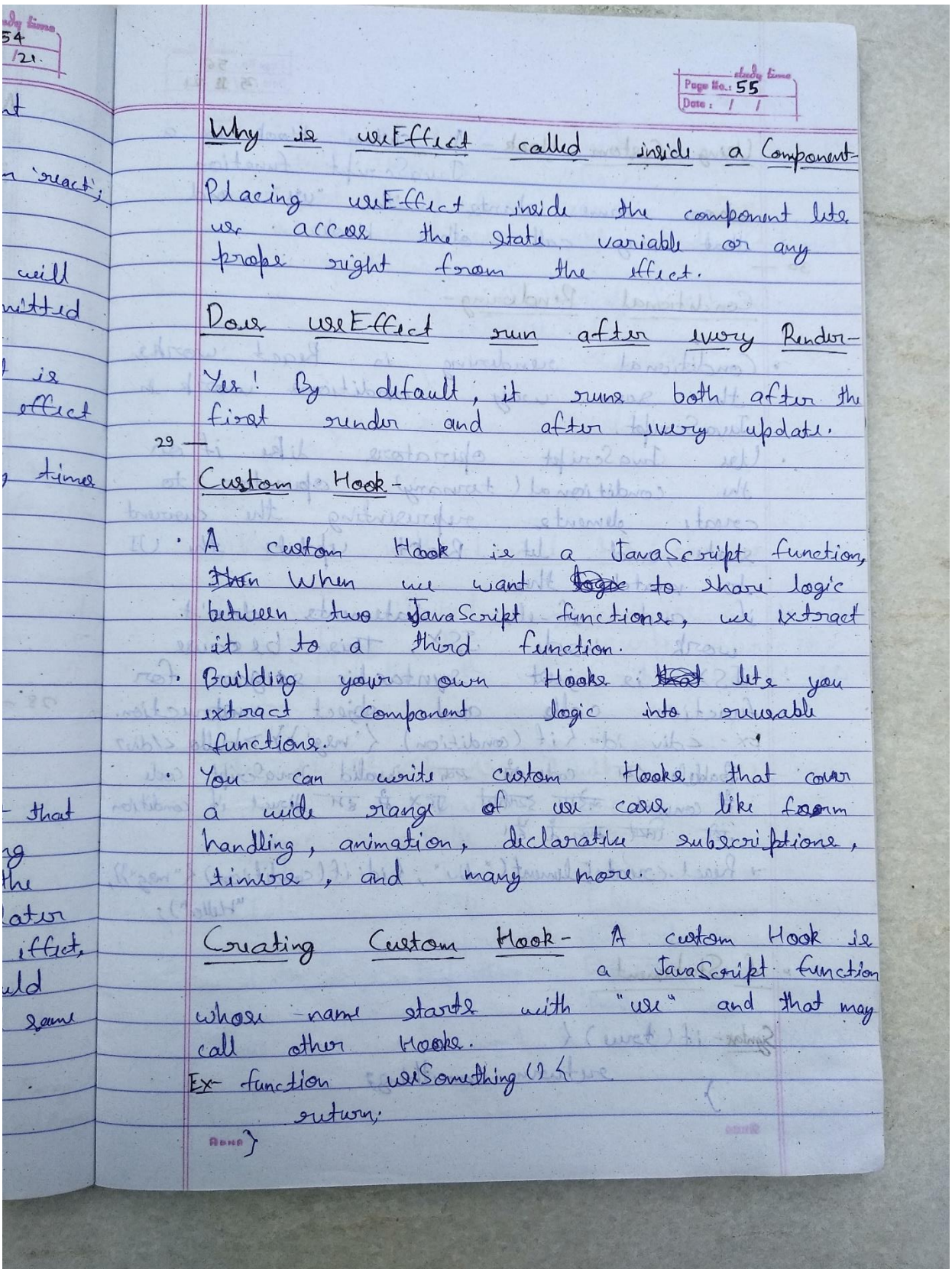
Note- you can call useEffect as many times as you want.

Ex- (i) useEffect(() => {
    console.log("Hello useEffect");
});

(ii) useEffect(() => {
    consol.log("Hello useEffect");
}, [count]);

What does useEffect do?

By using this Hook, you tell React that your component needs to do something after render. React will remember the function you passed and call it later performing the DOM updates. In this effect, we set the document title, we could also perform data fetching or call some other imperative API.



Why is useEffect called inside a Component-

Placing useEffect inside the component lets us access the state variable or any prope right from the effect.

Does useEffect run after every Render-

Yes! By default, it runs both after the first render and after every update.

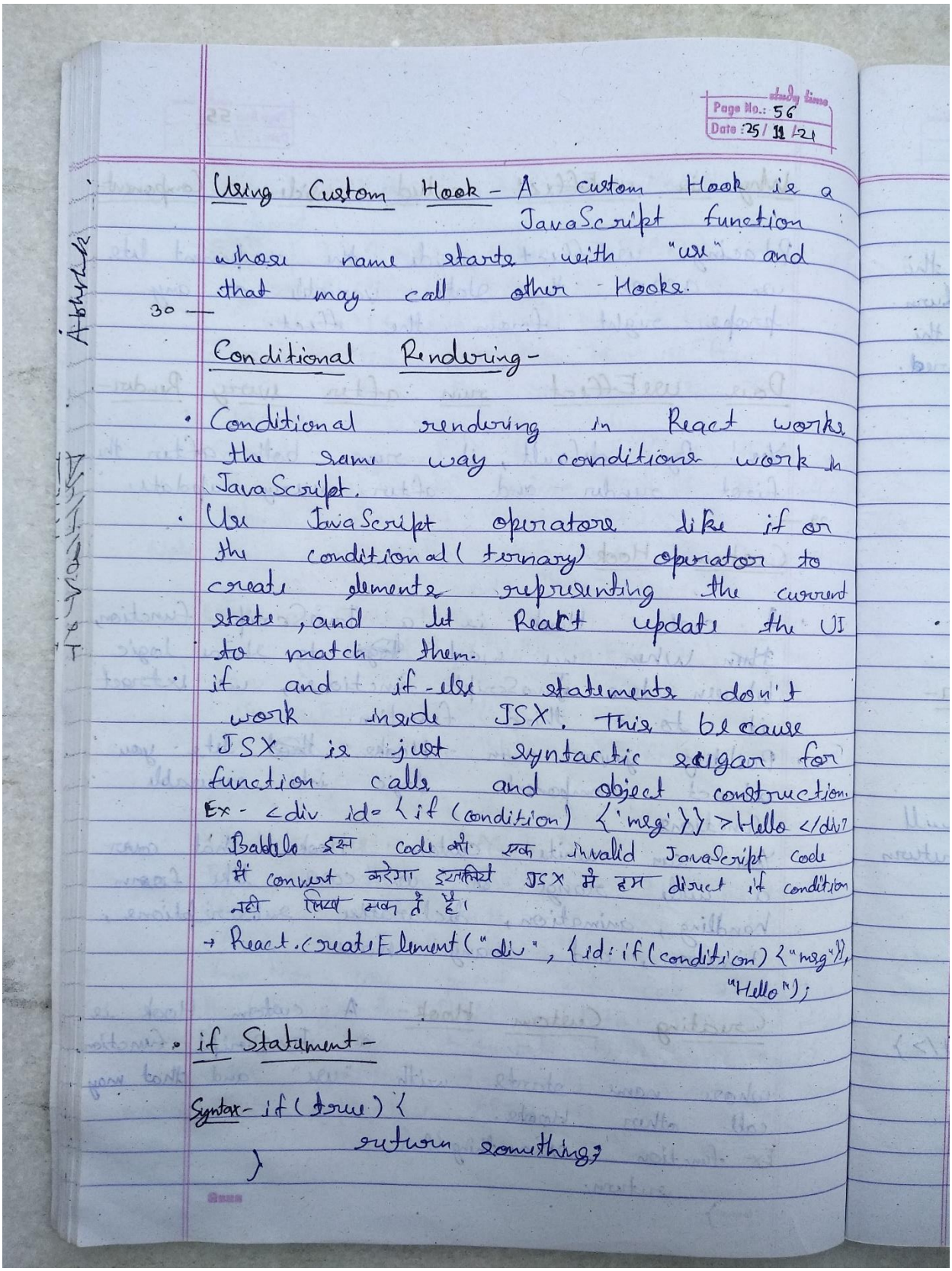
29 —

Custom Hook-

- A custom Hook is a JavaScript function, When we want to share logic between two JavaScript functions, we extract it to a third function.
- Building your own Hooks lets you extract component logic into reusable functions.
- You can write custom Hooks that cover a wide range of use case like form handling, animation, declarative subscriptions, timers, and many more.

Creating Custom Hook- A custom Hook is a JavaScript function whose name starts with "use" and that may call other Hooks.

Ex- function useSomething () {
    return;
}



Page No.: 56
Date: 25/11/21

Using Custom Hook - A custom Hook is a JavaScript function whose name starts with "use" and that may call other Hooks.

30 —

Conditional Rendering -

- Conditional rendering in React works the same way conditions work in JavaScript.
- Use JavaScript operators like if or the conditional (ternary) operator to create elements representing the current state, and let React update the UI to match them.
- if and if-else statements don't work inside JSX. This because JSX is just syntactic sugar for function calls and object construction.

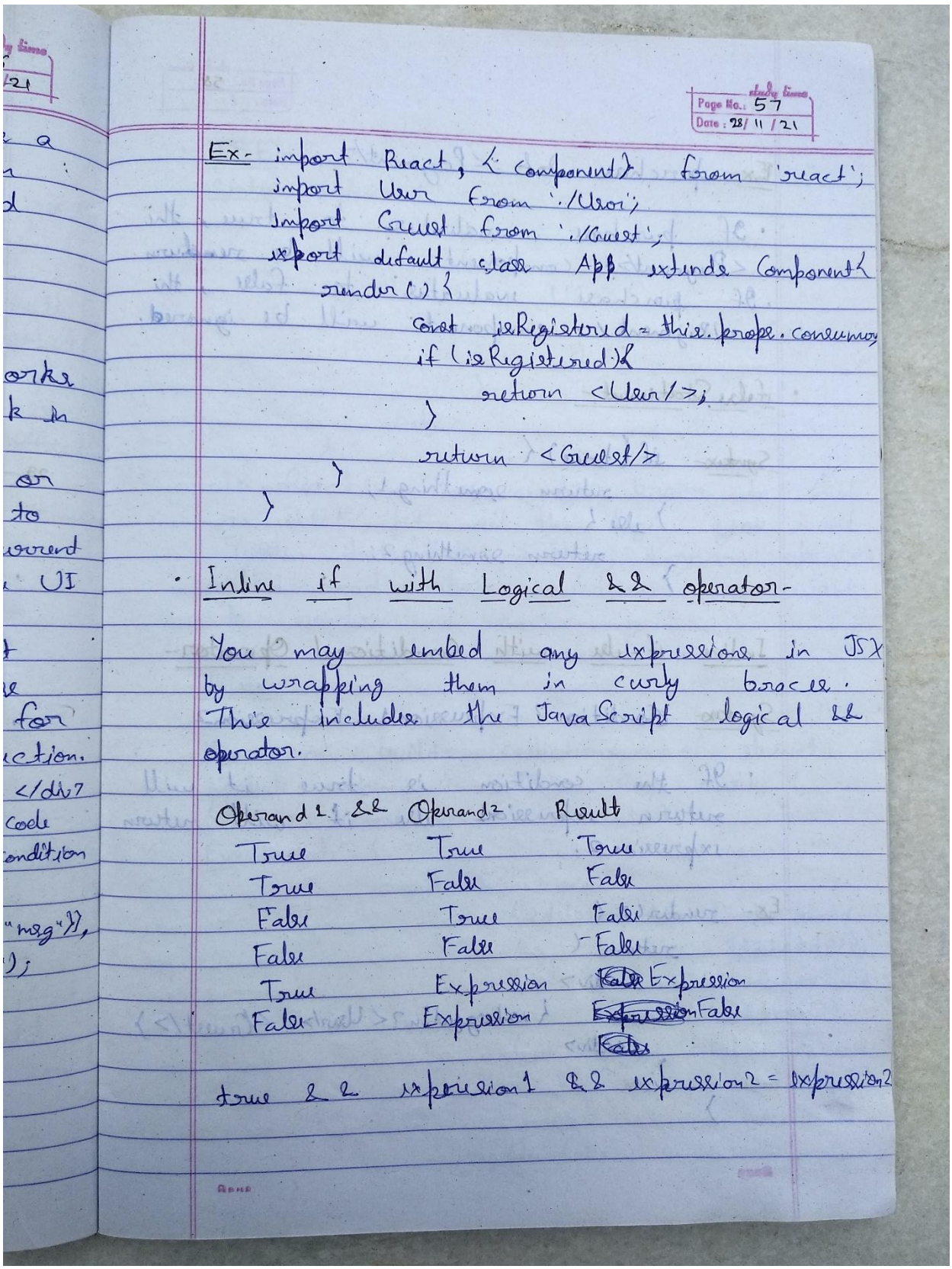
Ex - <div id={if (condition) {'msg'}}>Hello

Babble इस code को एक invalid JavaScript code में convert करेगा इसलिए JSX में हम direct if condition नहीं लिख सकते हैं।

→ React.createElement("div", {id: if(condition) {"msg"}}, "Hello");

