Advance JavaScript

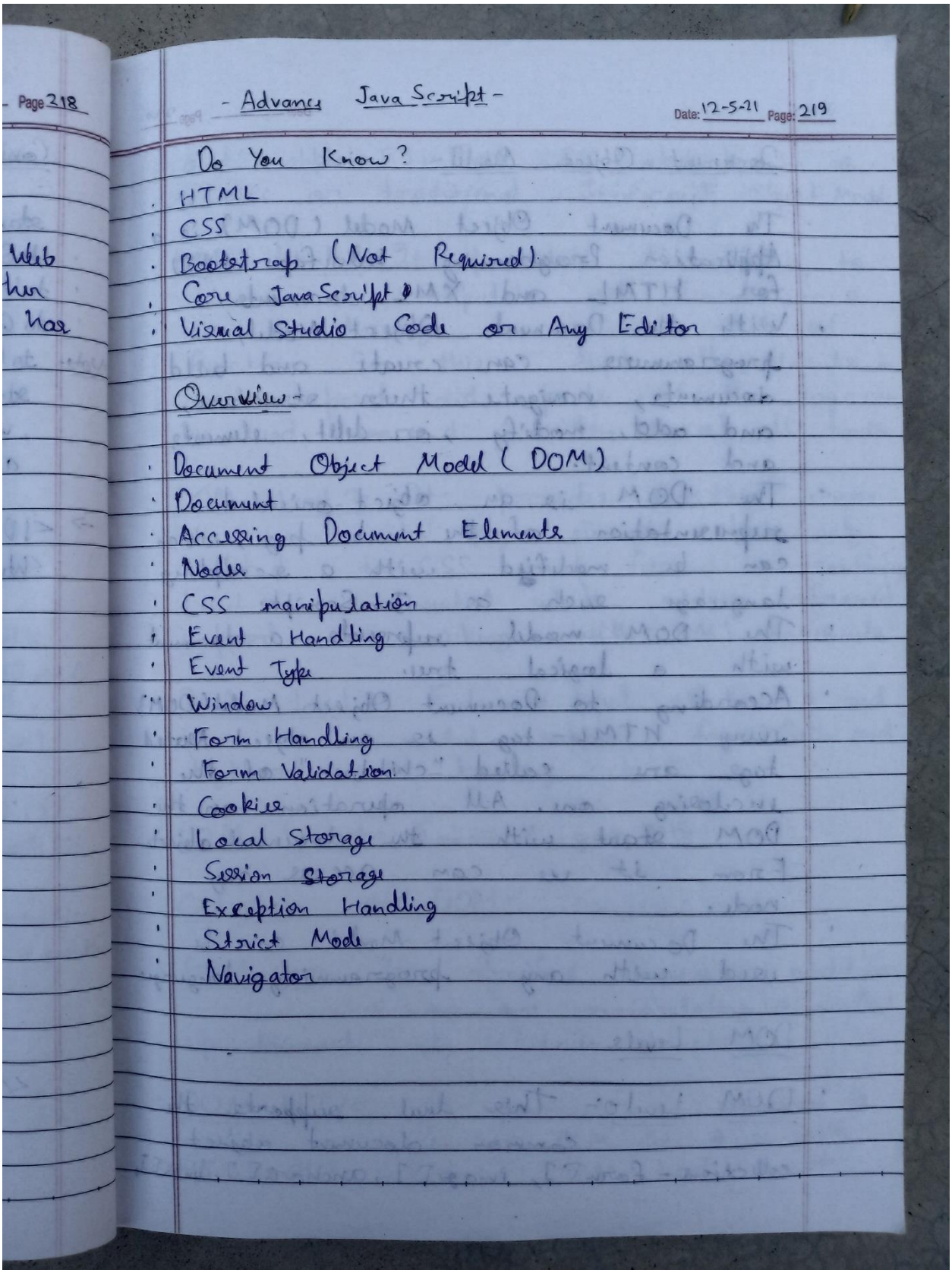
GEEKYSHOWS YOUTUBE CHANNEL LEARNING NOTES

Source Code- https://github.com/satyam-seth-learnings/javascript_learning/tree/master/Geekyshows/Advance%20Java%20Script

Playlist Link- https://youtube.com/playlist?list=PLbGui_ZYuhihZ-pDxNZuQ7xOQ8IS2z3XI

SATYAM SETH

26-09-2021



Page 218

- Advance Java Script -

Date: 12-5-21 Page: 219

Do You Know?

- HTML
- CSS
- Bootstrap (Not Required)
- Core JavaScript
- Visual Studio Code or Any Editor

Overview -

- Document Object Model (DOM)
- Document
- Accessing Document Elements
- Nodes
- CSS manipulation
- Event Handling
- Event Type
- Window
- Form Handling
- Form Validation
- Cookie
- Local Storage
- Session Storage
- Exception Handling
- Strict Mode
- Navigator

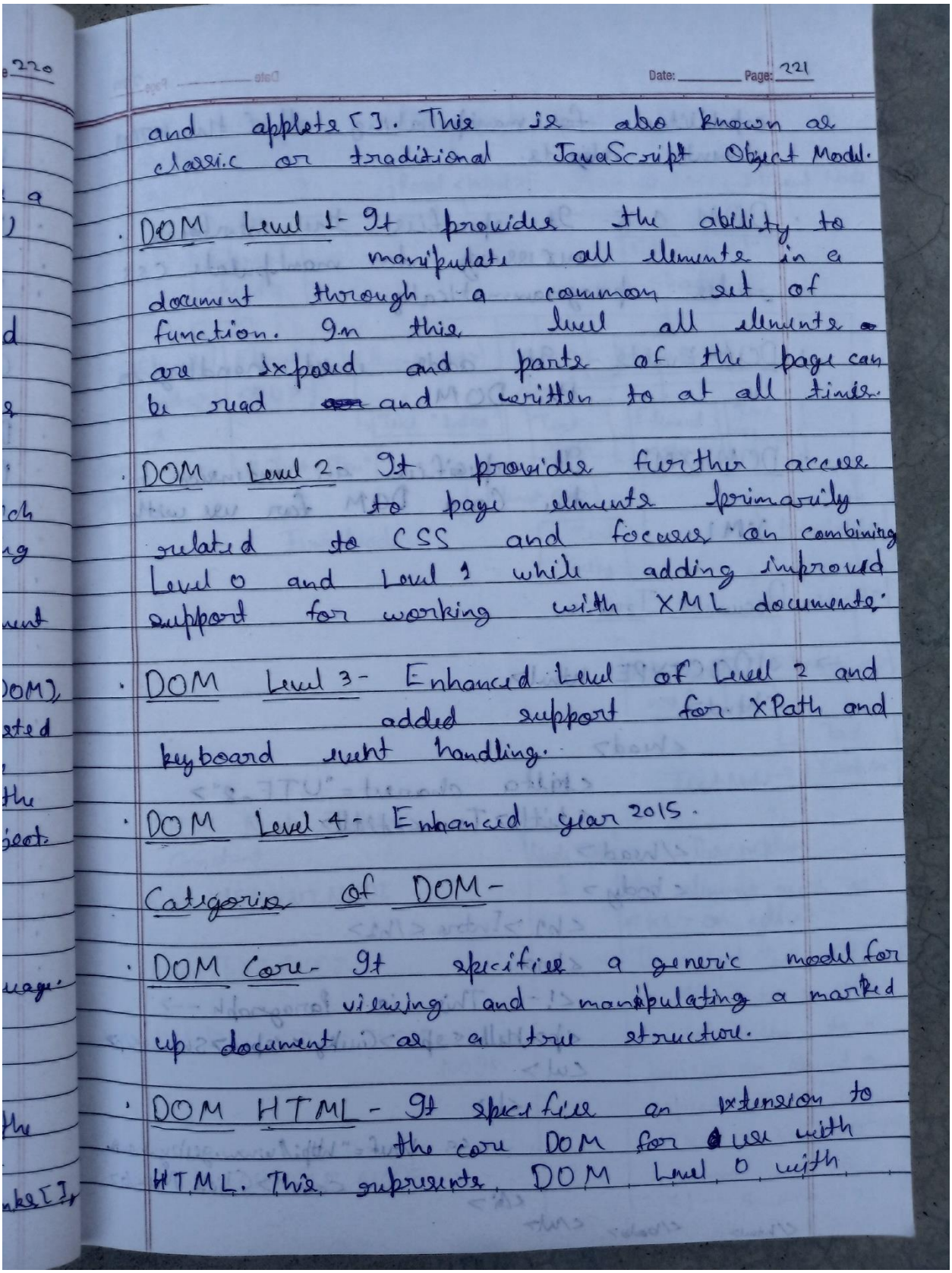
Date _____ Page 220

Document Object Model-

- The Document Object Model (DOM) is a Application Programming Interface (API) for HTML and XML documents.
- With the Document Object Model, programmers can create and build documents, navigate their structure, and add, modify, or delete elements and content.
- The DOM is an object-oriented representation of the web page, which can be modified with a scripting language such as JavaScript.
- The DOM model represents a document with a logical tree.
- According to Document Object Model (DOM), every HTML-tag is an object. Nested tags are called "children" of the enclosing one. All operations on the DOM start with the document object. From it we can access any node.
- The Document Object Model can be used with any programming language.

DOM Levels-

- DOM Level 0:- This level supports the common document object collections - form[], image[], anchors[], links[]

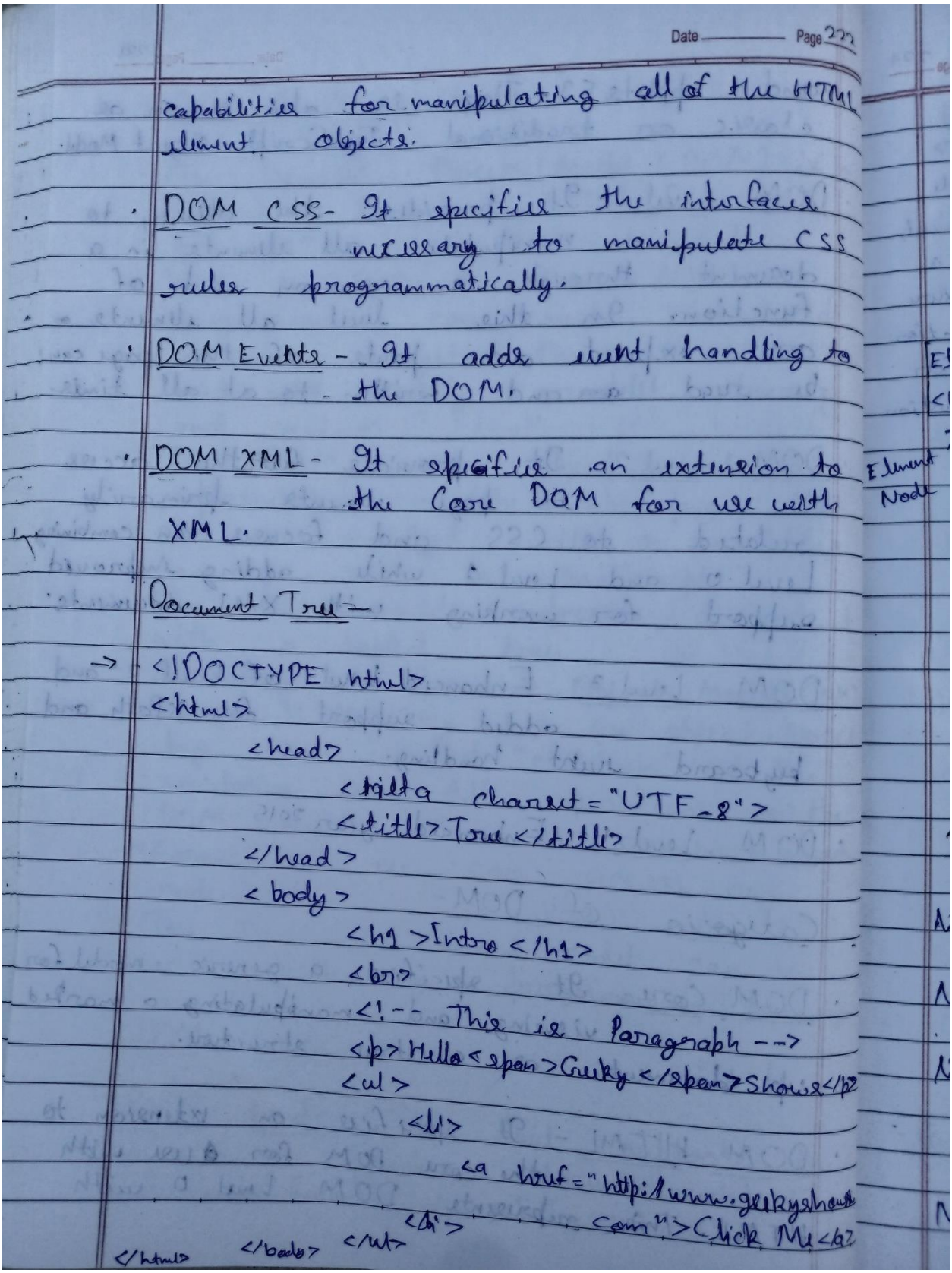


and applets[]. This is also known as classic or traditional JavaScript Object Model.

- DOM Level 1- It provides the ability to manipulate all elements in a document through a common set of function. In this level all elements are exposed and parts of the page can be read and written to at all times.
- DOM Level 2- It provides further access to page elements primarily related to CSS and focuses on combining Level 0 and Level 1 while adding improved support for working with XML documents.
- DOM Level 3- Enhanced Level of Level 2 and added support for XPath and keyboard event handling.
- DOM Level 4- Enhanced year 2015.

Categories of DOM-

- DOM Core- It specifies a generic model for viewing and manipulating a marked up document as a tree structure.
- DOM HTML- It specifies an extension to the core DOM for use with HTML. This represents DOM Level 0 with



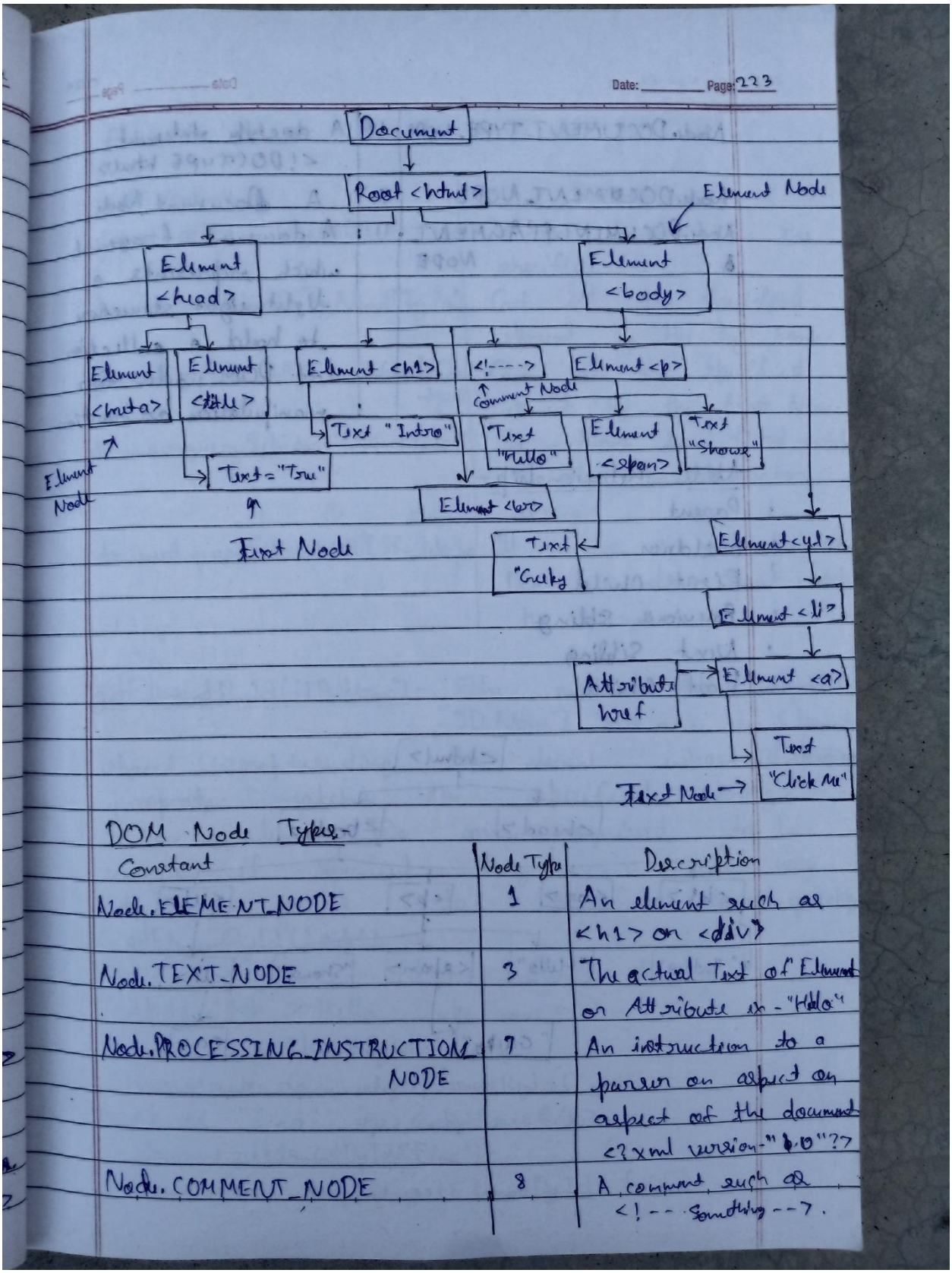
Date _____ Page 222

capabilities for manipulating all of the HTML element objects.

- DOM CSS - It specifies the interfaces necessary to manipulate CSS rules programmatically.
- DOM Events - It adds event handling to the DOM.
- DOM XML - It specifies an extension to the Core DOM for use with XML.

Document Tree -

→
```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="UTF-8">
        <title> Tree </title>
    </head>
    <body>
        <h1> Intro </h1>
        <br>
        <!-- This is Paragraph -->
        <p> Hello <span> Geeky </span> Shows </p>
        <ul>
            <li>
                <a href="http://www.geekyshows
                .com"> Click Me </a>
            </li>
        </ul>
    </body>
</html>
```



DOM Node Types-

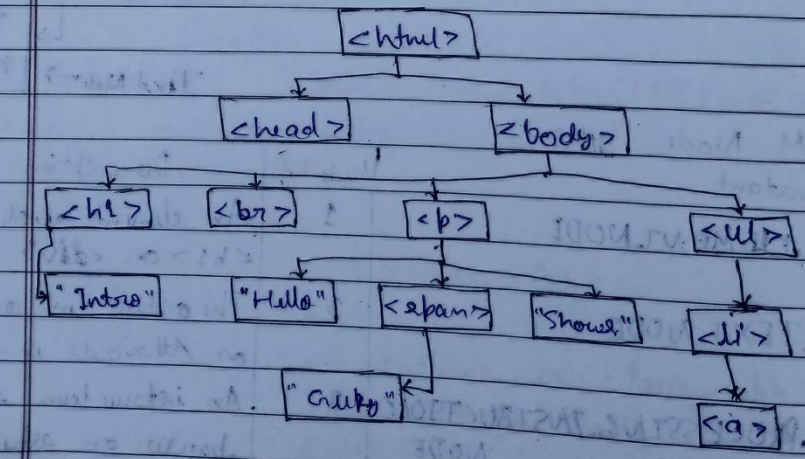
Constant	Node Type	Description
Node.ELEMENT_NODE	1	An element such as <h1> or
Node.TEXT_NODE	3	The actual Text of Element or Attribute is - "Hello"
Node.PROCESSING_INSTRUCTION_NODE	7	An instruction to a parser on object or aspect of the document <?xml version="1.0"?>
Node.COMMENT_NODE	8	A comment such as <!-- Something -->.

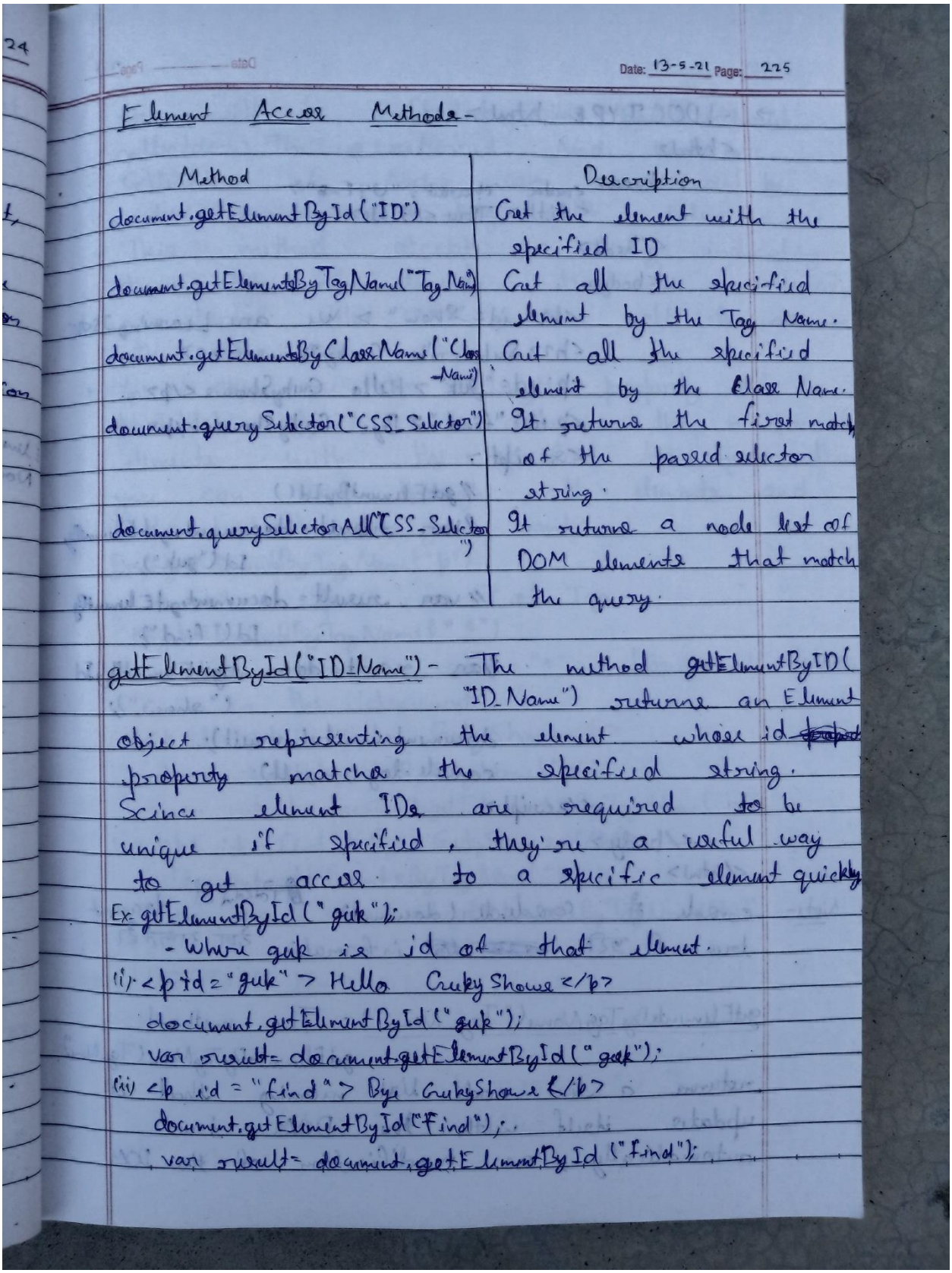
Date Page 224

Node.DOCUMENT_TYPE_NODE	10	A doctype statement <!DOCTYPE html>
Node.DOCUMENT_NODE	9	A Document Node
Node.DOCUMENT_FRAGMENT_NODE	11	A document fragment, which represents a lightweight structure to hold a collection of DOM nodes for manipulation or insertion

Node Relationship –

- Parent
- children
- First Child
- Previous Sibling
- Next Sibling
- Last Child





24

Date: 13-5-21 Page: 225

Element Access Methods -

Method	Description
document.getElementById("ID")	Get the element with the specified ID
document.getElementsByTagName("Tag_Name")	Get all the specified element by the Tag Name.
document.getElementsByClassName("Class_Name")	Get all the specified element by the Class Name.
document.querySelector("CSS_Selector")	It returns the first match of the passed selector string.
document.querySelectorAll("CSS_Selector")	It returns a node list of DOM elements that match the query.

getElementById("ID_Name") - The method getElementByID("ID_Name") returns an Element object representing the element whose id property matches the specified string. Since element IDs are required to be unique if specified, they're a useful way to get access to a specific element quickly.

Ex. getElementById("guk");

- Where guk is id of that element.

(i) Hello Cuky Shows

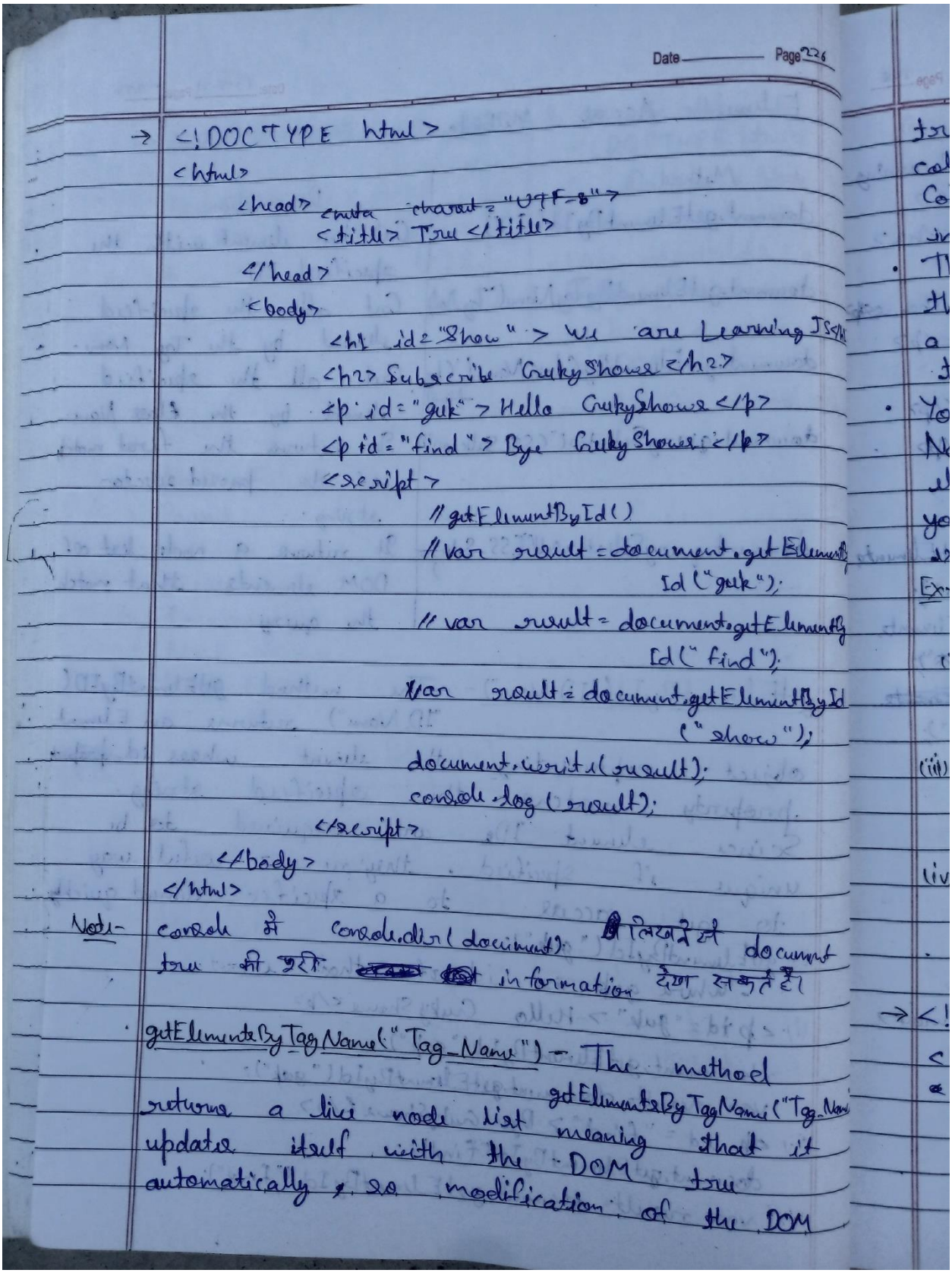
document.getElementById("guk");

var result = document.getElementById("guk");

(ii) Bye CukyShows

document.getElementById("find");

var result = document.getElementById("find");

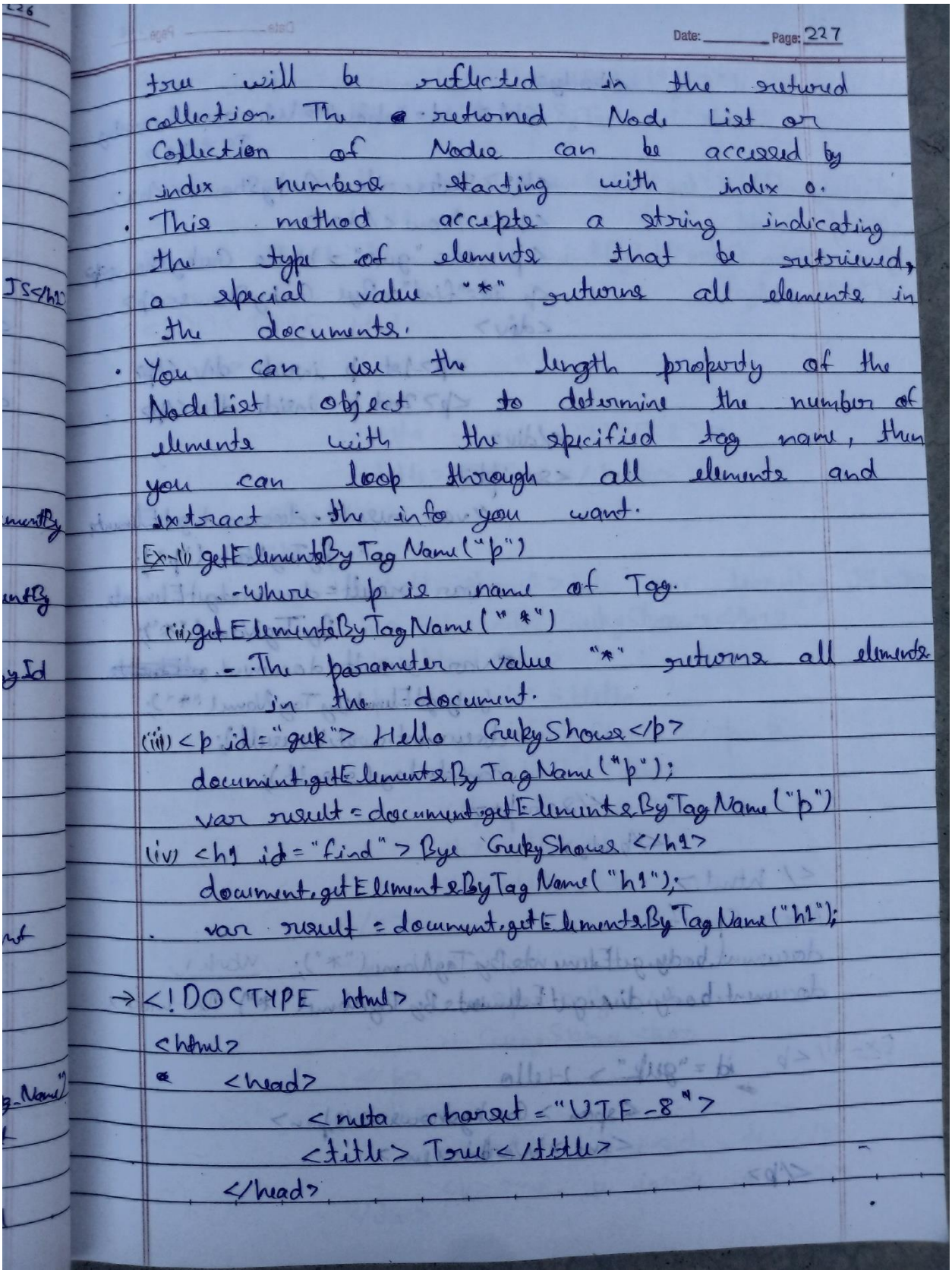


Date Page 226

```
<!DOCTYPE html>
<html>
    <head> <meta charset = "UTF-8">
        <title> True </title>
    </head>
    <body>
        <h1 id = "Show" > We are Learning JS</h1>
        <h2> Subscribe Cuky Shows </h2>
        <p id = "guk" > Hello CukyShows </p>
        <p id = "find" > Bye CukyShows </p>
        <script>
            // getElementById()
            // var result = document.getElementById("guk");
            // var result = document.getElementById("find");
            var result = document.getElementById("show");
            document.write(result);
            console.log(result);
        </script>
    </body>
</html>
```

Note- console में console.dir(document); लिखने से document true की तरह information देख सकते हैं।

getElementsByTagName("Tag_Name") - The method getElementsByTagName("Tag_Name") returns a live node list meaning that it updates itself with the DOM tree automatically so no modification of the DOM



Date: Page: 227

true will be reflected in the returned collection. The returned Node List or Collection of Node can be accessed by index numbers starting with index 0.

- This method accepts a string indicating the type of elements that be retrieved, a special value "*" returns all elements in the documents.
- You can use the length property of the NodeList object to determine the number of elements with the specified tag name, then you can loop through all elements and extract the info you want.

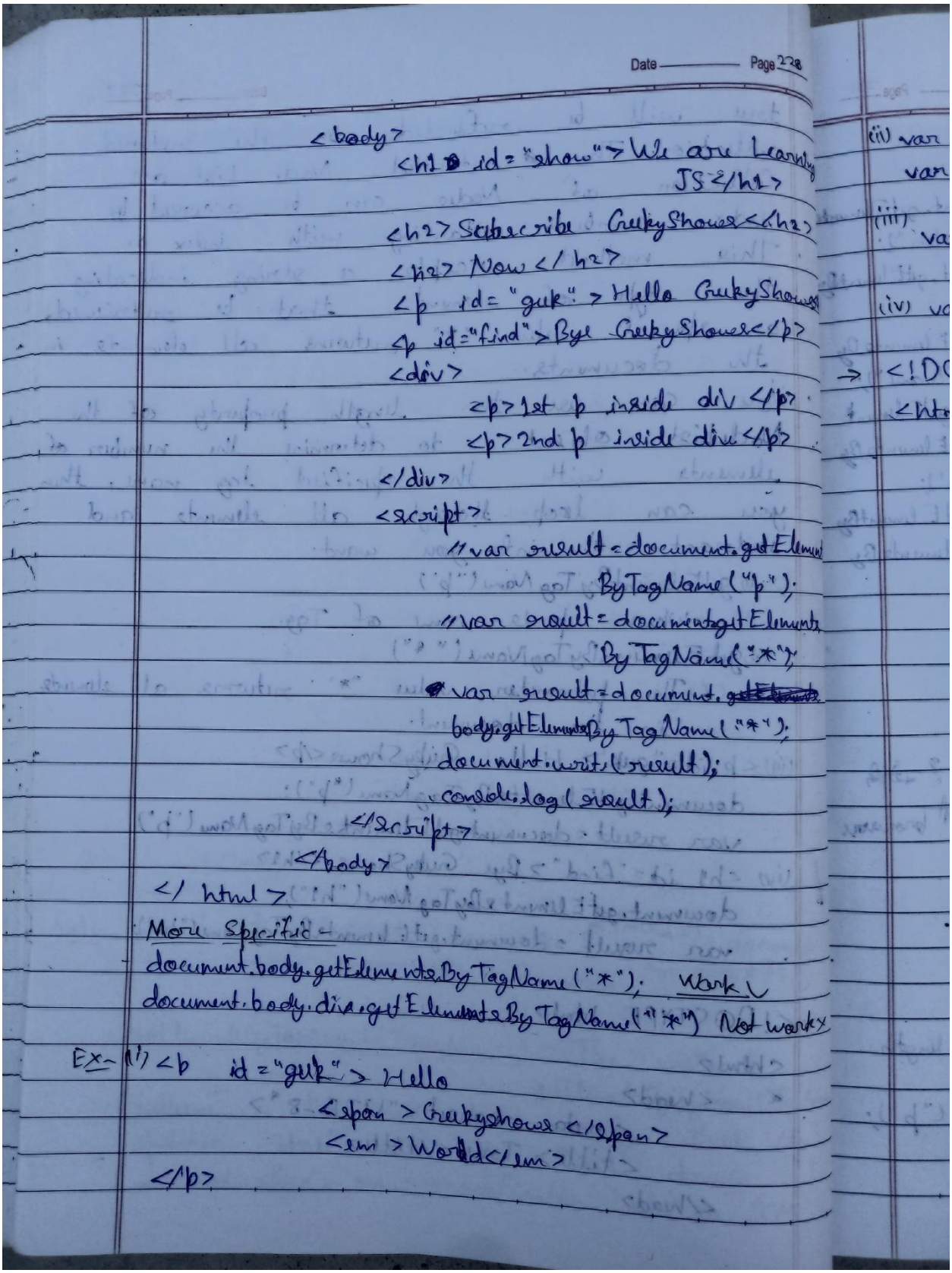
Ex- (i) getElementsByTagName("p")
- where p is name of Tag.

(ii) getElementsByTagName("*")
- The parameter value "*" returns all elements in the document.

(iii) Hello GeekyShows
document.getElementsByTagName("p");
var result = document.getElementsByTagName("p")

(iv) <h1 id="find"> Bye GeekyShows </h1>
document.getElementsByTagName("h1");
var result = document.getElementsByTagName("h1");

→
```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
    <title> True </title>
  </head>
```



```
<body>
    <h1 id="show"> We are Learning JS </h1>
    <h2> Subscribe GeekyShows </h2>
    <h2> Now </h2>
    <p id="guk"> Hello GeekyShows
    <p id="find"> Bye GeekyShows </p>
    <div>
        <p> 1st p inside div </p>
        <p> 2nd p inside div </p>
    </div>
    <script>
        // var result = document.getElement
            ByTagName("p");
        // var result = document.getElements
            ByTagName("*");
        var result = document.getElements
            body.getElementsByTagName("*");
        document.write(result);
        console.log(result);
    </script>
</body>
</html>
```

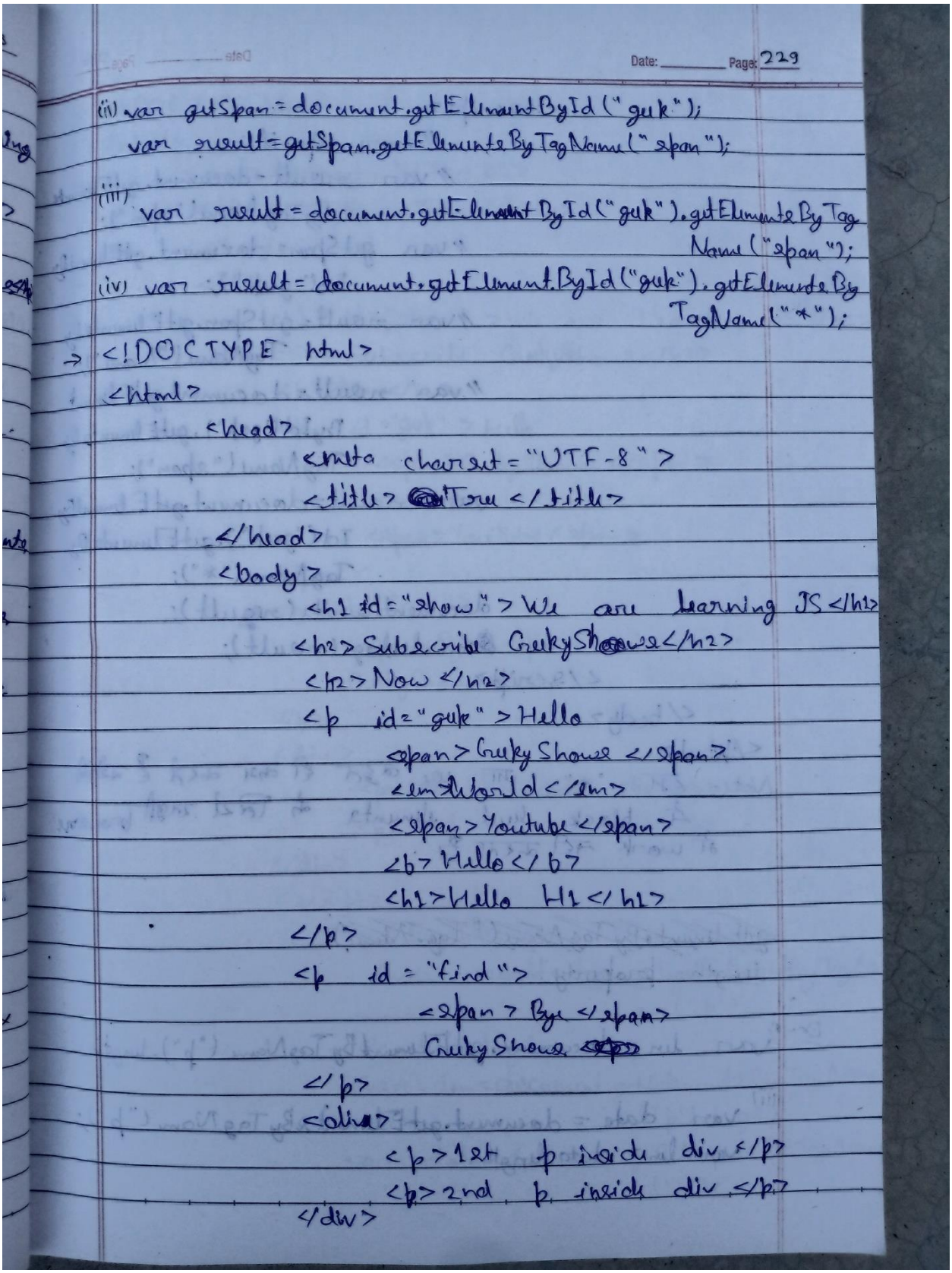
More Specific -

document.body.getElementsByTagName("*"); Work ✓

document.body.div.getElementsByTagName("*") Not work ✗

Ex- // <b id="guk"> Hello

```
<span> Geekyshows </span>
<em> World </em>
</p>
```



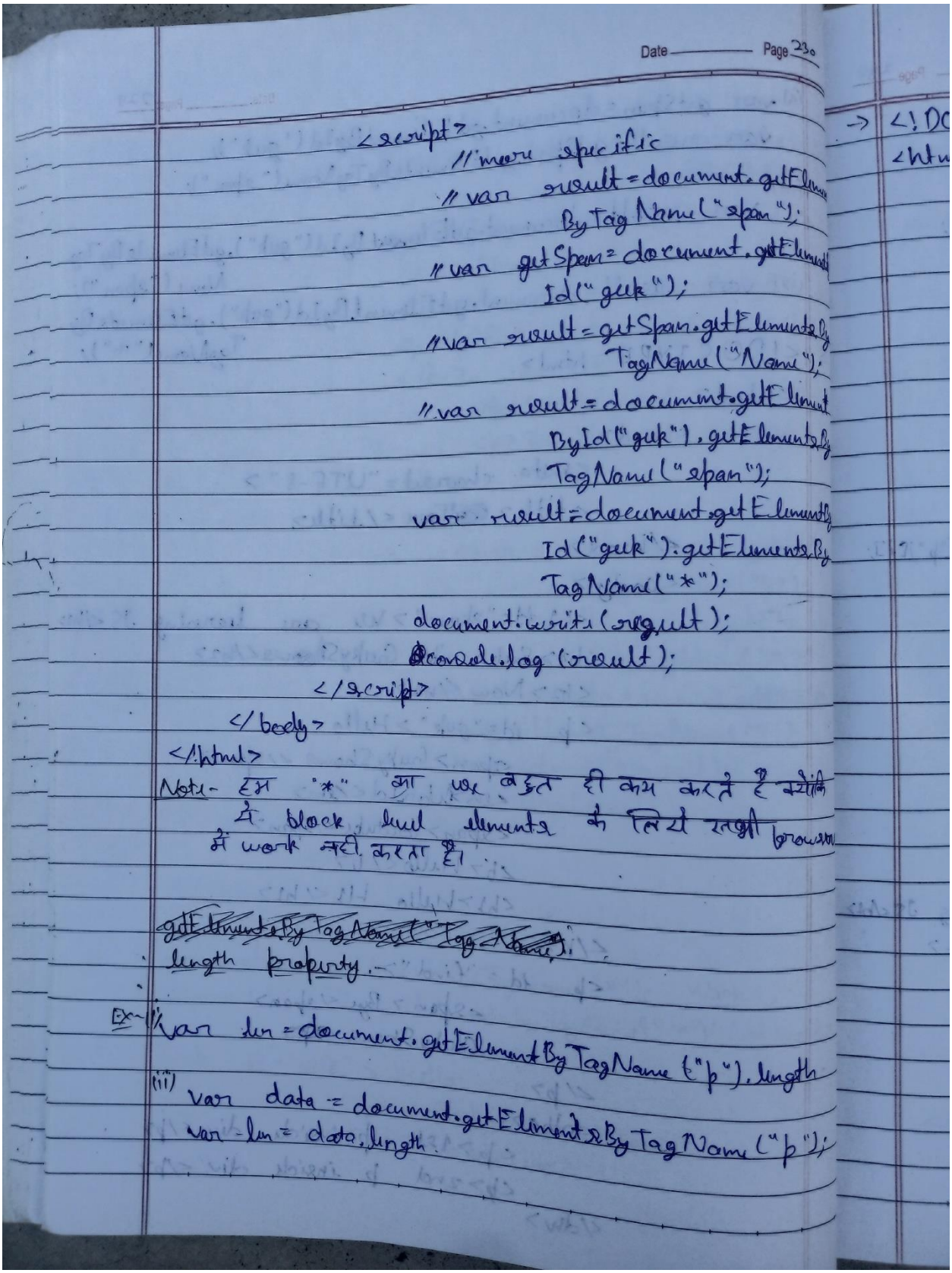
Date: Page: 229

(ii) var getSpan = document.getElementById("geek");
var result = getSpan.getElementsByTagName("span");

(iii) var result = document.getElementById("geek").getElementsByTagName("span");

(iv) var result = document.getElementById("geek").getElementsByTagName("*");

→
```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="UTF-8">
        <title> True </title>
    </head>
    <body>
        <h1 id="show"> We are learning JS </h1>
        <h2> Subscribe GeekyShows </h2>
        <h2> Now </h2>
        <p id="geek"> Hello
            <span> Geeky Shows </span>
            <em> World </em>
            <span> Youtube </span>
            <b> Hello </b>
            <h1> Hello H1 </h1>
        </p>
        <p id="find">
            <span> Bye </span>
            Geeky Shows
        </p>
        <div>
            <p> 1st p inside div </p>
            <p> 2nd p inside div </p>
        </div>
```



```
<script>
    // more specific
    // var result = document.getElement
        ByTagName("span");
    // var getSpan = document.getElement
        Id("geek");
    // var result = getSpan.getElements
        TagName("Name");
    // var result = document.getElement
        ById("geek").getElements
        TagName("span");
    var result = document.getElement
        Id("geek").getElements
        TagName("*");
    document.write(result);
    console.log(result);
</script>
</body>
</html>
```

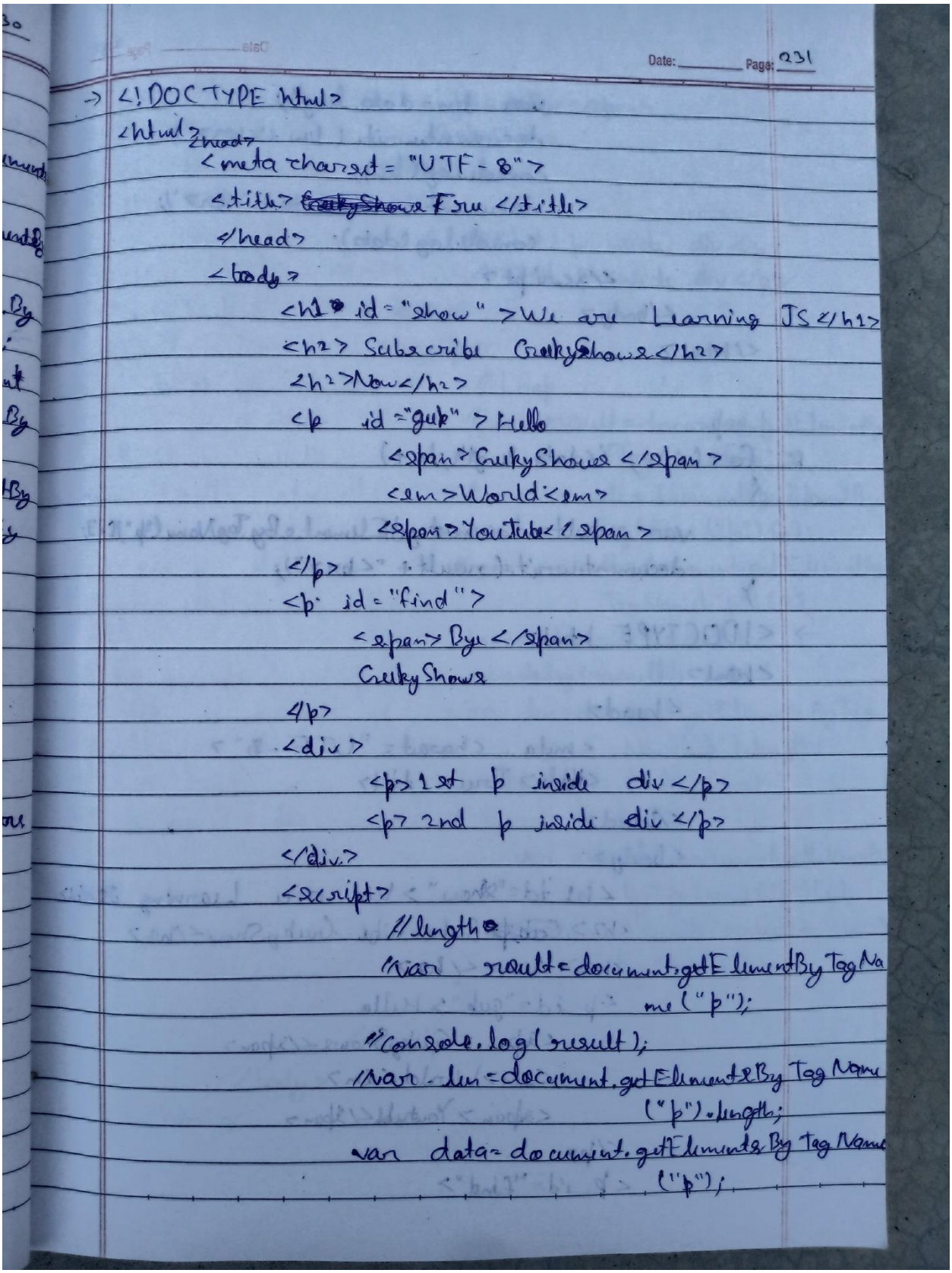
Note- हम "*" का use बहुत ही कम करते हैं क्योंकि ये block level elements के लिये रखी browser में work नहीं करता है।

~~getElementsByTagName("Tag Name");~~

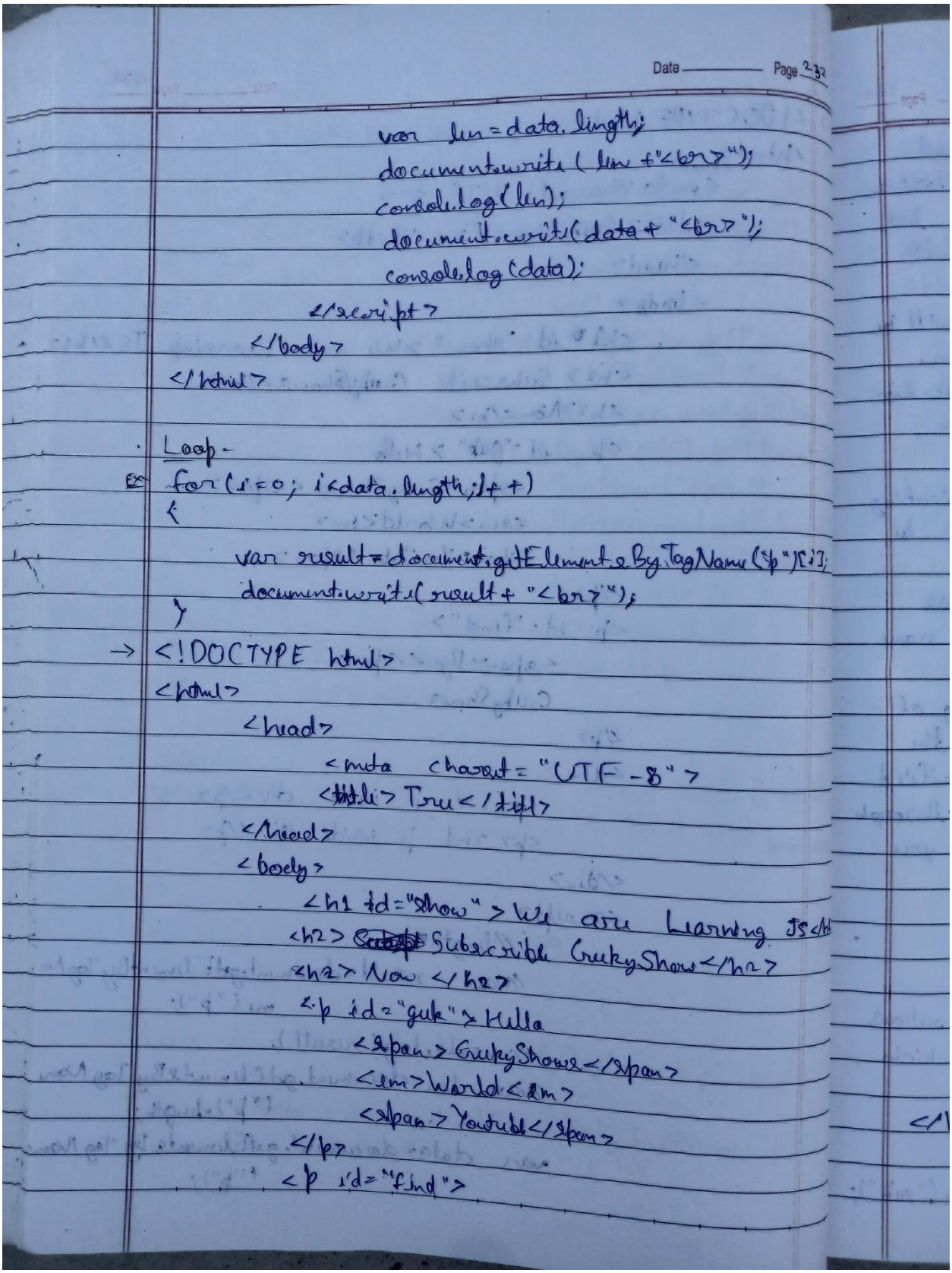
length property -

Ex- (i) var len = document.getElementByTagName("p").length

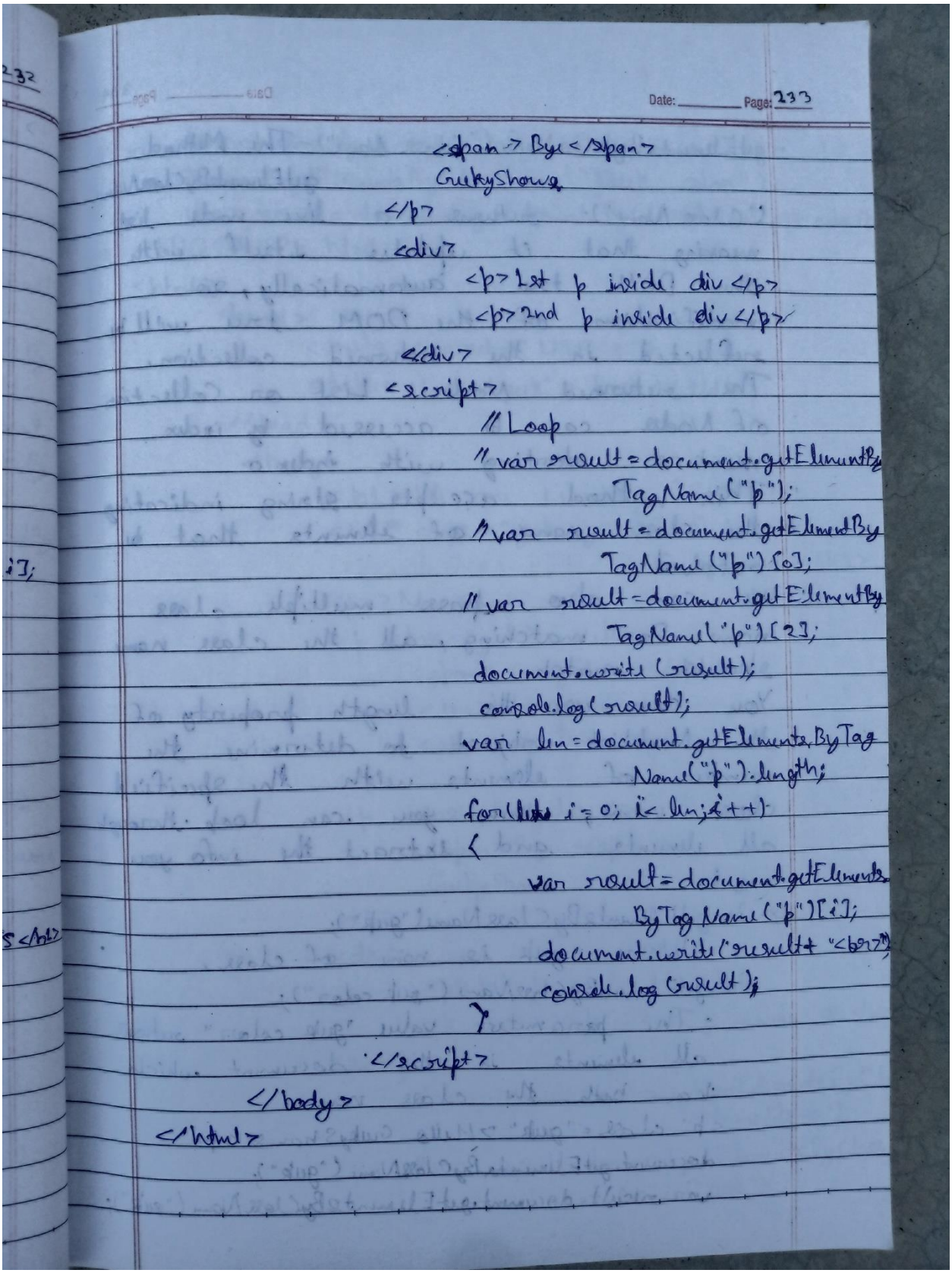
(ii) var data = document.getElementsByTagName("p");
var len = data.length



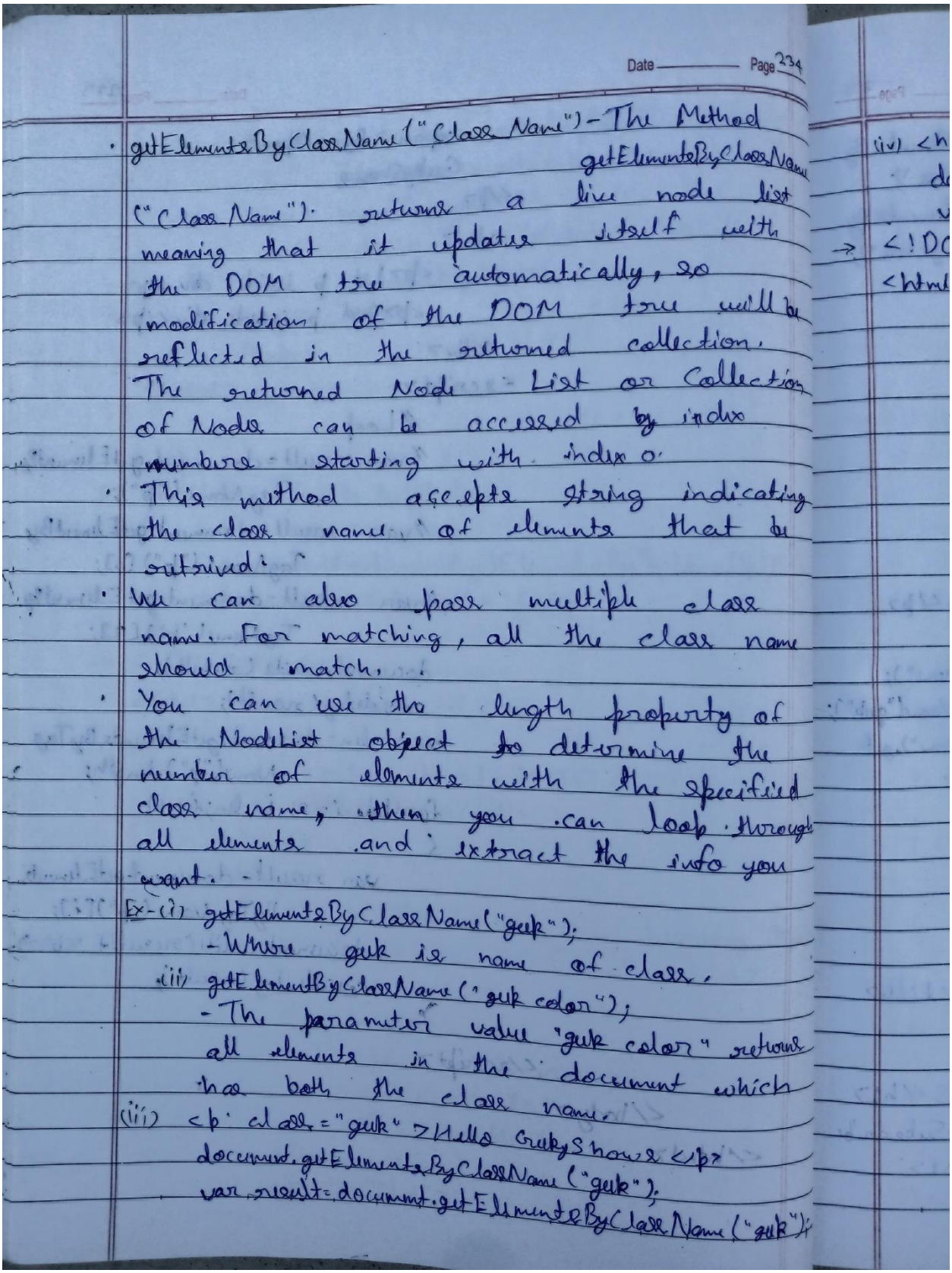
```
Date: Page: 231

-> <!DOCTYPE html>
<html>
    <head>
        <meta charset = "UTF-8">
        <title> GeekyShows Fun </title>
    </head>
    <body>
        <h1 id = "show" > We are Learning JS </h1>
        <h2> Subscribe GeekyShows </h2>
        <h2> Now </h2>
        <p  id = "gub" > Hello
            <span> GeekyShows </span>
            <em> World <em>
            <span> Youtube </span>
        </p>
        <p id = "find" >
            <span> By </span>
            GeekyShows
        </p>
        <div>
            <p> 1st p inside div </p>
            <p> 2nd p inside div </p>
        </div>
        <script>
            //length
            //var result = document.getElementsByTagNa
                                me ("p");
            //console.log (result);
            //var len = document.getElementsByTagName
                                ("p").length;
            var data = document.getElementsByTagName
                                ("p");
```



```
Date ____ Page 2-32

        var len = data.length;
        document.write(len + "<br>");
        console.log(len);
        document.write(data + "<br>");
        console.log(data);
    </script>
</body>
</html>

Loop -
for(i=0; i<data.length; i++)
{
    var result = document.getElementsByTagName('p')[i];
    document.write(result + "<br>");
}

<!DOCTYPE html>
<html>
    <head>
        <meta charset="UTF-8">
        <title> True </title>
    </head>
    <body>
        <h1 id="show"> We are Learning JS </h1>
        <h2> Subscribe GeekyShows </h2>
        <h2> Now </h2>
        <p id="guk"> Hello
            <span> GeekyShows </span>
            <em> World </em>
            <span> Youtube </span>
        </p>
        <p id="find">
```



```
<span> By </span>
GeekyShows
</p>
<div>
    <p> 1st p inside div </p>
    <p> 2nd p inside div </p>
</div>
<script>
    // Loop
    // var result = document.getElementBy
                    TagName("p");
    // var result = document.getElementBy
                    TagName("p")[0];
    // var result = document.getElementBy
                    TagName("p")[2];
    document.write(result);
    console.log(result);
    var len = document.getElements.ByTag
                    Name("p").length;
    for(let i = 0; i< len;i++)
    {
        var result = document.getElements
                    ByTagName("p")[i];
        document.write(result+ "<br>");
        console.log(result);
    }
</script>
</body>
</html>
```



Date _____ Page 234

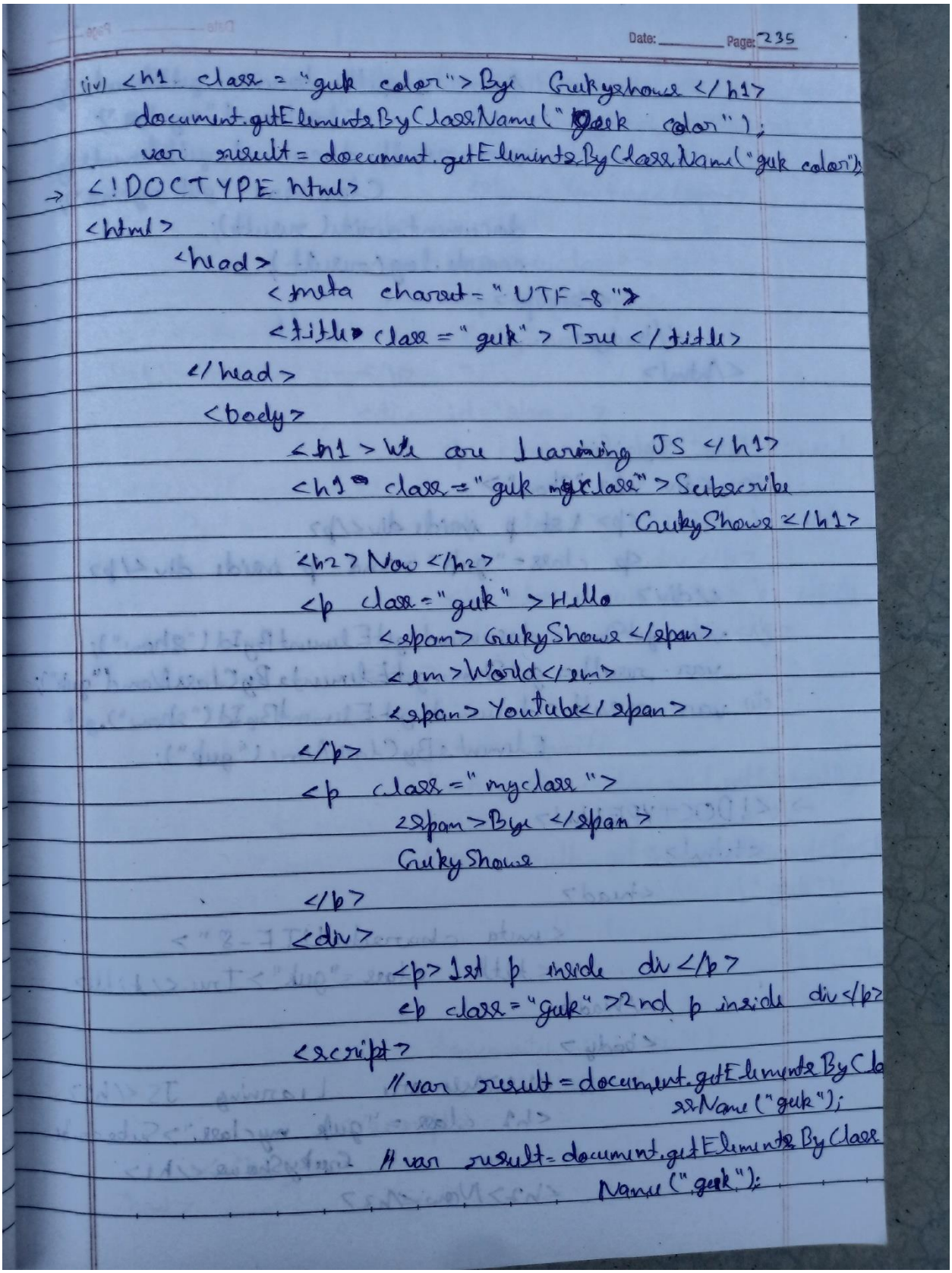
- getElementsByClassName("Class Name") - The Method getElementsByClassName ("Class Name"). returns a live node list meaning that it updates itself with the DOM tree automatically, so modification of the DOM tree will be reflected in the returned collection. The returned Node List or Collection of Nodes can be accessed by index numbers starting with index 0.
- This method accepts String indicating the class name of elements that to retrived.
- We can also pass multiple class name. For matching, all the class name should match.
- You can use the length property of the NodeList object to determine the number of elements with the specified class name, then you can loop through all elements and extract the info you want.

Ex- (i) getElementsByClassName("geek");
- Where geek is name of class.

(ii) getElementByClassName("geek color");
- The paramter value "geek color" returns all elements in the document which has both the class name.

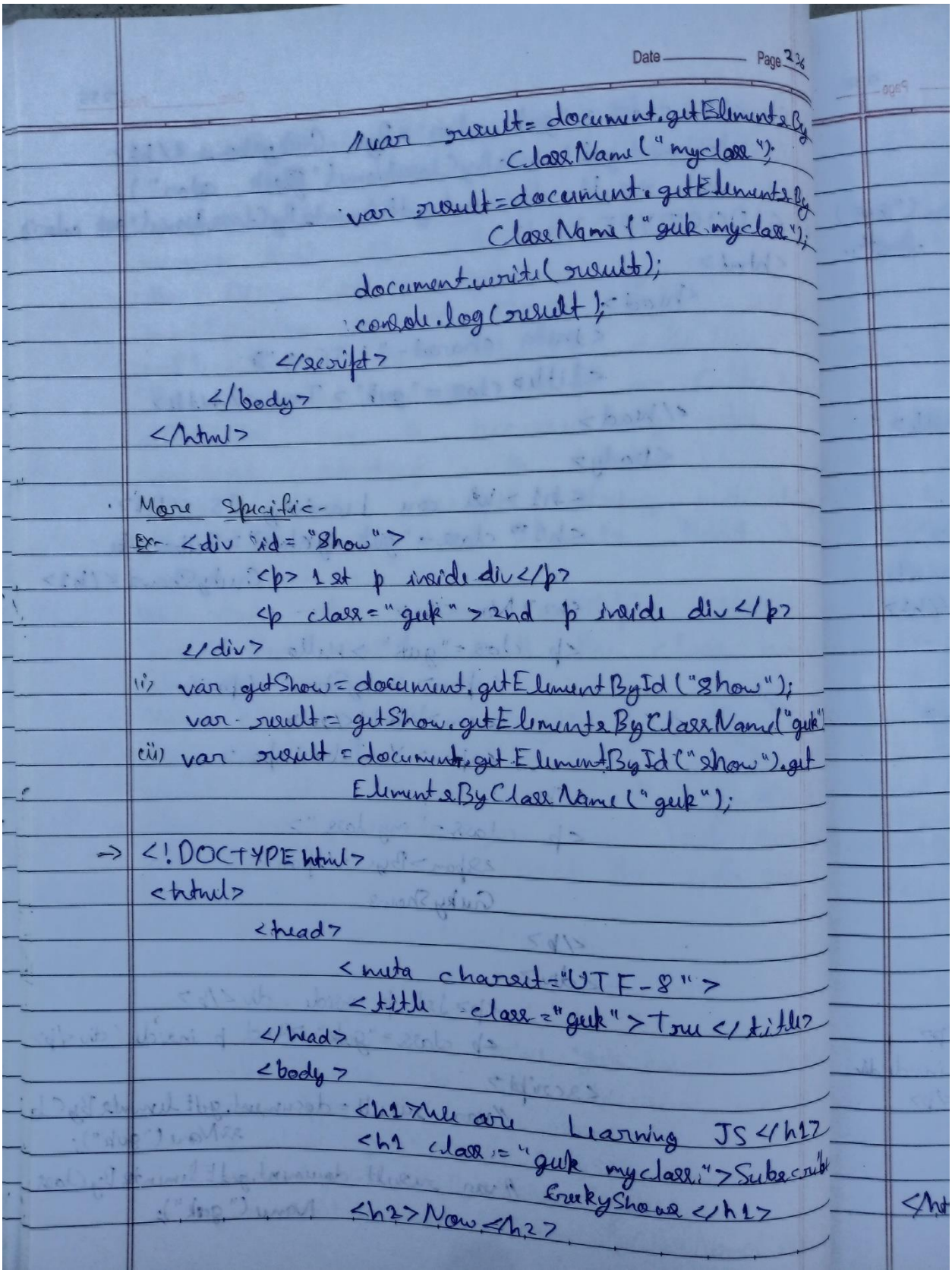
(iii) Hello Geekyshows
document.getElementsByClassName("geek");
var result = document.getElementsByClassName("geek");

(iv) <h
d
v
<!D
<html



(iv) <h1 class = "guk color"> By Gukyshows </h1>
document.getElementsByClassName("guk color");
var result = document.getElementsByClassName("guk color");

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="UTF-8">
        <title class="guk">True</title>
    </head>
    <body>
        <h1>We are learning JS</h1>
        <h1 class="guk myclass">Subscribe
            GukyShows</h1>
        <h2>Now</h2>
        <p class="guk">Hello
            <span>GukyShows</span>
            <em>World</em>
            <span>Youtube</span>
        </p>
        <p class="myclass">
            <span>By</span>
            GukyShows
        </p>
        <div>
            <p>1st p inside div</p>
            <p class="guk">2nd p inside div</p>
        </div>
        <script>
            // var result = document.getElementsByClassName("guk");
            // var result = document.getElementsByClassName("geek");
```



Date _____ Page 236

```
//var result= document.getElementsBy
                ClassName("myclass");
var result=document.getElementsBy
                ClassName("geek myclass");
document.write(result);
console.log(result);
</script>
</body>
</html>
```

More Specific-

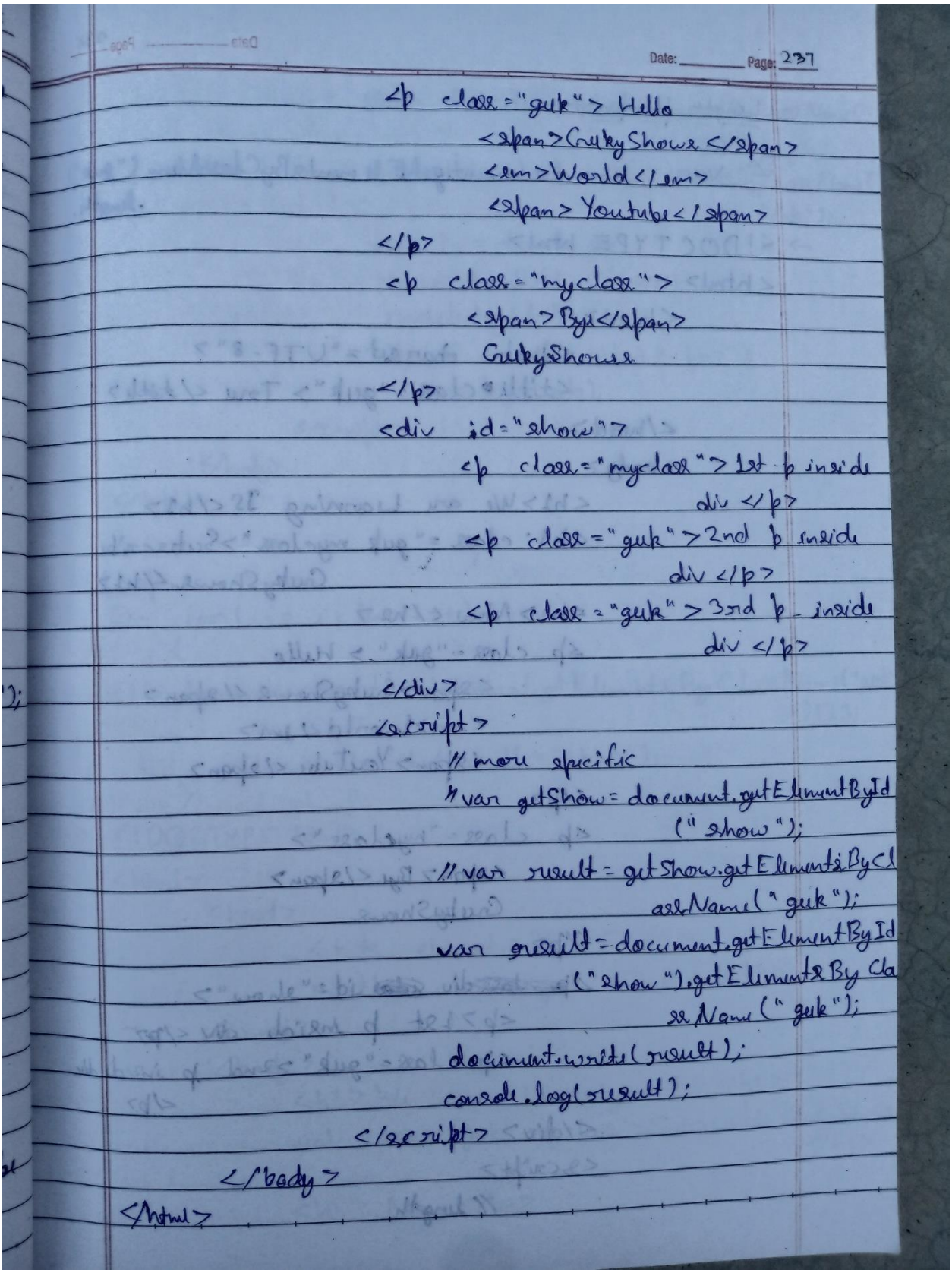
Ex- `<div id="show">`
`<p> 1st p inside div</p>`
`<p class="geek"> 2nd p inside div</p>`
`</div>`

(i) `var getShow=document.getElementById("show");`
`var result=getShow.getElementsByClassName("geek")`

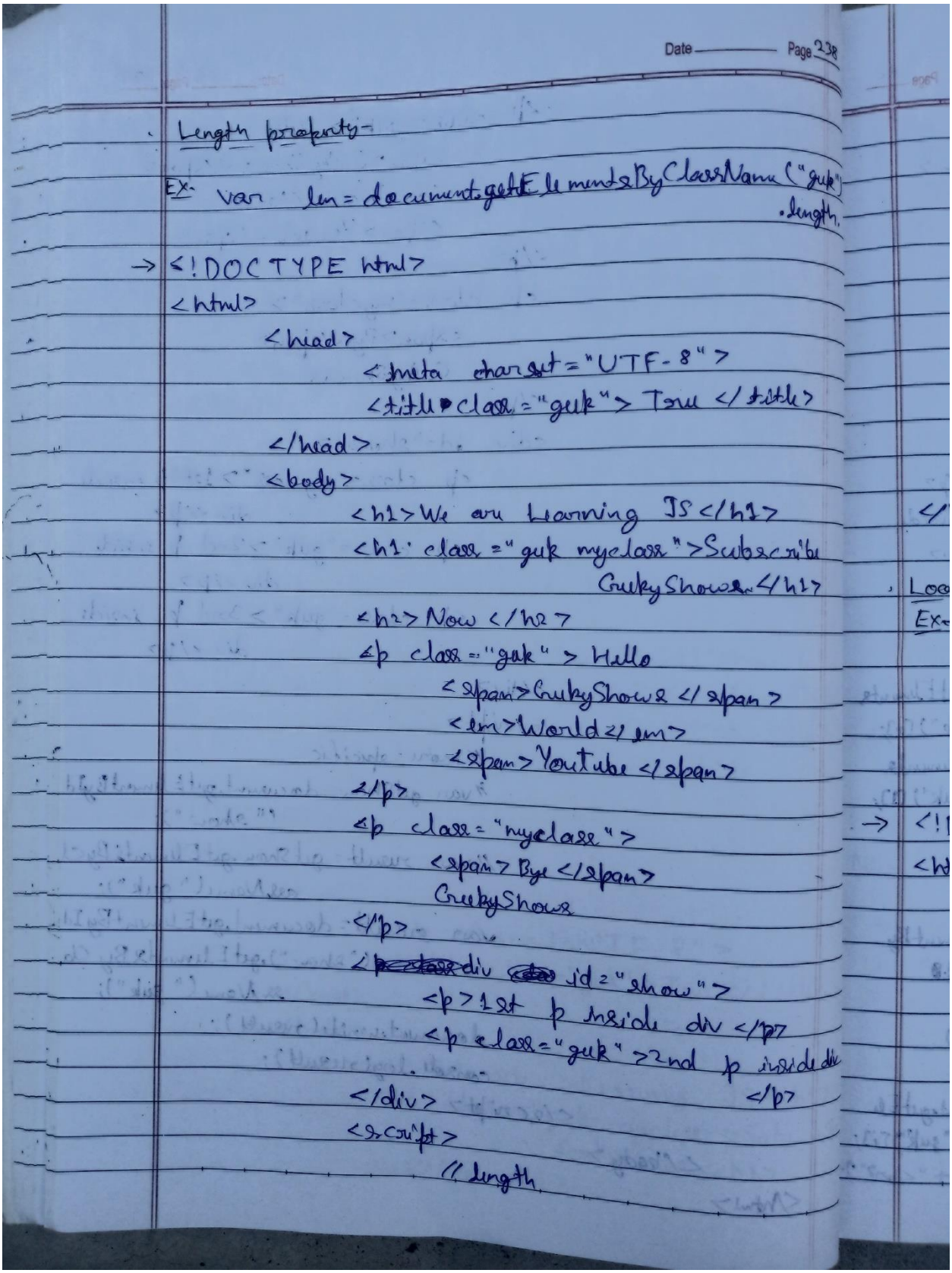
(ii) `var result=document.getElementById("show").getElementsByClassName("geek");`

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="UTF-8">
        <title class="geek">True</title>
    </head>
    <body>
        <h1>We are Learning JS</h1>
        <h1 class="geek myclass">Subscribe
                EnekyShows</h1>
        <h2>Now</h2>
```

</ht



```
<p class="guk"> Hello
    <span>GukyShows</span>
    <em>World</em>
    <span>Youtube</span>
</p>
<p class="myclass">
    <span>By</span>
    GukyShows
</p>
<div id="show">
    <p class="myclass">1st p inside
        div</p>
    <p class="guk">2nd p inside
        div</p>
    <p class="guk">3rd p inside
        div</p>
</div>
<script>
    // more specific
    // var getShow = document.getElementById
        ("show");
    // var result = getShow.getElementsByCl
        assName("guk");
    var result = document.getElementById
        ("show").getElementsByCla
        ssName("guk");
    document.write(result);
    console.log(result);
</script>
</body>
</html>
```

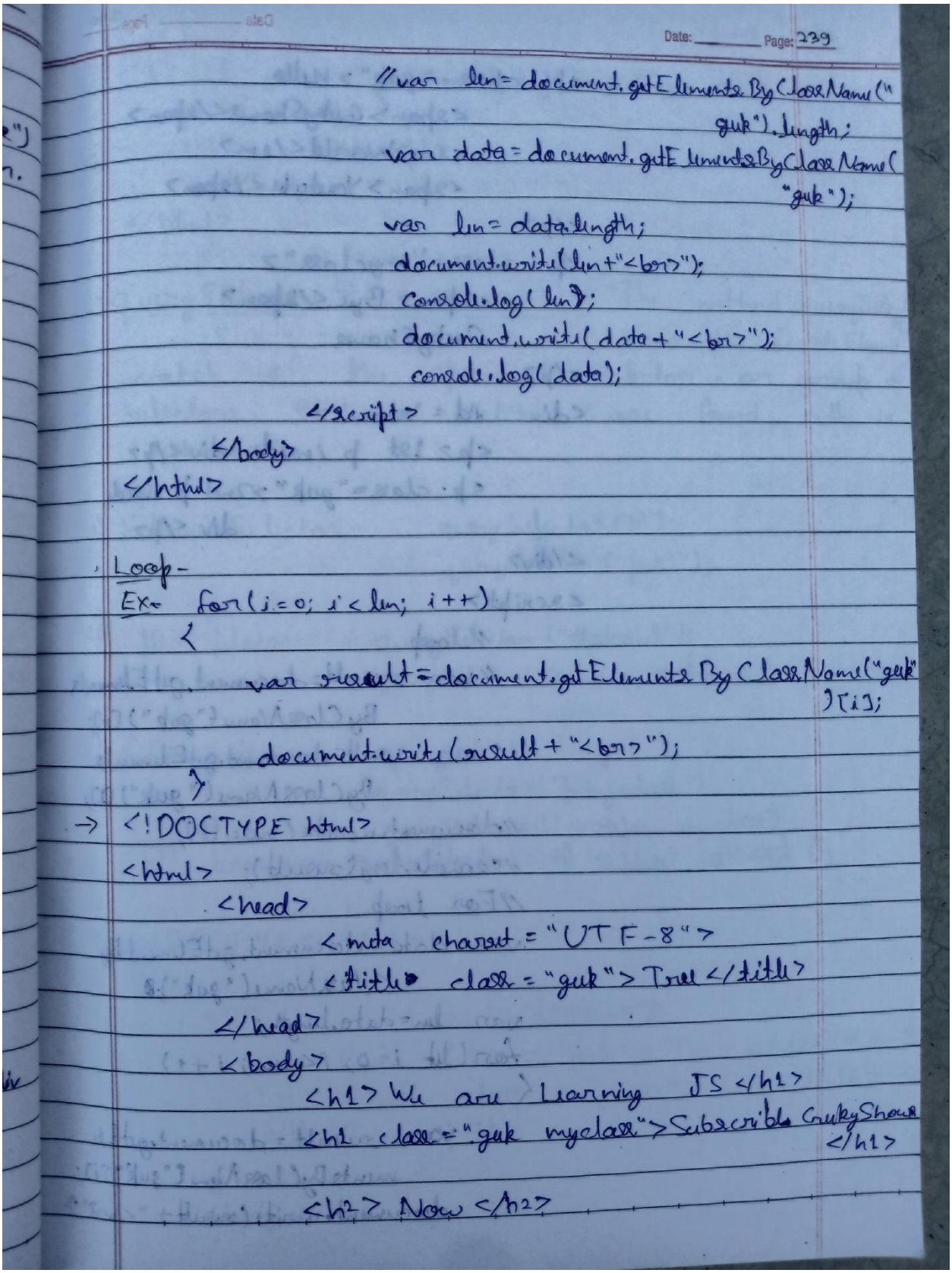


Date _____ Page 238

Length property-

EX- var len = document.getElementsByClassName("guk").length;

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="UTF-8">
        <title class="guk"> True </title>
    </head>
    <body>
        <h1> We are Learning JS </h1>
        <h1 class="guk myclass"> Subscribe GeekyShows </h1>
        <h2> Now </h2>
        <p class="guk"> Hello
            <span> GeekyShows </span>
            <em> World </em>
            <span> Youtube </span>
        </p>
        <p class="myclass">
            <span> Bye </span>
            GeekyShows
        </p>
        <div id="show">
            <p> 1st p inside div </p>
            <p class="guk"> 2nd p inside div </p>
        </div>
        <script>
            // length
```

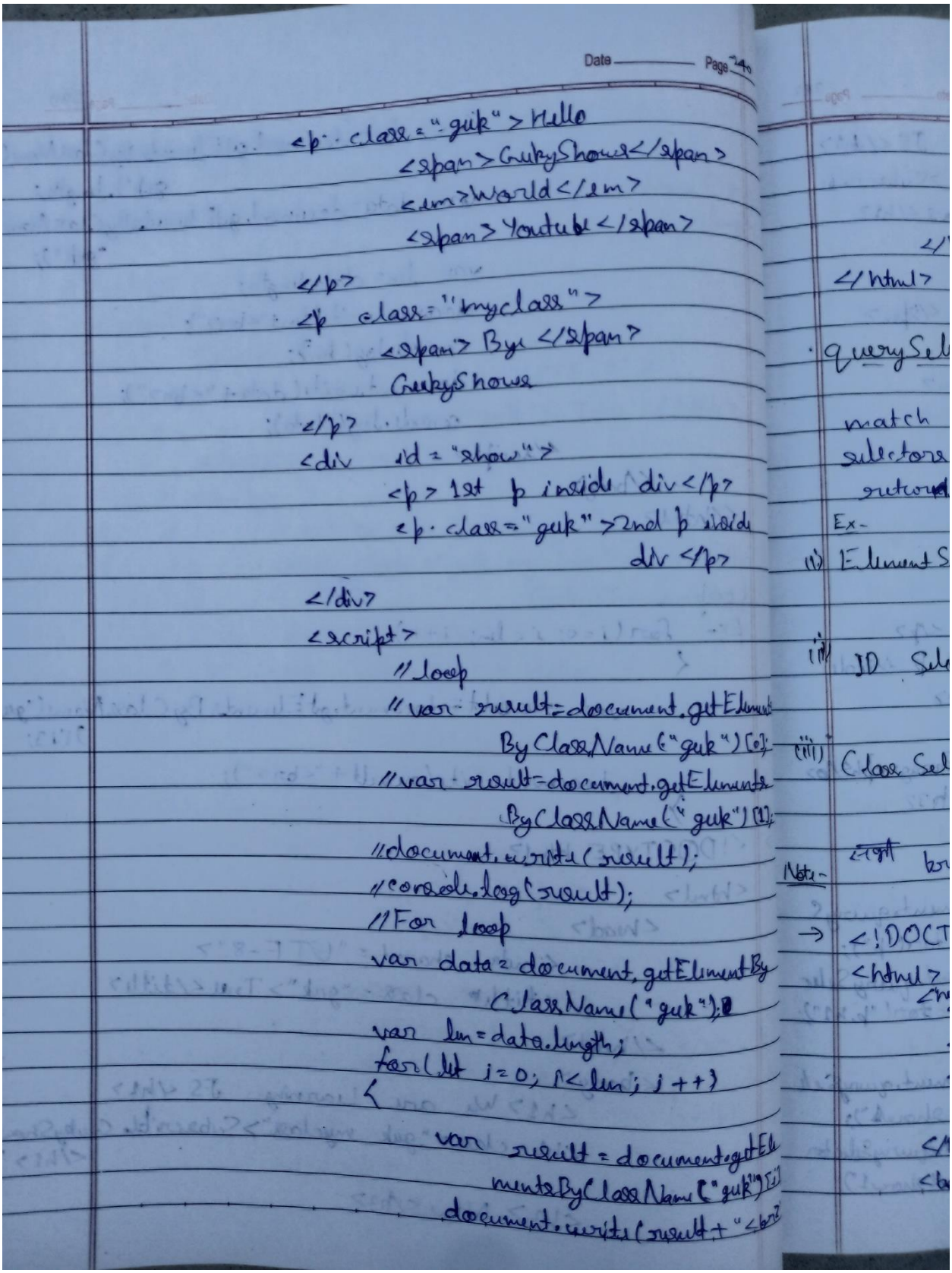


```
//var len= document.getElementsByClassName("
                                    guk").length;
var data= document.getElementsByClassName(
                                    "guk");
var len= data.length;
document.write(len+"<br>");
console.log(len);
document.write(data+"<br>");
console.log(data);
</script>
</body>
</html>
```

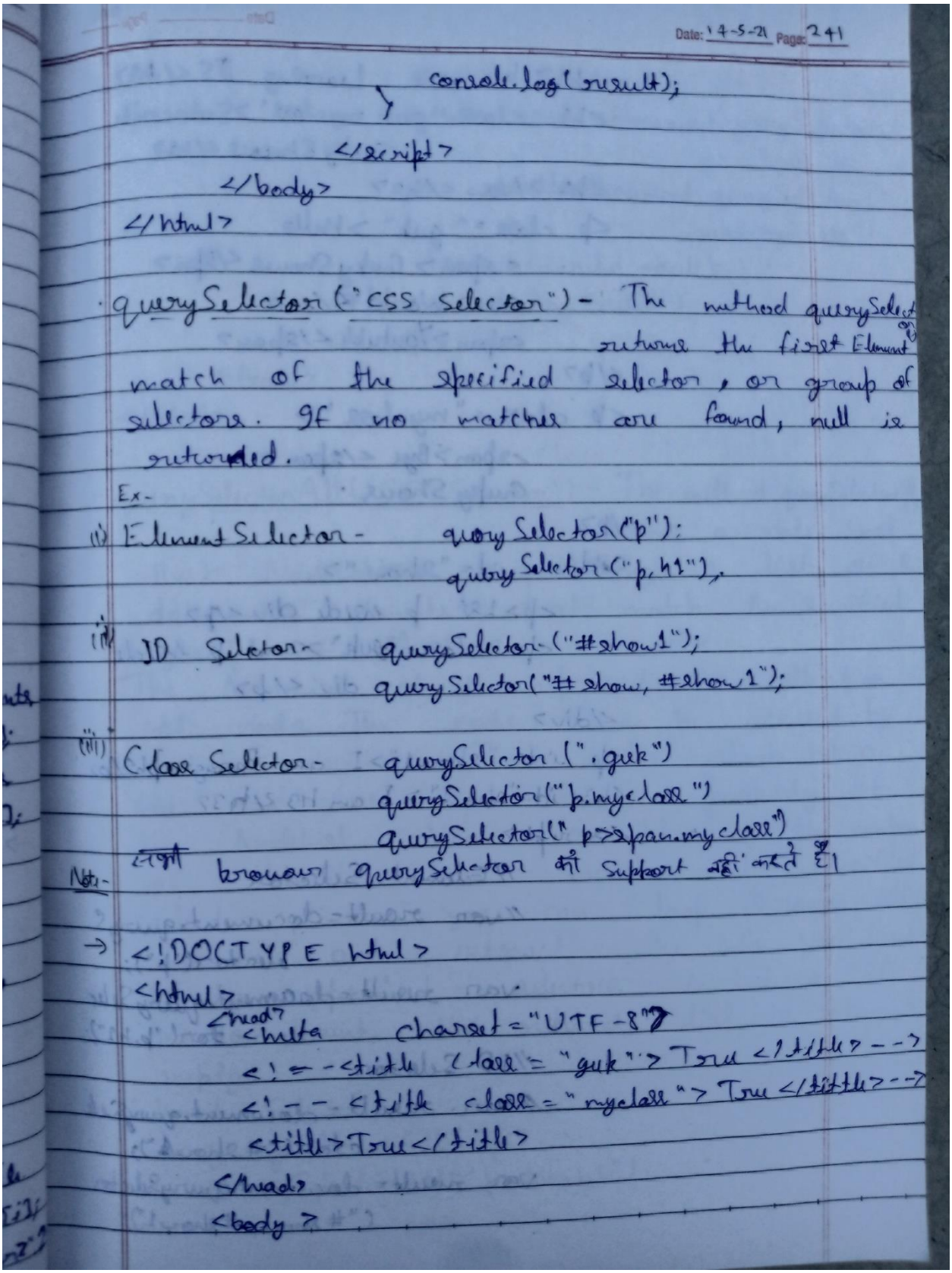
Loop-

Ex-
```
for(i=0; i<len; i++)
{
    var result=document.getElementsByClassName("guk"
                                    )[i];
    document.write(result+"<br>");
}
```

→
```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="UTF-8">
        <title class="guk">Trial</title>
    </head>
    <body>
        <h1>We are Learning JS</h1>
        <h1 class="guk myclass">Subscribe GukyShows
                                    </h1>
        <h2>Now</h2>
```



```
<p class="guk">Hello
    <span>GukyShows</span>
    <em>World</em>
    <span>Youtube</span>
</p>
<p class="myclass">
    <span>By</span>
    GeekyShows
</p>
<div id="show">
    <p>1st p inside div</p>
    <p class="guk">2nd p inside
        div</p>
</div>
<script>
    // loop
    // var result=document.getElement
        ByClassName("guk")[0];
    // var result=document.getElements
        ByClassName("guk")[1];
    // document.write(result);
    // console.log(result);
    // For loop
    var data=document.getElementBy
        ClassName("guk");
    var len=data.length;
    for(let i=0; i<len; i++)
    {
        var result=document.getEle
            mentsByClassName("guk")[i]
        document.write(result+"<br>
```



Date: 14-5-21 Page: 241

```
    console.log(result);
}
</script>
</body>
</html>
```

· querySelector("CSS Selector") - The method querySelector returns the first Element of match of the specified selector, or group of selectors. If no matches are found, null is returned.

Ex-

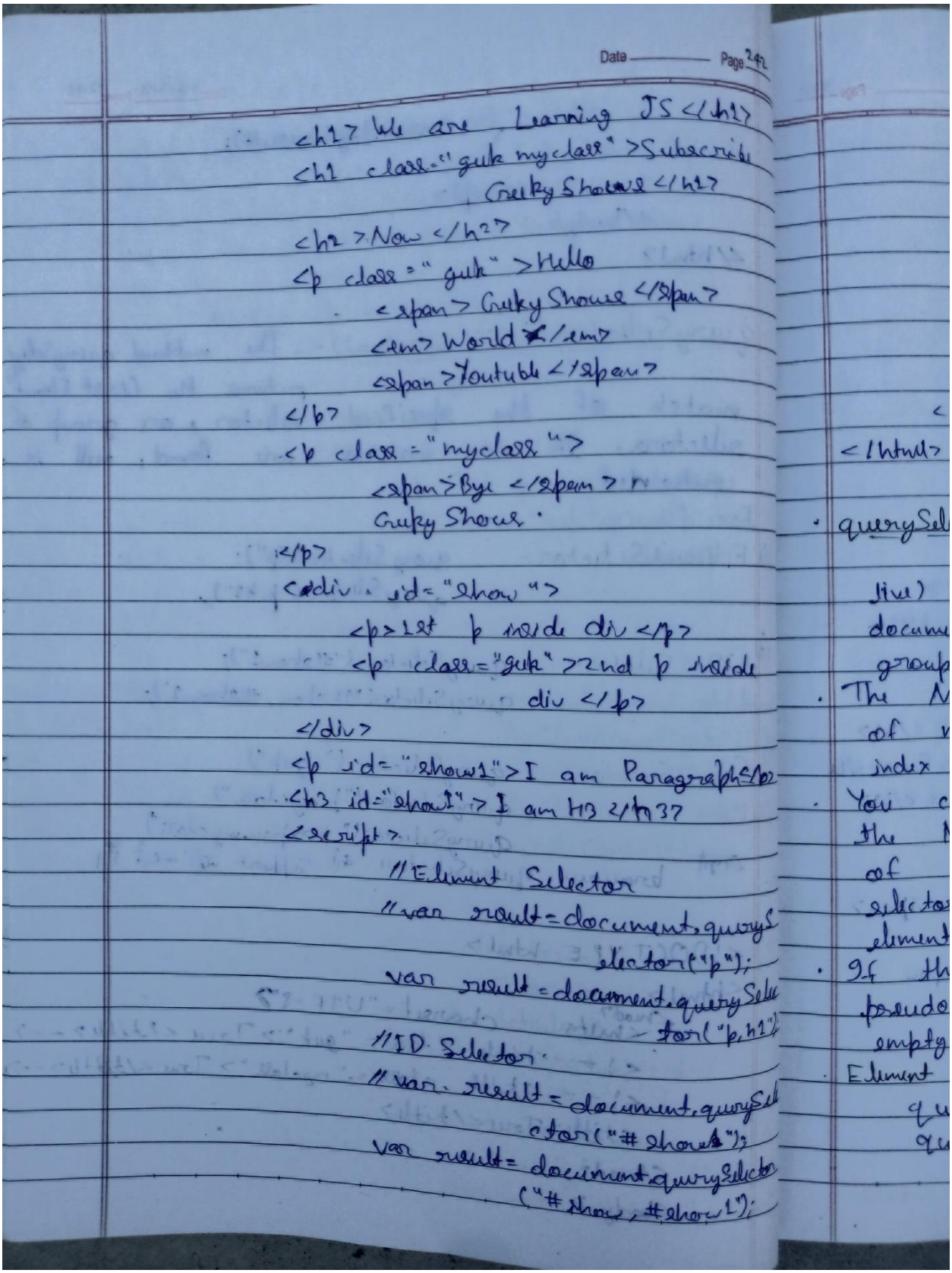
(i) Element Selector - querySelector("p");
querySelector("p, h1");

(ii) ID Selector - querySelector("#show1");
querySelector("#show, #show1");

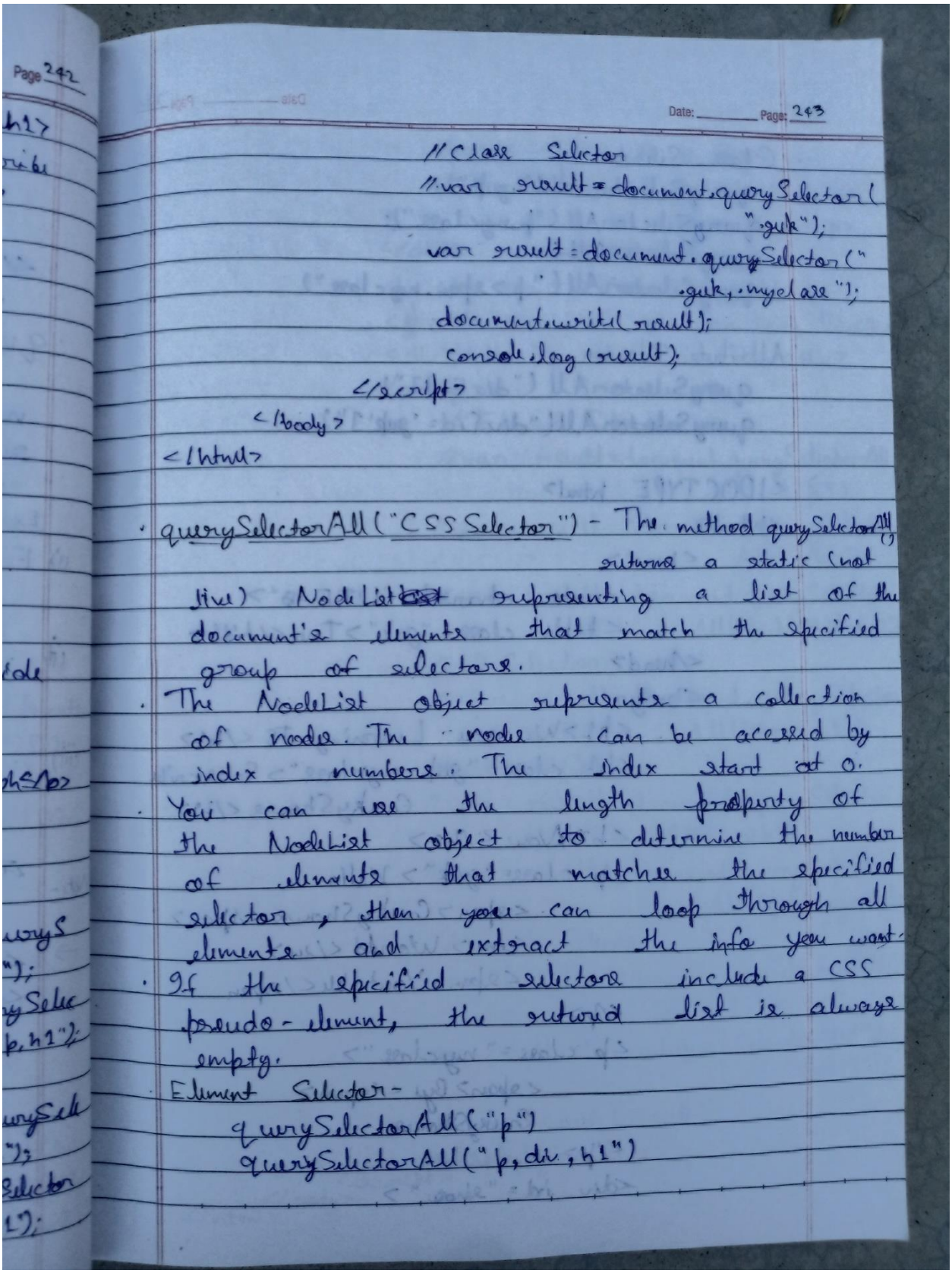
(iii) Class Selector - querySelector(".guk")
querySelector("p.myclass")
querySelector("p>span.myclass")

Note- ~~Right~~ browser querySelector को support नहीं करते हैं।

→
```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<!--<title class="guk">True</title>-->
<!--<title class="myclass">True</title>-->
<title>True</title>
</head>
<body>
```



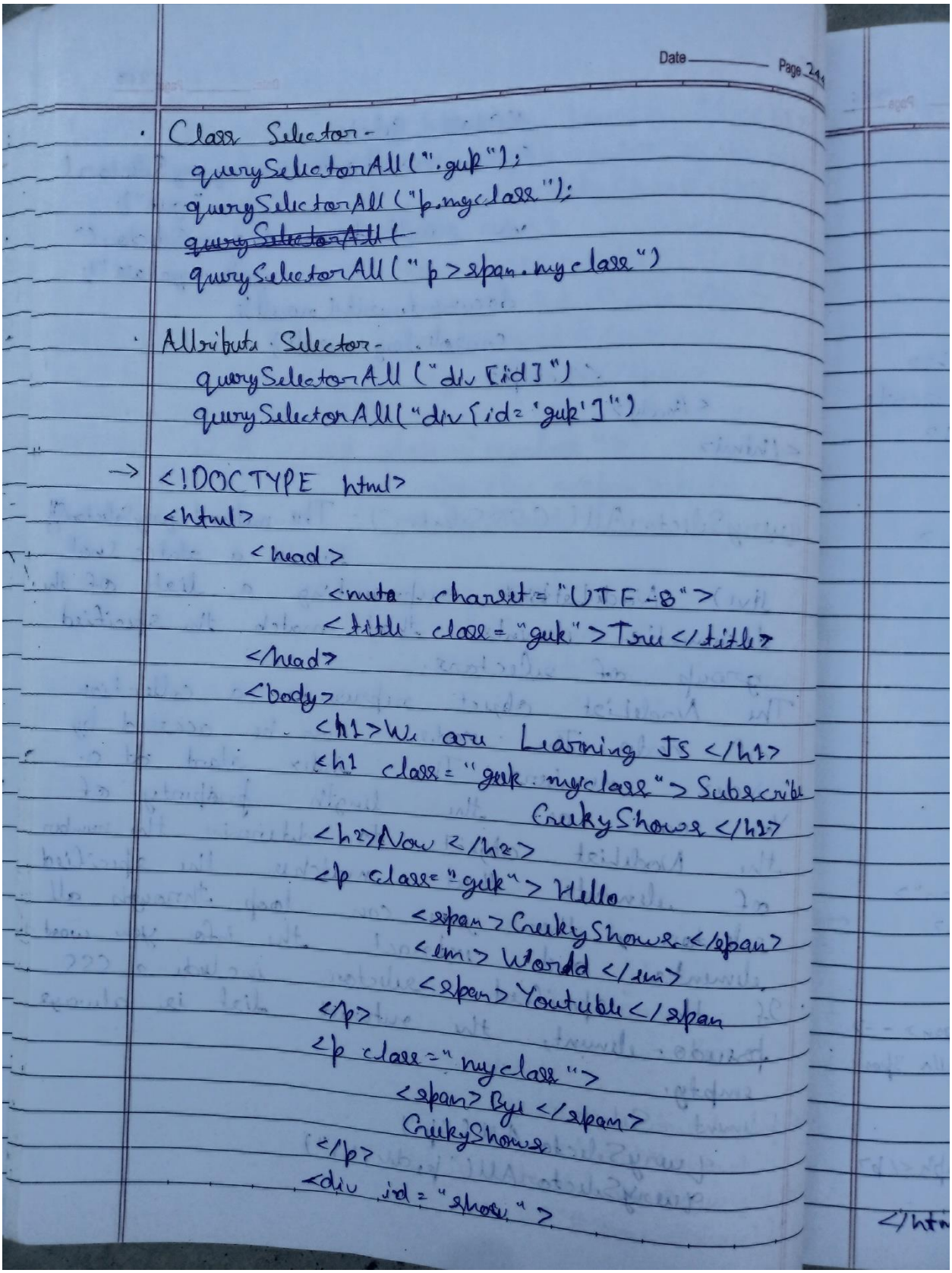
```
<h1> We are Learning JS </h1>
<h1 class="geek myclass" > Subscribe
        Geeky Shows </h1>
<h2 > Now </h2>
<p class = " geek " > Hello
    <span > Geeky Shows </span>
    <em> World </em>
    <span > Youtube </span>
</p>
<p class = " myclass " >
    <span > By </span >
    Geeky Shows.
</p>
<div id = " show " >
    <p > 1st p inside div </p>
    <p class = "geek" > 2nd p inside
        div </p>
</div>
<p id = " show1 " > I am Paragraph </p>
<h3 id = " show1 " > I am H3 </h3>
<script >
    // Element Selector
    // var result = document.querySelector("p");
    var result = document.querySelector("p, h1");
    // ID Selector
    // var result = document.querySelector("#show1");
    var result = document.querySelector("#show, #show1");
```



```
// Class Selector
// var result = document.querySelector(
                    ".geek");
var result = document.querySelector("
                    .geek, .myclass");
document.write(result);
console.log(result);
</script>
</body>
</html>
```

- querySelectorAll("CSS Selector") - The method querySelectorAll() returns a static (not live) NodeList representing a list of the document's elements that match the specified group of selectors.
- The NodeList object represents a collection of nodes. The nodes can be accessed by index numbers. The index start at 0.
- You can use the length property of the NodeList object to determine the number of elements that matches the specified selector, then you can loop through all elements and extract the info you want.
- If the specified selectors include a CSS pseudo-element, the returned list is always empty.
- Element Selector -

```
querySelectorAll("p")
querySelectorAll("p, div, h1")
```



Date Page 24

- Class Selector -

```
querySelectorAll(".guk");
querySelectorAll("p.myclass");
querySelectorAll("p>span.myclass")
```

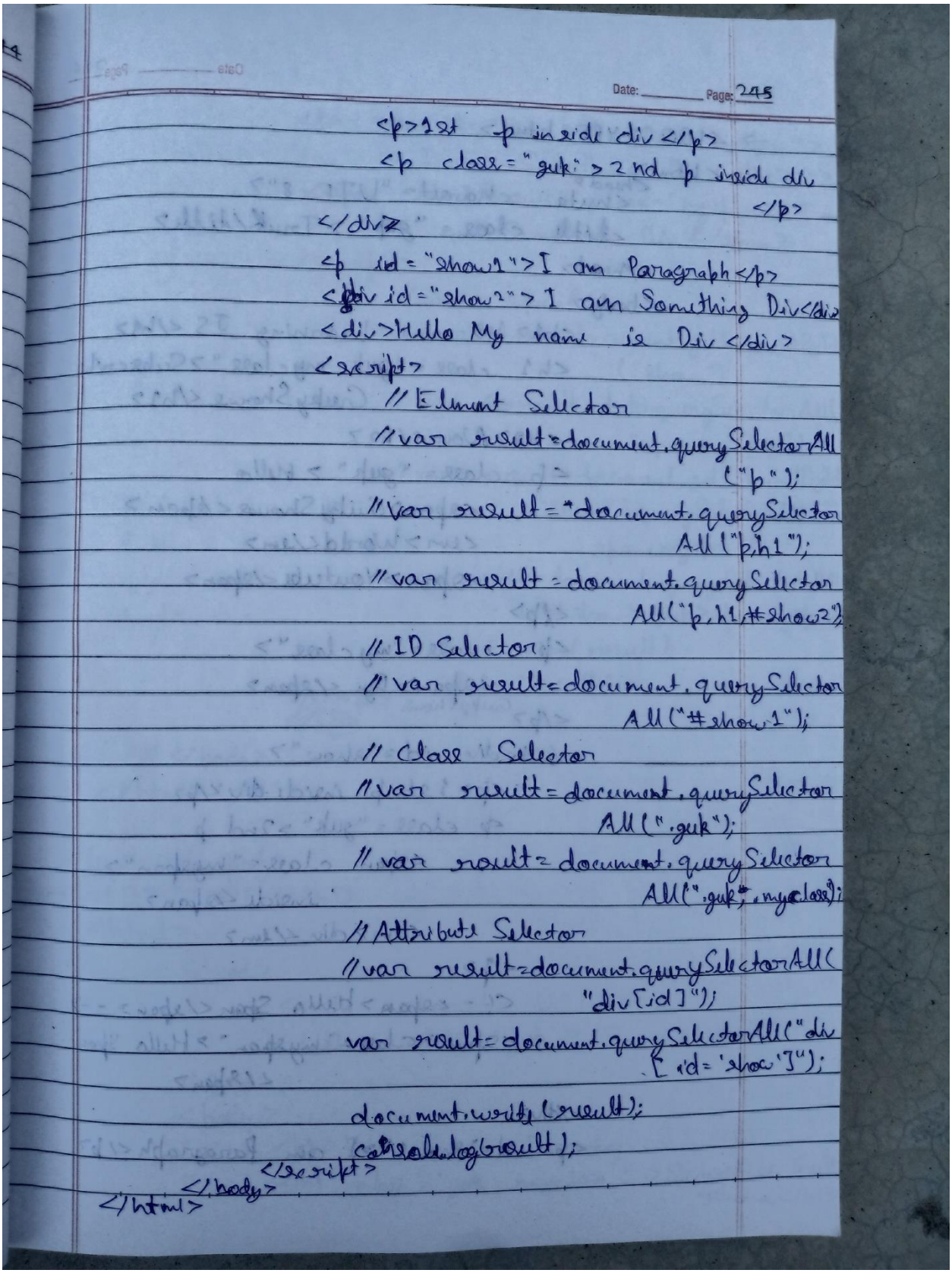
- Attribute Selector -

```
querySelectorAll("div[id]")
querySelectorAll("div[id='guk']")
```

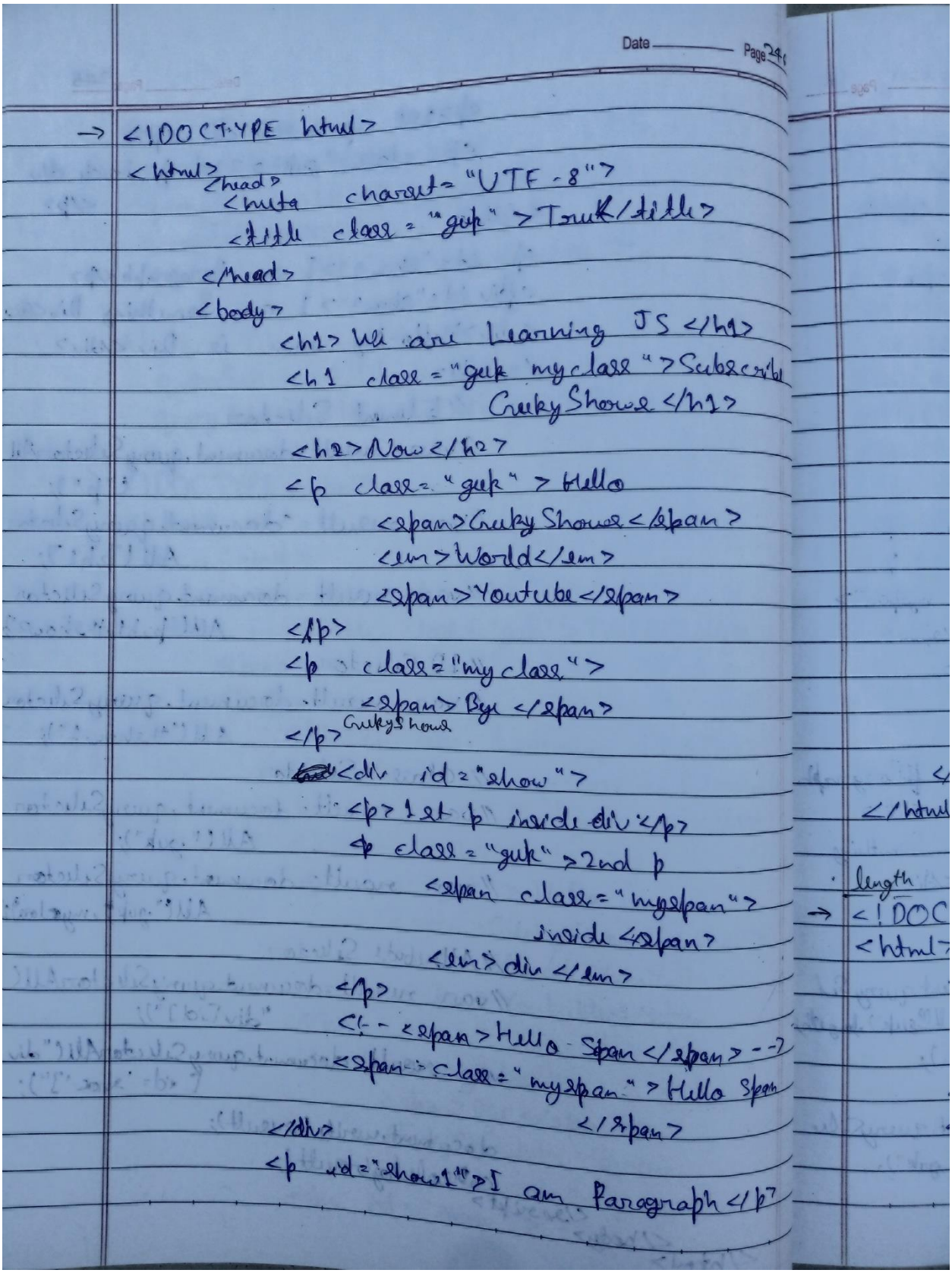
→

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="UTF-8">
        <title class="guk">True</title>
    </head>
    <body>
        <h1>We are Learning JS</h1>
        <h1 class="guk myclass">Subscribe
                                CodeyShows</h1>
        <h2>Now</h2>
        <p class="guk">Hello
            <span>CodeyShows</span>
            <em>World</em>
            <span>Youtube</span>
        </p>
        <p class="myclass">
            <span>Bye</span>
            CodeyShows
        </p>
        <div id="show">
```

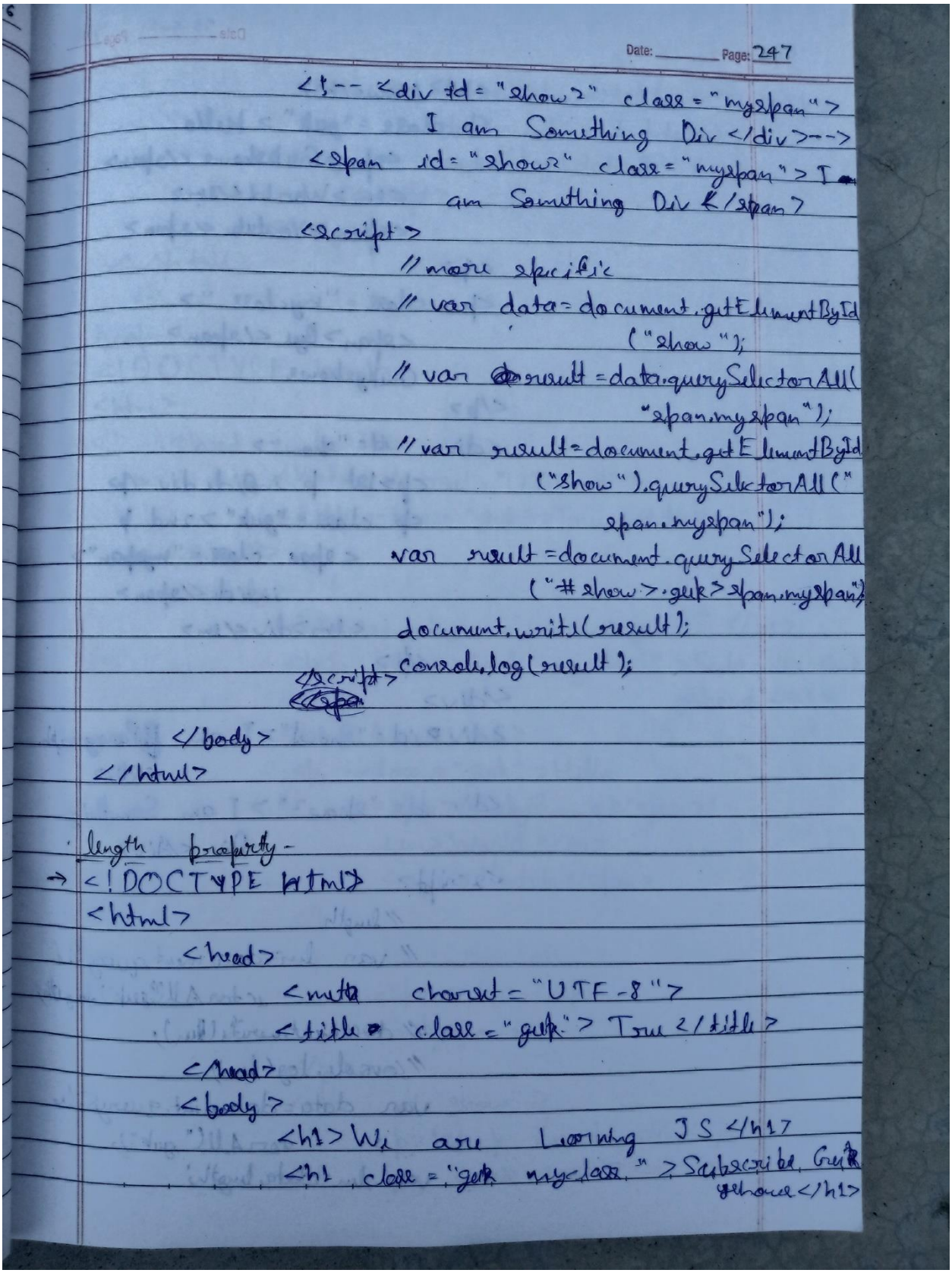
</html>



```
<p>1st p inside div</p>
<p class="guk">2nd p inside div</p>
</div>
<p id="show1">I am Paragraph</p>
<div id="show2">I am Something Div</div>
<div>Hello My name is Div</div>
<script>
// Element Selector
//var result=document.querySelectorAll("p");
//var result="document.querySelectorAll("p,h1");
//var result=document.querySelectorAll("p,h1,#show2");
// ID Selector
//var result=document.querySelectorAll("#show1");
// Class Selector
//var result=document.querySelectorAll(".guk");
//var result=document.querySelectorAll(".guk,.myclass");
// Attribute Selector
//var result=document.querySelectorAll("div[id]");
var result=document.querySelectorAll("div[id='show1']");

document.write(result);
console.log(result);
</script>
</body>
</html>
```



```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
    <title class="geek">Trunk</title>
  </head>
  <body>
    <h1>We are Learning JS</h1>
    <h1 class="geek myclass">Subscribe
        Cuky Shows</h1>
    <h2>Now</h2>
    <p class="geek">Hello
      <span>Cuky Shows</span>
      <em>World</em>
      <span>Youtube</span>
    </p>
    <p class="myclass">
      <span>Bye</span>
    </p>  Cuky's Shows
    <div id="show">
      <p>1st p inside div</p>
      <p class="geek">2nd p
        <span class="myspan">
          inside</span>
        <em>div</em>
      </p>
      <!-- <span>Hello Span</span> -->
      <span class="myspan">Hello Span
        </span>
    </div>
    <p id="show1">I am Paragraph</p>
```

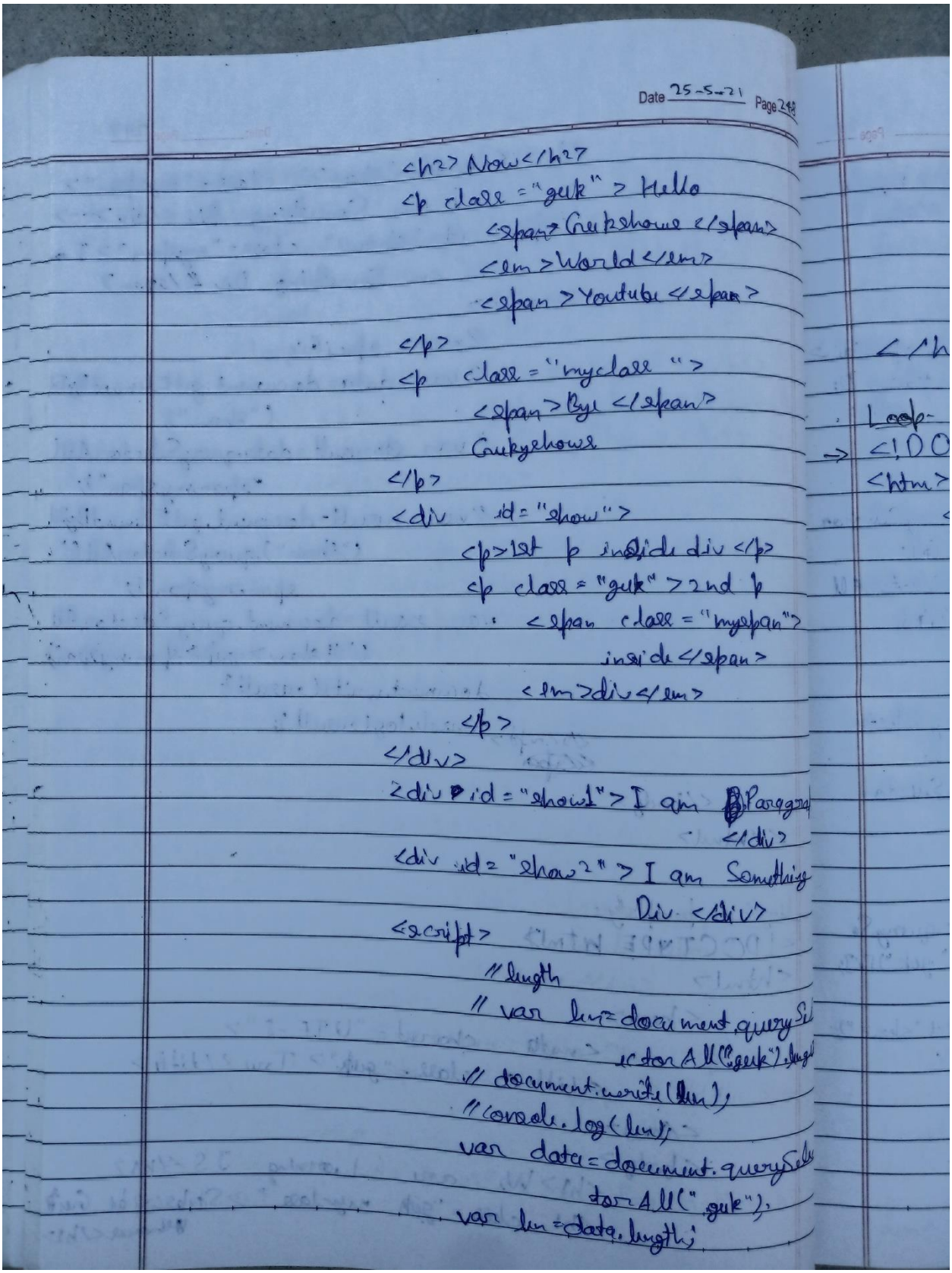


Date: Page: 247

```
<!-- <div id="show2" class="myspan">
        I am Something Div </div>-->
<span id="show2" class="myspan"> I
        am Something Div </span>
<script>
    // more specific
    // var data=document.getElementById
            ("show");
    // var result=data.querySelectorAll(
            "span.myspan");
    // var result=document.getElementById
            ("show").querySelectorAll("
            span.myspan");
    var result=document.querySelectorAll
            ("#show>.geek>span.myspan");
    document.write(result);
    console.log(result);
</script>
</body>
</html>
```

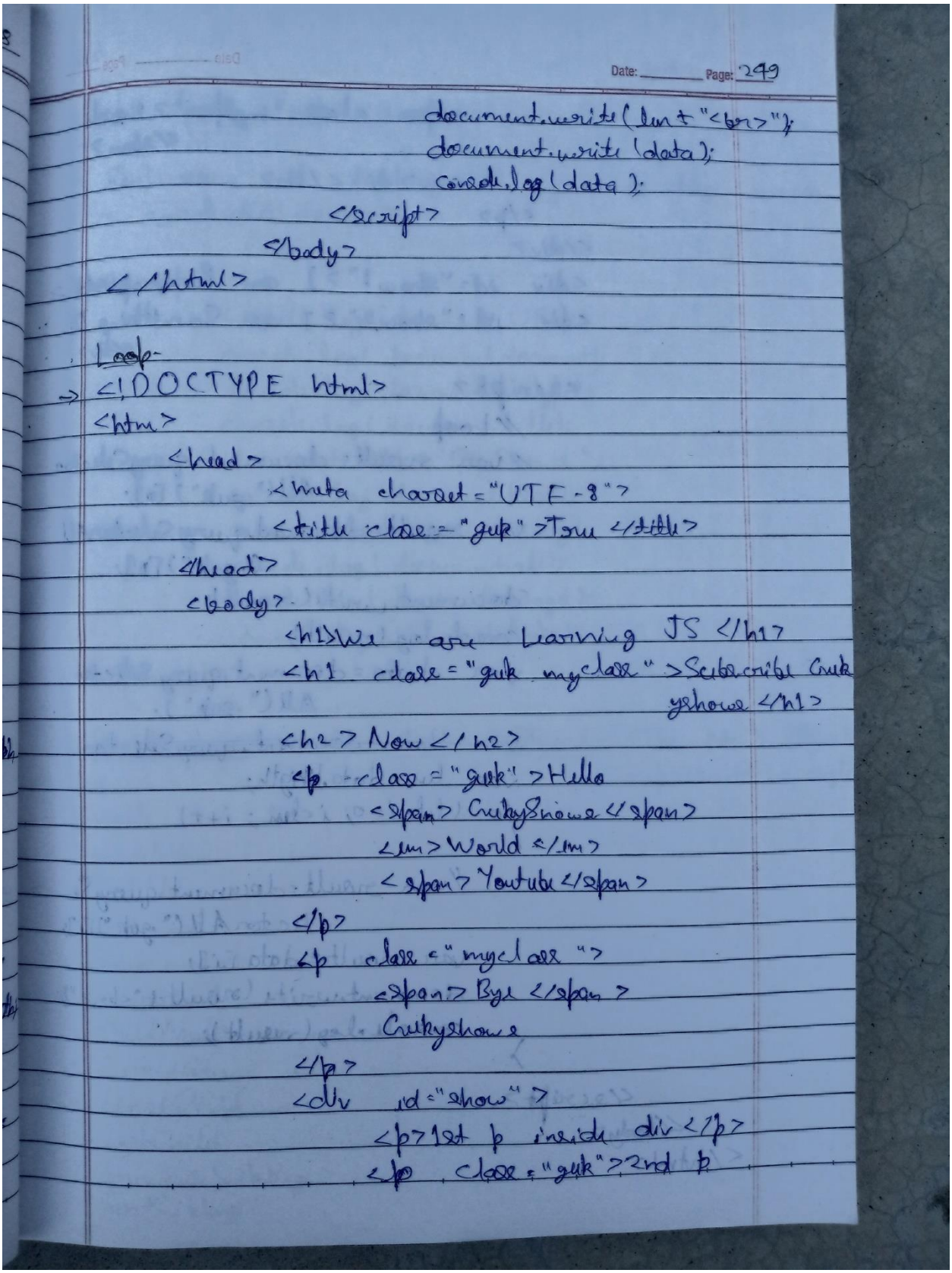
length property -

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="UTF-8">
        <title class="geek"> True </title>
    </head>
    <body>
        <h1> We are Learning JS </h1>
        <h1 class="geek myclass"> Subscribe Geeky
            shows </h1>
```

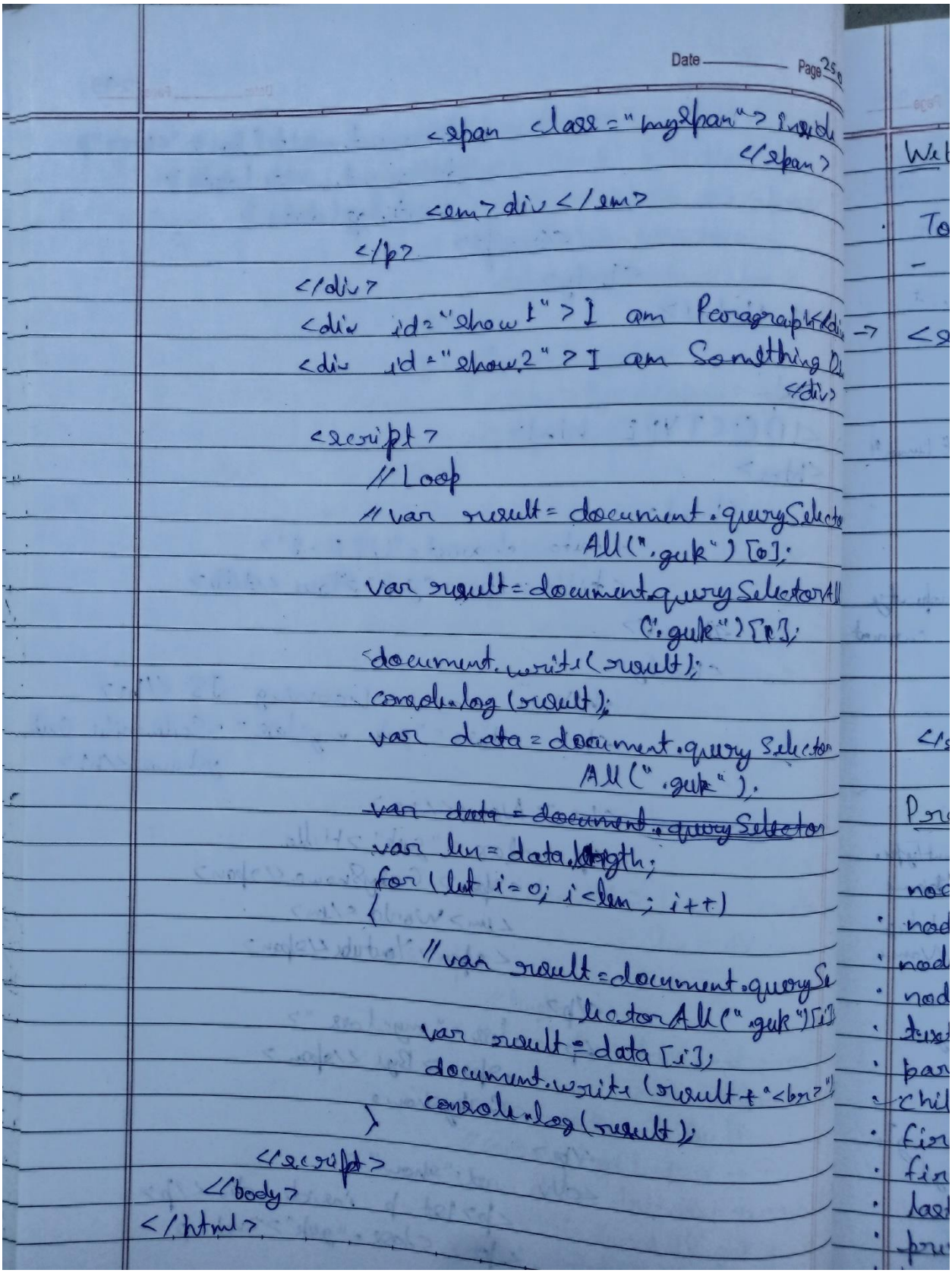


Date 25-5-21 Page 248

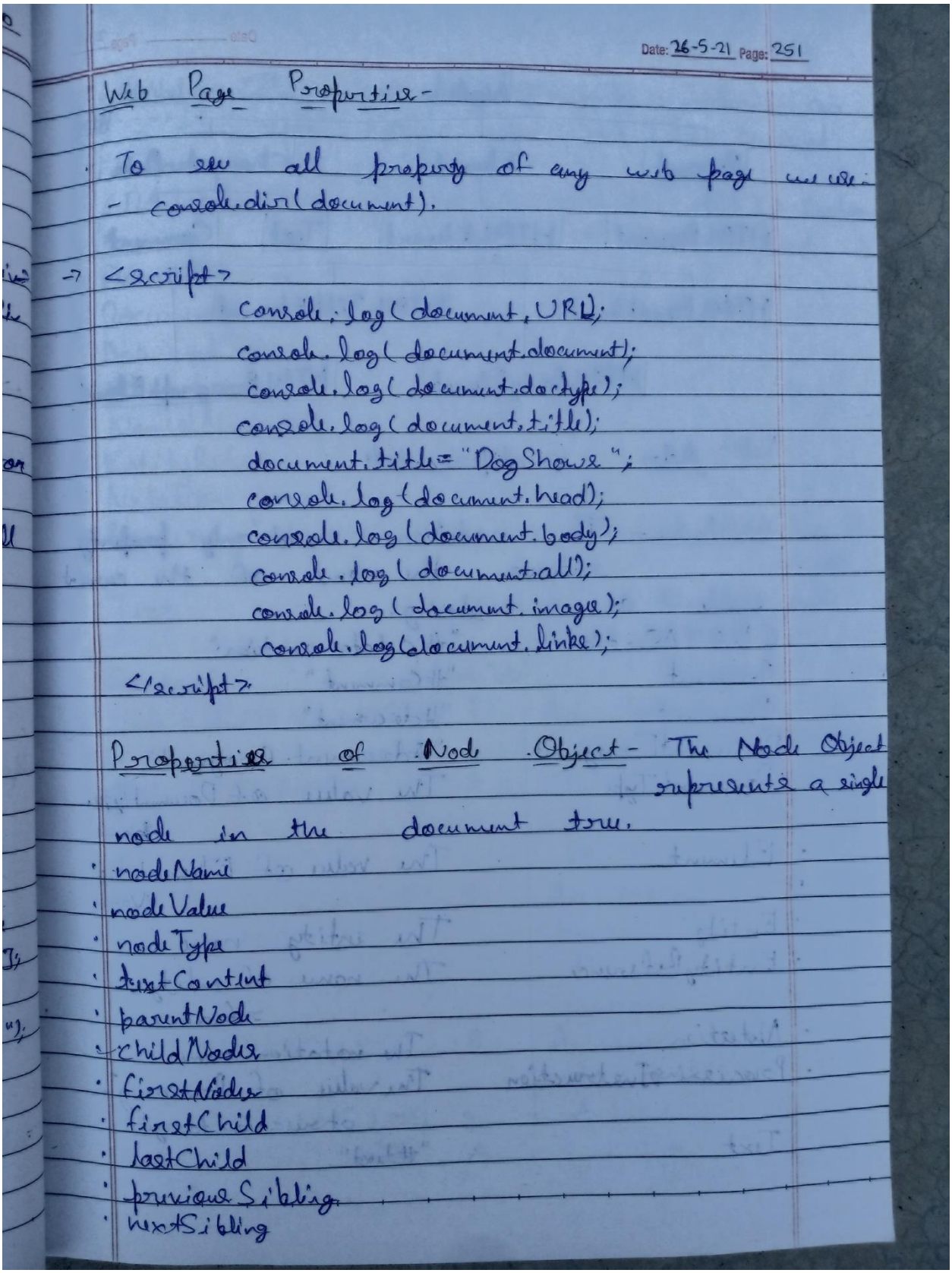
```
<h2> Now </h2>
<p class = "geek" > Hello
    <span> Geekshows </span>
    <em> World </em>
    <span > Youtube </span >
</p>
<p  class = "myclass " >
    <span> Bye </span>
    Geekyshows
</p>
<div  id = "show" >
    <p> 1st p inside div </p>
    <p class = "geek" > 2nd p
      . <span class = "myspan">
            inside </span>
        <em> div </em>
    </p>
</div>
<div id = "show" > I am Paragra
                </div>
<div id = "show2" > I am Something
                Div </div>
<script>
    // length
    // var len = document.querySe
                ctorAll(".geek").leng
    // document.write(len);
    // console.log(len);
    var data = document.querySele
                torAll(".geek");
    var len = data.length;
```



```
                document.write(lan + "<br>");
                document.write(data);
                console.log(data);
            </script>
        </body>
</html>

Loop-
<!DOCTYPE html>
<htm>
    <head>
        <meta charset="UTF-8">
        <title class="guk">Tom</title>
    </head>
    <body>
        <h1>We are Learning JS</h1>
        <h1 class="guk myclass">Subscribe Cuk
                                ykhows</h1>
        <h2>Now</h2>
        <p class="guk">Hello
            <span>CukyShows</span>
            <em>World</em>
            <span>Youtube</span>
        </p>
        <p class="myclass">
            <span>Bye</span>
            Cukyshows
        </p>
        <div id="show">
            <p>1st p inside div</p>
            <p class="guk">2nd p
```



```
<span class="myspan"> I am div
</span>
<em> div </em>
</p>
</div>
<div id="show1"> I am Paragraph</div>
<div id="show2"> I am Something D
</div>

<script>
// Loop
// var result = document.querySelector
                All(".guk")[0];
var result = document.querySelectorAll
                (".guk")[0];
document.write(result);
console.log(result);
var data = document.querySelector
                All(".guk");
var data = document.querySelector
var len = data.length;
for (let i=0; i<len; i++)
{
    // var result = document.querySe
                lectorAll(".guk")[i]
    var result = data[i];
    document.write(result + "<br>");
    console.log(result);
}
</script>
</body>
</html>
```



Date: 26-5-21 Page: 251

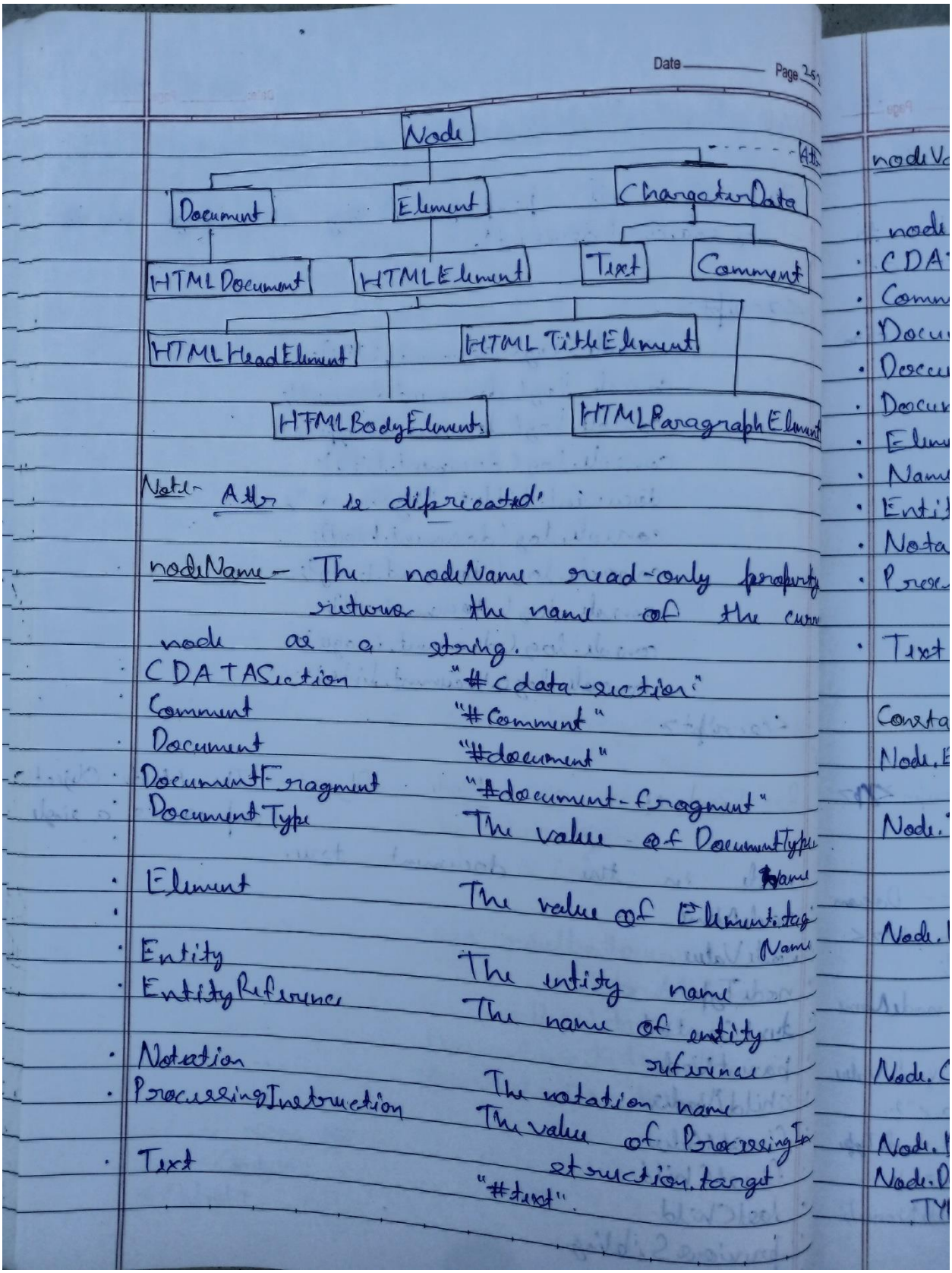
Web Page Properties -

To see all property of any web page we use:
- console.dir(document).

→
```
<script>
    console.log(document.URL);
    console.log(document.document);
    console.log(document.doctype);
    console.log(document.title);
    document.title = "Dog Shows";
    console.log(document.head);
    console.log(document.body);
    console.log(document.all);
    console.log(document.image);
    console.log(document.links);
</script>
```

Properties of Node Object - The Node Object represents a single node in the document tree.

- nodeName
- nodeValue
- nodeType
- textContent
- parentNode
- childNodes
- firstNodes
- firstChild
- lastChild
- previousSibling
- nextSibling



Note- Attr is dipricated.

nodeName - The nodeName read-only property returns the name of the current node as a string.

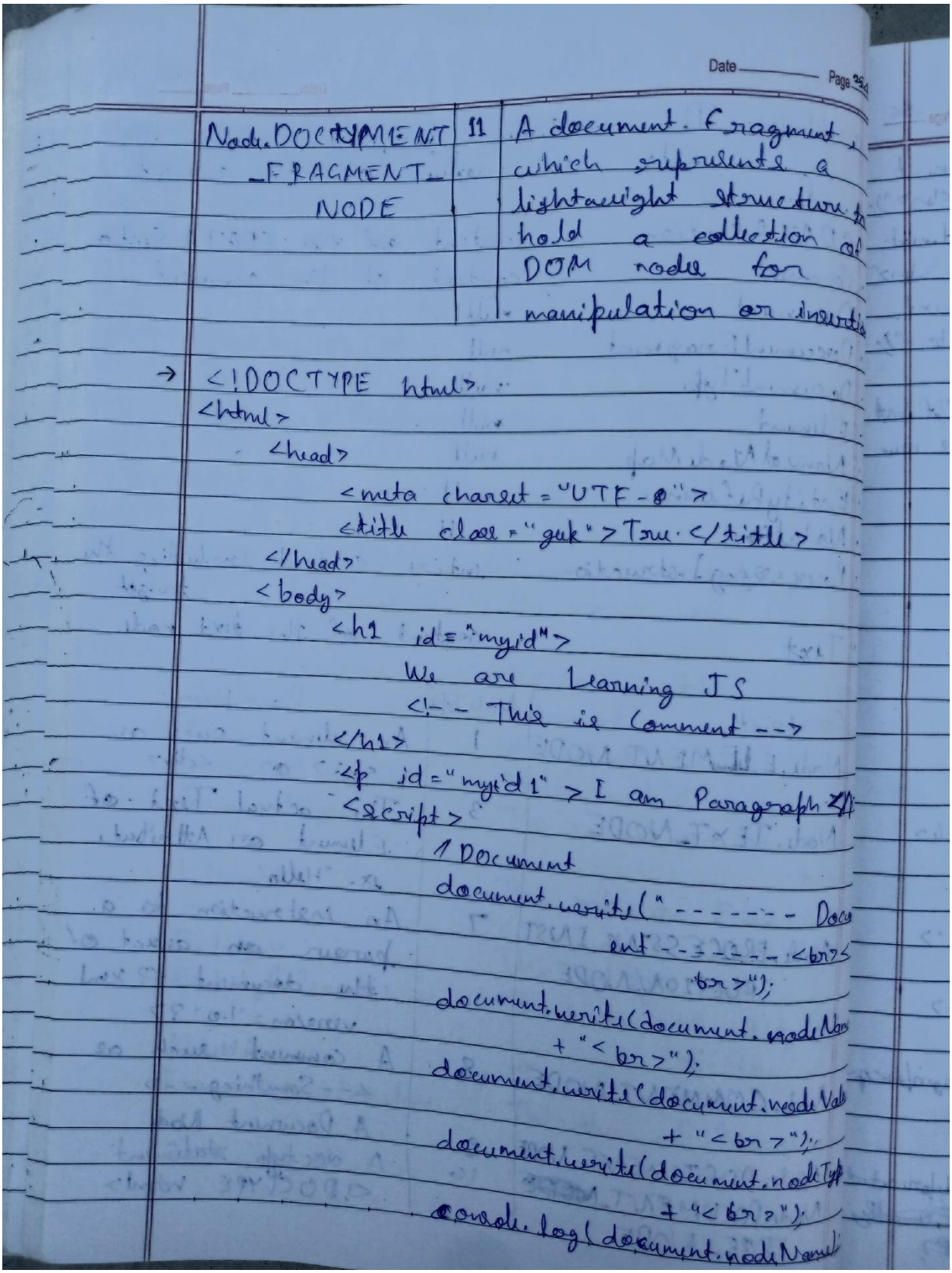
CDATASection	"#cdata-section"
Comment	"#comment"
Document	"#document"
DocumentFragment	"#document-fragment"
DocumentType	The value of DocumentType.name
Element	The value of Element.tagName
Entity	The entity name
EntityReference	The name of entity reference
Notation	The notation name
ProcessingInstruction	The value of ProcessingInstruction.target
Text	"#text"

Date: Page: 253

nodeValue - The nodeValue property returns or sets the value of the current node.

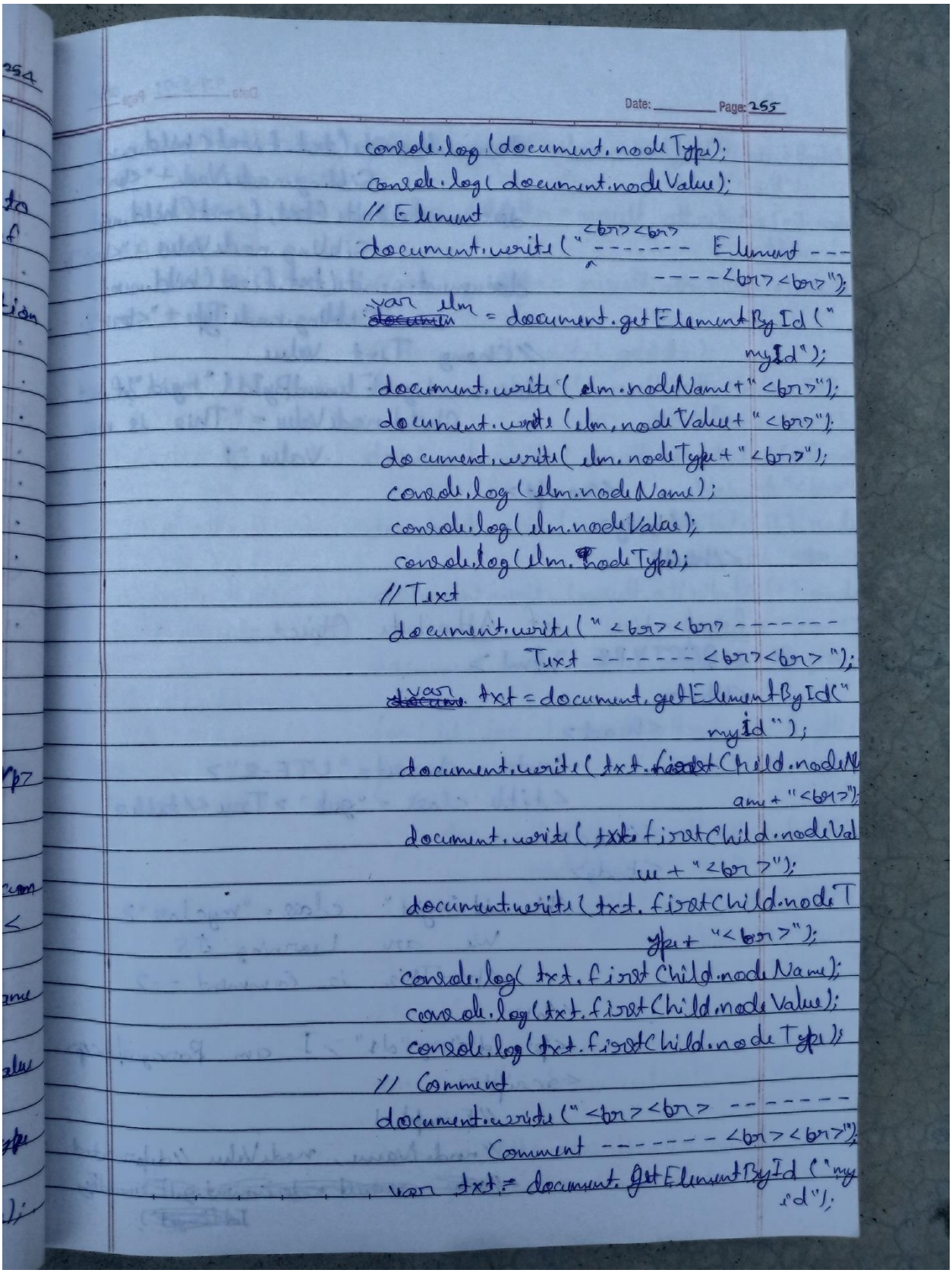
- CDATASection — content of the CDATA Section
- Comment — content of the comment
- Document — null
- DocumentFragment — null
- DocumentType — null
- Element — null
- NamedNodeMap — null
- EntityReference — null
- Notation — null
- ProcessingInstruction — entire content including the target
- Text — content of the text node.

Constant	Node Type	Discription
Node.ELEMENT_NODE	1	An element such as <h1> or
Node.TEXT_NODE	3	The actual Text of Element or Attribute ex- "Hello"
Node.PROCESSING_INSTRUCTION_NODE	7	An instruction to a parser on aspect of the document <? xml version = "1.0" ?>
Node.COMMENT_NODE	8	A comment such as <-- Something -->
Node.DOCUMENT_NODE	9	A Document Node
Node.DOCUMENT_TYPE_NODE	10	A doctype statement <!DOCTYPE html>

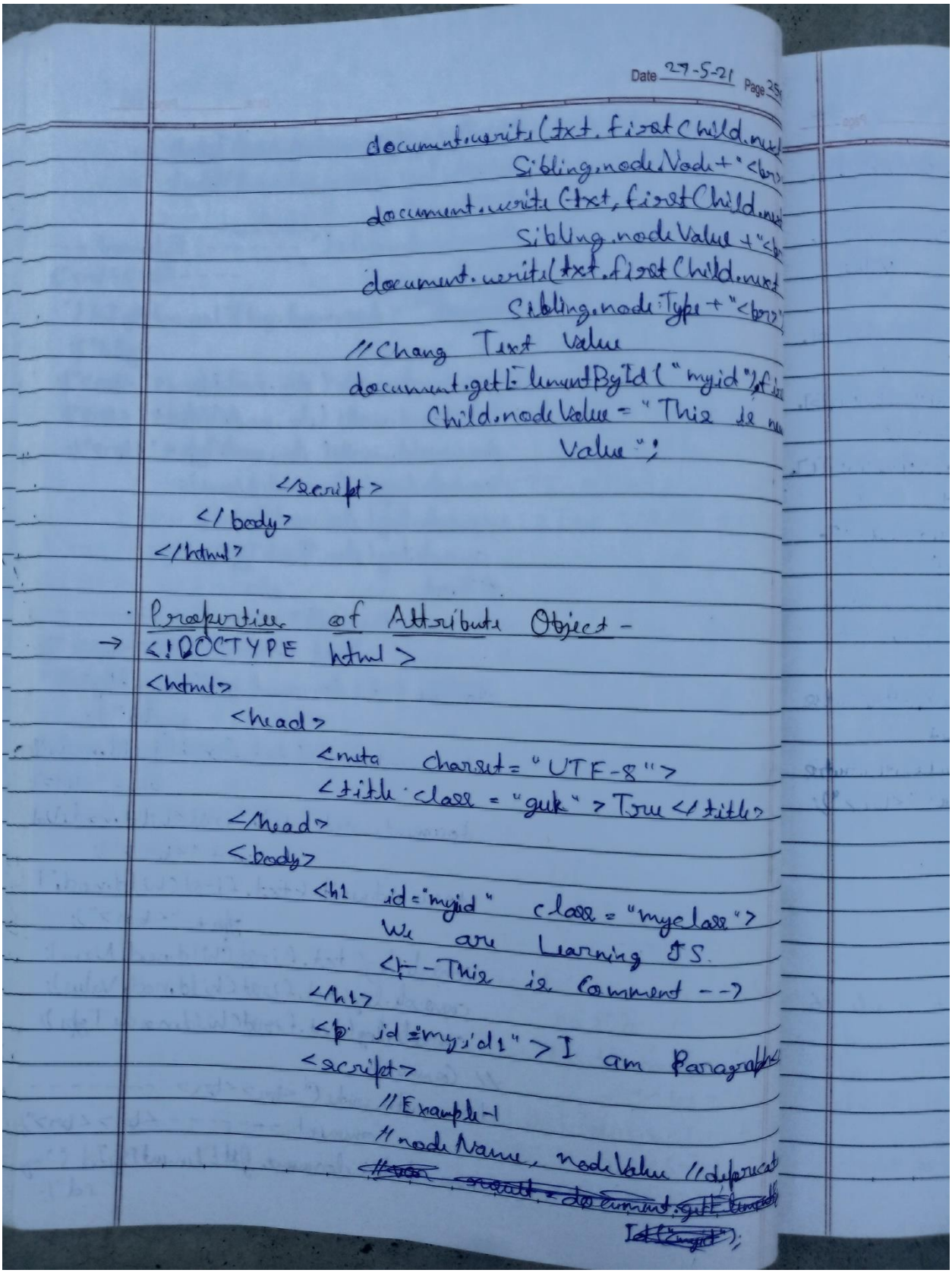


Node.DOCUMENT_FRAGMENT_NODE	11	A document fragment, which represents a lightweight structure to hold a collection of DOM nodes for manipulation or insertion

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset = "UTF-8">
        <title class = "guk"> True. </title>
    </head>
    <body>
        <h1 id = "myid">
            We are Learning JS
            <!-- This is Comment -->
        </h1>
        <p id = "myid1" > I am Paragraph </p>
        <script>
            // Document
            document.write("------- Document ------- <br><br>");
            document.write(document.nodeName + "<br>");
            document.write(document.nodeValue + "<br>");
            document.write(document.nodeType + "<br>");
            console.log(document.nodeName);
```



```
console.log(document.nodeType);
console.log(document.nodeValue);
// Element
document.write("<br><br> ------- Element ---
    ----<br><br>");
var elm = document.getElementById("
    myId");
document.write(elm.nodeName+"<br>");
document.write(elm.nodeValue+"<br>");
document.write(elm.nodeType+"<br>");
console.log(elm.nodeName);
console.log(elm.nodeValue);
console.log(elm.nodeType);
// Text
document.write("<br><br> -------
    Text -------<br><br>");
var txt = document.getElementById("
    myid");
document.write(txt.firstChild.nodeN
    ame+"<br>");
document.write(txt.firstChild.nodeVa
    ue+"<br>");
document.write(txt.firstChild.nodeT
    ype+"<br>");
console.log(txt.firstChild.nodeName);
console.log(txt.firstChild.nodeValue);
console.log(txt.firstChild.nodeType);
// Comment
document.write("<br><br> ------
    Comment -------<br><br>");
var txt = document.getElementById("my
    id");
```



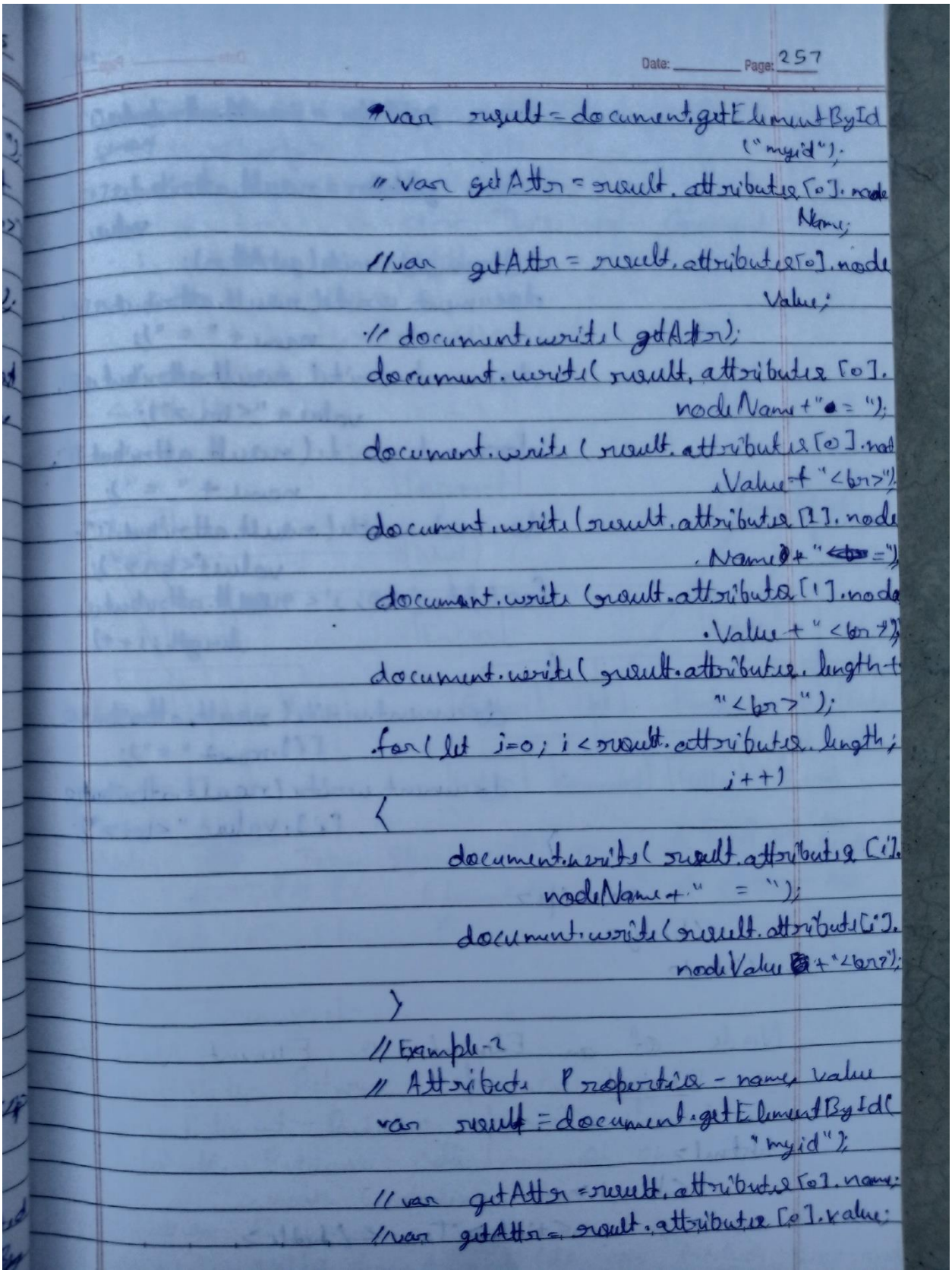
Date 27-5-21 Page 25

```
document.write(txt.firstChild.nextSibling.nodeName+"<br>");
document.write(txt.firstChild.nextSibling.nodeValue+"<br>");
document.write(txt.firstChild.nextSibling.nodeType+"<br>");
//Chang Text Value
document.getElementById("myid").firstChild.nodeValue="This is new Value";
</script>
</body>
</html>
```

Properties of Attribute Object -

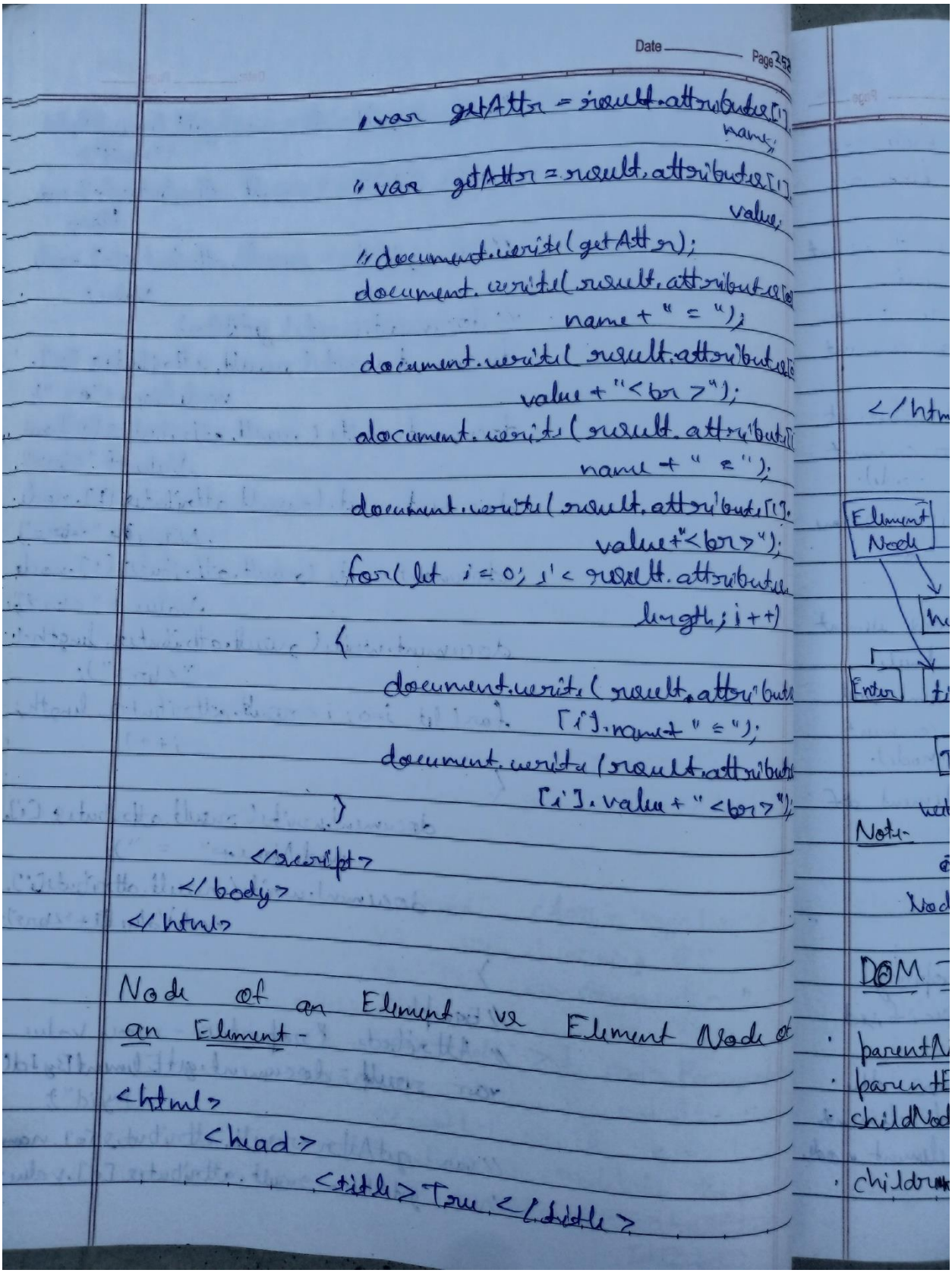
→

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="UTF-8">
        <title class="guk">True</title>
    </head>
    <body>
        <h1 id="myid" class="myclass">
            We are Learning JS.
            <!-- This is Comment -->
        </h1>
        <p id="myid1">I am Paragraph</p>
        <script>
            //Example-1
            //nodeName, nodeValue //deprecat
            var txt=document.getElementById("myid");
```



Date: _____ Page: 257

```
var result = document.getElementById
("myid");
// var getAttr = result.attributes[0].node
Name;
//var getAttr = result.attributes[0].node
Value;
// document.write( getAttr);
document.write( result.attributes [0].
nodeName + "= ");
document.write ( result.attributes [0].nod
Value + "<br>");
document.write( result.attributes [1].node
.Name + "=");
document.write (result.attributes [1].node
.Value + "<br>");
document.write( result.attributes.length +
"<br>");
for( let i=0; i < result.attributes.length;
i++)
{
    document.write( result.attributes [i].
    nodeName + " = ");
    document.write( result.attributes [i].
    nodeValue + "<br>");
}
// Example-2
// Attribute Properties - name, value
var result = document.getElementById(
"myid");
// var getAttr = result.attributes [0].name;
//var getAttr = result.attributes [0].value;
```



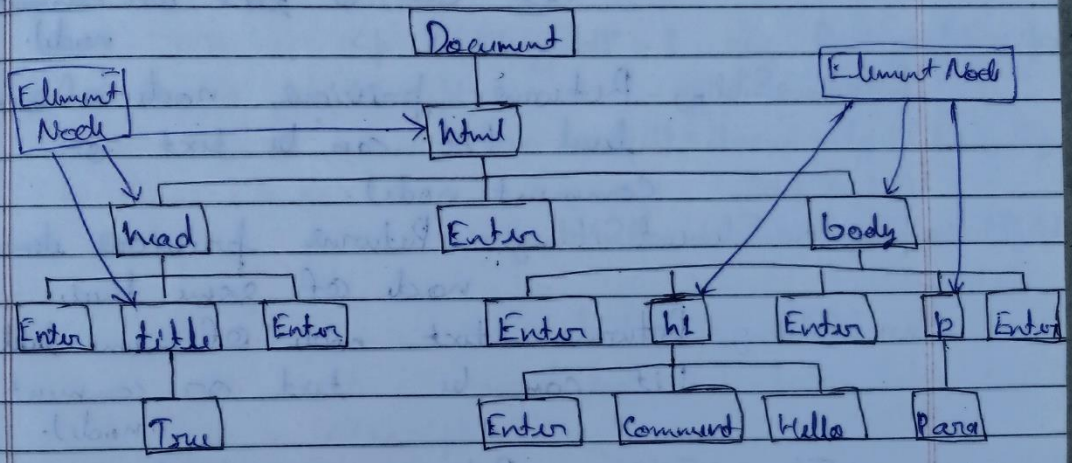
```
// var getAttr = result.attributes[1].
                                    name;
// var getAttr = result.attributes[1].
                                    value;
// document.write(getAttr);
document.write(result.attributes[0].
                    name + " = ");
document.write(result.attributes[0].
                    value + "<br>");
document.write(result.attributes[1].
                    name + " = ");
document.write(result.attributes[1].
                    value + "<br>");
for(let i = 0; i < result.attributes.
                    length; i++)
{
    document.write(result.attributes
        [i].name + " = ");
    document.write(result.attributes
        [i].value + "<br>");
}
</script>
</body>
</html>
```

Node of an Element vs Element Node of an Element -

```
<html>
    <head>
        <title> True </title>
```

Date: Page: 259

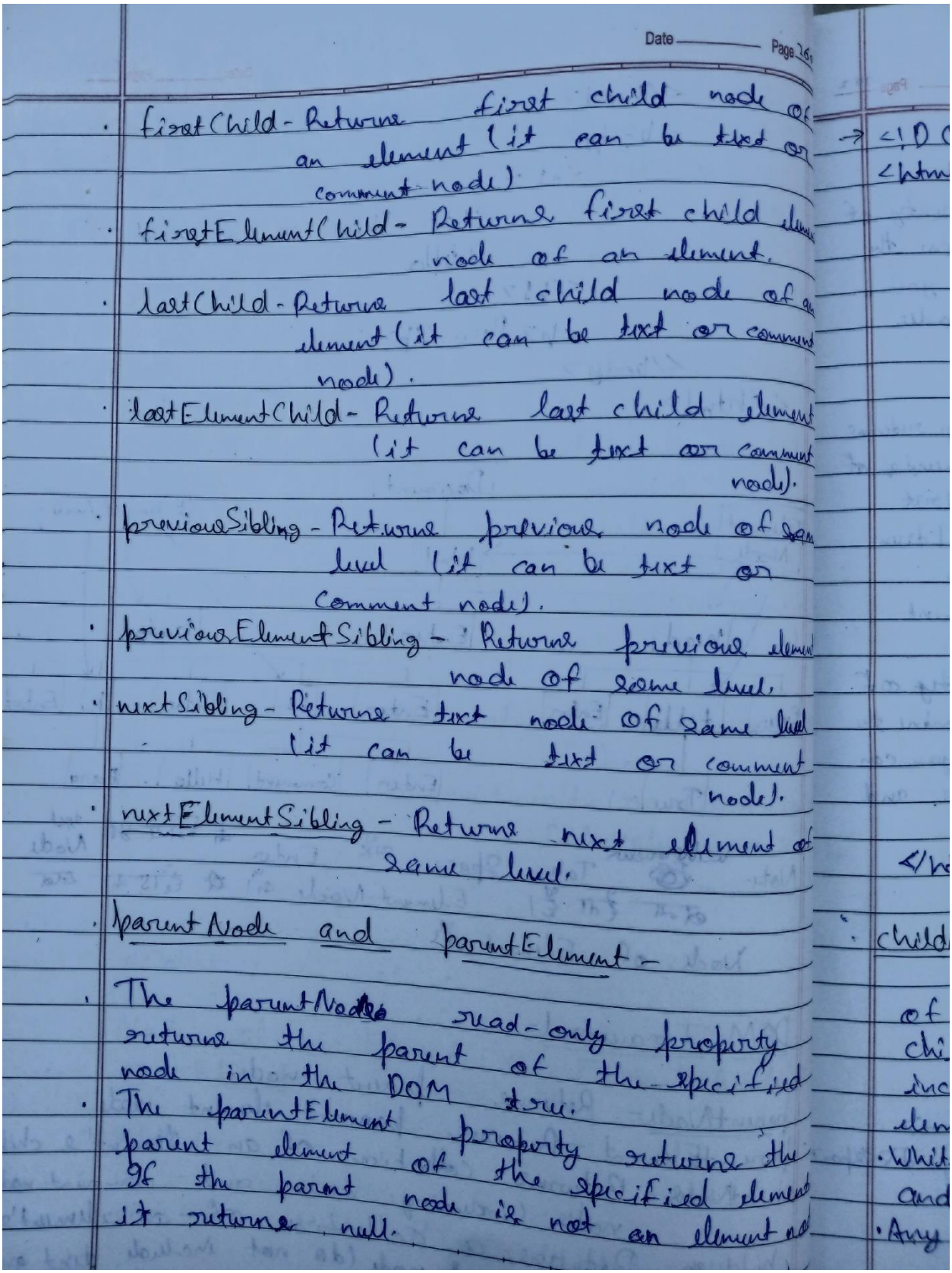
```
</head>
<body>
    <h1>
        <!-- This is Comment -->
        Hello
    </h1>
    <p> Para </p>
</body>
</html>
```



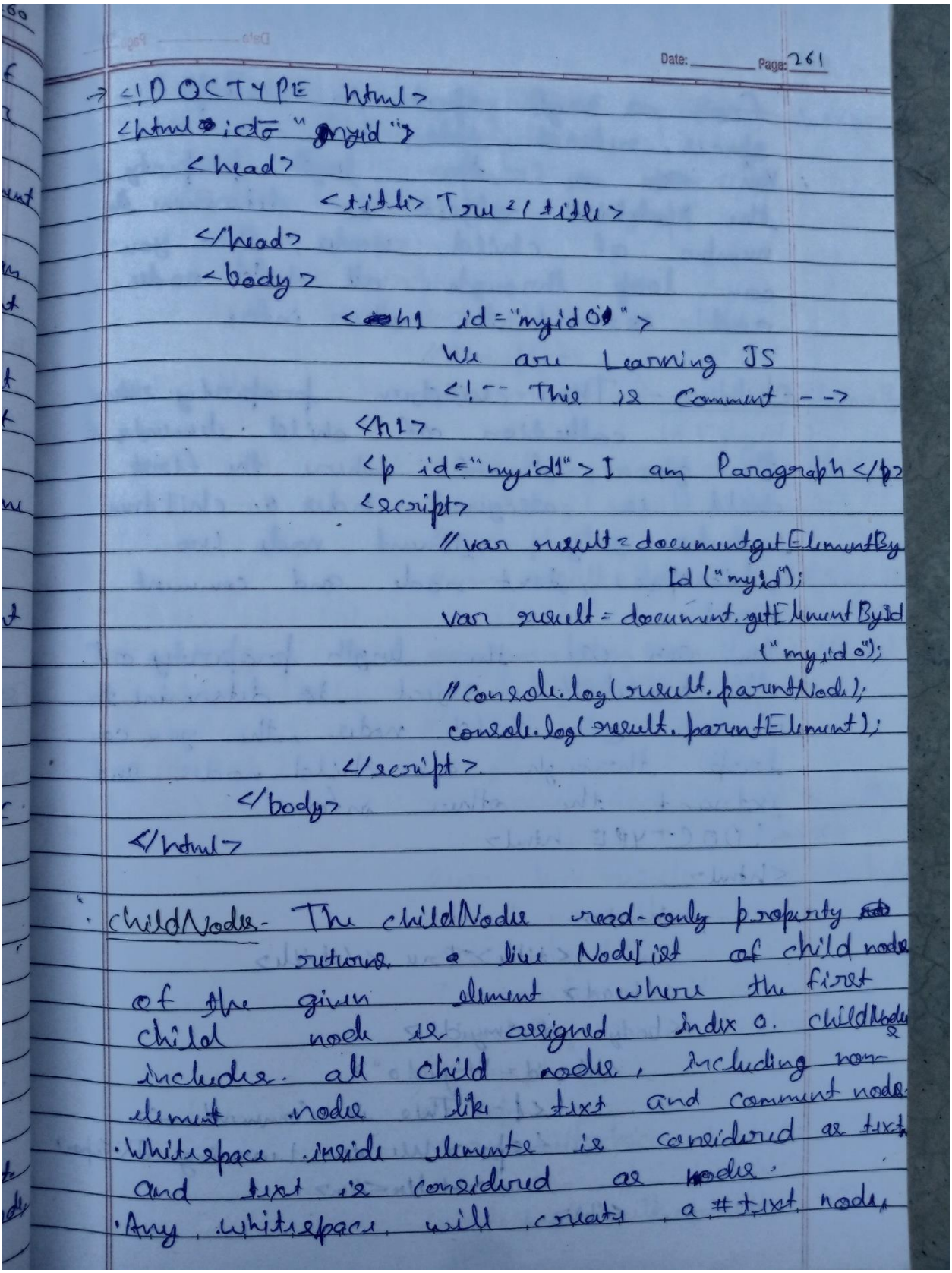
Note- Tab, Space और Enter के लिये भी Text Node बना देता है। Element Node को छोड़कर सब Node of Element है।

DOM Traversal-

- parentNode - Returns parent node.
- parentElement - Returns parent element node
- childNodes - Returns collection of an element's child node (including text and comment nodes)
- children - Returns a collection of an element's child element node (do not include text and comment nodes)



- firstChild - Returns first child node of an element (it can be text or comment node)
- firstElementChild - Returns first child element node of an element.
- lastChild - Returns last child node of an element (it can be text or comment node).
- lastElementChild - Returns last child element (it can be text or comment node).
- previousSibling - Returns previous node of same level (it can be text or comment node).
- previousElementSibling - Returns previous element node of same level.
- nextSibling - Returns next node of same level (it can be text or comment node).
- nextElementSibling - Returns next element of same level.
- parentNode and parentElement -
- The parentNode read-only property returns the parent of the specified node in the DOM tree.
- The parentElement property returns the parent element of the specified element if the parent node is not an element node it returns null.

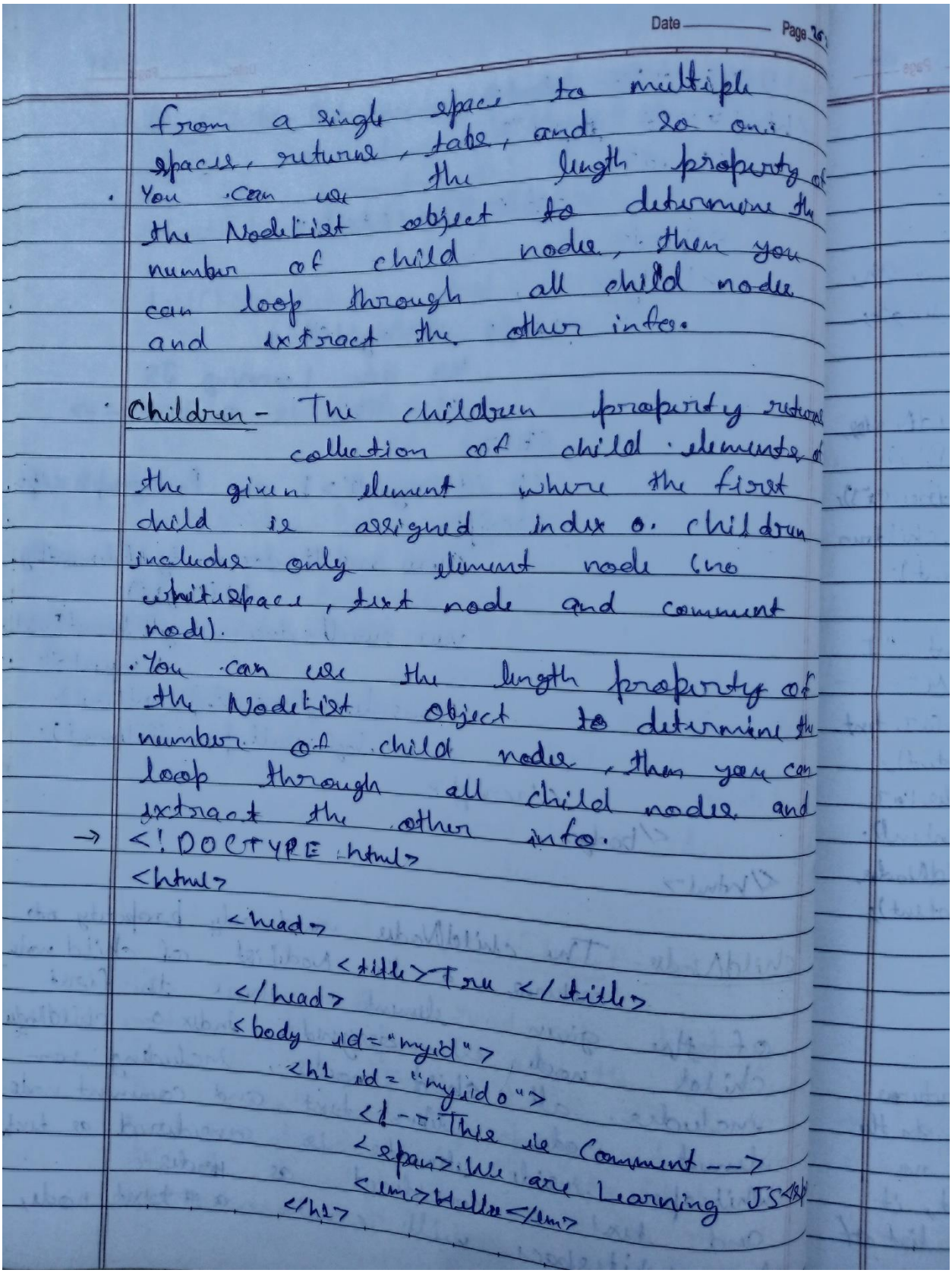


Date: Page: 261