- if Statement -

Syntax - if (true) {
    return something;
}



Page No.: 57
Date: 28/11/21

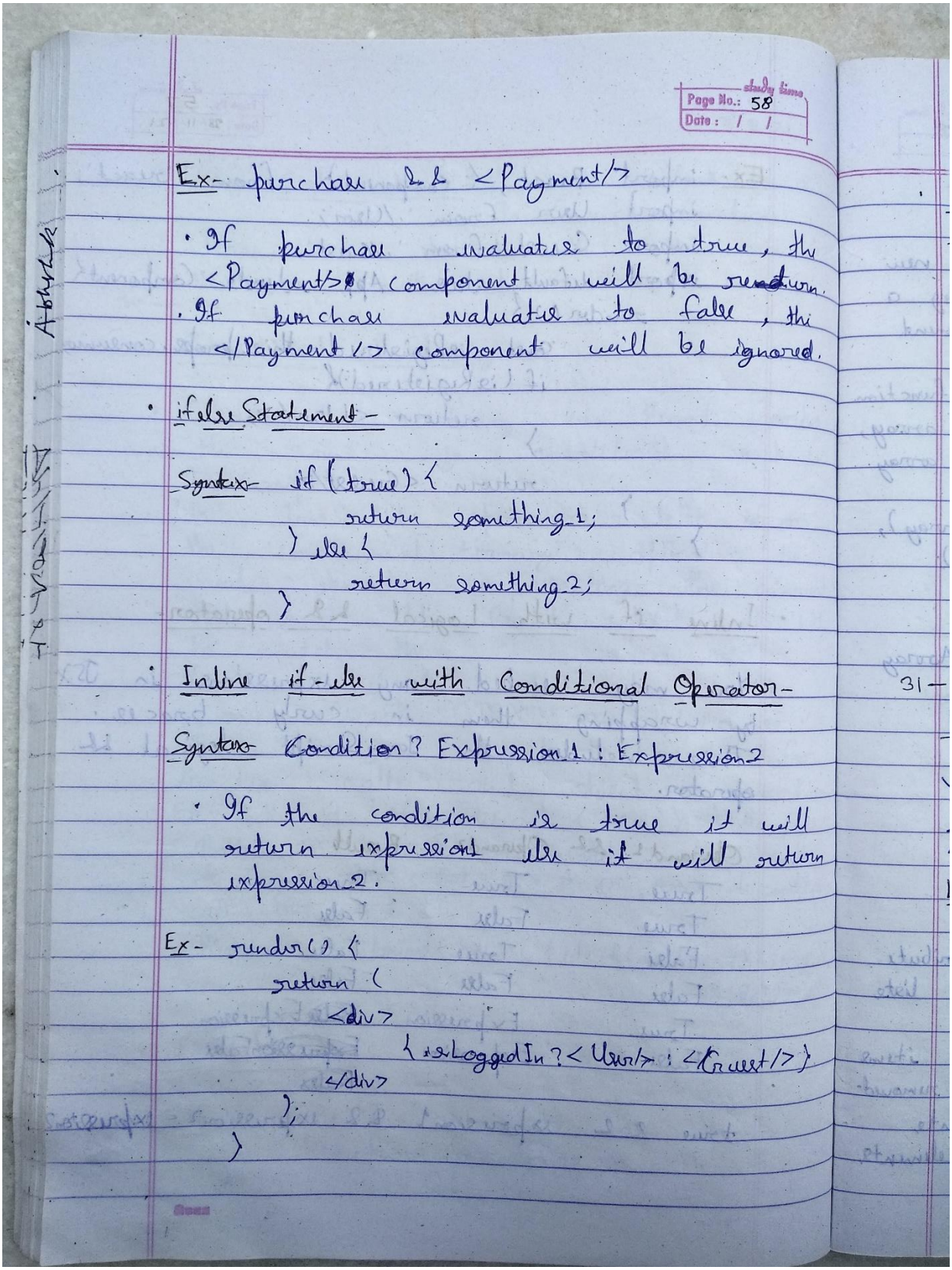
Ex- import React, { Component } from 'react';
import User from './User';
import Guest from './Guest';
export default class App extends Component {
render() {
const isRegistered = this.props.consumer;
if (isRegistered) {
return <User />;
}
return <Guest />
}
}

Inline if with Logical && operator-

You may embed any expressions in JSX by wrapping them in curly braces. This includes the JavaScript logical && operator.

Operand1	&&	Operand2	Result
True		True	True
True		False	False
False		True	False
False		False	False
True		Expression	Expression
False		Expression	False

true && expression1 && expression2 = expression2



Page No.: 58
Date: / /

Ex- purchase && <Payment/>

- If purchase evaluates to true, the <Payment/> component will be returned.
- If purchase evaluates to false, the </Payment/> component will be ignored.

- **if else Statement -**

Syntax- if (true) {
    return something_1;
} else {
    return something_2;
}

- **Inline if-else with Conditional Operator-**

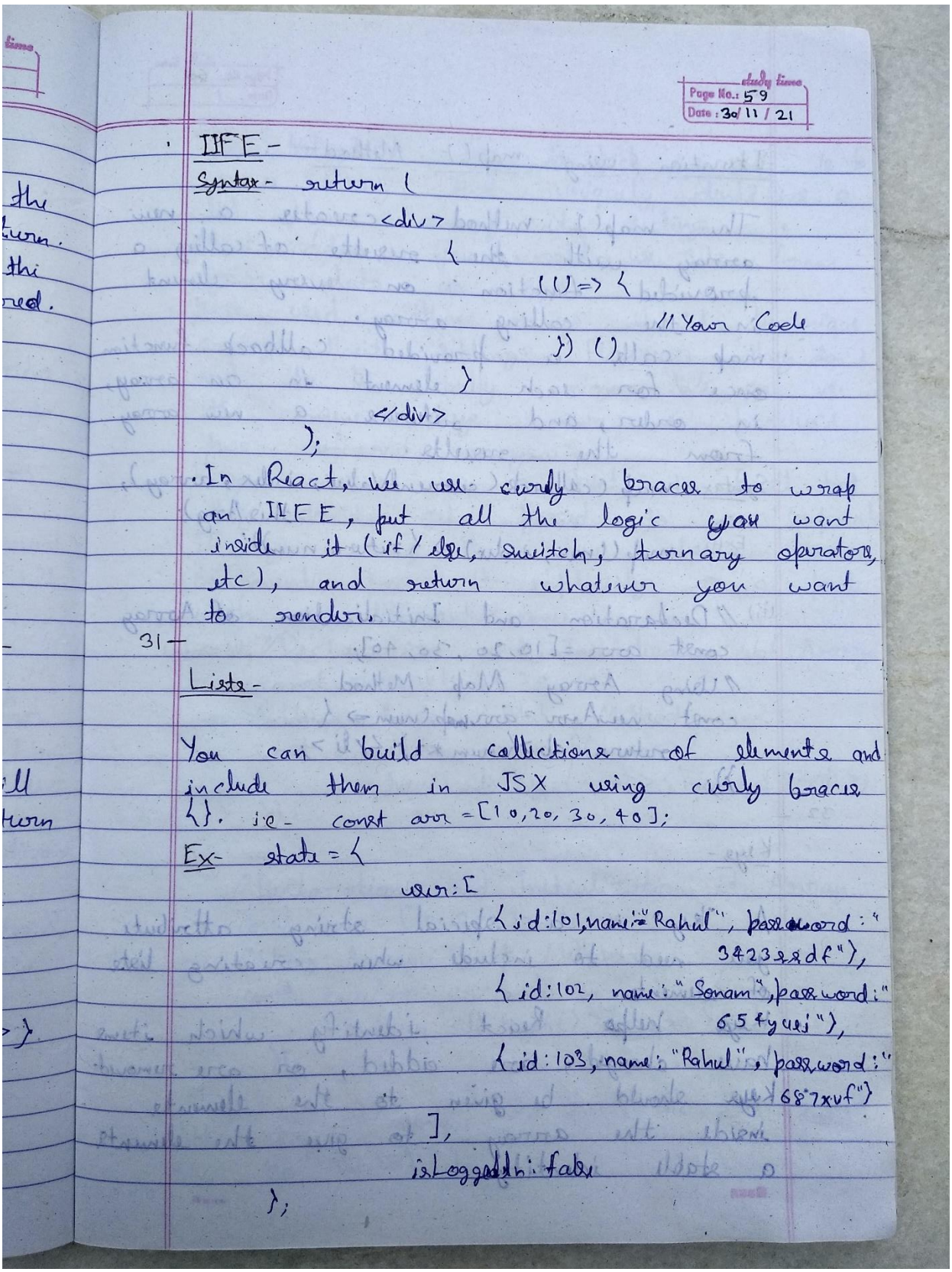
Syntax- Condition ? Expression_1 : Expression_2

- If the condition is true it will return expression1 else it will return expression_2.

Ex- render() {
    return (

            {isLoggedIn ? <User/> : <Guest/>}

    );
}



IIFE -

Syntax - return (

{
(()=> {
// Your Code
})()
}

);

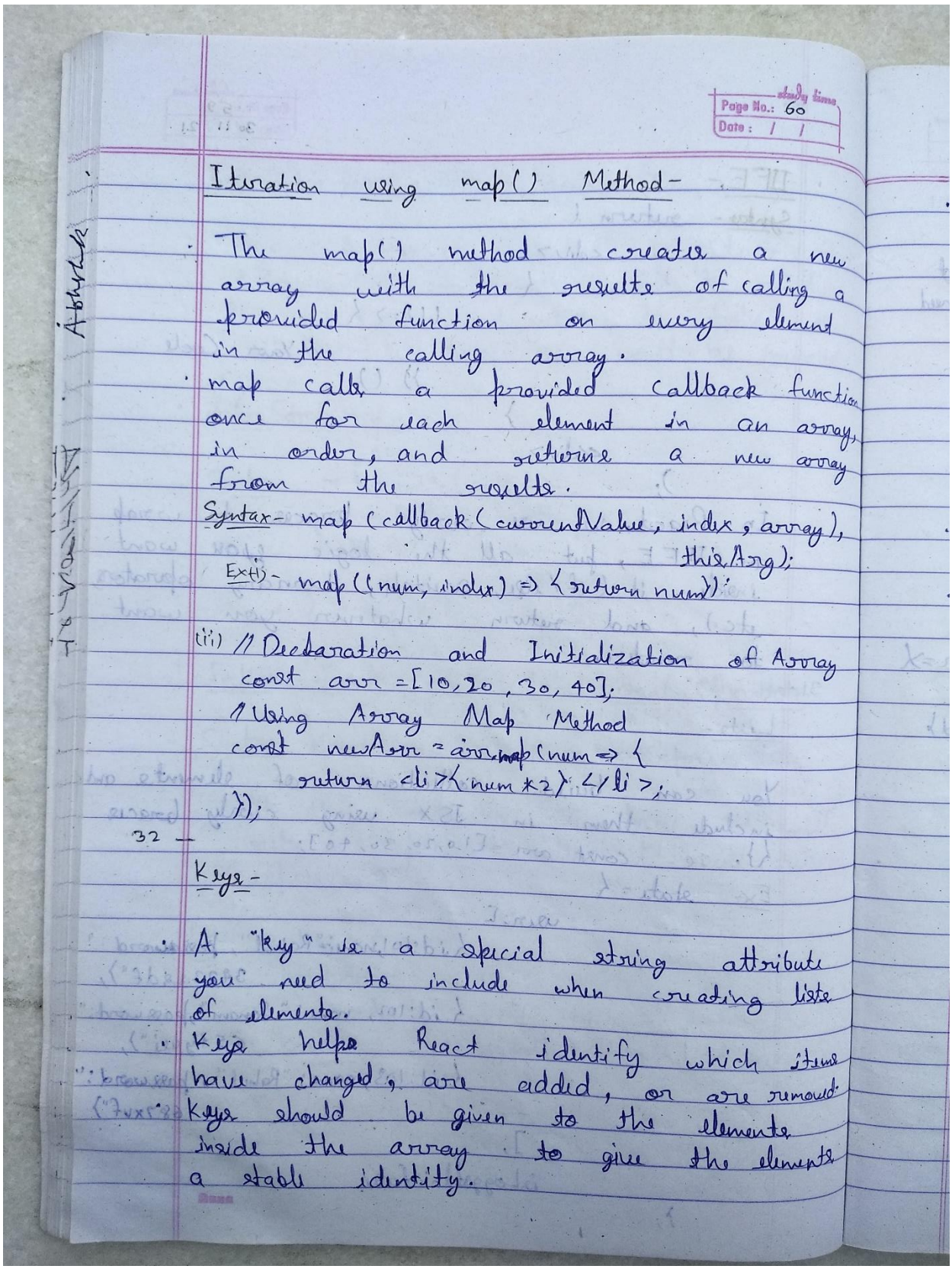
- In React, we use curly braces to wrap an IIFE, put all the logic you want inside it (if/else, switch, ternary operators, etc), and return whatever you want to render.

31 -

Lists -

You can build collections of elements and include them in JSX using curly braces {}. ie - const arr = [10, 20, 30, 40];

Ex- state = {
user: [
{id: 101, name: "Rahul", password: "3423zsdf"},
{id: 102, name: "Sonam", password: "654fyuki"},
{id: 103, name: "Rahul", password: "687xvf"}
],
isLoggedIn: false
};



Page No.: 60
Date: / /

Iteration using map() Method-

- The map() method creates a new array with the results of calling a provided function on every element in the calling array.
- map calls a provided callback function once for each element in an array, in order, and returns a new array from the results.