```
<!DOCTYPE html>
<html id="myid">
    <head>
        <title> True </title>
    </head>
    <body>
        <h1 id="myid0">
            We are Learning JS
            <!-- This is Comment -->
        </h1>
        <p id="myid1"> I am Paragraph </p>
        <script>
            // var result = document.getElementBy
                                Id("myid");
            var result = document.getElementById
                                ("myid0");
            // console.log(result.parentNode);
            console.log(result.parentElement);
        </script>
    </body>
</html>
```

childNodes - The childNodes read-only property returns a live NodeList of child nodes of the given element where the first child node is assigned index 0. childNodes includes all child nodes, including non-element nodes like text and comment nodes. Whitespace inside elements is considered as text and text is considered as nodes.

Any whitespace will create a #text node.



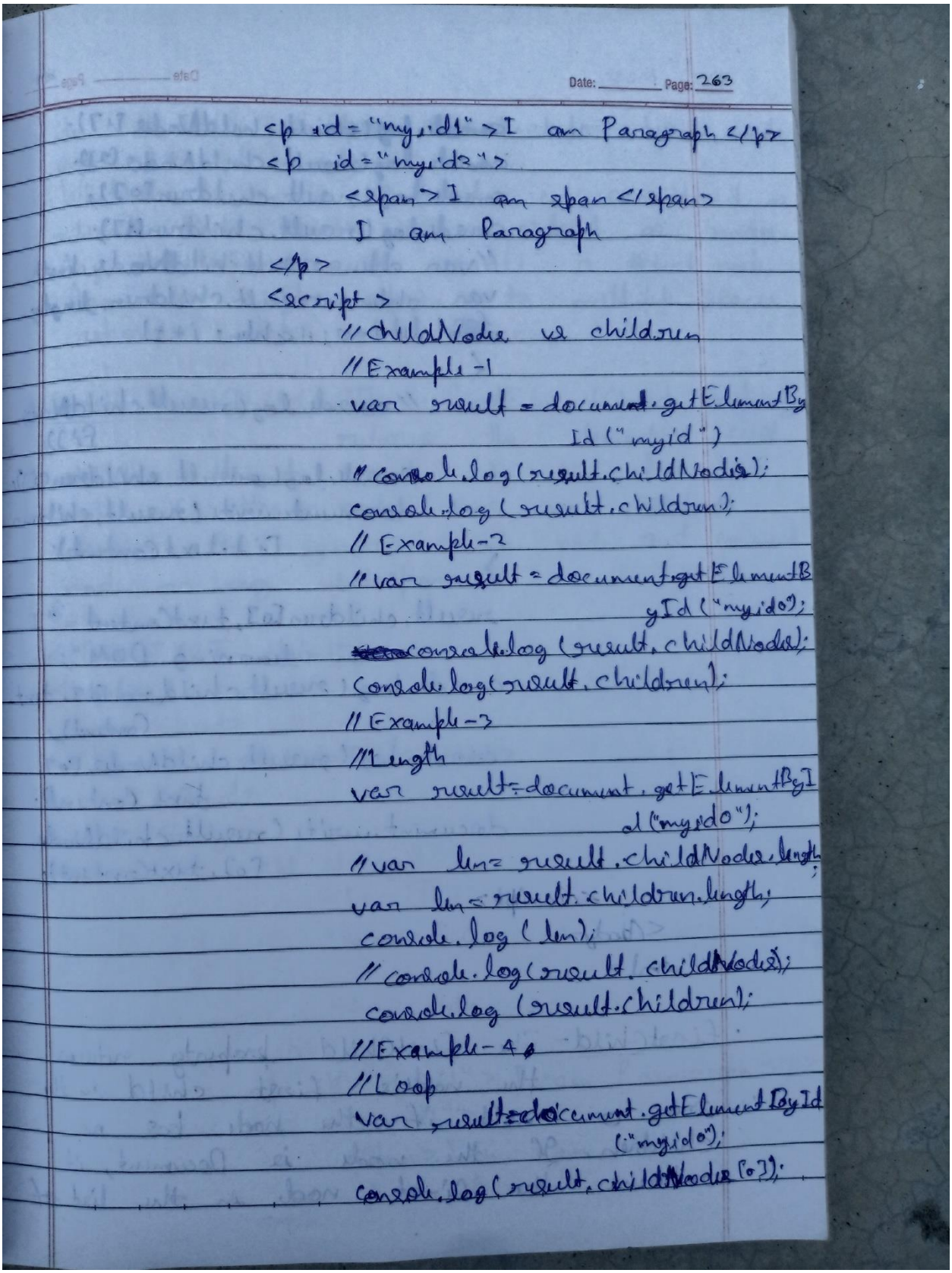
from a single space to multiple spaces, returns, tabs, and so on.

- You can use the length property of the NodeList object to determine the number of child nodes, then you can loop through all child nodes and extract the other info.

- Children - The children property returns a collection of child elements of the given element where the first child is assigned index 0. children includes only element nodes (no whitespace, text node and comment node).

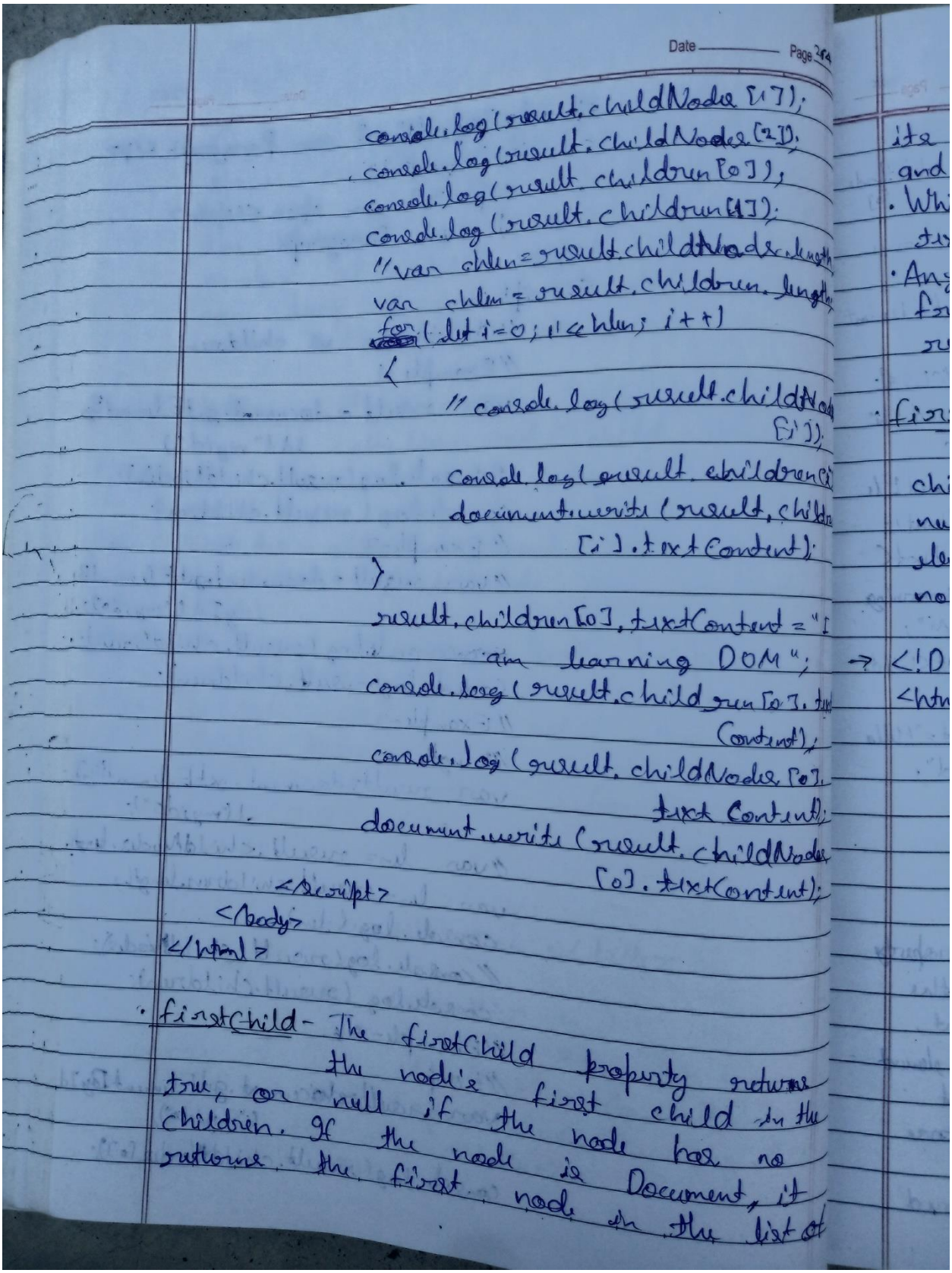
- You can use the length property of the NodeList object to determine the number of child nodes, then you can loop through all child nodes and extract the other info.

→
```
<!DOCTYPE html>
<html>
    <head>
        <title>True</title>
    </head>
    <body id="myid">
        <h1 id="myido">
            <!-- This is Comment -->
            <span>We are Learning JS</span>
            <em>Hello</em>
        </h1>
```



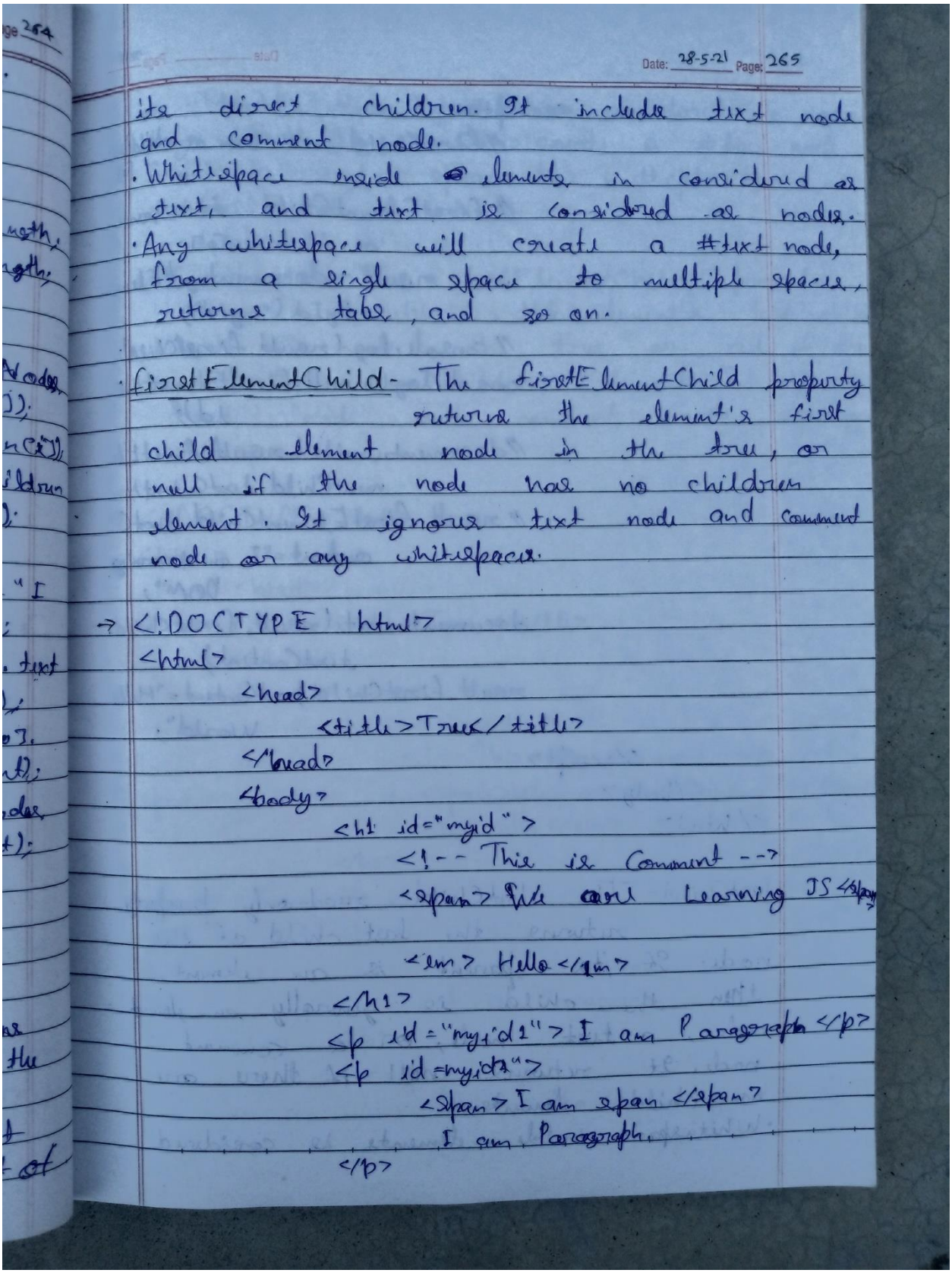
Date: Page: 263

```
<p id="myid1">I am Paragraph</p>
<p id="myid2">
    <span>I am span</span>
    I am Paragraph
</p>
<script>
    // childNodes vs children
    // Example-1
    var result = document.getElementBy
                 Id("myid")
    // console.log(result.childNodes);
    console.log(result.children);
    // Example-2
    // var result = document.getElementB
                 yId("myid");
    console.log(result.childNodes);
    console.log(result.children);
    // Example-3
    // Length
    var result = document.getElementByI
                 d("myid");
    // var len = result.childNodes.length;
    var len = result.children.length;
    console.log(len);
    // console.log(result.childNodes);
    console.log(result.children);
    // Example-4
    // Loop
    var result = document.getElementById
                 ("myid");
    console.log(result.childNodes[0]);
```



```
console.log(result.childNodes[1]);
console.log(result.childNodes[2]);
console.log(result.children[0]);
console.log(result.children[1]);
// var chlen = result.childNodes.length;
var chlen = result.children.length;
for(let i=0; i<chlen; i++)
{
    // console.log(result.childNodes[i]);
    console.log(result.children[i]);
    document.write(result.children[i].textContent);
}
result.children[0].textContent = "I am learning DOM";
console.log(result.children[0].textContent);
console.log(result.childNodes[0].textContent);
document.write(result.childNodes[0].textContent);
</script>
</body>
</html>
```

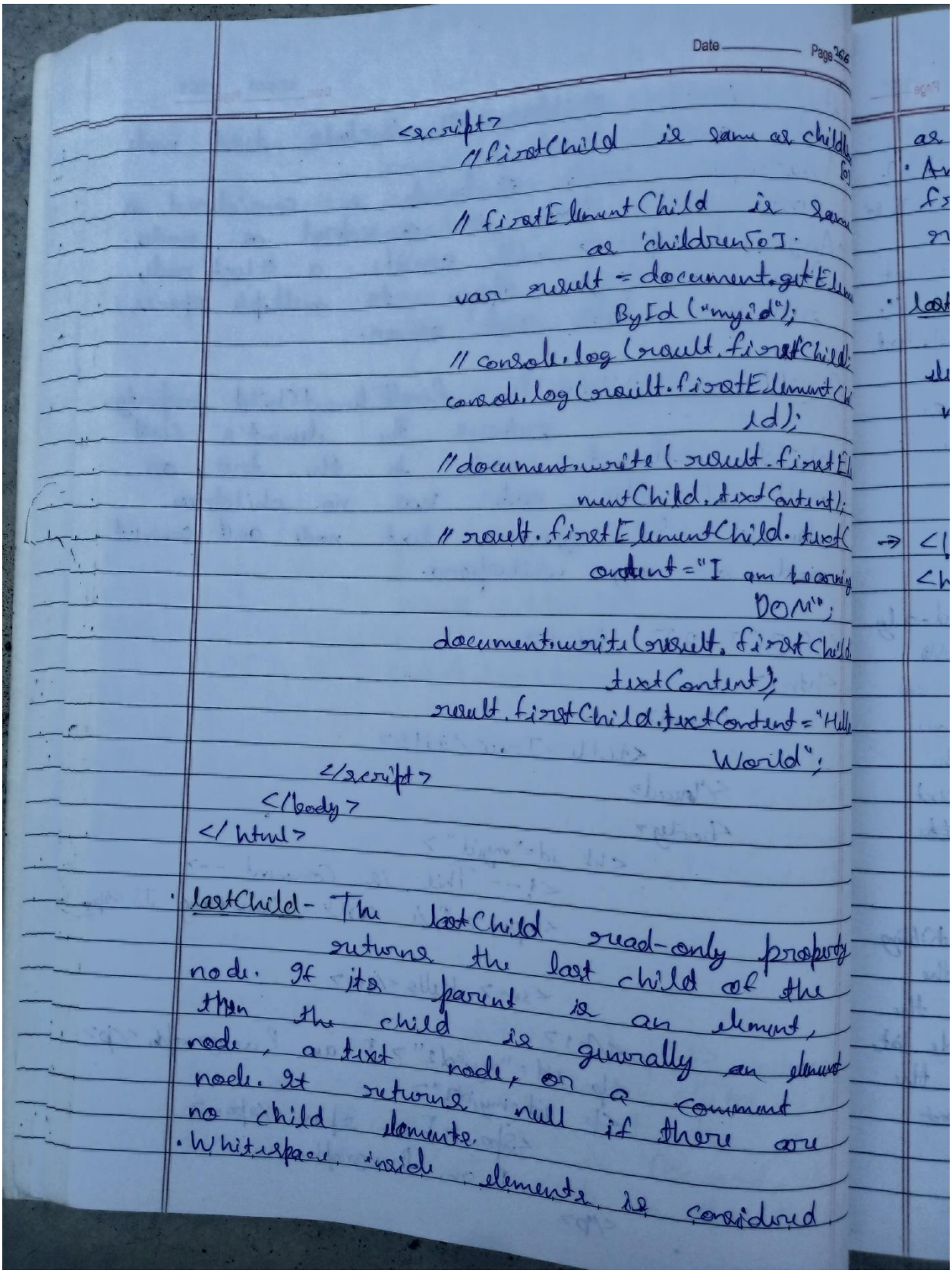
- **firstChild** - The firstChild property returns the node's first child in the tree, or null if the node has no children. If the node is Document, it returns the first node in the list of



its direct children. It includes text node and comment node.
- Whitespace inside elements is considered as text, and text is considered as nodes.
- Any whitespace will create a #text node, from a single space to multiple spaces, returns tabs, and so on.

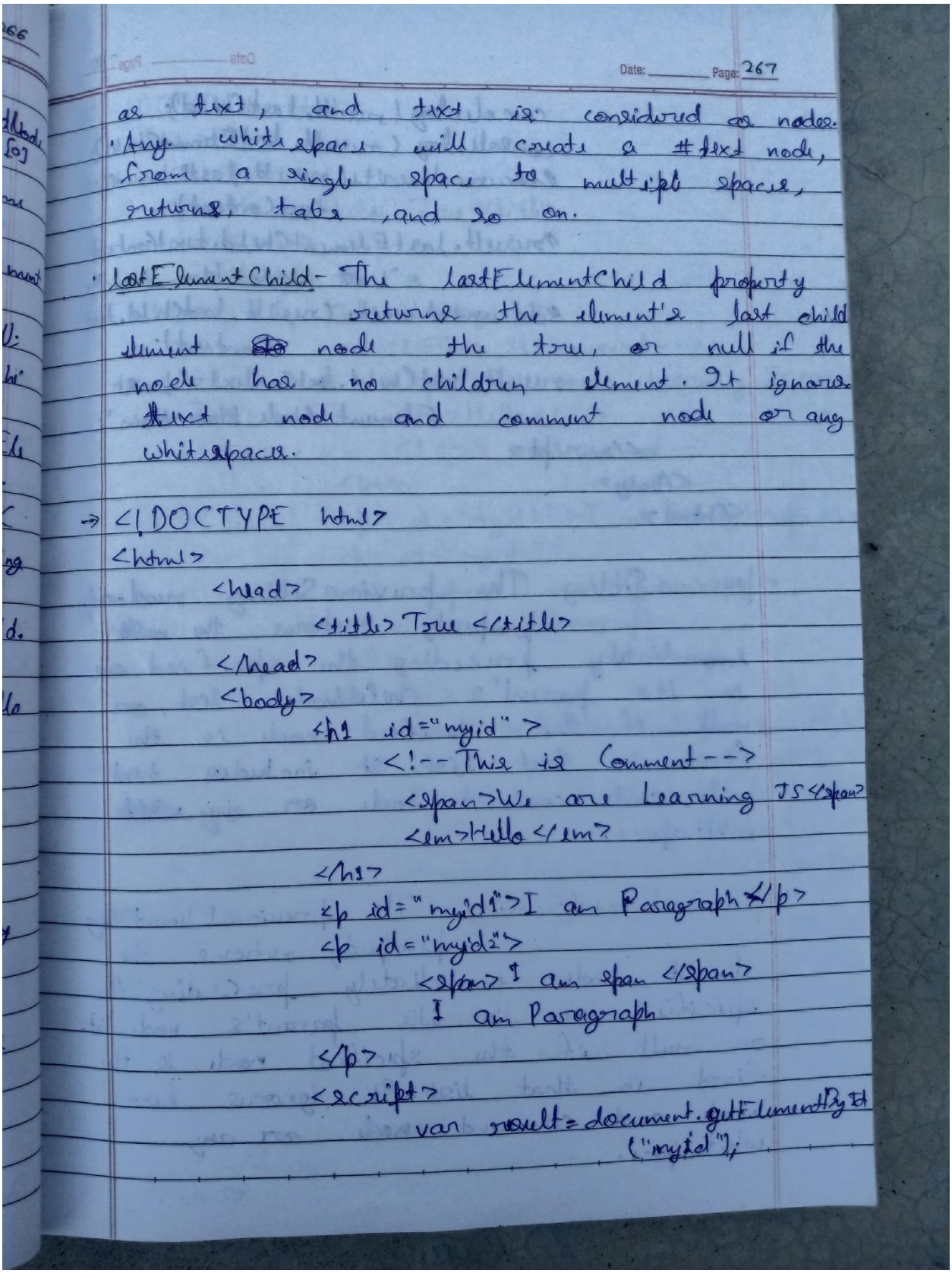
firstElementChild - The firstElementChild property returns the element's first child element node in the tree, or null if the node has no children element. It ignores text node and comment node or any whitespace.

→
```
<!DOCTYPE html>
<html>
    <head>
        <title>Track</title>
    </head>
    <body>
        <h1 id="myid">
            <!-- This is Comment -->
            <span> We are Learning JS </span>
            <em> Hello </em>
        </h1>
        <p id="myid1"> I am Paragraph </p>
        <p id="myid2">
            <span> I am span </span>
            I am Paragraph.
        </p>
```



```
<script>
// firstChild is same as childNodes[0]

// firstElementChild is same as 'children[0]'.
var result = document.getElementById("myid");
// console.log(result.firstChild);
console.log(result.firstElementChild);
// document.write(result.firstElementChild.textContent);
// result.firstElementChild.textContent = "I am learning DOM";
document.write(result.firstChild.textContent);
result.firstChild.textContent = "Hello World";
</script>
</body>
</html>
```

- **lastChild** - The lastChild read-only property returns the last child of the node. If its parent is an element, then the child is generally an element node, a text node, or a comment node. It returns null if there are no child elements.
- Whitespace inside elements is considered

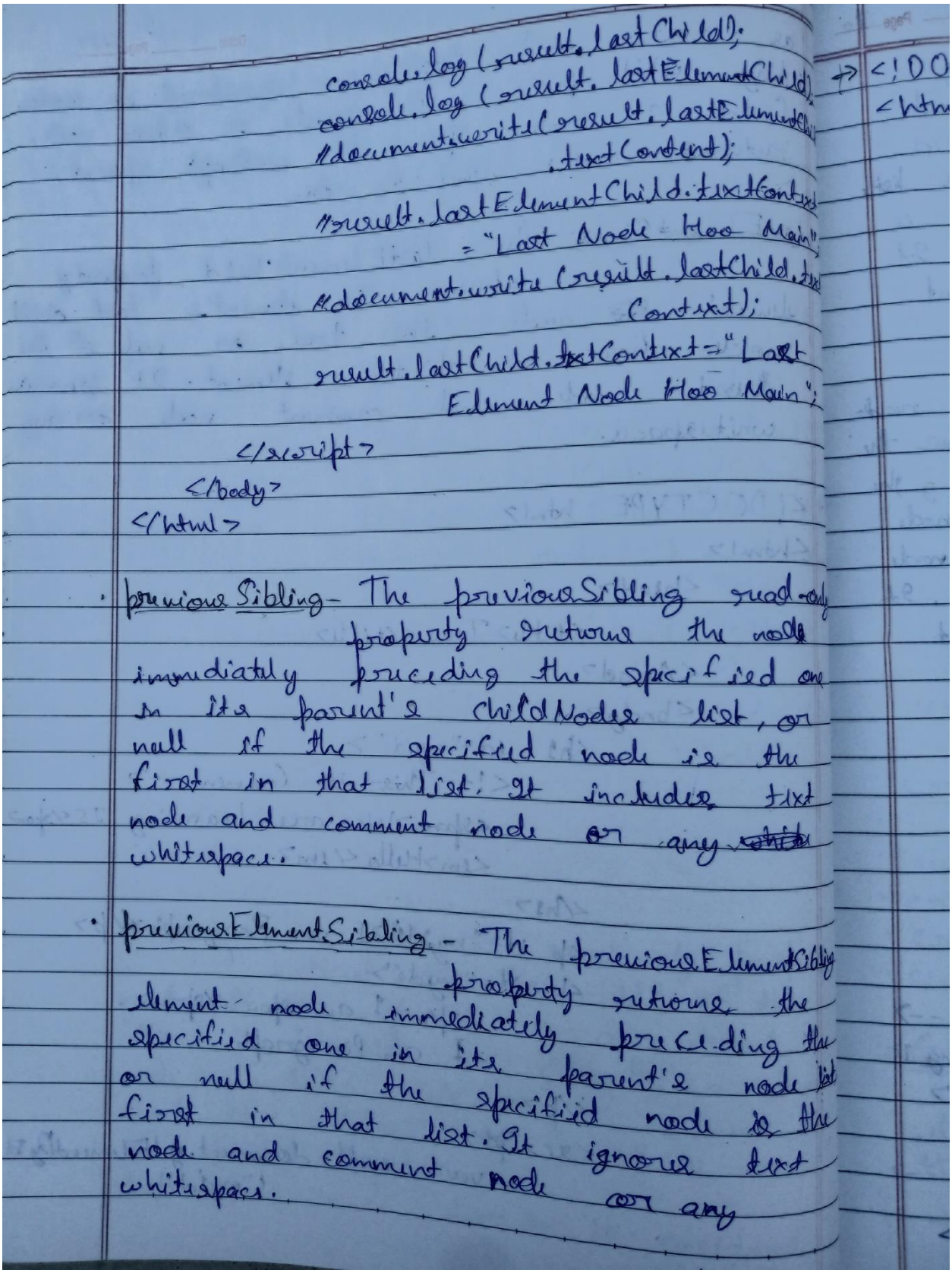


Date: Page: 267

as text, and text is considered as nodes. Any white space will create a #text node, from a single space to multiple spaces, returns, tabs, and so on.

- lastElementChild - The lastElementChild property returns the element's last child element or node the true, or null if the node has no children element. It ignores #text node and comment node or any white spaces.

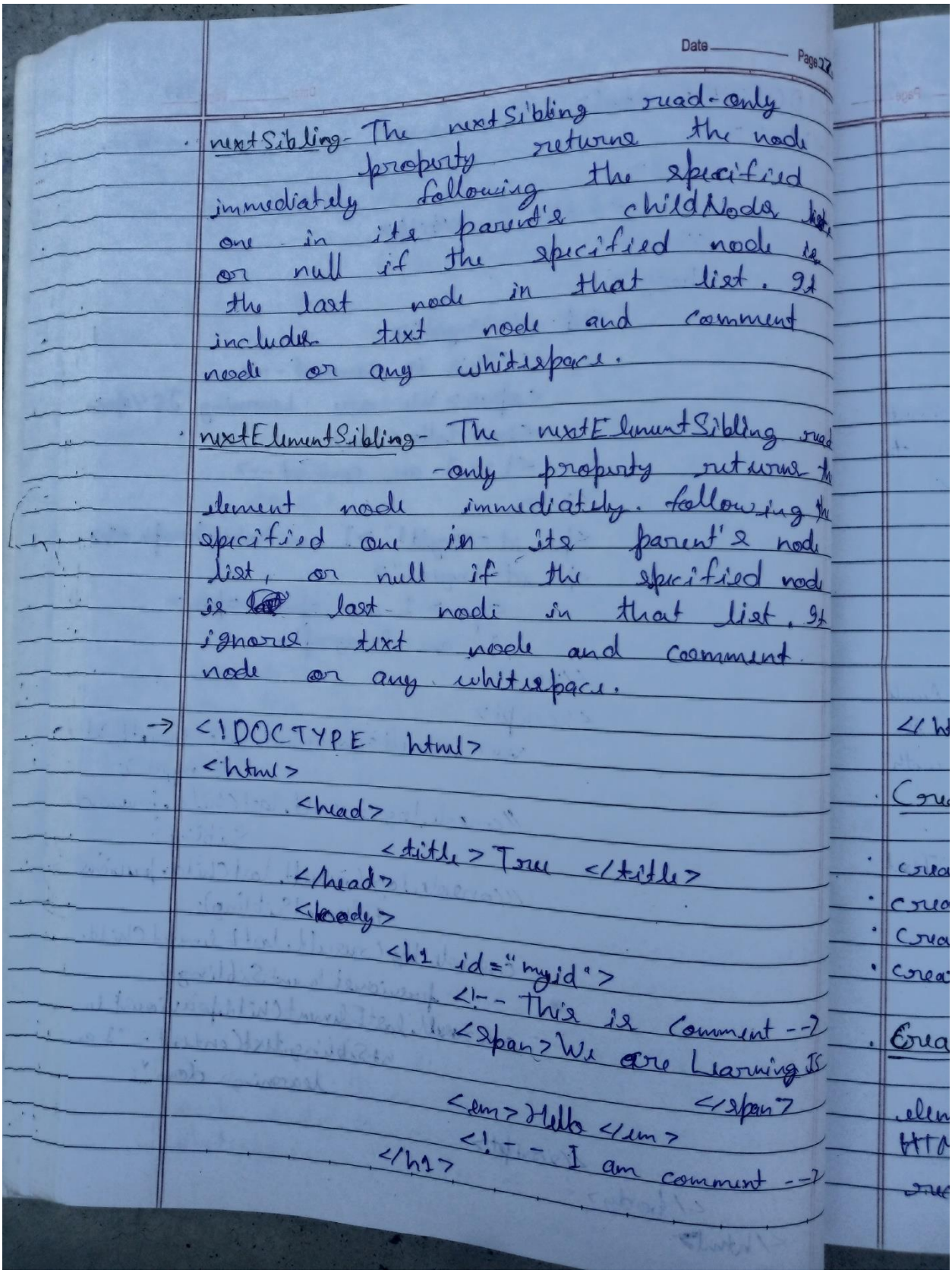
```
<!DOCTYPE html>
<html>
    <head>
        <title> True </title>
    </head>
    <body>
        <h1 id="myid">
            <!-- This is Comment -->
            <span>We are Learning JS</span>
            <em>Hello</em>
        </h1>
        <p id="myid1">I am Paragraph</p>
        <p id="myid2">
            <span> I am span </span>
            I am Paragraph
        </p>
        <script>
            var result = document.getElementById
                ("myid");
```



```
console.log(result.lastChild);
console.log(result.lastElementChild);
//document.write(result.lastElementChild
    .textContent);
//result.lastElementChild.textContent
    = "Last Node Hoo Main";
//document.write(result.lastChild.text
    Content);
result.lastChild.textContext = "Last
    Element Node Hoo Main";
</script>
</body>
</html>
```

- previousSibling - The previousSibling read-only property returns the node immediately preceding the specified one in its parent's childNodes list, or null if the specified node is the first in that list. It includes text node and comment node or any whitespace.

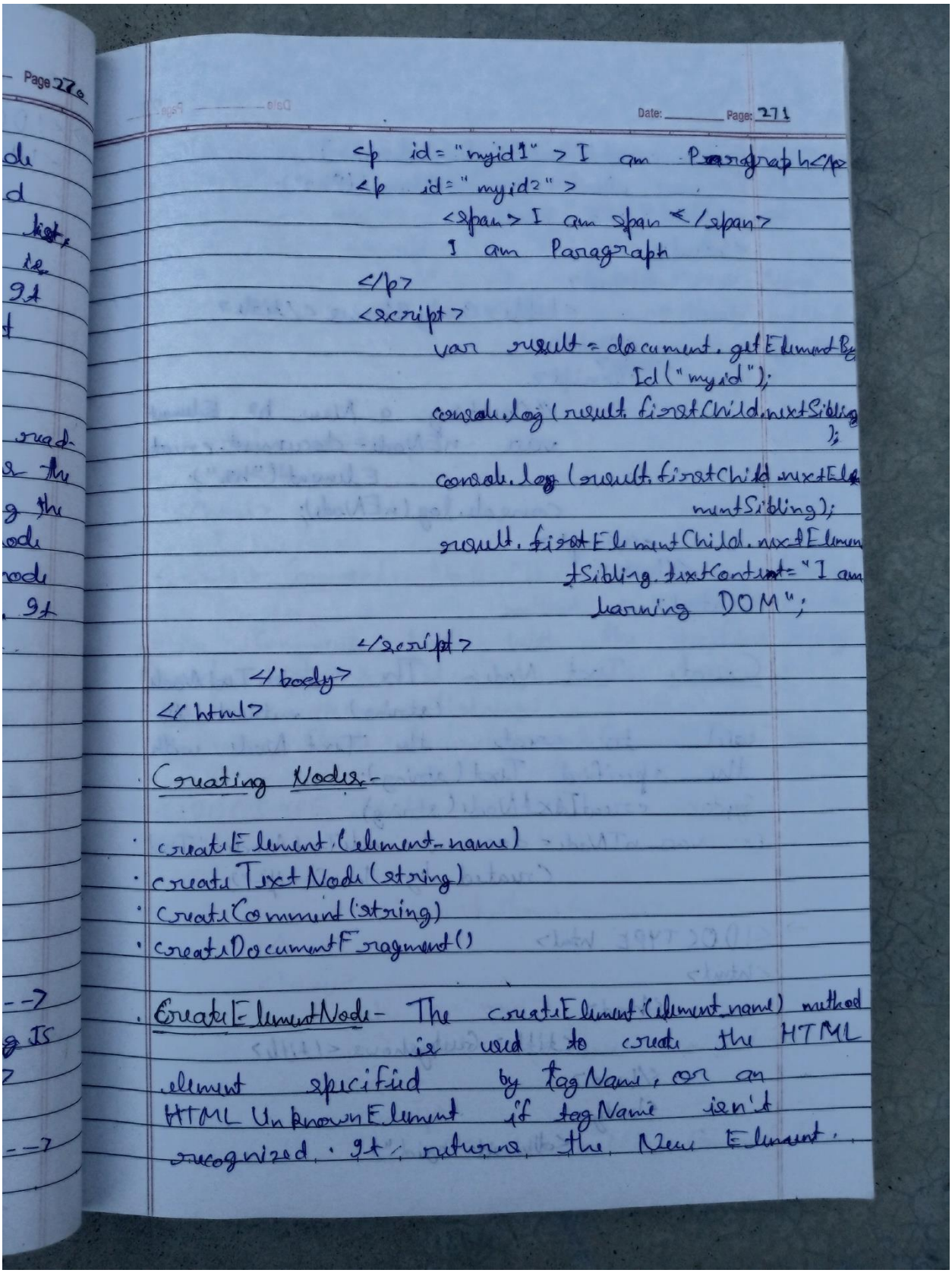
- previousElementSibling - The previousElementSibling property returns the Element node immediately preceding the specified one in its parent's node list or null if the specified node is the first in that list. It ignores text node and comment node or any whitespace.



nextSibling - The nextSibling read-only property returns the node immediately following the specified one in its parent's childNodes list, or null if the specified node is the last node in that list. It includes text node and comment node or any whitespace.

nextElementSibling - The nextElementSibling read-only property returns the element node immediately following the specified one in its parent's node list, or null if the specified node is last node in that list. It ignores text node and comment node or any whitespace.

```
<!DOCTYPE html>
<html>
    <head>
        <title> True </title>
    </head>
    <body>
        <h1 id="myid">
            <!-- This is Comment -->
            <span> We are Learning JS
            </span>
            <em> Hello </em>
            <!-- I am comment -->
        </h1>
```

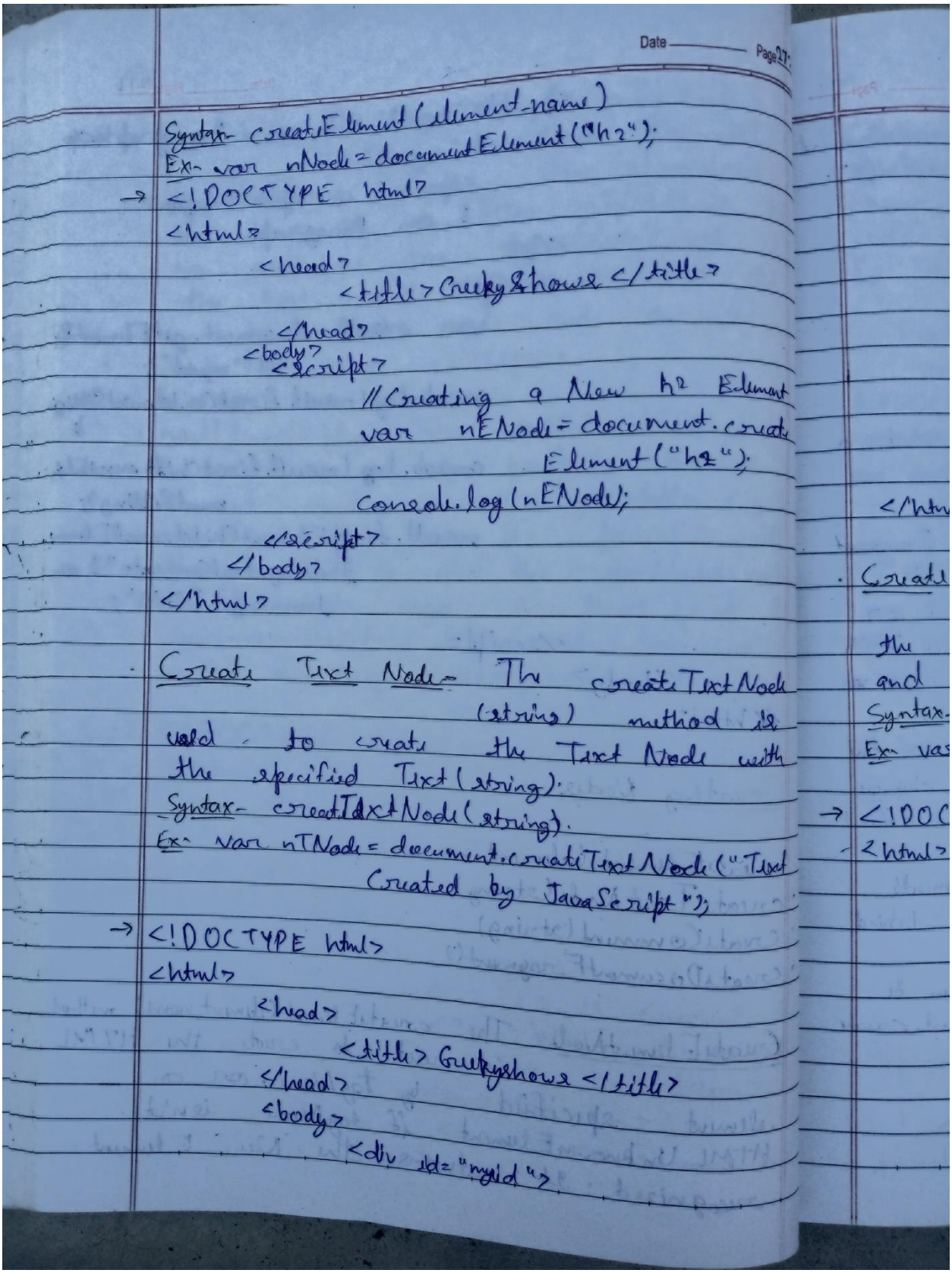


```
<p id="myid1">I am Paragraph</p>
<p id="myid2">
    <span>I am span</span>
    I am Paragraph
</p>
<script>
    var result = document.getElementById("myid");
    console.log(result.firstChild.nextSibling);
    console.log(result.firstChild.nextElementSibling);
    result.firstElementChild.nextElementSibling.textContent="I am learning DOM";
</script>
</body>
</html>
```

Creating Nodes:-

- createElement(element_name)
- createTextNode(string)
- createComment(string)
- createDocumentFragment()

- CreateElementNode- The createElement(element_name) method is used to create the HTML element specified by tagName, or an HTML UnknownElement if tagName isn't recognized. It returns the new Element.

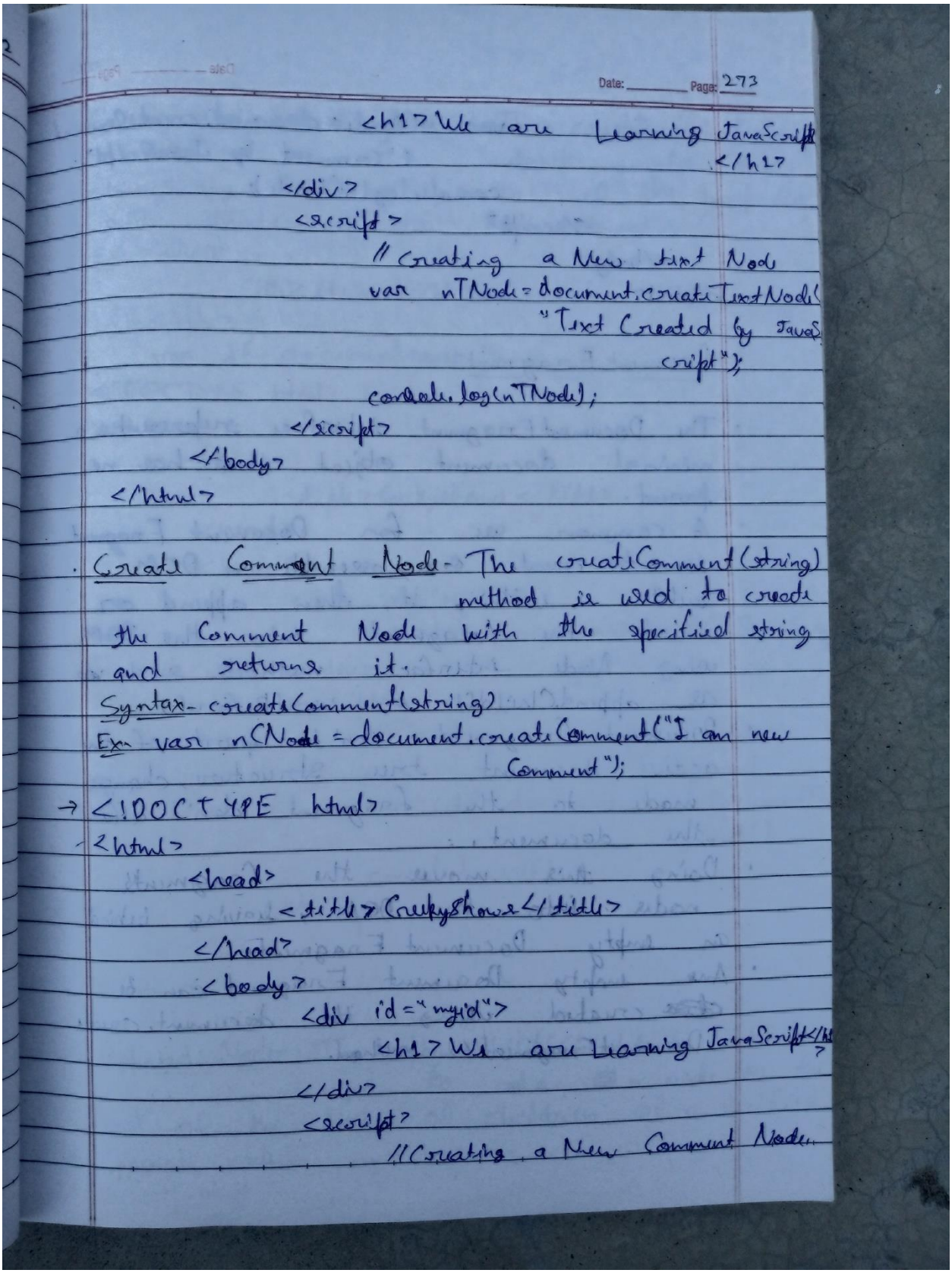


Syntax- createElement(element_name)
Ex- var nNode = document.Element("h2");

```
<!DOCTYPE html>
<html>
    <head>
        <title> Geeky Shows </title>
    </head>
    <body>
        <script>
            // Creating a New h2 Element
            var nENode = document.createElement("h2");
            console.log(nENode);
        </script>
    </body>
</html>
```

Create Text Node- The createTextNode(string) method is used to create the Text Node with the specified Text (string).
Syntax- createTextNode(string).
Ex- var nTNode = document.createTextNode("Text Created by JavaScript");

```
<!DOCTYPE html>
<html>
    <head>
        <title> Geekyshows </title>
    </head>
    <body>
        <div id="myid">
```



Date: Page: 273

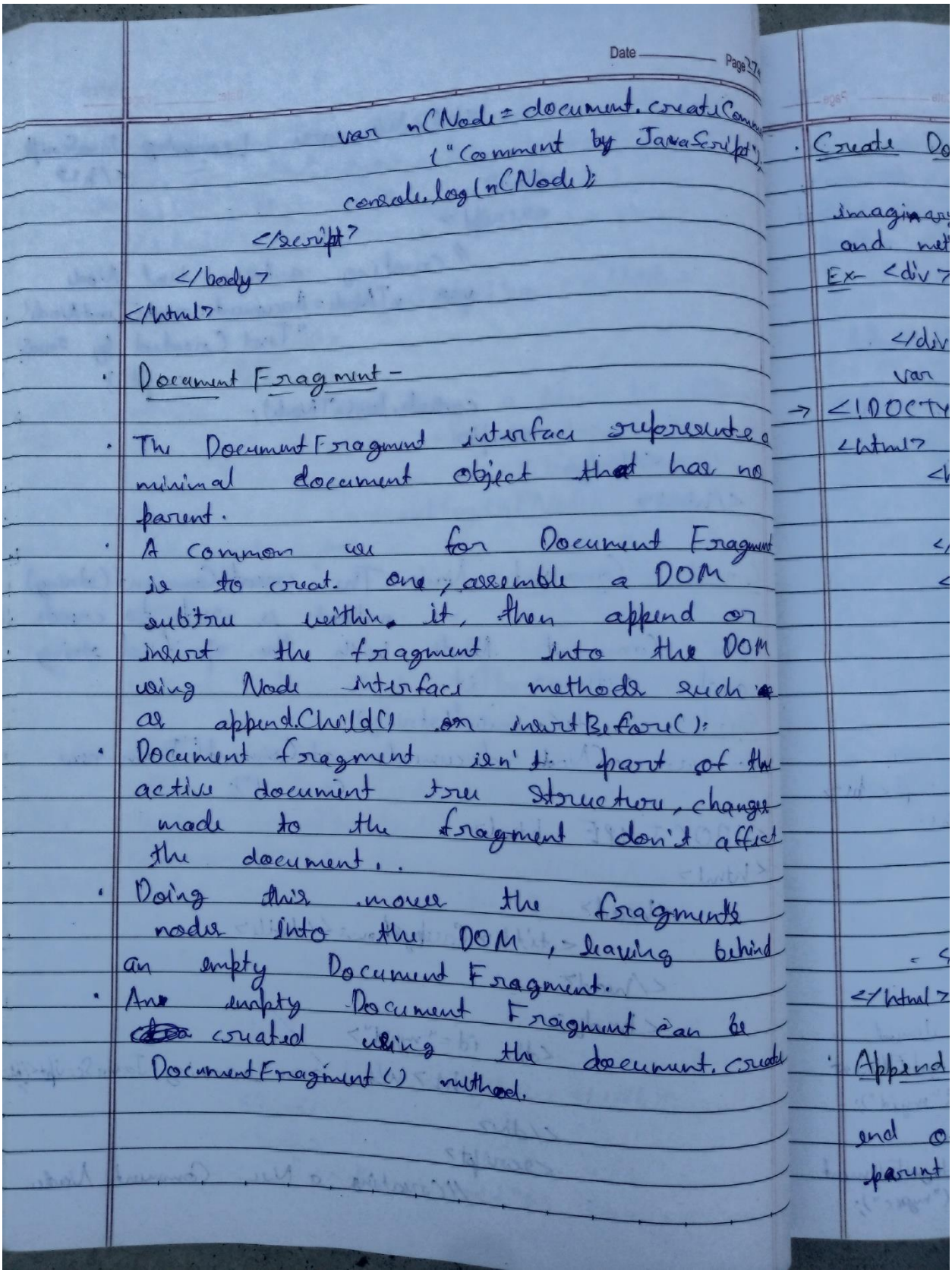
```
        <h1> We are Learning JavaScript
                                    </h1>
    </div>
    <script>
        // Creating a New Text Node
        var nTNode = document.createTextNode(
                    "Text Created by JavaS
                                    cript");
        console.log(nTNode);
    </script>
</body>
</html>
```

Create Comment Node - The createComment(string) method is used to create the Comment Node with the specified string and returns it.

Syntax - createComment(string)

Ex- var nCNode = document.createComment("I am new Comment");

```
<!DOCTYPE html>
<html>
    <head>
        <title> Geekyshows </title>
    </head>
    <body>
        <div id="myid">
            <h1> We are Learning JavaScript</h1
        </div>
        <script>
            // Creating a New Comment Node
```

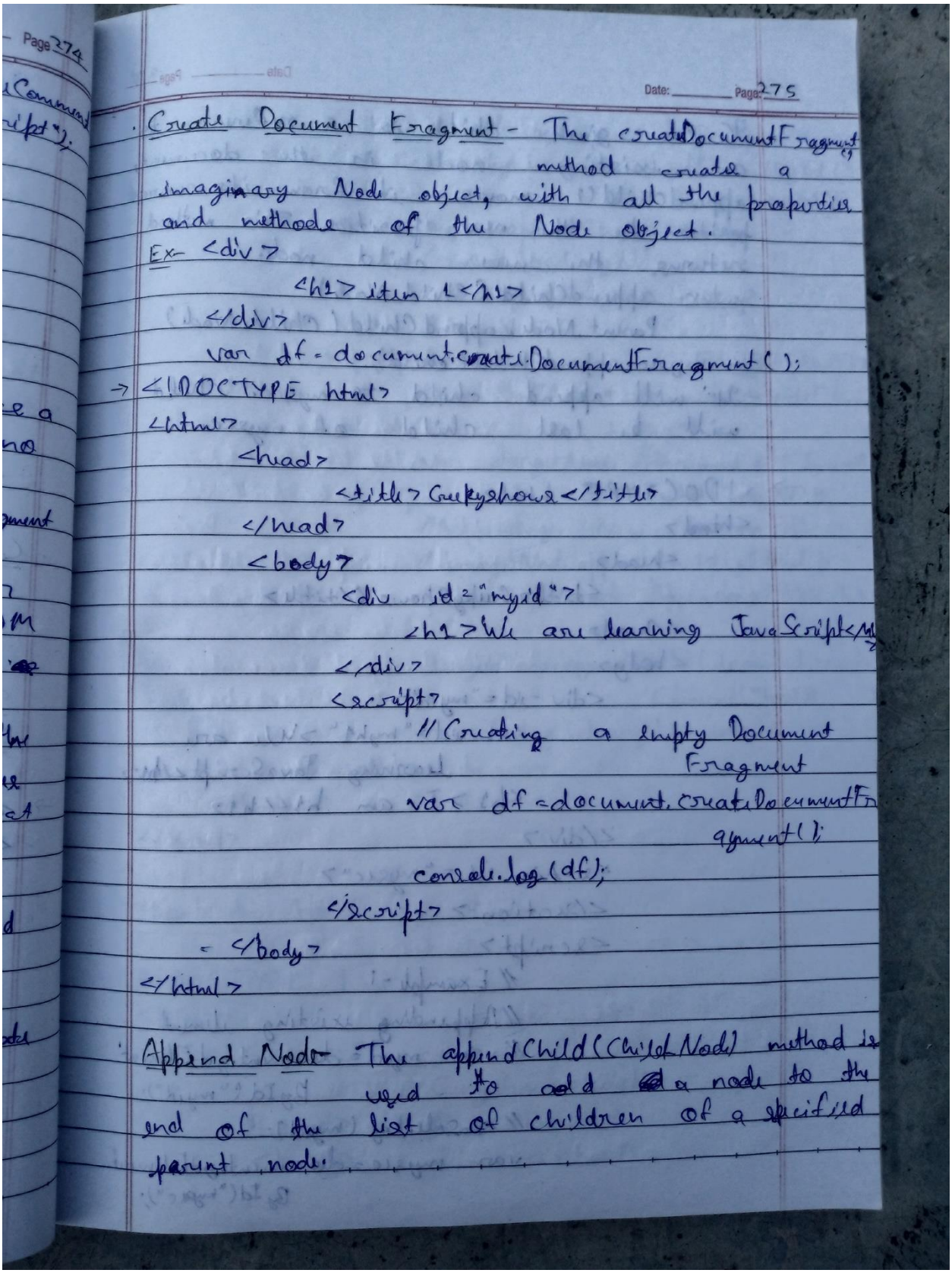


var nCNode = document.createComment
("Comment by JavaScript");
console.log(nCNode);
</script>
</body>
</html>

- Document Fragment -

- The DocumentFragment interface represents a minimal document object that has no parent.
- A common use for DocumentFragment is to create one, assemble a DOM subtree within it, then append or insert the fragment into the DOM using Node interface methods such as appendChild() or insertBefore().
- DocumentFragment isn't part of the active document tree structure. Changes made to the fragment don't affect the document.
- Doing this moves the fragment's nodes into the DOM, leaving behind an empty DocumentFragment.
- An empty DocumentFragment can be created using the document.createDocumentFragment() method.

Create Do
imagina
and nat
Ex-
</div
var
<!DOCTY
<html>
</html>
Append
end o
parent



Date: Page: 275

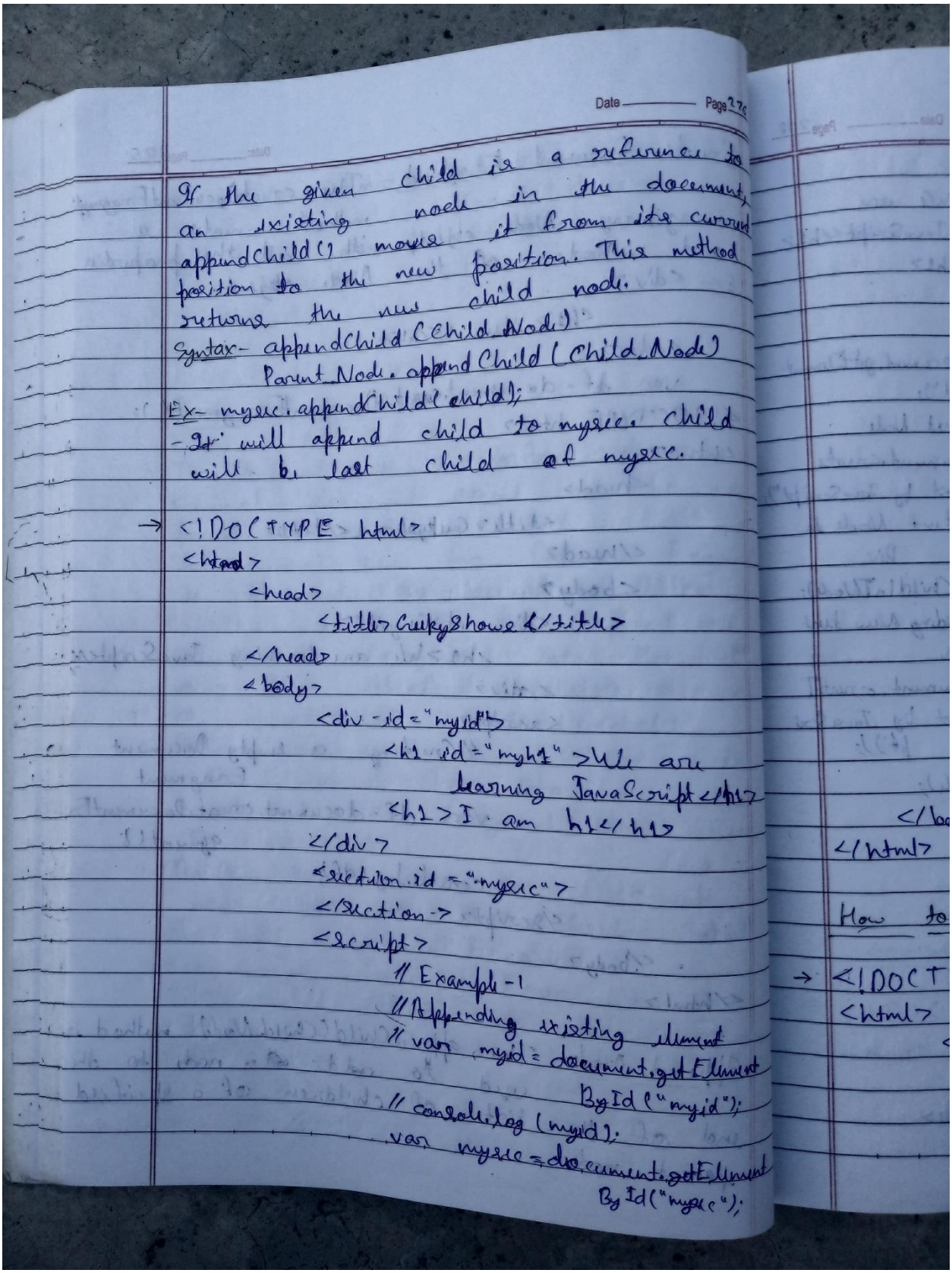
· Create Document Fragment - The createDocumentFragment() method creates a imaginary Node object, with all the properties and methods of the Node object.

Ex-
<h1> item 1 </h1>

var df = document.createDocumentFragment();

```
<!DOCTYPE html>
<html>
    <head>
        <title> Geekyshows </title>
    </head>
    <body>
        <div id = "myid">
            <h1> We are learning JavaScript </h1>
        </div>
        <script>
            // Creating a empty Document Fragment
            var df = document.createDocumentFragment();
            console.log(df);
        </script>
    </body>
</html>
```

· Append Node - The appendChild(ChildNode) method is used to add a node to the end of the list of children of a specified parent node.



If the given child is a reference to an existing node in the document, appendChild() moves it from its current position to the new position. This method returns the new child node.

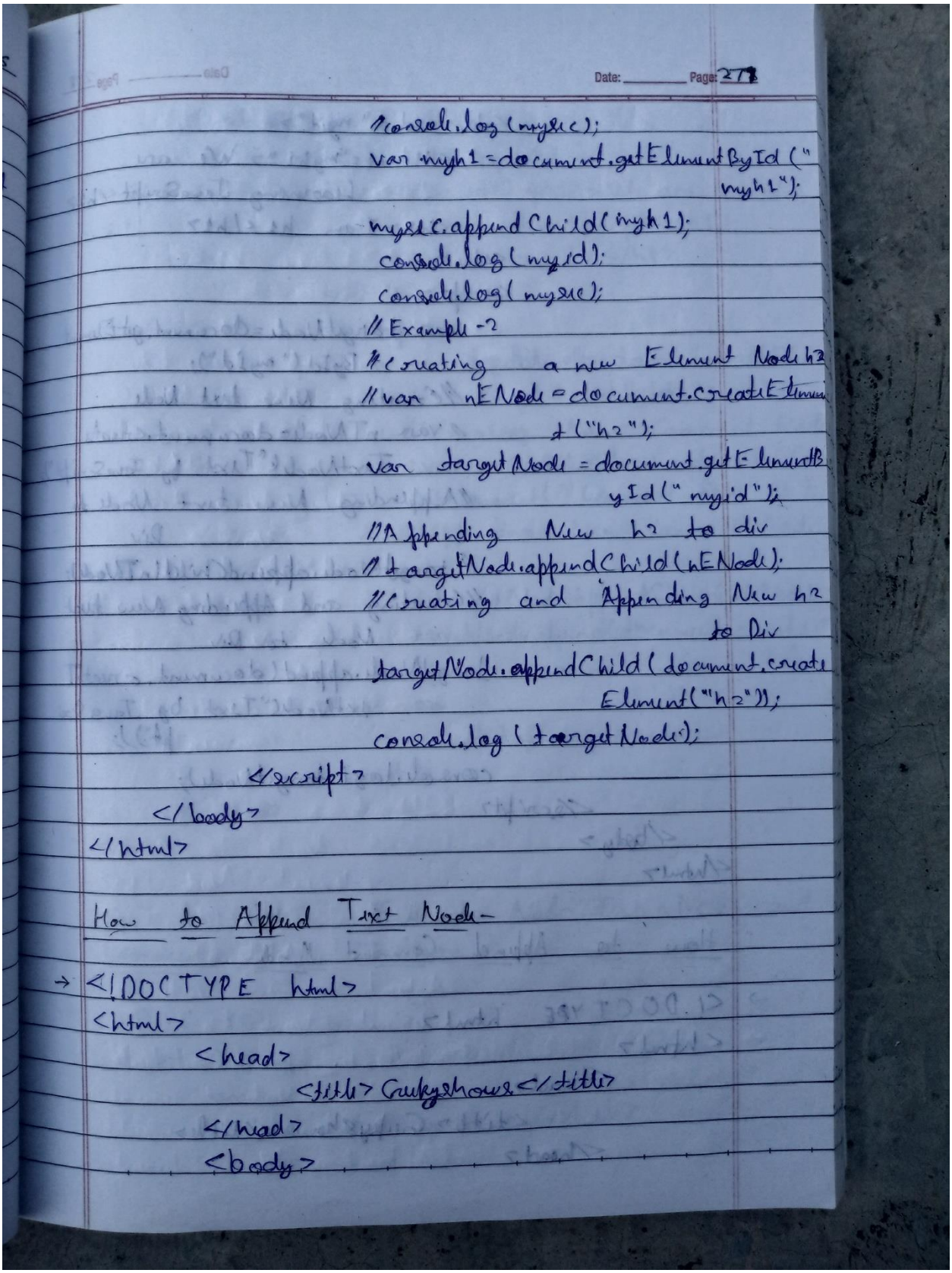
Syntax- appendChild(Child_Node)

Parent_Node.appendChild(Child_Node)

Ex- mysec.appendChild(child);

- It will append child to mysec. child will be last child of mysec.

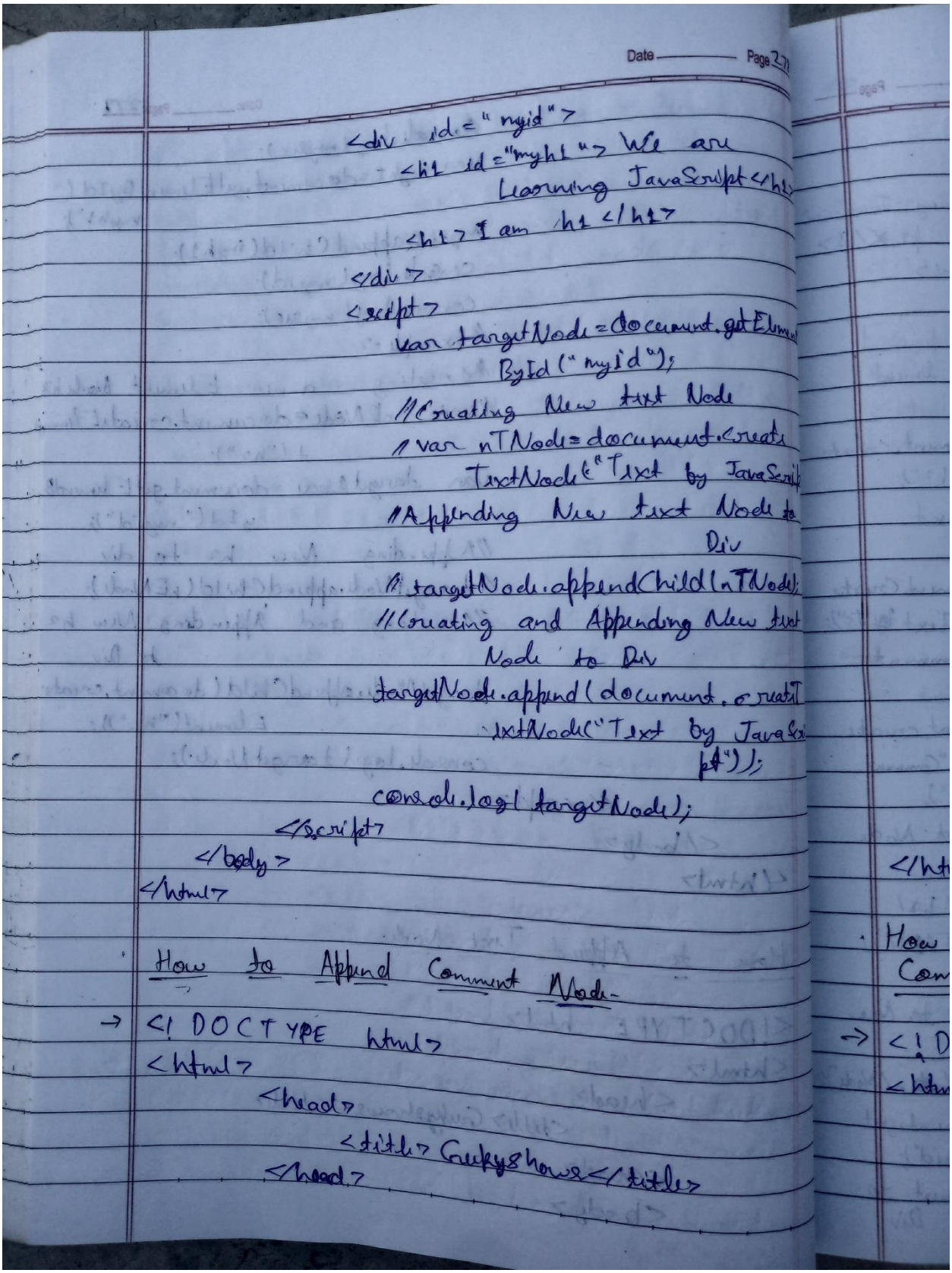
```
<!DOCTYPE html>
<html>
    <head>
        <title>Geekyshows</title>
    </head>
    <body>
        <div id="myid">
            <h1 id="myh1">We are learning JavaScript</h1>
            <h1>I am h1</h1>
        </div>
        <section id="mysec">
        </section>
        <script>
            // Example-1
            // Appending existing element
            // var myid = document.getElementById("myid");
            // console.log(myid);
            var mysec = document.getElementById("mysec");
```



```
//console.log(mysec);
var myh1 = document.getElementById("myh1");
mysec.appendChild(myh1);
console.log(myid);
console.log(mysec);
// Example-2
// Creating a new Element Node h2
// var nENode = document.createElement("h2");
var targetNode = document.getElementById("myid");
// Appending New h2 to div
// targetNode.appendChild(nENode);
// Creating and Appending New h2 to Div
targetNode.appendChild(document.createElement("h2"));
console.log(targetNode);
</script>
</body>
</html>
```

How to Append Text Node-

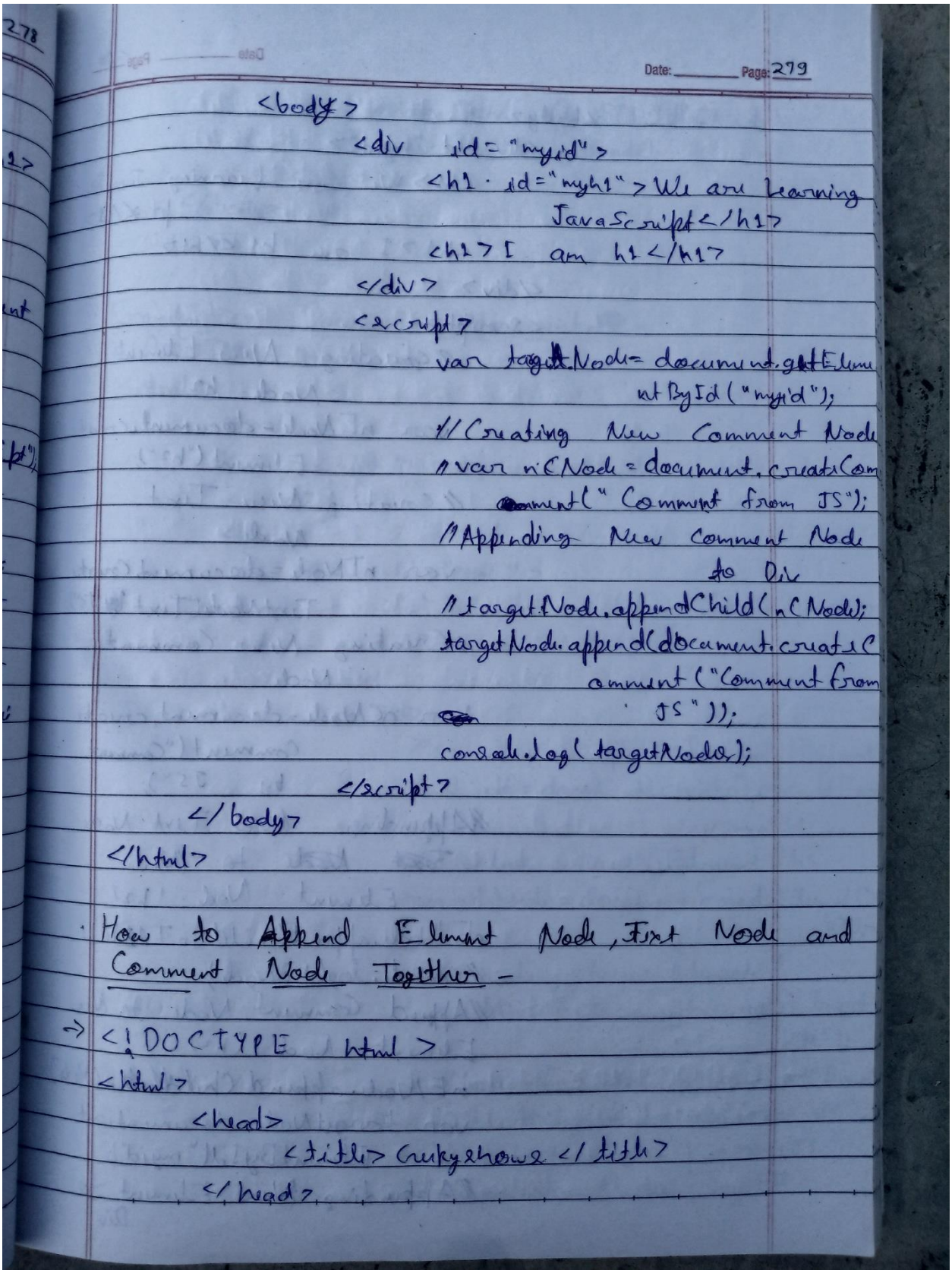
```
<!DOCTYPE html>
<html>
    <head>
        <title>Geekyshows</title>
    </head>
    <body>
```



```
<div id="myid">
    <h1 id="myh1">We are
        Learning JavaScript</h1>
    <h1>I am h1</h1>
</div>
<script>
    var targetNode=document.getElement
        ById("myid");
    //Creating New text Node
    // var nTNode=document.creat
        TextNode("Text by JavaScri
    //Appending New text Node to
        Div
    // targetNode.appendChild(nTNode);
    //Creating and Appending New text
        Node to Div
    targetNode.append(document.creatT
        extNode("Text by JavaScr
        pt"));
    console.log(targetNode);
</script>
</body>
</html>
```

How to Append Comment Node-

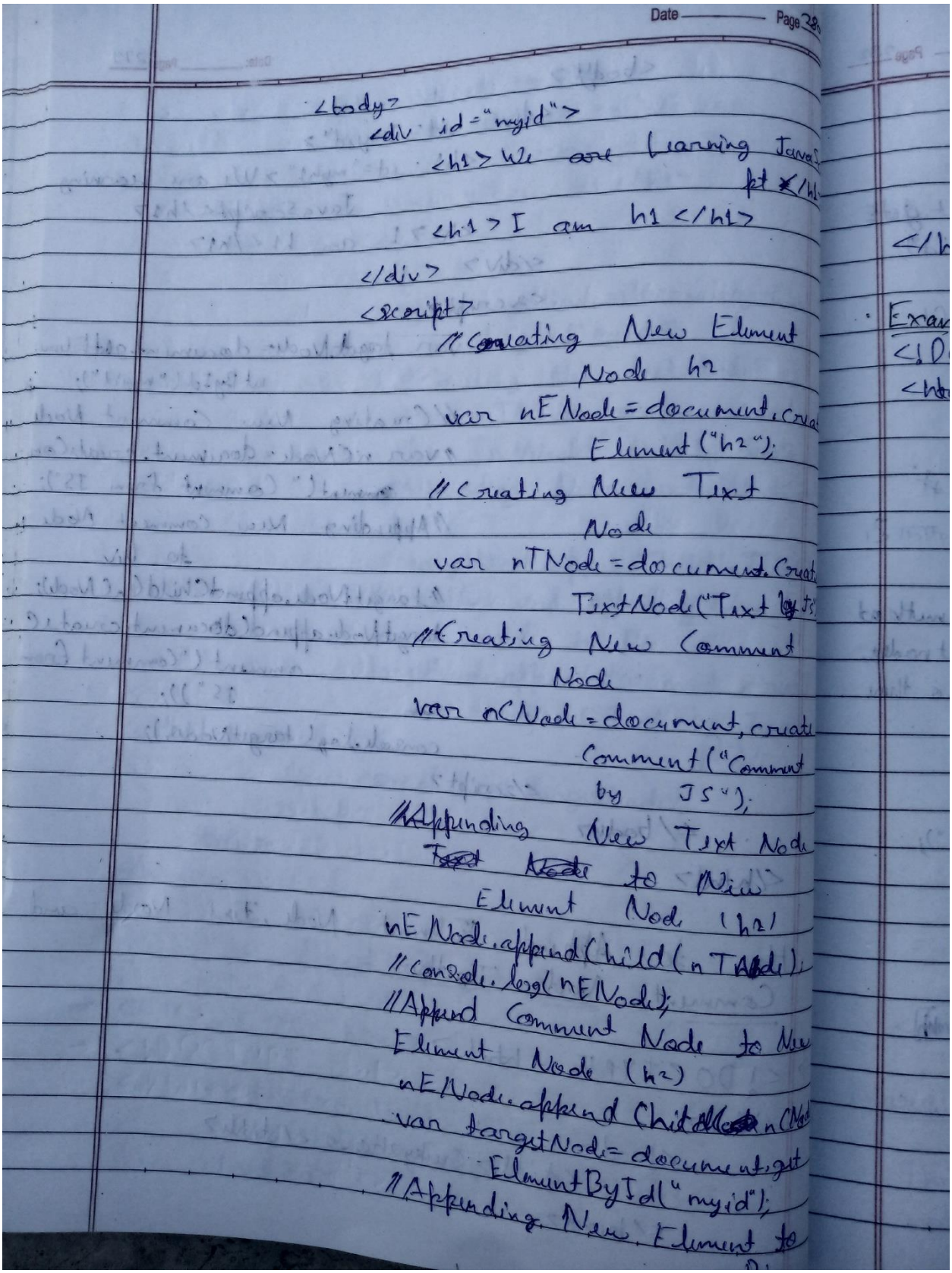
```
<!DOCTYPE html>
<html>
    <head>
        <title>Geekyshows</title>
    </head>
```



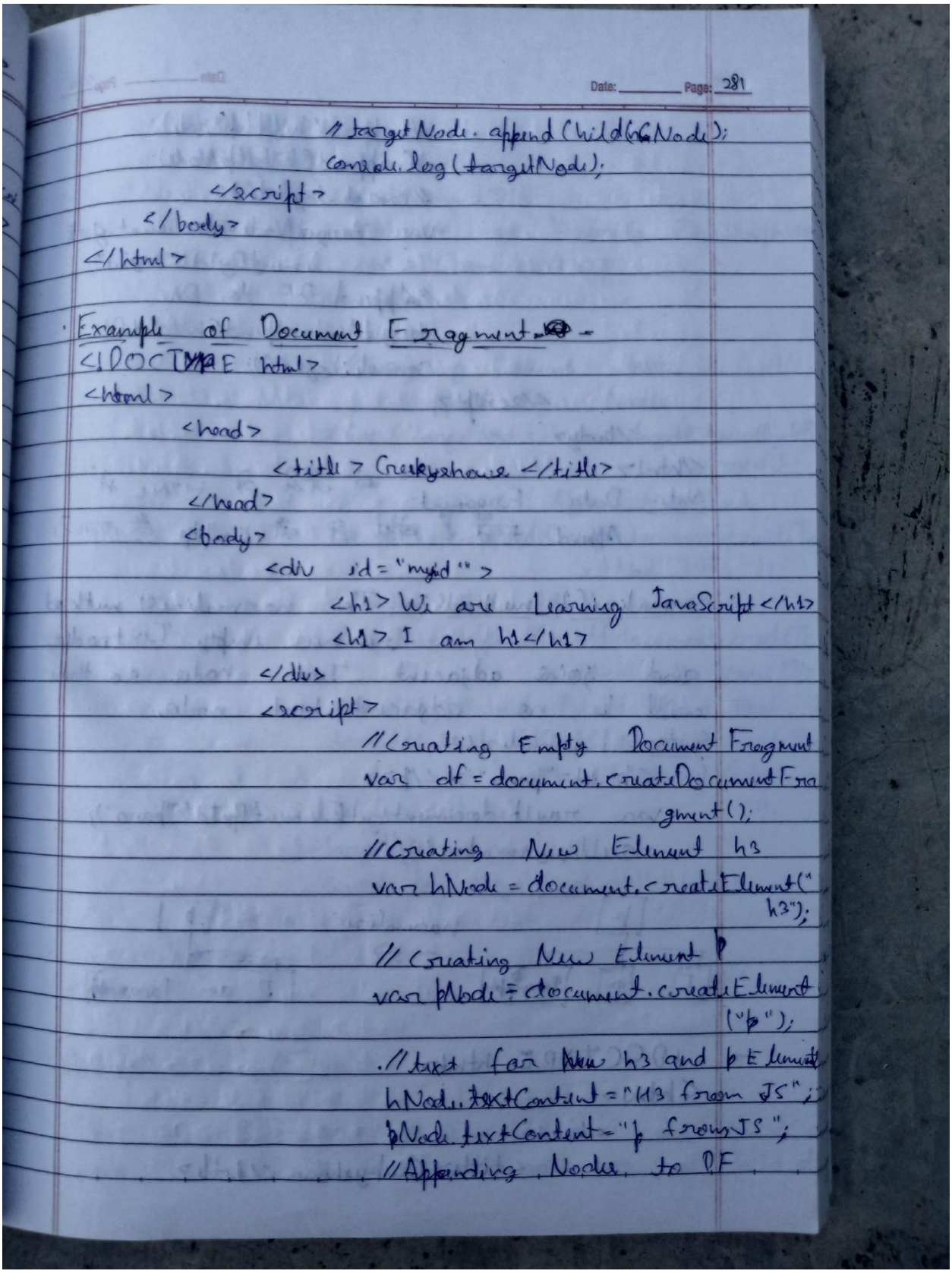
```
<body>
    <div id="myid">
        <h1 id="myh1">We are learning JavaScript</h1>
        <h1>I am h1</h1>
    </div>
    <script>
        var targetNode = document.getElementById("myid");
        // Creating New Comment Node
        // var nCNode = document.createComment("Comment from JS");
        // Appending New Comment Node to Div
        // targetNode.appendChild(nCNode);
        targetNode.append(document.createComment("Comment from JS"));
        console.log(targetNode);
    </script>
</body>
</html>
```

- How to Append Element Node, Text Node and Comment Node Together -

```
<!DOCTYPE html>
<html>
    <head>
        <title>Geekyshows</title>
    </head>
```



```
<body>
<div id="myid">
<h1> We are Learning JavaS
pt </h1>
<h1> I am h1 </h1>
</div>
<script>
// Creating New Element
   Node h2
var nENode = document.crea
   Element("h2");
// Creating New Text
   Node
var nTNode = document.Creat
   TextNode("Text by JS");
// Creating New Comment
   Node
var nCNode = document.creati
   Comment("Comment
   by JS");
// Appending New Text Node
   Text Node to New
   Element Node (h2)
nENode.appendChild(nTNode);
// console.log(nENode);
// Append Comment Node to New
Element Node (h2)
nENode.appendChild(nCNode
var targetNode = document.get
   ElementById("myid");
// Appending New Element to
```



```
Date: _____ Page: 281

        // targetNode.appendChild(childNode);
        console.log(targetNode);
    </script>
</body>
</html>

Example of Document Fragment -
<!DOCTYPE html>
<html>
    <head>
        <title> Geekyshows </title>
    </head>
    <body>
        <div id="myid">
            <h1> We are Learning JavaScript </h1>
            <h1> I am h1 </h1>
        </div>
        <script>
            // Creating Empty Document Fragment
            var df = document.createDocumentFragment();
            // Creating New Element h3
            var hNode = document.createElement("h3");
            // Creating New Element p
            var pNode = document.createElement("p");
            // text for New h3 and p Element
            hNode.textContent = "H3 from JS";
            pNode.textContent = "p from JS";
            // Appending Nodes to DF
```

```
df.appendChild(pNode);
// console.log(df);
var targetNode = document.getElementById("myid");
// Append DF to DN
targetNode.appendChild(df);
console.log(df);
</script>
</body>
</html>
```

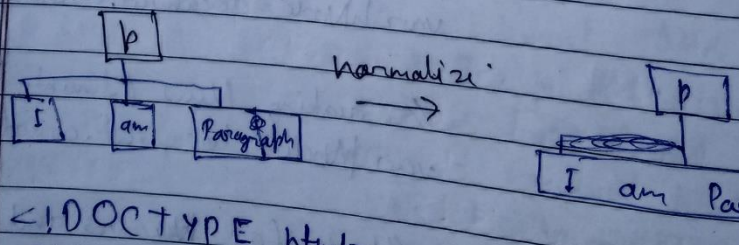
Note- Data Fragment को जैसे ही HTML में Append करते हैं वैसे ही वो Empty हो जाता है।

normalize() method() - The normalize() method removes empty Text nodes and joins adjacent Text nodes so there will be no adjacent text nodes.

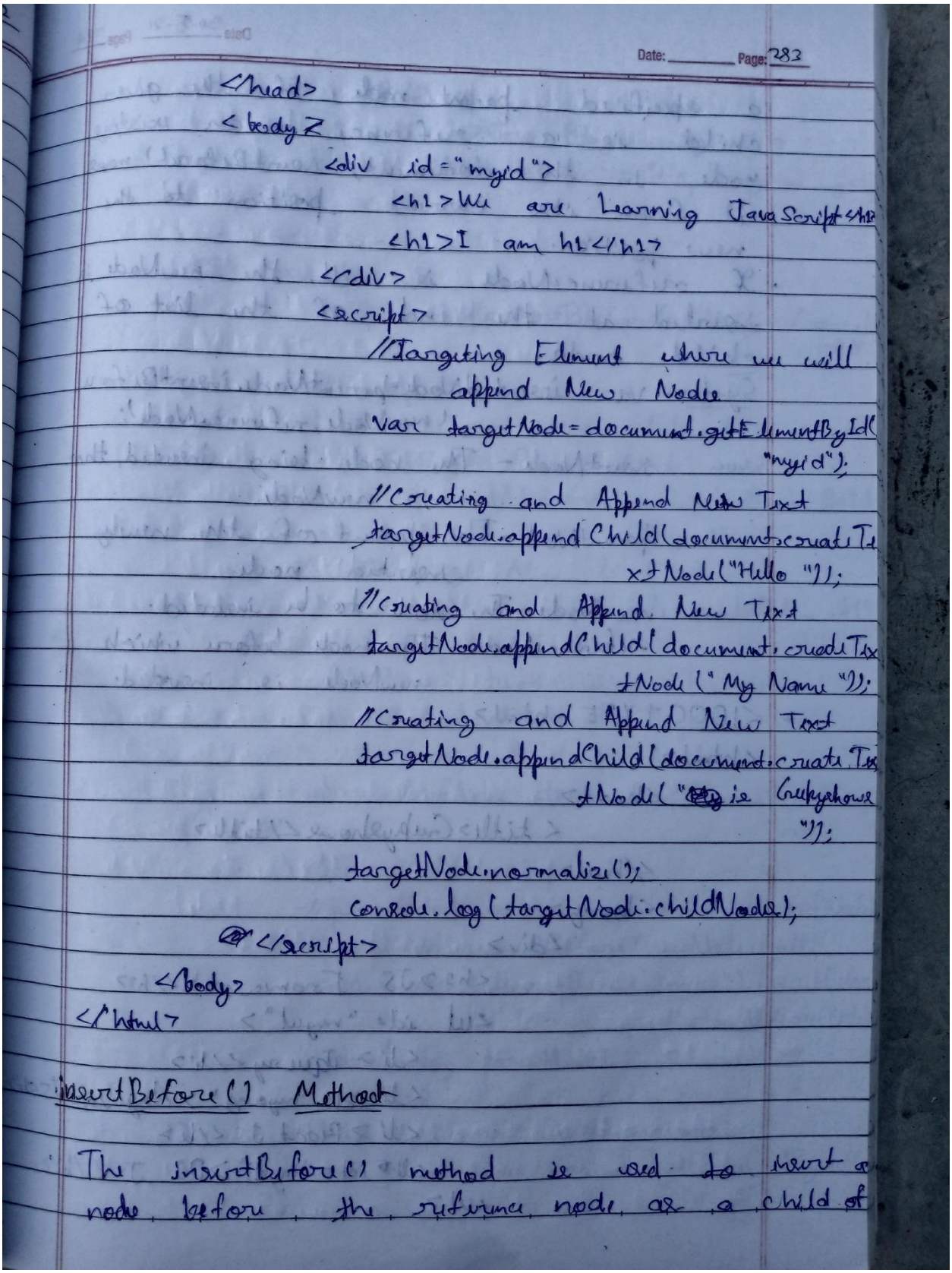
Syntax - normalize()

Ex- `<p id="para"></p>`

```
var result = document.getElementById("para");
result.normalize();
```



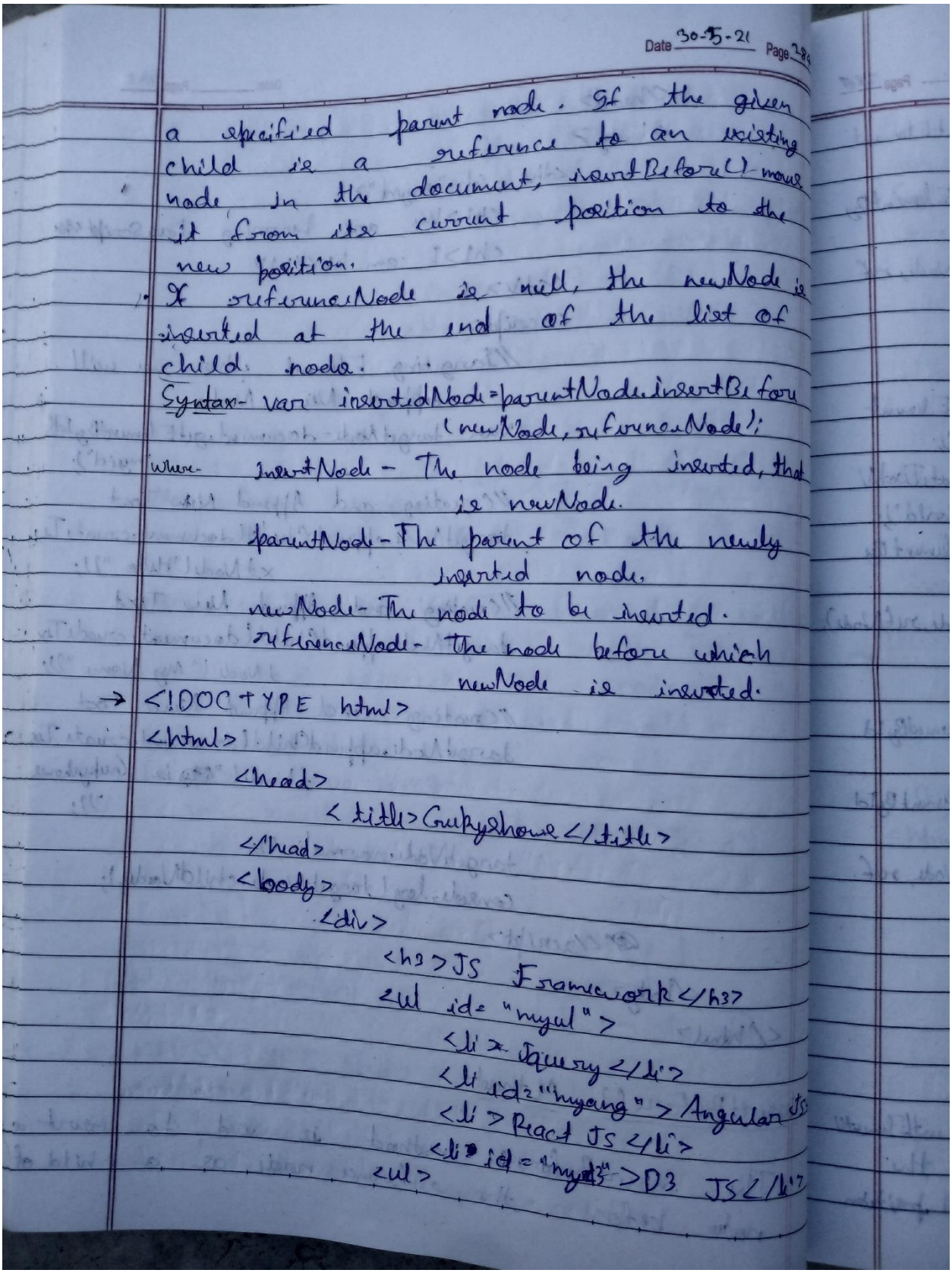

```
<!DOCTYPE html>
<html>
  <head>
    <title>Geekyshows</title>
```



```
</head>
<body>
    <div id="myid">
        <h1>We are Learning JavaScript</h1>
        <h1>I am h1</h1>
    </div>
    <script>
        //Targeting Element where we will
            append New Node
        var targetNode = document.getElementById(
                                "myid");
        //Creating and Append New Text
        targetNode.appendChild(document.creatTe
                        xtNode("Hello "));
        //Creating and Append New Text
        targetNode.appendChild(document.creatTex
                        tNode("My Name "));
        //Creating and Append New Text
        targetNode.appendChild(document.creatTex
                        tNode("[illegible] is Geekyshows
                        "));
        targetNode.normalize();
        console.log(targetNode.childNodes);
    </script>
</body>
</html>
```

insertBefore() Method

The insertBefore() method is used to insert a node before the reference node as a child of



Date 30-5-21 Page 284

a specified parent node. If the given child is a reference to an existing node in the document, insertBefore() moves it from its current position to the new position.

* If referenceNode is null, the newNode is inserted at the end of the list of child nodes.

Syntax- var insertedNode = parentNode.insertBefore(newNode, referenceNode);

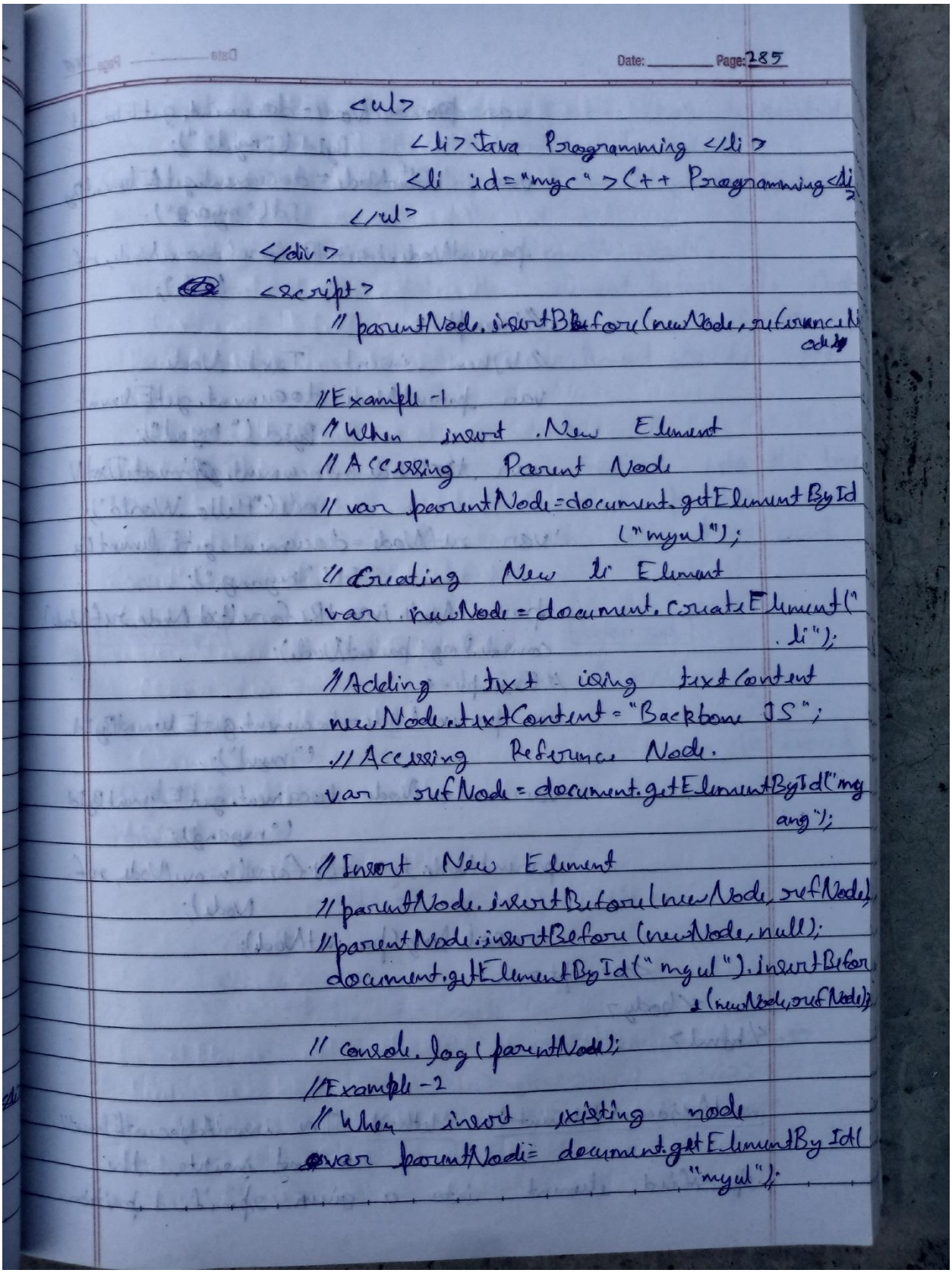
where- insertNode - The node being inserted, that is newNode.

parentNode - The parent of the newly inserted node.

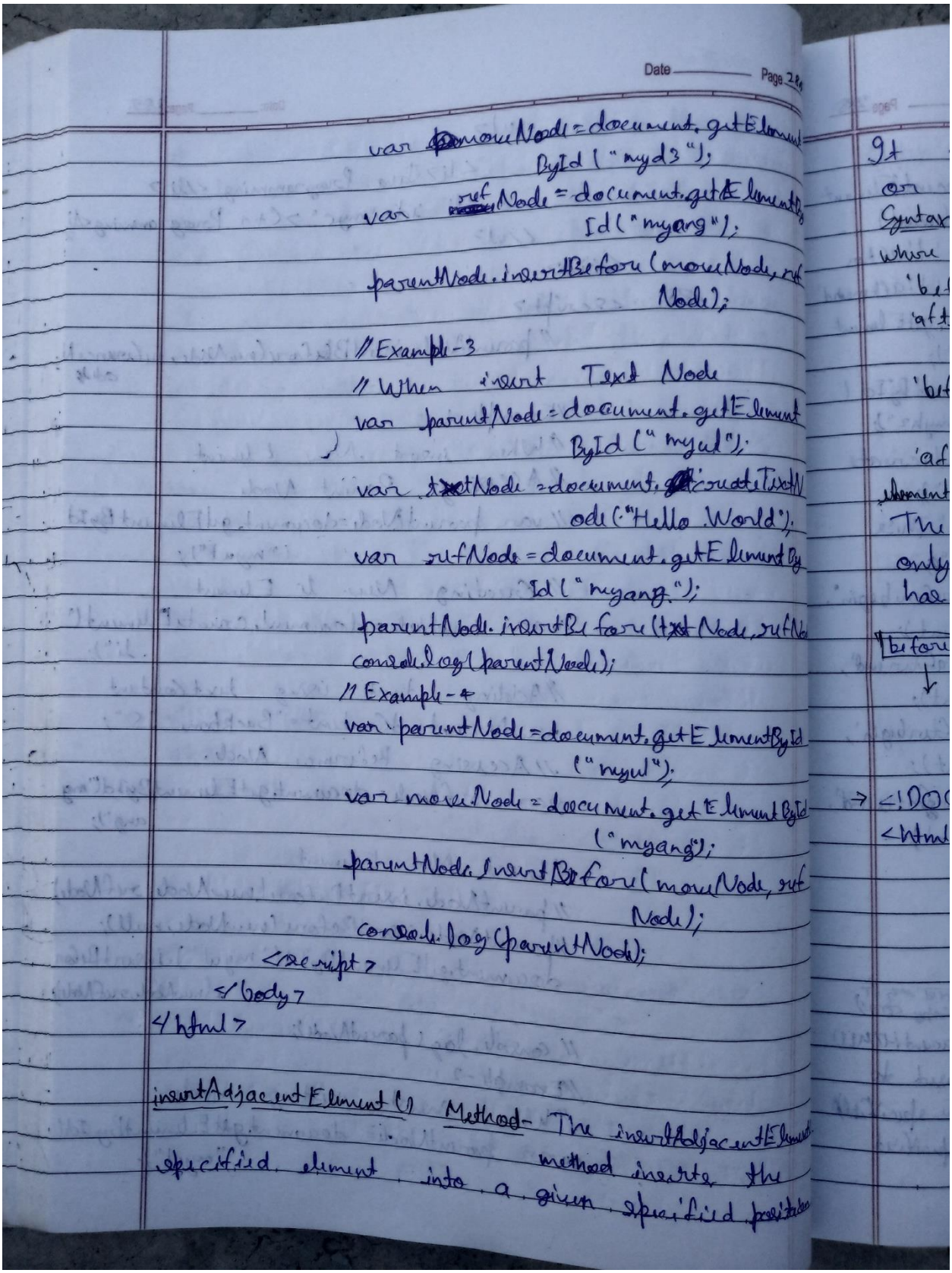
newNode - The node to be inserted.

referenceNode - The node before which newNode is inserted.

→
```
<!DOCTYPE html>
<html>
    <head>
        <title>Geekyshows</title>
    </head>
    <body>
        <div>
            <h3>JS Framework</h3>
            <ul id="myul">
                <li>Jquery</li>
                <li id="myang">Angular JS</li>
                <li>React JS</li>
                <li id="myd3">D3 JS</li>
            </ul>
```

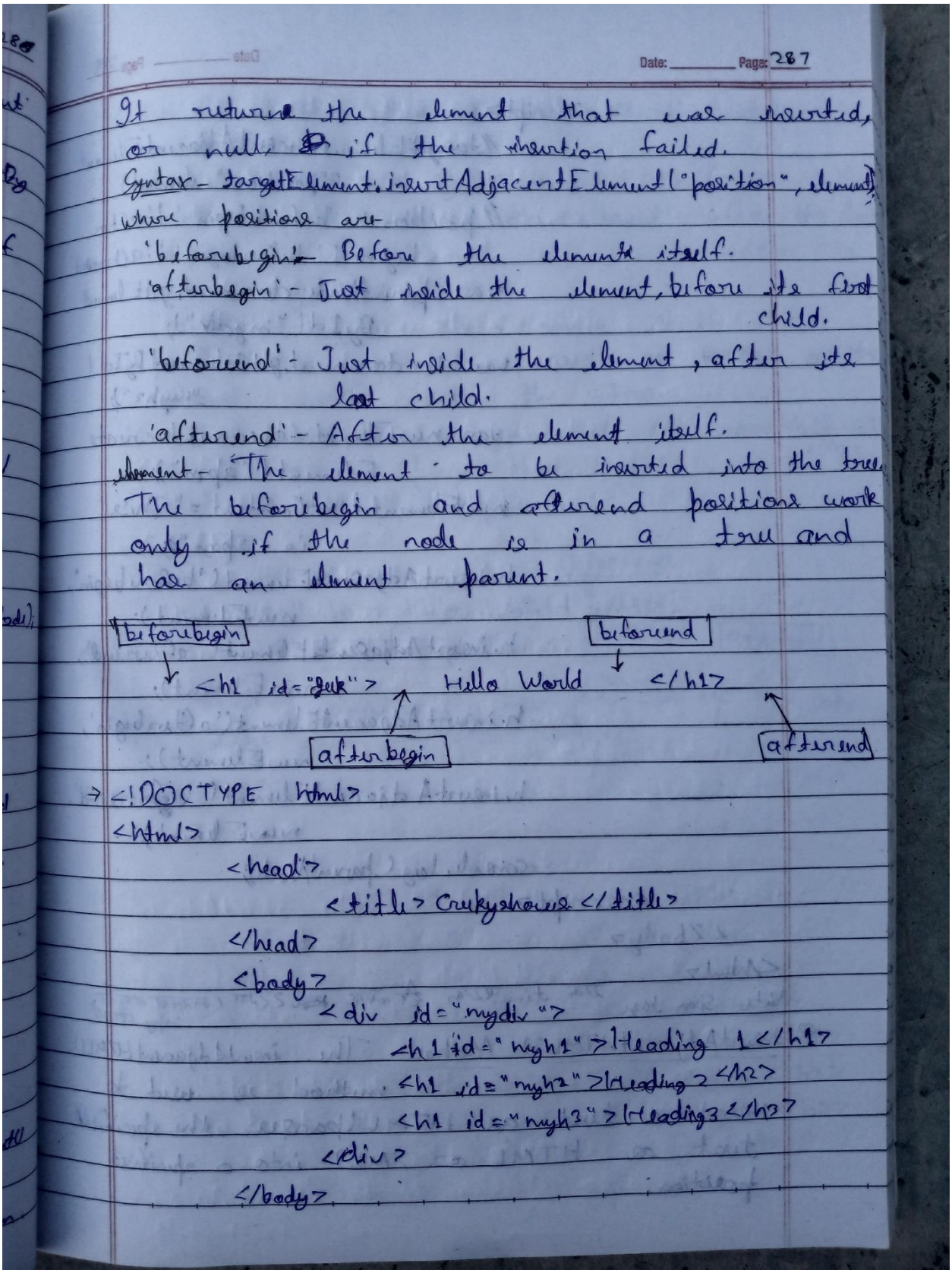


```
Date: Page: 285

<ul>
    <li>Java Programming</li>
    <li id="myc">C++ Programming</li>
</ul>
</div>
<script>
// parentNode.insertBefore(newNode, referenceNode)

//Example-1
// When insert New Element
// Accessing Parent Node
// var parentNode=document.getElementById
                ("myul");
// Creating New li Element
var newNode = document.createElement("
                li");
// Adding text using textContent
newNode.textContent = "Backbone JS";
// Accessing Reference Node.
var refNode = document.getElementById("my
                ang");

// Insert New Element
// parentNode.insertBefore(newNode, refNode);
// parentNode.insertBefore(newNode, null);
document.getElementById("myul").insertBefor
    e(newNode, refNode);

// console.log(parentNode);
//Example-2
// When insert existing node
var parentNode= document.getElementById
                ("myul");
```



```
var moveNode = document.getElementById("myd3");
var refNode = document.getElementById("myang");
parentNode.insertBefore(moveNode, refNode);

// Example-3
// When insert Text Node
var parentNode = document.getElementById("myul");
var textNode = document.createTextNode("Hello World");
var refNode = document.getElementById("myang");
parentNode.insertBefore(textNode, refNode);
console.log(parentNode);
// Example-4
var parentNode = document.getElementById("myul");
var moveNode = document.getElementById("myang");
parentNode.insertBefore(moveNode, refNode);
console.log(parentNode);
</script>
</body>
</html>
```

insertAdjacentElement() Method- The insertAdjacentElement() method inserts the specified element into a given specified position



Date: Page: 287

It returns the element that was inserted, or null if the insertion failed.

Syntax - targetElement.insertAdjacentElement("position", element)

where positions are

'beforebegin' - Before the element itself.

'afterbegin' - Just inside the element, before its first child.

'beforeend' - Just inside the element, after its last child.

'afterend' - After the element itself.

element - The element to be inserted into the tree.

The beforebegin and afterend positions work only if the node is in a tree and has an element parent.

beforebegin

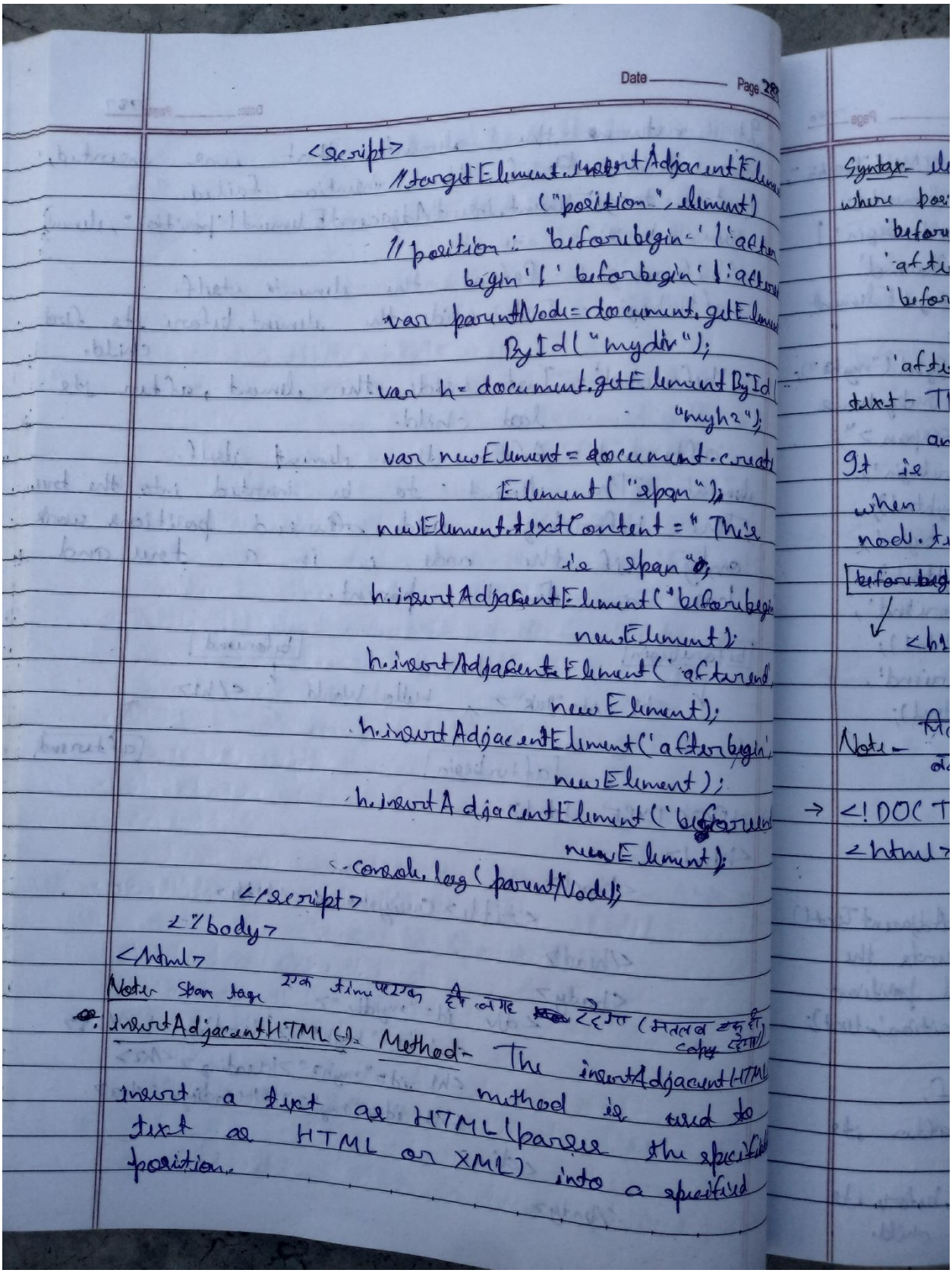
beforeend

<h1 id="geek"> Hello World </h1>

afterbegin

afterend

```
<!DOCTYPE html>
<html>
    <head>
        <title> Geekyshows </title>
    </head>
    <body>
        <div id="mydiv">
            <h1 id="myh1">Heading 1</h1>
            <h1 id="myh2">Heading 2</h2>
            <h1 id="myh3">Heading 3</h3>
        </div>
    </body>
```



```
<script>
// targetElement.insertAdjacentElement
("position", element)
// position : 'beforebegin' | 'afterbegin' | 'beforeend' | 'afterend'
var parentNode = document.getElementById("mydiv");
var h = document.getElementById("myh2");
var newElement = document.createElement("span");
newElement.textContent = "This is span";
h.insertAdjacentElement('beforebegin', newElement);
h.insertAdjacentElement('afterend', newElement);
h.insertAdjacentElement('afterbegin', newElement);
h.insertAdjacentElement('beforeend', newElement);
console.log(parentNode);
</script>
</body>
</html>
```

Note:- Span tag

insertAdjacentHTML() Method:- The insertAdjacentHTML method is used to insert a text as HTML (parses the specified text as HTML or XML) into a specified position.

Date: ________ Page: 289

Syntax- element.insertAdjacentHTML('position', text);

where positions are-

- 'beforebegin' - Before the element itself.
- 'afterbegin' - Just inside the element, before its first
- 'beforeend' - Just inside the element, after its last child.
- 'afterend' - After the element itself.

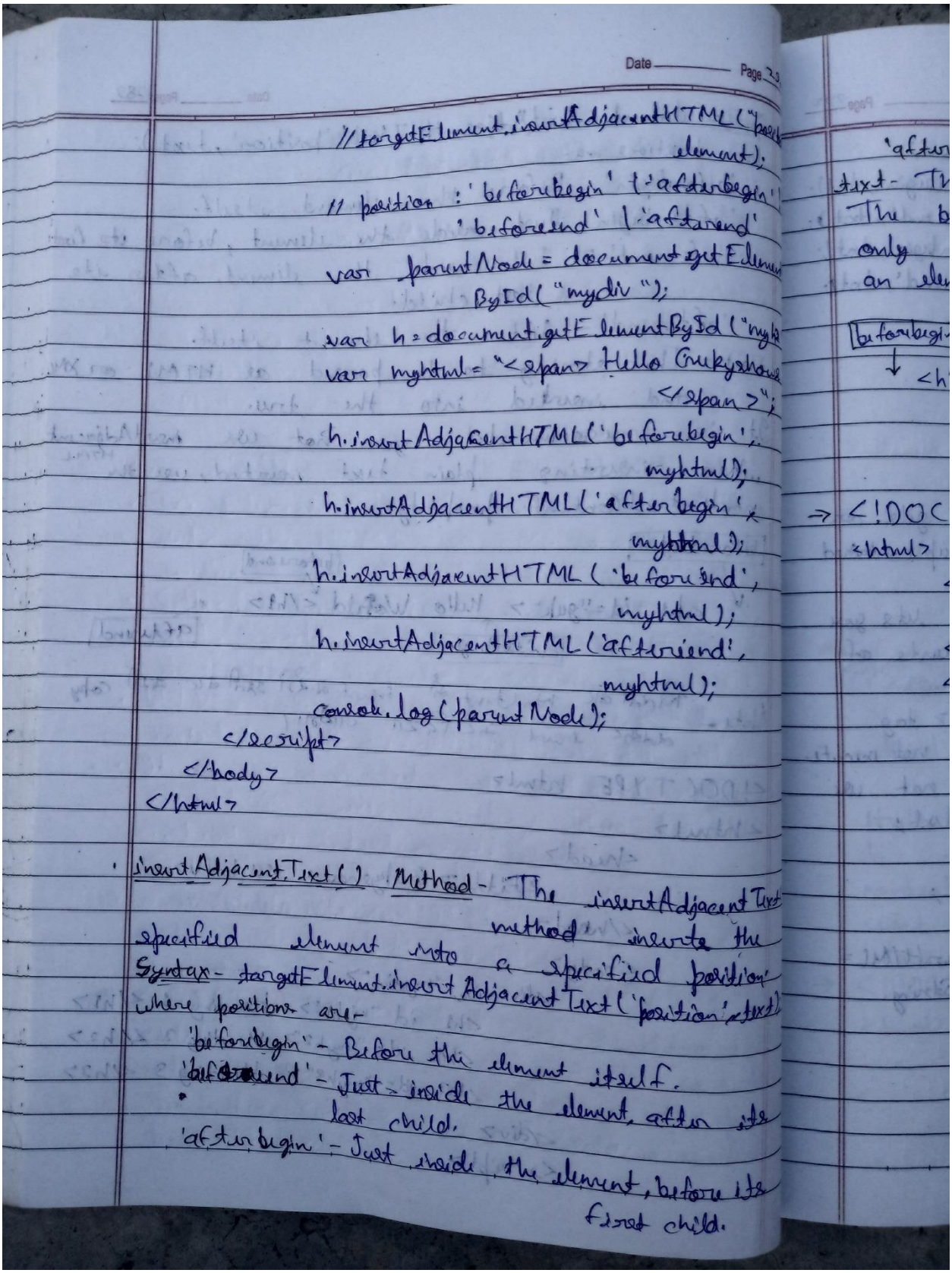
text - The string to be parsed as HTML or XML and inserted into the tree.

It is recommended - you not use insertAdjacentHTML when inserting plain text inserted, use the node.textContent property.

```
[beforebegin]                        [beforeend]
     ↓                                    ↓
     <h1 id="geek"> Hello World </h1>           ↖
                   ↑                          [afterend]
              [afterbegin]
```

Note- जितनी बार html text को insert करेंगे उतनी बार उसी copy बनाकर insert कर दिया जायेगा।

```
→ <!DOCTYPE html>
<html>
    <head>
        <title> Geekyshows </title>
    </head>
    <body>
        <div id="mydiv">
            <h1 id="myh1"> Heading 1 </h1>
            <h2 id="myh2"> Heading 2 </h2>
            <h3 id="myh3"> Heading 3 </h3>
        </div>
        <script>
```



```
//targetElement.insertAdjacentHTML("position",
                                    element);
// position : 'beforebegin' | 'afterbegin'
              'beforeend' | 'afterend'
var parentNode = document.getElementById("mydiv");
var h = document.getElementById("myh2");
var myhtml = "<span> Hello Geekyshows
                                    </span>";
h.insertAdjacentHTML('beforebegin',
                                    myhtml);
h.insertAdjacentHTML('afterbegin',
                                    myhtml);
h.insertAdjacentHTML('beforeend',
                                    myhtml);
h.insertAdjacentHTML('afterend',
                                    myhtml);
console.log(parentNode);
</script>
</body>
</html>
```

insertAdjacentText() Method - The insertAdjacentText method inserts the specified element into a specified position.

Syntax - targetElement.insertAdjacentText('position', text)

where positions are -

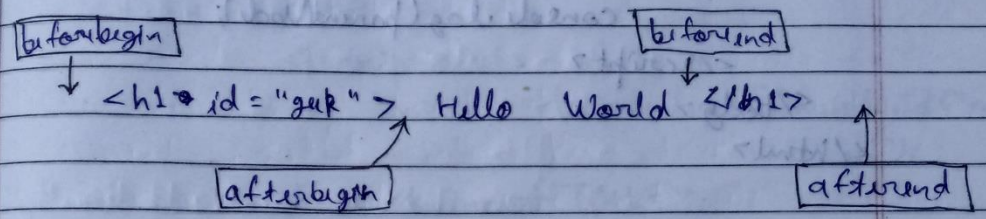
'beforebegin' - Before the element itself.

'beforeend' - Just inside the element, after its last child.

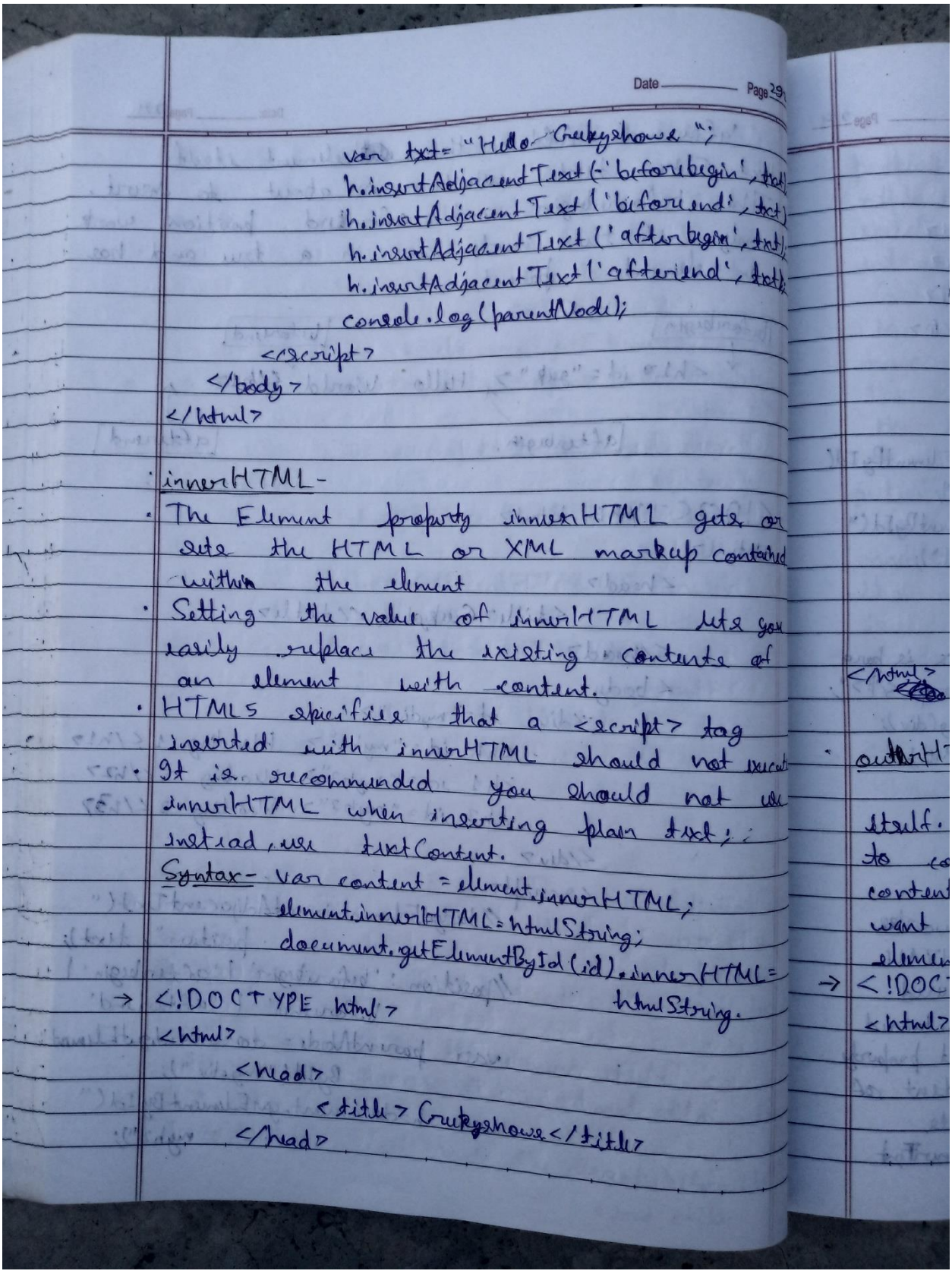
'afterbegin' - Just inside the element, before its first child.

Date: Page: 291

'afterend' - After the element itself.
text - The text which is about to insert.
The beforebegin and afterend positions work only if the node is in a tree and has an element parent.



→
```
<!DOCTYPE html>
<html>
    <head>
        <title> Geekyshows </title>
    </head>
    <body>
        <div id = "mydiv">
            <h1 id = "myh1"> Heading 1 </h1>
            <h1 id = "myh2"> Heading 2 </h2>
            <h3 id = "myh3"> Heading 3 </h3>
        </div>
        <script>
            // targetElement.insertAdjacentText("
                    position", text);
            //position : 'beforebegin' | 'afterbegin' |
                    'beforeend' | 'afterend'
            var parentNode = document.getElement
                    ById("mydiv");
            var h=document.getElementById("
                    myh2");
```



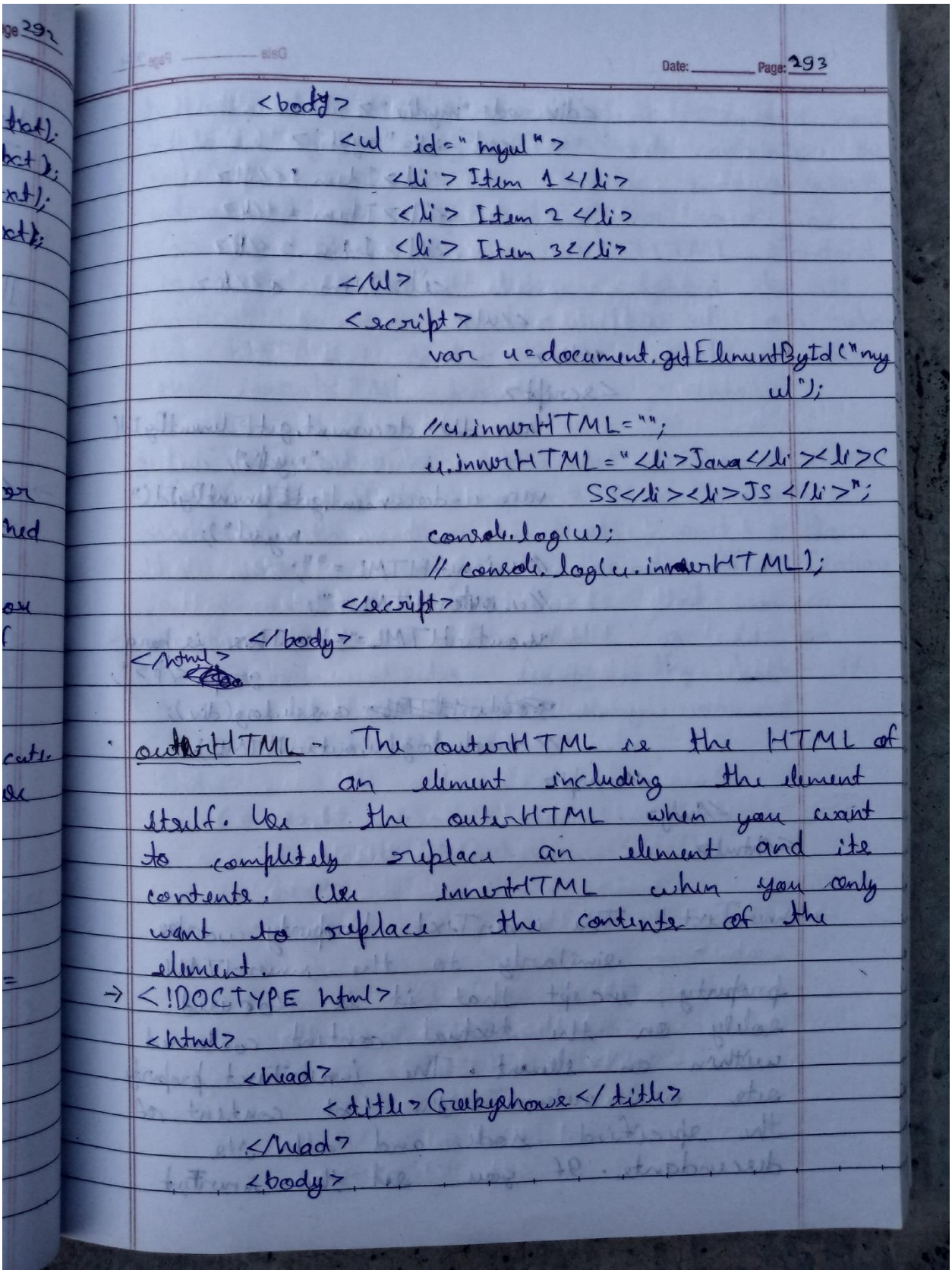
```
var txt= "Hello Geekyshows";
h.insertAdjacentText('beforebegin', txt);
h.insertAdjacentText('beforeend', txt);
h.insertAdjacentText('afterbegin', txt);
h.insertAdjacentText('afterend', txt);
console.log(parentNode);
    </script>
  </body>
</html>
```

innerHTML-

- The Element property innerHTML gets or sets the HTML or XML markup contained within the element
- Setting the value of innerHTML lets you easily replace the existing contents of an element with content.
- HTML5 specifies that a <script> tag inserted with innerHTML should not execute.
- It is recommended you should not use innerHTML when inserting plain text; instead, use textContent.

Syntax- var content = element.innerHTML;
element.innerHTML = htmlString;
document.getElementById(id).innerHTML = htmlString.

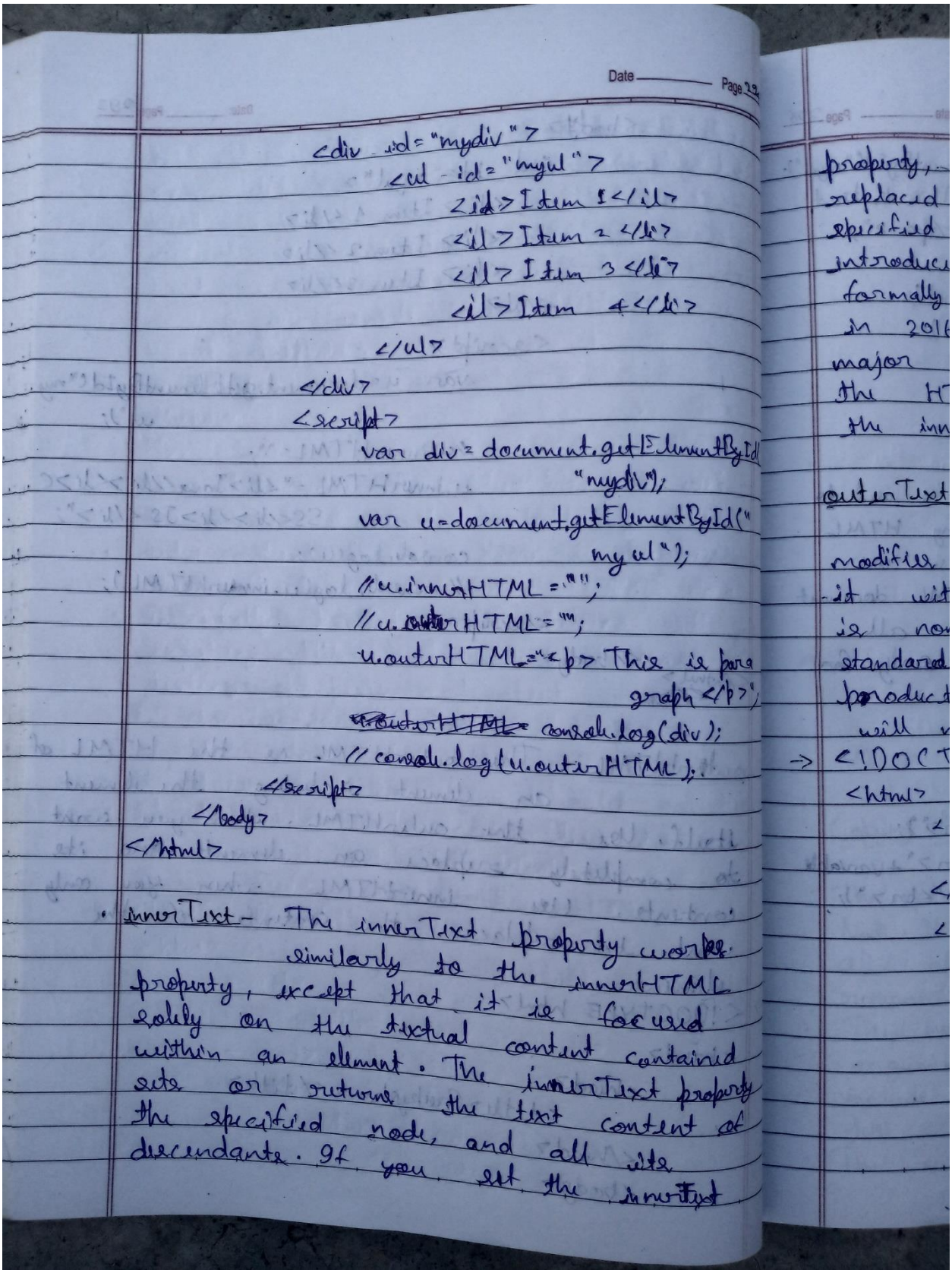
```
<!DOCTYPE html>
<html>
  <head>
    <title> Geekyshows </title>
  </head>
```



```
<body>
    <ul id="myul">
        <li> Item 1 </li>
        <li> Item 2 </li>
        <li> Item 3 </li>
    </ul>
    <script>
        var u = document.getElementById("myul");
        //u.innerHTML="";
        u.innerHTML = "<li>Java</li><li>CSS</li><li>JS</li>";
        console.log(u);
        // console.log(u.innerHTML);
    </script>
</body>
</html>
```

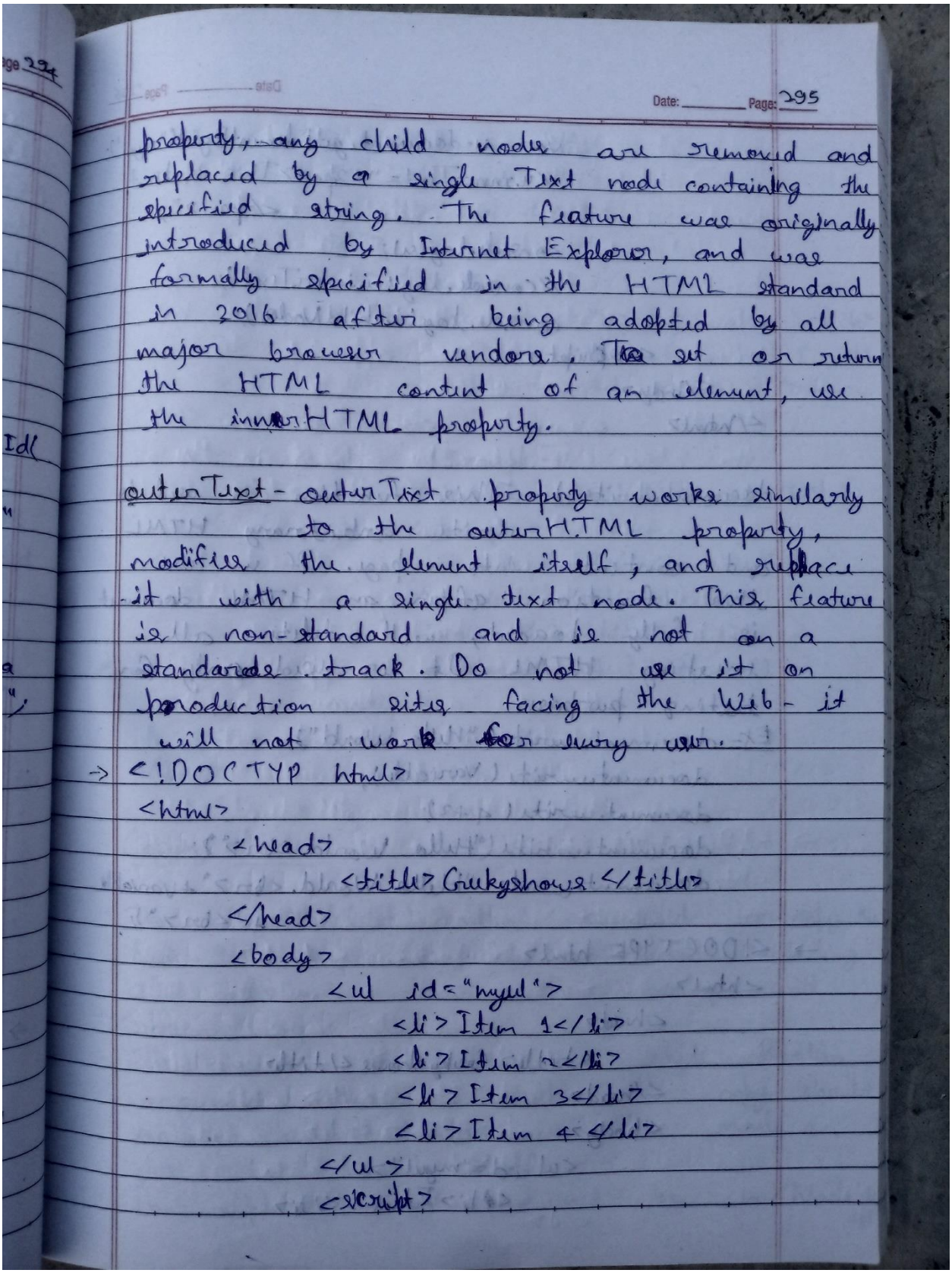
outerHTML - The outerHTML is the HTML of an element including the element itself. Use the outerHTML when you want to completely replace an element and its contents. Use innerHTML when you only want to replace the contents of the element.

```
<!DOCTYPE html>
<html>
    <head>
        <title>Geekyshows</title>
    </head>
    <body>
```



```
<div id="mydiv">
    <ul id="myul">
        <li>Item 1</li>
        <li>Item 2</li>
        <li>Item 3</li>
        <li>Item 4</li>
    </ul>
</div>
<script>
    var div = document.getElementById("mydiv");
    var u = document.getElementById("myul");
    //u.innerHTML = "";
    //u.outerHTML = "";
    u.outerHTML = "<p> This is paragraph </p>";
    console.log(div);
    //console.log(u.outerHTML);
</script>
</body>
</html>
```

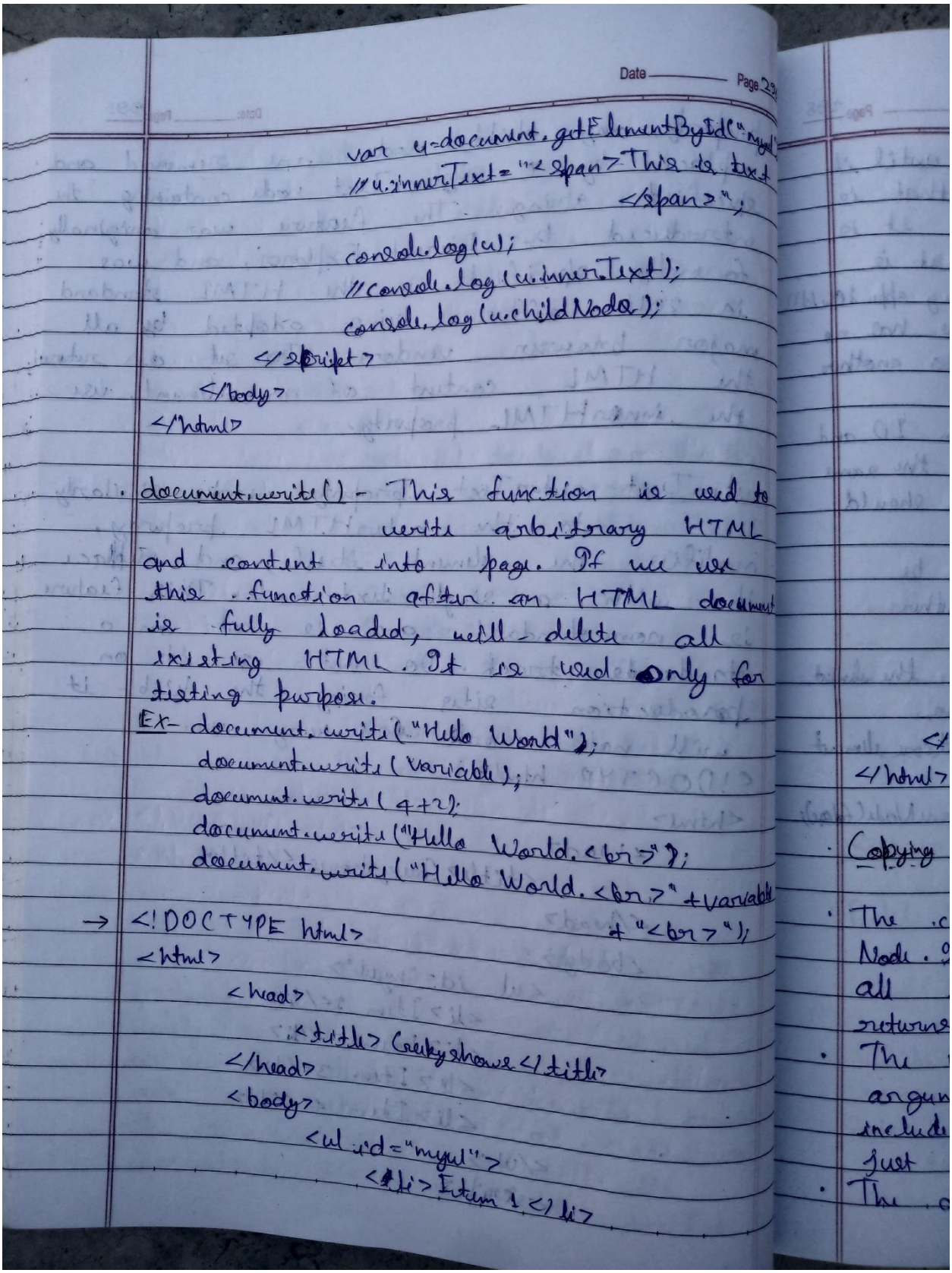
- innerText - The innerText property works similarly to the innerHTML property, except that it is focused solely on the textual content contained within an element. The innerText property sets or returns the text content of the specified node, and all its descendants. If you set the innerText



property, any child nodes are removed and replaced by a single Text node containing the specified string. The feature was originally introduced by Internet Explorer, and was formally specified in the HTML standard in 2016 after being adopted by all major browser vendors. To set or return the HTML content of an element, use the innerHTML property.

outerText - outerText property works similarly to the outerHTML property, modifies the element itself, and replaces it with a single text node. This feature is non-standard and is not on a standards track. Do not use it on production sites facing the Web - it will not work for every user.

```
<!DOCTYP html>
<html>
    <head>
        <title> Geekyshows </title>
    </head>
    <body>
        <ul id="myul">
            <li> Item 1</li>
            <li> Item 2</li>
            <li> Item 3</li>
            <li> Item 4 </li>
        </ul>
        <script>
```

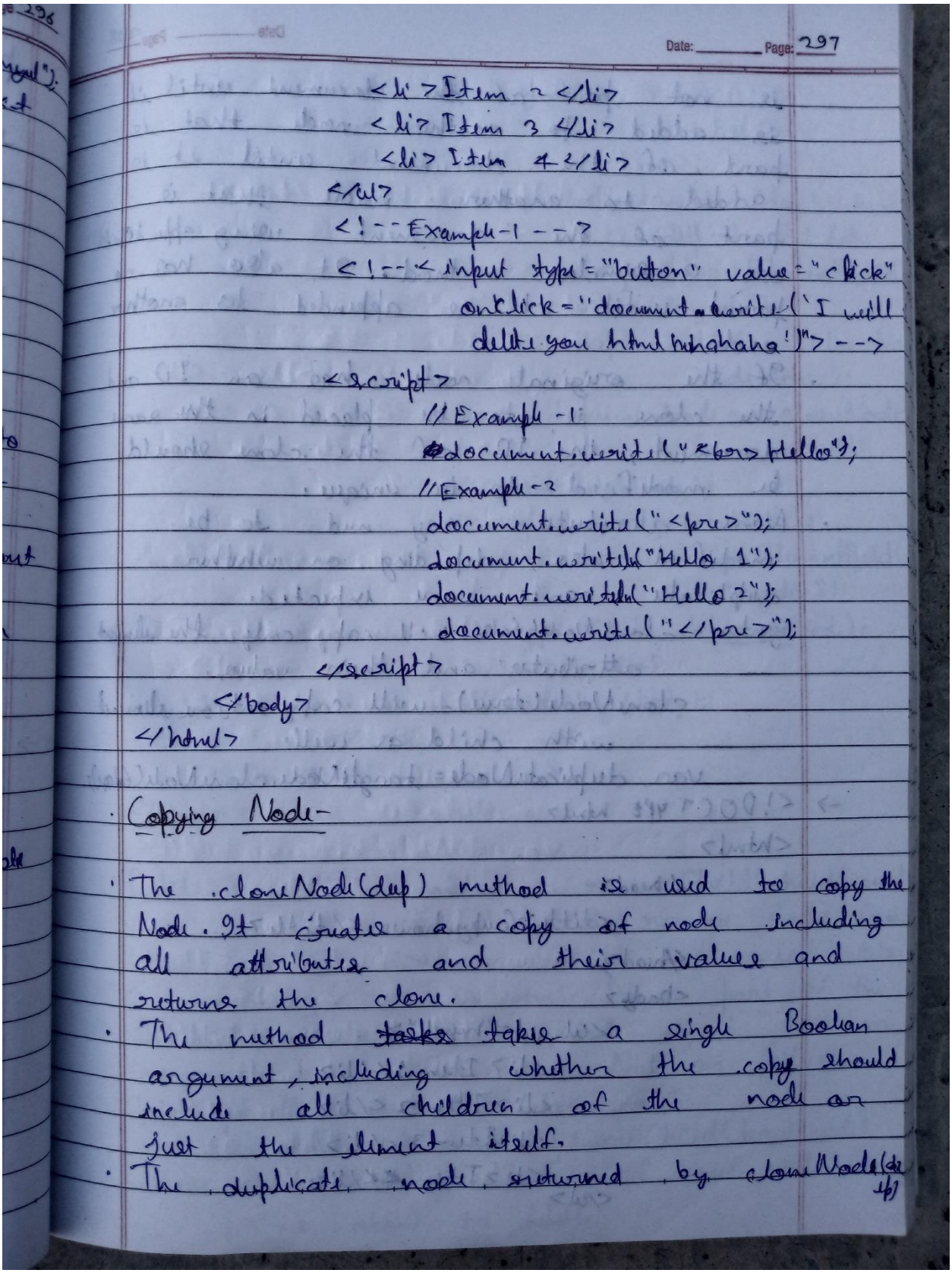


```
var u=document.getElementById("myul");
//u.innerText="<span>This is text
                </span>";
console.log(u);
//console.log(u.innerText);
console.log(u.childNodes);
</script>
</body>
</html>
```

document.write() - This function is used to write arbitrary HTML and content into page. If we use this function after an HTML document is fully loaded, it will delete all existing HTML. It is used only for testing purpose.

Ex- document.write("Hello World");
document.write(variable);
document.write(4+2);
document.write("Hello World.
");
document.write("Hello World.
"+variable+"
");

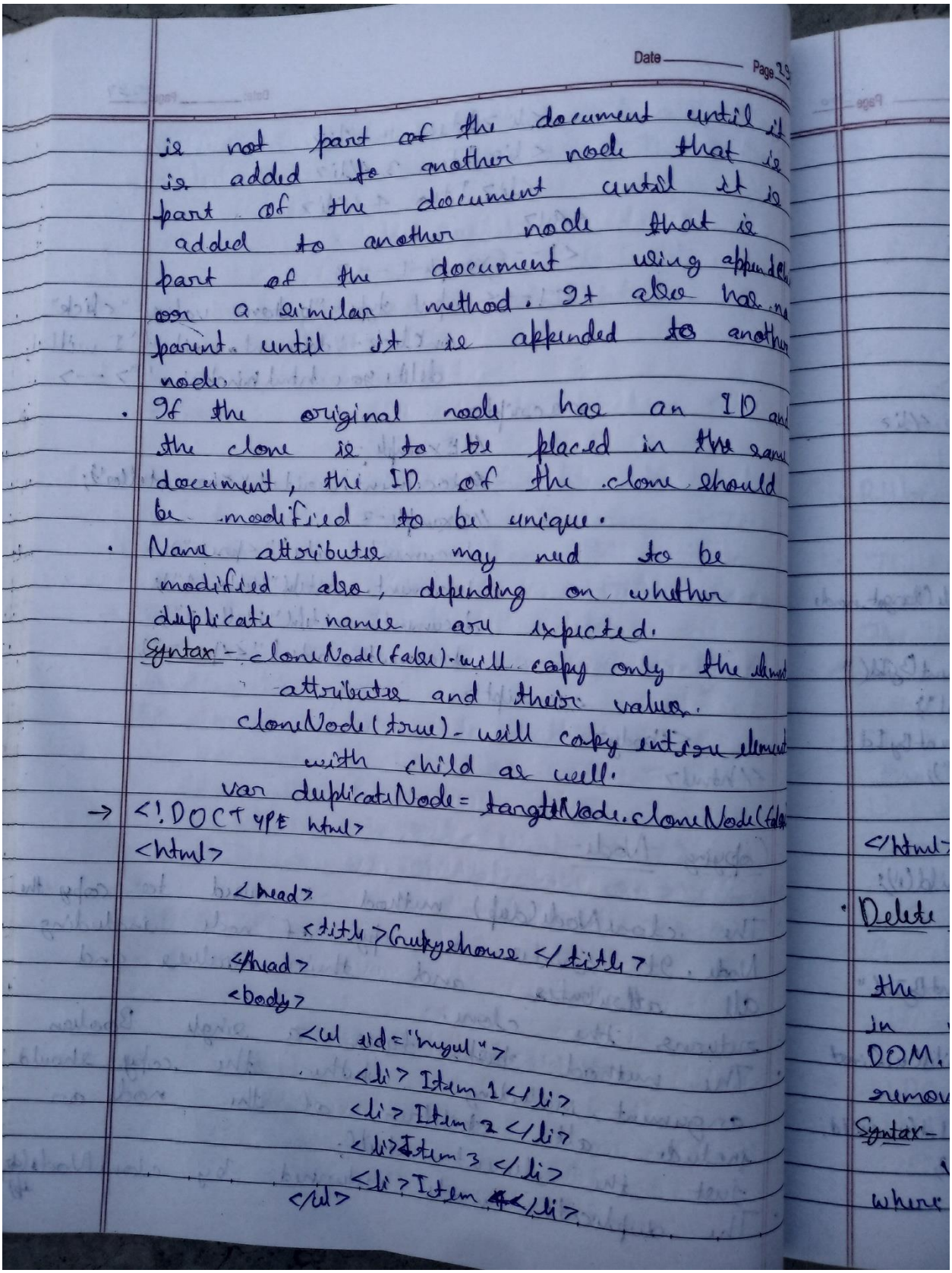
```
<!DOCTYPE html>
<html>
    <head>
        <title>Geekyshows</title>
    </head>
    <body>
        <ul id="myul">
            <li>Item 1</li>
```



```
        <li> Item 2 </li>
        <li> Item 3 </li>
        <li> Item 4 </li>
    </ul>
    <!-- Example-1 -->
    <!-- <input type="button" value="Click"
        onclick="document.write('I will
        delete you html hahaha!')"> -->
    <script>
        //Example-1
        document.write("<br> Hello");
        //Example-2
        document.write("<pre>");
        document.write("Hello 1");
        document.write("Hello 2");
        document.write("</pre>");
    </script>
</body>
</html>
```

Copying Node-

- The .cloneNode(dup) method is used to copy the Node. It creates a copy of node including all attributes and their values and returns the clone.
- The method ~~tasks~~ takes a single Boolean argument, including whether the copy should include all children of the node or just the element itself.
- The duplicate node returned by cloneNode(dup)



is not part of the document until it is added to another node that is part of the document until it is added to another node that is part of the document using appendChild() or a similar method. It also has no parent until it is appended to another node.

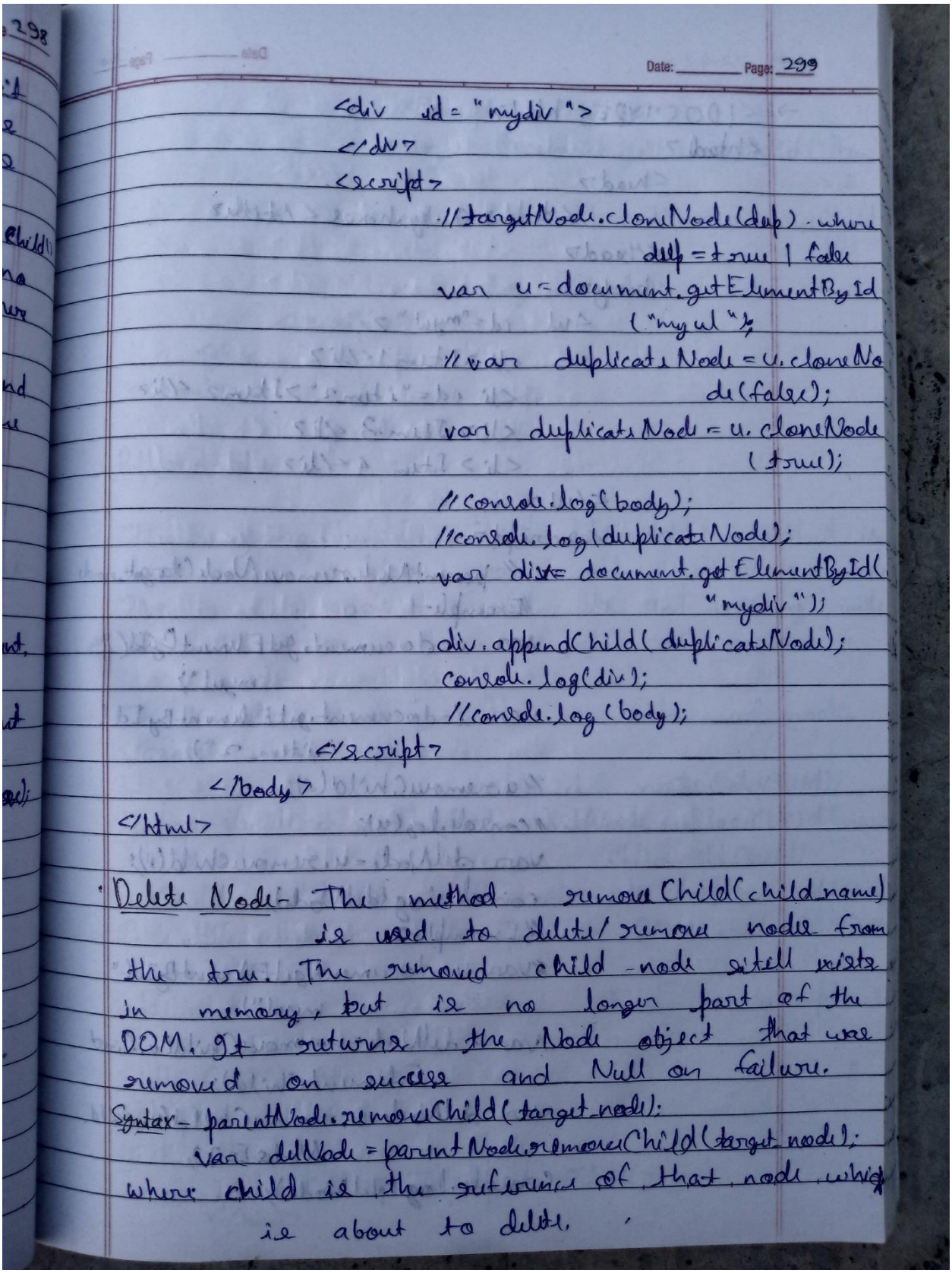
- If the original node has an ID and the clone is to be placed in the same document, the ID of the clone should be modified to be unique.
- Name attributes may need to be modified also, depending on whether duplicate names are expected.

Syntax - cloneNode(false) - will copy only the element attributes and their values.
cloneNode(true) - will copy entire element with child as well.
var duplicateNode = targetNode.cloneNode(false);

```
<!DOCTYPE html>
<html>
    <head>
        <title>Geekyshows</title>
    </head>
    <body>
        <ul id="myul">
            <li>Item 1</li>
            <li>Item 2</li>
            <li>Item 3</li>
            <li>Item 4</li>
        </ul>
```

</html>

Delete the in DOM. remov Syntax- where



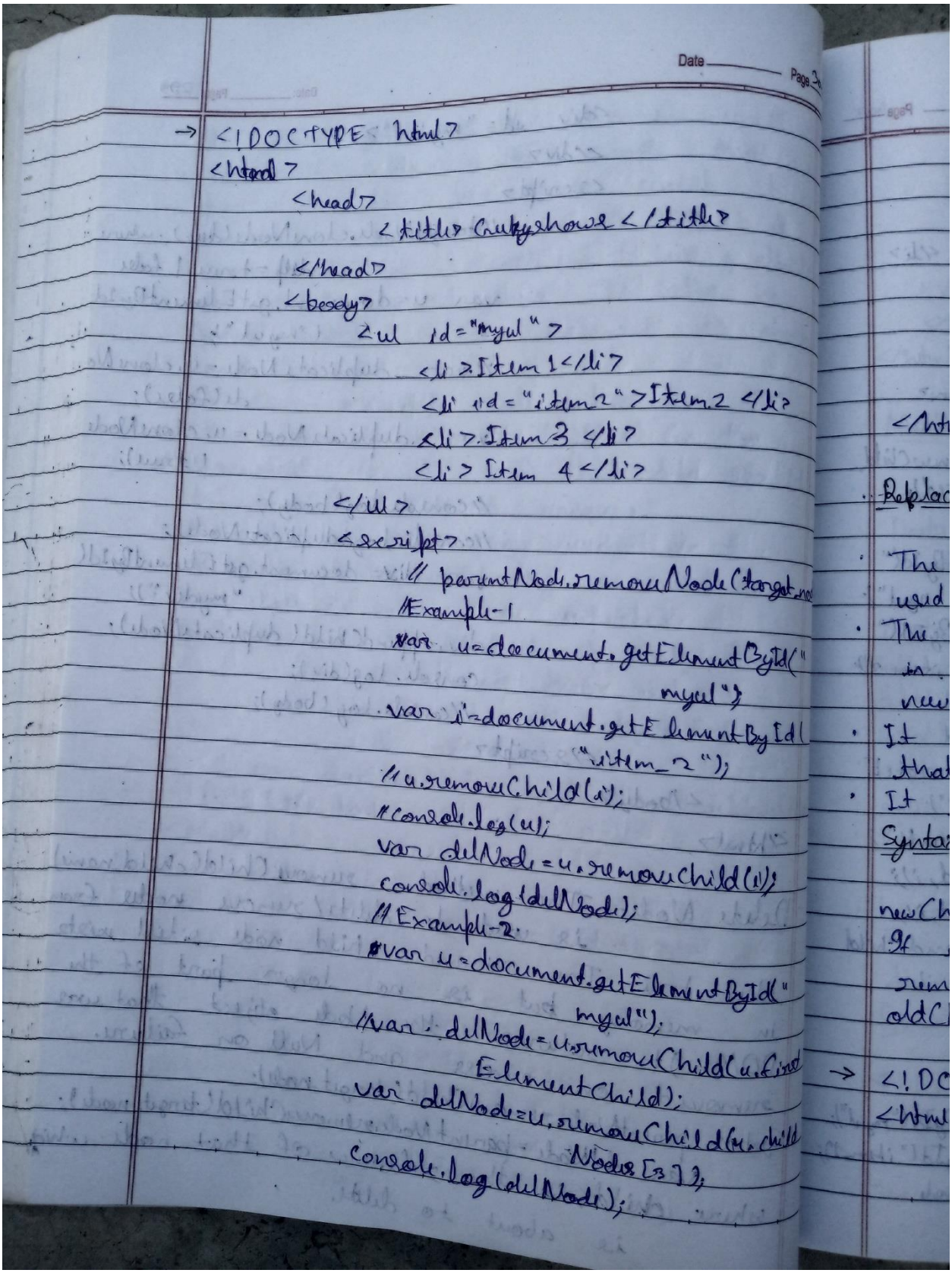
```
<div id = "mydiv">
</div>
<script>
    //targetNode.cloneNode(deep) . where
                deep = true | false
    var u = document.getElementById
            ("myul");
    //var duplicateNode = u.cloneNo
                de(false);
    var duplicateNode = u.cloneNode
                (true);
    //console.log(body);
    //console.log(duplicateNode);
    var div = document.getElementById(
                "mydiv");
    div.appendChild(duplicateNode);
    console.log(div);
    //console.log(body);
</script>
</body>
</html>
```

* Delete Node - The method removeChild(child_name) is used to delete/remove node from the tree. The removed child node still exists in memory, but is no longer part of the DOM. It returns the Node object that was removed on success and Null on failure.

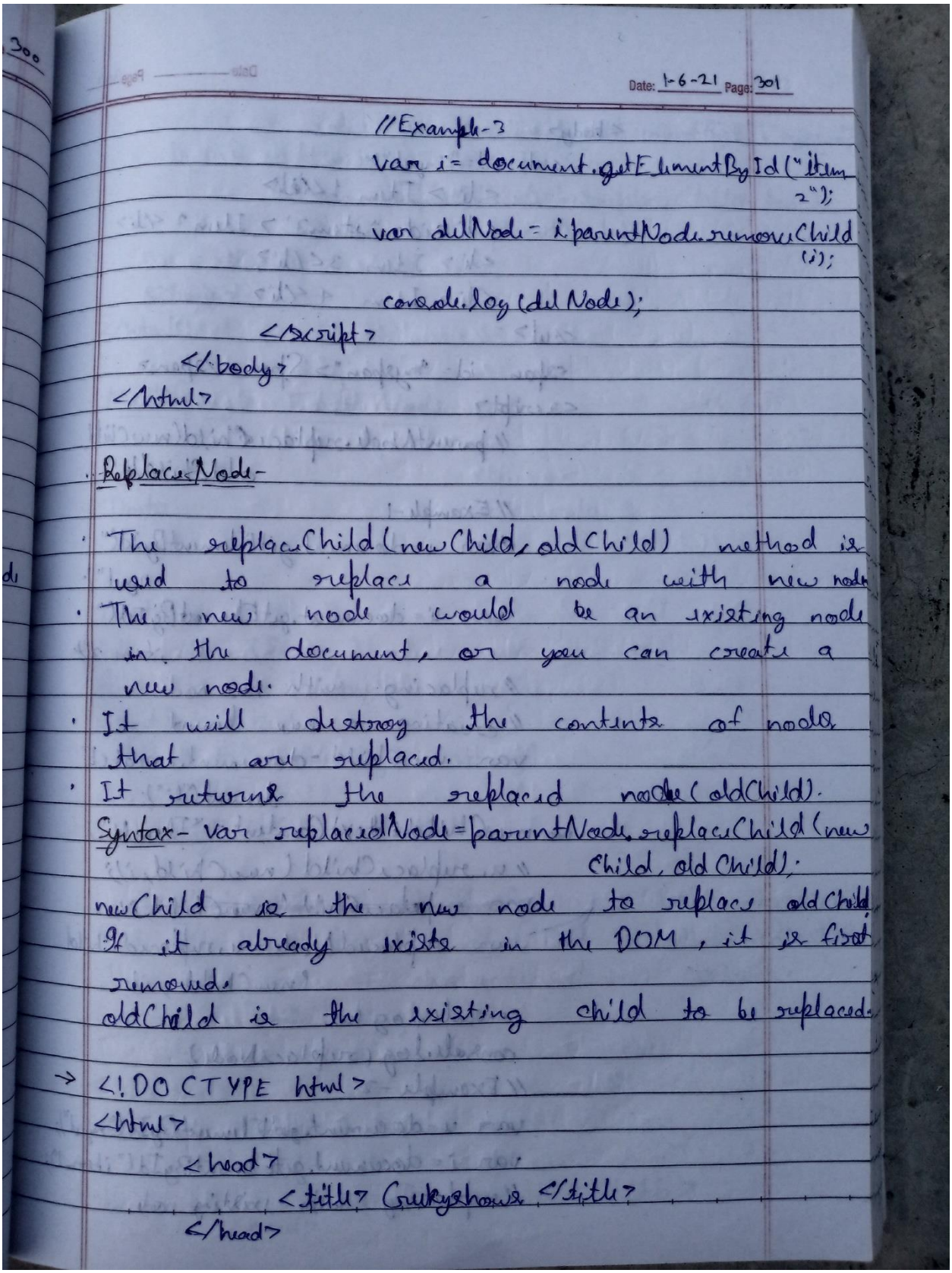
Syntax - parentNode.removeChild(target_node);

var delNode = parentNode.removeChild(target_node);

where child is the reference of that node which is about to delete.



```
<!DOCTYPE html>
<html>
    <head>
        <title> Geekyshows </title>
    </head>
    <body>
        <ul id="myul">
            <li> Item 1 </li>
            <li id="item_2"> Item 2 </li>
            <li> Item 3 </li>
            <li> Item 4 </li>
        </ul>
        <script>
            // parentNode.removeNode(target.no
            //Example-1
            var u=document.getElementById("
                                myul");
            var i=document.getElementById(
                                "item_2");
            //u.removeChild(i);
            //console.log(u);
            var delNode=u.removeChild(i);
            console.log(delNode);
            //Example-2.
            //var u=document.getElementById("
                                myul");
            //var delNode=u.removeChild(u.first
                                ElementChild);
            var delNode=u.removeChild(u.childE
                                Nodes[3]);
            console.log(delNode);
```



```
//Example-3
var i= document.getElementById("item
                                2");
var delNode = i.parentNode.removeChild
                                (i);
    console.log(delNode);
</script>
</body>
</html>
```

Replace Node-

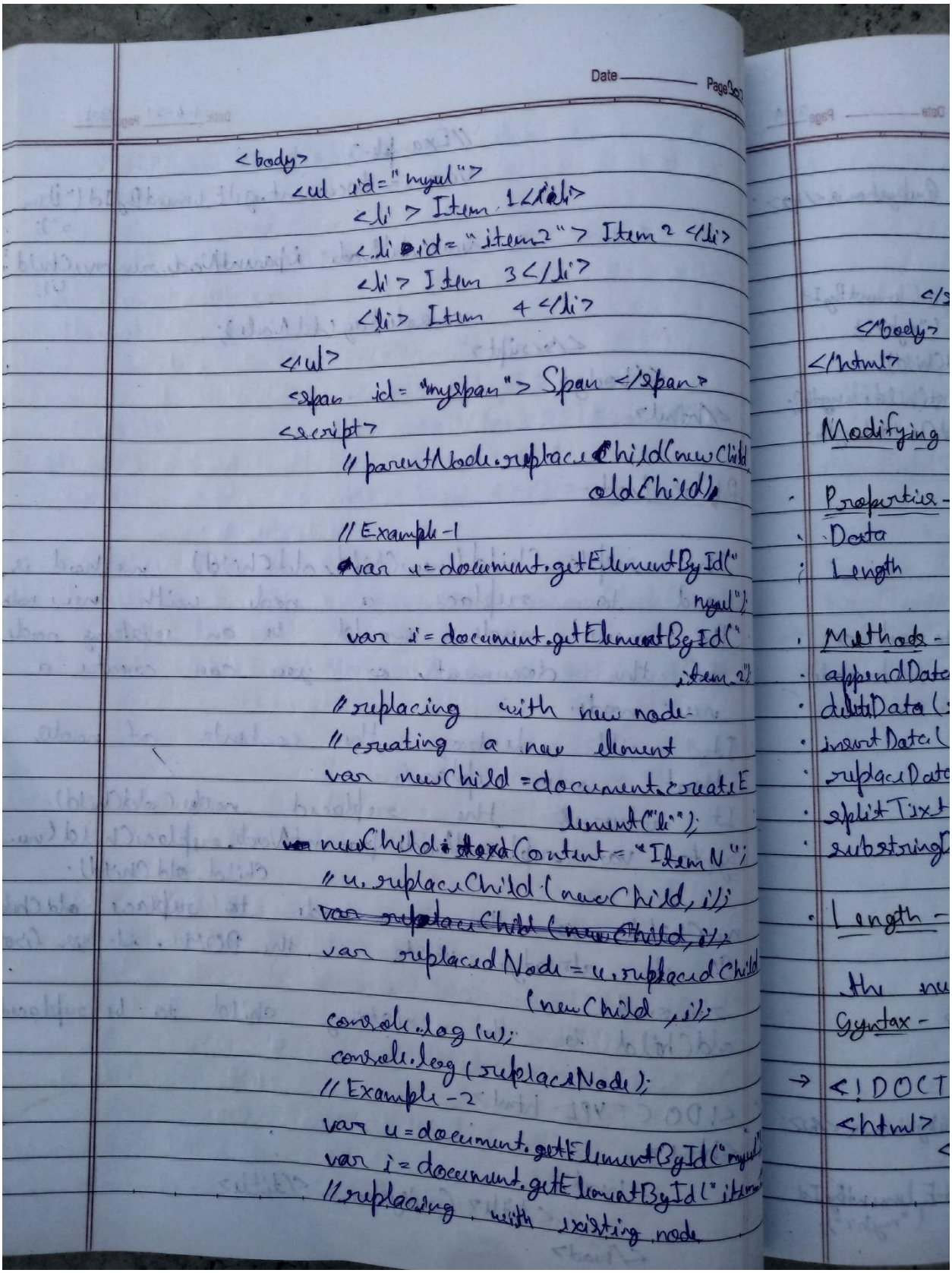
- The replaceChild(newChild, oldChild) method is used to replace a node with new node.
- The new node would be an existing node in the document, or you can create a new node.
- It will destroy the contents of nodes that are replaced.
- It returns the replaced node(oldChild).

Syntax- var replacedNode = parentNode.replaceChild(new Child, old Child);

newChild is the new node to replace oldChild. If it already exists in the DOM, it is first removed.

oldChild is the existing child to be replaced.

```
<!DOCTYPE html>
<html>
    <head>
        <title> Geekyshows </title>
    </head>
```



```
<body>
    <ul id="myul">
        <li> Item 1</li>
        <li id="item2"> Item 2 </li>
        <li> Item 3</li>
        <li> Item 4 </li>
    </ul>
    <span id="myspan"> Span </span>
    <script>
        // parentNode.replaceChild(newChild, oldChild);

        // Example-1
        var u = document.getElementById("myul");
        var i = document.getElementById("item2");
        // replacing with new node
        // creating a new element
        var newChild = document.createElement("li");
        newChild.textContent = "Item N";
        // u.replaceChild(newChild, i);
        var replacedNode = u.replaceChild(newChild, i);
        console.log(u);
        console.log(replacedNode);
        // Example-2
        var u = document.getElementById("myul");
        var i = document.getElementById("item2");
        // replacing with existing node
```

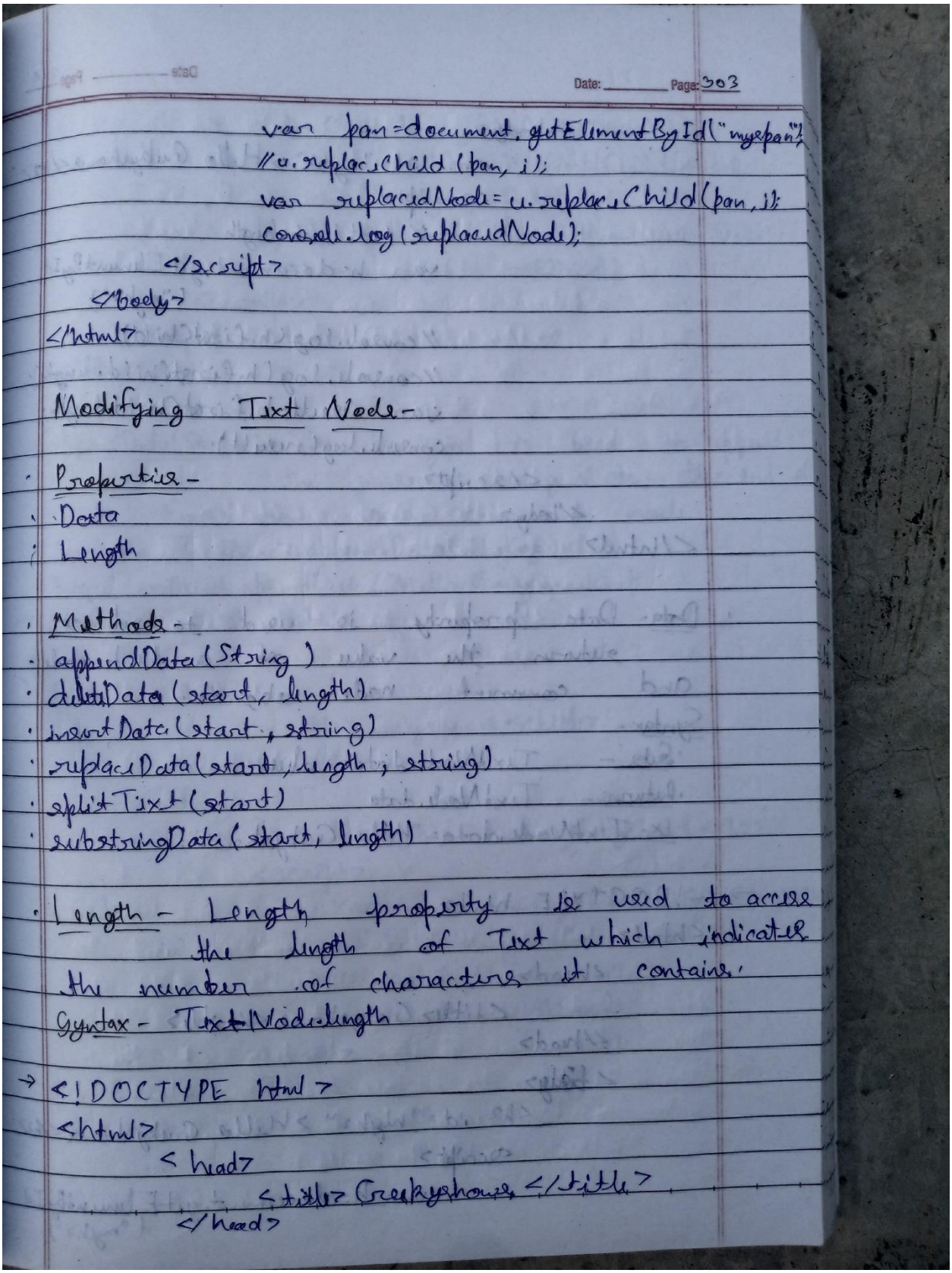
Modifying

- Properties
  - Data
  - Length
- Methods
  - appendData
  - deleteData
  - insertData
  - replaceData
  - splitText
  - substringData
- Length

the nu

Syntax

```
<!DOCT
<html>
```



```
var pan=document.getElementById("mypan");
//u.replaceChild(pan, i);
var replacedNode=u.replaceChild(pan,i);
console.log(replacedNode);
</script>
</body>
</html>
```

Modifying Text Node -

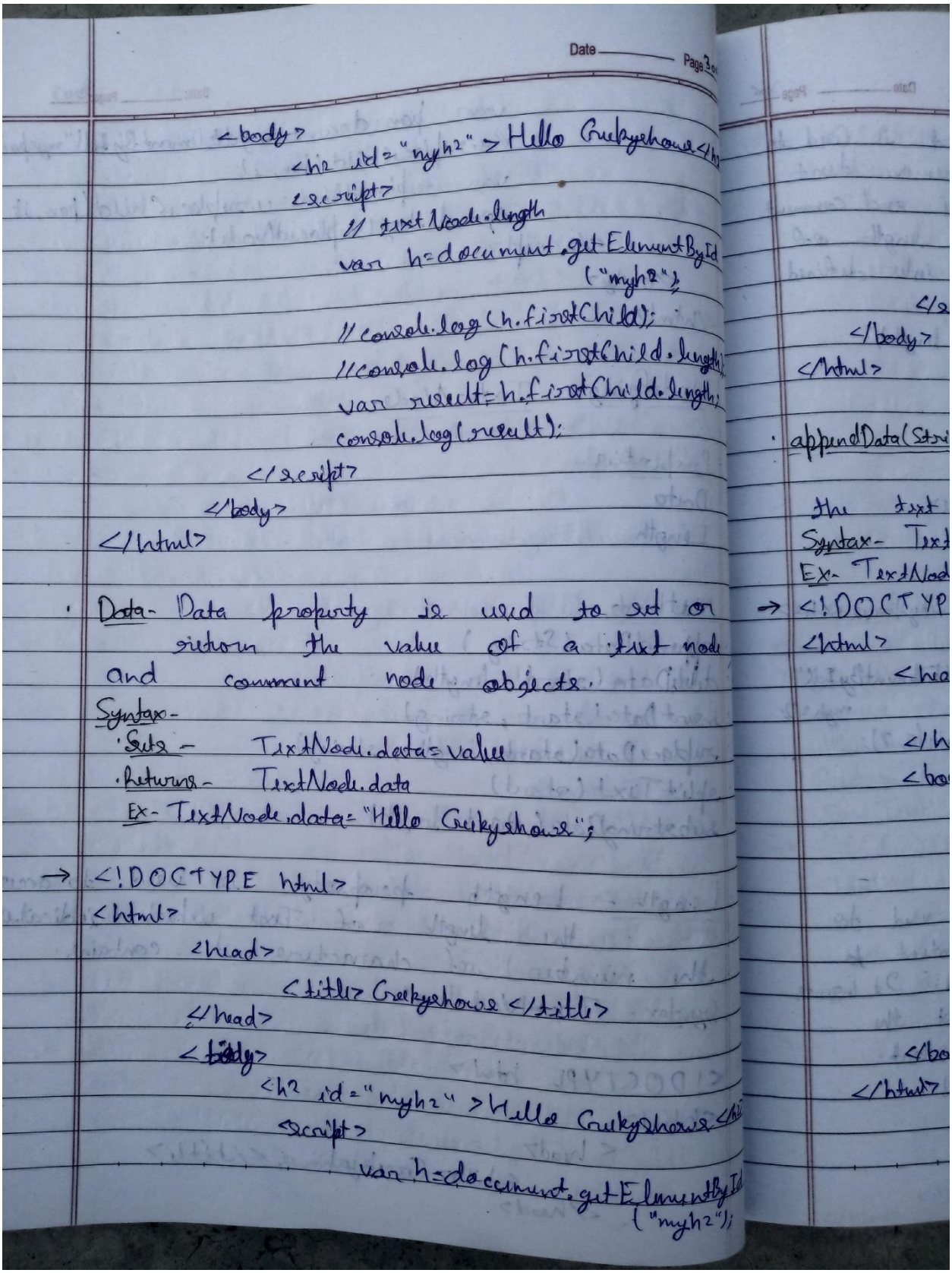
- Properties -
  - Data
  - Length

- Methods -
  - appendData(String)
  - deleteData(start, length)
  - insertData(start, string)
  - replaceData(start, length, string)
  - splitText(start)
  - substringData(start, length)

- Length - Length property is used to access the length of Text which indicates the number of characters it contains.

Syntax - TextNode.length

```
<!DOCTYPE html>
<html>
    <head>
        <title> Geekyshows </title>
    </head>
```



```
<body>
    <h2 id="myh2">Hello Geekyshows</h2>
    <script>
        // textNode.length
        var h=document.getElementById
                ("myh2");

        //console.log(h.firstChild);
        //console.log(h.firstChild.length);
        var result=h.firstChild.length;
        console.log(result);
    </script>
</body>
</html>
```

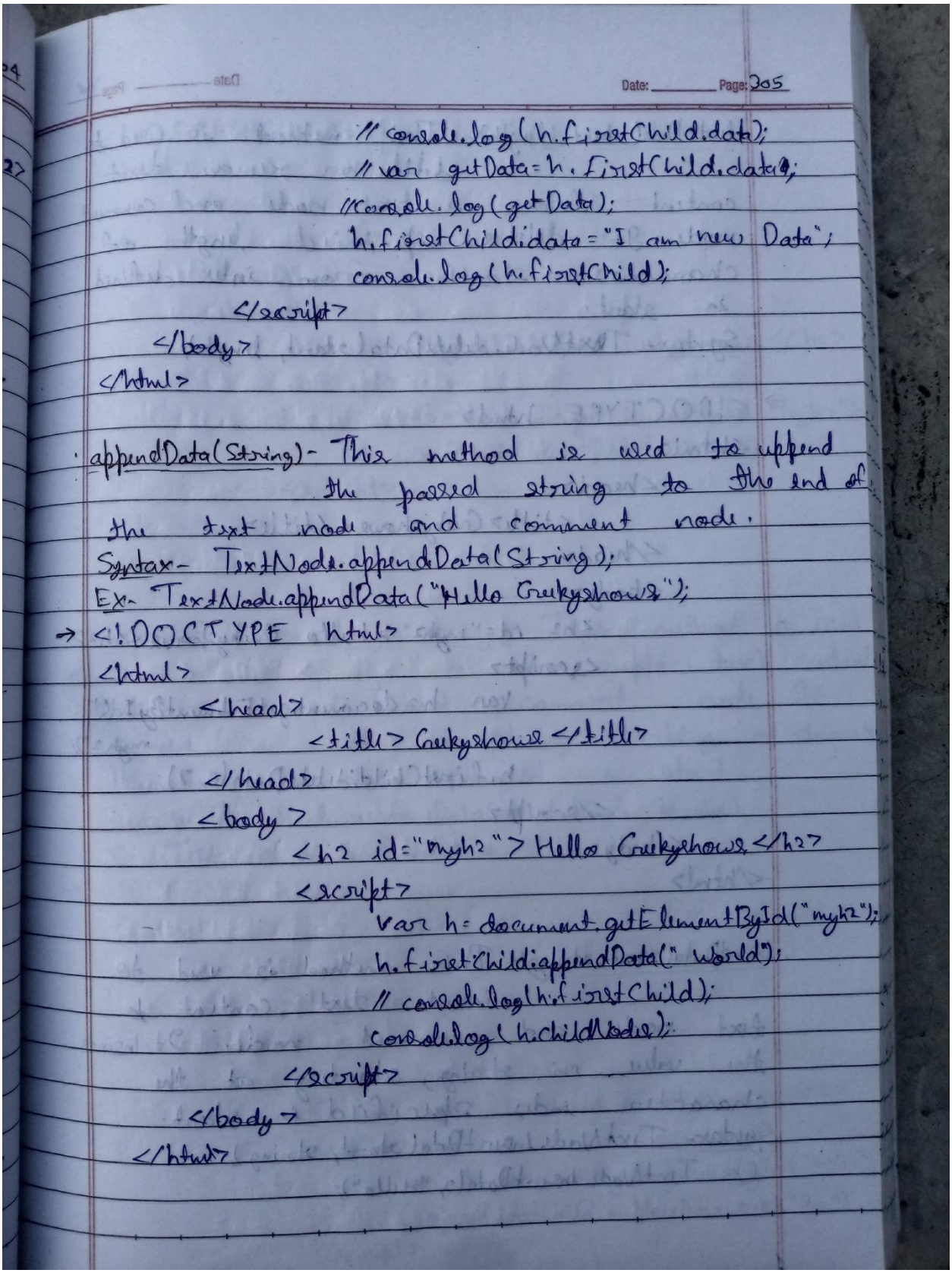
- **Data** - Data property is used to set or return the value of a text node and comment node objects.

Syntax -

- Sets - TextNode.data=value
- Returns - TextNode.data

Ex - TextNode.data="Hello Geekyshows";