Syntax- map(callback(currentValue, index, array), thisArg);

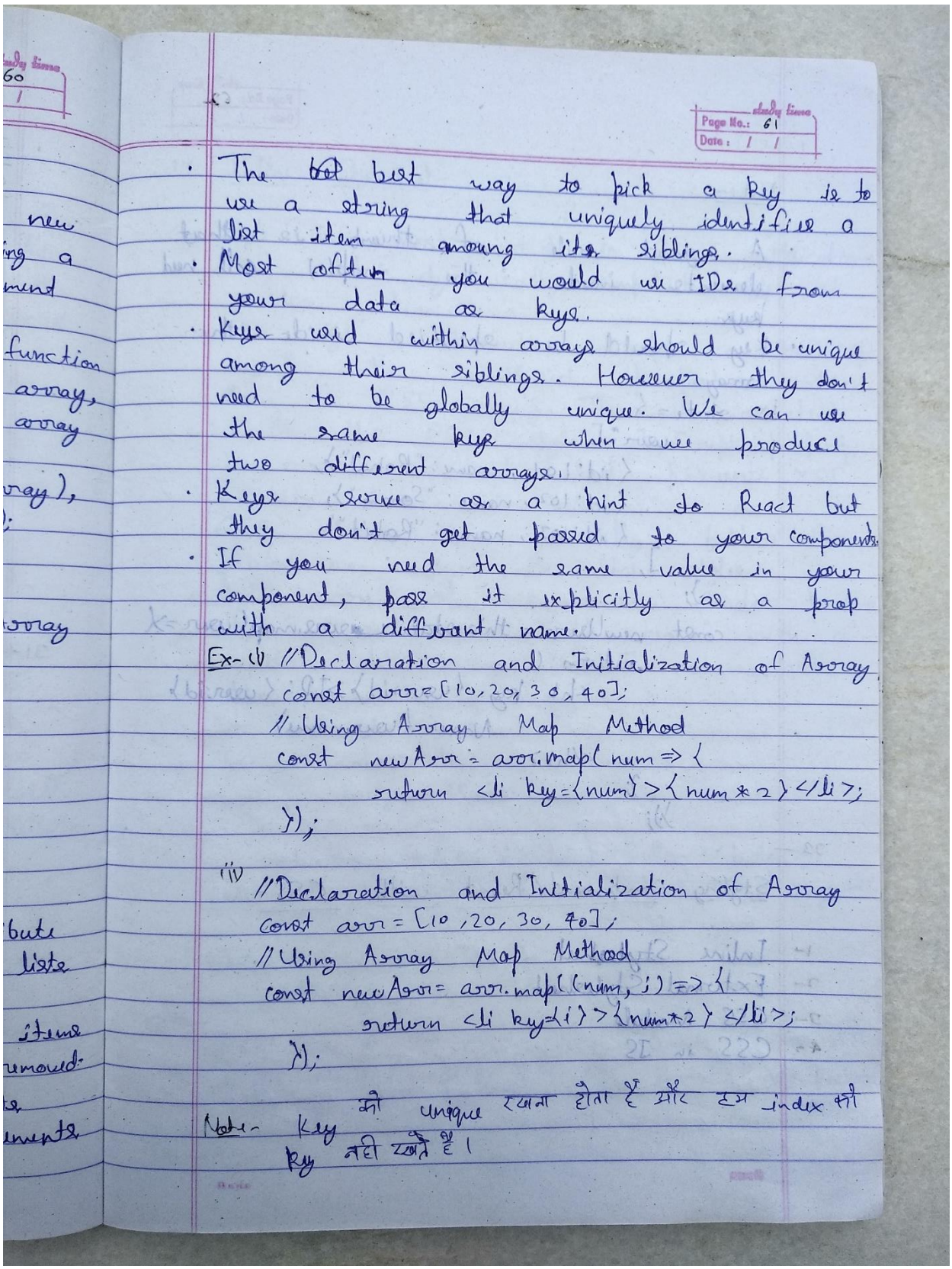
Ex(i)- map((num, index) => {return num});

(ii) // Declaration and Initialization of Array
const arr = [10, 20, 30, 40];
// Using Array Map Method
const newArr = arr.map(num => {
return {num * 2};
});

32 —

Keys-

- A "key" is a special string attribute you need to include when creating lists of elements.
- Keys helps React identify which items have changed, are added, or are removed. Keys should be given to the elements inside the array to give the elements a stable identity.

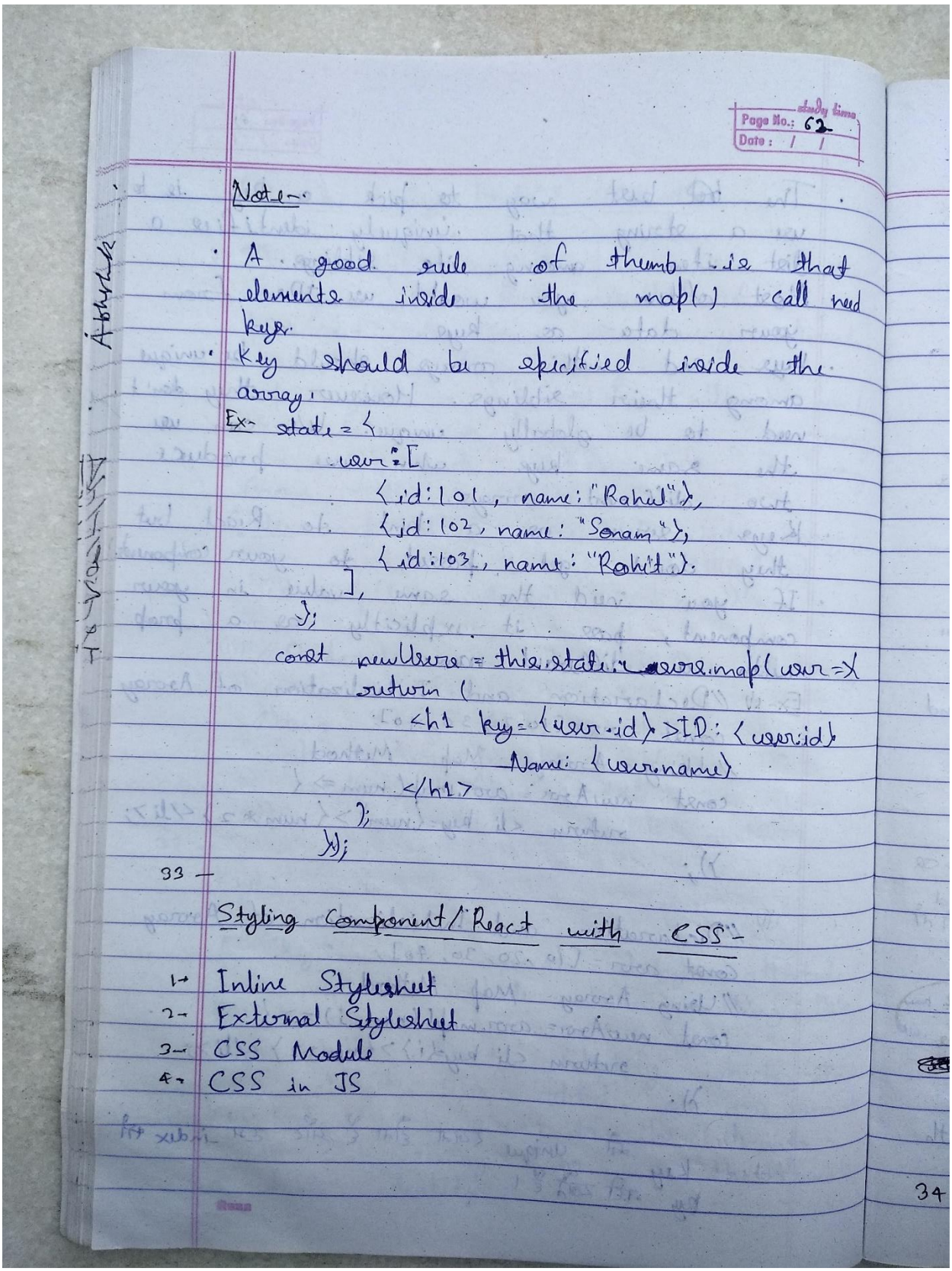


- The best way to pick a key is to use a string that uniquely identifies a list item among its siblings.
- Most of time you would use IDs from your data as keys.
- Keys used within arrays should be unique among their siblings. However they don't need to be globally unique. We can use the same keys when we produce two different arrays.
- Keys serve as a hint to React but they don't get passed to your components.
- If you need the same value in your component, pass it explicitly as a prop with a different name.

Ex- (i) //Declaration and Initialization of Array
const arr=[10,20,30,40];
//Using Array Map Method
const newArr = arr.map(num => {
return <li key={num}>{num*2};
});

(ii) //Declaration and Initialization of Array
const arr = [10,20,30,40];
//Using Array Map Method
const newArr = arr.map((num,i) => {
return <li key={i}>{num*2};
});

Note- Key को unique रखना होता है और हम index को Key नहीं रखते हैं।



Page No.: 62
Date: / /

Note:-

- A good rule of thumb is that elements inside the map() call need keys.
- Key should be specified inside the array.

Ex- state = {
  user: [
    {id: 101, name: "Rahul"},
    {id: 102, name: "Sonam"},
    {id: 103, name: "Rohit"},
  ],
};

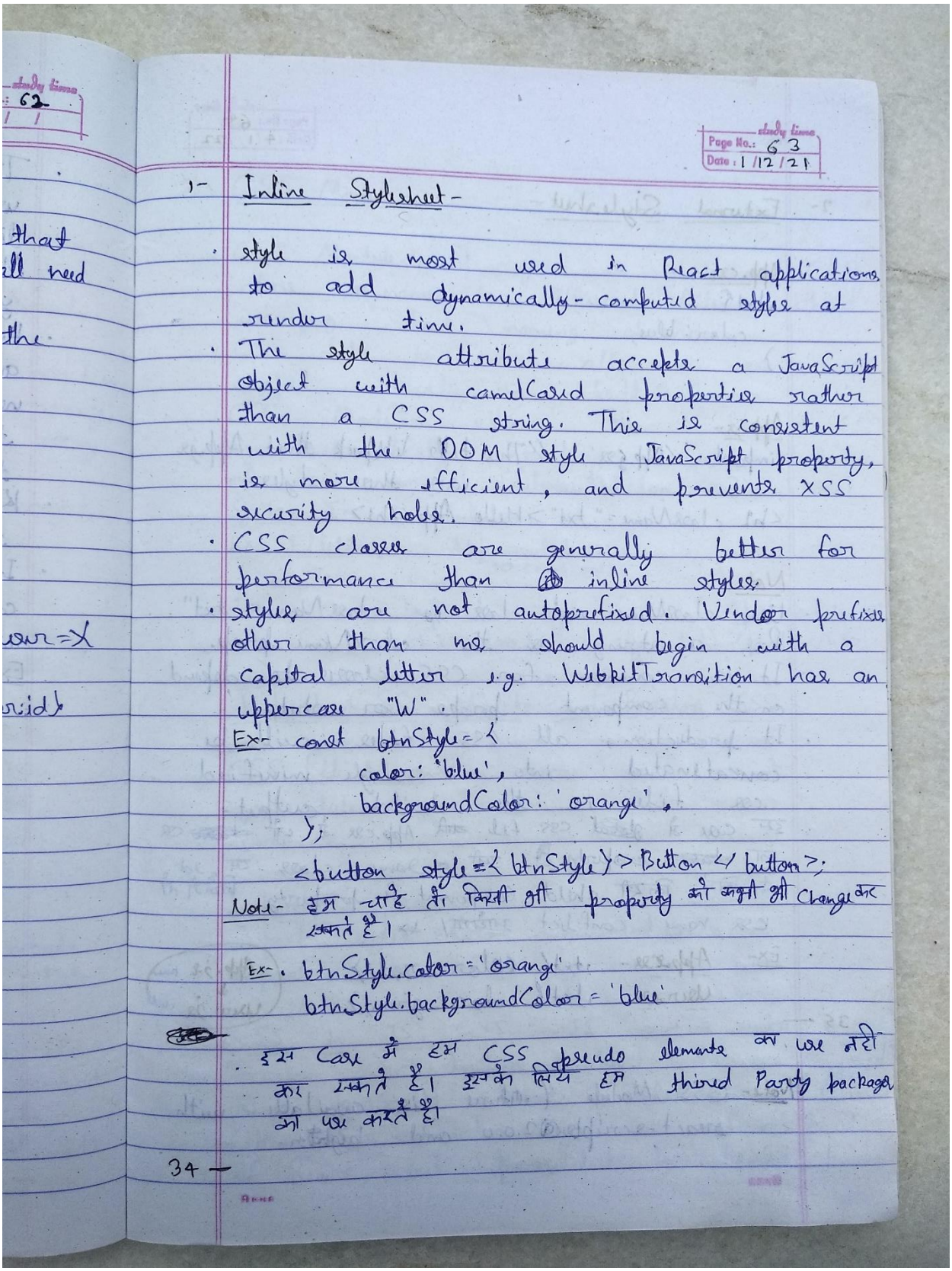
const newUsers = this.state.users.map(user => {
  return (
    <h1 key={user.id}>ID: {user.id}
      Name: {user.name}
    </h1>
  );
});

33 —

Styling Component/React with CSS-

1- Inline Stylesheet
2- External Stylesheet
3- CSS Module
4- CSS in JS

34



1- Inline Stylesheet -

- style is most used in React applications to add dynamically-computed styles at render time.
- The style attribute accepts a JavaScript object with camelCased properties rather than a CSS string. This is consistent with the DOM style JavaScript property, is more efficient, and prevents XSS security holes.
- CSS classes are generally better for performance than inline styles.
- styles are not autoprefixed. Vendor prefixes other than ms should begin with a capital letter e.g. WebkitTransition has an uppercase "W".

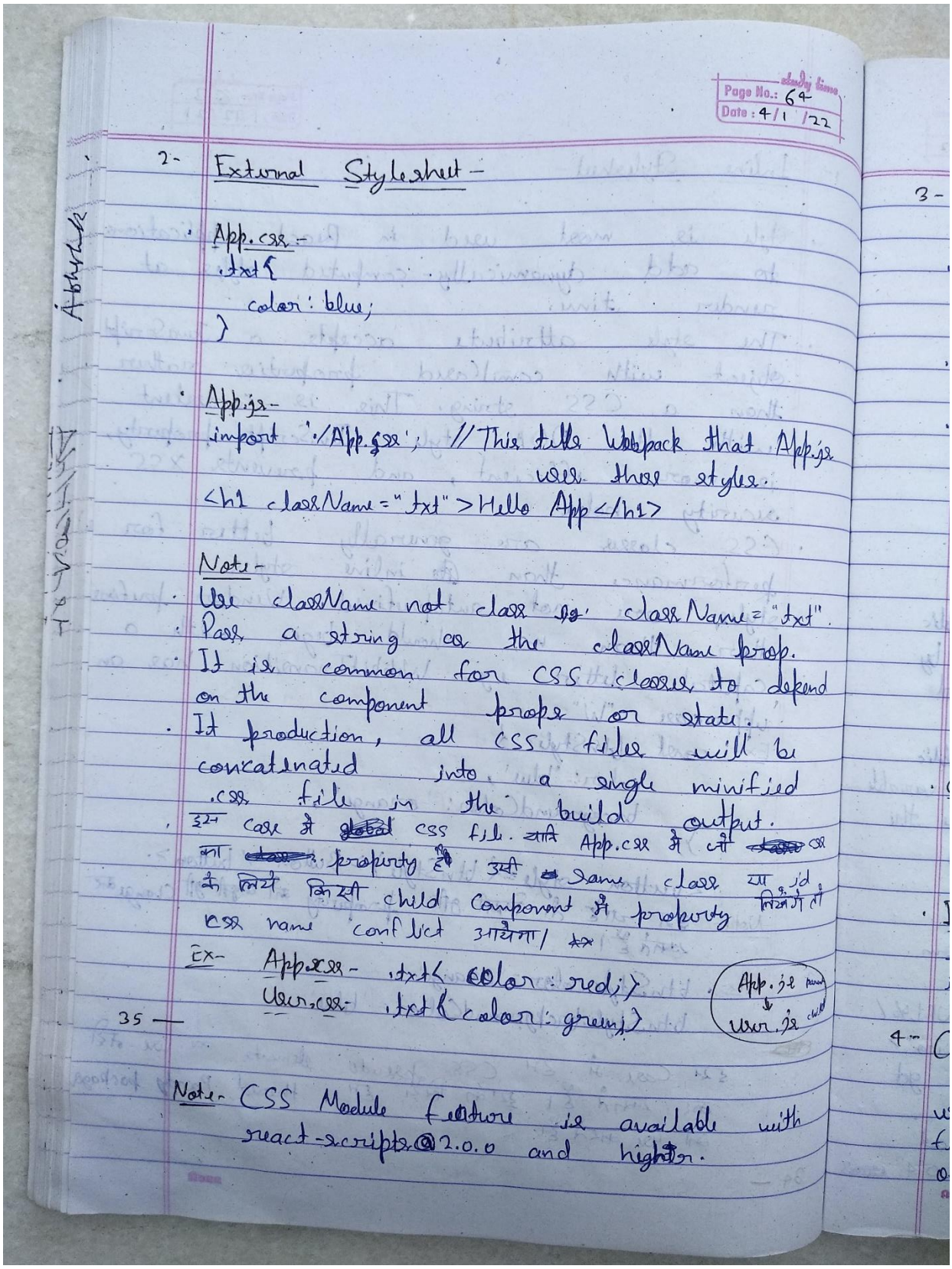
Ex- const btnStyle = {
color: 'blue',
backgroundColor: 'orange',
};