```
<!DOCTYPE html>
<html>
    <head>
        <title>Geekyshows</title>
    </head>
    <body>
        <h2 id="myh2">Hello Geekyshows</h2>
        <script>
            var h=document.getElementById
                ("myh2");
```



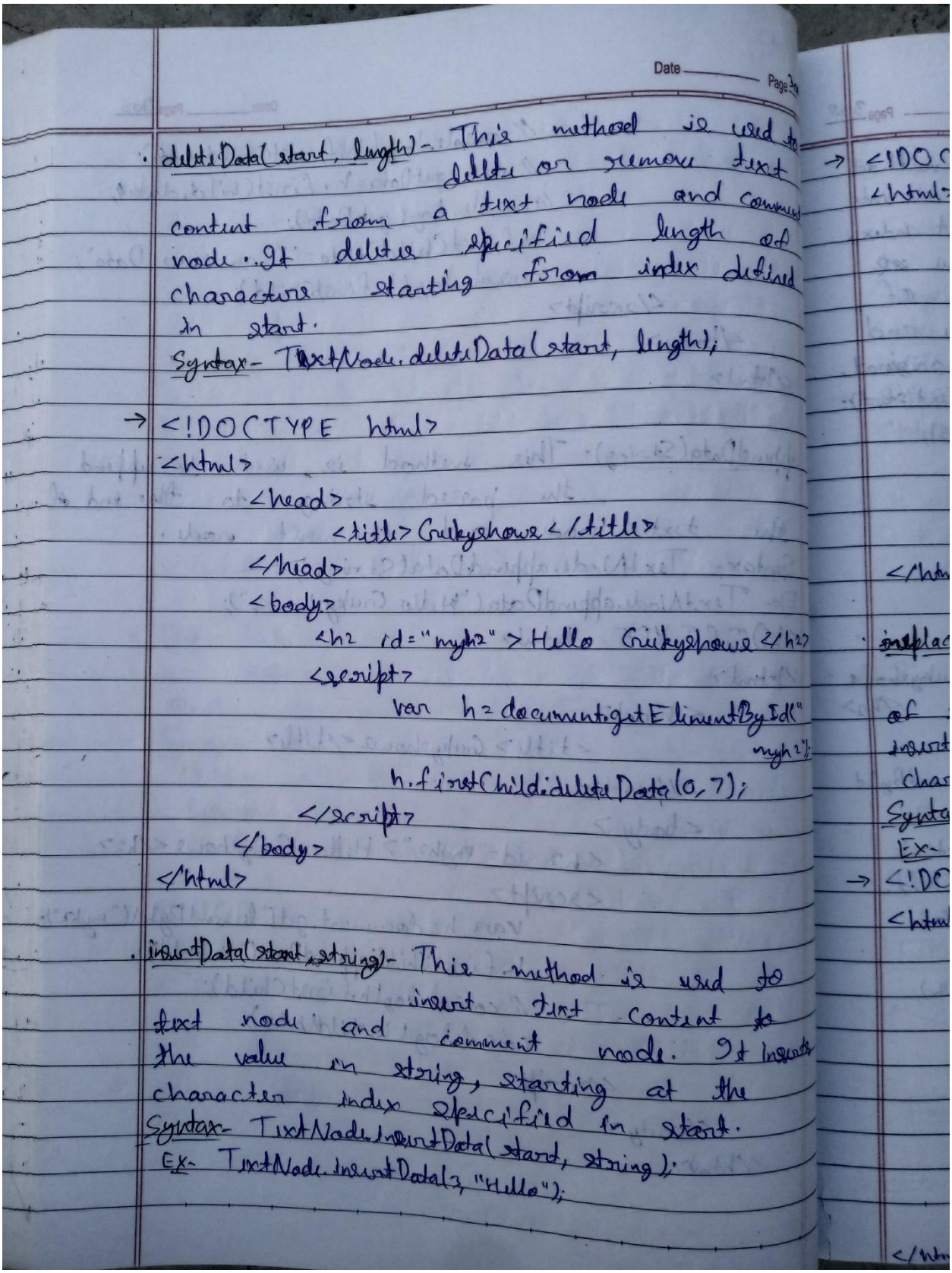
```
// console.log(h.firstChild.data);
// var getData=h.firstChild.data;
//console.log(getData);
h.firstChild.data="I am new Data";
console.log(h.firstChild);
</script>
</body>
</html>
```

appendData(String)- This method is used to append the passed string to the end of the text node and comment node.

Syntax- TextNode.appendData(String);

Ex- TextNode.appendData("Hello Geekyshows");

```
<!DOCTYPE html>
<html>
    <head>
        <title> Geekyshows </title>
    </head>
    <body>
        <h2 id="myh2"> Hello Geekyshows </h2>
        <script>
            var h=document.getElementById("myh2");
            h.firstChild.appendData(" World");
            // console.log(h.firstChild);
            console.log(h.childNodes);
        </script>
    </body>
</html>
```



• deleteData(start, length)- This method is used to delete or remove text content from a text node and comment node. It delete specified length of characters starting from index defined in start.

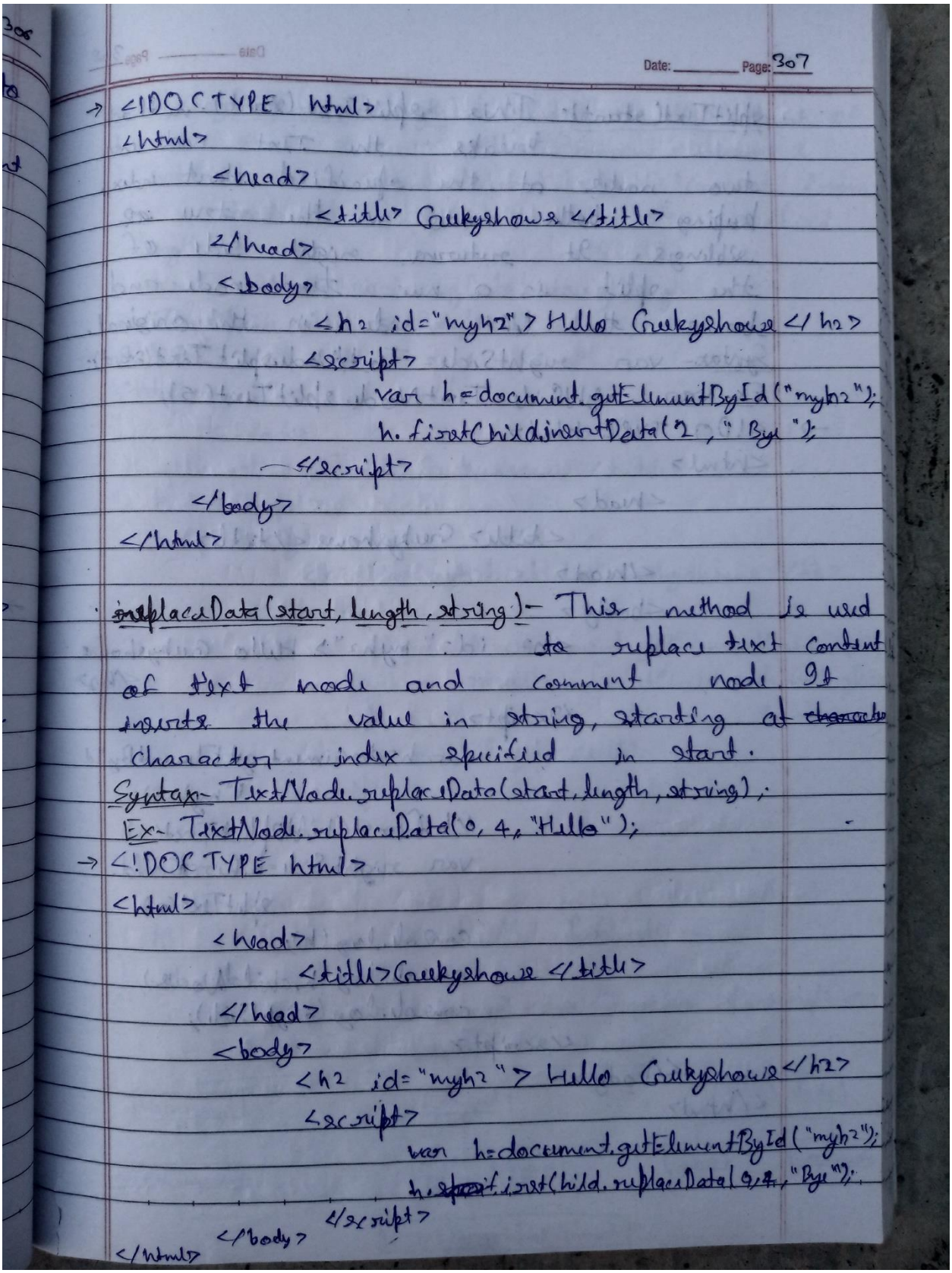
Syntax- TextNode.deleteData(start, length);

```
<!DOCTYPE html>
<html>
    <head>
        <title> Geekyshows </title>
    </head>
    <body>
        <h2 id="myh2"> Hello Geekyshows </h2>
        <script>
            var h = document.getElementById("myh2");
            h.firstChild.deleteData(0, 7);
        </script>
    </body>
</html>
```

• insertData(start, string)- This method is used to insert text content to text node and comment node. It inserts the value in string, starting at the character index specified in start.

Syntax- TextNode.insertData(start, string);

Ex- TextNode.insertData(3, "Hello");



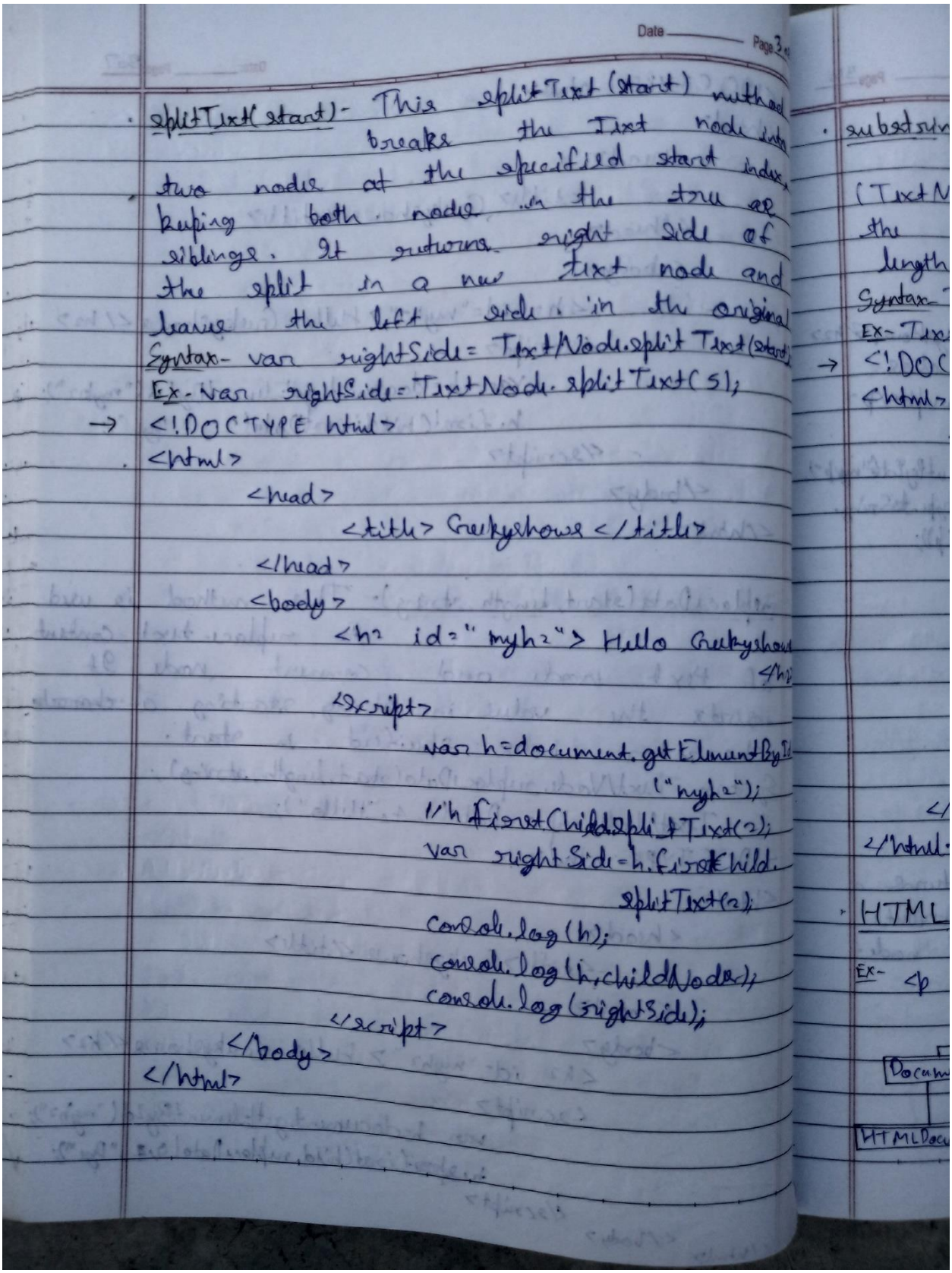
```
<!DOCTYPE html>
<html>
    <head>
        <title> Geekyshows </title>
    </head>
    <body>
        <h2 id="myh2"> Hello Geekyshows </h2>
        <script>
            var h=document.getElementById("myh2");
            h.firstChild.insertData(2, " Bye ");
        </script>
    </body>
</html>
```

replaceData(start, length, string) - This method is used to replace text content of text node and comment node. It inserts the value in string, starting at character index specified in start.

Syntax- TextNode.replaceData(start, length, string);

Ex- TextNode.replaceData(0, 4, "Hello");

```
<!DOCTYPE html>
<html>
    <head>
        <title>Geekyshows</title>
    </head>
    <body>
        <h2 id="myh2"> Hello Geekyshows</h2>
        <script>
            var h=document.getElementById("myh2");
            h.firstChild.replaceData(0,4, "Bye");
        </script>
    </body>
</html>
```



splitText(start)- This splitText(start) method breaks the Text node into two nodes at the specified start index, keeping both nodes in the tree as siblings. It returns right side of the split in a new Text node and leave the left side in the original

Syntax- var rightSide = TextNode.splitText(start);

Ex- var rightSide = TextNode.splitText(5);

```
<!DOCTYPE html>
<html>
    <head>
        <title> Geekyshows </title>
    </head>
    <body>
        <h2 id="myh2"> Hello Geekyshows
                                    </h2>
        <script>
            var h=document.getElementById
                                ("myh2");
            //h.firstChild.splitText(2);
            var rightSide=h.firstChild.
                                splitText(2);
            console.log(h);
            console.log(h.childNodes);
            console.log(rightSide);
        </script>
    </body>
</html>
```

Date: Page: 309

- substringData(start, length) - It returns the part of the current element's (TextNode and CommentNode) text content from the specified position with the specified length.

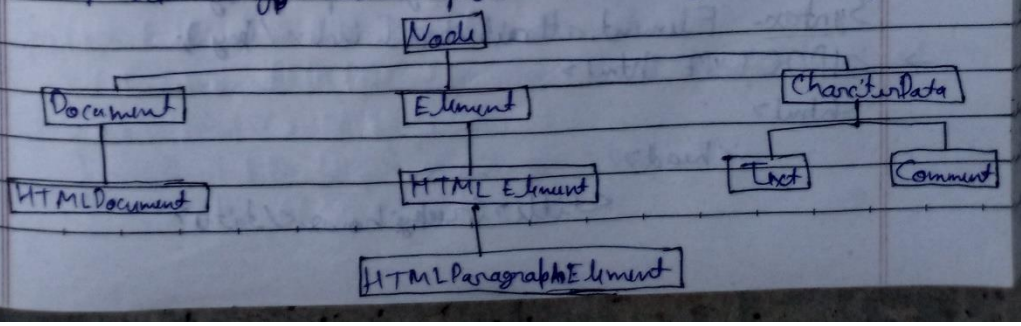
Syntax- TextNode.substringData(start, length);

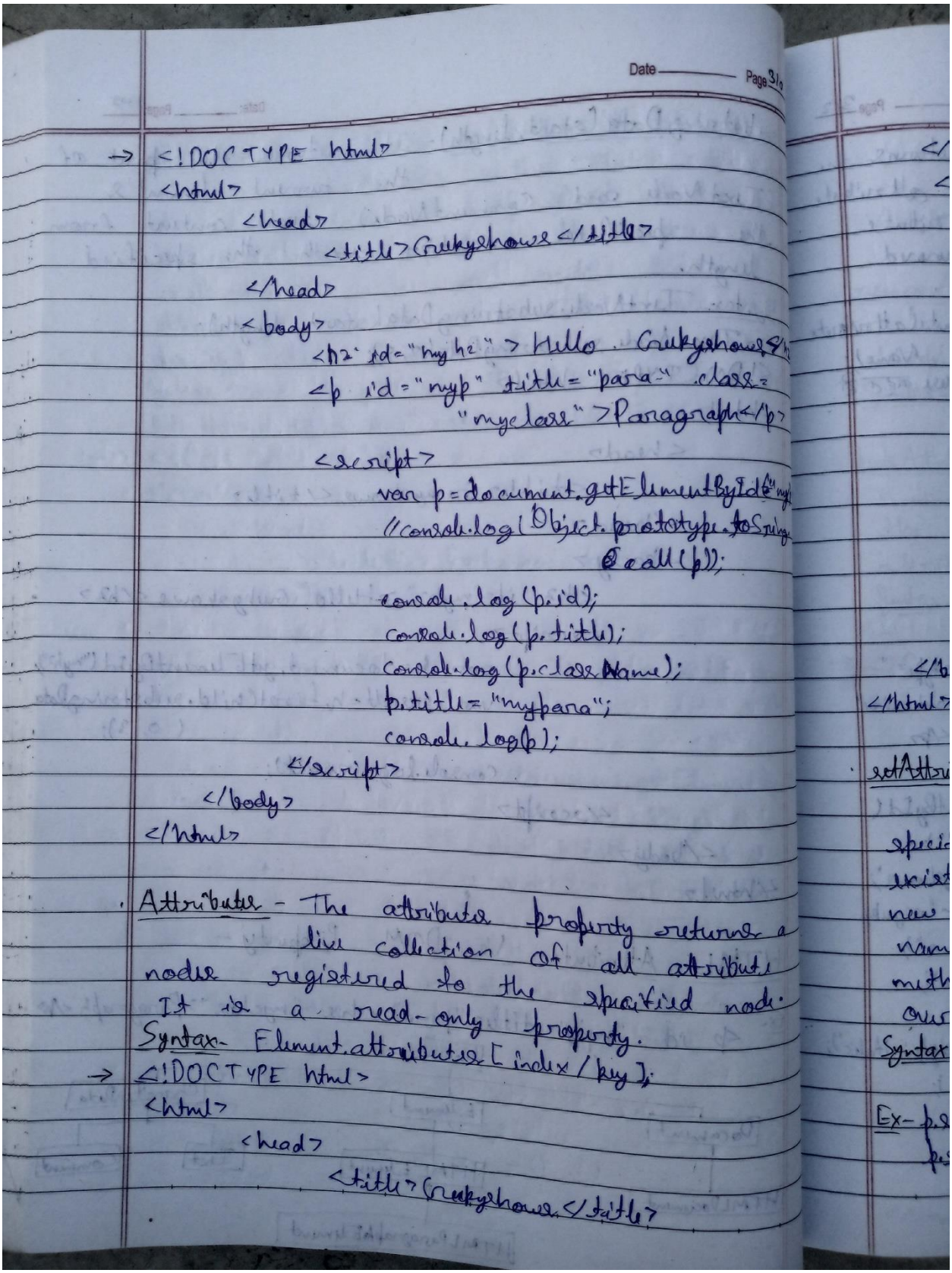
EX- TextNode.substringData(2, 5);

```
<!DOCTYPE html>
<html>
    <head>
        <title> Geekyshows </title>
    </head>
    <body>
        <h2 id="myh2" > Hello Geekyshows </h2>
        <script>
            var h = document.getElementById("myh2");
            var result = h.firstChild.substringData
                                    ( 0, 3);
            console.log(result);
        </script>
    </body>
</html>
```

- HTML Attribute Vs DOM Property -

EX- `<p id = "myp" title = "para" class = "myclass" > Paragraph </p>`



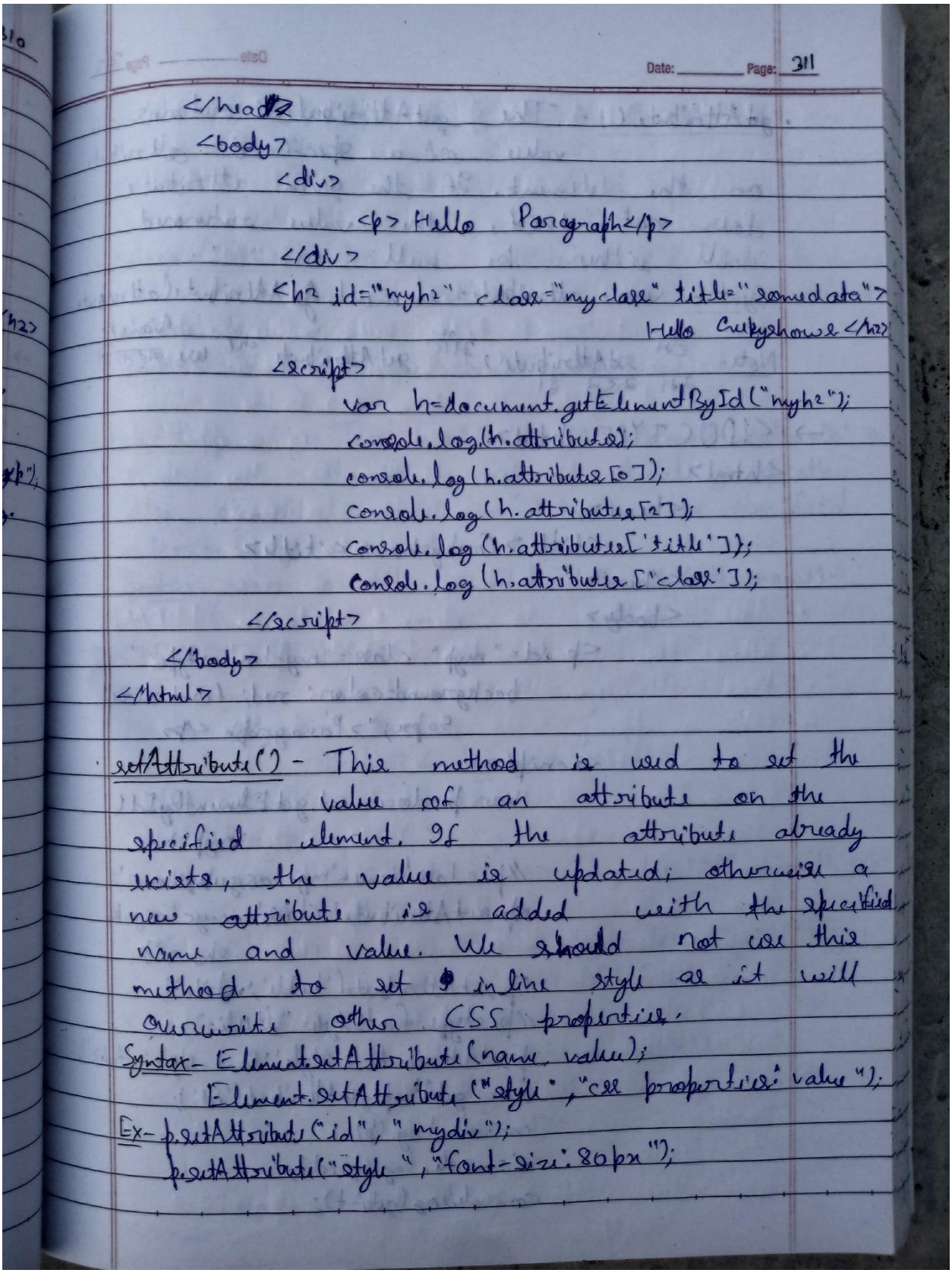


```
<!DOCTYPE html>
<html>
    <head>
        <title>Geekyshows</title>
    </head>
    <body>
        <h2 id="myh2">Hello Geekyshows</h2>
        <p id="myp" title="para" class="myclass">Paragraph</p>
        <script>
            var p=document.getElementById("myp");
            //console.log(Object.prototype.toString.call(p));
            console.log(p.id);
            console.log(p.title);
            console.log(p.className);
            p.title="mypara";
            console.log(p);
        </script>
    </body>
</html>
```

Attributes - The attributes property returns a live collection of all attribute nodes registered to the specified node. It is a read-only property.

Syntax- Element.attributes[index/key];

```
<!DOCTYPE html>
<html>
    <head>
        <title>Geekyshows</title>
```

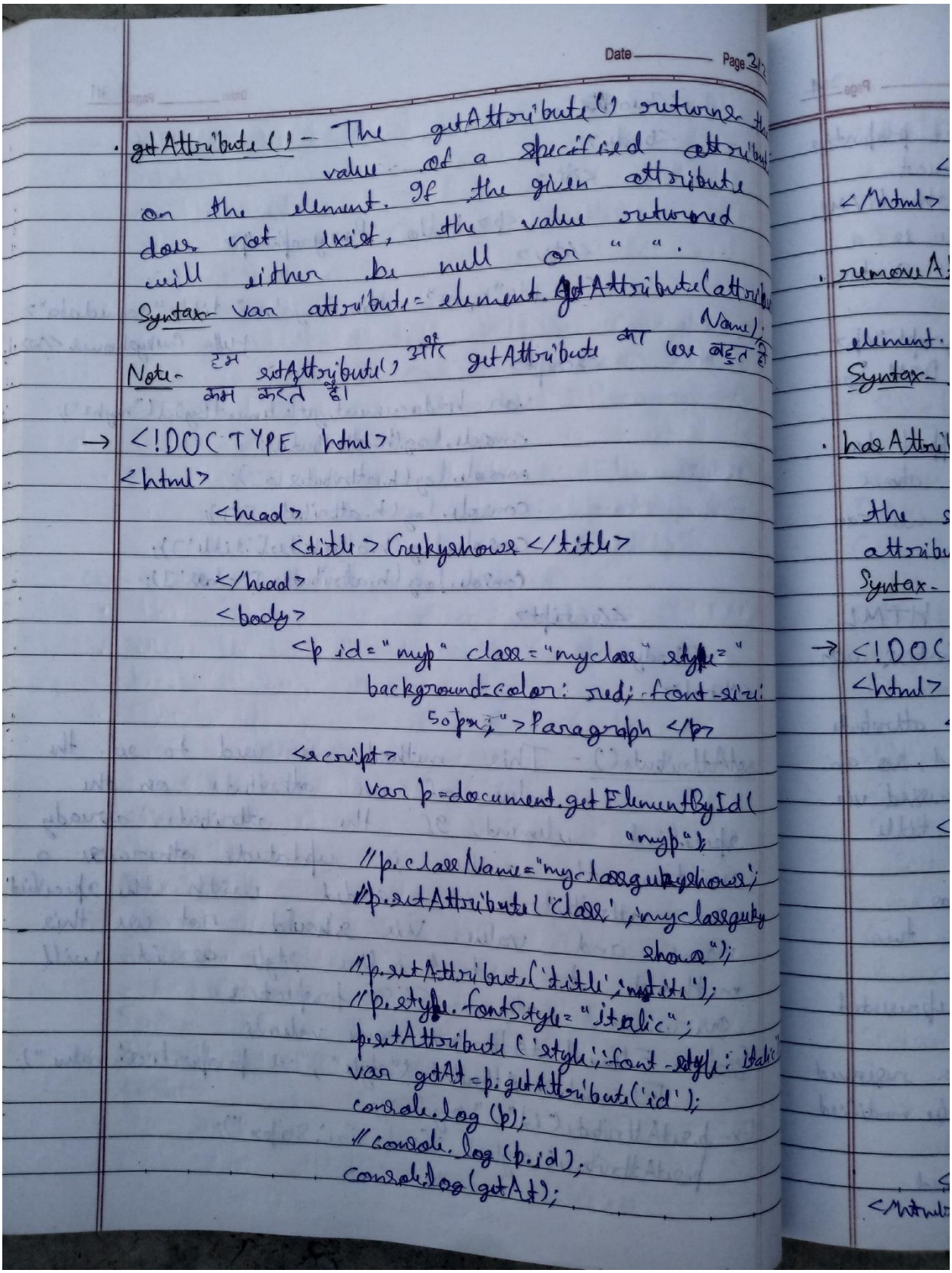


```
</head>
<body>
    <div>
        <p> Hello Paragraph</p>
    </div>
    <h2 id="myh2" class="myclass" title="somedata">
                    Hello Geekyshows</h2>
    <script>
        var h=document.getElementById("myh2");
        console.log(h.attributes);
        console.log(h.attributes[0]);
        console.log(h.attributes[2]);
        console.log(h.attributes['title']);
        Console.log(h.attributes['class']);
    </script>
</body>
</html>
```

setAttribute() - This method is used to set the value of an attribute on the specified element. If the attribute already exists, the value is updated; otherwise a new attribute is added with the specified name and value. We should not use this method to set inline style as it will overwrite other CSS properties.

Syntax- Element.setAttribute(name, value);
Element.setAttribute("style", "css properties: value");

Ex- p.setAttribute("id", "mydiv");
p.setAttribute("style", "font-size: 80px");



getAttribute() - The getAttribute() returns the value of a specified attribute on the element. If the given attribute does not exist, the value returned will either be null or "".

Syntax- var attribute = element.getAttribute(attributeName);

Note- हम setAttribute() और getAttribute का use करते है।

```
<!DOCTYPE html>
<html>
    <head>
        <title> Geekyshows </title>
    </head>
    <body>
        <p id="myp" class="myclass" style="
            background-color: red; font-size:
            50px;"> Paragraph </p>
        <script>
            var p = document.getElementById(
                "myp");
            //p.className = "myclassgeekyshows";
            //p.setAttribute('class', 'myclassgeeky
                shows");
            //p.setAttribute('title', 'mytitle');
            //p.style.fontStyle = "italic";
            p.setAttribute('style', 'font-style: italic');
            var getAt = p.getAttribute('id');
            console.log(p);
            //console.log(p.id);
            console.log(getAt);
```

Date: Page: 313

```
    </script>
  </body>
</html>
```

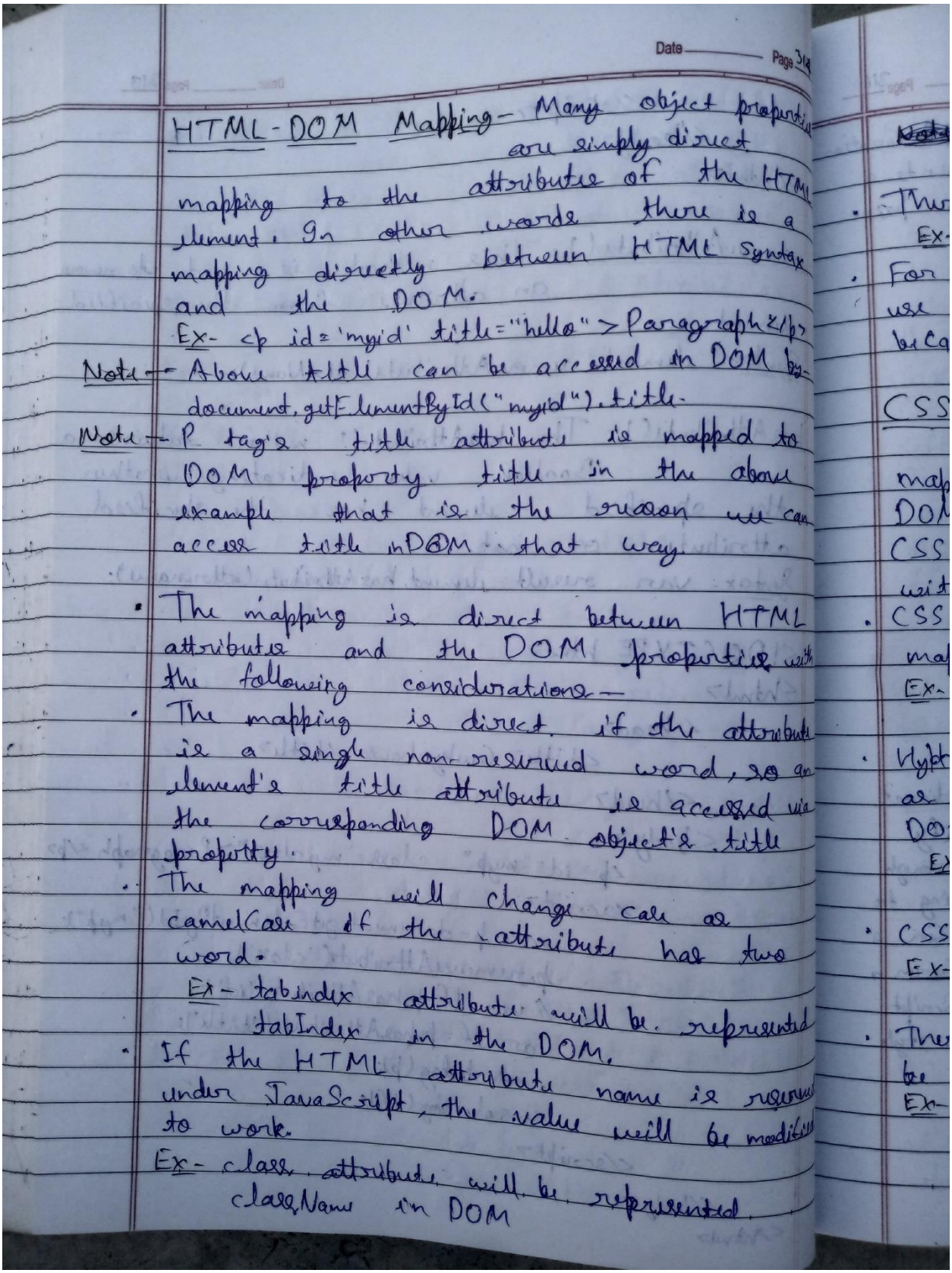
. removeAttribute() - This method is used to remove an attribute from the specified element.

Syntax- element.removeAttribute(attrName);

. hasAttribute() - The hasAttribute() method returns a Boolean value indicating whether the specified element has the specified attribute or not.

Syntax- var result = element.hasAttribute(attr_name);

→
```
<!DOCTYPE html>
<html>
  <head>
    <title> Geekyshows </title>
  </head>
  <body>
    <p id="myp" class="myclass"> Paragraph </p>
    <script>
      var p = document.getElementById("myp");
      p.removeAttribute('class');
      // var tf = p.hasAttribute('id');
      var tf = p.hasAttribute('title');
      console.log(p);
      console.log(tf);
    </script>
  </body>
</html>
```



HTML-DOM Mapping – Many object properties are simply direct mapping to the attributes of the HTML element. In other words there is a mapping directly between HTML syntax and the DOM.

Ex- Paragraph

Note – Above title can be accessed in DOM by- document.getElementById("myid").title.

Note – P tag's title attribute is mapped to DOM property title in the above example that is the reason we can access title in DOM that way.

- The mapping is direct between HTML attributes and the DOM properties with the following considerations –
- The mapping is direct if the attribute is a single non-reserved word, so an element's title attribute is accessed via the corresponding DOM object's title property.
- The mapping will change case as camelCase if the attribute has two word.

  Ex- tabindex attribute will be represented tabIndex in the DOM.
- If the HTML attribute name is reserved under JavaScript, the value will be modified to work.

  Ex- class attribute will be represented className in DOM

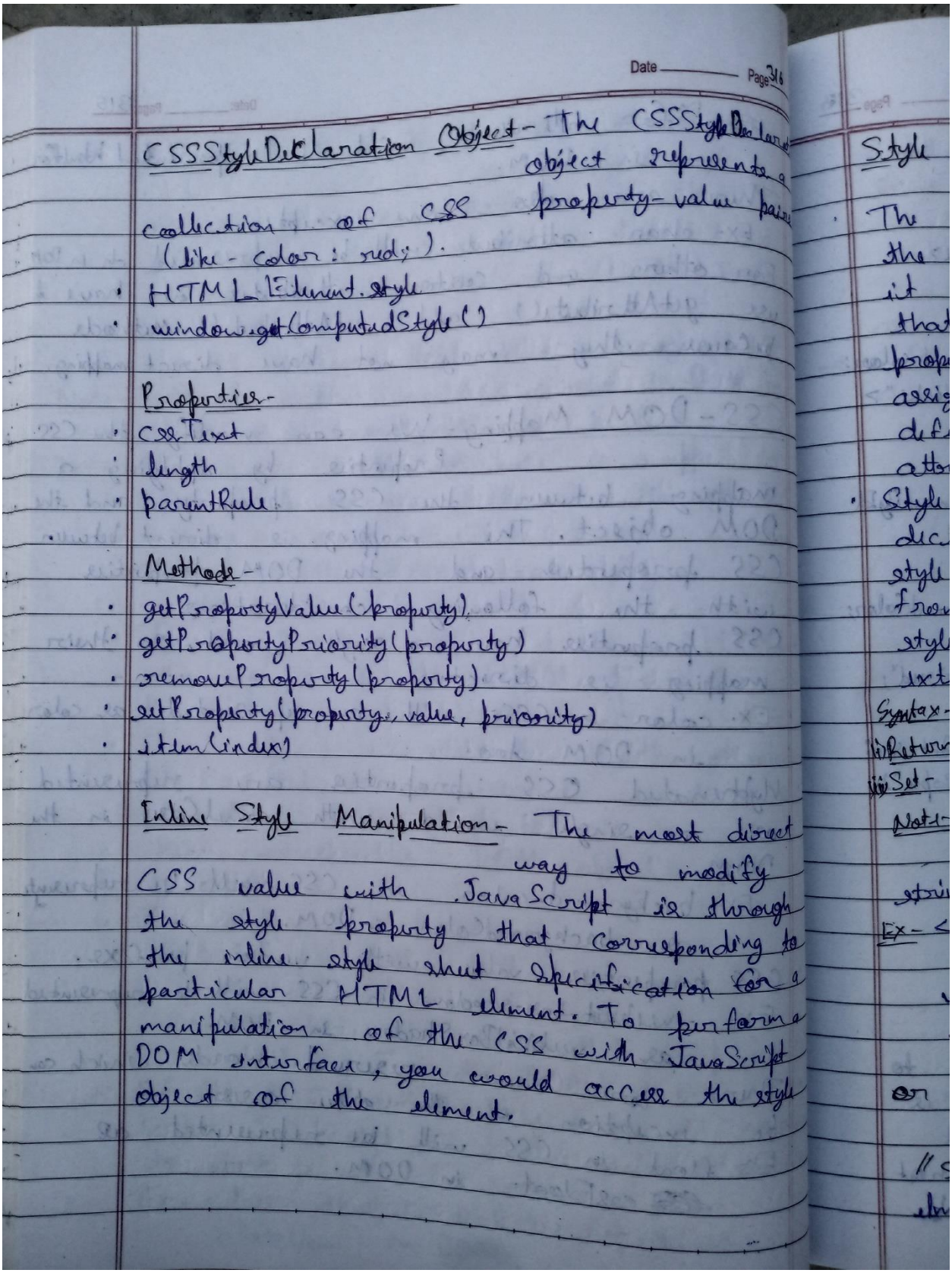
Date: Page: 315

Ex- for attribute will be represented htmlFor in DOM.

- There are also some exception.
  Ex- char attribute will be represented ch in DOM
- For others and custom attributes we have to use getAttribute() and setAttribute() Methods because they may not have direct mapping.

CSS-DOM Mapping - We can modify the CSS Properties by applying a mapping between the CSS property and the DOM object. The mapping is direct between CSS properties and the DOM properties with the following considerations -

- CSS properties have a single word, so their mapping is direct.
  Ex- color in CSS will be represented as color in DOM too.
- Hyphenated CSS properties are represented as a single word with camelCase in the DOM.
  Ex- background-color in CSS will be represented as .backgroundColor in DOM.
- CSS properties value with vendor prefixes.
  Ex- -webkit-box-shadow in CSS will be represented as webkitBoxShadow in DOM.
- There are some reserved words which can be exception in the above cases.
  Ex- float in CSS will be represented as cssFloat in DOM.



CSSStyleDeclaration Object - The CSSStyleDeclaration object represents a collection of CSS property-value pairs (like - color : red;).

- HTMLElement.style
- window.getComputedStyle()

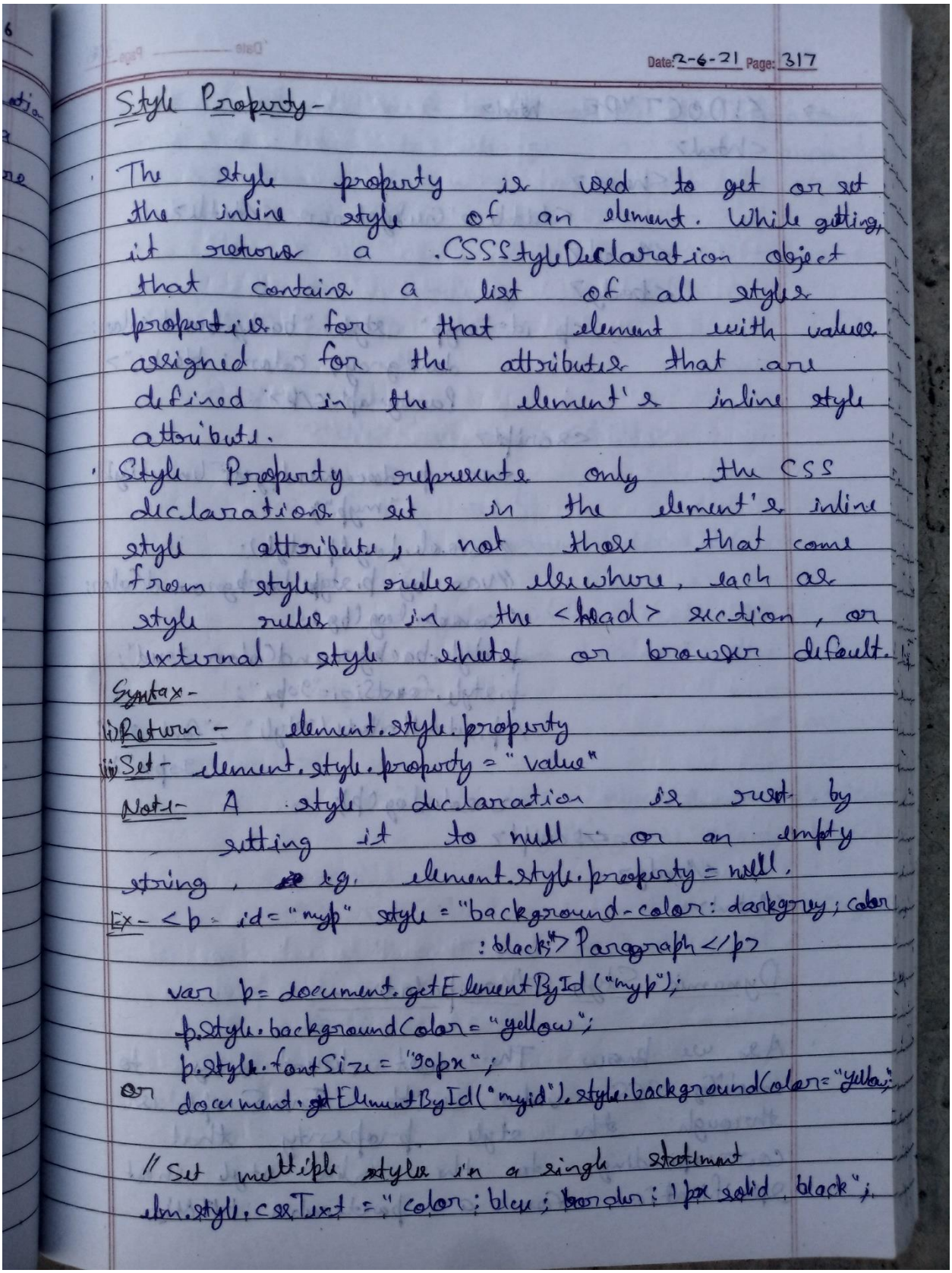
Properties -

- cssText
- length
- parentRule

Methods -

- getPropertyValue(property)
- getPropertyPriority(property)
- removeProperty(property)
- setProperty(property, value, priority)
- item(index)

Inline Style Manipulation - The most direct way to modify CSS value with JavaScript is through the style property that corresponding to the inline style sheet specification for a particular HTML element. To perform a manipulation of the CSS with JavaScript DOM interface, you would access the style object of the element.



Date: 2-6-21 Page: 317

Style Property -

- The style property is used to get or set the inline style of an element. While getting, it returns a .CSSStyleDeclaration object that contains a list of all styles properties for that element with values assigned for the attributes that are defined in the element's inline style attributes.
- Style Property represents only the CSS declarations set in the element's inline style attribute, not those that come from style rules elsewhere, such as style rules in the <head> section, or external style sheets or browser default.

Syntax -

i) Return - element.style.property

ii) Set - element.style.property = "value"

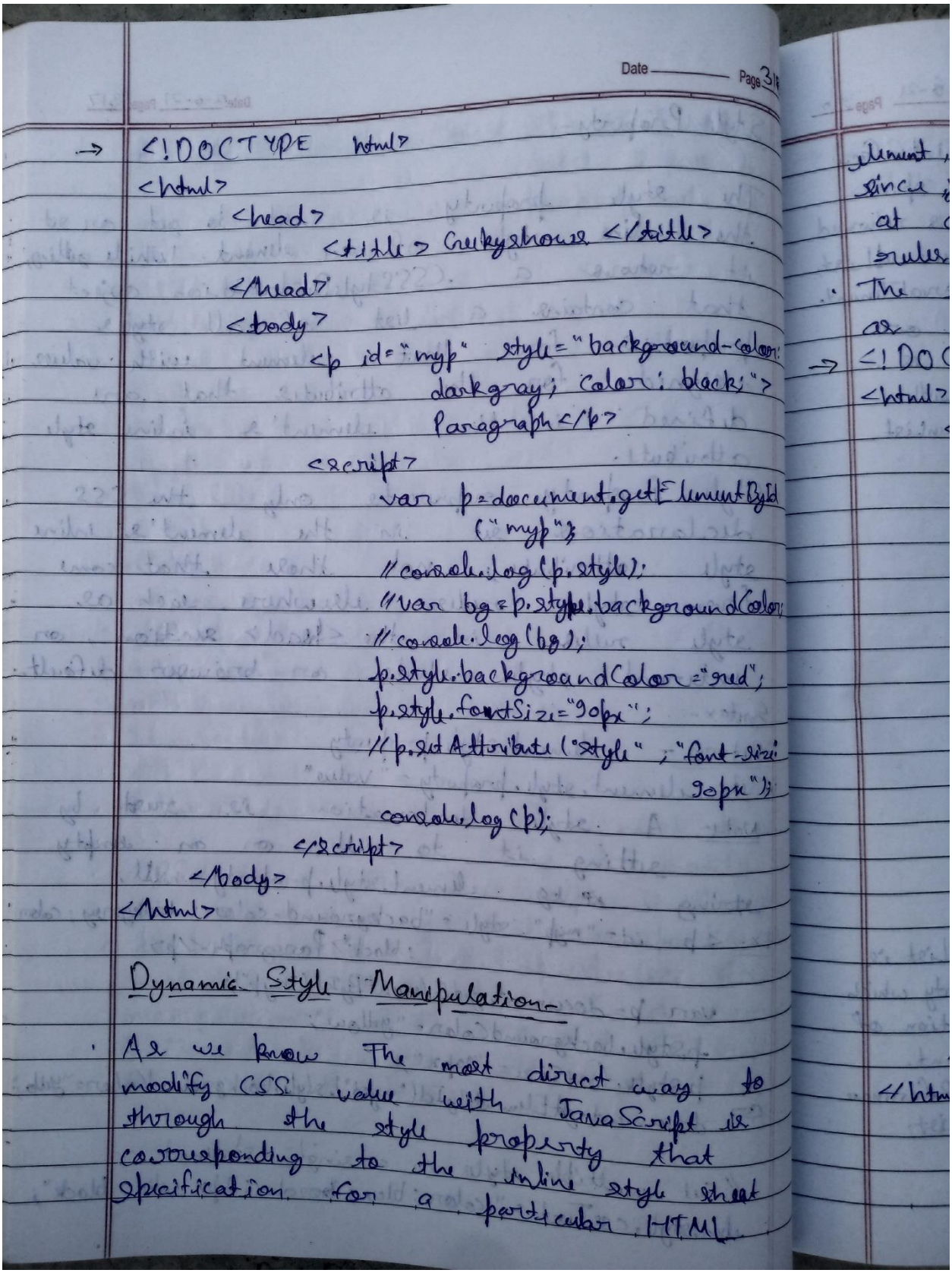
Note - A style declaration is reset by setting it to null or an empty string, e.g. element.style.property = null.

Ex - Paragraph

```
var p = document.getElementById("myp");
p.style.backgroundColor = "yellow";
p.style.fontSize = "30px";
```

or `document.getElementById("myid").style.backgroundColor = "yellow";`

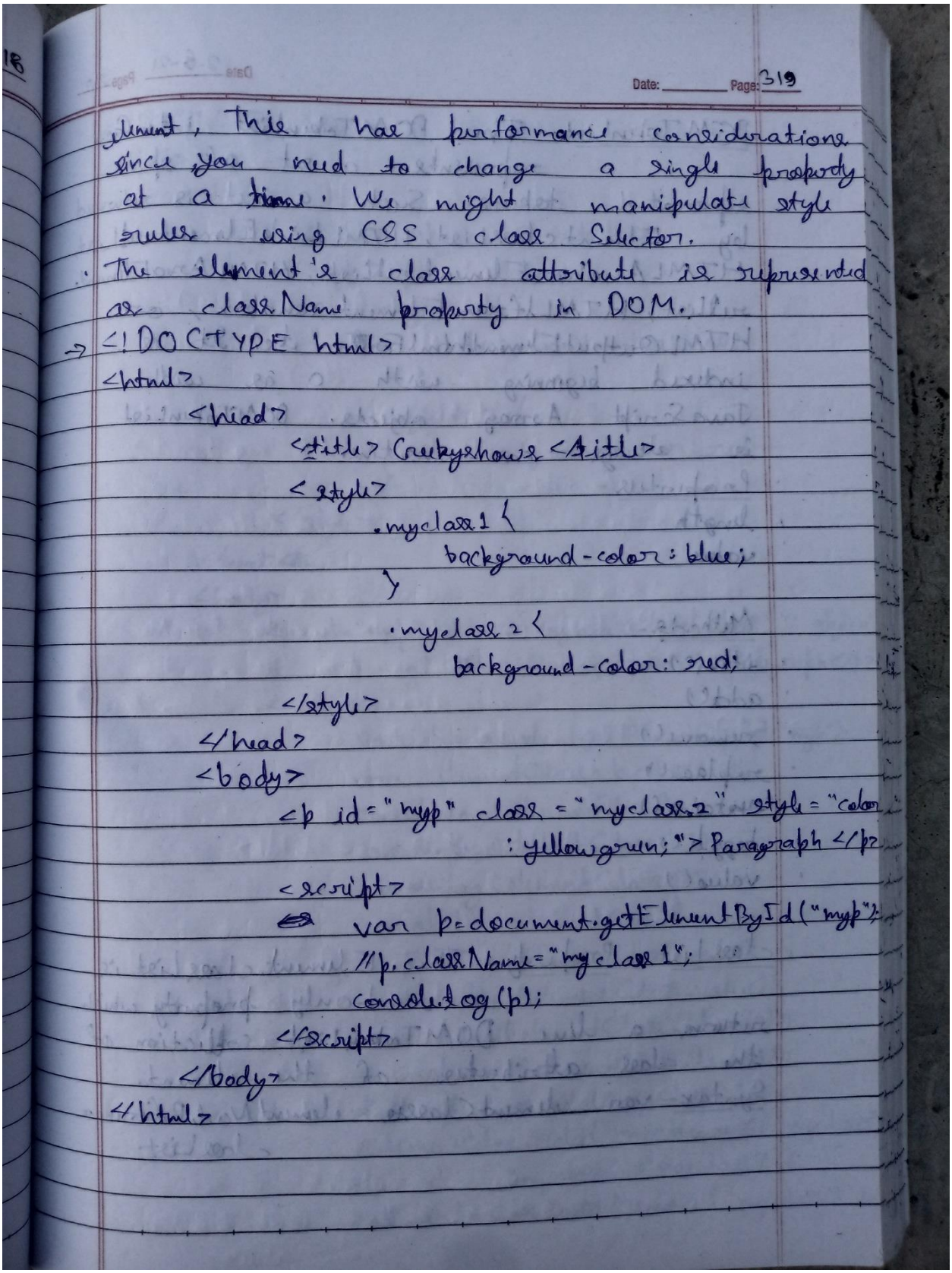
```
// Set multiple styles in a single statement
elm.style.cssText = "color: blue; border: 1px solid black";
```



```
<!DOCTYPE html>
<html>
    <head>
        <title> Geekyshows </title>
    </head>
    <body>
        <p id="myp" style="background-color:
            darkgray; color: black;">
            Paragraph</p>
        <script>
            var p=document.getElementById
                ("myp");
            //console.log(p.style);
            //var bg=p.style.backgroundColor;
            //console.log(bg);
            p.style.backgroundColor="red";
            p.style.fontSize="30px";
            //p.setAttribute("style","font-size:
                30px");
            console.log(p);
        </script>
    </body>
</html>
```

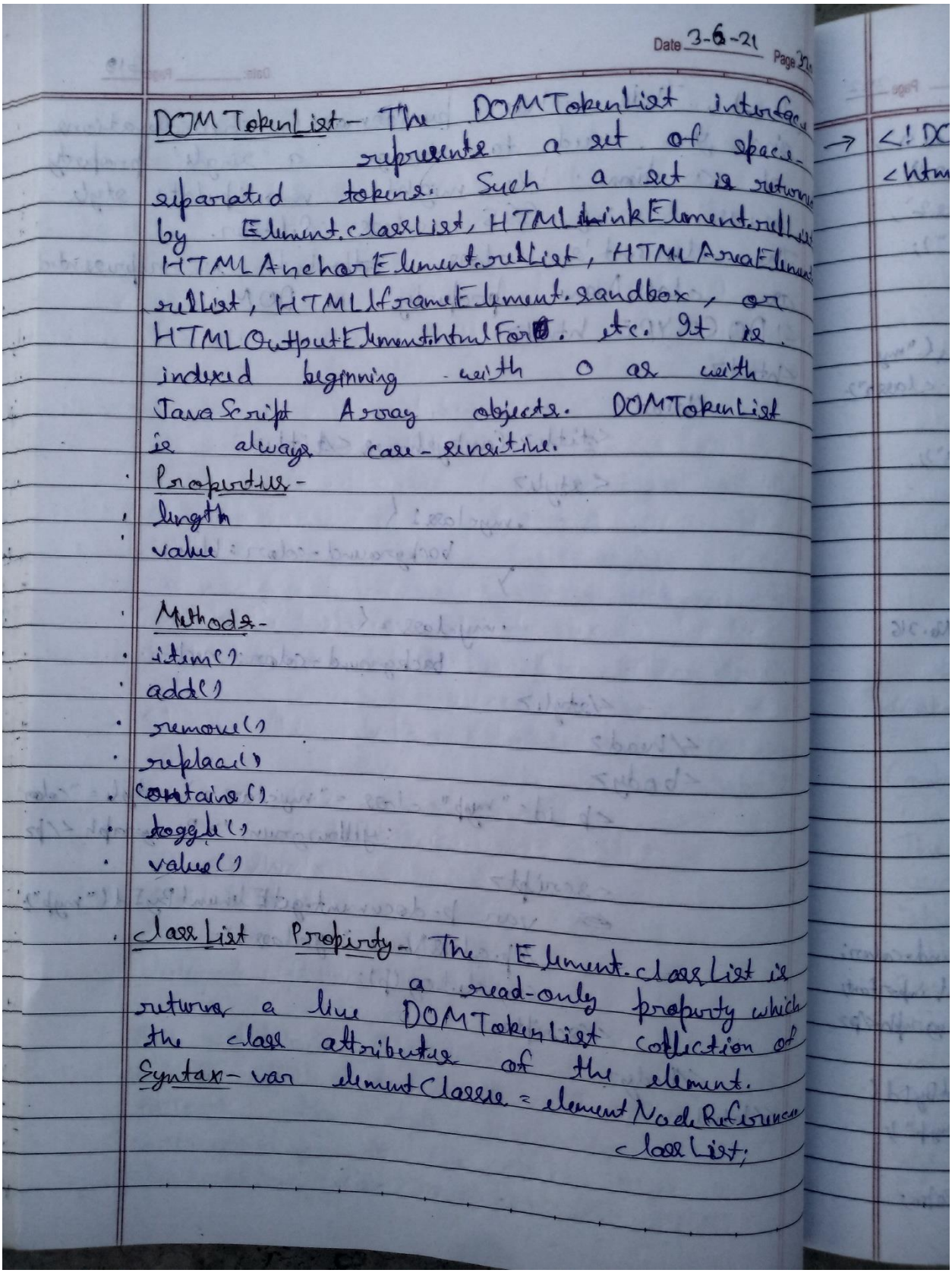
Dynamic Style Manipulation-

As we know The most direct way to modify CSS value with JavaScript is through the style property that corresponding to the inline style sheet specification for a particular HTML



element, This has performance considerations since you need to change a single property at a time. We might manipulate style rules using CSS class Selector.
The element's class attribute is represented as className property in DOM.

```
<!DOCTYPE html>
<html>
    <head>
        <title> Geekyshows </title>
        <style>
            .myclass1 {
                background-color : blue;
            }
            .myclass2 {
                background-color: red;
            }
        </style>
    </head>
    <body>
        <p id = "myp" class = "myclass2" style = "color
                : yellowgreen;"> Paragraph </p>
        <script>
            var p=document.getElementById("myp");
            //p.className="myclass1";
            console.log(p);
        </script>
    </body>
</html>
```



Date 3-6-21 Page 31

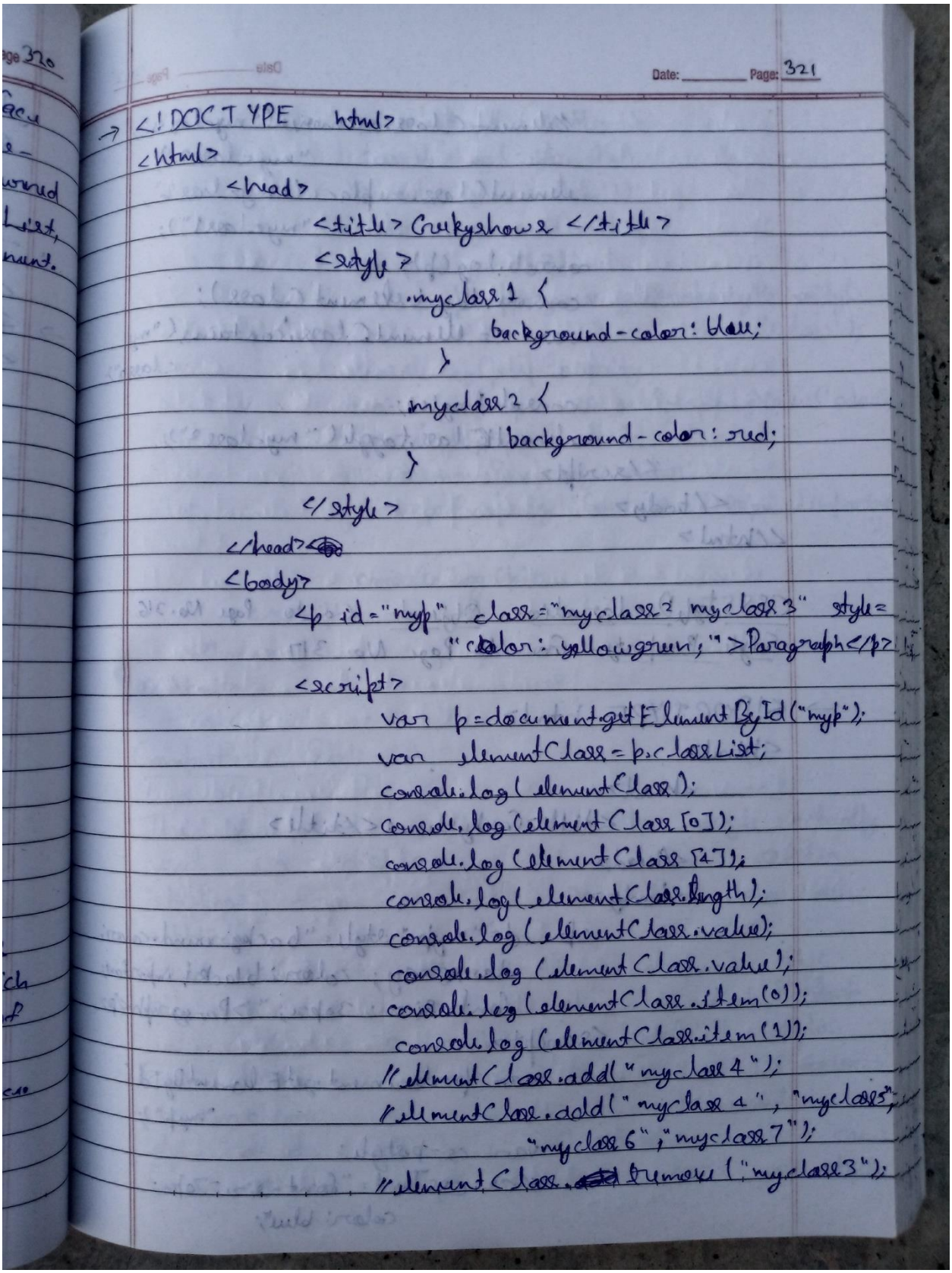
DOMTokenList - The DOMTokenList interface represents a set of space-separated tokens. Such a set is returned by Element.classList, HTMLLinkElement.relList, HTMLAnchorElement.relList, HTMLAreaElement.relList, HTMLIframeElement.sandbox, or HTMLOutputElement.htmlFor. etc. It is indexed beginning with 0 as with JavaScript Array objects. DOMTokenList is always case-sensitive.

- Properties -
- length
- value

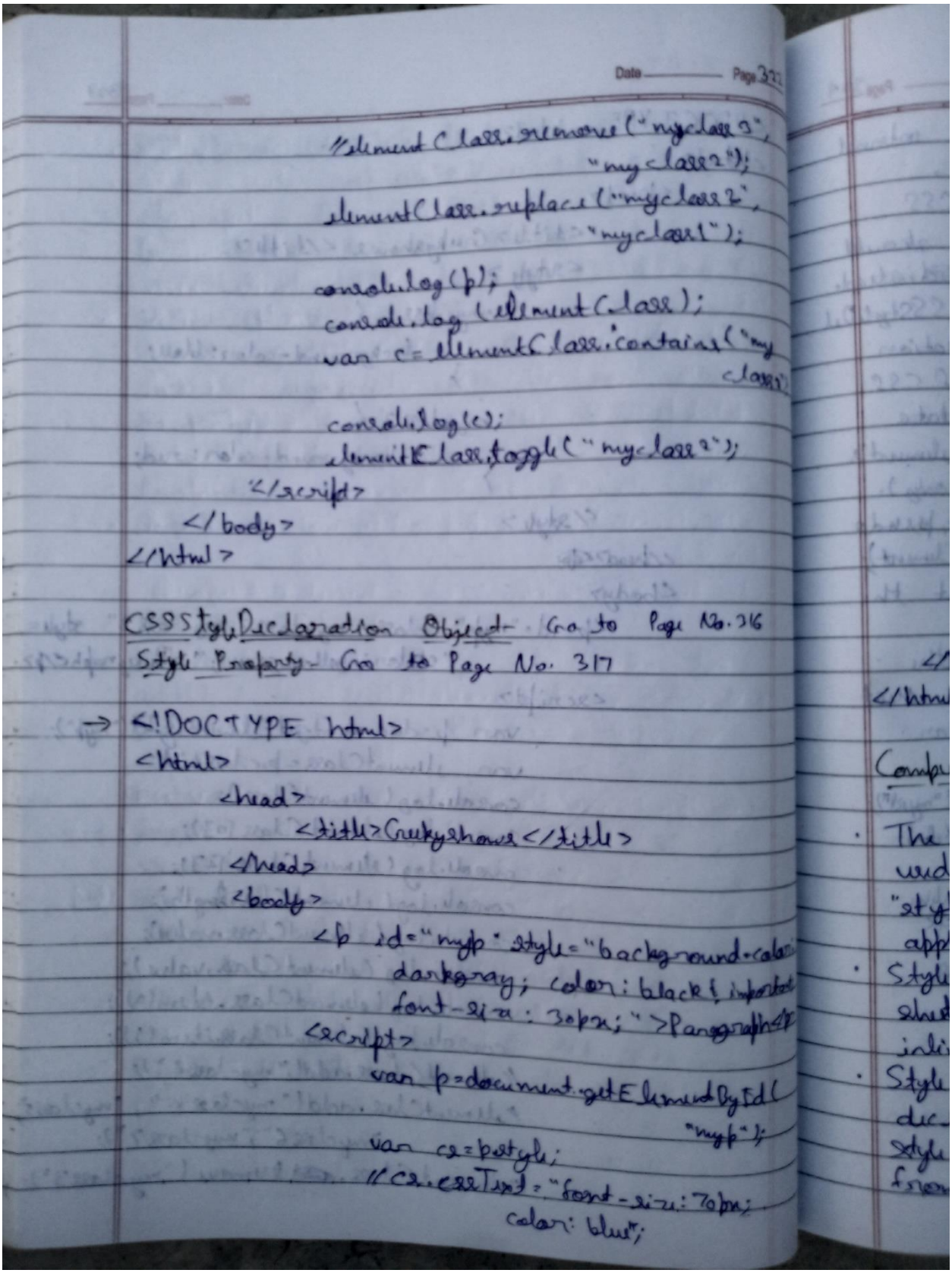
- Methods -
- item()
- add()
- remove()
- replace()
- contains()
- toggle()
- value()

- classList Property - The Element.classList is a read-only property which returns a live DOMTokenList collection of the class attributes of the element.

Syntax - var elementClasses = elementNodeReference.classList;



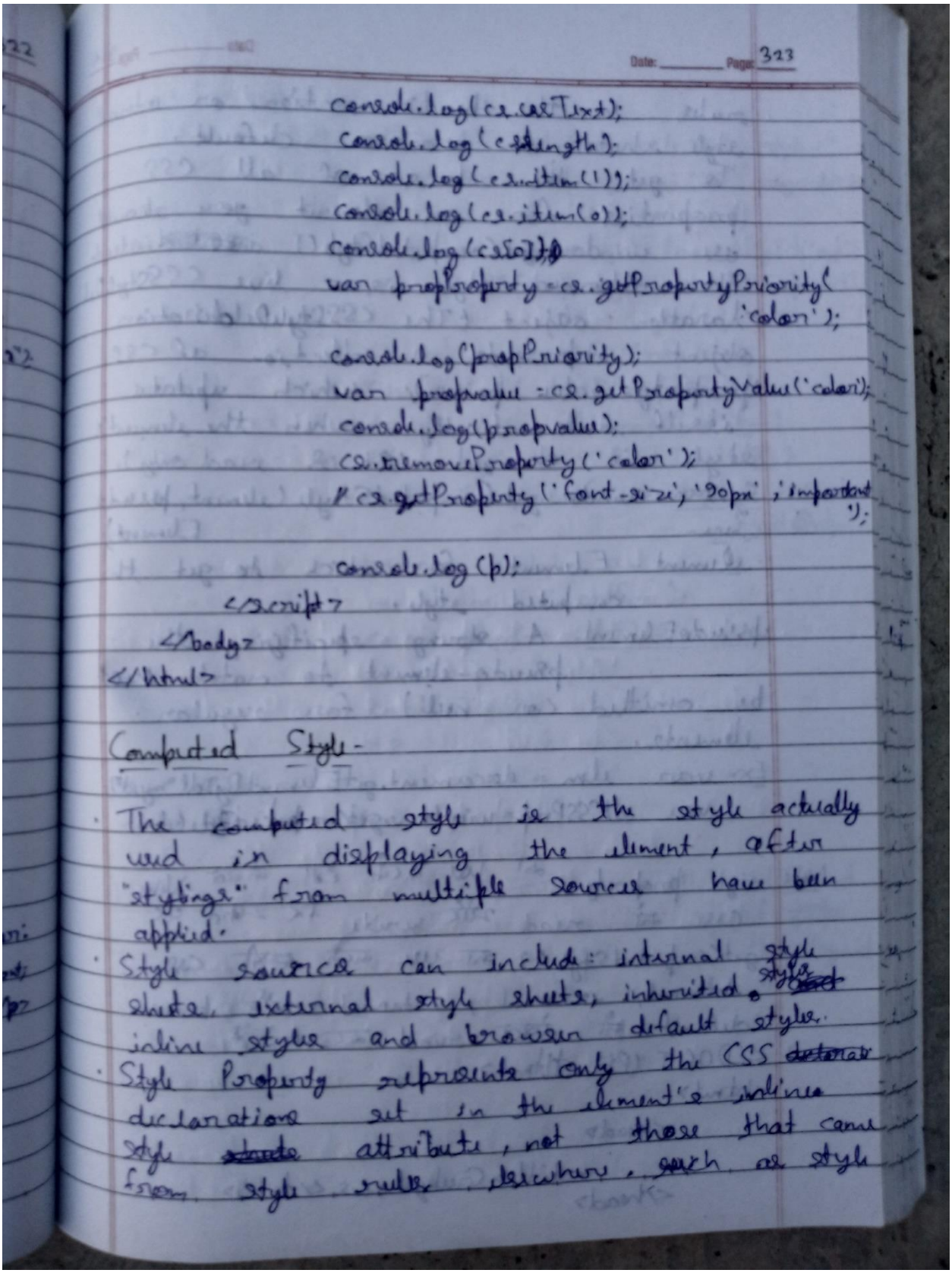
```
<!DOCTYPE html>
<html>
    <head>
        <title> Geekyshows </title>
        <style>
            .myclass1 {
                background-color: blue;
            }
            myclass2 {
                background-color: red;
            }
        </style>
    </head>
    <body>
        <p id="myp" class="myclass2 myclass3" style=
            "color: yellowgreen;" >Paragraph</p>
        <script>
            var p=document.getElementById("myp");
            var elementClass=p.classList;
            console.log(elementClass);
            console.log(elementClass[0]);
            console.log(elementClass[1]);
            console.log(elementClass.length);
            console.log(elementClass.value);
            console.log(elementClass.value);
            console.log(elementClass.item(0));
            console.log(elementClass.item(1));
            //elementClass.add("myclass4");
            //elementClass.add("myclass4", "myclass5",
                "myclass6", "myclass7");
            //elementClass.remove("myclass3");
```



```
//element Class.remove("myclass3",
                        "myclass2");
elementClass.replace("myclass2",
                        "myclass1");

console.log(p);
console.log(elementClass);
var c=elementClass.contains("my
                        class1");

console.log(c);
elementClass.toggle("myclass2");
</script>
</body>
</html>
```

CSSStyleDeclaration Object - Go to Page No. 316
Style Property - Go to Page No. 317

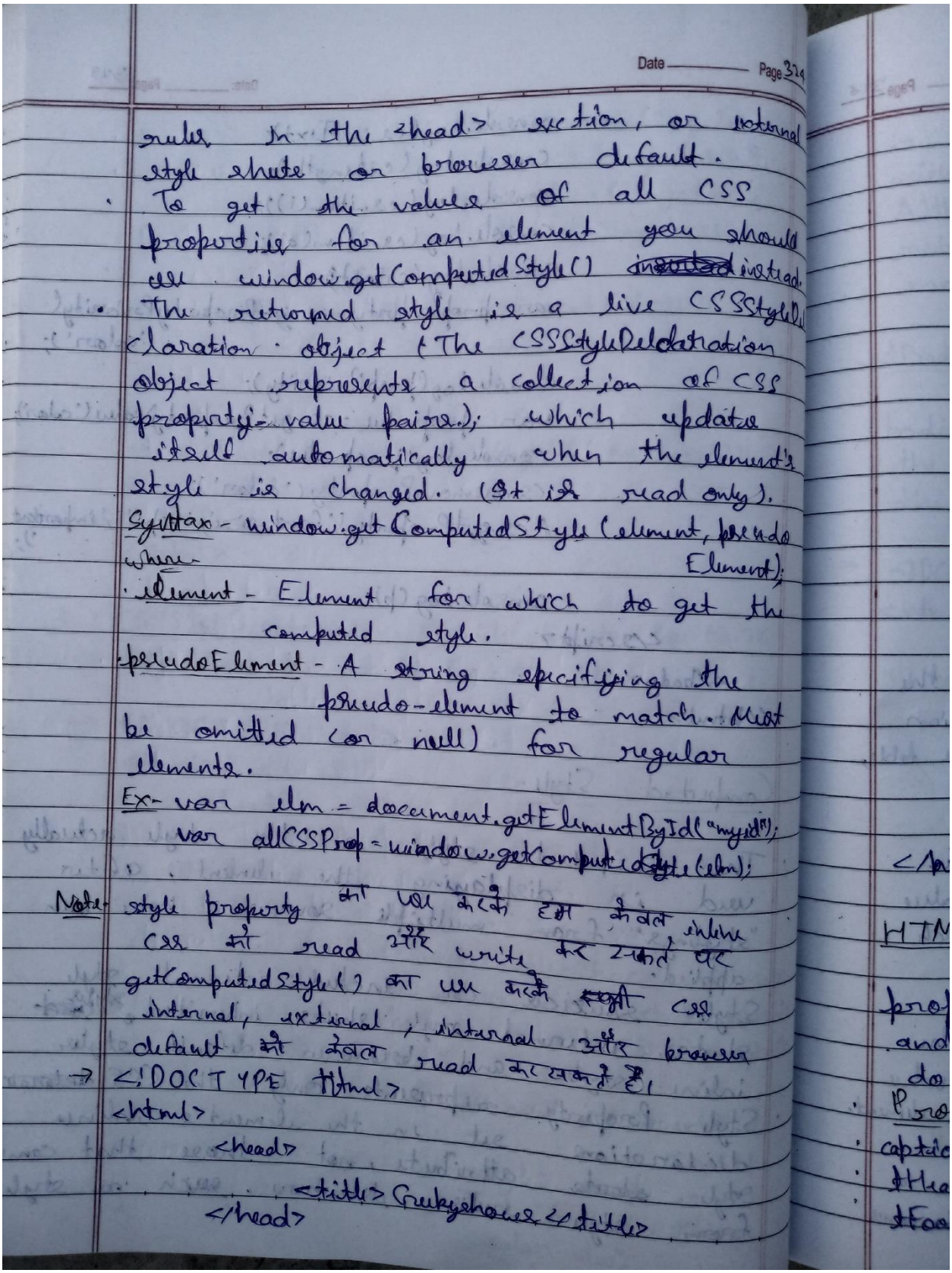
```
<!DOCTYPE html>
<html>
    <head>
        <title>Geekyshows</title>
    </head>
    <body>
        <p id="myp" style="background-color:
            darkgray; color:black !important;
            font-size:30px;">Paragraph</p>
        <script>
            var p=document.getElementById(
                                "myp");

            var cs=p.style;
            //cs.cssText="font-size:70px;
                        color:blue";
```



```
console.log(cs.cssText);
console.log(cs.length);
console.log(cs.item(1));
console.log(cs.item(0));
console.log(cs[0]);
var propProperty = cs.getPropertyPriority(
                                'color');
console.log(propPriority);
var propvalue = cs.getPropertyValue('color');
console.log(propvalue);
cs.removeProperty('color');
// cs.getProperty('font-size', '30px', 'important');

console.log(p);
</script>
</body>
</html>
```

Computed Style -

- The computed style is the style actually used in displaying the element, after "stylings" from multiple sources have been applied.
- Style sources can include: internal style sheets, external style sheets, inherited styles, inline styles and browser default styles.
- Style Property represents only the CSS declarations set in the element's inline style attribute, not those that came from style rules elsewhere, such as style



rules in the <head> section, or internal style sheets or browser default.

To get the values of all CSS properties for an element you should use window.getComputedStyle() instead.

The returned style is a live CSSStyleDeclaration object (The CSSStyleDeclaration object represents a collection of CSS property-value pairs.), which updates itself automatically when the element's style is changed. (It is read only).

Syntax - window.getComputedStyle(element, pseudoElement);

where

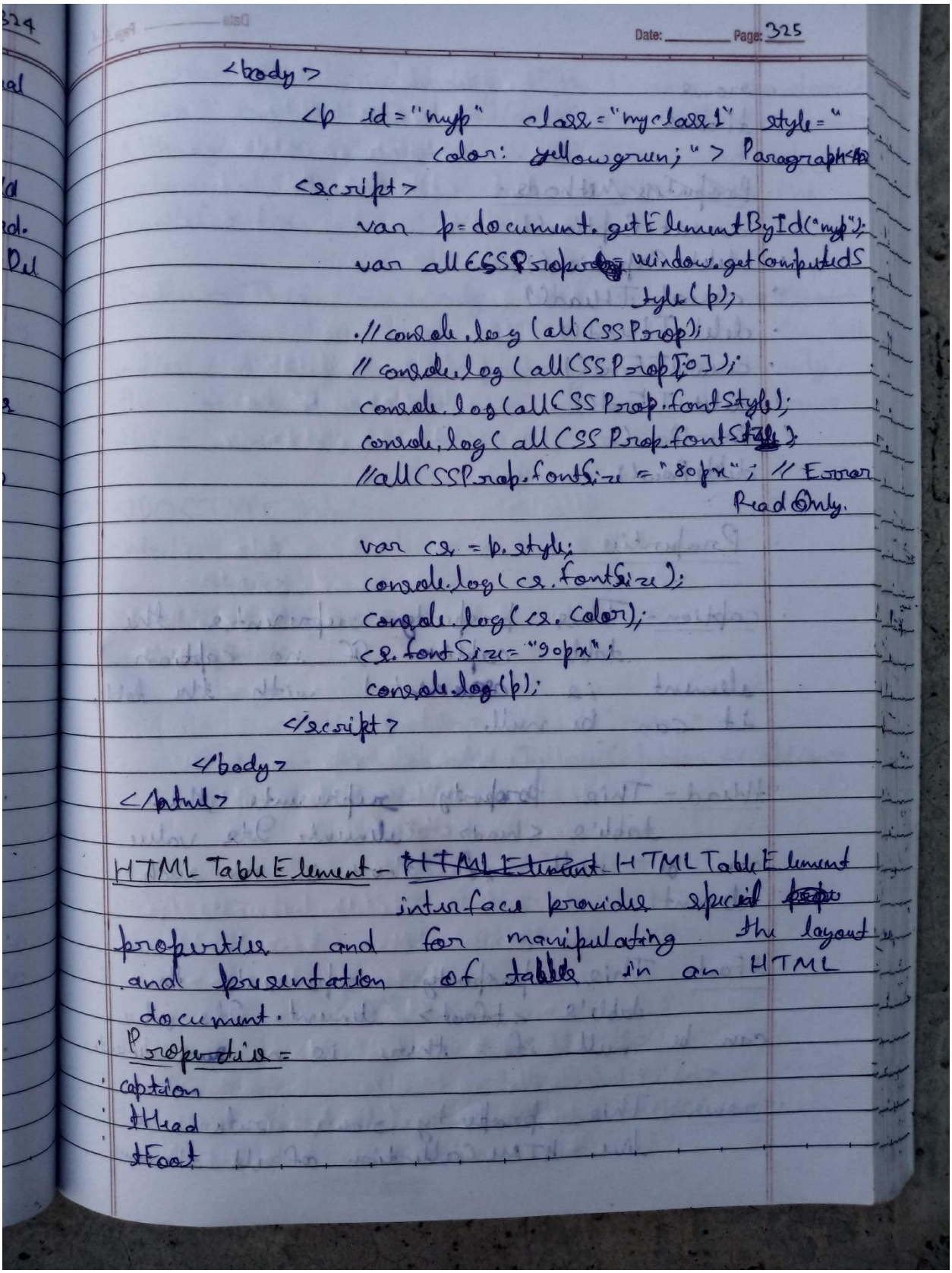
element - Element for which to get the computed style.

pseudoElement - A string specifying the pseudo-element to match. Must be omitted (or null) for regular elements.

Ex- var elm = document.getElementById("myid");
var allCSSProp = window.getComputedStyle(elm);

Note- style property का use करके हम केवल inline CSS को read और write कर सकते है पर getComputedStyle() का use करके हम CSS internal, external, internal और browser default को केवल read कर सकते है।

→ <!DOCTYPE html>
<html>
<head>
<title> Geekyshows </title>
</head>

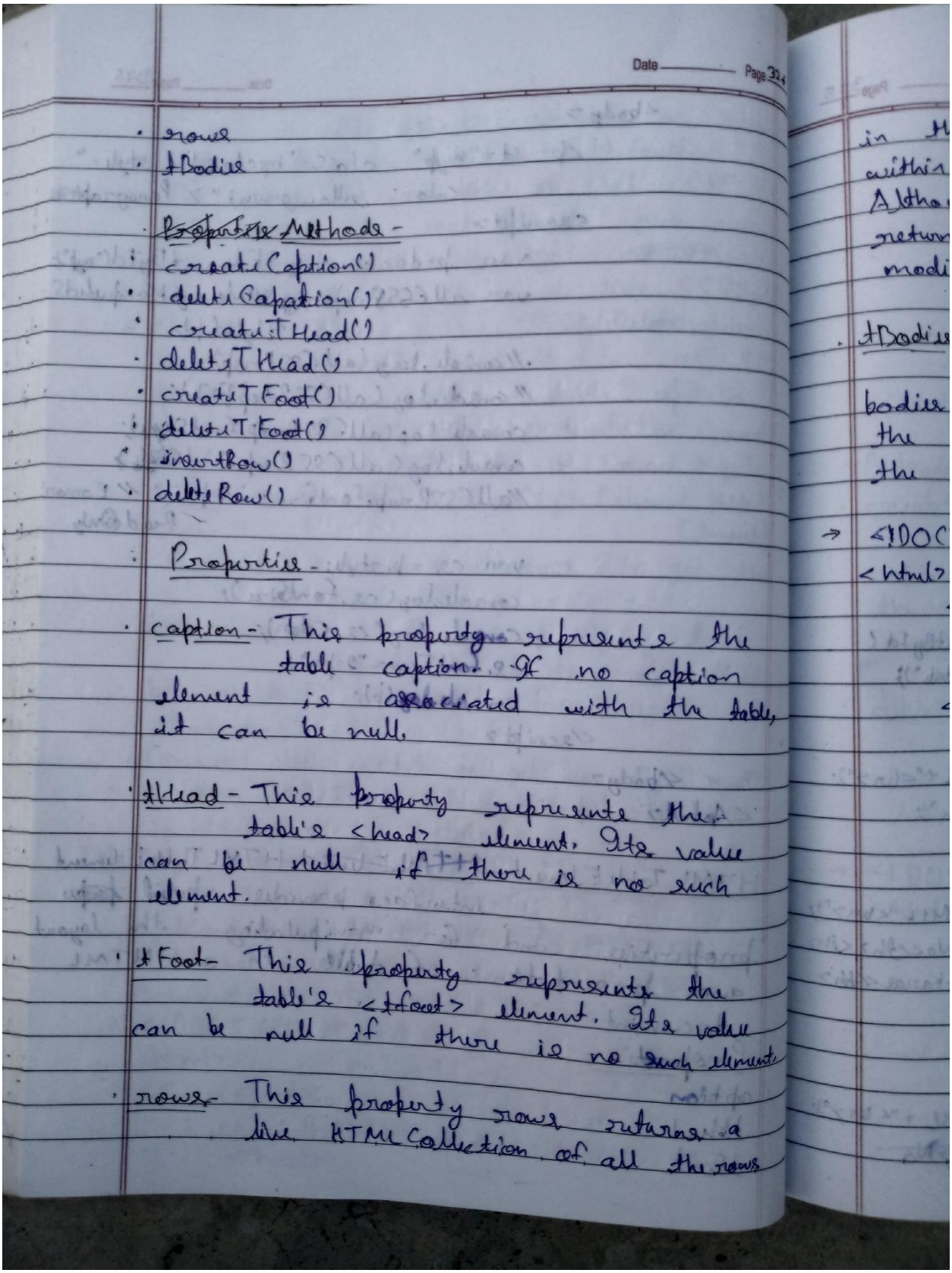


```
<body>
    <p id="myp" class="myclass1" style="
        color: yellowgreen;"> Paragraph</p>
    <script>
        var p=document.getElementById("myp");
        var allCSSProp= window.getComputedS
                        tyle(p);
        //console.log(allCSSProp);
        // console.log(allCSSProp[0]);
        console.log(allCSSProp.fontStyle);
        console.log(allCSSProp.fontStyle);
        //allCSSProp.fontSize="80px"; // Error
                                        Read Only.

        var cs=p.style;
        console.log(cs.fontSize);
        console.log(cs.color);
        cs.fontSize="90px";
        console.log(p);
    </script>
</body>
</html>
```

HTML Table Element - ~~HTMLElement~~ HTMLTableElement interface provides special ~~prop~~ properties and for manipulating the layout and presentation of ~~table~~ in an HTML document.

Properties =

- caption
- tHead
- tFoot



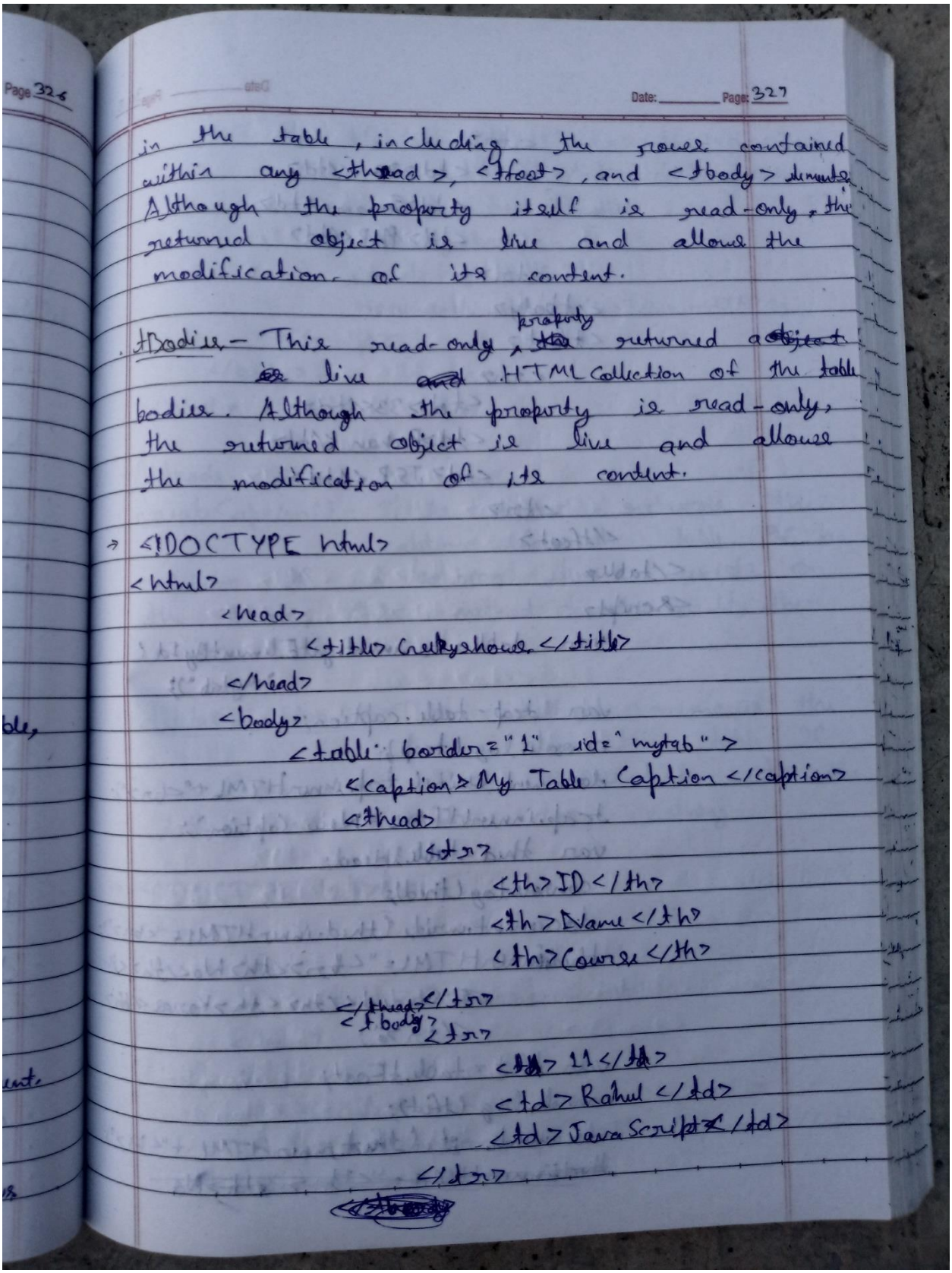
- rows
- tBodies

~~Properties~~ Methods -

- createCaption()
- deleteCaption()
- createTHead()
- deleteTHead()
- createTFoot()
- deleteTFoot()
- insertRow()
- deleteRow()

Properties -

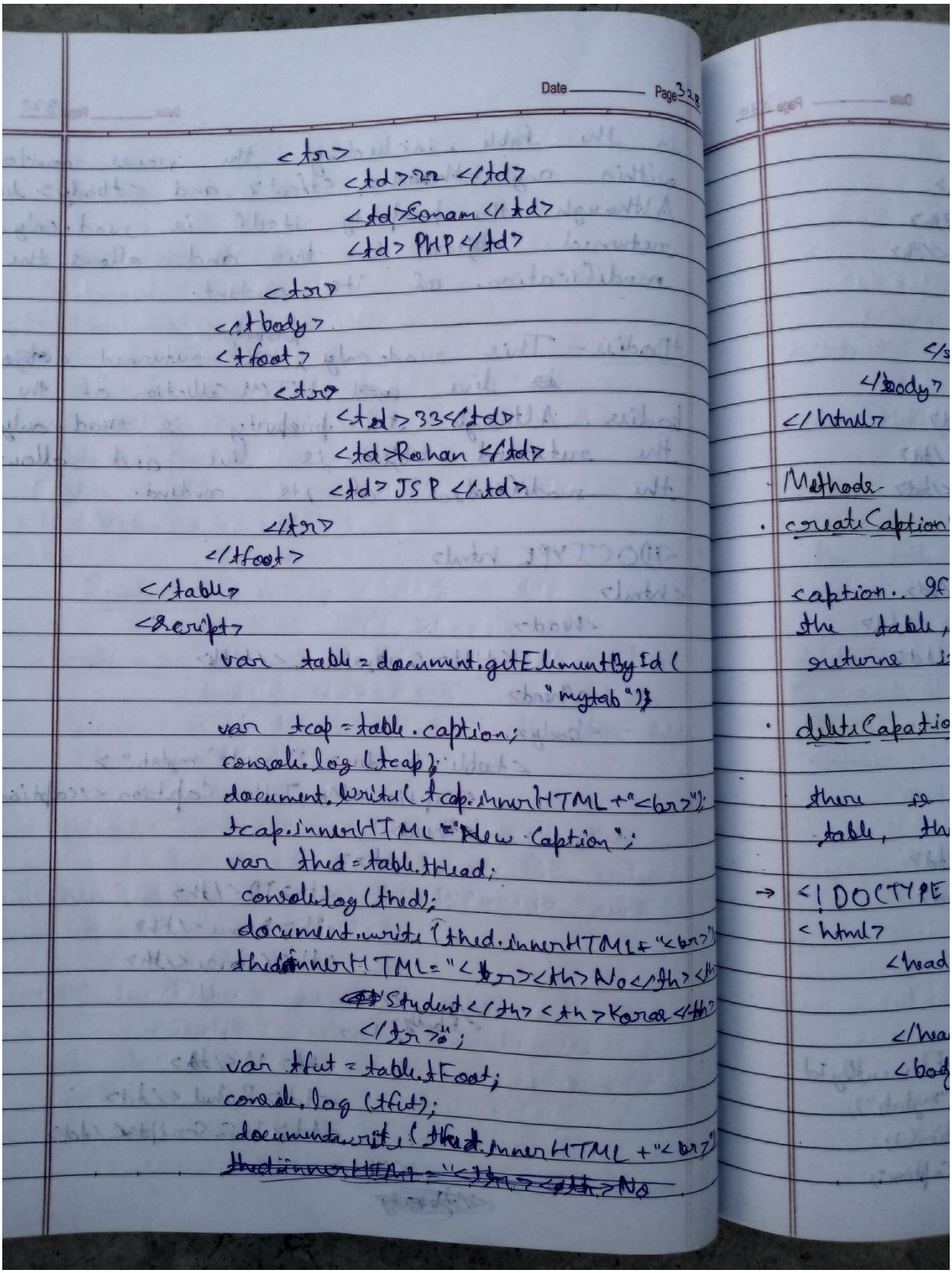
- caption - This property represents the table caption. If no caption element is associated with the table, it can be null.
- tHead - This property represents the table's <head> element. Its value can be null if there is no such element.
- tFoot - This property represents the table's <tfoot> element. It's value can be null if there is no such element.
- rows - This property rows returns a live HTML Collection of all the rows



in the table, including the rows contained within any <thead>, <tfoot>, and <tbody> elements. Although the property itself is read-only, the returned object is live and allows the modification of its content.

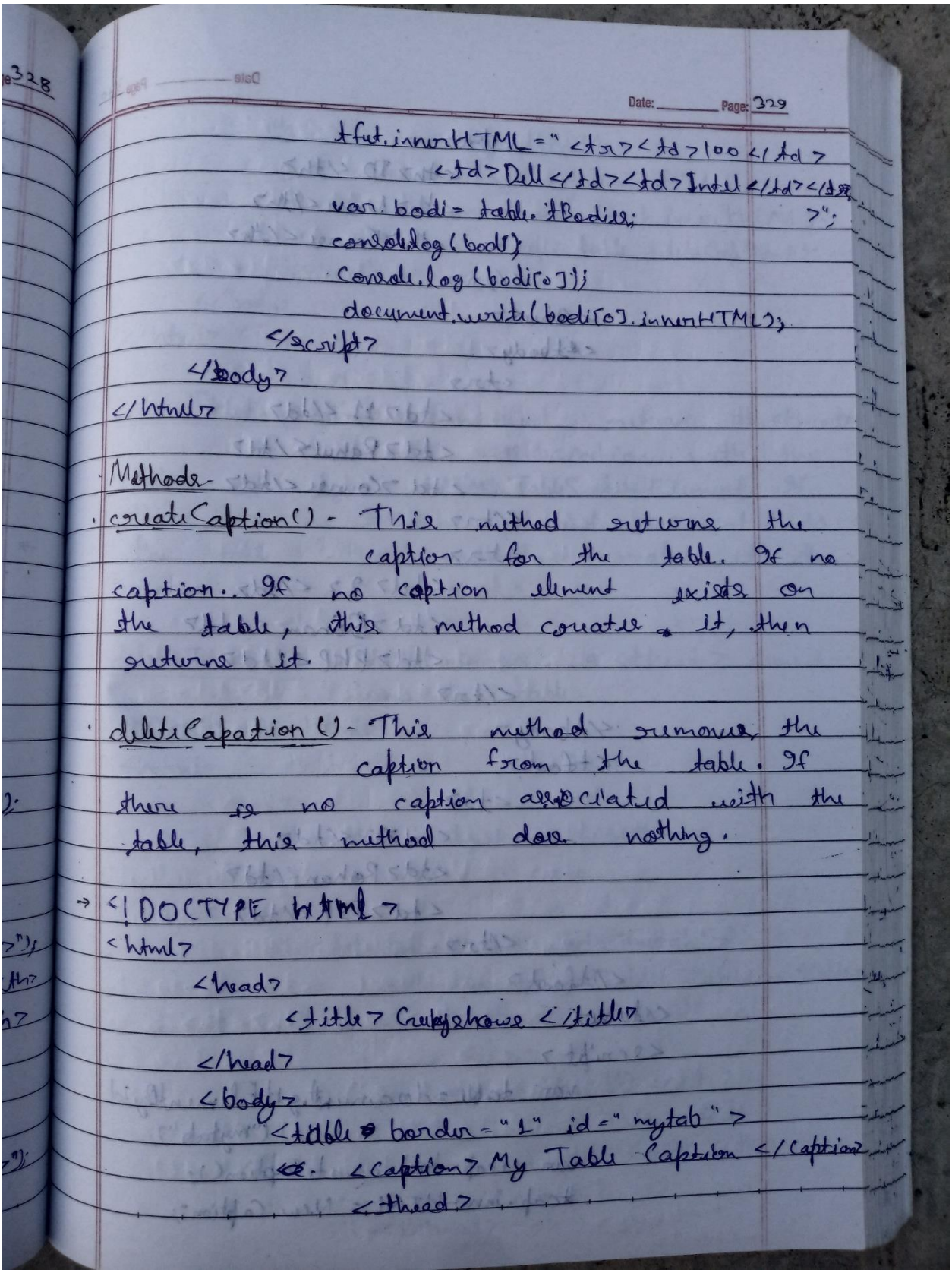
tBodies – This read-only property returned a live HTML collection of the table bodies. Although the property is read-only, the returned object is live and allows the modification of its content.