<button style={btnStyle}>Button</button>;

Note- हम चाहे तो किसी भी property को कभी भी change कर सकते है।

Ex- . btnStyle.color = 'orange'
btnStyle.backgroundColor = 'blue'

- इस case में हम CSS pseudo elements का use नहीं कर सकते है। इसके लिये हम third Party package का use करते है।



2- External Stylesheet -

- App.css -

```
.txt{
  color: blue;
}
```

- App.js -

```
import './App.css'; // This tells Webpack that App.js uses these styles.
<h1 className="txt">Hello App</h1>
```

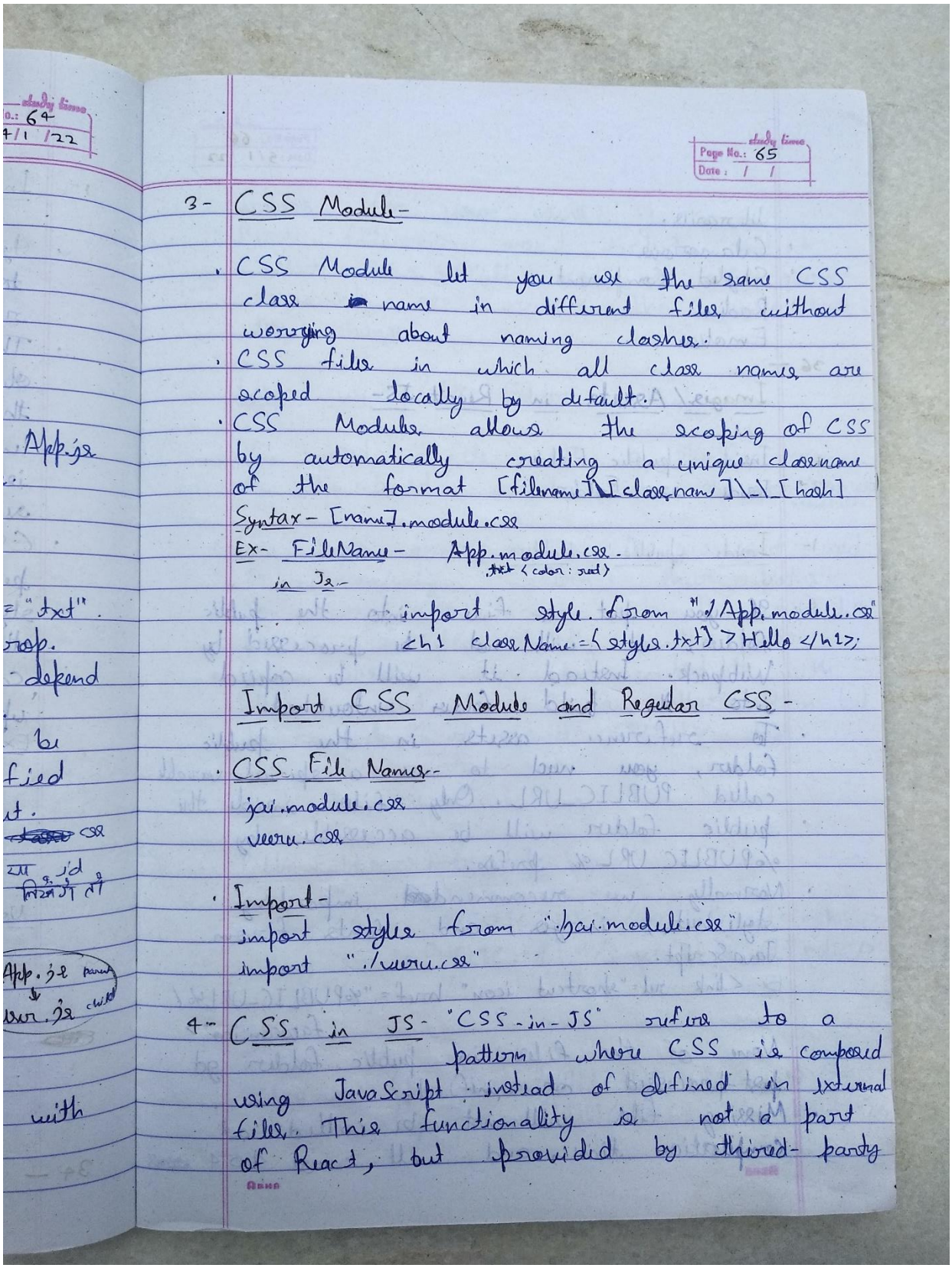
Note -

- Use className not class. ex. className="txt".
- Pass a string as the className prop.
- It is common for CSS classes to depend on the component props or state.
- In production, all CSS files will be concatenated into a single minified .css file in the build output.
- इस case में global css file यानि App.css में जो class का property है उसी same class या id के लिये किसी child Component में property लिखेंगे तो css name conflict आयेगा।

Ex- App.css - .txt{ color: red; }
User.css - .txt{ color: green; }

App.js (parent) → User.js (child)

Note - CSS Module feature is available with react-scripts@2.0.0 and higher.



3- CSS Module-

- CSS Module let you use the same CSS class name in different files without worrying about naming clashes.
- CSS file in which all class names are scoped locally by default.
- CSS Modules allows the scoping of CSS by automatically creating a unique classname of the format [filename]_[classname]_[hash]

Syntax - [name].module.css

Ex- FileName - App.module.css.

in Js -

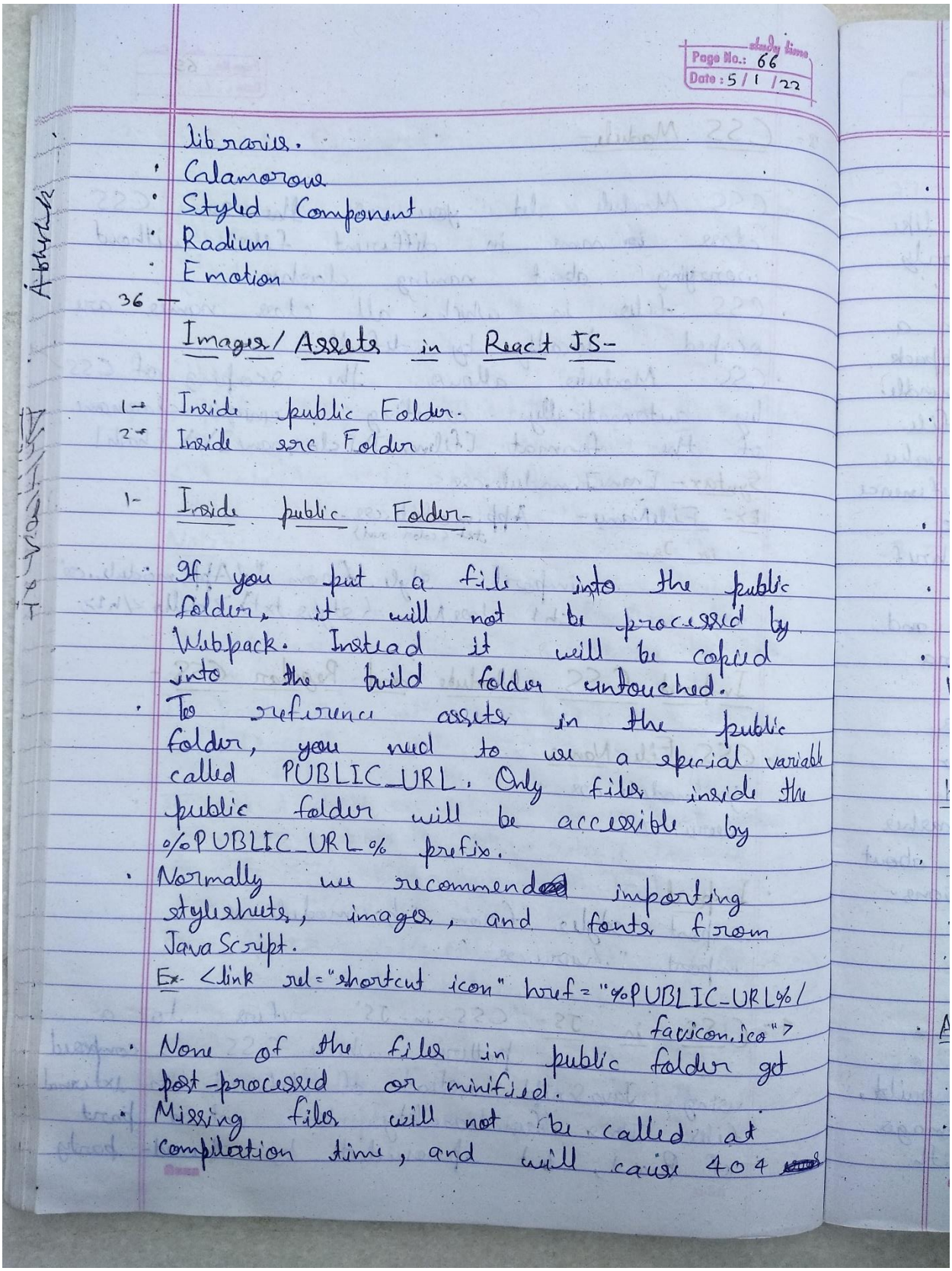
```
import style from "./App.module.css"
<h1 className={style.txt}>Hello</h1>;
```

Import CSS Module and Regular CSS -

- CSS File Names-
  jai.module.css
  veeru.css

- Import-
  import styles from "./jai.module.css"
  import "./veeru.css"

4- CSS in JS - "CSS-in-JS" refers to a pattern where CSS is composed using JavaScript instead of defined in external files. This functionality is not a part of React, but provided by third-party



libraries.
- Calamorow
- Styled Component
- Radium
- Emotion

36 —

Images / Assets in React JS-

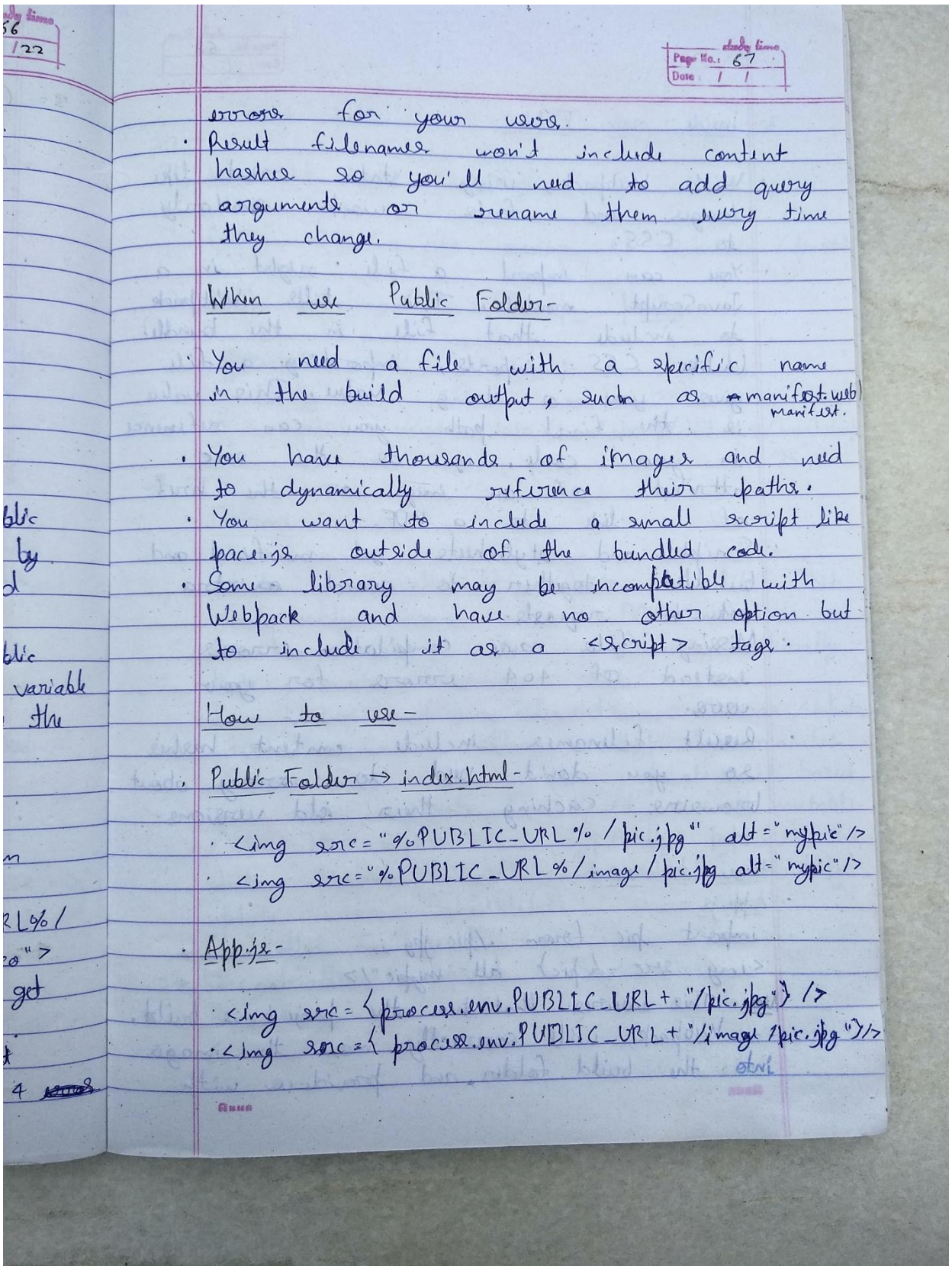
1- Inside public Folder.
2- Inside src Folder

1- Inside public Folder-

- If you put a file into the public folder, it will not be processed by Webpack. Instead it will be copied into the build folder untouched.
- To reference assets in the public folder, you need to use a special variable called PUBLIC_URL. Only files inside the public folder will be accessible by %PUBLIC_URL% prefix.
- Normally we recommended importing stylesheets, images, and fonts from JavaScript.

Ex. <link rel="shortcut icon" href="%PUBLIC_URL%/favicon.ico">

- None of the files in public folder get post-processed or minified.
- Missing files will not be called at compilation time, and will cause 404



errors for your users.
- Result filenames won't include content hashes so you'll need to add query arguments or rename them every time they change.

When use Public Folder-

- You need a file with a specific name in the build output, such as ~~manifest.webmanifest~~ manifest.
- You have thousands of images and need to dynamically reference their paths.
- You want to include a small script like pace.js outside of the bundled code.
- Some library may be incompatible with Webpack and have no other option but to include it as a <script> tags.

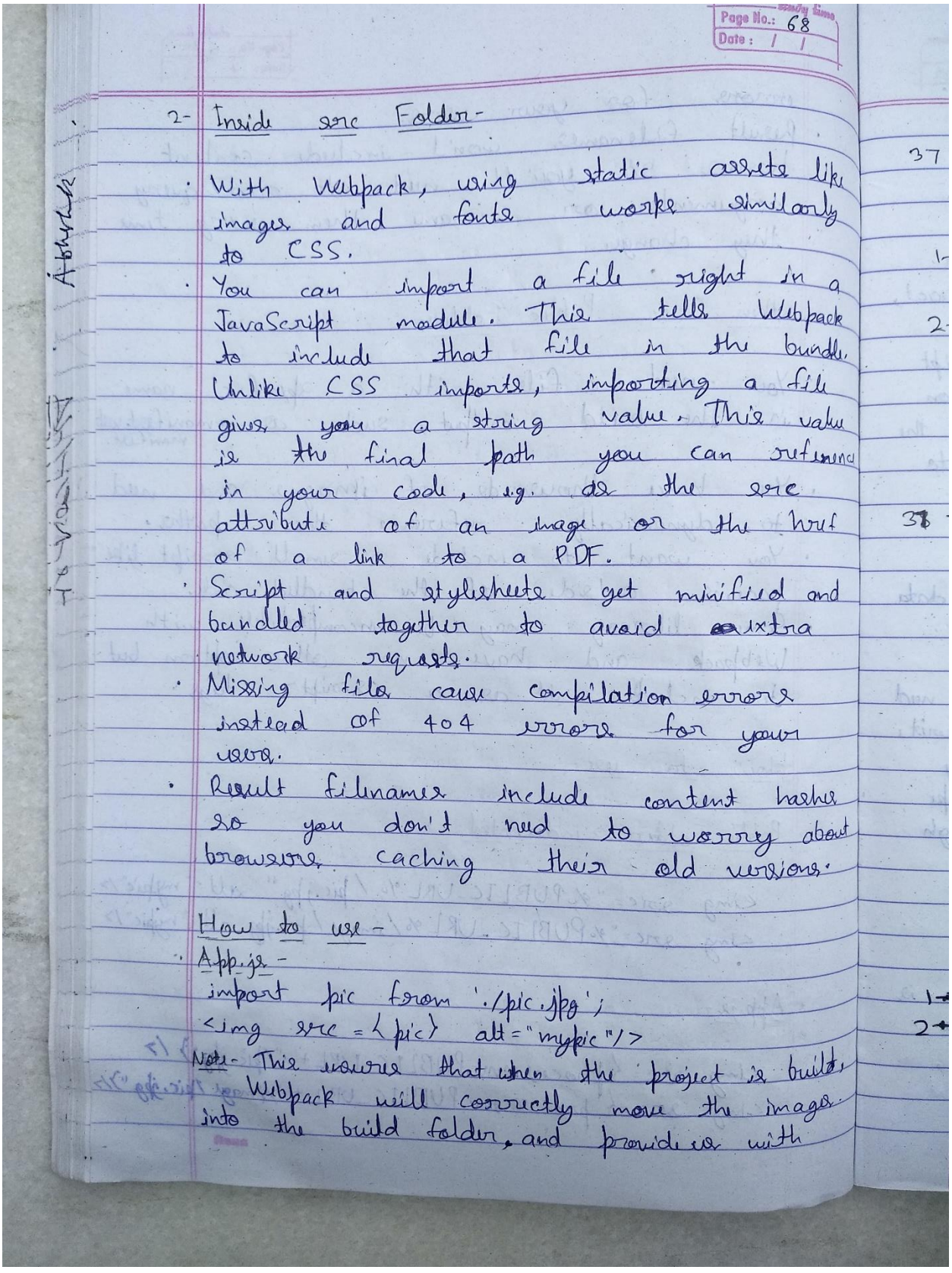
How to use-

1. Public Folder → index.html-

-
-

- App.js-

-
-



Page No.: 68
Date: / /

2- Inside src Folder-

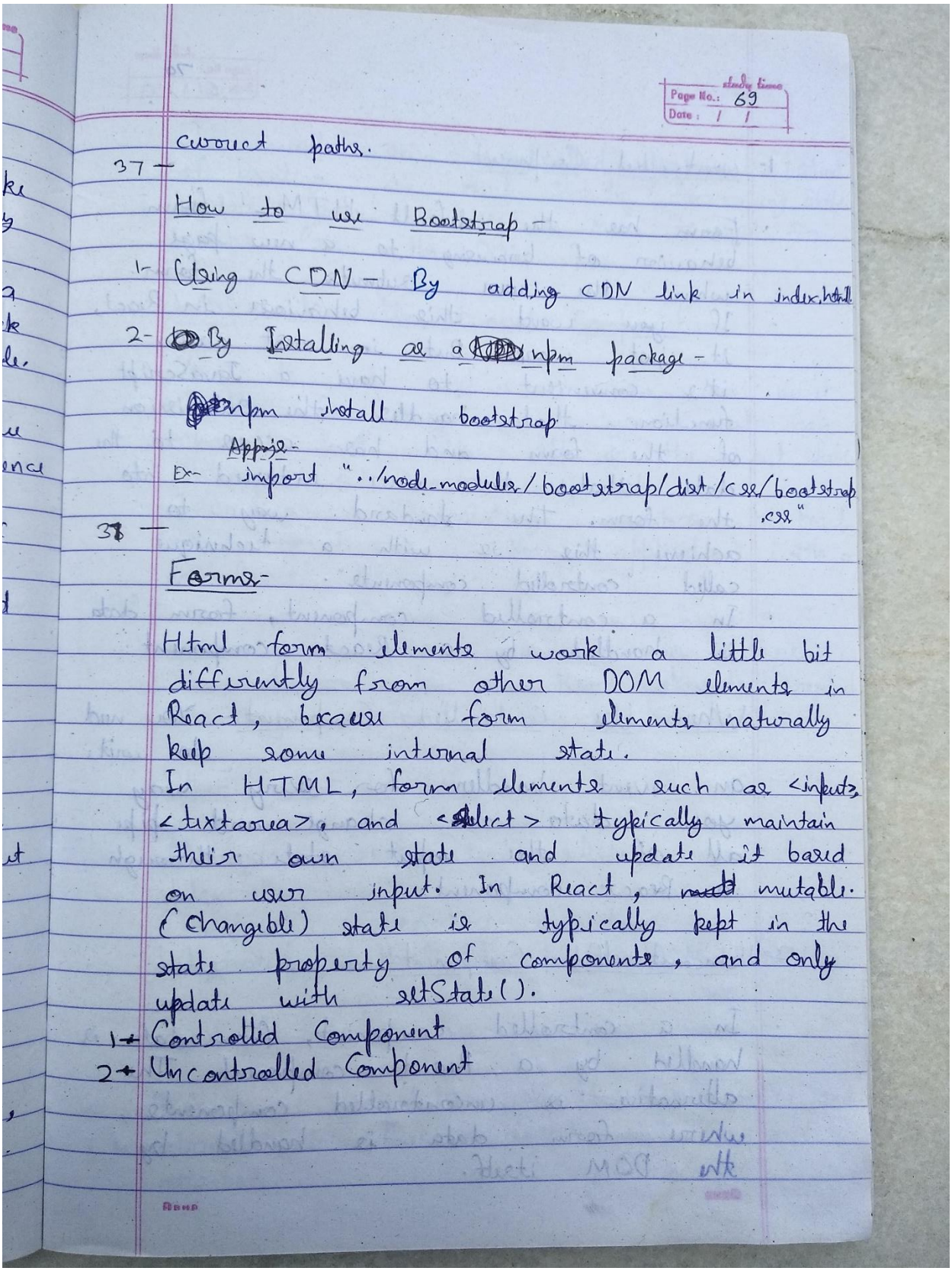
- With Webpack, using static assets like images and fonts works similarly to CSS.
- You can import a file right in a JavaScript module. This tells Webpack to include that file in the bundle. Unlike CSS imports, importing a file gives you a string value. This value is the final path you can reference in your code, e.g. as the src attribute of an image or the href of a link to a PDF.
- Script and stylesheets get minified and bundled together to avoid extra network requests.
- Missing files cause compilation errors instead of 404 errors for your users.
- Result filenames include content hashes so you don't need to worry about browsers caching their old versions.

How to use -

- App.js -

```
import pic from './pic.jpg';
<img src={pic} alt="mypic" />
```

Note- This means that when the project is built, Webpack will correctly move the images into the build folder, and provide us with



Page No.: 69
Date: / /

correct paths.

37 —

How to use Bootstrap -

1- Using CDN - By adding CDN link in index.html

2- By Installing as a npm package -

npm install bootstrap

Apply -

Ex- import "../node_modules/bootstrap/dist/css/bootstrap.css"

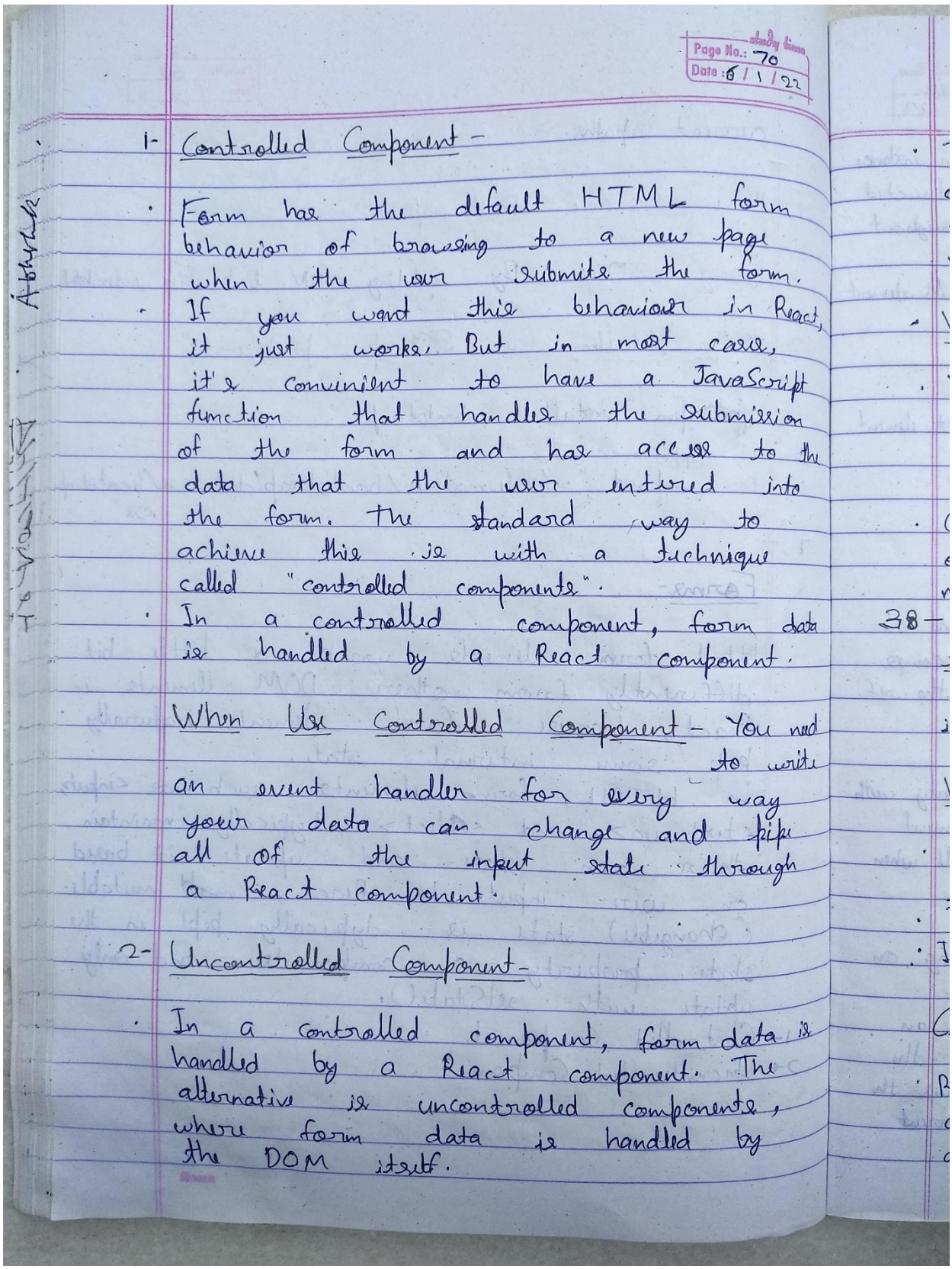
38 —

Forms -

Html form elements work a little bit differently from other DOM elements in React, because form elements naturally keep some internal state.