```
<!DOCTYPE html>
<html>
    <head>
        <title> Geekyshows </title>
    </head>
    <body>
        <table border="1" id="mytab">
            <caption> My Table Caption </caption>
            <thead>
                <tr>
                    <th> ID </th>
                    <th> Name </th>
                    <th> Course </th>
                </tr>
            </thead>
            <tbody>
                <tr>
                    <td> 11 </td>
                    <td> Rahul </td>
                    <td> JavaScript </td>
                </tr>
```



```
<tr>
    <td>22</td>
    <td>Sonam</td>
    <td>PHP</td>
</tr>
</tbody>
<tfoot>
    <tr>
        <td>33</td>
        <td>Rohan</td>
        <td>JSP</td>
    </tr>
</tfoot>
</table>
<script>
    var table = document.getElementById("mytab");
    var tcap = table.caption;
    console.log(tcap);
    document.write(tcap.innerHTML + "<br>");
    tcap.innerHTML = "New Caption";
    var thed = table.tHead;
    console.log(thed);
    document.write(thed.innerHTML + "<br>");
    thed.innerHTML = "<tr><th>No</th><th>Student</th><th>Karan</th></tr>";
    var tfut = table.tFoot;
    console.log(tfut);
    document.write(tfut.innerHTML + "<br>");
    tfut.innerHTML = "<tr><th>No</th>
```

```
</tbody>
</html>

Methods
createCaption

caption..
the table,
returns

deleteCapatio

there
table, th

<!DOCTYPE
<html>
<head
</hea
<bod
```



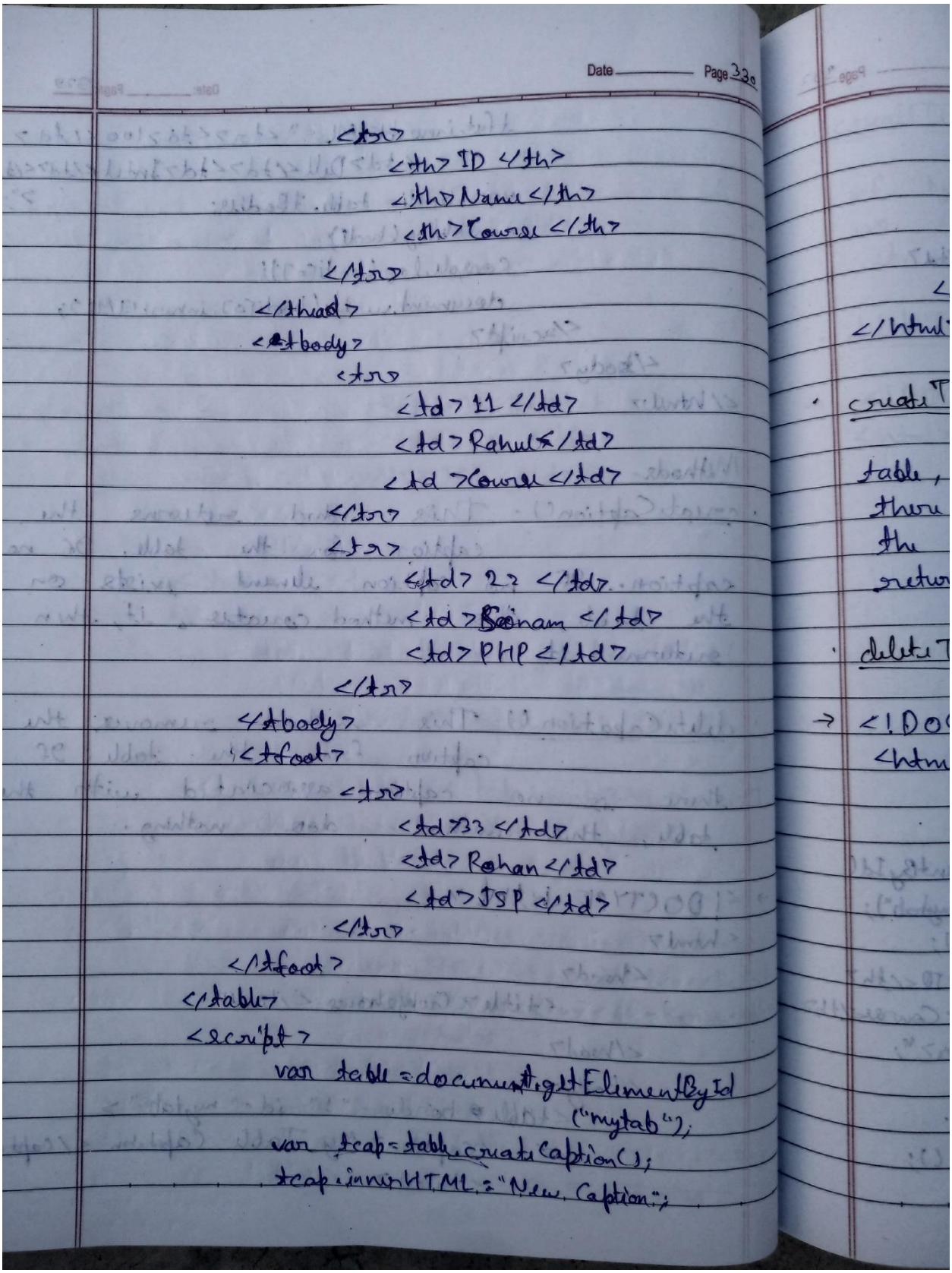
```
tfut.innerHTML = "<tr><td>100</td>
    <td>Dell</td><td>Intel</td></tr>
    ";
var bodi = table.tBodies;
console.log(bodi);
console.log(bodi[0]);
document.write(bodi[0].innerHTML);
</script>
</body>
</html>
```

Methods-

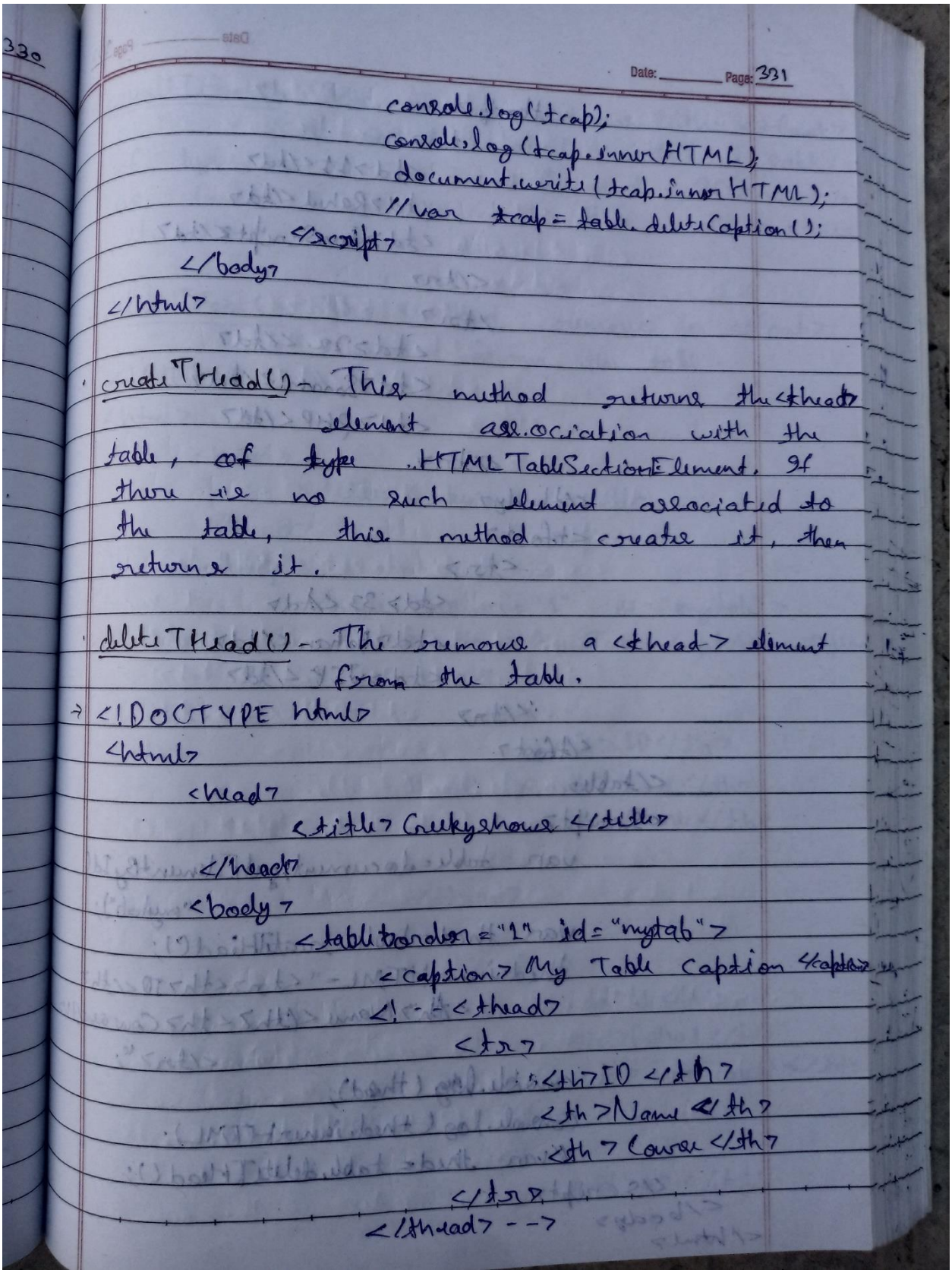
- createCaption() - This method returns the caption for the table. If no caption. If no caption element exists on the table, this method creates it, then returns it.

- deleteCaption() - This method removes the caption from the table. If there is no caption associated with the table, this method does nothing.

```
<!DOCTYPE html>
<html>
    <head>
        <title> Geekyshows </title>
    </head>
    <body>
        <table border="1" id="mytab">
            <caption> My Table Caption </caption>
            <thead>
```



```
<tr>
    <th> ID </th>
    <th> Name </th>
    <th> Course </th>
</tr>
</thead>
<tbody>
    <tr>
        <td> 11 </td>
        <td> Rahul </td>
        <td> Course </td>
    </tr>
    <tr>
        <td> 2 </td>
        <td> Sonam </td>
        <td> PHP </td>
    </tr>
</tbody>
<tfoot>
    <tr>
        <td> 33 </td>
        <td> Rohan </td>
        <td> JSP </td>
    </tr>
</tfoot>
</table>
<script>
    var table = document.getElementById
                ("mytab");
    var tcap = table.createCaption();
    tcap.innerHTML = "New Caption";
```

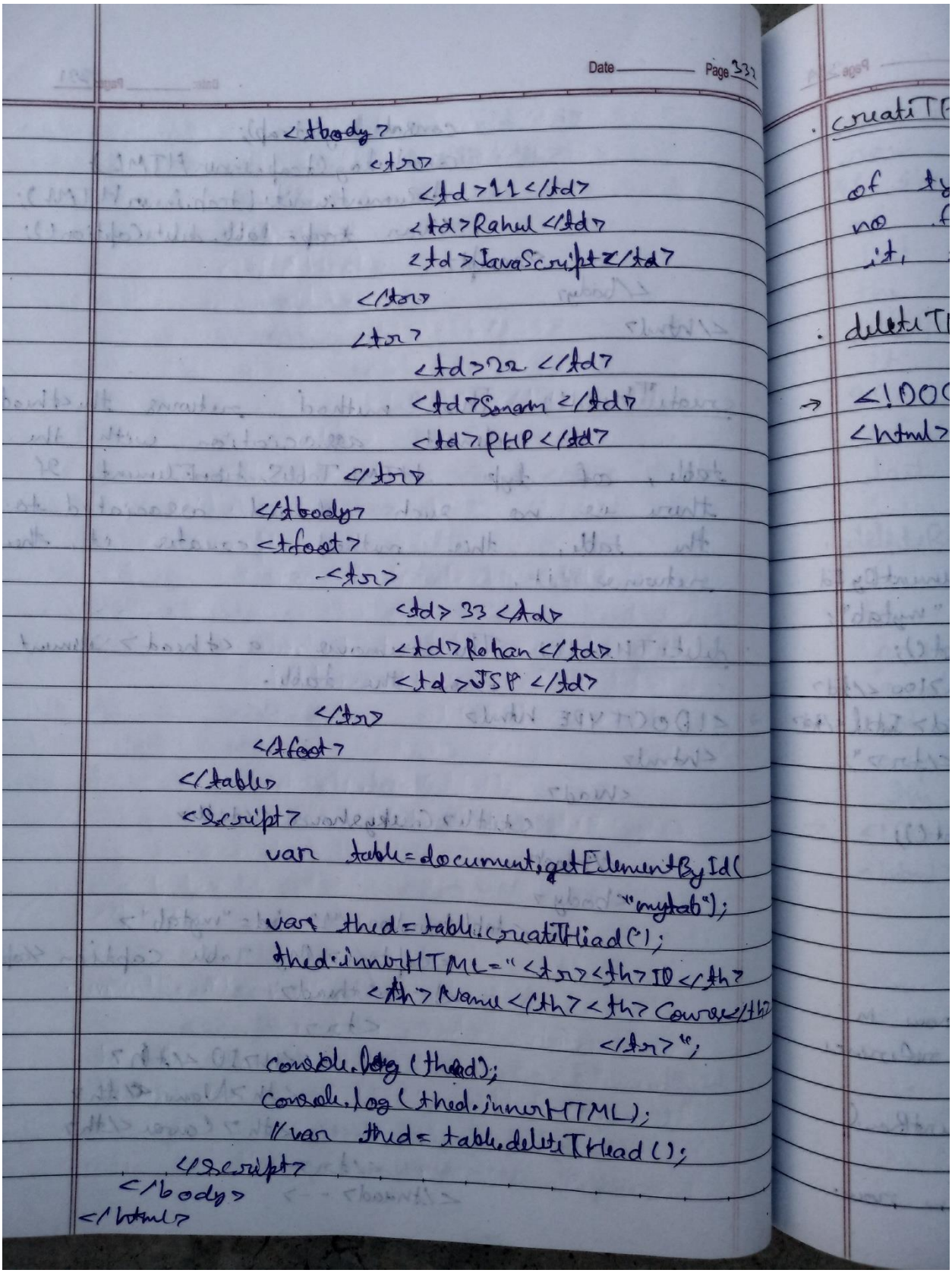


```
330                                                    Date:_____ Page: 331

        console.log(tcap);
        console.log(tcap.innerHTML);
        document.write(tcap.innerHTML);
        //var tcap = table.deleteCaption();
    </script>
</body>
</html>
```

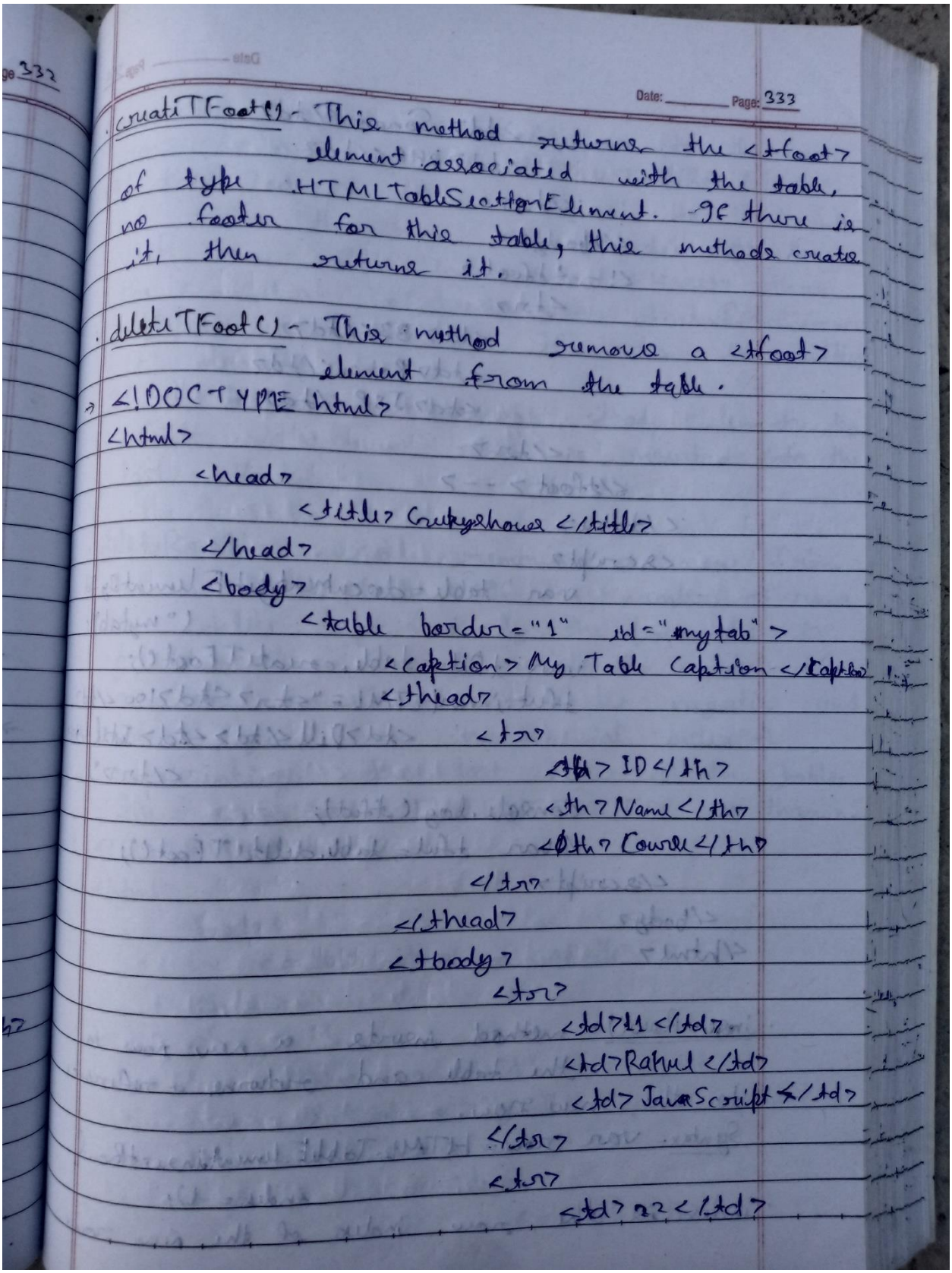
createTHead() - This method returns the <thead> element association with the table, of type .HTMLTableSectionElement. If there is no such element associated to the table, this method create it, then returns it.

deleteTHead() - The removes a <thead> element from the table.

```
<!DOCTYPE html>
<html>
    <head>
        <title> Geekyshows </title>
    </head>
    <body>
        <table border="1" id="mytab">
            <caption> My Table Caption </caption>
            <!--<thead>
                <tr>
                    <th> ID </th>
                    <th> Name </th>
                    <th> Course </th>
                </tr>
            </thead> -->
```



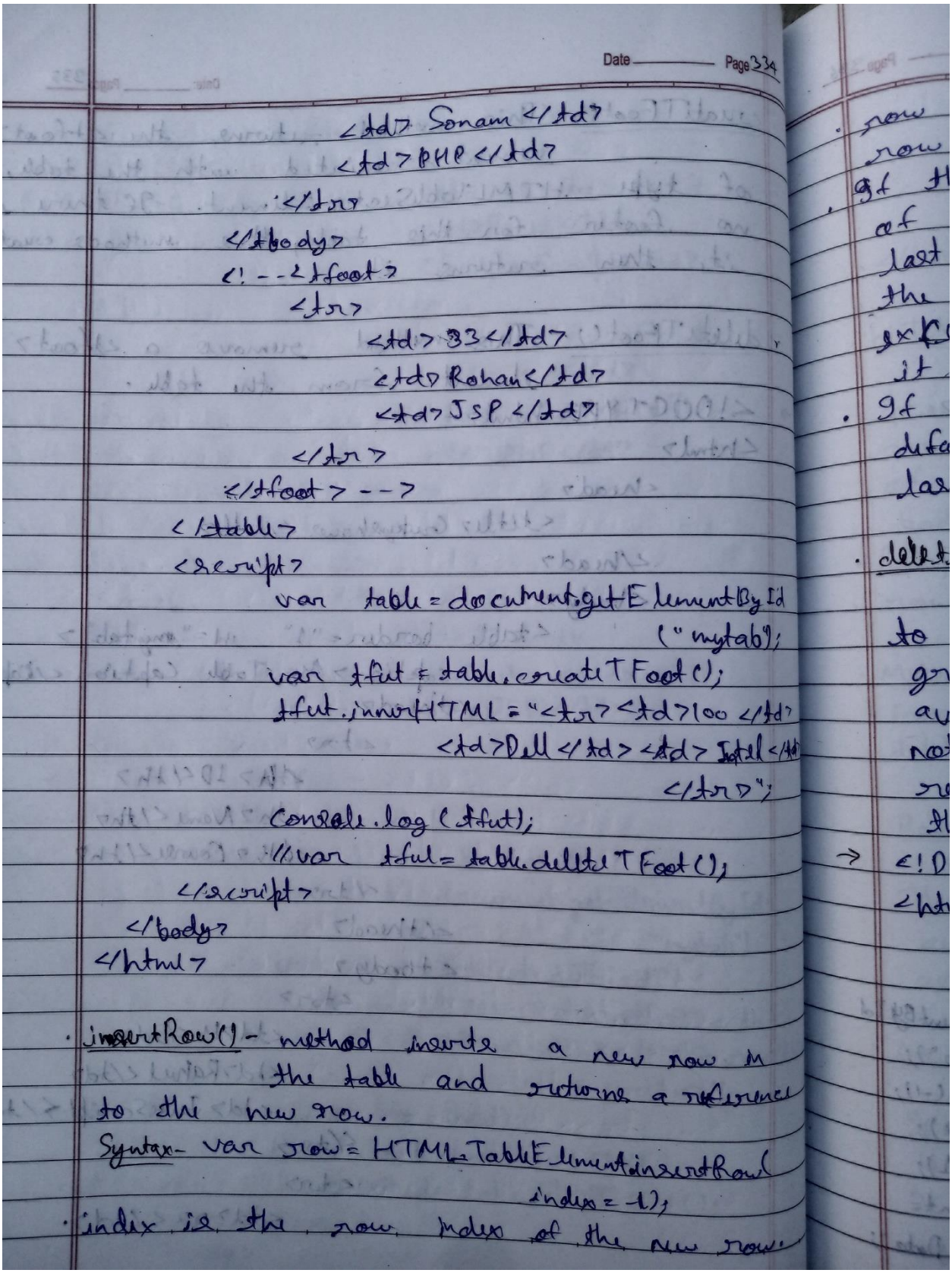
```
<tbody>
    <tr>
        <td>11</td>
        <td>Rahul</td>
        <td>JavaScript</td>
    </tr>
    <tr>
        <td>22</td>
        <td>Sonam</td>
        <td>PHP</td>
    </tr>
</tbody>
<tfoot>
    <tr>
        <td>33</td>
        <td>Rohan</td>
        <td>JSP</td>
    </tr>
</tfoot>
</table>
<script>
    var table=document.getElementById(
                "mytab");
    var thead=table.createTHead();
    thead.innerHTML="<tr><th>ID</th>
        <th>Name</th><th>Course</th
                </tr>";
    console.log(thead);
    console.log(thead.innerHTML);
    //var thead=table.deleteTHead();
</script>
</body>
</html>
```



createTFoot() - This method returns the <tfoot> element associated with the table, of type HTMLTableSectionElement. If there is no footer for this table, this methods create it, then returns it.

deleteTFoot() - This method removes a <tfoot> element from the table.

```
<!DOCTYPE html>
<html>
    <head>
        <title> Geekyshows </title>
    </head>
    <body>
        <table border="1" id="mytab">
            <caption> My Table Caption </caption>
            <thead>
                <tr>
                    <th> ID </th>
                    <th> Name </th>
                    <th> Course </th>
                </tr>
            </thead>
            <tbody>
                <tr>
                    <td> 11 </td>
                    <td> Rahul </td>
                    <td> JavaScript </td>
                </tr>
                <tr>
                    <td> 22 </td>
```

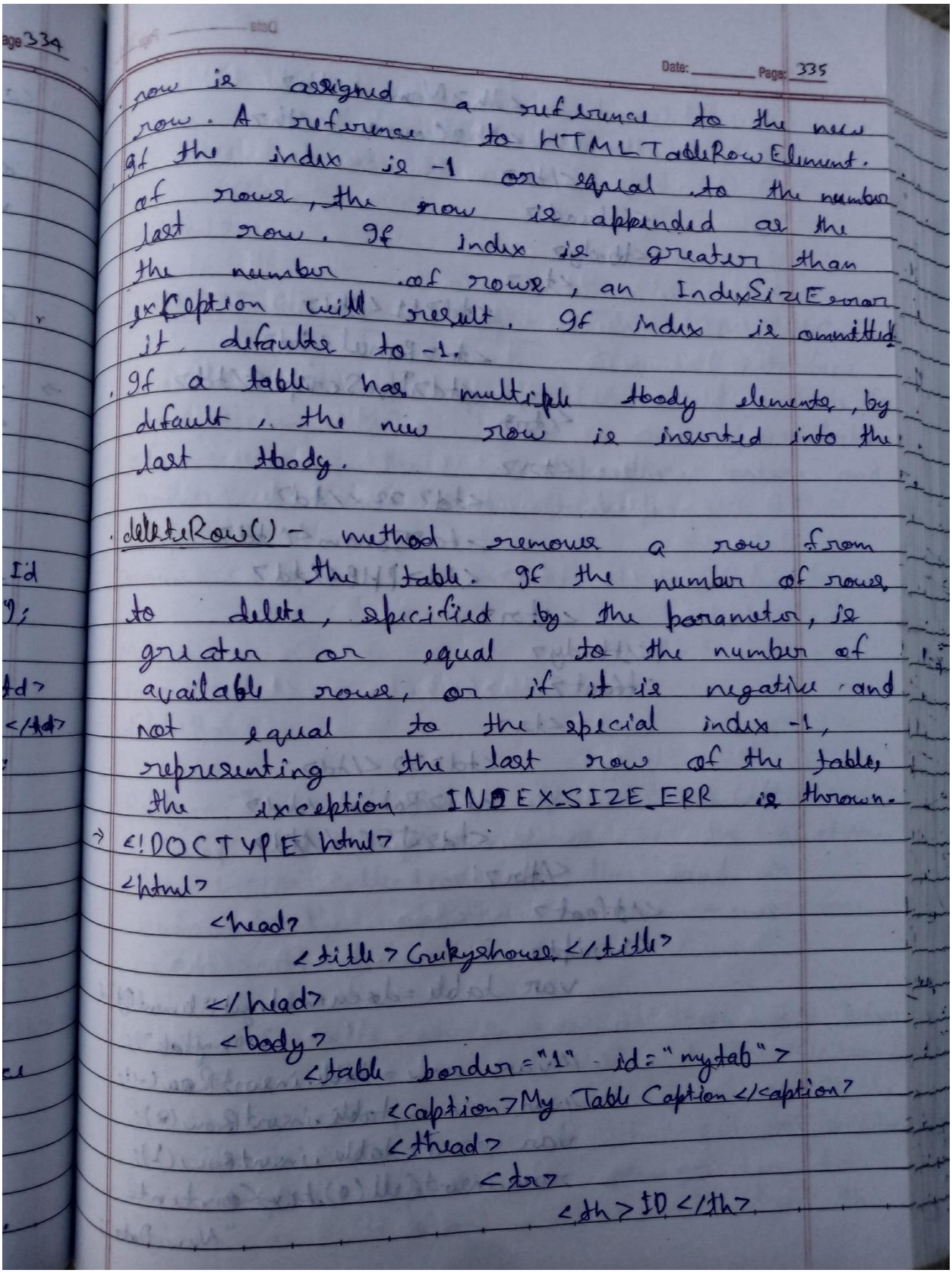


```
            <td> Sonam </td>
            <td> PHP </td>
        </tr>
    </tbody>
    <!--<tfoot>
        <tr>
            <td> 33 </td>
            <td> Rohan </td>
            <td> JSP </td>
        </tr>
    </tfoot> -->
</table>
<script>
    var table = document.getElementById
                ("mytab");
    var tfut = table.createTFoot();
    tfut.innerHTML = "<tr> <td> 100 </td>
        <td> Dell </td> <td> Intel </td>
        </tr>";
    console.log(tfut);
    //var tful = table.deleteTFoot();
</script>
</body>
</html>
```

- insertRow() - method inserts a new row in the table and returns a reference to the new row.

Syntax- var row = HTMLTableElement.insertRow(index = -1);

- index is the row index of the new row.

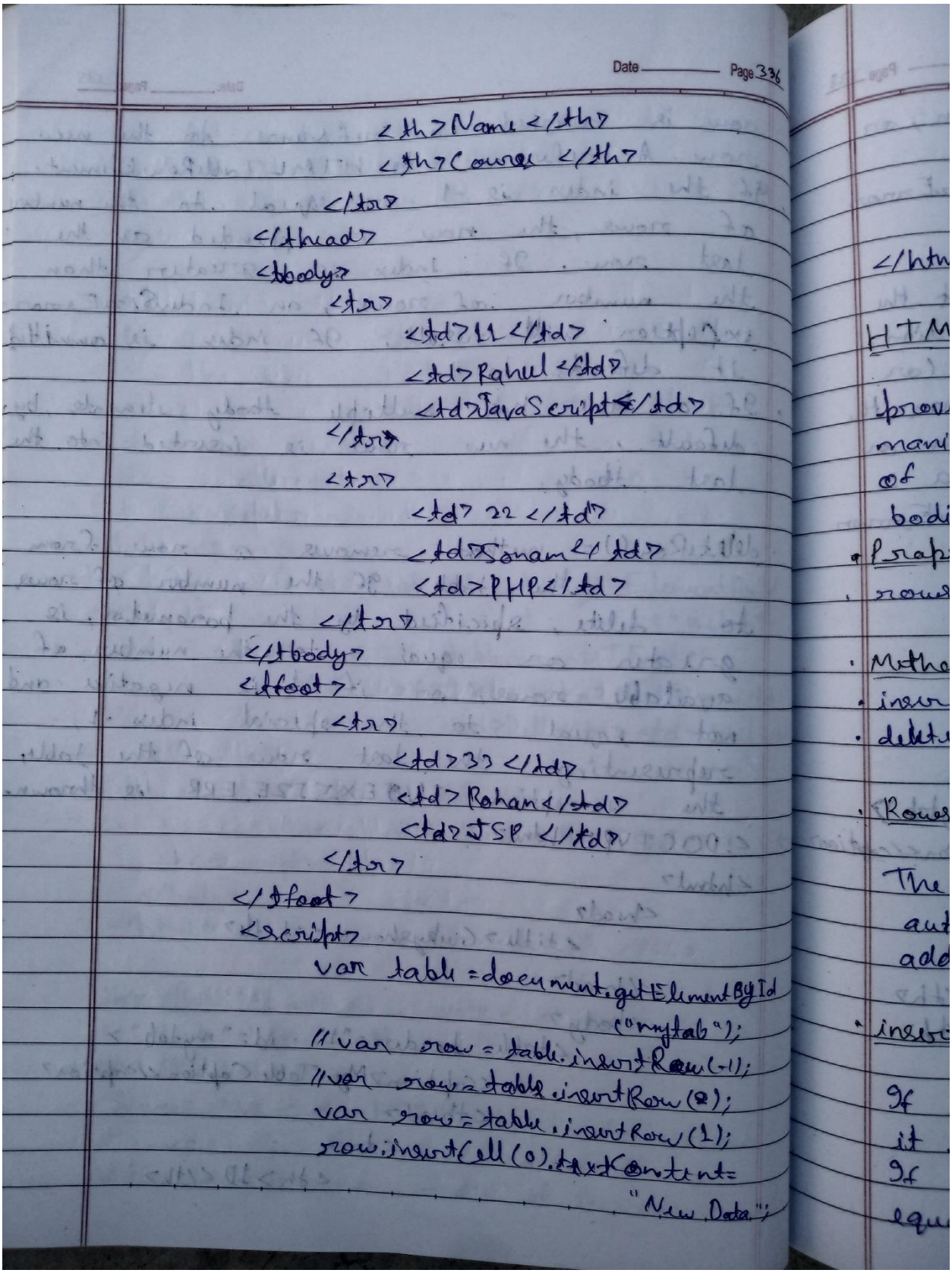


row is assigned a reference to the new row. A reference to HTMLTableRowElement. If the index is -1 or equal to the number of rows, the row is appended as the last row. If index is greater than the number of rows, an IndexSizeError exception will result. If index is omitted it defaults to -1.

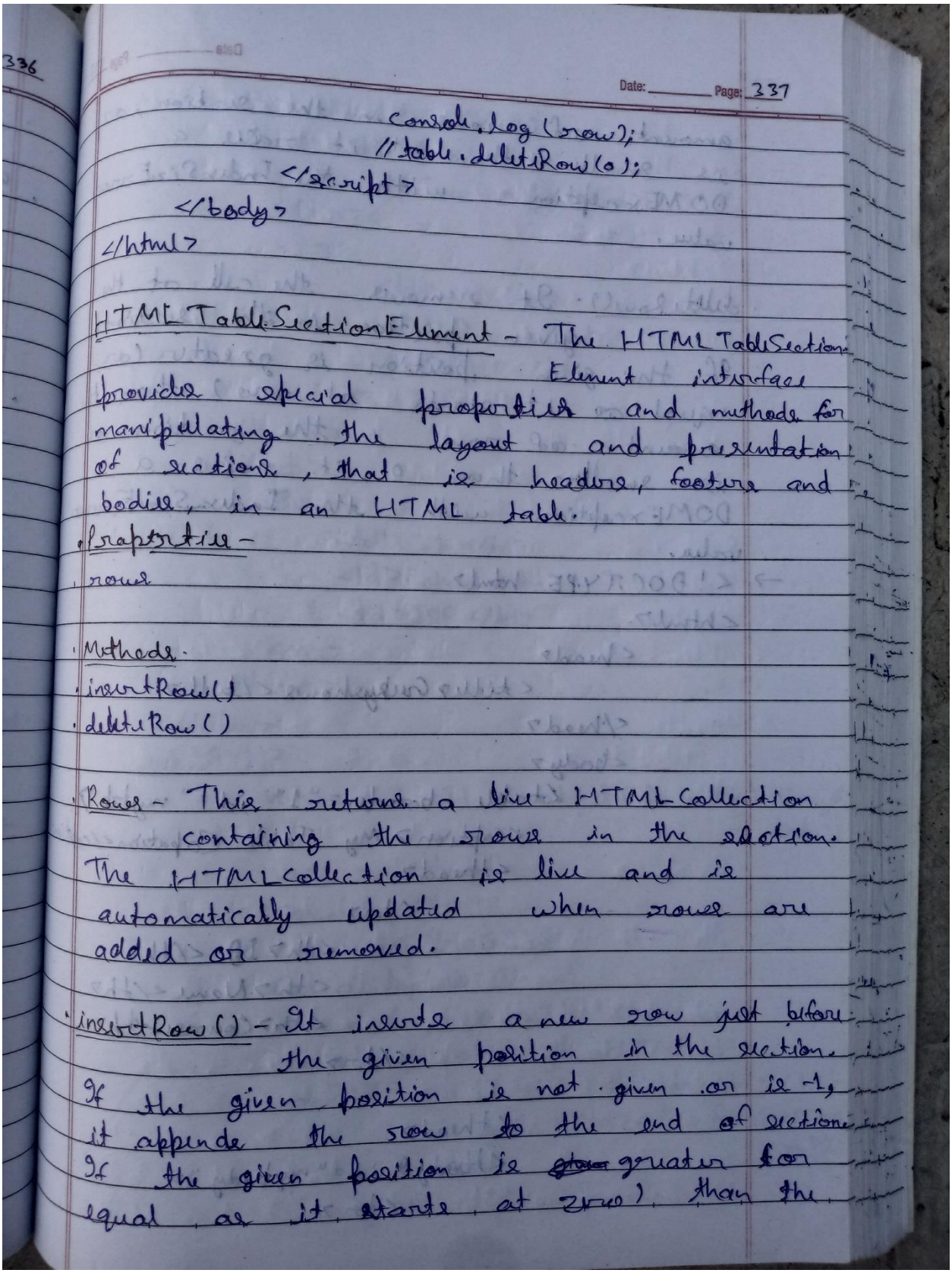
If a table has multiple tbody elements, by default, the new row is inserted into the last tbody.

deleteRow() - method removes a row from the table. If the number of rows to delete, specified by the parameter, is greater or equal to the number of available rows, or if it is negative and not equal to the special index -1, representing the last row of the table, the exception INDEX_SIZE_ERR is thrown.

```
<!DOCTYPE html>
<html>
    <head>
        <title> Geekyshows </title>
    </head>
    <body>
        <table border="1" id="mytab">
            <caption>My Table Caption</caption>
            <thead>
                <tr>
                    <th> ID </th>
```



```
            <th>Name</th>
            <th>Course</th>
        </tr>
    </thead>
    <tbody>
        <tr>
            <td>11</td>
            <td>Rahul</td>
            <td>JavaScript</td>
        </tr>
        <tr>
            <td>22</td>
            <td>Sonam</td>
            <td>PHP</td>
        </tr>
    </tbody>
    <tfoot>
        <tr>
            <td>33</td>
            <td>Rohan</td>
            <td>JSP</td>
        </tr>
    </tfoot>
    <script>
        var table = document.getElementById
                                ("mytab");
        // var row = table.insertRow(-1);
        // var row = table.insertRow(2);
        var row = table.insertRow(1);
        row.insertCell(0).textContent=
                                "New Data";
```



336

Date: Page: 337

```
        console.log(row);
        //table.deleteRow(0);
    </script>
</body>
</html>
```

HTMLTableSectionElement - The HTMLTableSectionElement interface provides special properties and methods for manipulating the layout and presentation of sections, that is headers, footers and bodies, in an HTML table.

Properties -

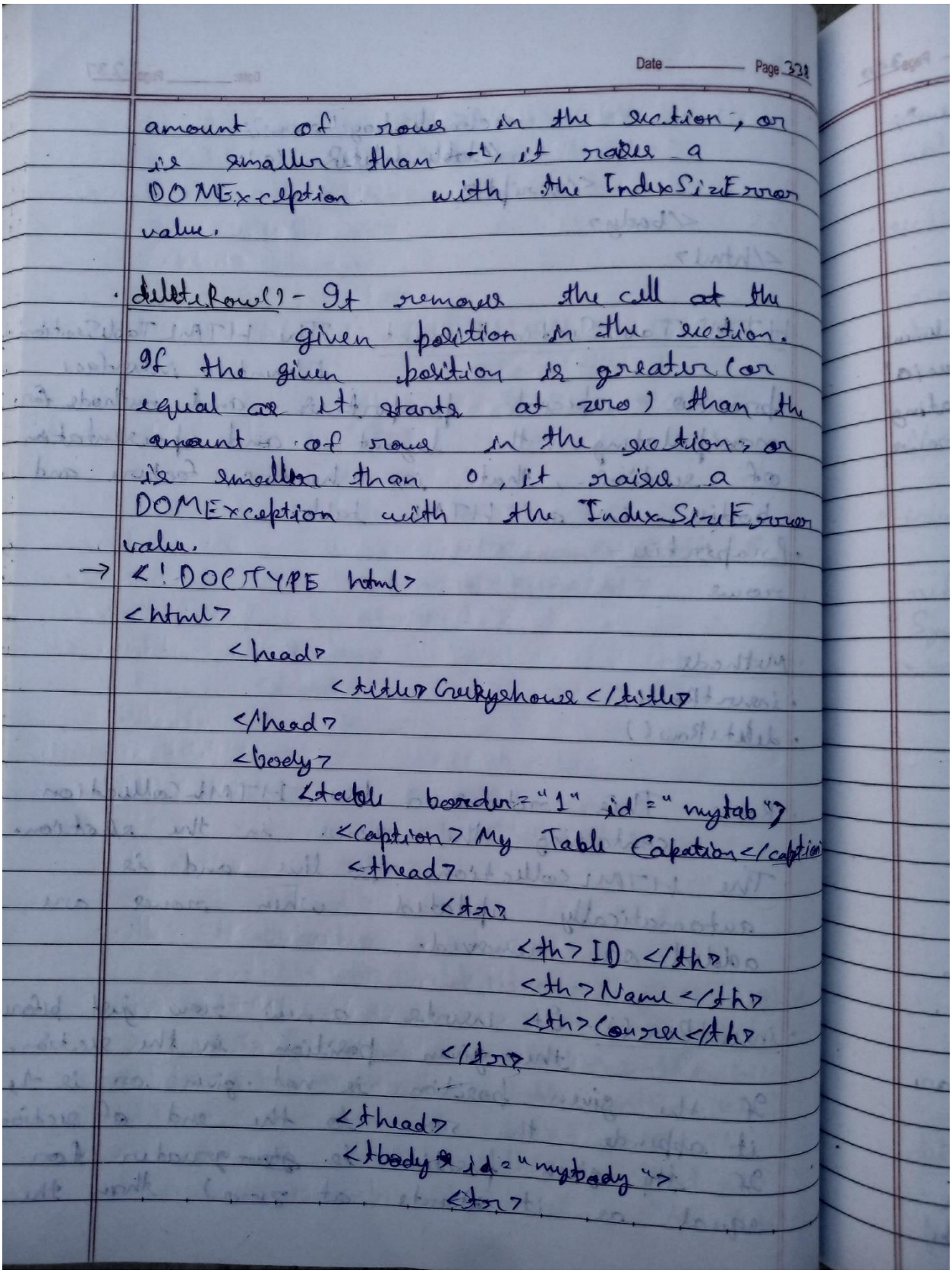
- rows

Methods -

- insertRow()
- deleteRow()

- Rows - This returns a live HTMLCollection containing the rows in the section. The HTMLCollection is live and is automatically updated when rows are added or removed.

- insertRow() - It inserts a new row just before the given position in the section. If the given position is not given or is -1, it appends the row to the end of section. If the given position is ~~zero~~ greater than equal or it starts at zero, than the

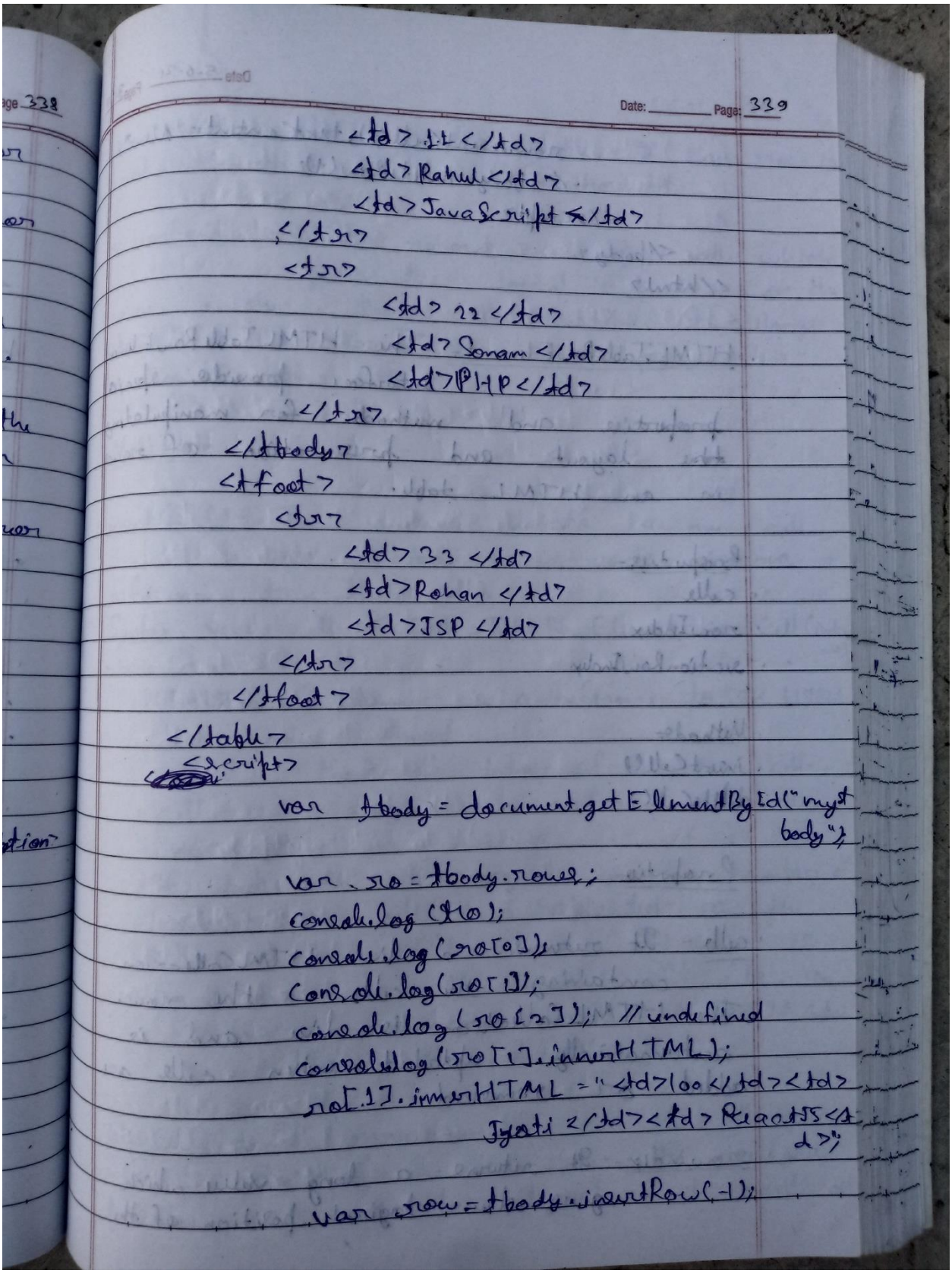


Date _____ Page 321

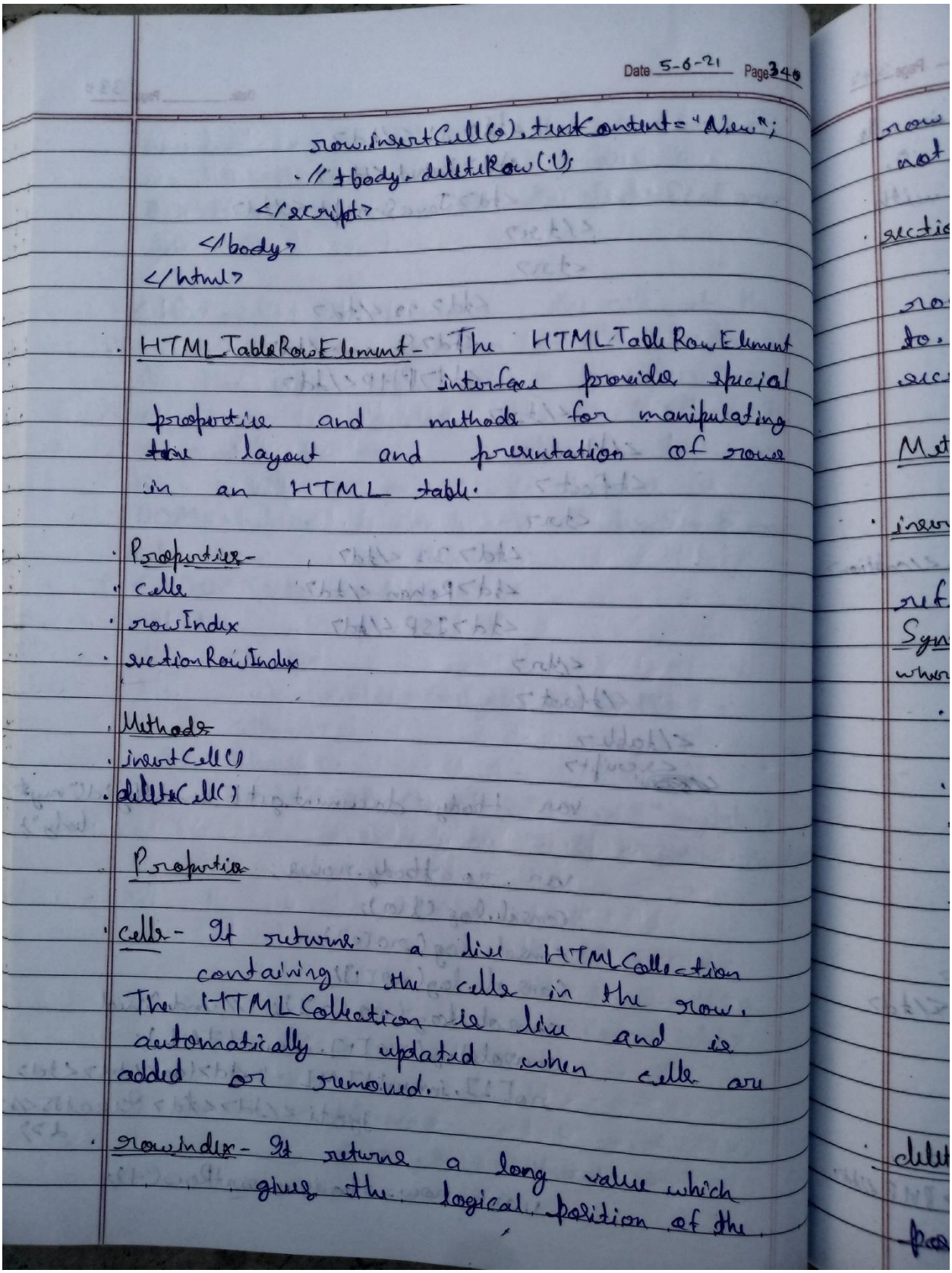
amount of rows in the section, or is smaller than -1, it raises a DOMException with the IndexSizeError value.

- deleteRow() - It removes the call at the given position in the section. If the given position is greater (or equal as it starts at zero) than the amount of rows in the section, or is smaller than 0, it raises a DOMException with the IndexSizeError value.

→
```
<!DOCTYPE html>
<html>
    <head>
        <title> Geekyshows </title>
    </head>
    <body>
        <table border="1" id="mytab">
            <caption> My Table Caption </caption>
            <thead>
                <tr>
                    <th> ID </th>
                    <th> Name </th>
                    <th> Course </th>
                </tr>
            </thead>
            <tbody id="mytbody">
                <tr>
```



```
        <td> 11 </td>
        <td> Rahul </td>
        <td> JavaScript </td>
    </tr>
    <tr>
        <td> 22 </td>
        <td> Sonam </td>
        <td> PHP </td>
    </tr>
</tbody>
<tfoot>
    <tr>
        <td> 33 </td>
        <td> Rohan </td>
        <td> JSP </td>
    </tr>
</tfoot>
</table>
<script>
    var tbody = document.getElementById("mytbody");

    var ro = tbody.rows;
    console.log(tbo);
    console.log(ro[0]);
    Console.log(ro[1]);
    console.log(ro[2]); // undefined
    console.log(ro[1].innerHTML);
    ro[1].innerHTML = "<td>100</td><td>Jyoti</td><td>React</td>";

    var row = tbody.insertRow(-1);
```



Date 5-6-21 Page 340

```
        row.insertCell(0).textContent = "New";
        // tbody.deleteRow(1);
    </script>
</body>
</html>
```

HTMLTableRowElement - The HTMLTableRowElement interface provides special properties and methods for manipulating the layout and presentation of rows in an HTML table.

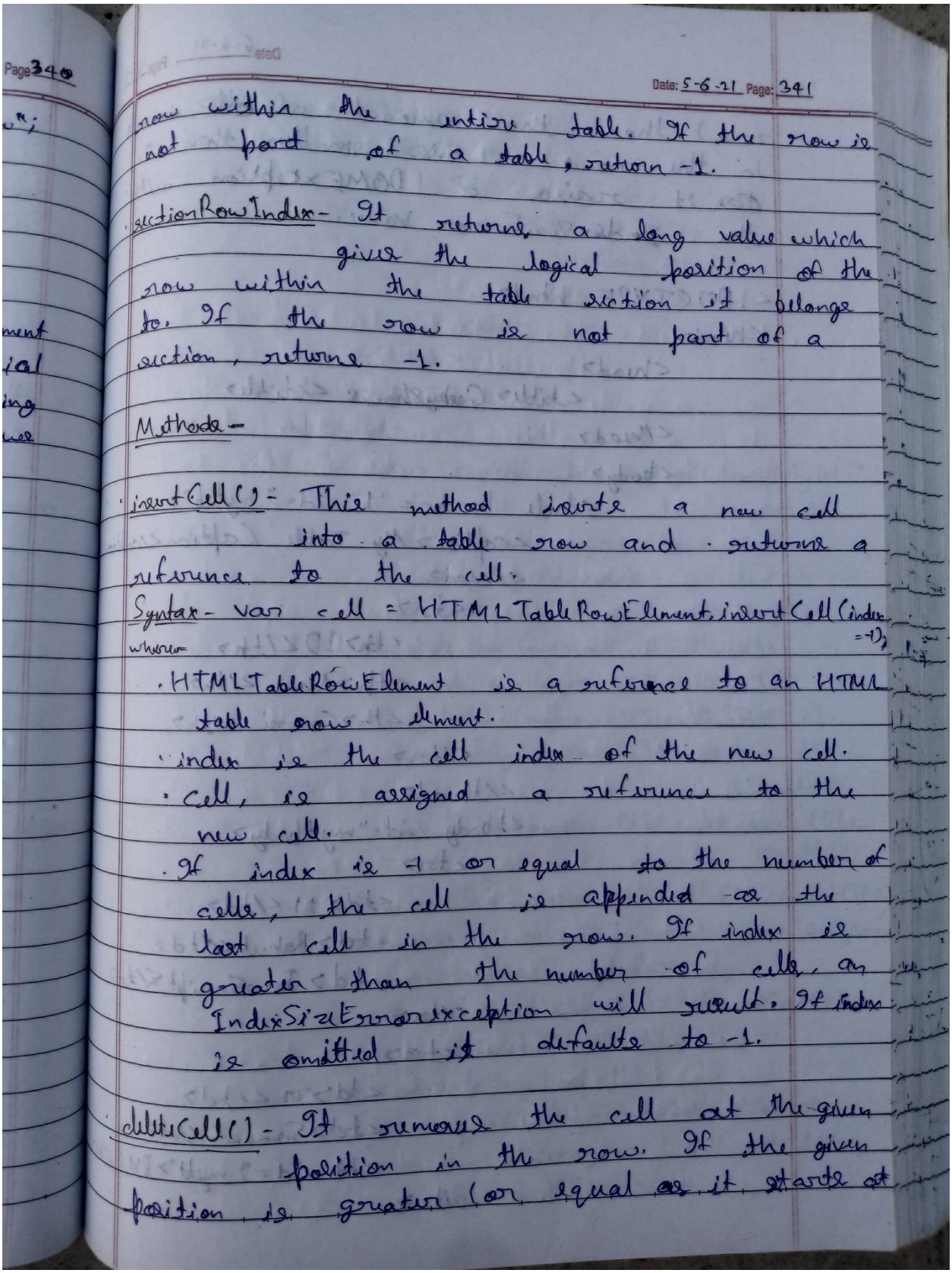
Properties -
- cells
- rowIndex
- sectionRowIndex

Methods
- insertCell()
- deleteCell()

Properties

cells - It returns a live HTMLCollection containing the cells in the row. The HTMLCollection is live and is automatically updated when cells are added or removed.

rowIndex - It returns a long value which gives the logical position of the



Page 340

Date: 5-6-21 Page: 341

row within the entire table. If the row is not part of a table, return -1.

sectionRowIndex - It returns a long value which gives the logical position of the row within the table section it belongs to. If the row is not part of a section, returns -1.

Methods -

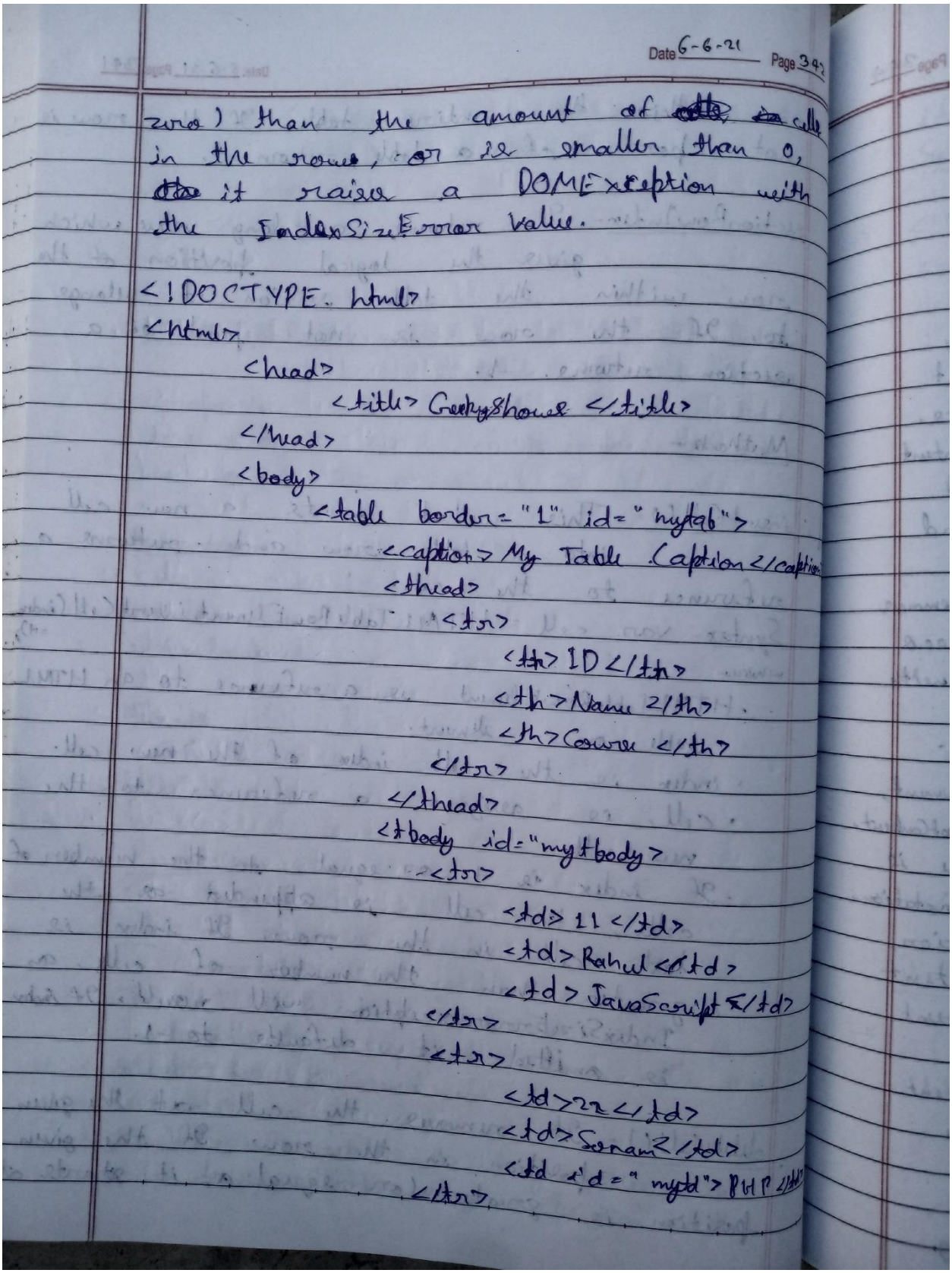
insertCell() - This method inserts a new cell into a table row and returns a reference to the cell.

Syntax - var cell = HTMLTableRowElement.insertCell(index=-1);

where -

- HTMLTableRowElement is a reference to an HTML table row element.
- index is the cell index of the new cell.
- cell, is assigned a reference to the new cell.
- If index is -1 or equal to the number of cells, the cell is appended as the last cell in the row. If index is greater than the number of cells, an IndexSizeError exception will result. If index is omitted it defaults to -1.

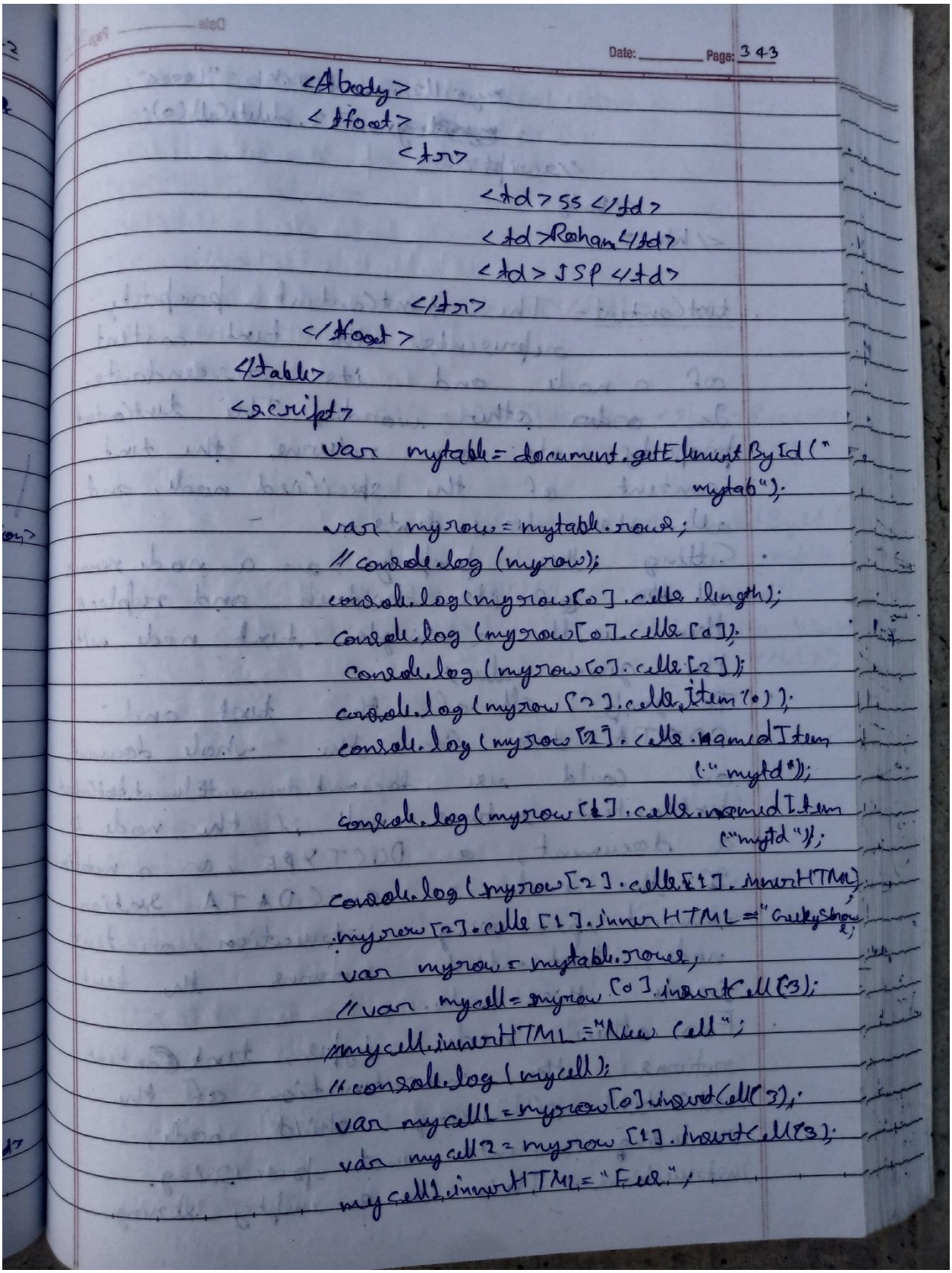
deleteCell() - It removes the cell at the given position in the row. If the given position is greater (or equal as it starts at



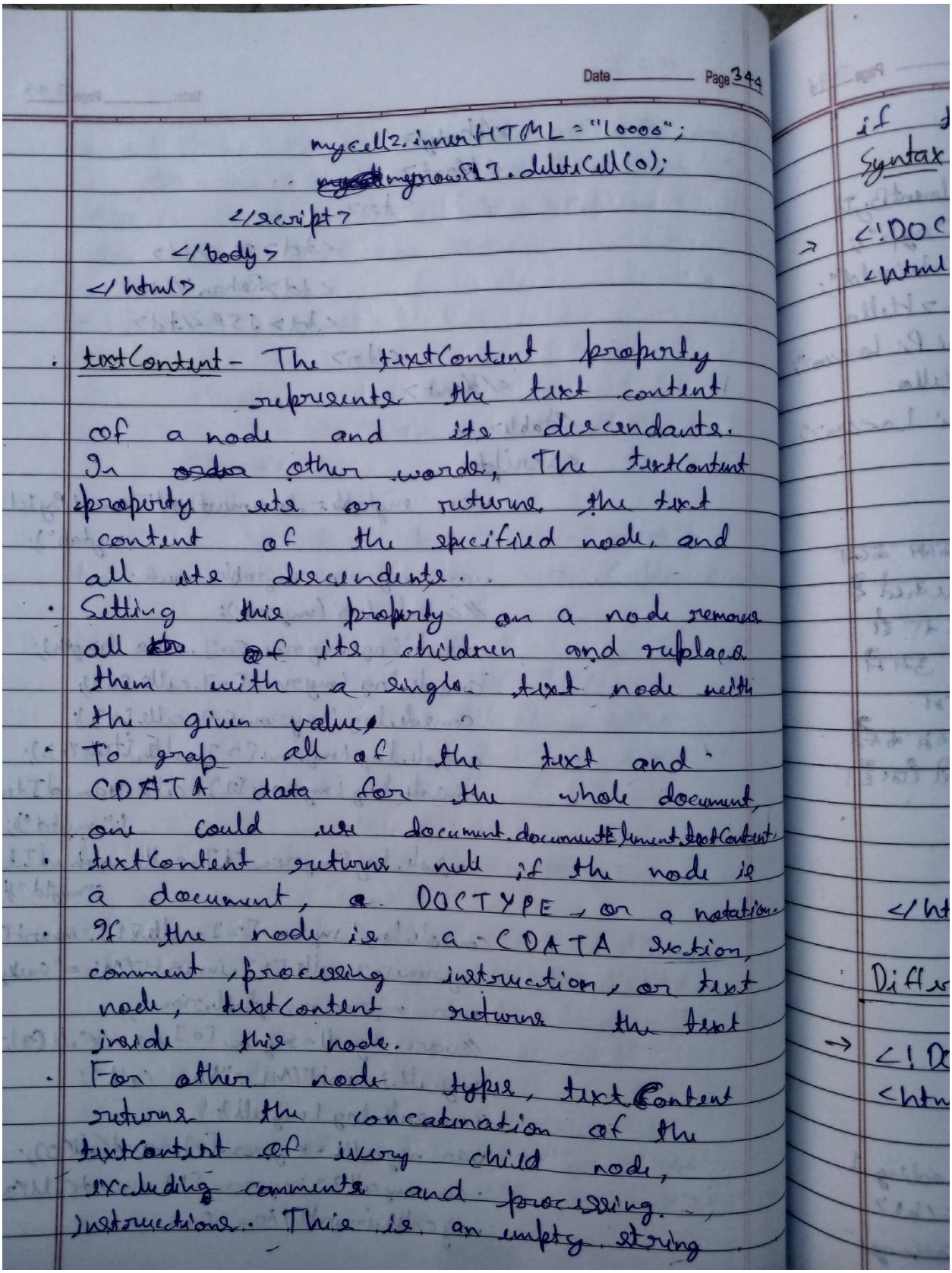
Date 6-6-21 Page 342

zero) than the amount of cells in the rows, or is smaller than 0, it raise a DOMException with the IndexSizeError value.

```
<!DOCTYPE. html>
<html>
    <head>
        <title> GeekyShows </title>
    </head>
    <body>
        <table border = "1" id = "mytab">
            <caption> My Table Caption </caption>
            <thead>
                <tr>
                    <th> ID </th>
                    <th> Name </th>
                    <th> Course </th>
                </tr>
            </thead>
            <tbody id = "mytbody">
                <tr>
                    <td> 11 </td>
                    <td> Rahul </td>
                    <td> JavaScript </td>
                </tr>
                <tr>
                    <td> 22 </td>
                    <td> Sonam </td>
                    <td id = "mytd"> PHP </td>
                </tr>
```

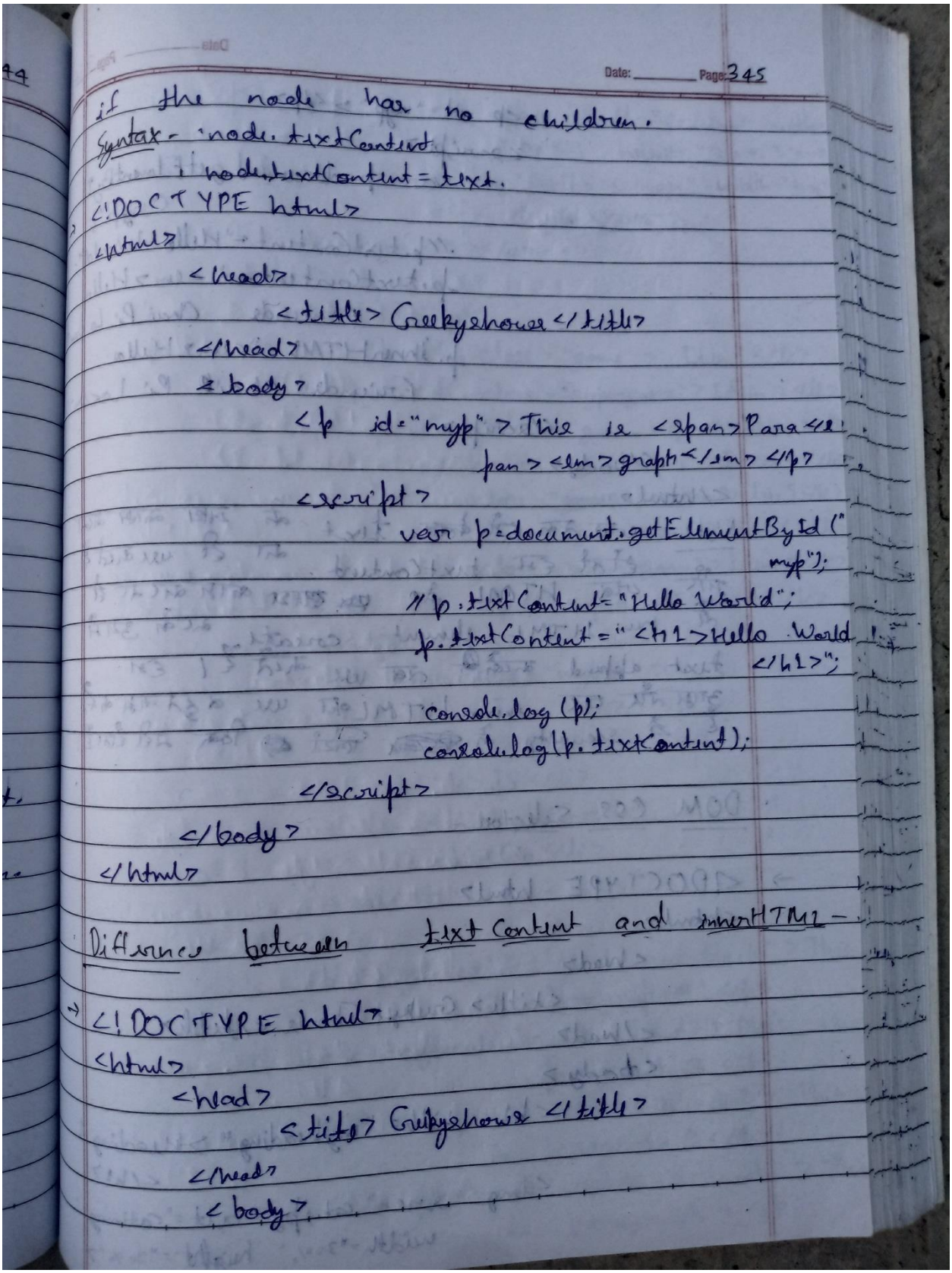


```
</tbody>
<tfoot>
    <tr>
        <td>55</td>
        <td>Rohan</td>
        <td>JSP</td>
    </tr>
</tfoot>
</table>
<script>
var mytable = document.getElementById("mytab");
var myrow = mytable.rows;
// console.log(myrow);
console.log(myrow[0].cells.length);
console.log(myrow[0].cells[0]);
console.log(myrow[0].cells[2]);
console.log(myrow[2].cells.item(0));
console.log(myrow[1].cells.namedItem("mytd"));
console.log(myrow[1].cells.namedItem("mytd"));
console.log(myrow[2].cells[1].innerHTML);
myrow[2].cells[1].innerHTML = "GeekyShows";
var myrow = mytable.rows;
// var mycell = myrow[0].insertCell(3);
mycell.innerHTML = "New Cell";
// console.log(mycell);
var mycell1 = myrow[0].insertCell(3);
var mycell2 = myrow[1].insertCell(3);
mycell1.innerHTML = "Eee";
```



```
        mycell2.innerHTML = "10000";
        mymyrow[1].deleteCell(0);
    </script>
</body>
</html>
```

- textContent - The textContent property represents the text content of a node and its descendants. In other words, The textContent property sets or returns the text content of the specified node, and all its descendants.
- Setting this property on a node removes all of its children and replaces them with a single text node with the given value.
- To grab all of the text and CDATA data for the whole document, one could use document.documentElement.textContent.
- textContent returns null if the node is a document, a DOCTYPE, or a notation.
- If the node is a CDATA section, comment, processing instruction, or text node, textContent returns the text inside this node.
- For other node types, textContent returns the concatenation of the textContent of every child node, excluding comments and processing instructions. This is an empty string



if the node has no children.

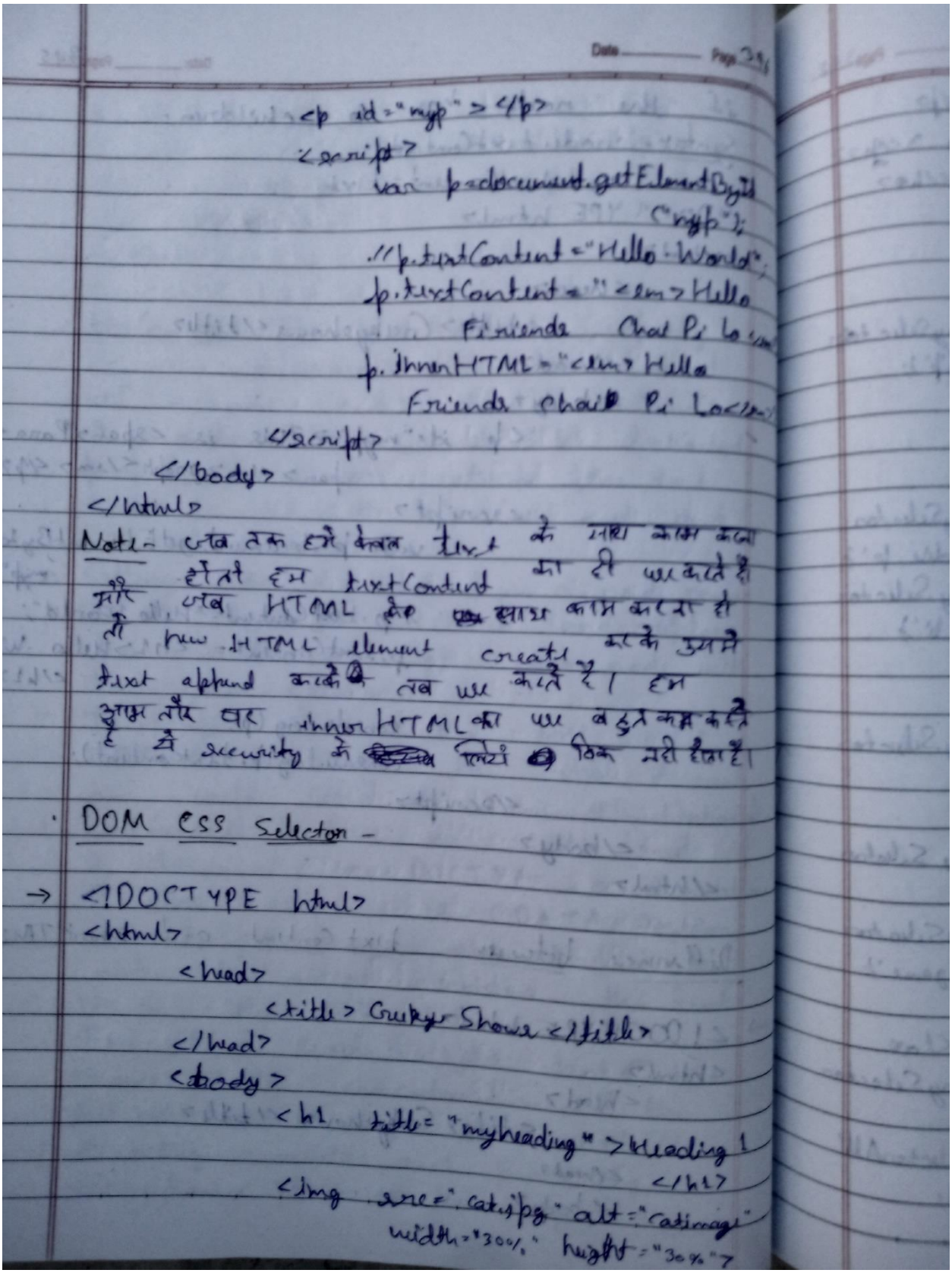
Syntax - node.textContent

node.textContent = text.