In HTML, form elements such as <input>, <textarea>, and <select> typically maintain their own state and update it based on user input. In React, mutable. (changible) state is typically kept in the state property of components, and only update with setState().

1- Controlled Component
2- Uncontrolled Component



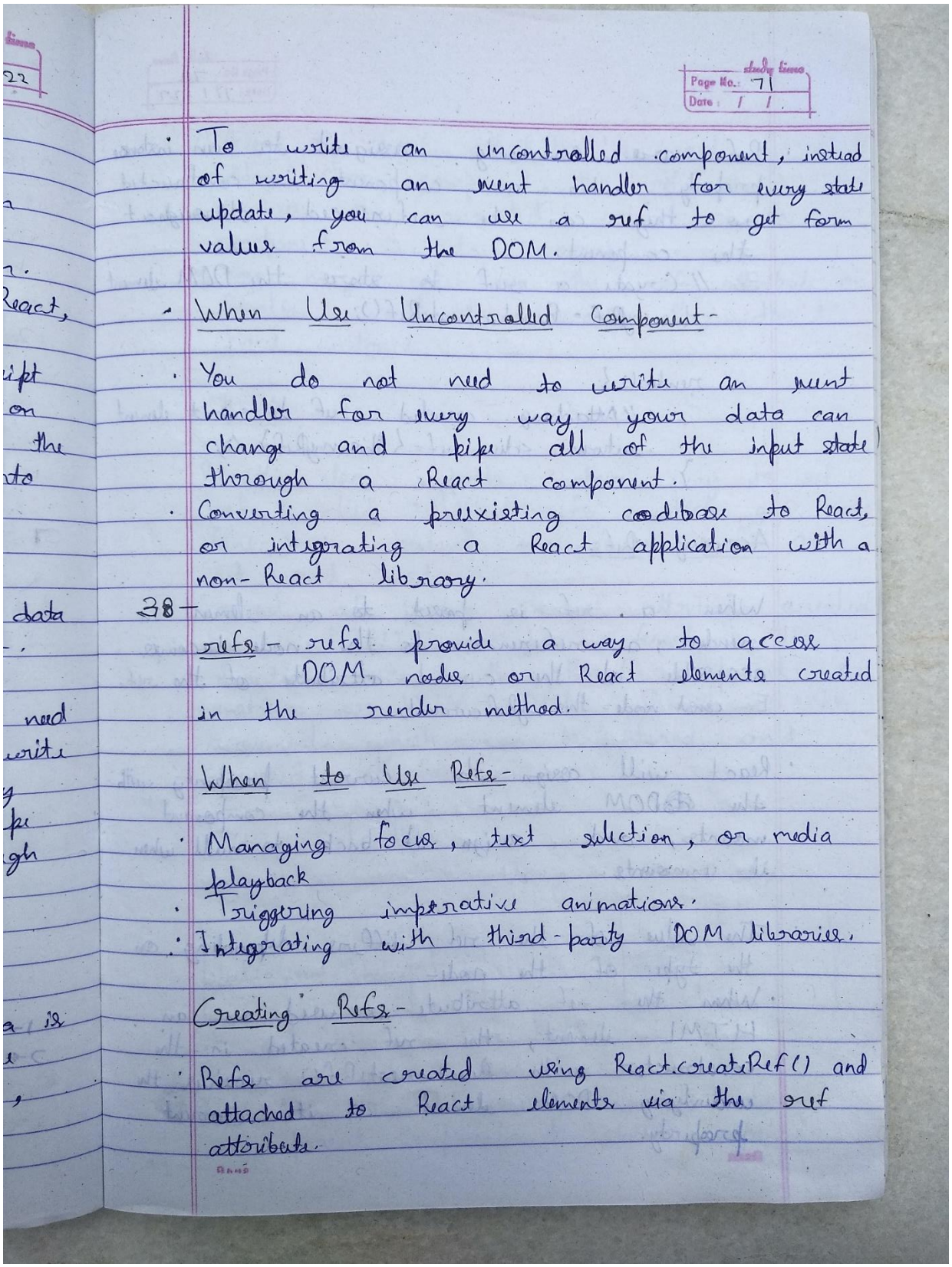
1- Controlled Component -

- Form has the default HTML form behavior of browsing to a new page when the user submits the form.
- If you want this behavior in React, it just works. But in most case, it's convinient to have a JavaScript function that handles the submission of the form and has access to the data that the user entered into the form. The standard way to achieve this is with a technique called "controlled components".
- In a controlled component, form data is handled by a React component.

When Use Controlled Component - You need to write an event handler for every way your data can change and pipe all of the input state through a React component.

2- Uncontrolled Component -

- In a controlled component, form data is handled by a React component. The alternative is uncontrolled components, where form data is handled by the DOM itself.



- To write an uncontrolled component, instead of writing an event handler for every state update, you can use a ref to get form values from the DOM.

When Use Uncontrolled Component-

- You do not need to write an event handler for every way your data can change and pipe all of the input state through a React component.
- Converting a prexisting codebase to React, or integrating a React application with a non-React library.

38-

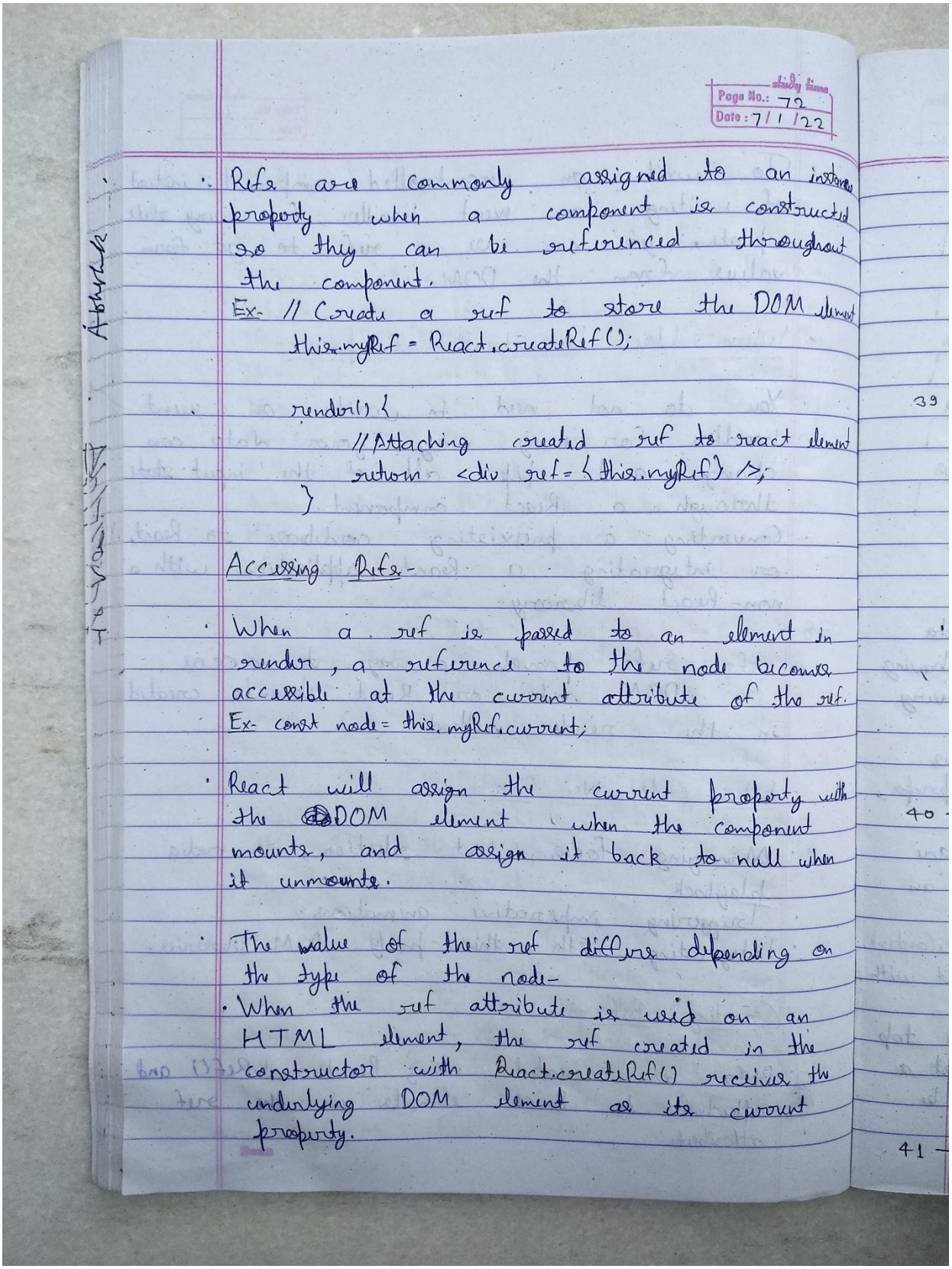
refs - refs provide a way to access DOM nodes or React elements created in the render method.

When to Use Refs-

- Managing focus, text selection, or media playback
- Triggering imperative animations.
- Integrating with third-party DOM libraries.

Creating Refs-

- Refs are created using React.createRef() and attached to React elements via the ref attributes.



Page No.: 72
Date: 7/1/22

- Refs are commonly assigned to an instance property when a component is constructed so they can be referenced throughout the component.

Ex- // Create a ref to store the DOM element
this.myRef = React.createRef();

render() {
// Attaching created ref to react element
return ;
}

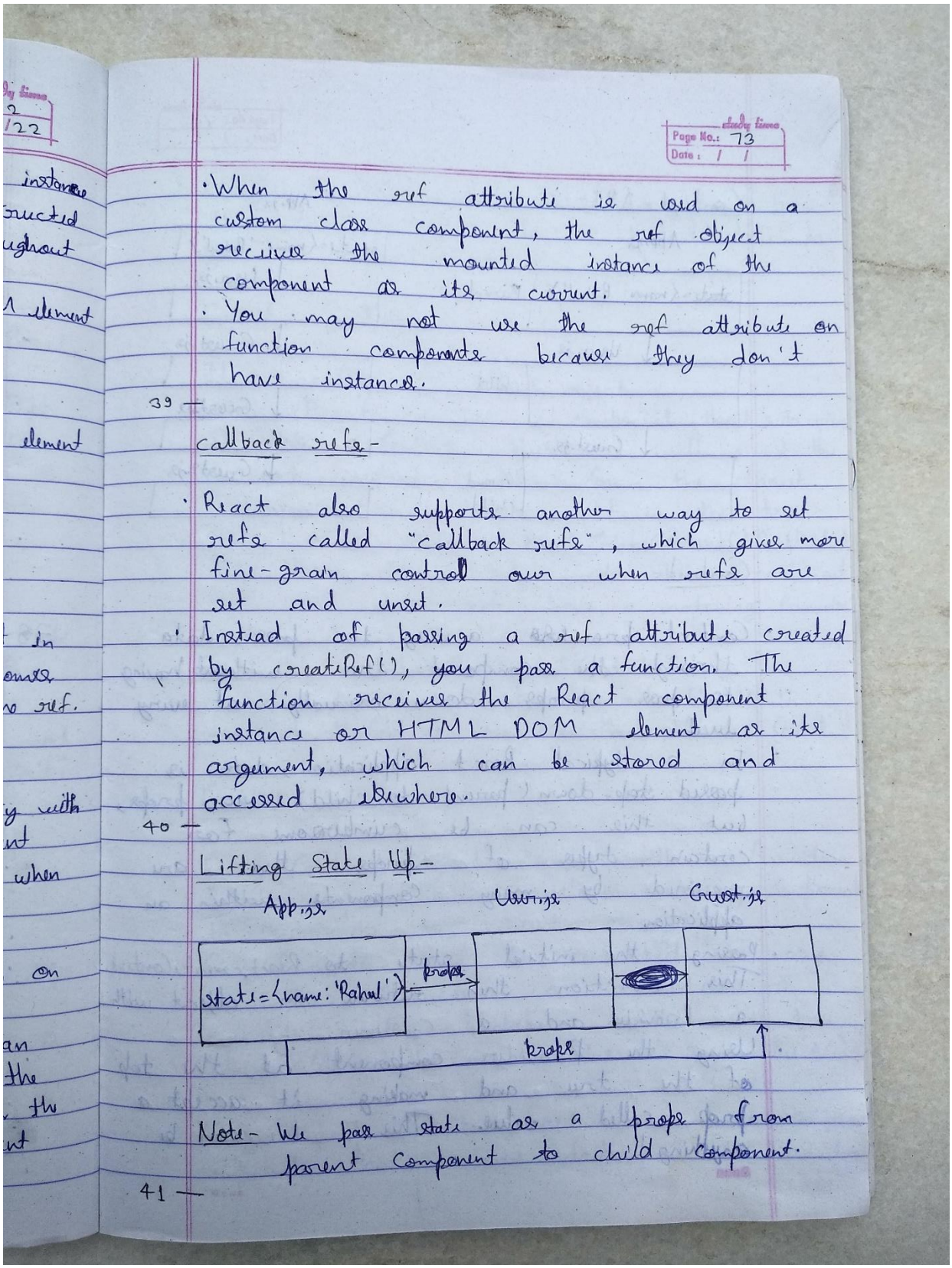
Accessing Refs -

- When a ref is passed to an element in render, a reference to the node becomes accessible at the current attribute of the ref.

Ex- const node = this.myRef.current;

- React will assign the current property with the DOM element when the component mounts, and assign it back to null when it unmounts.

- The value of the ref differs depending on the type of the node-
- When the ref attribute is used on an HTML element, the ref created in the constructor with React.createRef() receives the underlying DOM element as its current property.



- When the ref attribute is used on a custom class component, the ref object receives the mounted instance of the component as its current.
- You may not use the ref attribute on function components because they don't have instances.

39 —

Callback refs -

- React also supports another way to set refs called "callback refs", which gives more fine-grain control over when refs are set and unset.
- Instead of passing a ref attribute created by createRef(), you pass a function. The function receives the React component instance or HTML DOM element as its argument, which can be stored and accessed elsewhere.

40 —

Lifting State Up -

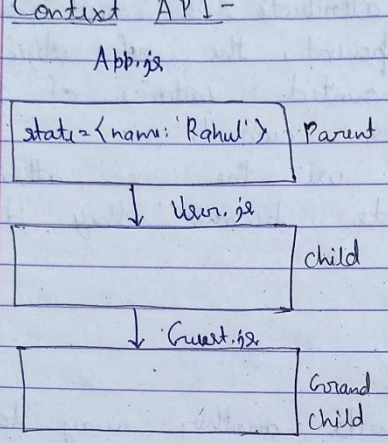
App.js User.js Guest.js

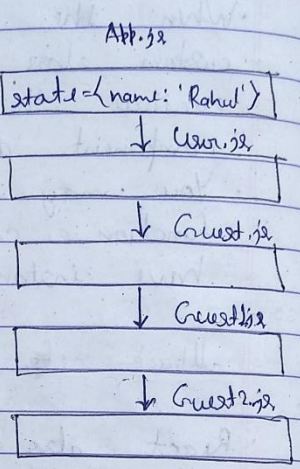
Note - We pass state as a prop from parent component to child component.

41 —

Page No.: 74
Date: / /

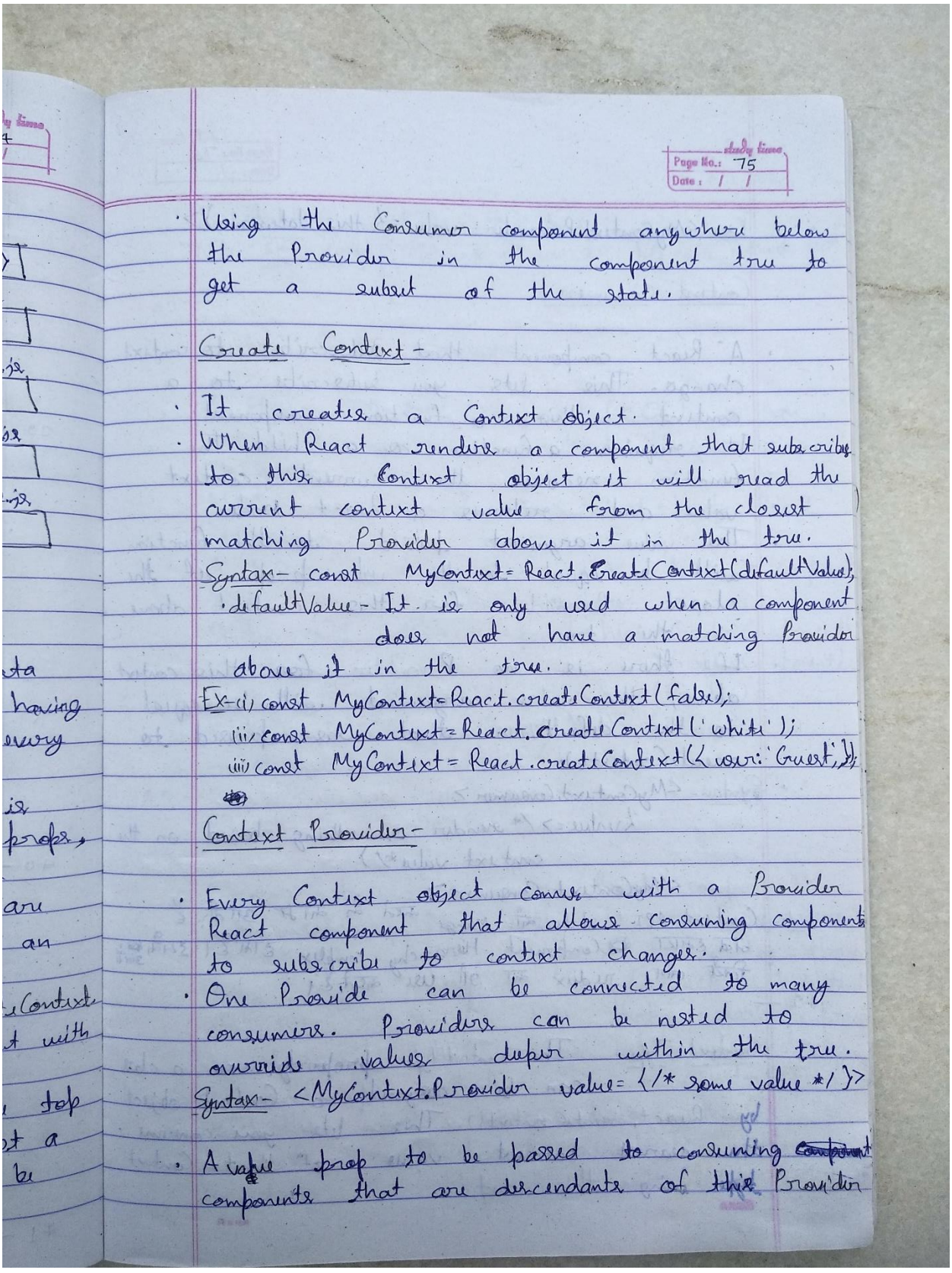
Context API -





Context -

- Context provides a way to pass data through the component tree without having to pass props down manually at every level.
- In a typical React application, data is passed top-down (parent to child) via props, but this can be cumbersome for certain types of props that are required by many components within an application.
- Passing the initial state to React.createContext. This function then returns an object with a Provider and a Consumer.
- Using the Provider component at the top of the tree and making it accept a prop called value. This value can be anything!



- Using the Consumer component anywhere below the Provider in the component tree to get a subset of the state.

Create Context -

- It creates a Context object.
- When React renders a component that subscribes to this Context object it will read the current context value from the closest matching Provider above it in the tree.

Syntax - const MyContext = React.CreateContext(defaultValue);

- defaultValue - It is only used when a component does not have a matching Provider above it in the tree.

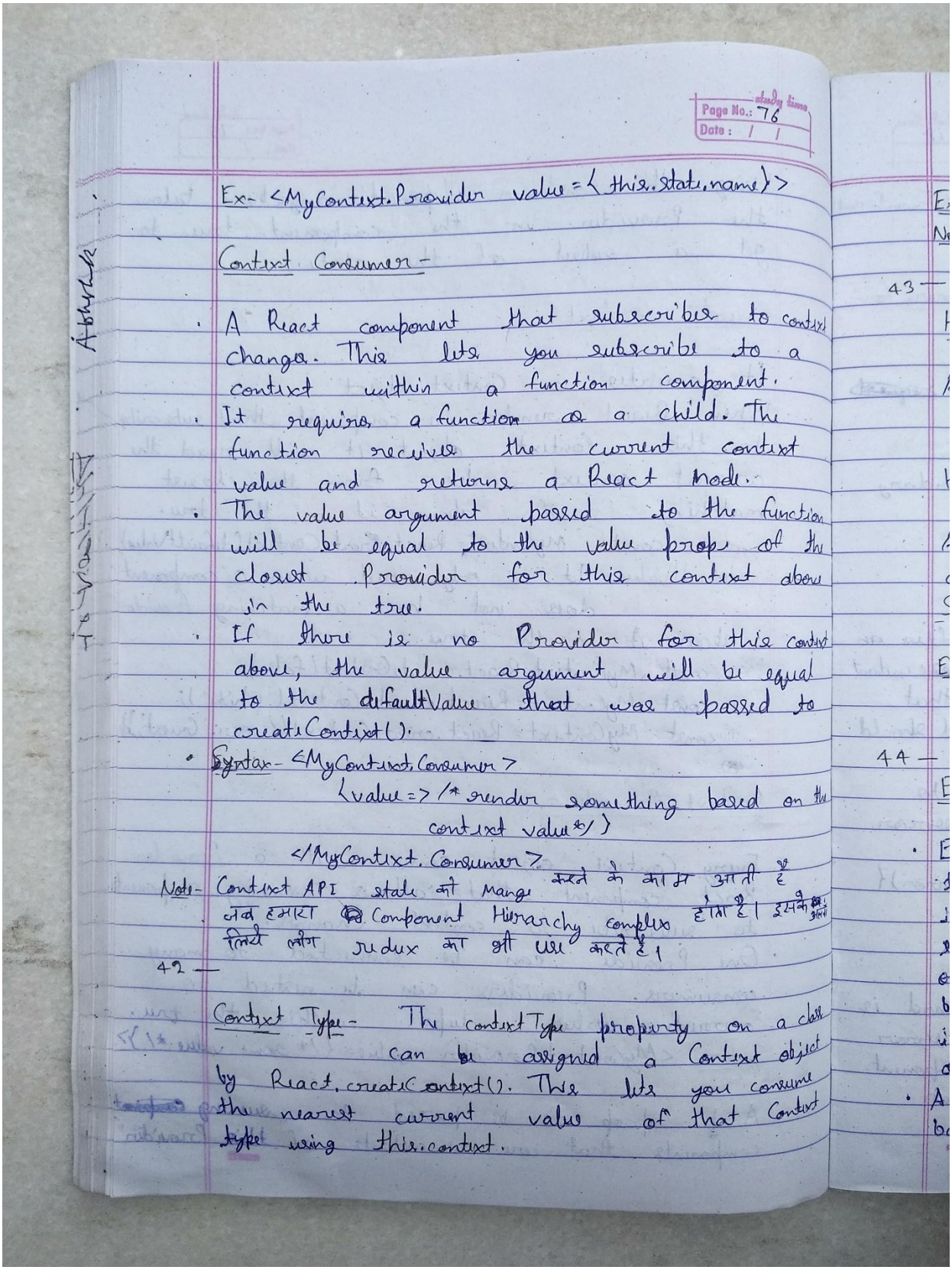
Ex - (i) const MyContext = React.createContext(false);
(ii) const MyContext = React.createContext('white');
(iii) const MyContext = React.createContext({user: 'Guest'});

Context Provider -

- Every Context object comes with a Provider React component that allows consuming components to subscribe to context changes.
- One Provide can be connected to many consumers. Providers can be nested to override values deeper within the tree.

Syntax - <MyContext.Provider value={/* some value */}>

- A value prop to be passed to consuming components that are descendants of this Provider



Ex- <MyContext.Provider value={this.state.name}>

Context Consumer -

- A React component that subscribes to context changes. This lets you subscribe to a context within a function component.
- It requires a function as a child. The function receives the current context value and returns a React node.
- The value argument passed to the function will be equal to the value prop of the closest Provider for this context above in the tree.
- If there is no Provider for this context above, the value argument will be equal to the defaultValue that was passed to createContext().
- Syntax- <MyContext.Consumer>
  {value => /* render something based on the context value */}
  </MyContext.Consumer>

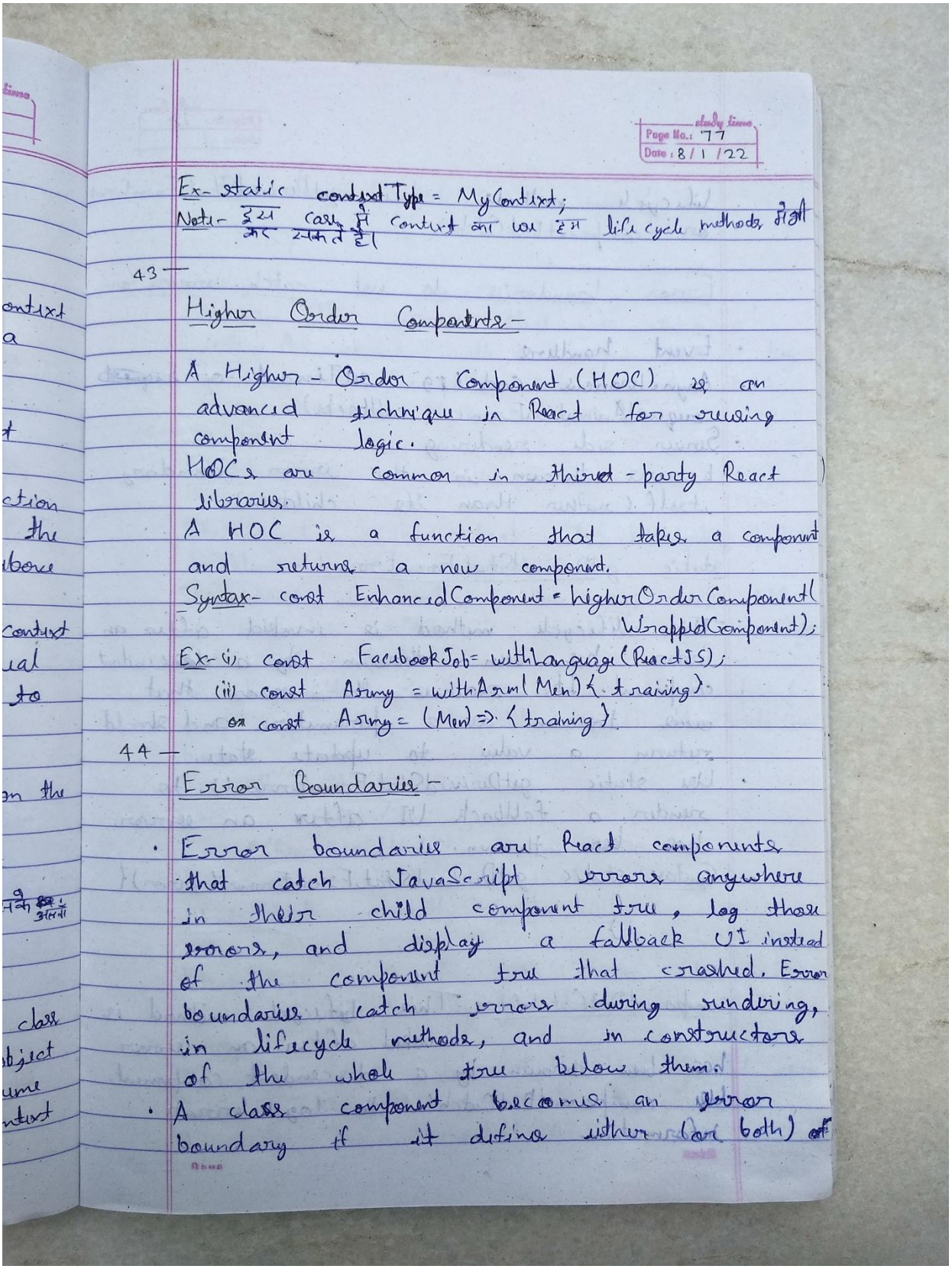
Note- Context API state को manage करने के काम आती है जब हमारा Component Hierarchy complex होता है। इसके अलावा लोग redux का भी use करते है।

42 —

Context Type - The contextType property on a class can be assigned a Context object by React.createContext(). This lets you consume the nearest current value of that Context type using this.context.