```
<!DOCTYPE html>
<html>
    <head>
        <title> Geekyshows </title>
    </head>
    <body>
        <p id="myp"> This is <span> Para </span> <em> graph </em> </p>
        <script>
            var p = document.getElementById("myp");
            // p.textContent = "Hello World";
            p.textContent = "<h1> Hello World </h1>";
            console.log(p);
            console.log(p.textContent);
        </script>
    </body>
</html>
```

Difference between textContent and innerHTML -

```
<!DOCTYPE html>
<html>
    <head>
        <title> Geekyshows </title>
    </head>
    <body>
```

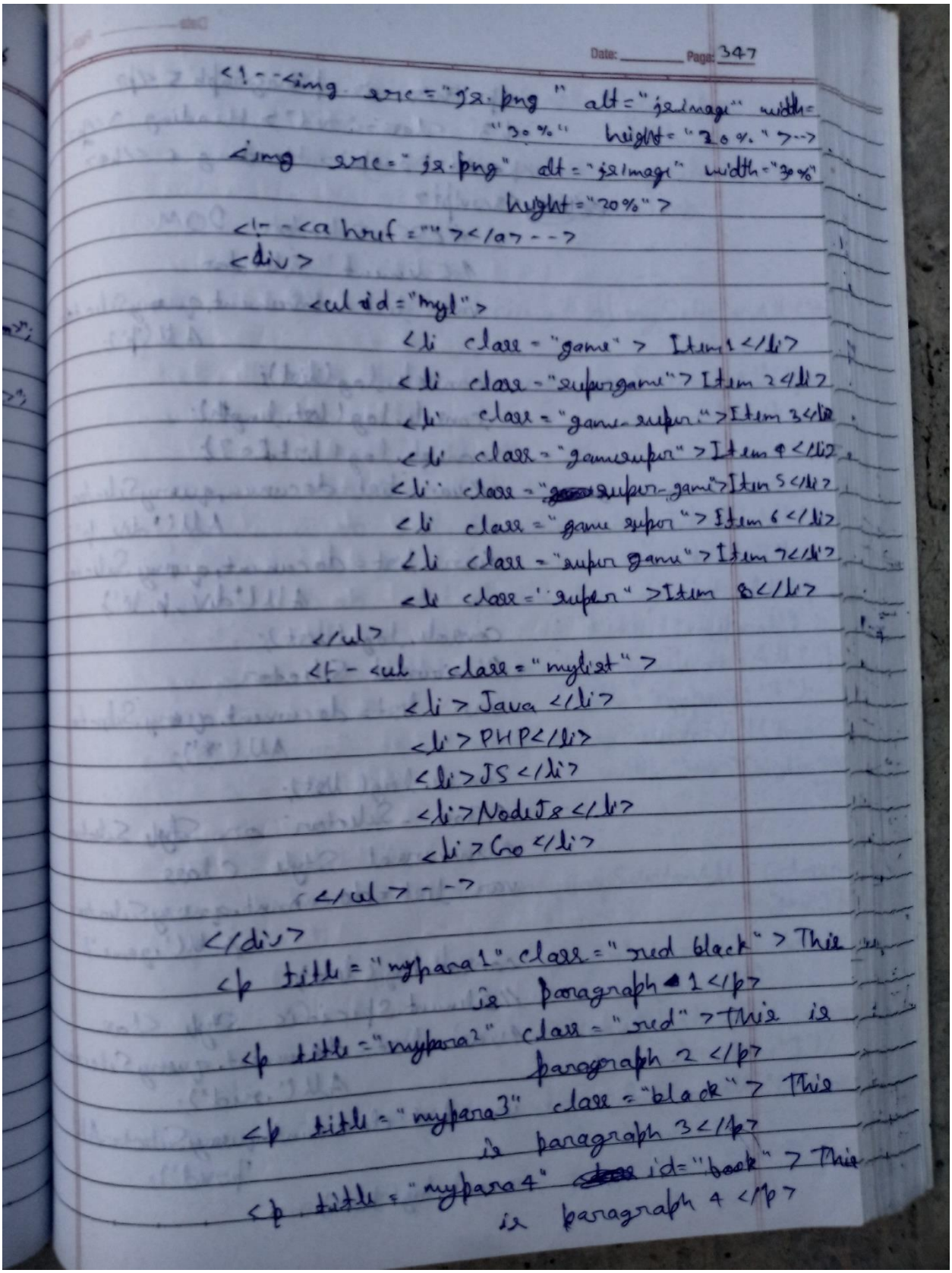


```
<p id="myp"></p>
<script>
    var p=document.getElementById
          ("myp");
    //p.textContent="Hello World";
    p.textContent="<em>Hello
        Friends Chal Pi Lo</em>";
    p.innerHTML="<em>Hello
        Friends Chal Pi Lo</em>";
</script>
</body>
</html>
```

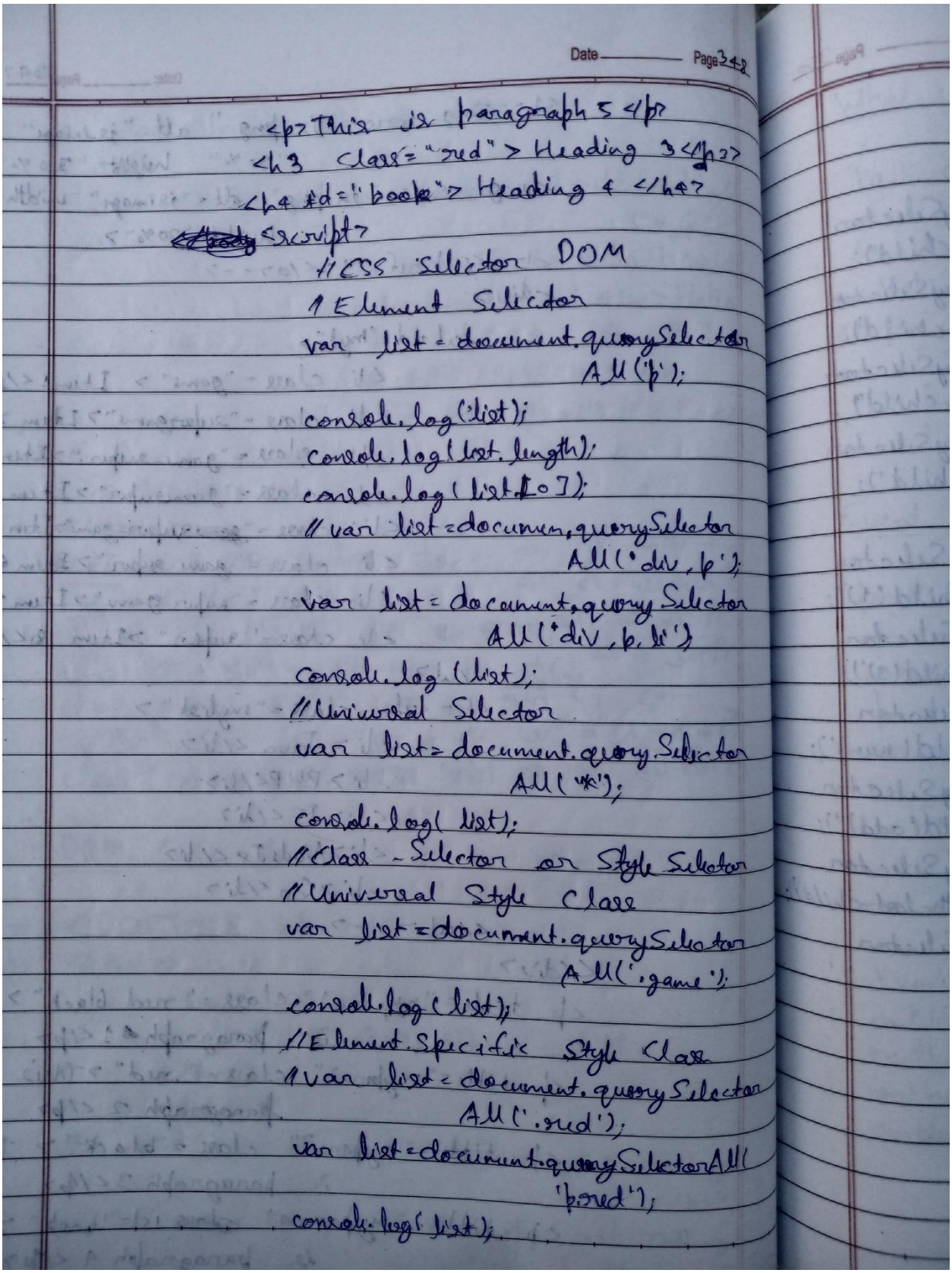
Note- जब तक हमें केवल text के साथ काम करना होता है तब हम textContent का ही use करते है और जब HTML के साथ काम करना हो तो new HTML element create करके उसमे text append करके तब use करते है। हम आम तौर पर innerHTML का use बहुत कम करते है क्योंकि security के लिये ठीक नही होता है।

DOM CSS Selector -

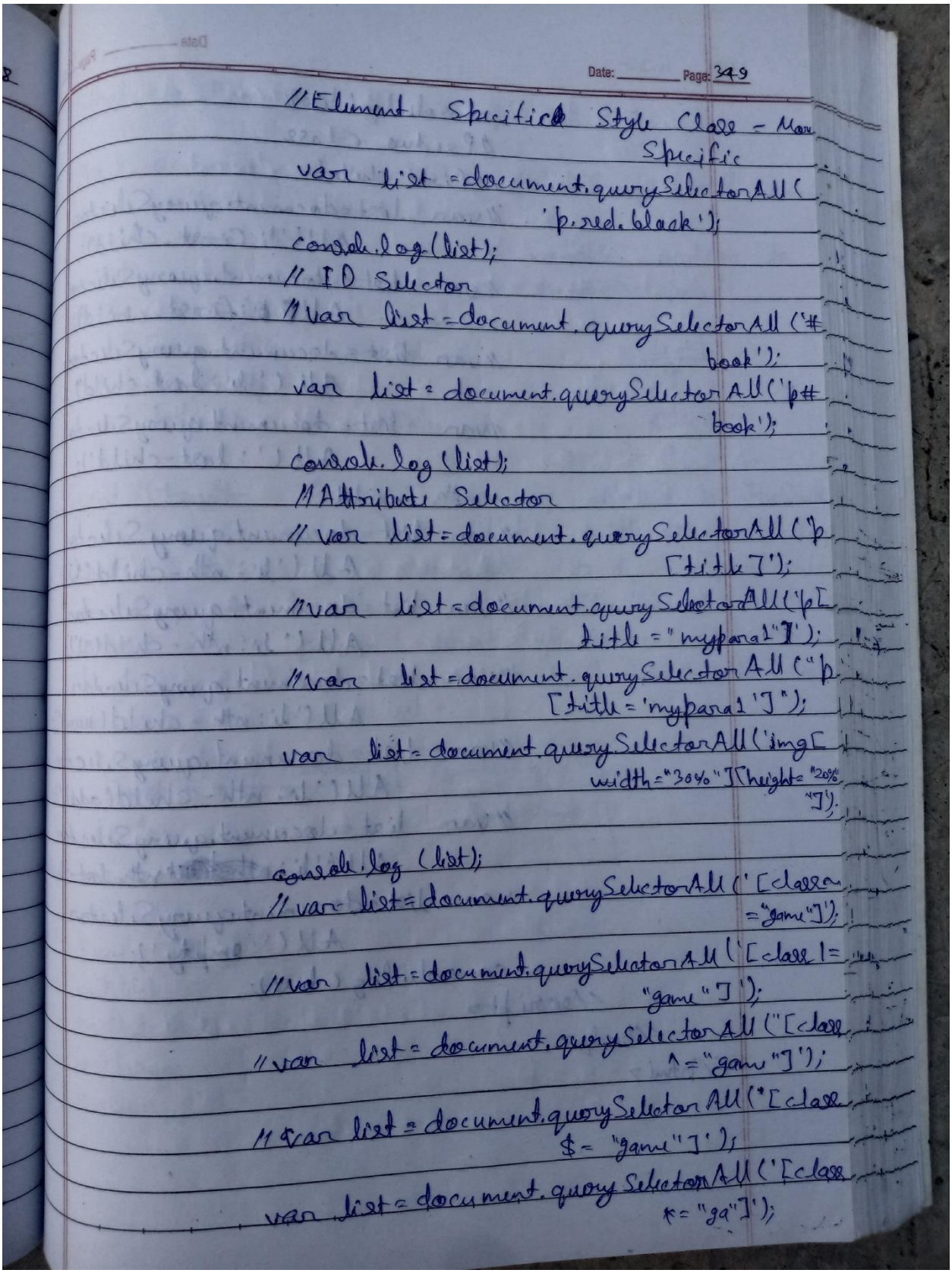
→
```
<!DOCTYPE html>
<html>
    <head>
        <title>Geekyshows</title>
    </head>
    <body>
        <h1 title="myheading">Heading 1
                                </h1>
        <img src="cat.jpg" alt="catimage"
             width="30%" height="30%">
```



```
<!-- <img src="js.png" alt="jsImage" width=
                "30%" height="20%"> -->
<img src="js.png" alt="jsImage" width="30%"
                height="20%">
<!-- <a href="#"></a> -->
<div>
    <ul id="myul">
        <li class="game"> Item 1 </li>
        <li class="supergame"> Item 2 </li>
        <li class="game-super"> Item 3 </li>
        <li class="gamesuper"> Item 4 </li>
        <li class="super-game"> Item 5 </li>
        <li class="game super"> Item 6 </li>
        <li class="super game"> Item 7 </li>
        <li class="super"> Item 8 </li>
    </ul>
    <!-- <ul class="mylist">
        <li> Java </li>
        <li> PHP </li>
        <li> JS </li>
        <li> NodeJs </li>
        <li> Go </li>
    </ul> -->
</div>
<p title="mypara1" class="red black"> This
        is paragraph 1 </p>
<p title="mypara2" class="red"> This is
        paragraph 2 </p>
<p title="mypara3" class="black"> This
        is paragraph 3 </p>
<p title="mypara4" id="book"> This
        is paragraph 4 </p>
```

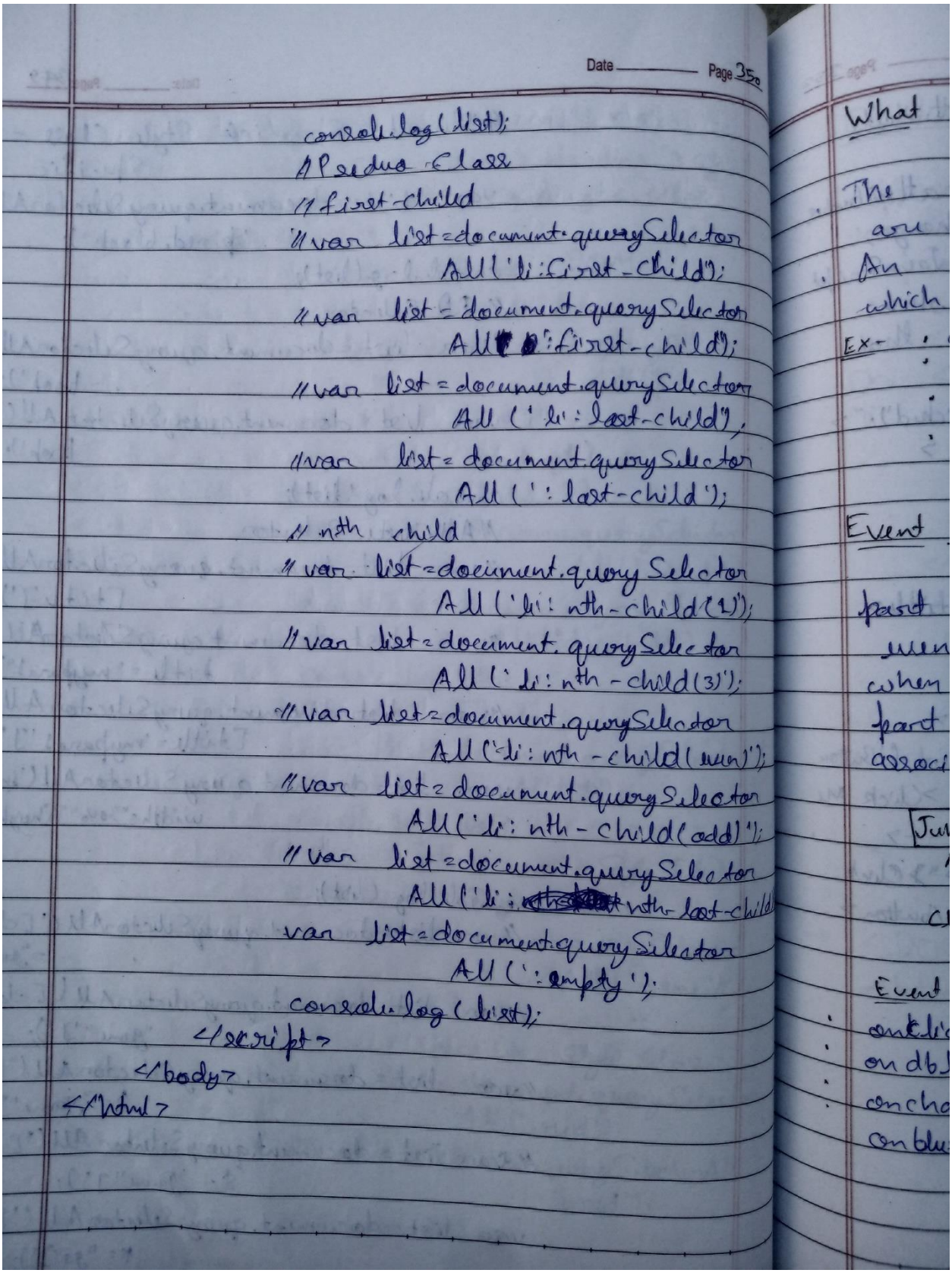


```
<p> This is paragraph 5 </p>
<h3 class="red"> Heading 3 </h3>
<h4 id="book"> Heading 4 </h4>
<script>
// CSS Selector DOM
// Element Selector
var list = document.querySelectorAll('p');
console.log(list);
console.log(list.length);
console.log(list[0]);
// var list=documen.querySelectorAll('div,p');
var list = document.querySelectorAll('div,b,li');
console.log(list);
// Universal Selector
var list=document.querySelectorAll('*');
consol.log(list);
// Class - Selector or Style Selector
// Universal Style Class
var list=document.querySelectorAll('.game');
consol.log(list);
// Element Specific Style Class
// var list = document.querySelectorAll('.red');
var list=document.querySelectorAll('bored');
consol.log(list);
```



Date: Page: 349

```
// Element Specific Style Class - More Specific
var list = document.querySelectorAll('p.red.black');
console.log(list);
// ID Selector
// var list = document.querySelectorAll('#book');
var list = document.querySelectorAll('p#book');
console.log(list);
// Attribute Selector
// var list = document.querySelectorAll('p[title]');
// var list = document.querySelectorAll('p[title="mypara1"]');
// var list = document.querySelectorAll("p[title='mypara1']");
var list = document.querySelectorAll('img[width="30%"][height="20%"]');

console.log(list);
// var list = document.querySelectorAll('[class="game"]');
// var list = document.querySelectorAll('[class!="game"]');
// var list = document.querySelectorAll("[class^="game"]");
// var list = document.querySelectorAll("[class$="game"]");
var list = document.querySelectorAll('[class*="ga"]');
```



```
console.log(list);
// Pseudo Class
// first-child
// var list = document.querySelector
        All('li:first-child');
// var list = document.querySelector
        All('li:first-child');
// var list = document.querySelector
        All('li:last-child');
// var list = document.querySelector
        All(':last-child');
// nth child
// var list = document.querySelector
        All('li:nth-child(2)');
// var list = document.querySelector
        All('li:nth-child(3)');
// var list = document.querySelector
        All('li:nth-child(even)');
// var list = document.querySelector
        All('li:nth-child(odd)');
// var list = document.querySelector
        All('li:[illegible]nth-last-child
var list = document.querySelector
        All(':empty');
console.log(list);
</script>
</body>
</html>
```

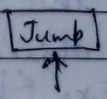
Date: 7-6-21 Page: 351

What is Event -

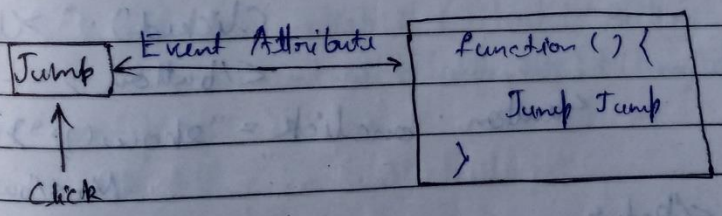
The actions to which JavaScript can respond are called Events.

An event is some notable action to which a script can ~~respond~~ respond.

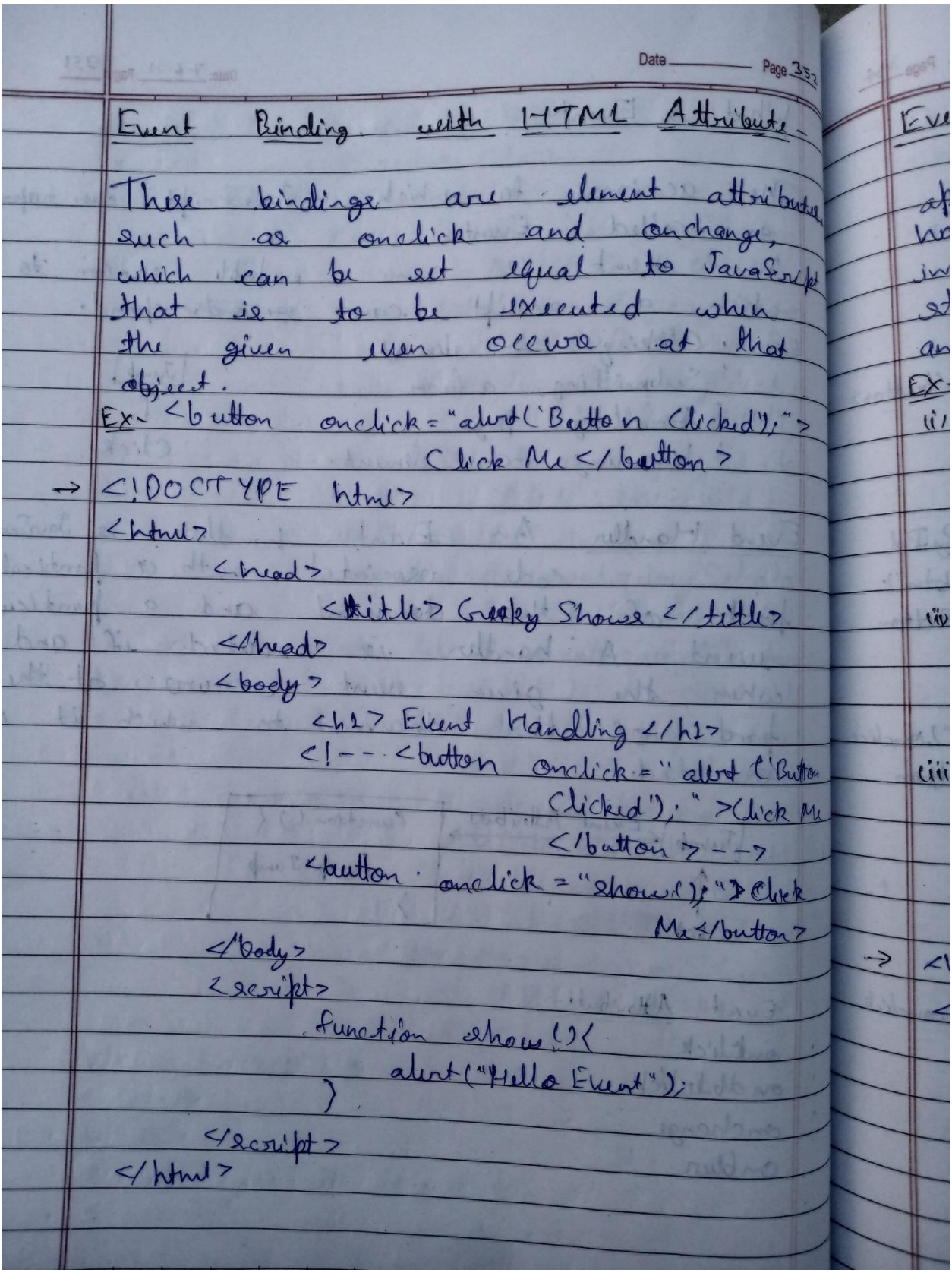
Ex- : Clicking an element
: Submitting a form
: Scrolling page
: Hovering an element.



Event Handler - An Event handler is JavaScript code associated with a particular part of the document and a particular event. A handler is executed if and when the given event occurs at the part of the document to which it is associated.



Event Attribute -
- onclick
- ondblclick
- onchange
- onblur



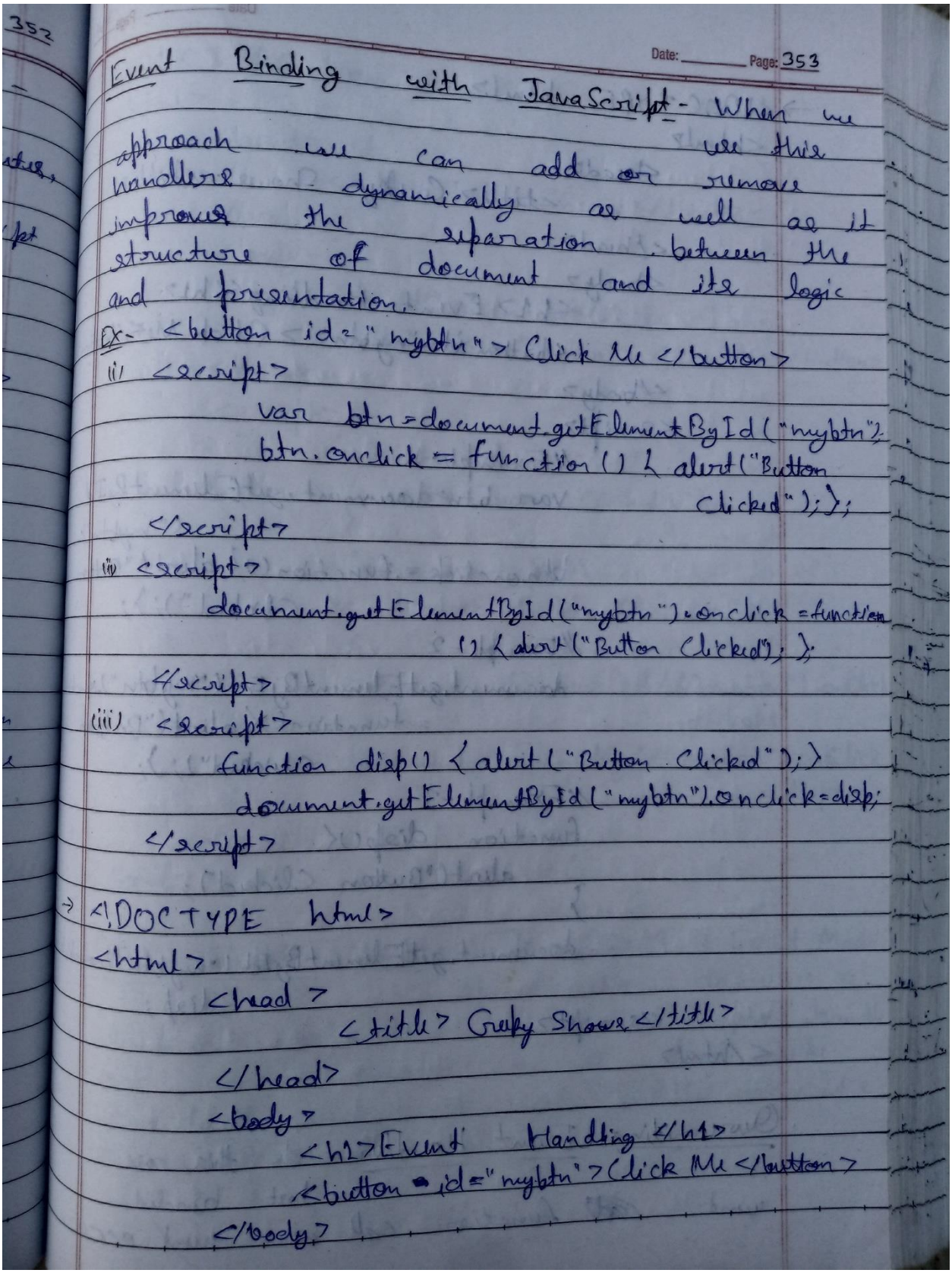
Date _____ Page 352

Event Binding with HTML Attributes -

These bindings are element attributes such as onclick and onchange, which can be set equal to JavaScript that is to be executed when the given event occurs at that object.

Ex- `<button onclick="alert('Button Clicked');">Click Me</button>`

```
<!DOCTYPE html>
<html>
    <head>
        <title> Geeky Shows </title>
    </head>
    <body>
        <h1> Event Handling </h1>
        <!-- <button onclick = " alert ('Button Clicked'); " >Click Me
                </button > -->
        <button onclick = "show(); "> Click Me </button>
    </body>
    <script>
        function show(){
            alert("Hello Event");
        }
    </script>
</html>
```



352

Date: Page: 353

Event Binding with JavaScript - When we use this approach we can add or remove handlers dynamically as well as it improves the separation between the structure of document and its logic and presentation.

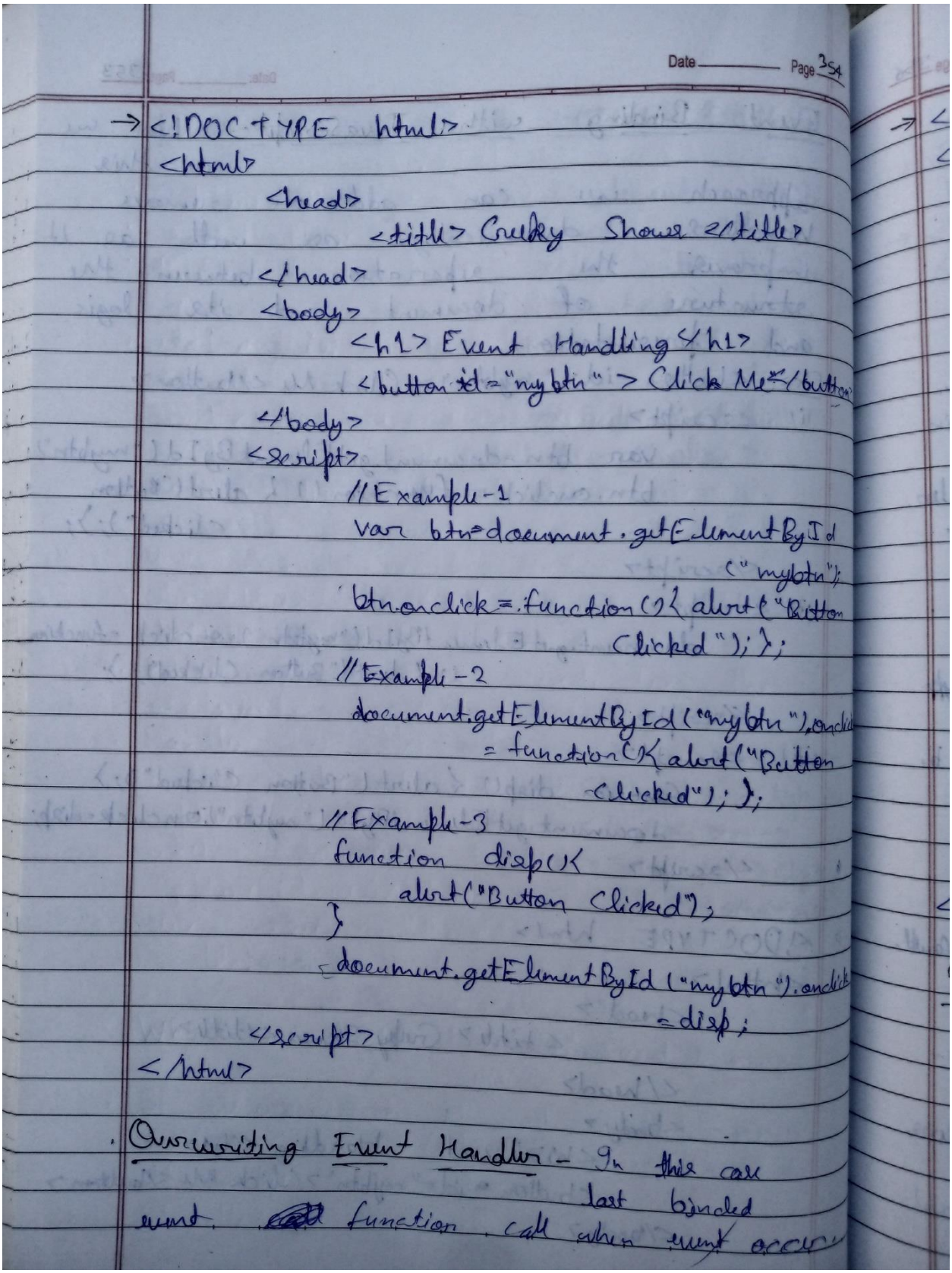
Ex- <button id="mybtn"> Click Me </button>

(i)
```
<script>
    var btn=document.getElementById("mybtn");
    btn.onclick = function () { alert("Button Clicked"); };
</script>
```

(ii)
```
<script>
    document.getElementById("mybtn").onclick=function
        () { alert("Button Clicked"); };
</script>
```

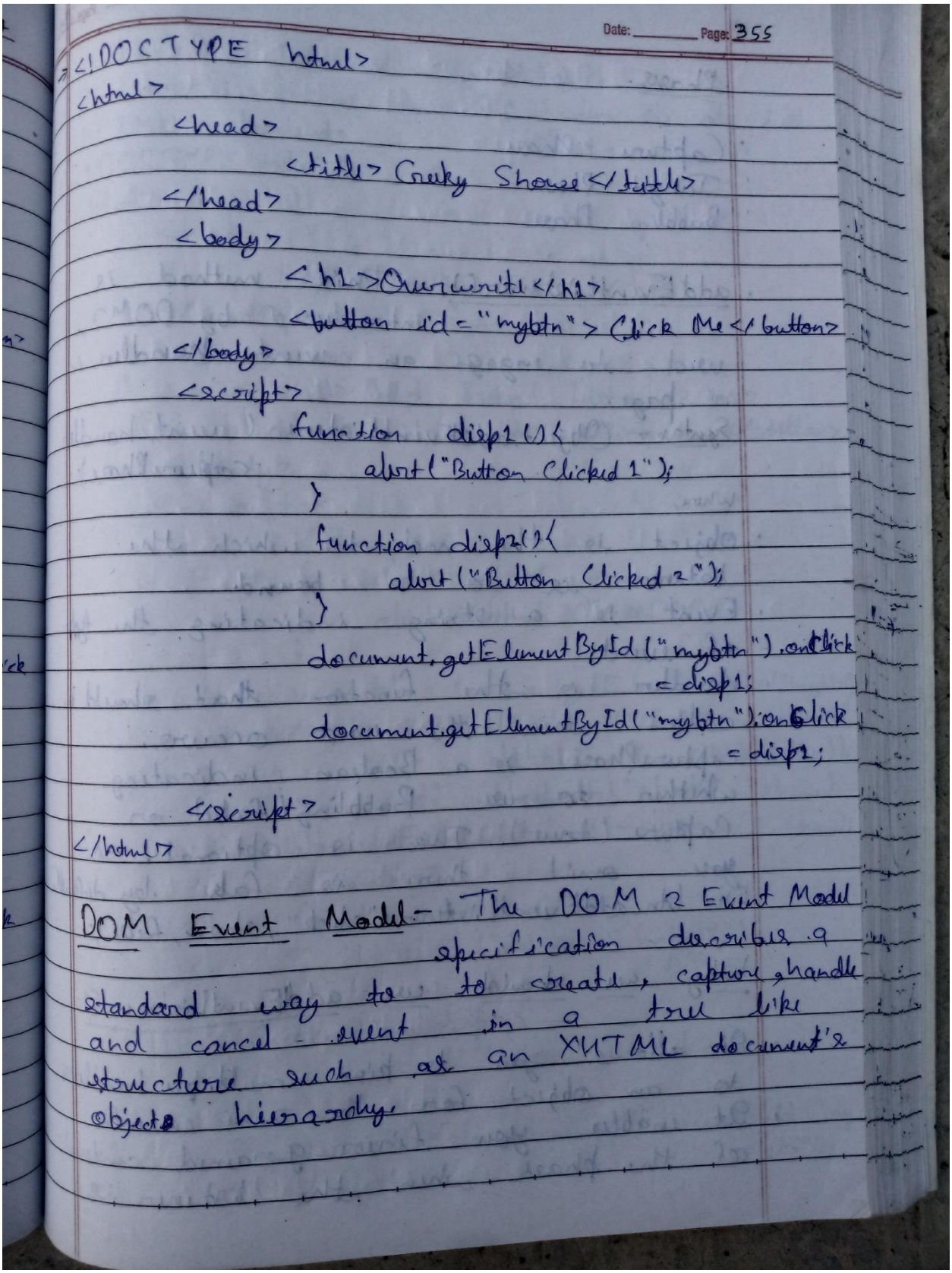
(iii)
```
<script>
    function disp() { alert("Button Clicked"); }
    document.getElementById("mybtn").onclick=disp;
</script>
```

```
<!DOCTYPE html>
<html>
    <head>
        <title> Geeky Shows </title>
    </head>
    <body>
        <h2> Event Handling </h2>
        <button id="mybtn"> Click Me </button>
    </body>
```



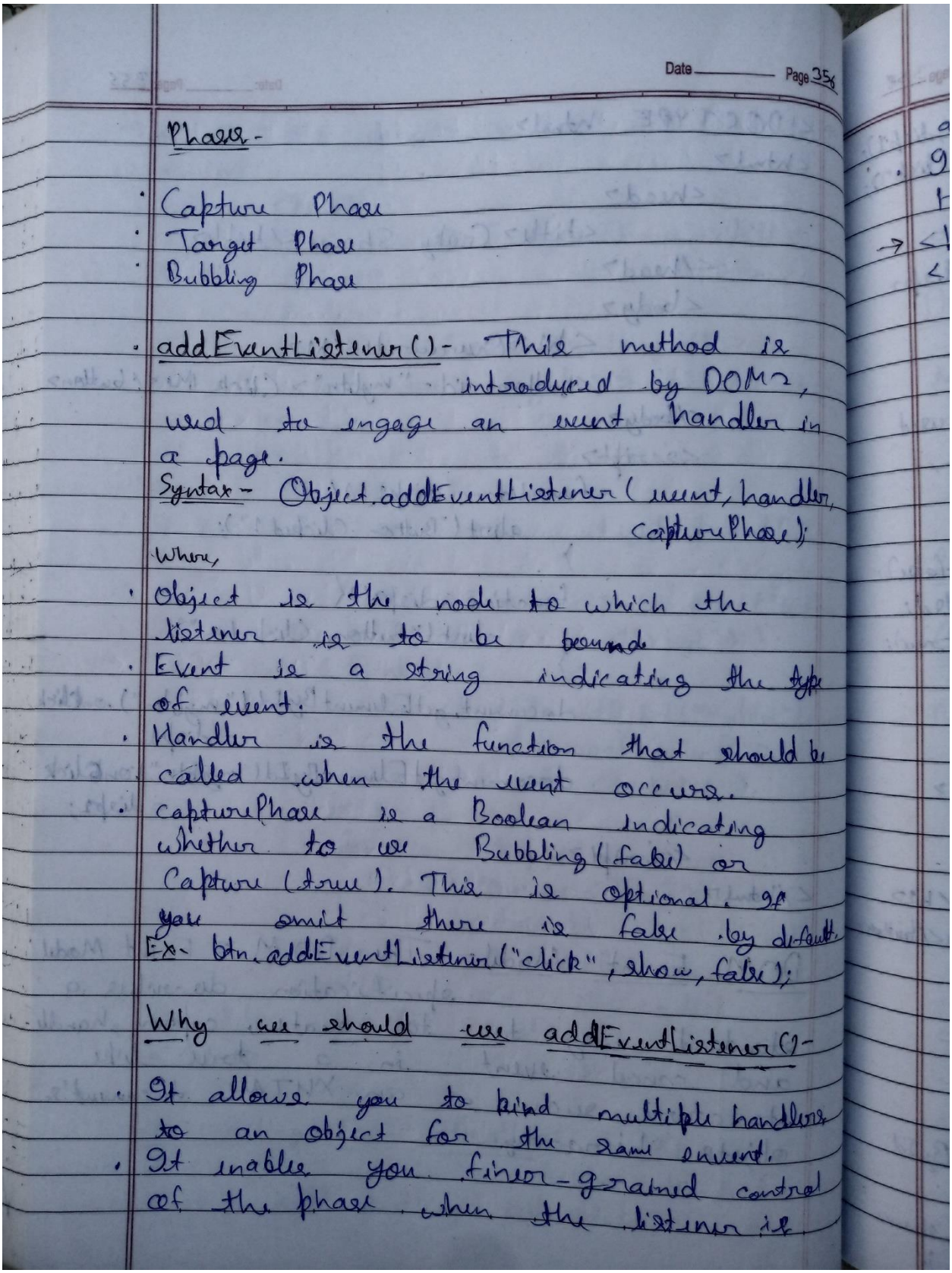
```
<!DOCTYPE html>
<html>
    <head>
        <title> Geeky Shows </title>
    </head>
    <body>
        <h1> Event Handling </h1>
        <button id="mybtn"> Click Me </button>
    </body>
    <script>
        //Example-1
        var btn=document.getElementById
                ("mybtn");
        btn.onclick=function(){alert("Button
                clicked");};
        //Example-2
        document.getElementById("mybtn").onclick
                =function(){alert("Button
                clicked");};
        //Example-3
        function disp(){
            alert("Button Clicked");
        }
        document.getElementById("mybtn").onclick
                =disp;
    </script>
</html>
```

Overwriting Event Handler - In this case last binded event function call when event occur.



```
<!DOCTYPE html>
<html>
    <head>
        <title> Geeky Shows </title>
    </head>
    <body>
        <h1>Our write</h1>
        <button id = "mybtn"> Click Me </button>
    </body>
    <script>
        function disp1(){
            alert("Button Clicked 1");
        }
        function disp2(){
            alert("Button Clicked 2");
        }
        document.getElementById("mybtn").onclick
                                    = disp1;
        document.getElementById("mybtn").onclick
                                    = disp2;

    </script>
</html>
```

DOM Event Modul- The DOM 2 Event Modul specification describes a standard way to to create, capture, handle and cancel event in a tree like structure such as an XHTML document's objects hierarchy.



Date _____ Page 356

Phase -

- Capture Phase
- Target Phase
- Bubbling Phase

- addEventListener() - This method is introduced by DOM2, used to engage an event handler in a page.

Syntax - Object.addEventListener(event, handler, capturePhase);

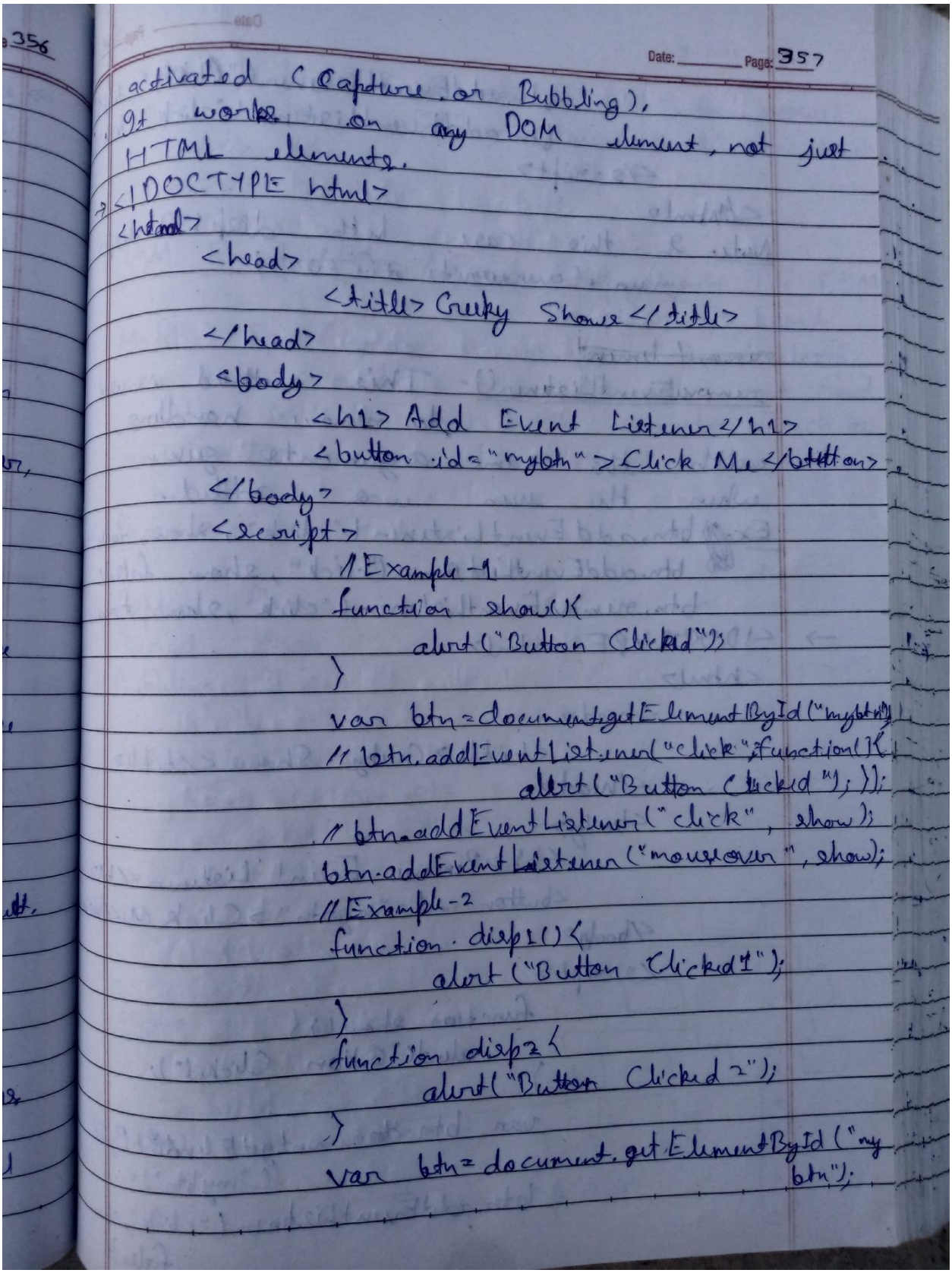
Where,

- Object is the node to which the listener is to be bound.
- Event is a string indicating the type of event.
- Handler is the function that should be called when the event occurs.
- capturePhase is a Boolean indicating whether to use Bubbling (false) or Capture (true). This is optional. If you omit there is false by default.

Ex - btn.addEventListener("click", show, false);

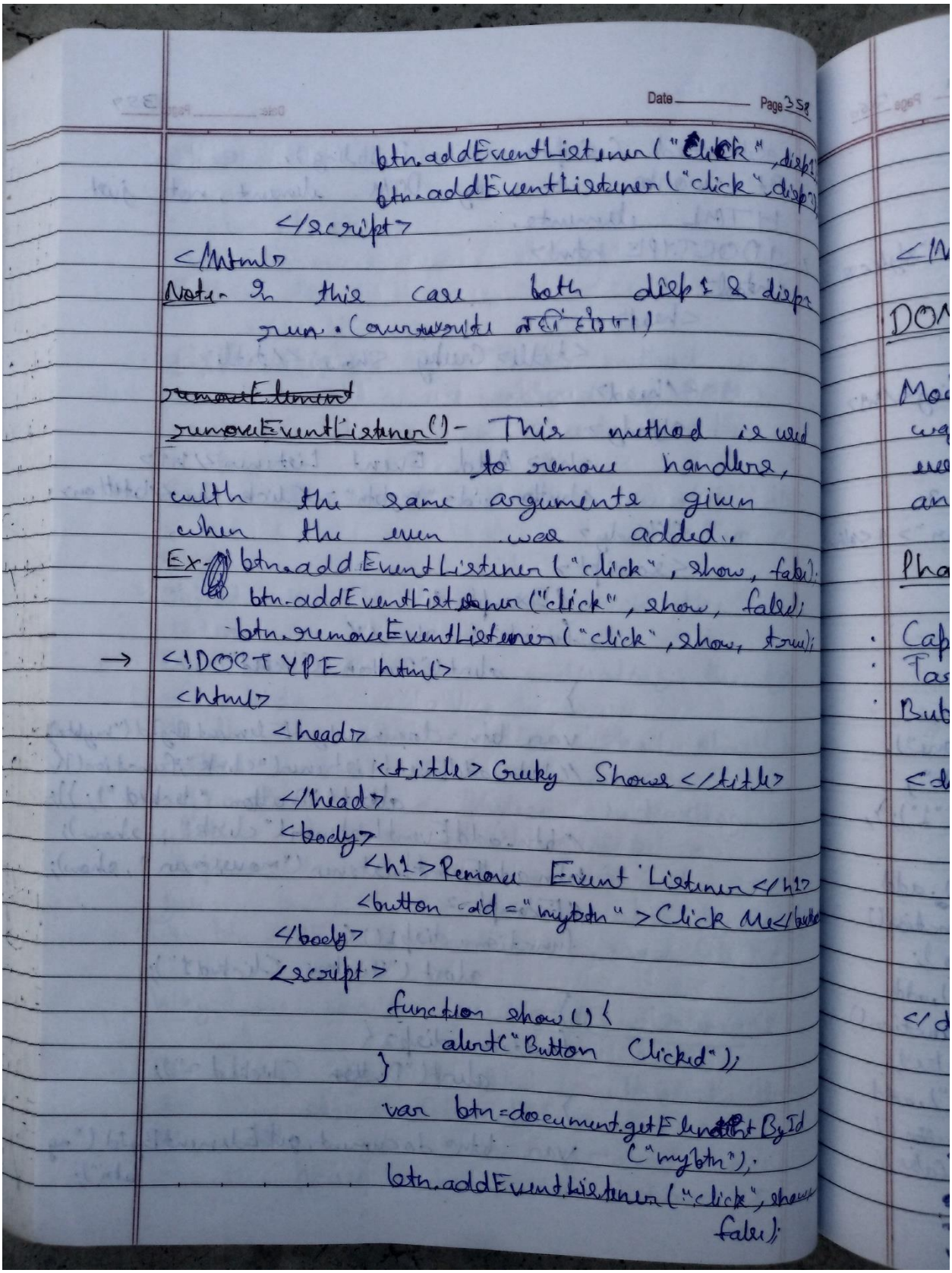
Why we should use addEventListener() -

- It allows you to bind multiple handlers to an object for the same event.
- It enables you finer-grained control of the phase when the listener is



activated (Capture or Bubbling).
It works on any DOM element, not just HTML elements.

```
<!DOCTYPE html>
<html>
    <head>
        <title> Geeky Shows </title>
    </head>
    <body>
        <h1> Add Event Listener </h1>
        <button id="mybtn"> Click Me </button>
    </body>
    <script>
        //Example-1
        function show(){
            alert("Button Clicked");
        }
        var btn=document.getElementById("mybtn");
        // btn.addEventListener("click",function(){
                alert("Button Clicked");});
        // btn.addEventListener("click", show);
        btn.addEventListener("mouseover", show);
        //Example-2
        function disp1(){
            alert("Button Clicked1");
        }
        function disp2{
            alert("Button Clicked 2");
        }
        var btn=document.getElementById("my
                btn");
```



```
        btn.addEventListener("click", disp1);
        btn.addEventListener("click", disp2);
    </script>
</html>
```

Note - In this case both disp1 & disp2 run. (overwrite नहीं होता)

removeEventListener() - This method is used to remove handlers, with the same arguments given when the even was added.

Ex- btn.addEventListener("click", show, false);
btn.addEventListener("click", show, false);
btn.removeEventListener("click", show, true);

```
<!DOCTYPE html>
<html>
    <head>
        <title> Geeky Shows </title>
    </head>
    <body>
        <h1> Remove Event Listener </h1>
        <button id="mybtn"> Click Me </button>
    </body>
    <script>
        function show(){
            alert("Button Clicked");
        }
        var btn=document.getElementById("mybtn");
        btn.addEventListener("click", show, false);
```

Date: 9-6-21 Page: 359

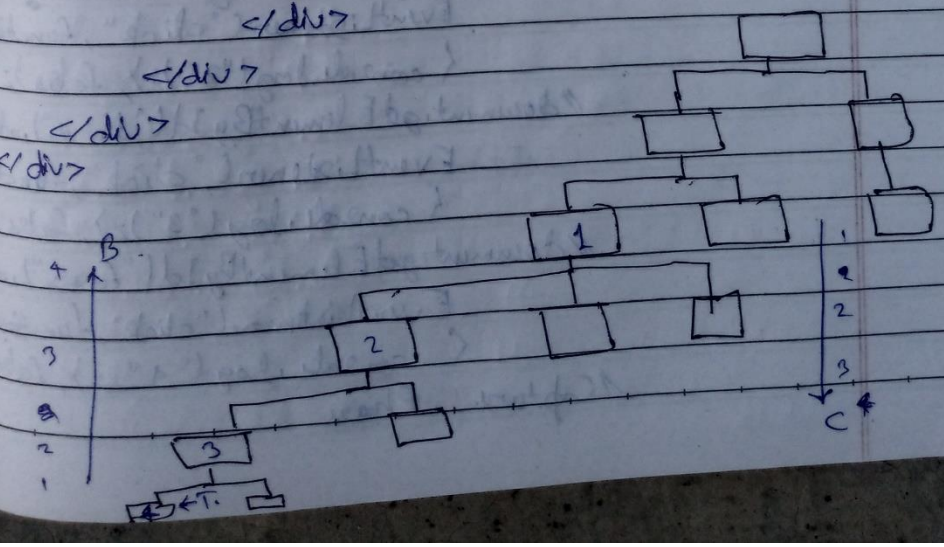
```
btn.addEventListener("click", show, true);
btn.removeEventListener("click", show, true);
</script>
</html>
```

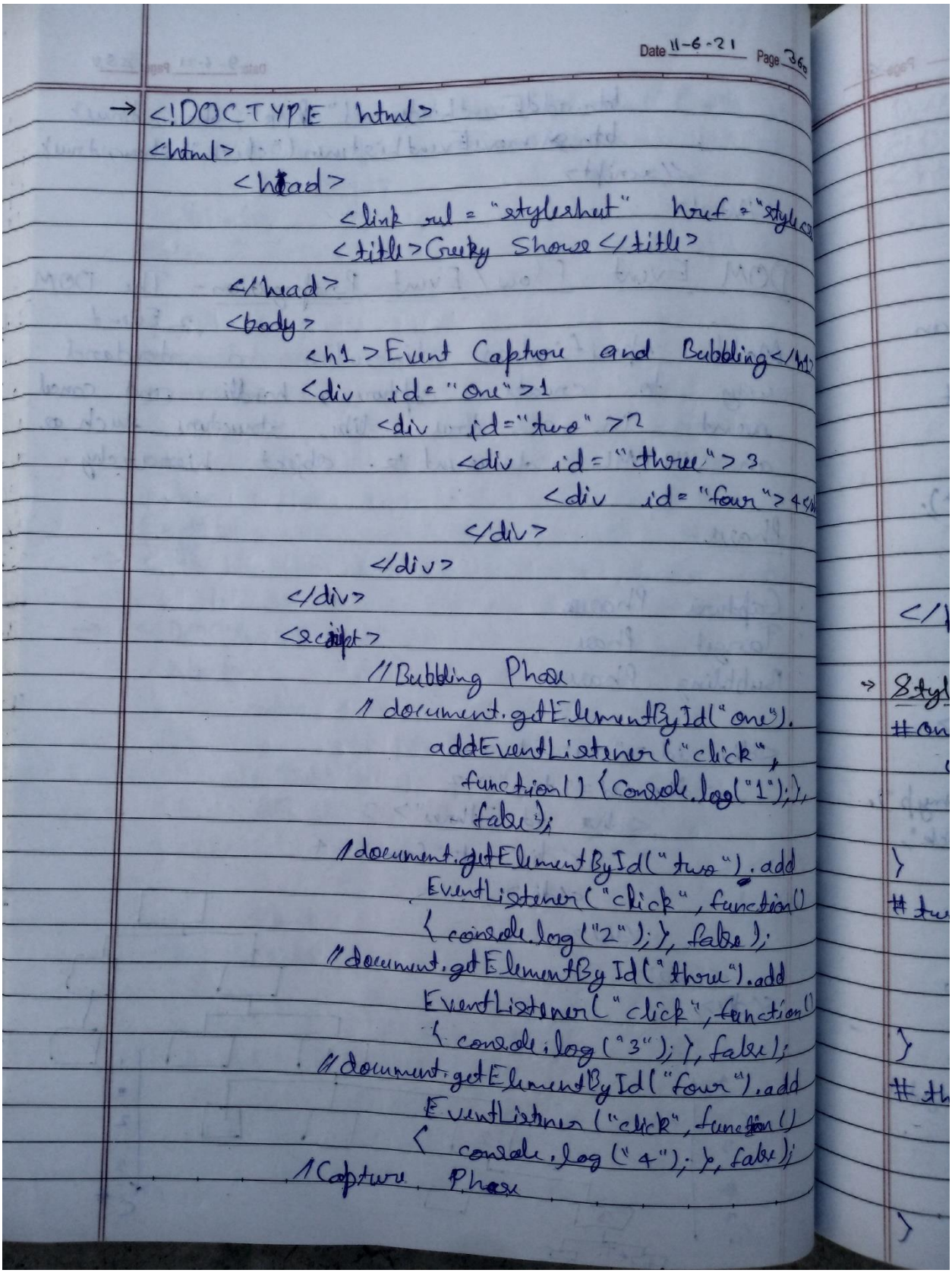
DOM Event Flow / Event Propagation - The DOM 2 Event Model specification describe a standard way to create, capture, handle and cancel event in a tree like structure such as an XHTML document's object hierarchy.

Phase -

- Capture Phase
- Target Phase
- Bubbling Phase

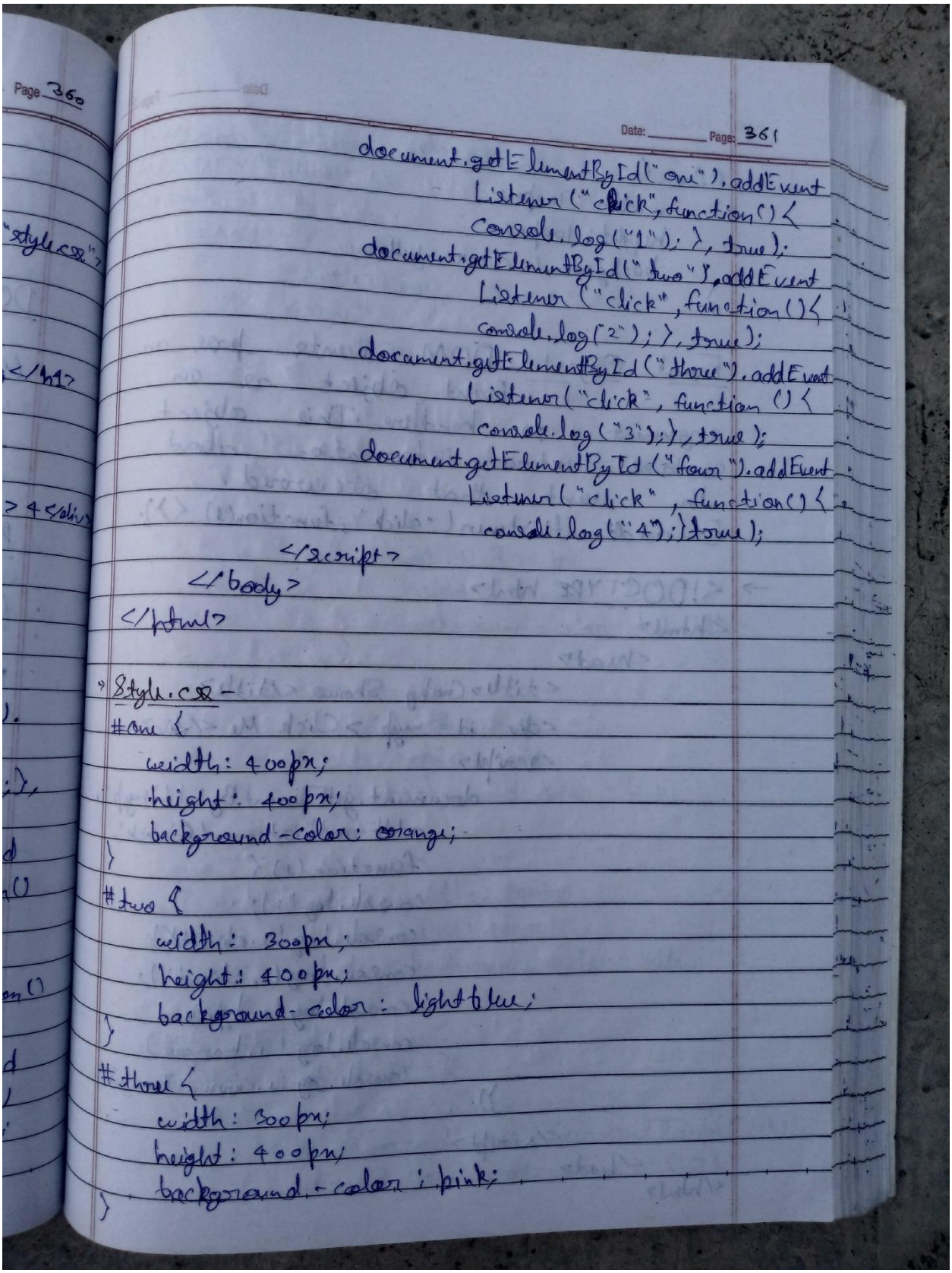
```
<div id="one">1
    <div id="two">2
        <div id="three">3
            <div id="four">4
            </div>
        </div>
    </div>
</div>
```





Date 11-6-21 Page 36

```
<!DOCTYPE html>
<html>
    <head>
        <link rel = "stylesheet" href = "style.cs
        <title>Geeky Shows</title>
    </head>
    <body>
        <h1>Event Capture and Bubbling</h1
        <div id = "one">1
            <div id="two">2
                <div id = "three">3
                    <div id = "four">4</di
                </div>
            </div>
        </div>
        <script>
            //Bubbling Phase
            // document.getElementById("one").
                addEventListener("click",
                function() {console.log("1");},
                false);
            //document.getElementById("two").add
                EventListener("click", function()
                {console.log("2");}, false);
            //document.getElementById("three").add
                EventListener("click", function()
                {console.log("3");}, false);
            //document.getElementById("four").add
                EventListener("click", function()
                {console.log("4");}, false);
            //Capture Phase
```



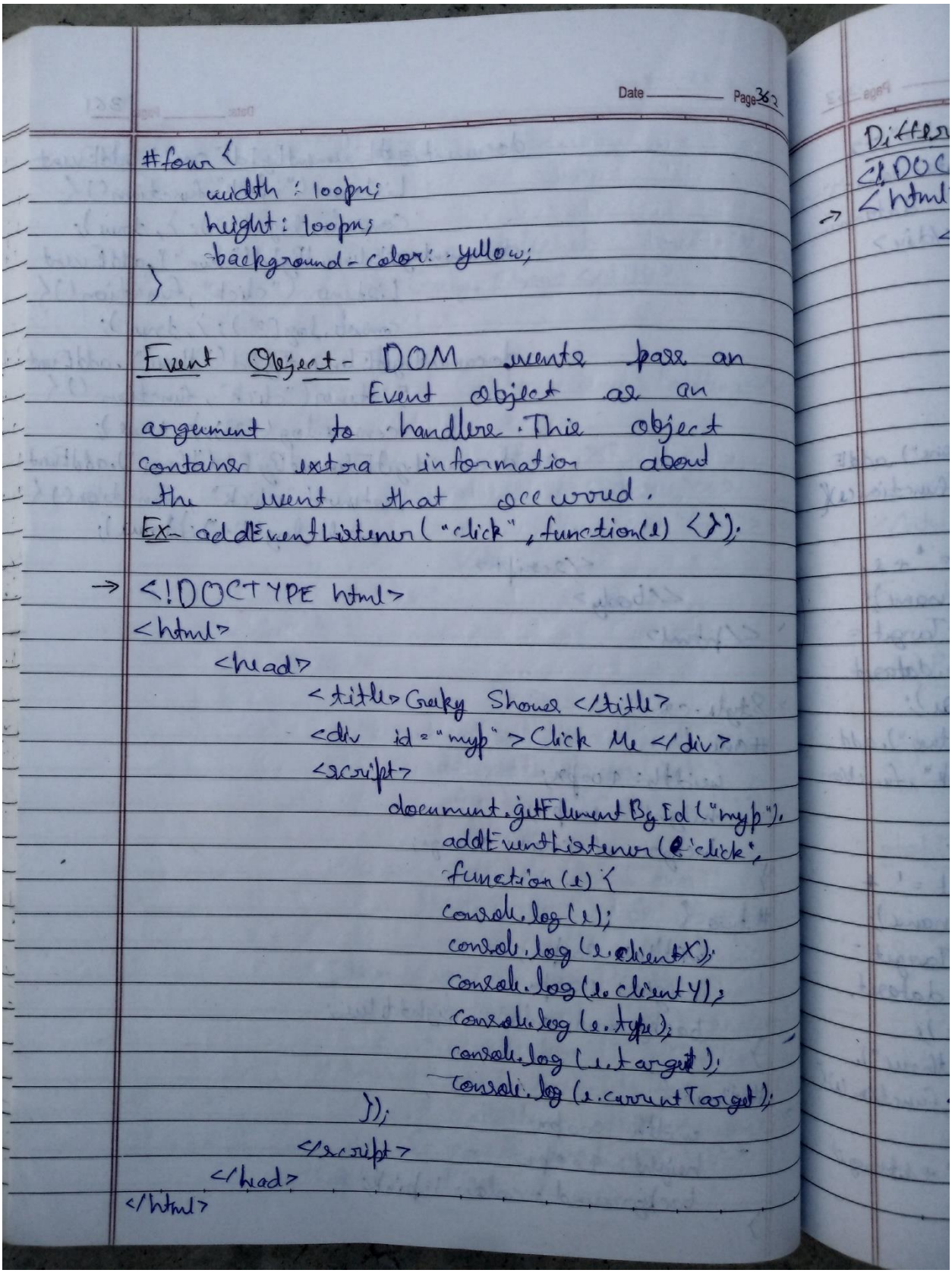
Page 360

Date: Page: 361

```
document.getElementById("one").addEvent
        Listener("click", function(){
        console.log("1"); }, true);
document.getElementById("two").addEvent
        Listener("click", function(){
        console.log("2"); }, true);
document.getElementById("three").addEvent
        Listener("click", function(){
        console.log("3"); }, true);
document.getElementById("four").addEvent
        Listener("click", function(){
        console.log("4"); }, true);
</script>
</body>
</html>
```

Style.css -

```
#one {
    width: 400px;
    height: 400px;
    background-color: orange;
}
#two {
    width: 300px;
    height: 400px;
    background-color: lightblue;
}
#three {
    width: 300px;
    height: 400px;
    background-color: pink;
}
```

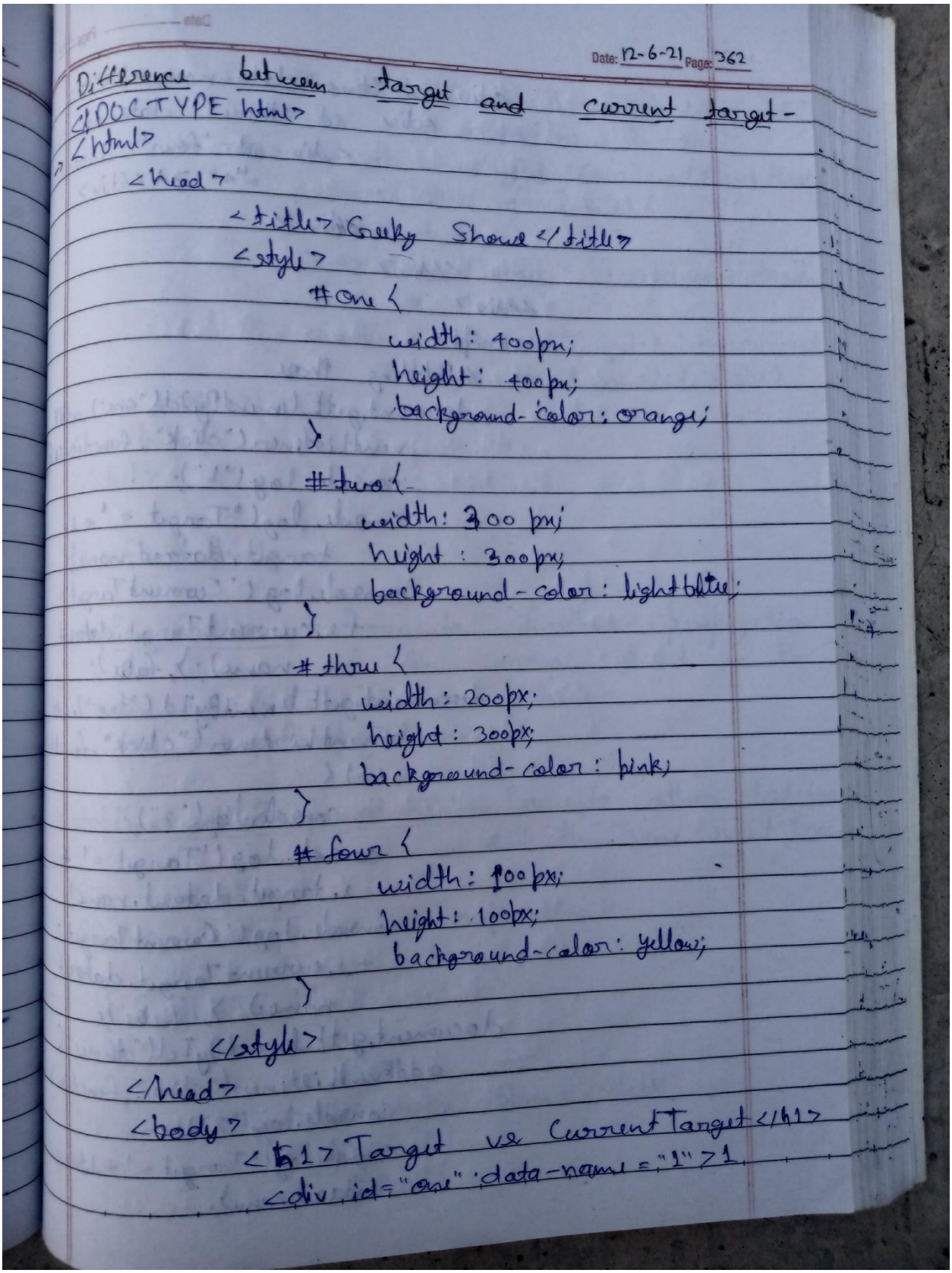


```
#four {
    width : 100px;
    height: 100px;
    background-color: yellow;
}
```

Event Object - DOM events pass an Event object as an argument to handlers. This object contains extra information about the event that occurred.

Ex- addEventListener("click", function(e) {});

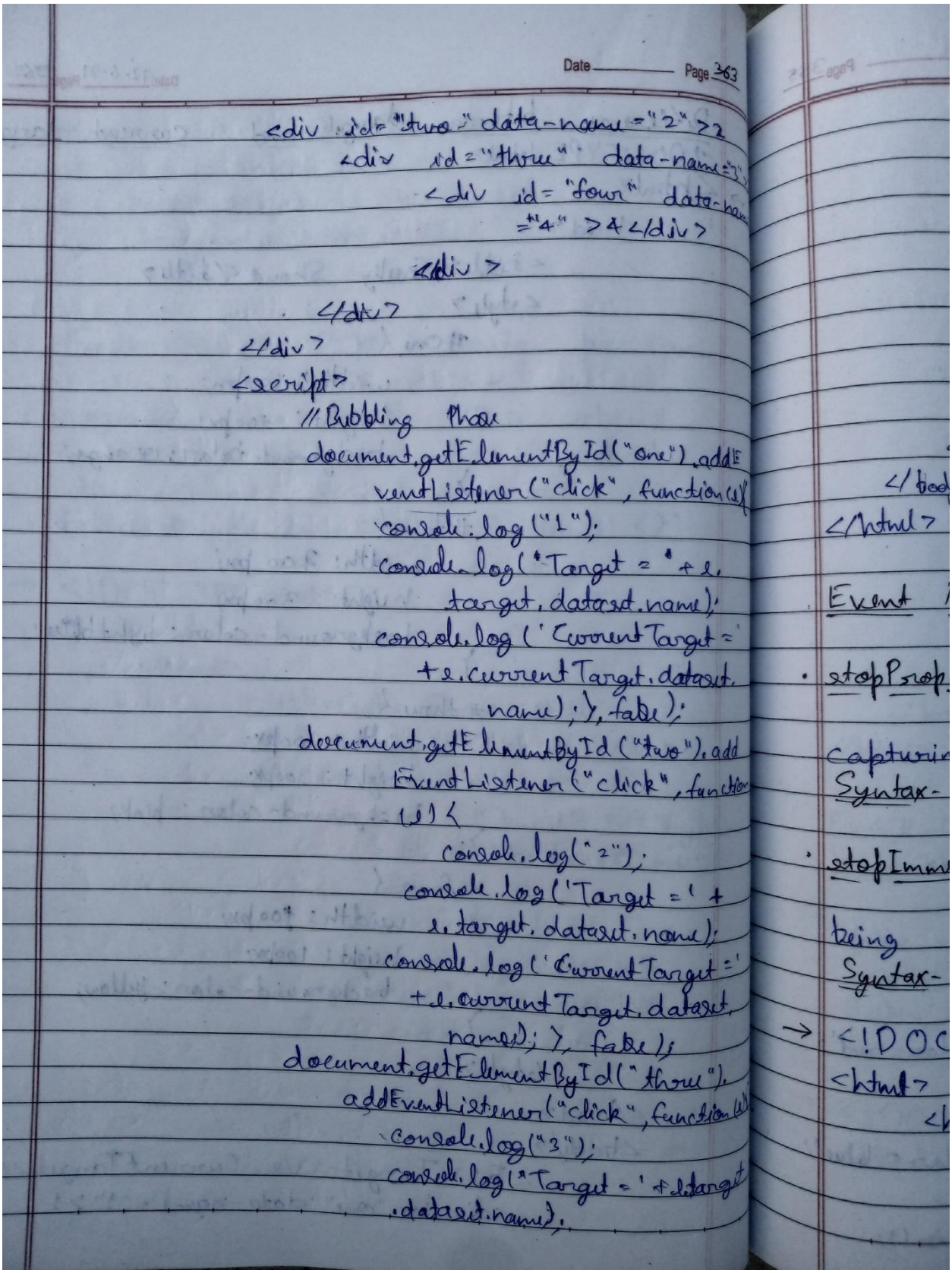
```
<!DOCTYPE html>
<html>
    <head>
        <title> Geeky Shows </title>
        <div id="myp"> Click Me </div>
        <script>
            document.getElementById("myp").
                addEventListener('click',
                function(e) {
                console.log(e);
                console.log(e.clientX);
                console.log(e.clientY);
                console.log(e.type);
                console.log(e.target);
                console.log(e.currentTarget);
            });
        </script>
    </head>
</html>
```



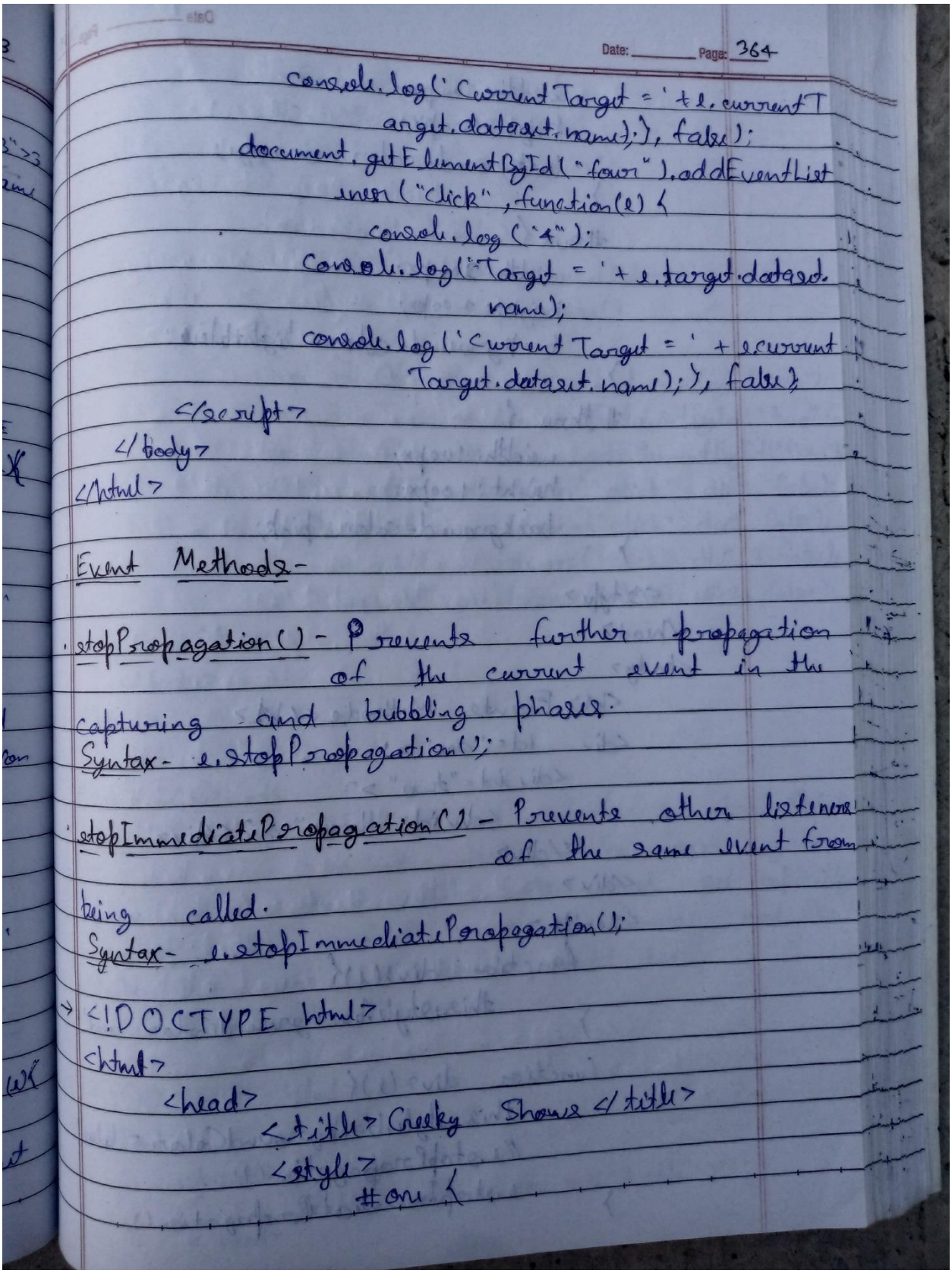
Date: 12-6-21 Page: 362

Difference between target and current target -

```
<!DOCTYPE html>
<html>
    <head>
        <title> Geeky Shows </title>
        <style>
            #one {
                width: 400px;
                height: 400px;
                background-color: orange;
            }
            #two {
                width: 300 px;
                height : 300px;
                background-color : light blue;
            }
            #three {
                width : 200px;
                height : 300px;
                background-color : pink;
            }
            #four {
                width : 100 px;
                height : 100px;
                background-color : yellow;
            }
        </style>
    </head>
    <body>
        <h1> Target vs Current Target </h1>
        <div id="one" data-name = "1" > 1.
```



```
<div id="two" data-name="2">2
    <div id="three" data-name="3">
        <div id="four" data-name="4">4</div>
    </div>
</div>
<script>
    // Bubbling Phase
    document.getElementById("one").addEventListener("click", function(e){
        console.log("1");
        console.log("Target = " + e.target.dataset.name);
        console.log("Current Target = " + e.currentTarget.dataset.name);
    }, false);
    document.getElementById("two").addEventListener("click", function(e){
        console.log("2");
        console.log("Target = " + e.target.dataset.name);
        console.log("Current Target = " + e.currentTarget.dataset.name);
    }, false);
    document.getElementById("three").addEventListener("click", function(e){
        console.log("3");
        console.log("Target = " + e.target.dataset.name);
```



```
Date: _____ Page: 364

        console.log('Current Target = ' + e.currentTarget.dataset.name);}, false);
    document.getElementById("four").addEventListener("click", function(e) {
        console.log("4");
        console.log("Target = " + e.target.dataset.name);
        console.log('Current Target = ' + e.currentTarget.dataset.name);}, false);
    </script>
</body>
</html>
```

Event Methods-

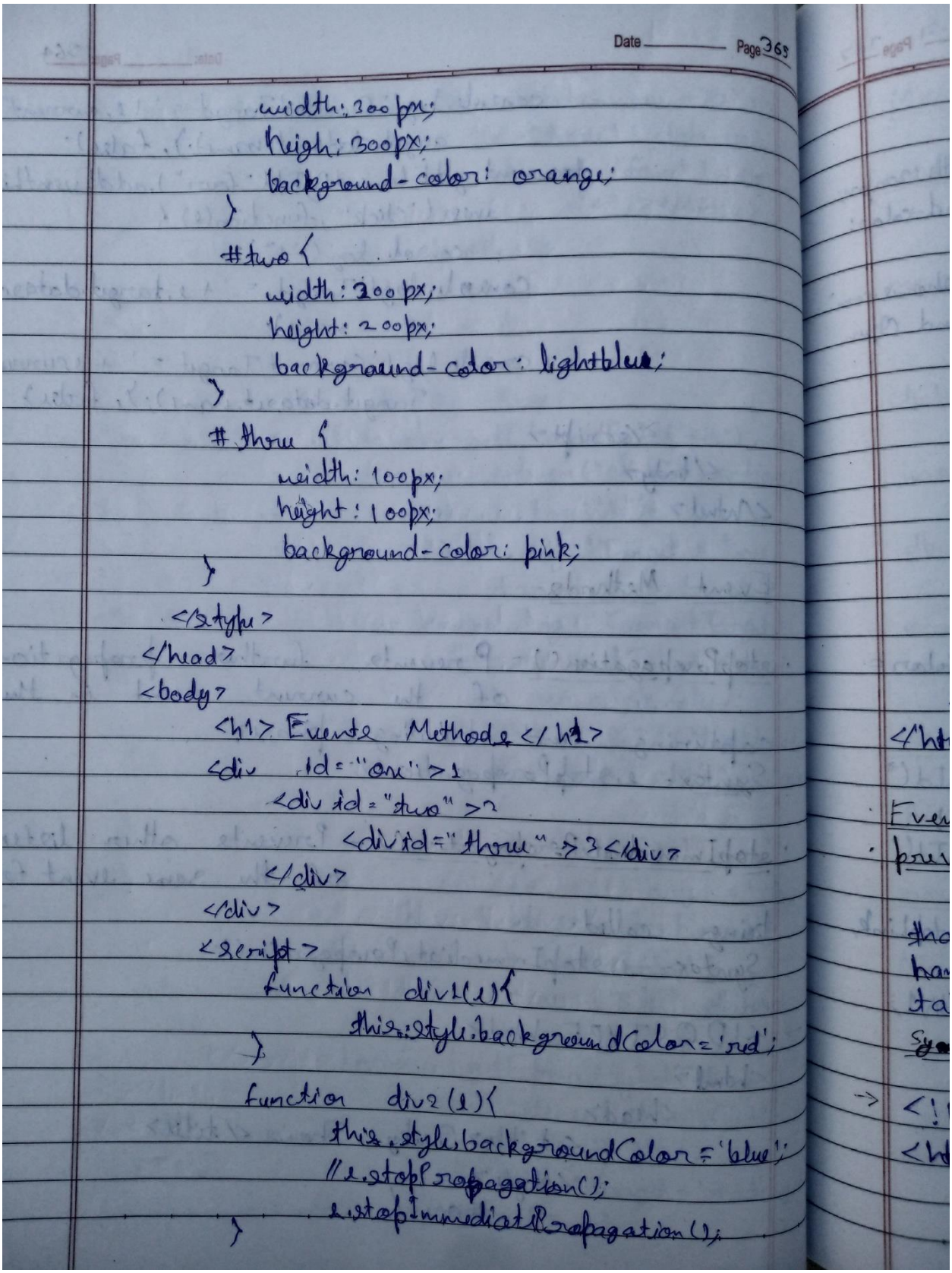
- stopPropagation() - Prevents further propagation of the current event in the capturing and bubbling phases.

Syntax- e.stopPropagation();

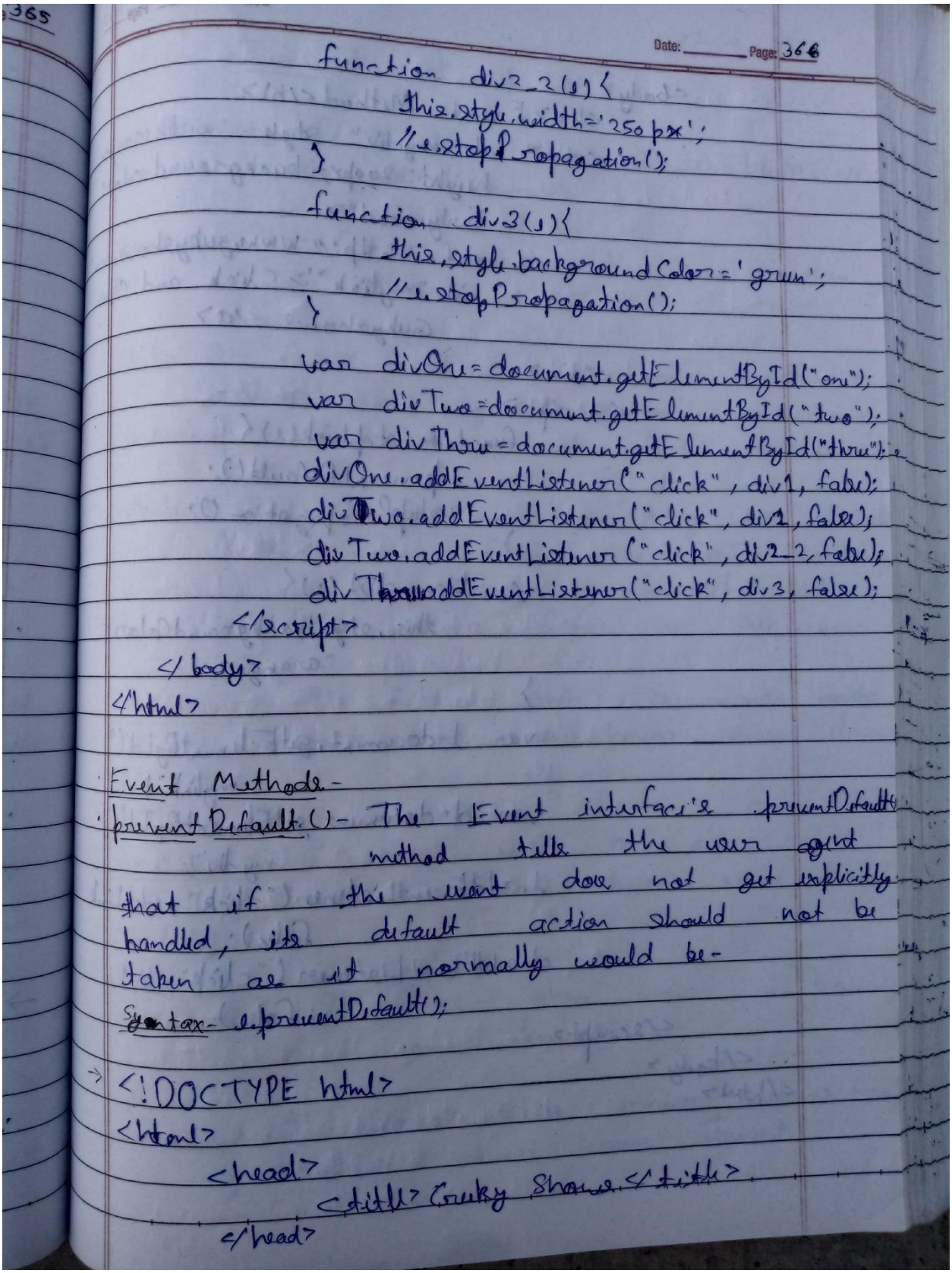
- stopImmediatePropagation() - Prevents other listeners of the same event from being called.

Syntax- e.stopImmediatePropagation();

```
<!DOCTYPE html>
<html>
    <head>
        <title> Geeky Shows </title>
        <style>
            #one {
```



```
        width: 300px;
        heigh: 300px;
        background-color: orange;
    }
    #two {
        width: 200px;
        height: 200px;
        background-color: lightblue;
    }
    #three {
        width: 100px;
        height: 100px;
        background-color: pink;
    }
</style>
</head>
<body>
    <h1> Events Methods </h1>
    <div id="one">1
        <div id="two">2
            <div id="three">3</div>
        </div>
    </div>
    <script>
        function div1(e){
            this.style.backgroundColor='red';
        }
        function div2(e){
            this.style.backgroundColor='blue';
            //e.stopPropagation();
            e.stopImmediatePropagation();
        }
```



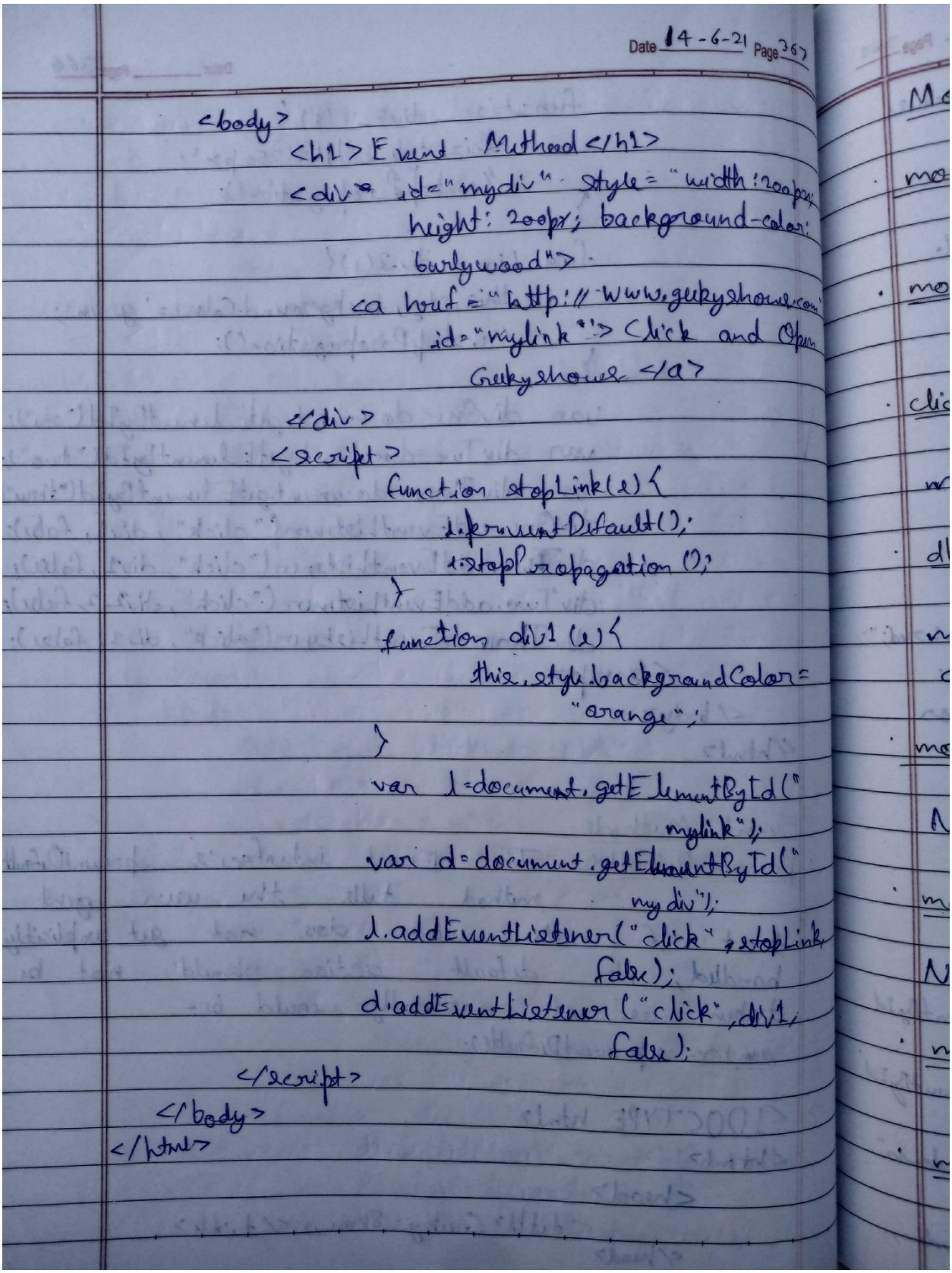
```
function div2_2(){
    this.style.width='250px';
    //e.stopPropagation();
}
function div3(){
    this.style.backgroundColor='green';
    //e.stopPropagation();
}
var divOne=document.getElementById("one");
var divTwo=document.getElementById("two");
var divThree=document.getElementById("three");
divOne.addEventListener("click",div1,false);
divTwo.addEventListener("click",div2,false);
divTwo.addEventListener("click",div2_2,false);
divThree.addEventListener("click",div3,false);
</script>
</body>
</html>
```

Event Methods -

preventDefault() - The Event interface's preventDefault() method tells the user agent that if the event does not get explicitly handled, its default action should not be taken as it normally would be -

Syntax - e.preventDefault();

```
<!DOCTYPE html>
<html>
    <head>
        <title> Geeky Shows </title>
    </head>
```



Date 14-6-21 Page 367