43 —

44 —



Page No.: 77
Date: 8/1/22

Ex- static contextType = MyContext;
Note- इस तरह class component context का use हम life cycle methods में भी कर सकते हैं।

43 —

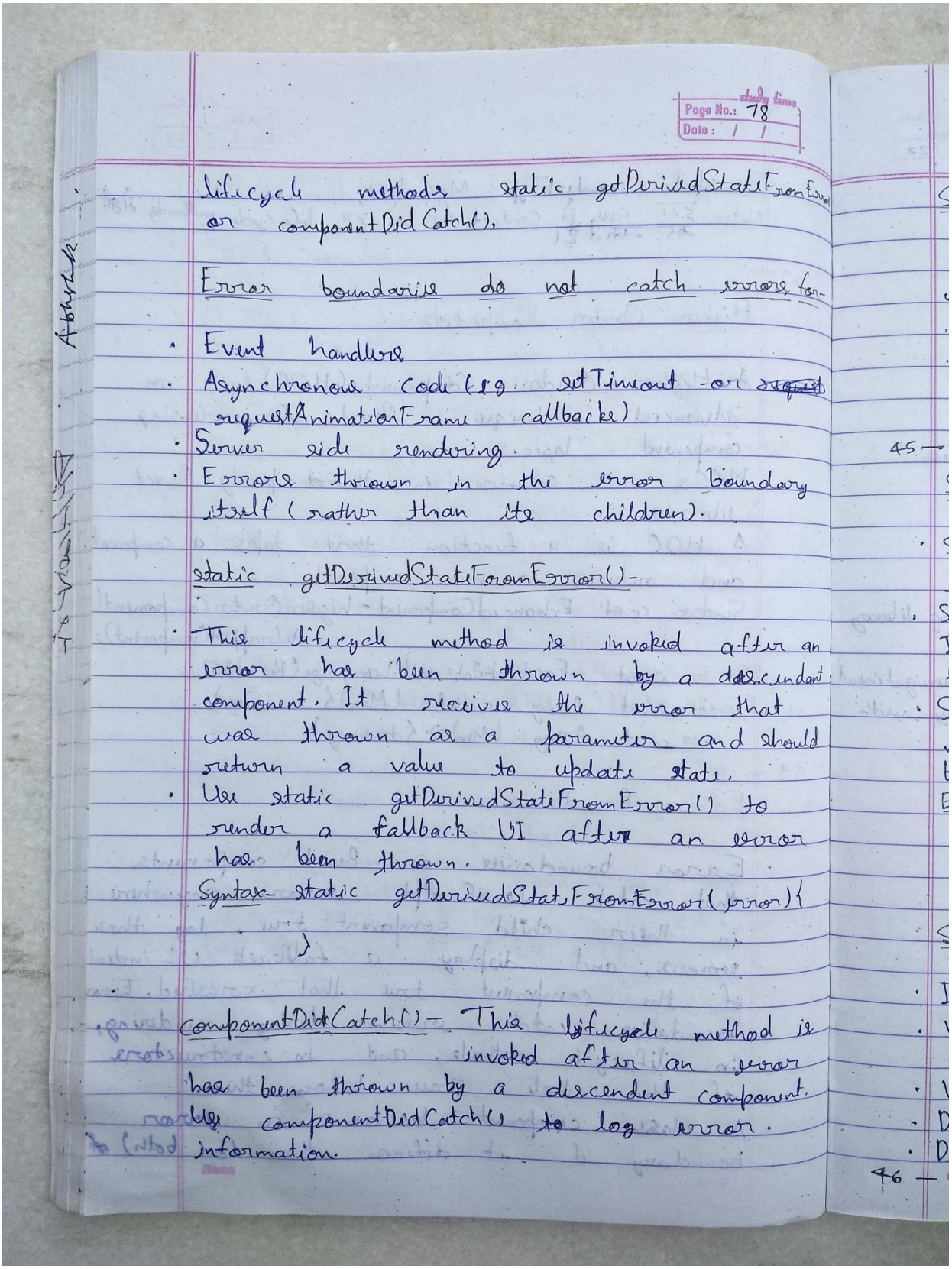
Higher Order Components –

A Higher-Order Component (HOC) is an advanced technique in React for reusing component logic.
HOCs are common in third-party React libraries.
A HOC is a function that takes a component and returns a new component.
Syntax- const EnhancedComponent = higherOrderComponent(WrappedComponent);
Ex- (i) const FacebookJob = withLanguage(ReactJS);
(ii) const Army = withArm(Men){training}
or const Army = (Men) => {training}

44 —

Error Boundaries –

- Error boundaries are React components that catch JavaScript errors anywhere in their child component tree, log those errors, and display a fallback UI instead of the component tree that crashed. Error boundaries catch errors during rendering, in lifecycle methods, and in constructors of the whole tree below them.
- A class component becomes an error boundary if it defines either (or both) of



lifecycle methods static getDerivedStateFromError or componentDidCatch().

Error boundaries do not catch errors for-

- Event handlers
- Asynchronous code (eg. setTimeout or requestAnimationFrame callbacks)
- Server side rendering.
- Errors thrown in the error boundary itself (rather than its children).

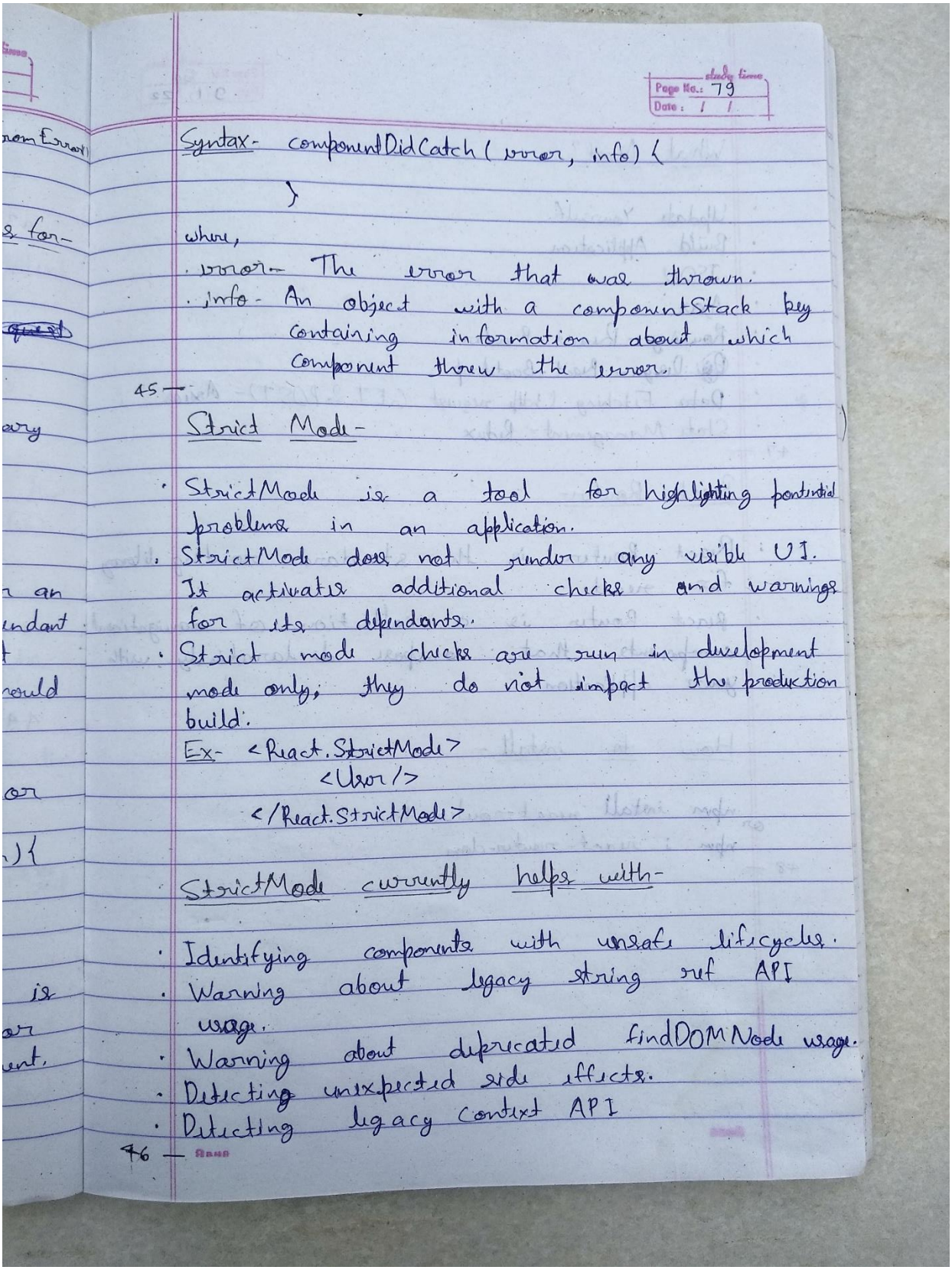
static getDerivedStateFromError()-

- This lifecycle method is invoked after an error has been thrown by a descendant component. It receives the error that was thrown as a parameter and should return a value to update state.
- Use static getDerivedStateFromError() to render a fallback UI after an error has been thrown.

Syntax- static getDerivedStateFromError(error){

}

componentDidCatch()- This lifecycle method is invoked after an error has been thrown by a descendent component. Use componentDidCatch() to log error information.



Syntax- componentDidCatch(error, info){

}

where,

- error- The error that was thrown.
- info- An object with a componentStack key containing information about which component threw the error.

45 —

Strict Mode-

- StrictMode is a tool for highlighting potential problems in an application.
- StrictMode does not render any visible UI. It activates additional checks and warnings for its dependants.
- Strict mode checks are run in development mode only; they do not impact the production build.

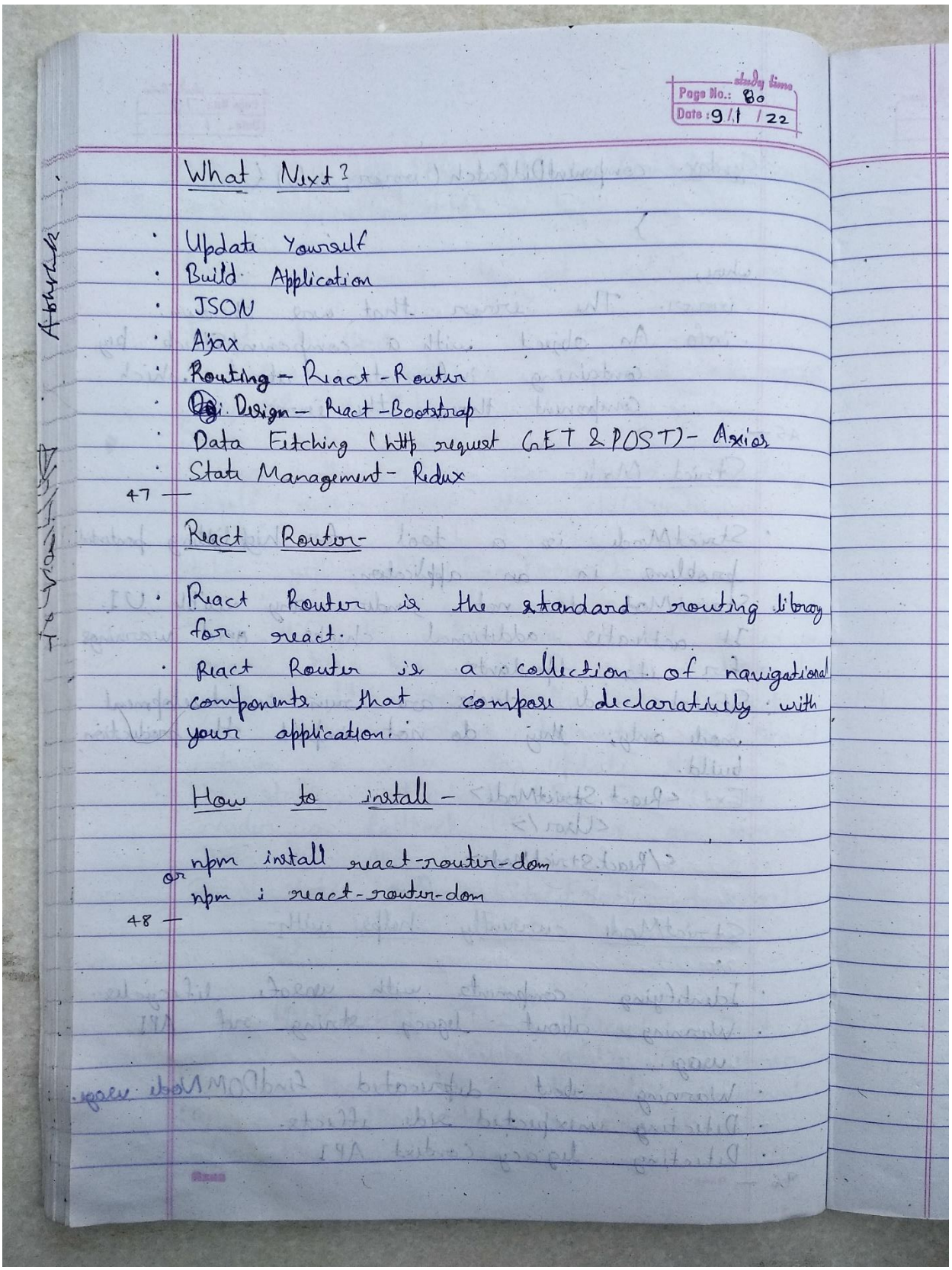
Ex- <React.StrictMode>
<User />
</React.StrictMode>

StrictMode currently helps with-

- Identifying components with unsafe lifecycles.
- Warning about legacy string ref API usage.
- Warning about deprecated findDOMNode usage.
- Detecting unexpected side effects.
- Detecting legacy context API

46 —

d



Page No.: 80
Date: 9/1/22

What Next?

- Update Yourself
- Build Application
- JSON
- Ajax
- Routing - React-Router
- UI Design - React-Bootstrap
- Data Fetching (http request GET & POST) - Axios
- State Management - Redux

47 —

React Router-

- React Router is the standard routing library for react.
- React Router is a collection of navigational components that compose declaratively with your application.

How to install -

npm install react-router-dom
or
npm i react-router-dom

48 —