```
<body>
    <h1> Event Method </h1>
    <div id="mydiv" style="width:200px;
        height:200px; background-color:
        burlywood">
        <a href="http://www.geekyshows.com"
            id="mylink"> Click and Open
            Geekyshows </a>
    </div>
    <script>
        function stopLink(e){
            e.preventDefault();
            e.stopPropagation();
        }
        function div1(e){
            this.style.backgroundColor=
                "orange";
        }
        var l=document.getElementById("
                mylink");
        var d=document.getElementById("
                mydiv");
        l.addEventListener("click",stopLink,
                false);
        d.addEventListener("click",div1,
                false);
    </script>
</body>
</html>
```

Date: 17-6-21 Page: 368

Mouse Event -

mousedown - It fires when mouse button is pressed down.

mouseup - It fires when the mouse button is released.

click - It fires when ~~the mouse button is released.~~ something is clicked. mousedown, mouseup and click events fires in sequence.

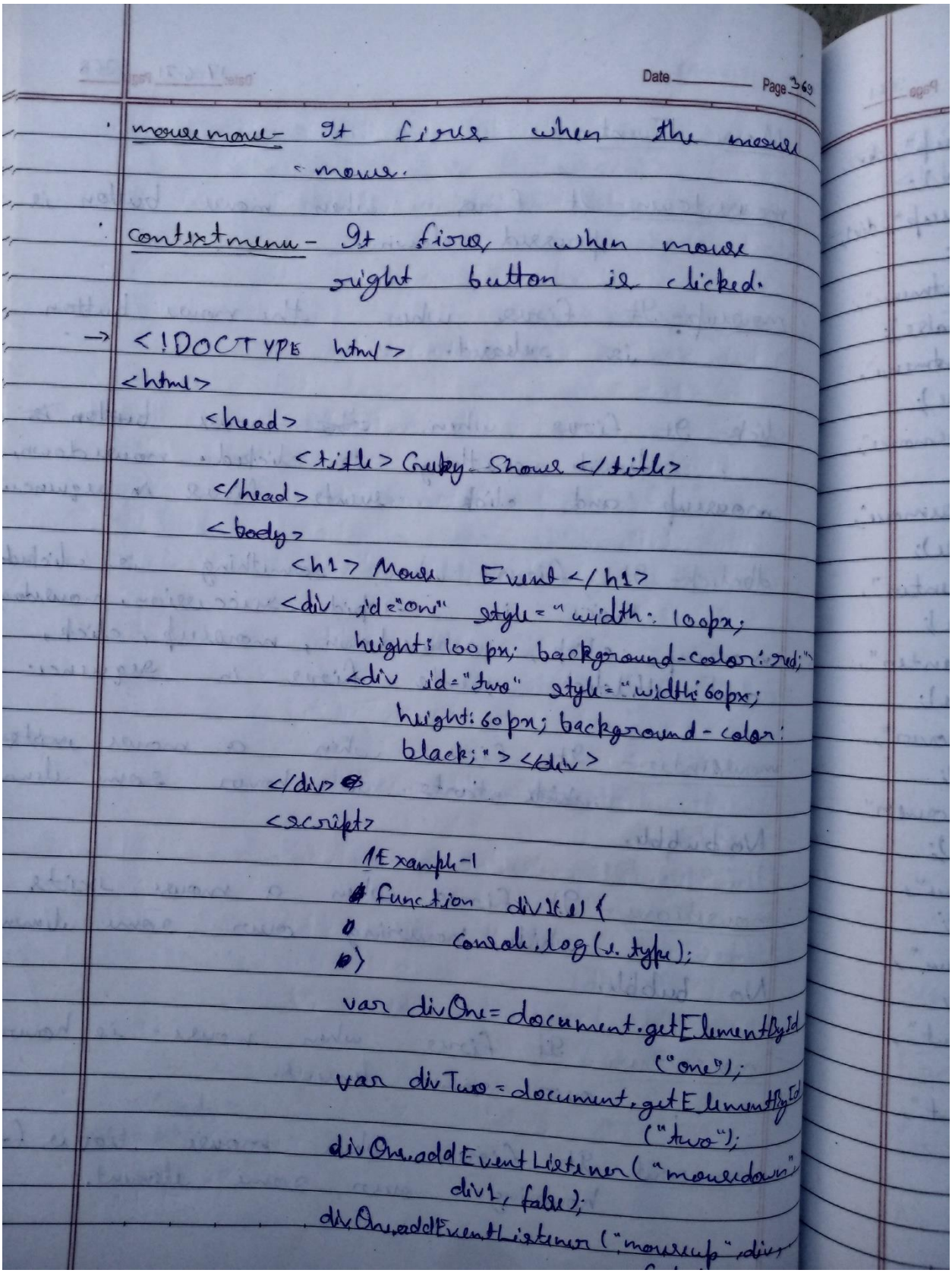
dbclick - It fires when something is clicked twice in rapid succession. mousedown, mouseup, click, mousedown, mouseup, click, and dbclick events fires in sequence.

mouseenter - It fires when a mouse ~~exits~~ starts to hover some element
No bubble.

mouseleave - It fires when a mouse exits while hovering over some element.
No bubble.

mouseover - It fires when mouse is hovering over some element.

mouseout - It fires when mouse leaves from hovering over some element.

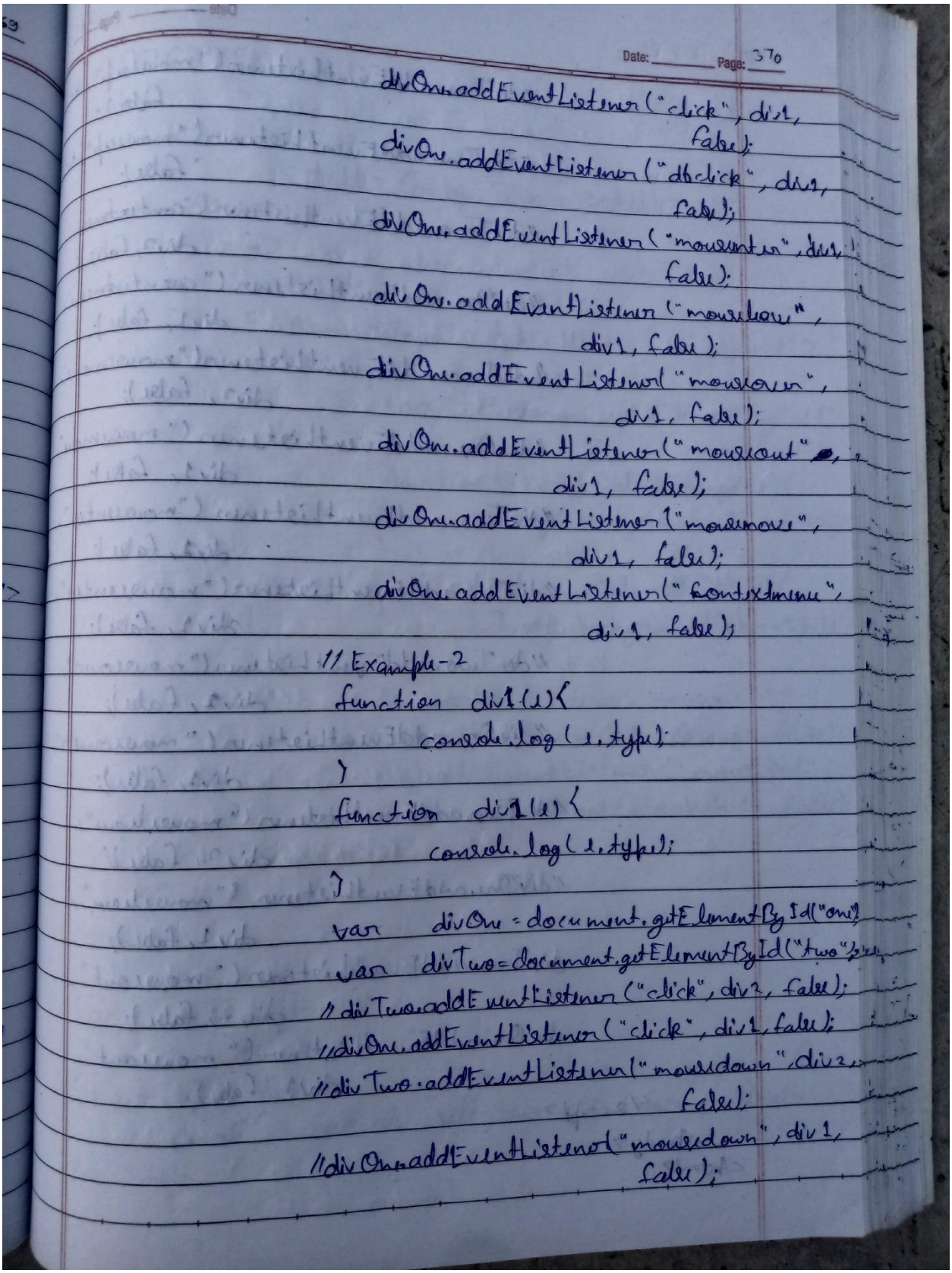


Date _____ Page 369

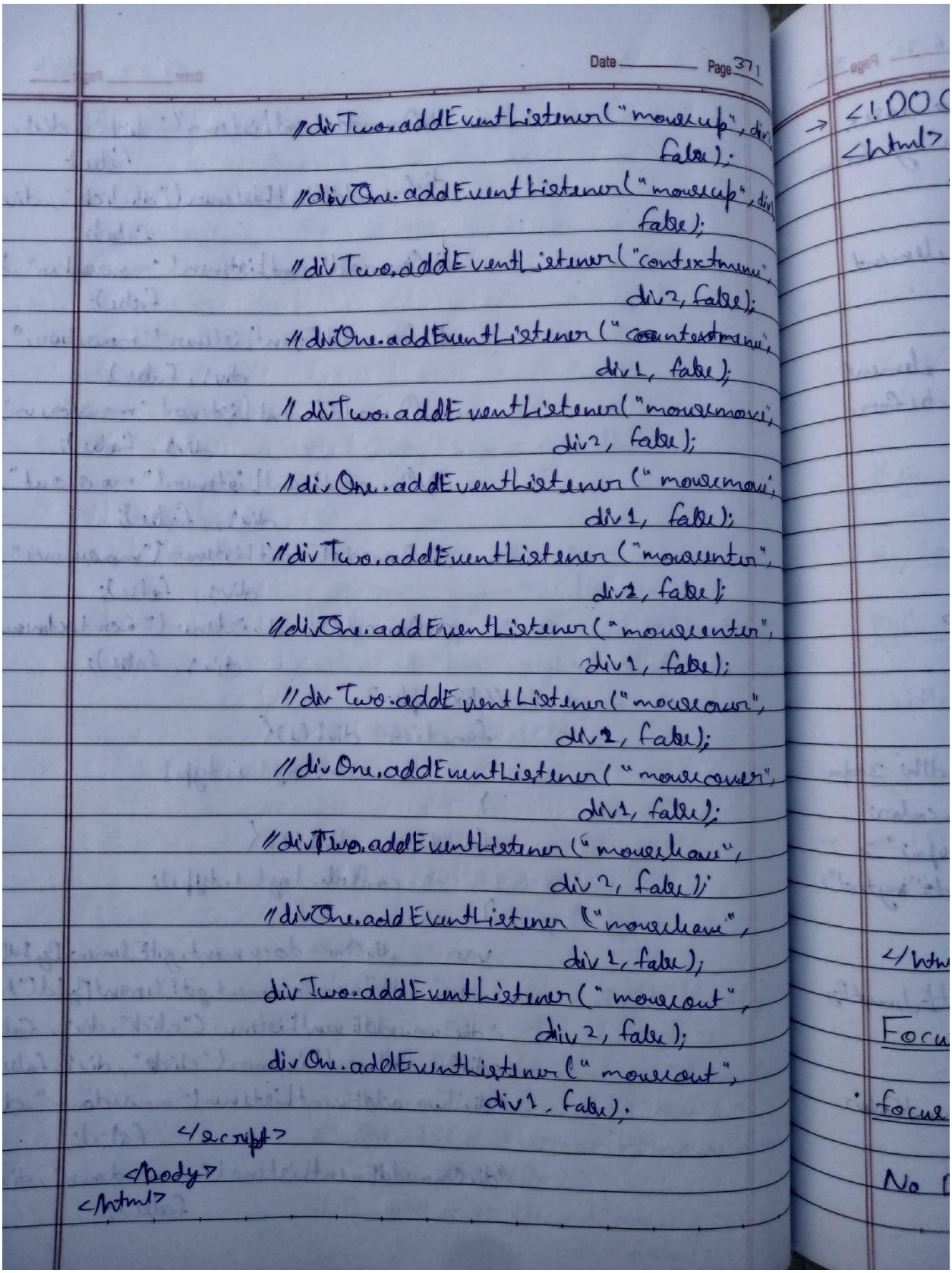
mousemove - It fires when the mouse move.

contextmenu - It fires when mouse right button is clicked.

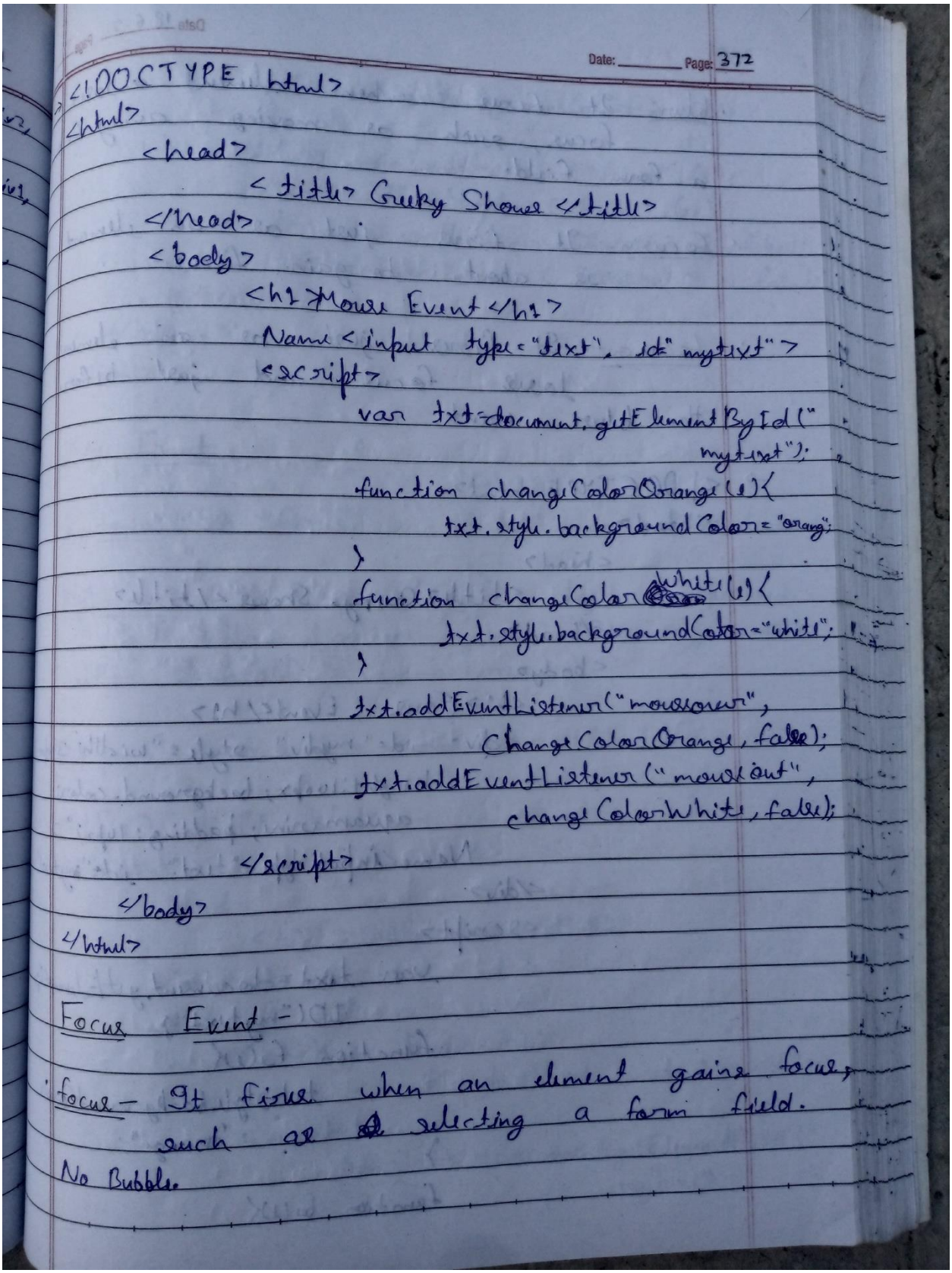
→
```
<!DOCTYPE html>
<html>
    <head>
        <title> Geeky Shows </title>
    </head>
    <body>
        <h1> Mouse Event </h1>
        <div id="one" style="width: 100px;
            height: 100px; background-color: red;">
            <div id="two" style="width: 60px;
                height: 60px; background-color:
                black;"> </div>
        </div>
        <script>
            // Example-1
            function div1(e) {
                console.log(e.type);
            }
            var divOne = document.getElementById
                ("one");
            var divTwo = document.getElementById
                ("two");
            divOne.addEventListener("mousedown",
                div1, false);
            divOne.addEventListener("mouseup", div
```



```
Date: Page: 370

divOne.addEventListener("click", div1,
                        false);
divOne.addEventListener("dbclick", div1,
                        false);
divOne.addEventListener("mouseenter", div1,
                        false);
divOne.addEventListener("mouseleave",
                        div1, false);
divOne.addEventListener("mouseover",
                        div1, false);
divOne.addEventListener("mouseout",
                        div1, false);
divOne.addEventListener("mousemove",
                        div1, false);
divOne.addEventListener("contextmenu",
                        div1, false);

// Example-2
function div1(e){
    console.log(e.type);
}
function div2(e){
    console.log(e.type);
}
var divOne = document.getElementById("one");
var divTwo = document.getElementById("two");
// divTwo.addEventListener("click", div2, false);
//divOne.addEventListener("click", div1, false);
//divTwo.addEventListener("mousedown", div2,
                        false);
//divOne.addEventListener("mousedown", div1,
                        false);
```



```
//divTwo.addEventListener("mouseup", div2,
                                false);
//divOne.addEventListener("mouseup", div1,
                                false);
//divTwo.addEventListener("contextmenu",
                                div2, false);
//divOne.addEventListener("contextmenu",
                                div1, false);
//divTwo.addEventListener("mousemove",
                                div2, false);
//divOne.addEventListener("mousemove",
                                div1, false);
//divTwo.addEventListener("mouseenter",
                                div2, false);
//divOne.addEventListener("mouseenter",
                                div1, false);
//divTwo.addEventListener("mouseover",
                                div2, false);
//divOne.addEventListener("mouseover",
                                div1, false);
//divTwo.addEventListener("mouseleave",
                                div2, false);
//divOne.addEventListener("mouseleave",
                                div1, false);
divTwo.addEventListener("mouseout",
                                div2, false);
divOne.addEventListener("mouseout",
                                div1, false);
</script>
</body>
</html>
```

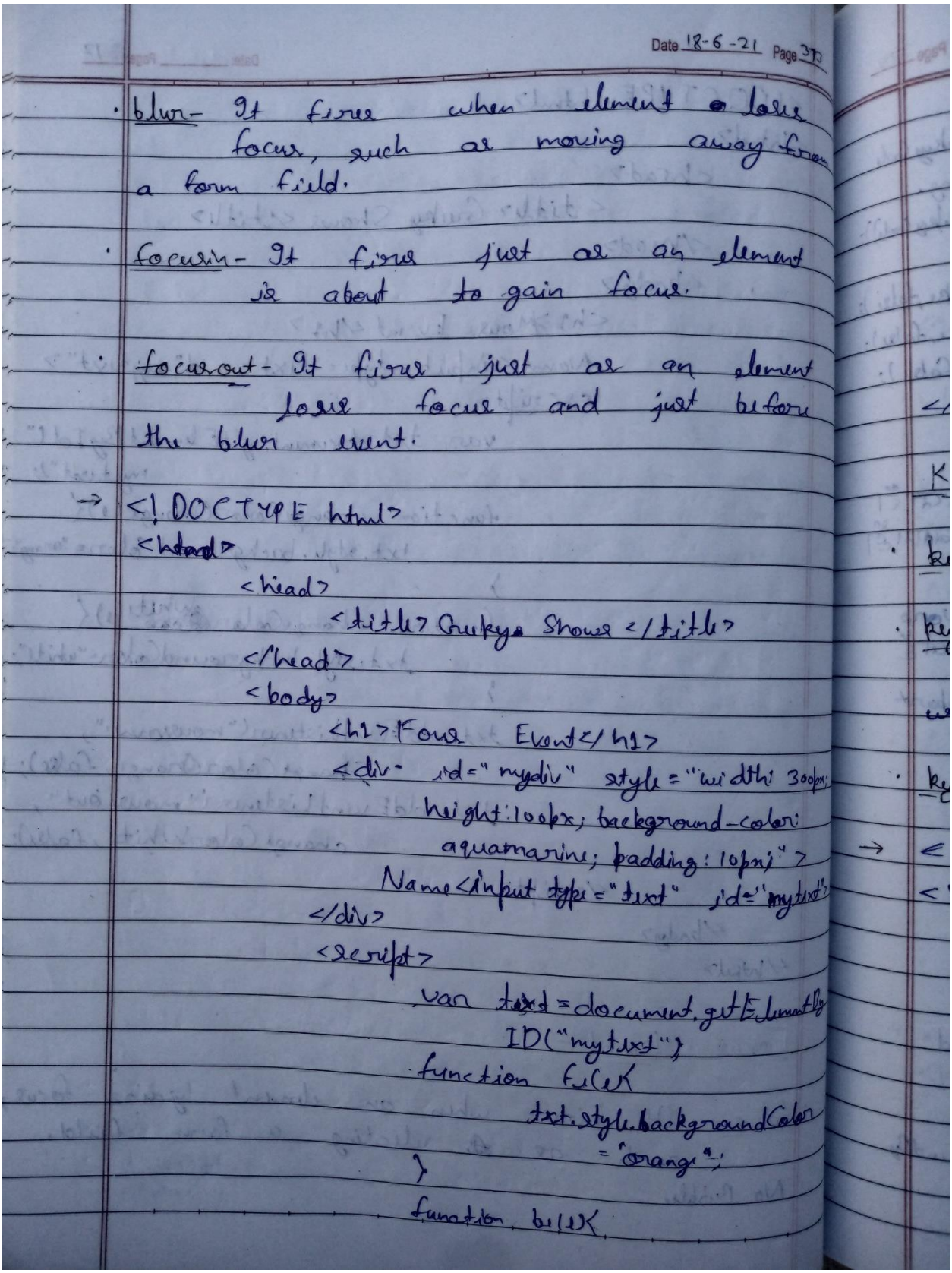


```
<!DOCTYPE html>
<html>
    <head>
        <title> Geeky Shows </title>
    </head>
    <body>
        <h1> Mouse Event </h1>
        Name <input type="text", id="mytext" >
        <script>
            var txt=document.getElementById("
                                    mytext");
            function changeColorOrange(e){
                txt.style.backgroundColor="orang";
            }
            function changeColorWhite(e){
                txt.style.backgroundColor="white";
            }
            txt.addEventListener("mouseover",
                        ChangeColorOrange, false);
            txt.addEventListener("mouseout",
                        changeColorWhite, false);
        </script>
    </body>
</html>
```

Focus Event -

focus - It fires when an element gains focus, such as selecting a form field.

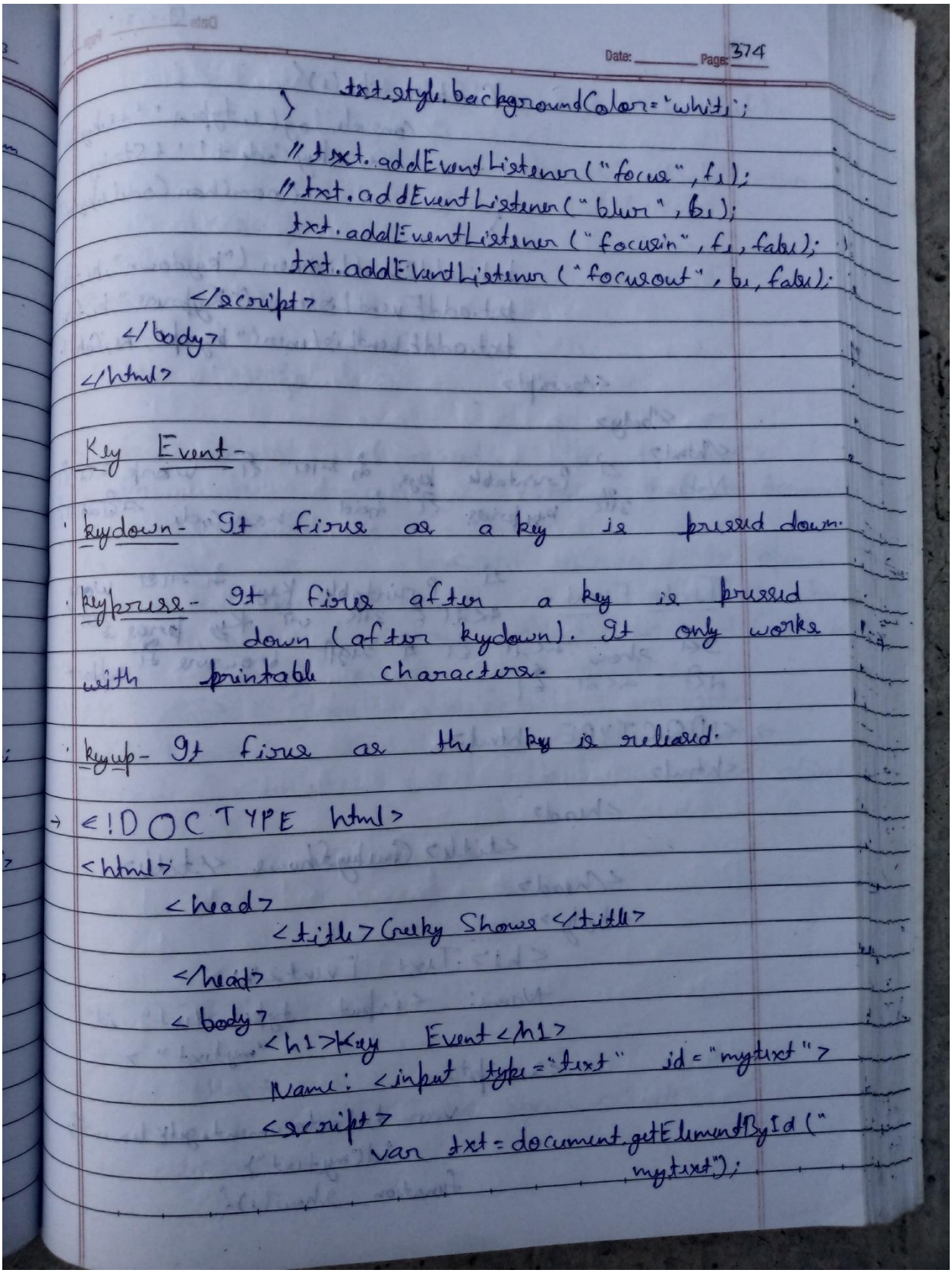
No Bubble.



Date 18-6-21 Page 375

- blur - It fires when element o loses focus, such as moving away from a form field.
- focusin - It fires just as an element is about to gain focus.
- focusout - It fires just as an element loses focus and just before the blur event.

→
```
<!DOCTYPE html>
<html>
    <head>
        <title> Geekyshows </title>
    </head>
    <body>
        <h1> Focus Event </h1>
        <div id="mydiv" style="width: 300px;
            height: 100px; background-color:
            aquamarine; padding: 10px;">
            Name <input type="text" id="mytxt">
        </div>
        <script>
            var txt = document.getElementById("mytxt")
            function fx(){
                txt.style.backgroundColor
                = "orange";
            }
            function bl(){
```



Date: Page: 374

```
    txt.style.backgroundColor="white";
}
// txt.addEventListener("focus", fx);
// txt.addEventListener("blur", bx);
txt.addEventListener("focusin", fx, false);
txt.addEventListener("focusout", bx, false);
</script>
</body>
</html>
```

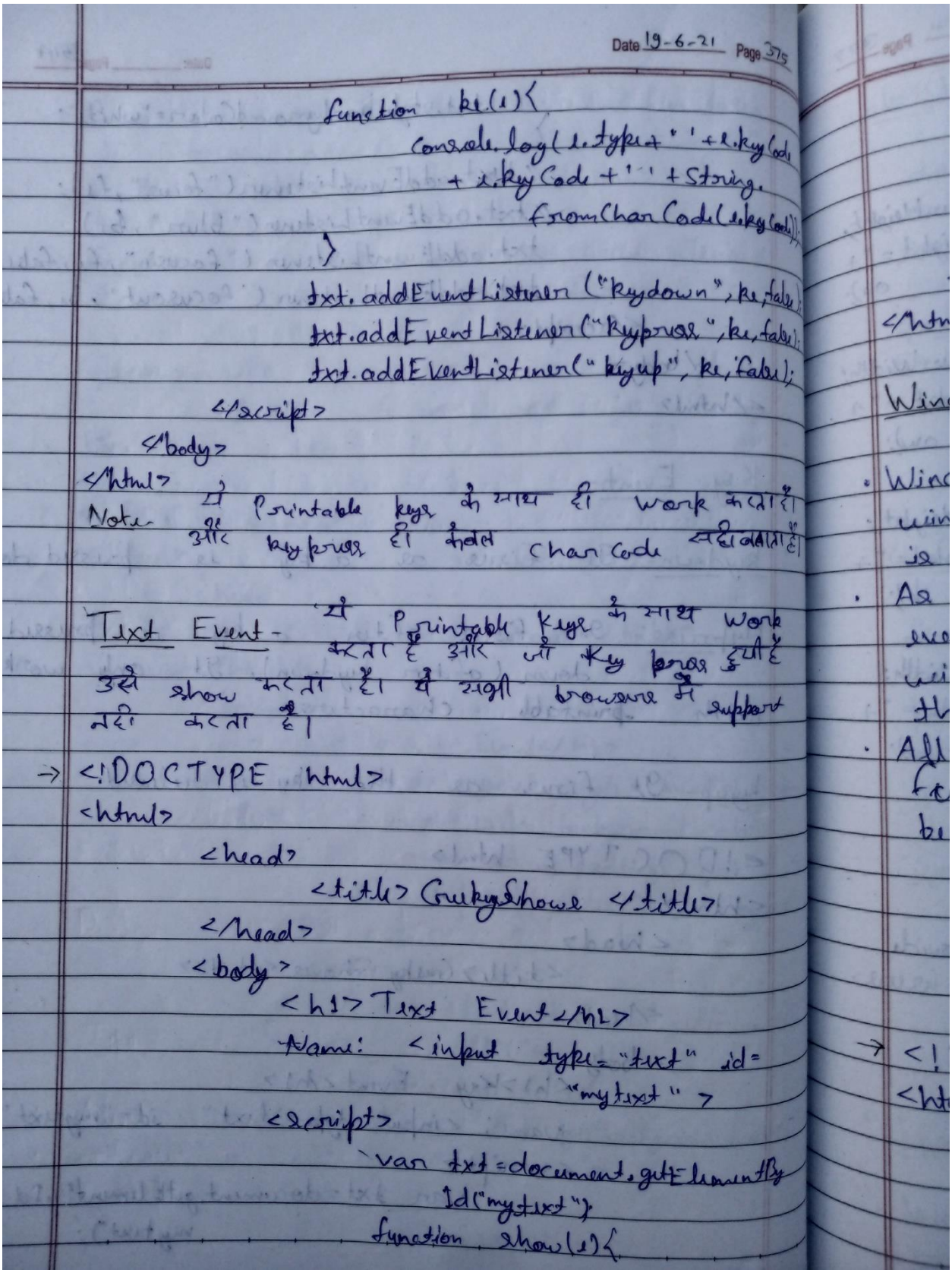
Key Event-

· keydown- It fires as a key is pressed down.

· keypress- It fires after a key is pressed down (after keydown). It only works with printable characters.

· keyup- It fires as the key is released.

```
<!DOCTYPE html>
<html>
    <head>
        <title>Geeky Shows</title>
    </head>
    <body>
        <h1>Key Event</h1>
        Name: <input type="text" id="mytext">
        <script>
            var txt=document.getElementById("
                mytext");
```



Date 19-6-21 Page 375

```
function ke(e){
    console.log(e.type+' '+e.keyCode
    +e.keyCode+' '+String.
    fromCharCode(e.keyCode));
}
txt.addEventListener("keydown",ke,false);
txt.addEventListener("keypress",ke,false);
txt.addEventListener("keyup",ke,false);
</script>
</body>
</html>
```

Note- यह Printable keys के साथ ही work करता है और keypress ही केवल char code दिखाता है।

Text Event- यह Printable keys के साथ work करता है और जो key press हुई है उसे show करता है। यह सभी browsers में support नहीं करता है।

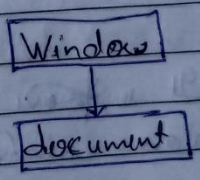
```
<!DOCTYPE html>
<html>
    <head>
        <title> GeekyShows </title>
    </head>
    <body>
        <h1> Text Event </h1>
        Name: <input type="text" id=
            "mytext" >
        <script>
            var txt=document.getElementBy
            Id("mytext");
            function show(e){
```

Date: Page: 376

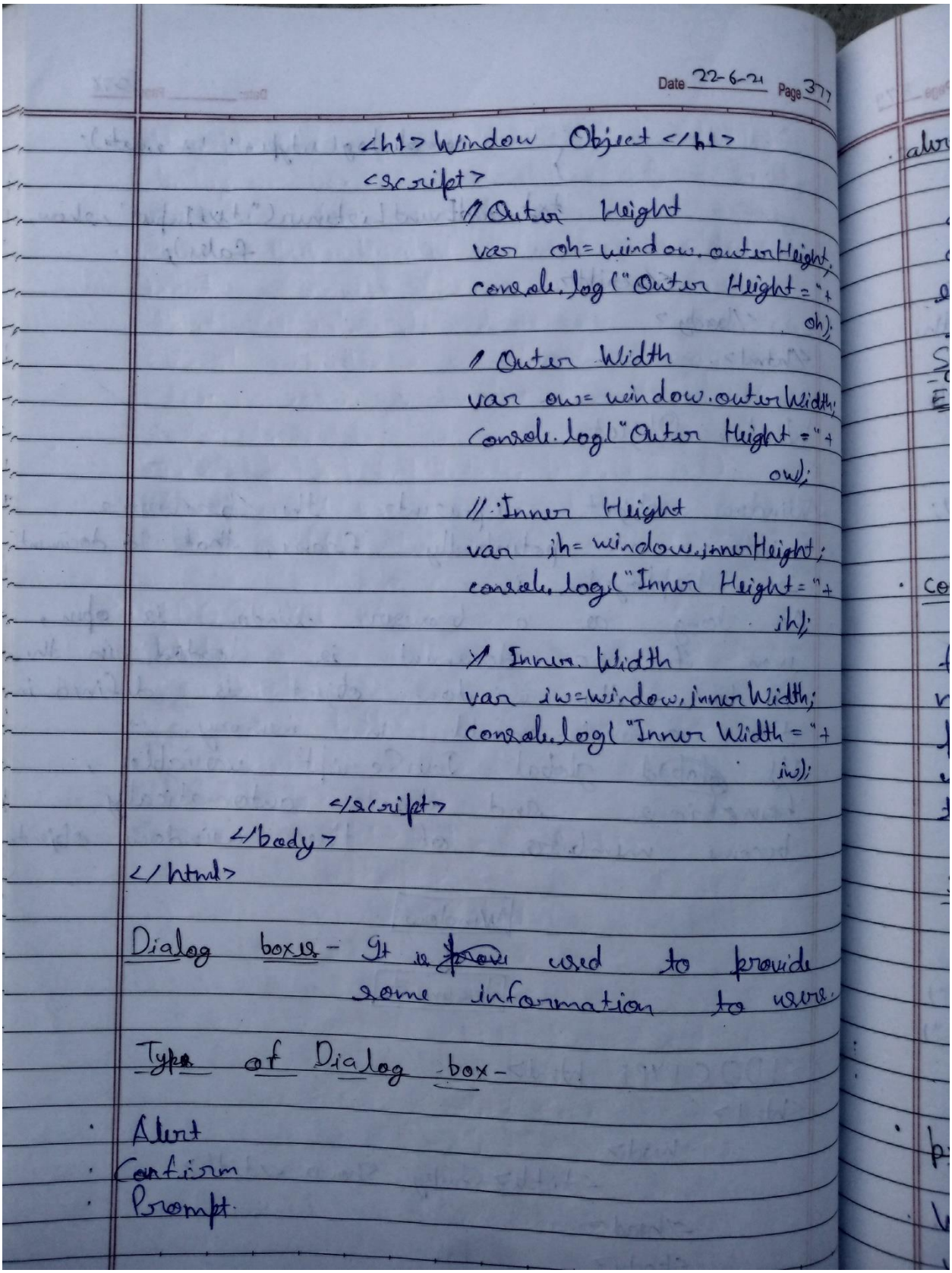
```
    }
    console.log(e.type + ' ' + e.data);
    txt.addEventListener("textInput", show,
                         false);
</script>
</body>
</html>
```

Window Object-

- Window object represents the browser's window or potentially frame, that a document is displayed in.
- As long as a browser window is open, even if no document is loaded in the window, the window object is defined in the current model in memory.
- All ~~global~~ global JavaScript variables, functions and objects automatically become members of the window object.




```
<!DOCTYPE html>
<html>
    <head>
        <title> Geeky Shows </title>
    </head>
    <body>
```



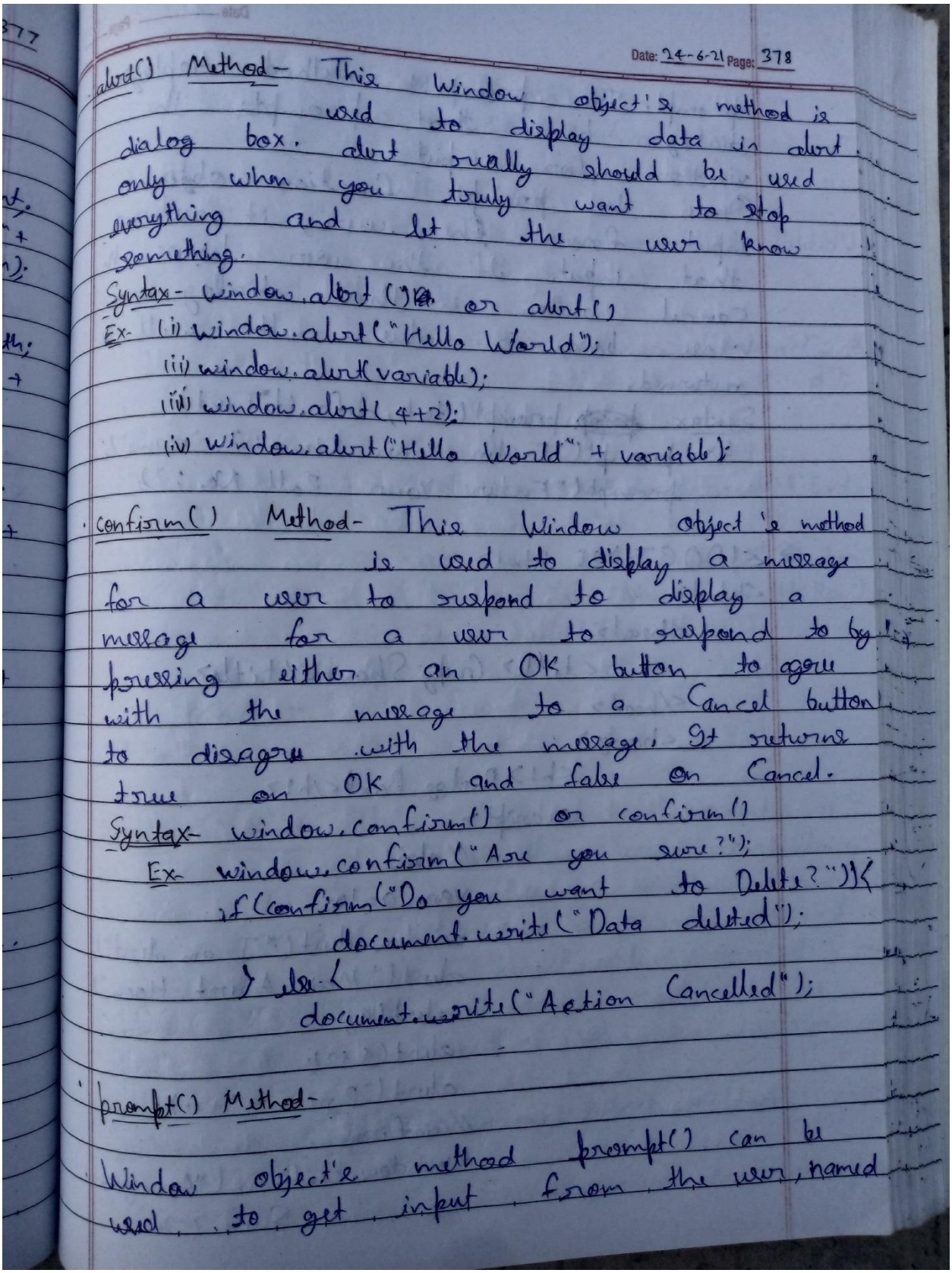
Date 22-6-21 Page 377

```
<h1> Window Object </h1>
<script>
    // Outer Height
    var oh = window.outerHeight;
    console.log("Outer Height = " +
                oh);
    // Outer Width
    var ow = window.outerWidth;
    console.log("Outer Height = " +
                ow);
    // Inner Height
    var ih = window.innerHeight;
    console.log("Inner Height = " +
                ih);
    // Inner Width
    var iw = window.innerWidth;
    console.log("Inner Width = " +
                iw);
</script>
</body>
</html>
```

Dialog box - It is used to provide some information to user.

Type of Dialog box -

- Alert
- Confirm
- Prompt



Date: 24-6-21 Page: 378

alert() Method- This Window object's method is used to display data in alert dialog box. alert really should be used only when you truly want to stop everything and let the user know something.

Syntax- window.alert() or alert()

Ex- (i) window.alert("Hello World");
(ii) window.alert(variable);
(iii) window.alert(4+2);
(iv) window.alert("Hello World" + variable);

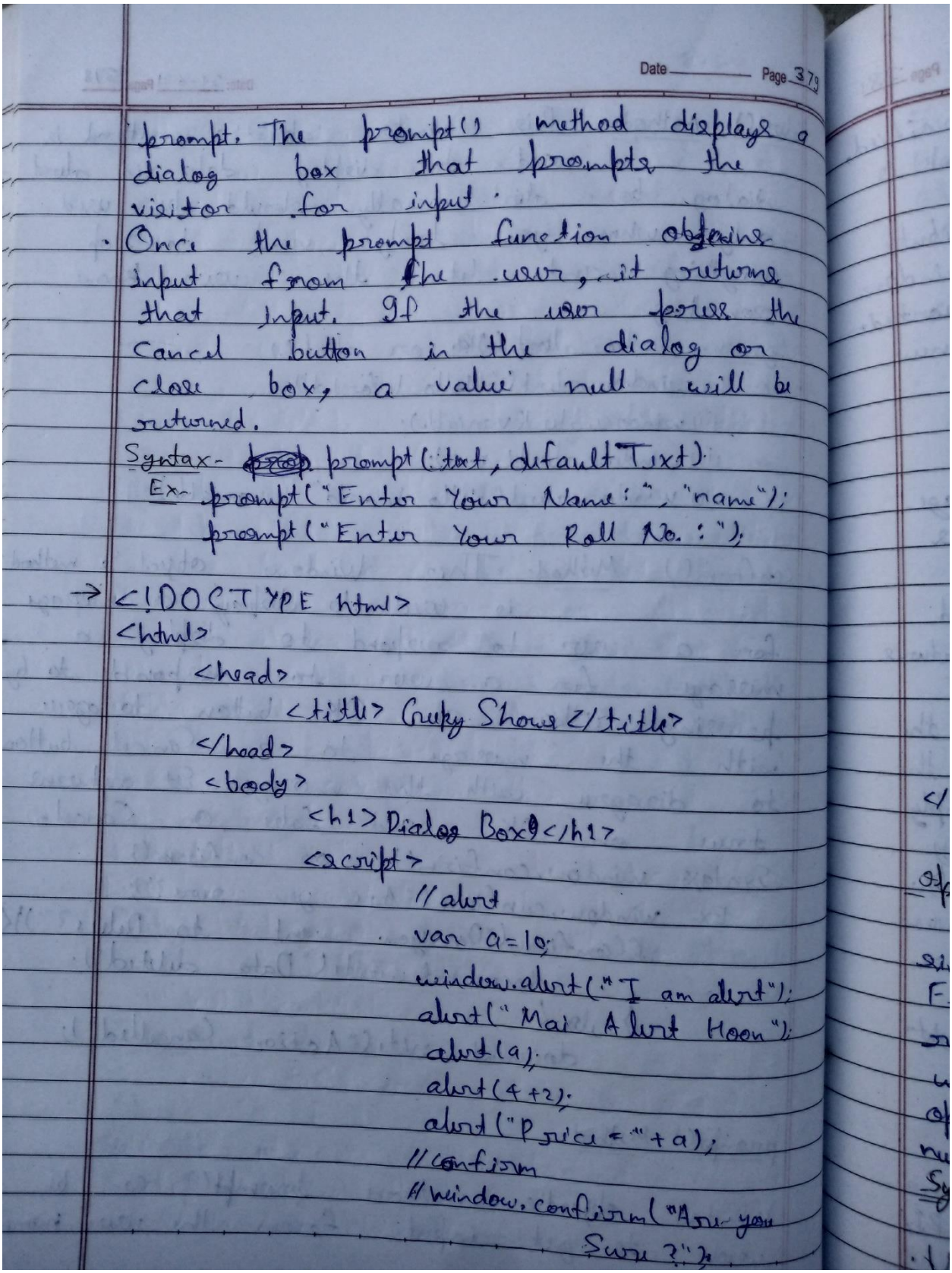
confirm() Method- This Window object's method is used to display a message for a user to respond to display a message for a user to respond to by pressing either an OK button to agree with the message to a Cancel button to disagree with the message. It returns true on OK and false on Cancel.

Syntax- window.confirm() or confirm()

Ex- window.confirm("Are you sure?");
if(confirm("Do you want to Delete?")){
document.write("Data deleted");
} else {
document.write("Action Cancelled");

prompt() Method-

Window object's method prompt() can be used to get input from the user, named



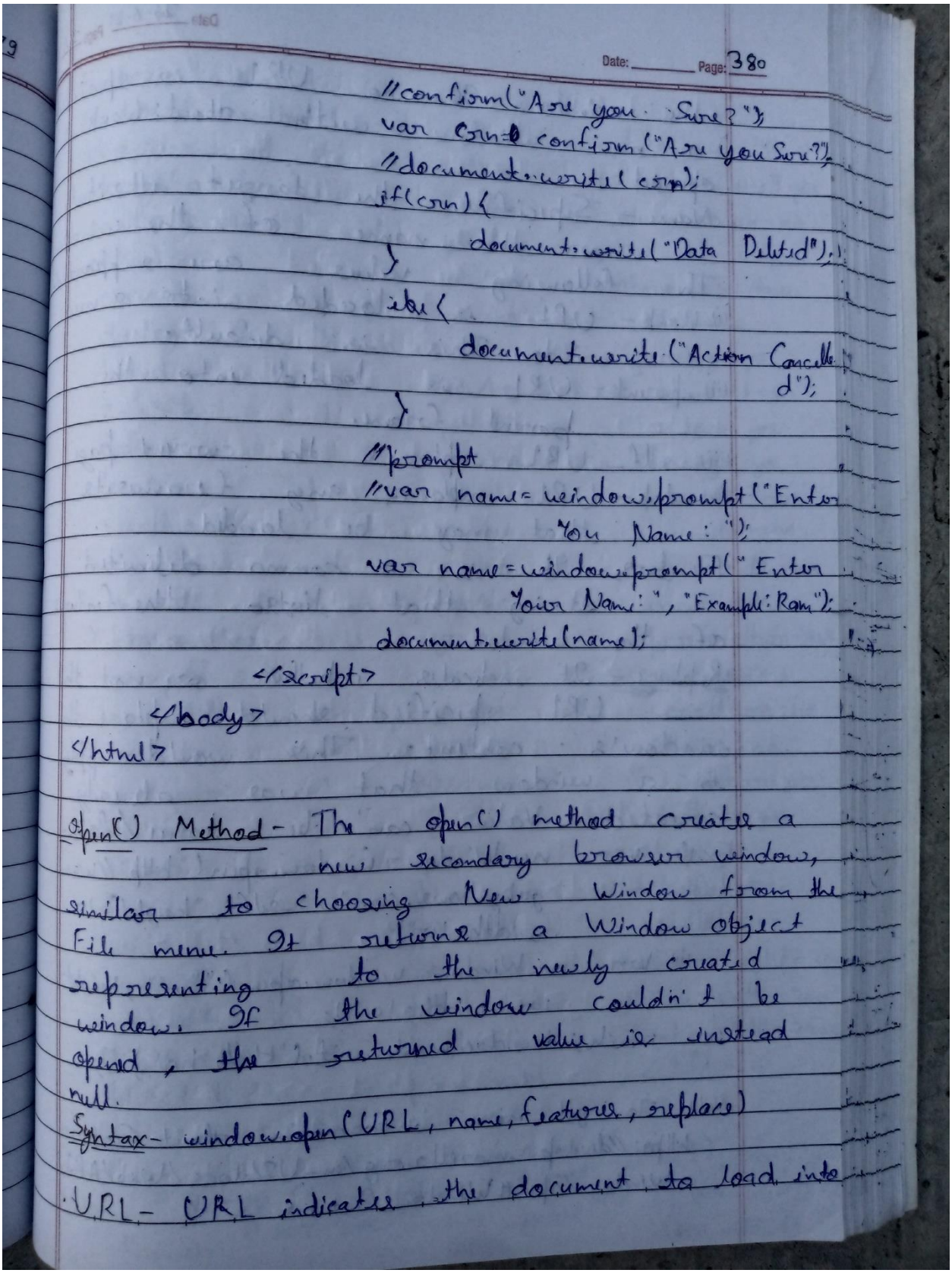
prompt. The prompt() method displays a dialog box that prompts the visitor for input.

- Once the prompt function obtains input from the user, it returns that input. If the user press the cancel button in the dialog or close box, a value null will be returned.

Syntax- prompt(text, default Text)

Ex- prompt("Enter Your Name:", "name");
prompt("Enter Your Roll No.:");

→
```
<!DOCTYPE html>
<html>
    <head>
        <title> Geeky Shows </title>
    </head>
    <body>
        <h1> Dialog Box </h1>
        <script>
            // alert
            var a=10;
            window.alert("I am alert");
            alert("Mai Alert Hoon");
            alert(a);
            alert(4+2);
            alert("Price =" +a);
            // confirm
            // window.confirm("Are you
                                Sure ?");
```



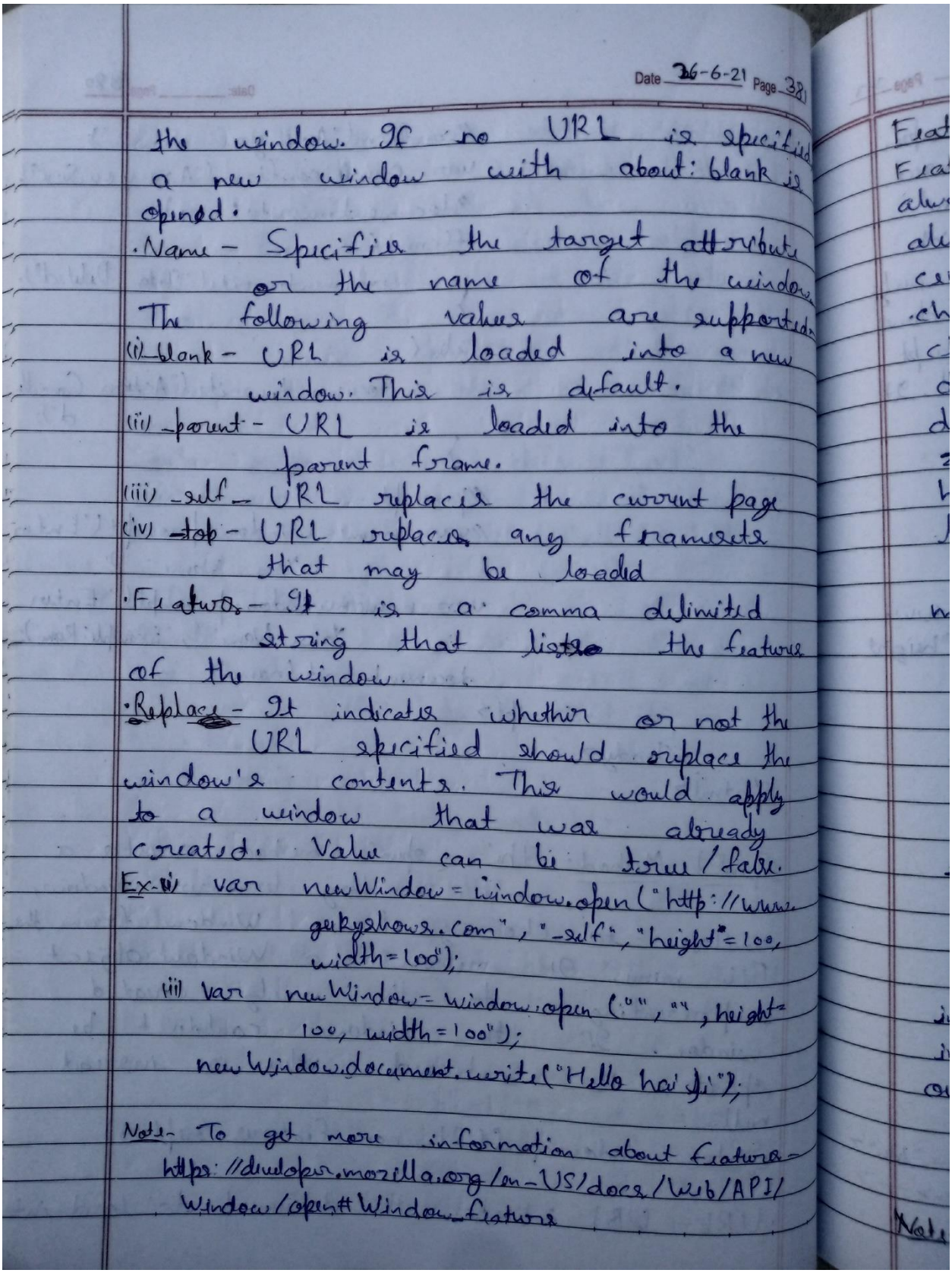
Date: Page: 380

```
//confirm("Are you Sure?");
var crm=confirm("Are you Sure?");
//document.write(crm);
if(crm){
        document.write("Data Deleted");
}
else{
        document.write("Action Cancelled");
}
//prompt
//var name=window.prompt("Enter
                Your Name:");
var name=window.prompt("Enter
                Your Name:","Example:Ram");
document.write(name);
</script>
</body>
</html>
```

open() Method - The open() method creates a new secondary browser window, similar to choosing New Window from the File menu. It returns a Window object representing to the newly created window. If the window couldn't be opened, the returned value is instead null.

Syntax - window.open(URL, name, features, replace)

URL - URL indicates the document to load into



Date 26-6-21 Page 381

the window. If no URL is specified a new window with about:blank is opened.

- Name - Specifies the target attribute or the name of the window. The following values are supported.
  (i) _blank - URL is loaded into a new window. This is default.
  (ii) _parent - URL is loaded into the parent frame.
  (iii) _self - URL replaces the current page
  (iv) _top - URL replaces any framesets that may be loaded
- Features - It is a comma delimited string that lists the features of the window.
- Replace - It indicates whether or not the URL specified should replace the window's contents. This would apply to a window that was already created. Value can be true / false.

Ex- (ii) var newWindow = window.open("http://www.geekyshows.com", "_self", "height=100, width=100");

(iii) var newWindow = window.open("", "", height=100, width=100");
newWindow.document.write("Hello hai ji");

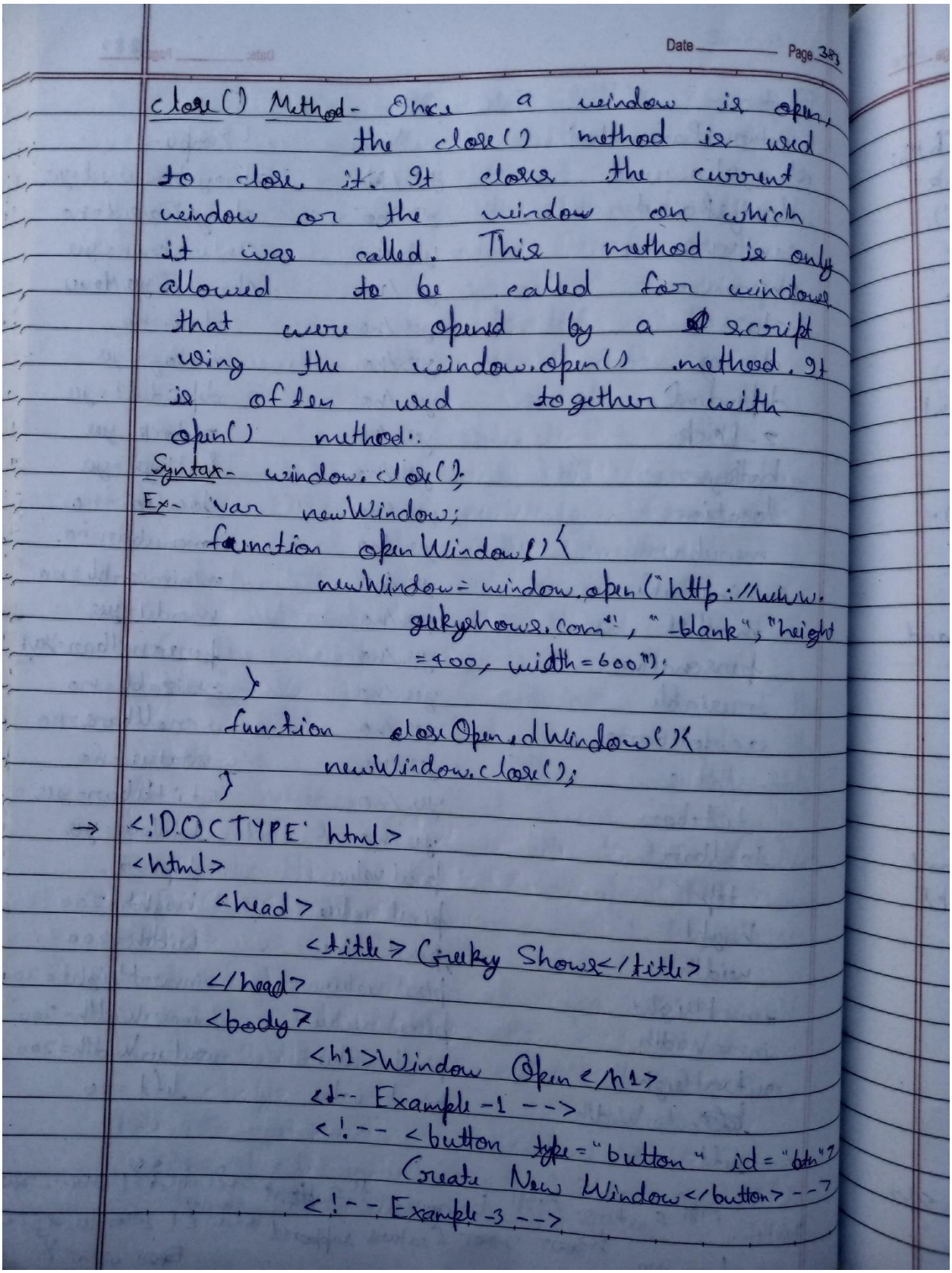
Note - To get more information about features - https://developer.mozilla.org/en-US/docs/Web/API/Window/open#Window_features

Date: ________ Page: 382

Feature -

Feature Parameter	Value	Example -
alwaysLowered	yes / no	alwaysLowered=yes
alwaysRaised	yes / no	alwaysRaised=no
centerscreen	yes / no	centerscreen=yes
chrome	yes / no	chrome=yes/true
close	yes / no	close=no
dialog	yes / no	dialog=yes
dependent	yes / no	dependent=yes
z-lock	yes / no	z-lock=yes
hotkeys	yes / no	hotkeys=yes
location	yes / no	location=no
menubar	yes / no	menubar=no
minimizable	yes / no	minimizable=no
modal	yes / no	modal=yes
personalbar	yes / no	personalbar=yes/true
resizable	yes / no	resizable=no
scrollbars	yes / no	scrollbars=no
status	yes / no	status=no
titlebar	yes / no	titlebar=yes
toolbar	yes / no	toolbar=yes
top	pixel value	top=20
height	pixel value	height=200
width	pixel value	width=200
innerHeight	pixel value	innerHeight=200
innerWidth	pixel value	innerWidth=200
outerHeight	pixel value	outerWidth=200
outerWidth	pixel value	left=20
left		

Note- सभी Feature सभी browser में support नहीं करते हैं। अलग-अलग browser में अलग-अलग feature support करते हैं। इसका use बहुत कम किया जाता है।



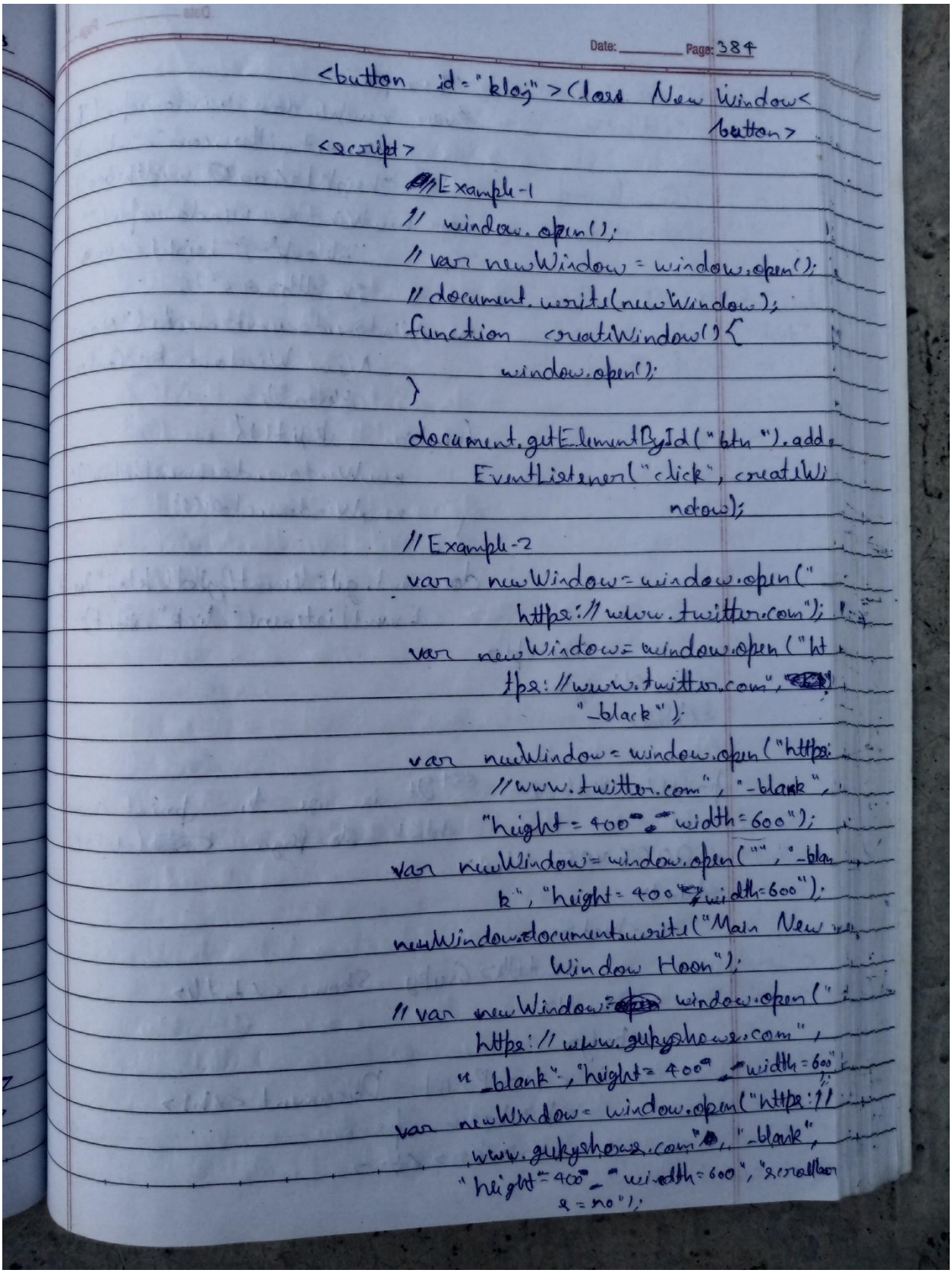
Date _____ Page 383

close() Method - Once a window is open, the close() method is used to close it. It closes the current window or the window on which it was called. This method is only allowed to be called for windows that were opened by a script using the window.open() method. It is often used together with open() method.

Syntax - window.close();

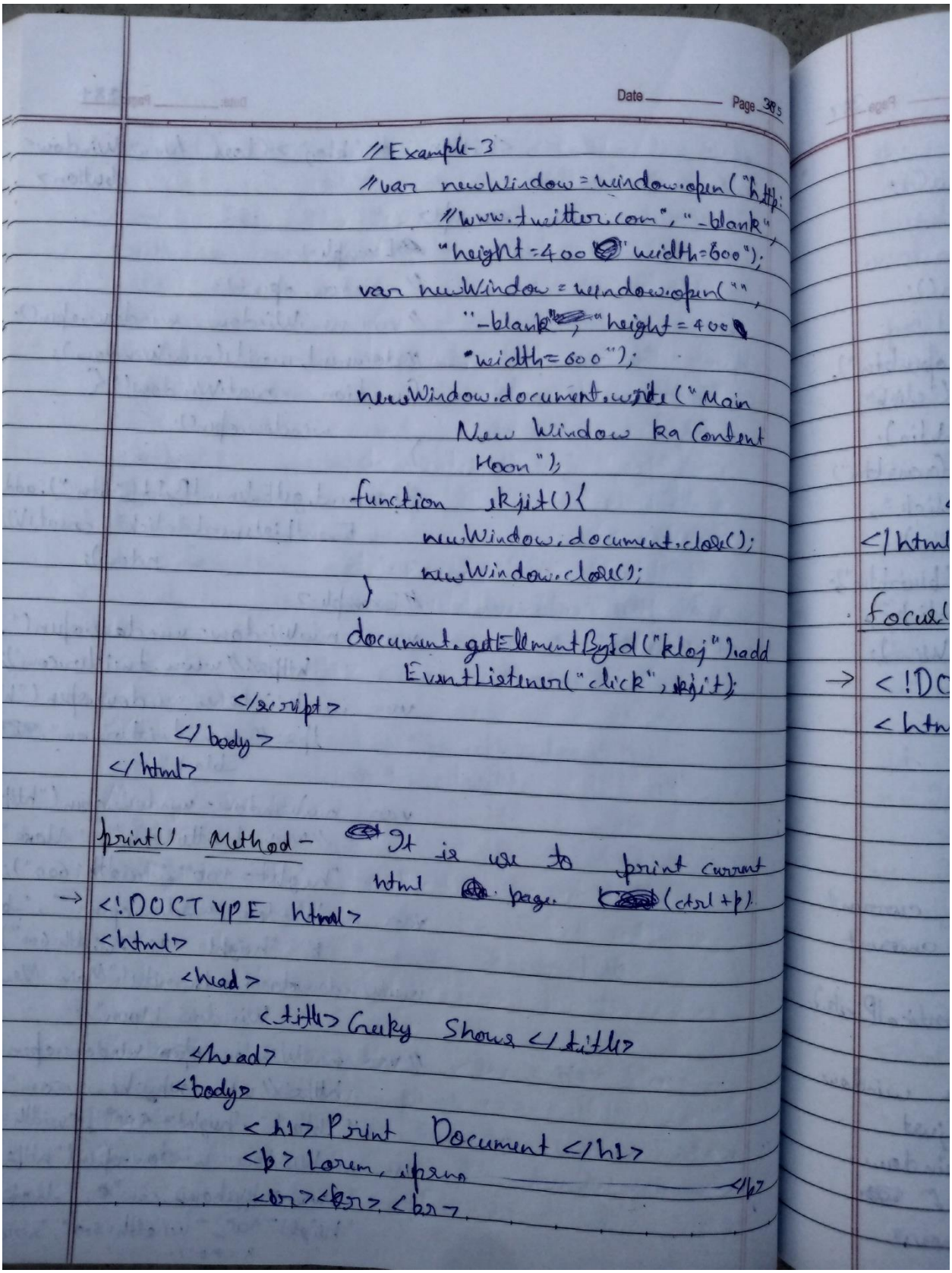
Ex - var newWindow;
function openWindow(){
newWindow = window.open("http://www.geekyshows.com", "_blank", "height=400, width=600");
}
function closeOpenedWindow(){
newWindow.close();
}

→ <!DOCTYPE html>
<html>
<head>
<title> Geeky Shows</title>
</head>
<body>
<h1>Window Open</h1>
<!-- Example-1 -->
<!-- <button type="button" id="btn">Create New Window</button> -->
<!-- Example-3 -->



Date: Page: 384

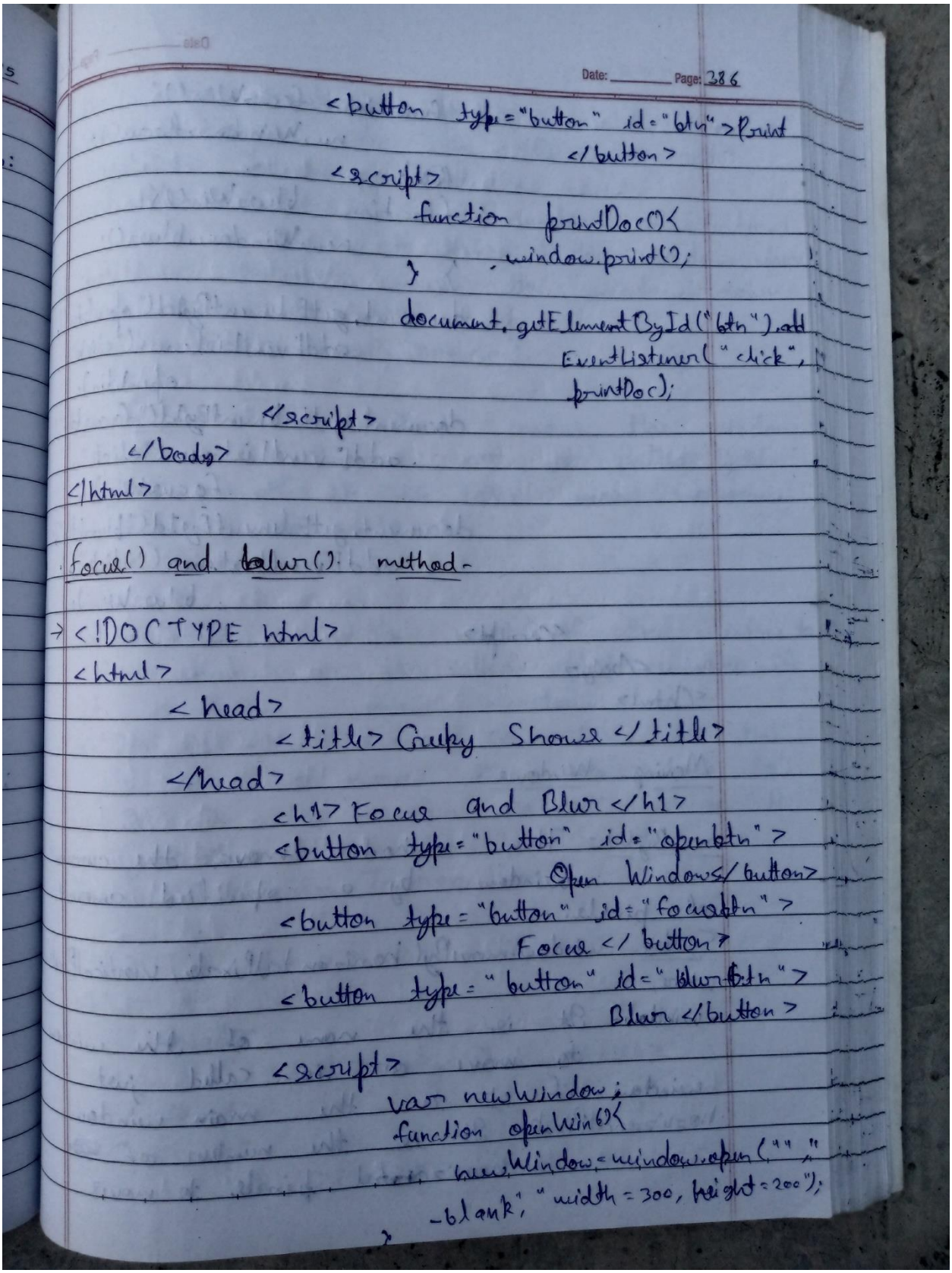
```
<button id="klaj">Close New Window</button>

<script>
    //Example-1
    // window.open();
    // var newWindow = window.open();
    // document.write(newWindow);
    function creatiWindow(){
        window.open();
    }
    document.getElementById("btn").addEventListener("click", creatiWindow);

    //Example-2
    var newWindow = window.open("https://www.twitter.com");
    var newWindow = window.open("https://www.twitter.com", "_black");
    var newWindow = window.open("https://www.twitter.com", "_blank",
        "height=400, width=600");
    var newWindow = window.open("", "_blank", "height=400, width=600");
    newWindow.document.write("Main New Window Hoon");
    // var newWindow = window.open("https://www.geekyshows.com",
    "_blank", "height=400, width=600");
    var newWindow = window.open("https://www.geekyshows.com", "_blank",
    "height=400, width=600", "scrollbars=no");
```



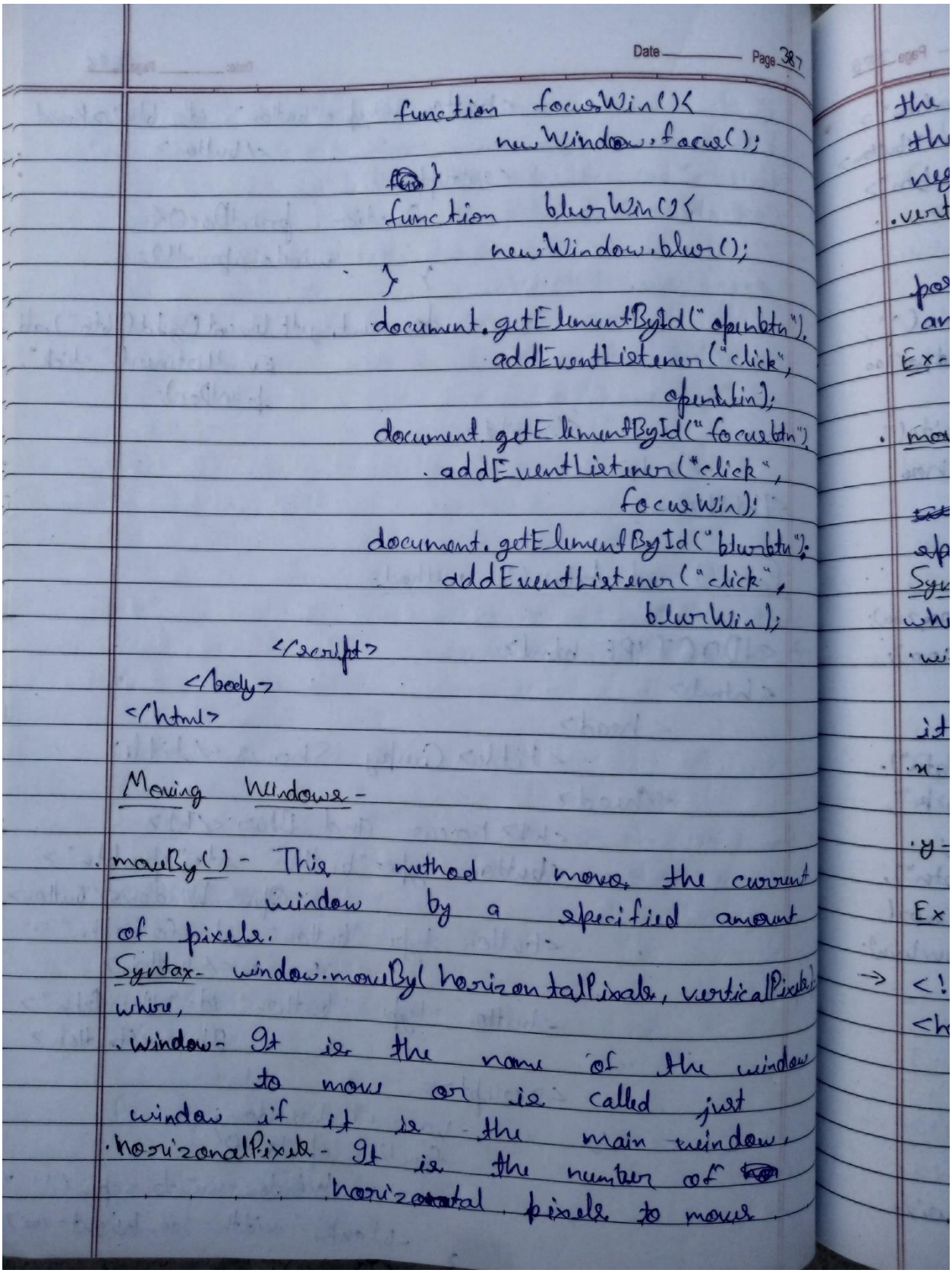
```
//Example-3
//var newWindow = window.open("http:
    //www.twitter.com", "_blank",
    "height=400, width=600");
var newWindow = window.open("",
    "_blank", "height=400,
    "width=600");
newWindow.document.write("Main
    New Window ka Content
    Hoon");
function jkjit(){
    newWindow.document.close();
    newWindow.close();
}
document.getElementById("kloj").add
    EventListener("click", jkjit);
</script>
</body>
</html>
```

print() Method - It is use to print current html page. (ctrl+p)

```
<!DOCTYPE html>
<html>
    <head>
        <title> Geeky Shows </title>
    </head>
    <body>
        <h1> Print Document </h1>
        <p> Lorem, ipsum
        <br><br><br>
```



```
Date: _____ Page: 386

<button type="button" id="btn">Print
</button>
<script>
    function printDoc(){
        window.print();
    }
    document.getElementById("btn").add
        EventListener("click",
        printDoc);
</script>
</body>
</html>

focus() and blur() method-

<!DOCTYPE html>
<html>
    <head>
        <title> Geeky Shows </title>
    </head>
    <h1> Focus and Blur </h1>
    <button type="button" id="openbtn">
        Open Windows</button>
    <button type="button" id="focusbtn">
        Focus </button>
    <button type="button" id="blurbtn">
        Blur </button>
    <script>
        var newWindow;
        function openWin(){
            newWindow=window.open("","
            _blank", "width=300, height=200");
        }
```



```
function focusWin(){
    newWindow.focus();
}
function blurWin(){
    newWindow.blur();
}
document.getElementById("openbtn").
    addEventListener("click",
        openWin);
document.getElementById("focusbtn").
    addEventListener("click",
        focusWin);
document.getElementById("blurbtn").
    addEventListener("click",
        blurWin);
</script>
</body>
</html>
```

Moving Windows -

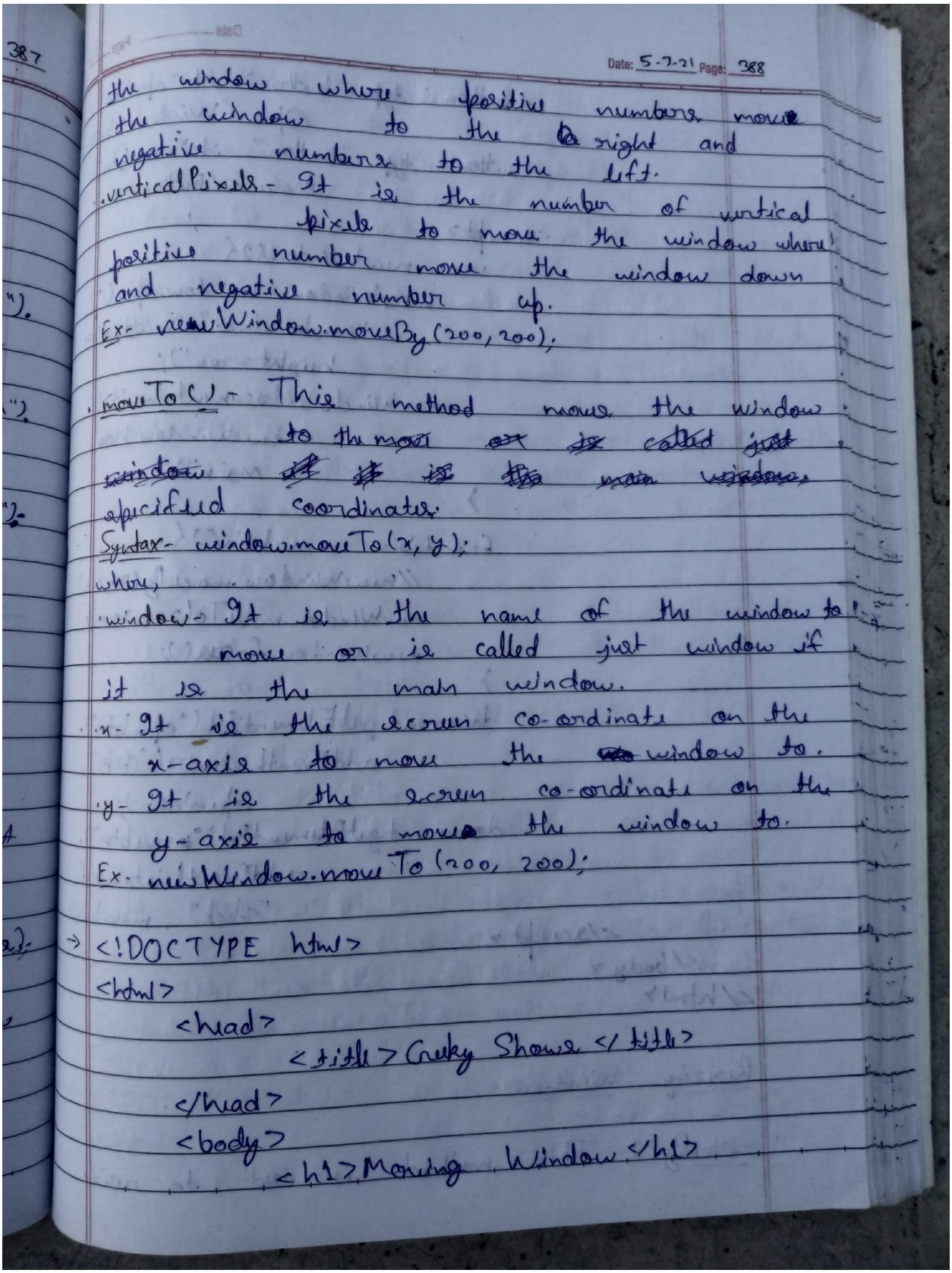
.moveBy() - This method moves the current window by a specified amount of pixels.

Syntax- window.moveBy(horizontalPixels, verticalPixels);

where,

.window- It is the name of the window to move or is called just window if it is the main window.

.horizontalPixels- It is the number of horizontal pixels to move



Date: 5-7-21 Page: 388

the window where positive numbers move the window to the right and negative numbers to the left.

.verticalPixels - It is the number of vertical pixels to move the window where positive number move the window down and negative number up.

Ex- newWindow.moveBy(200, 200);

.moveTo() - This method moves the window to the ~~most or is called just window if it is the main window~~ specified coordinates.

Syntax- window.moveTo(x, y);

where,

.window - It is the name of the window to move or is called just window if it is the main window.

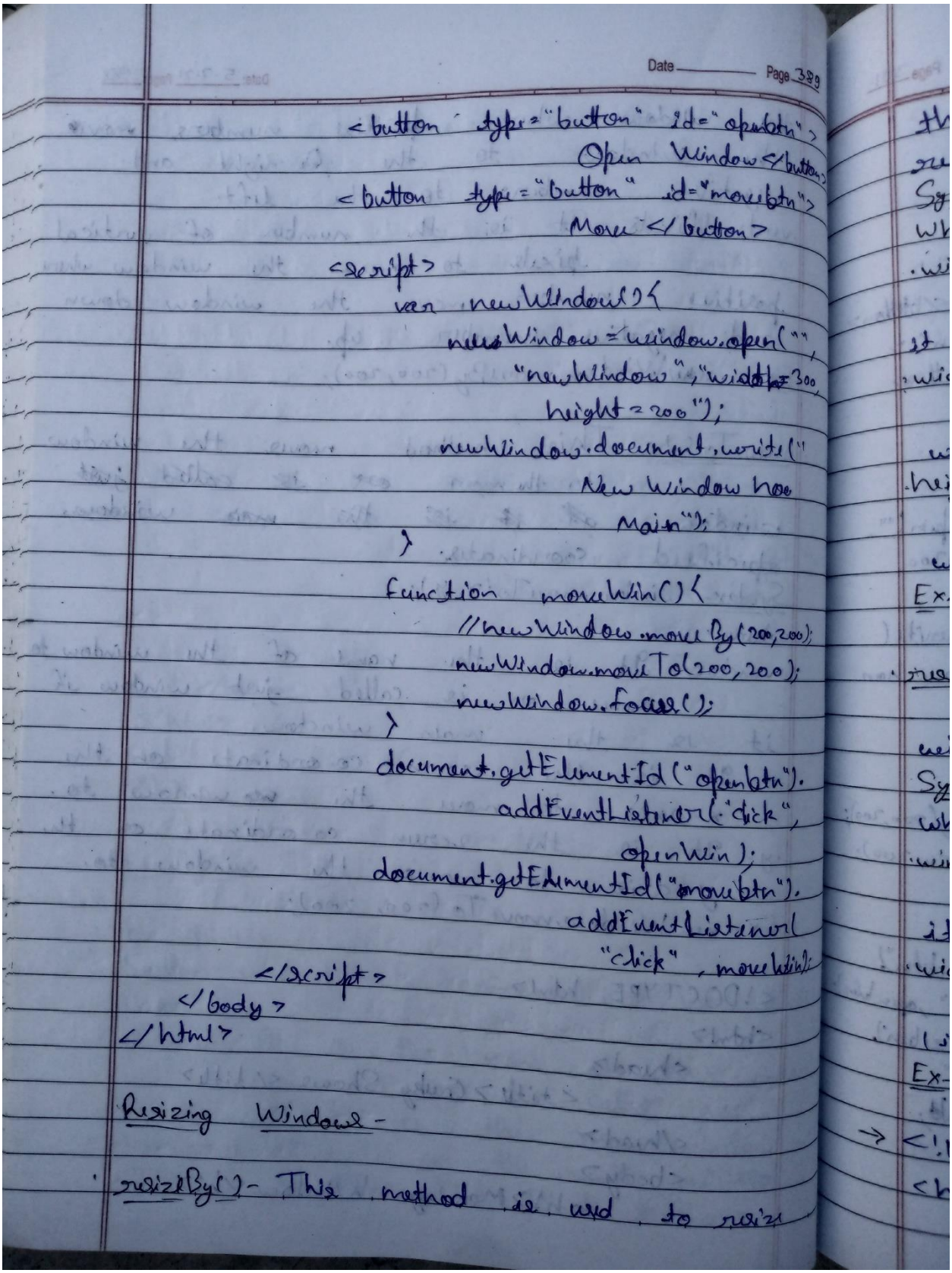
.x - It is the screen co-ordinate on the x-axis to move the window to.

.y - It is the screen co-ordinate on the y-axis to move the window to.

Ex- newWindow.moveTo(200, 200);

→

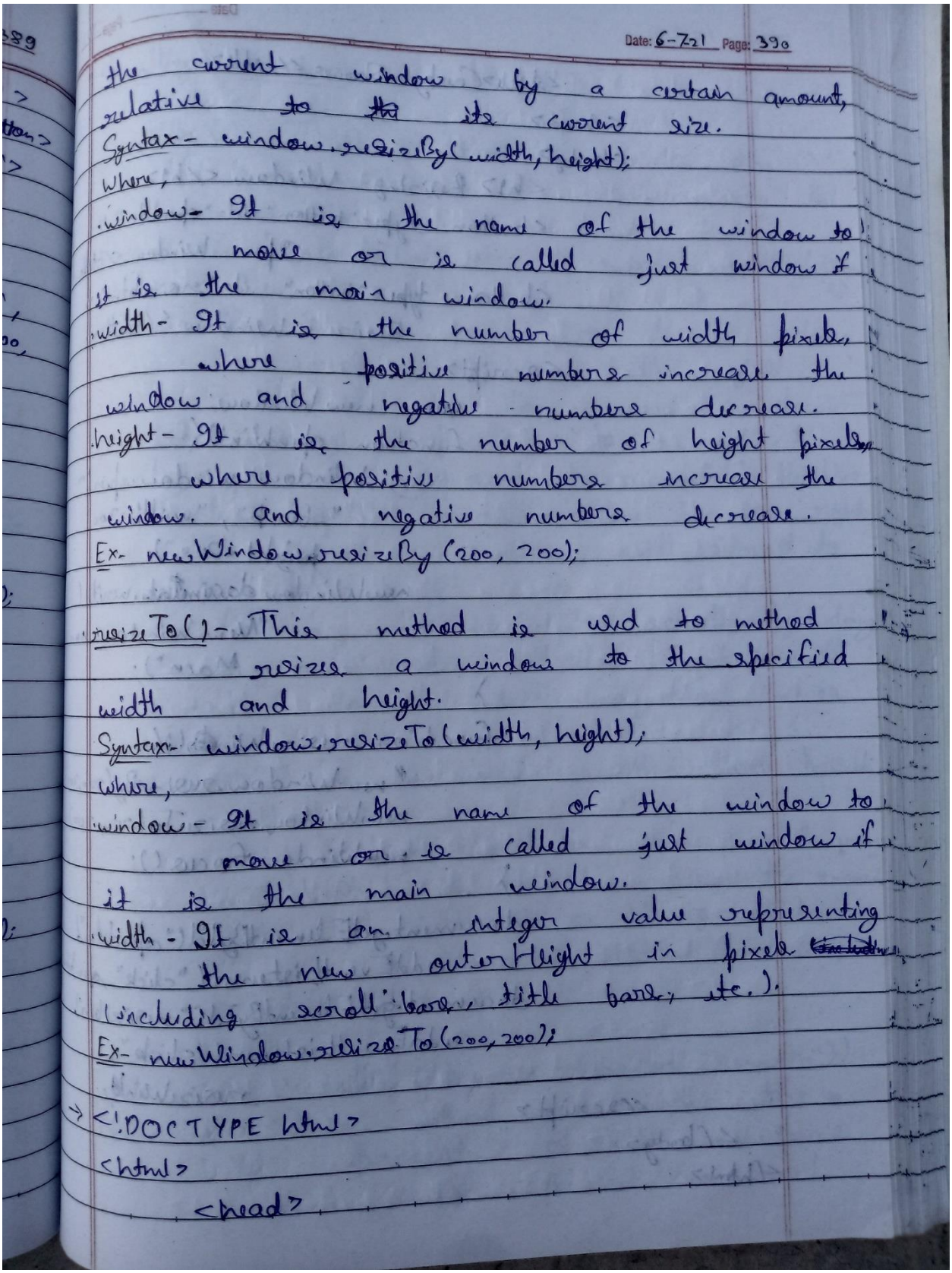
```
<!DOCTYPE html>
<html>
    <head>
        <title> Geeky Shows </title>
    </head>
    <body>
        <h1> Moving Window </h1>
```



```
<button type="button" id="openbtn">
        Open Window</button>
<button type="button" id="movebtn">
        Move </button>
<script>
    var newWindow(){
        newWindow = window.open("",
        "newWindow","width=300,
        height=200");
        newWindow.document.write("
            New Window now
            Main");
    }
    function moveWin(){
        //newWindow.moveBy(200,200);
        newWindow.moveTo(200,200);
        newWindow.focus();
    }
    document.getElementId("openbtn").
        addEventListener("click",
            openWin);
    document.getElementId("movebtn").
        addEventListener(
        "click", moveWin);
</script>
</body>
</html>
```

Resizing Windows -

resizeBy() - This method is used to resize



Date: 6-7-21 Page: 390

the current window by a certain amount, relative to its current size.

Syntax- window.resizeBy(width, height);

Where,

window- It is the name of the window to move or is called just window if it is the main window.

width- It is the number of width pixels where positive numbers increase the window and negative numbers decrease.

height- It is the number of height pixels where positive numbers increase the window. and negative numbers decrease.

Ex- newWindow.resizeBy(200, 200);

resizeTo()- This method is used to method resizes a window to the specified width and height.

Syntax- window.resizeTo(width, height);

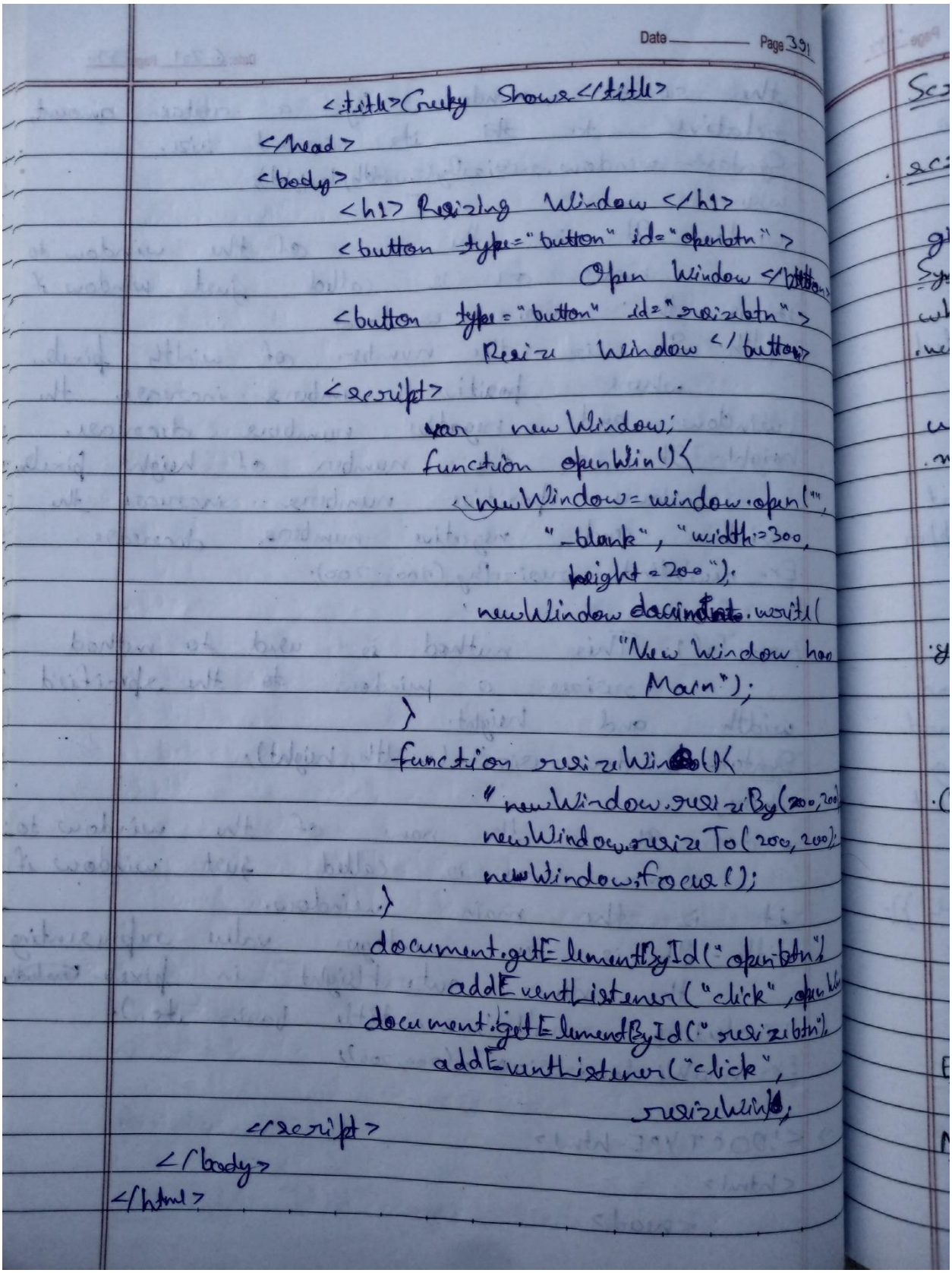
Where,

window- It is the name of the window to move or is called just window if it is the main window.

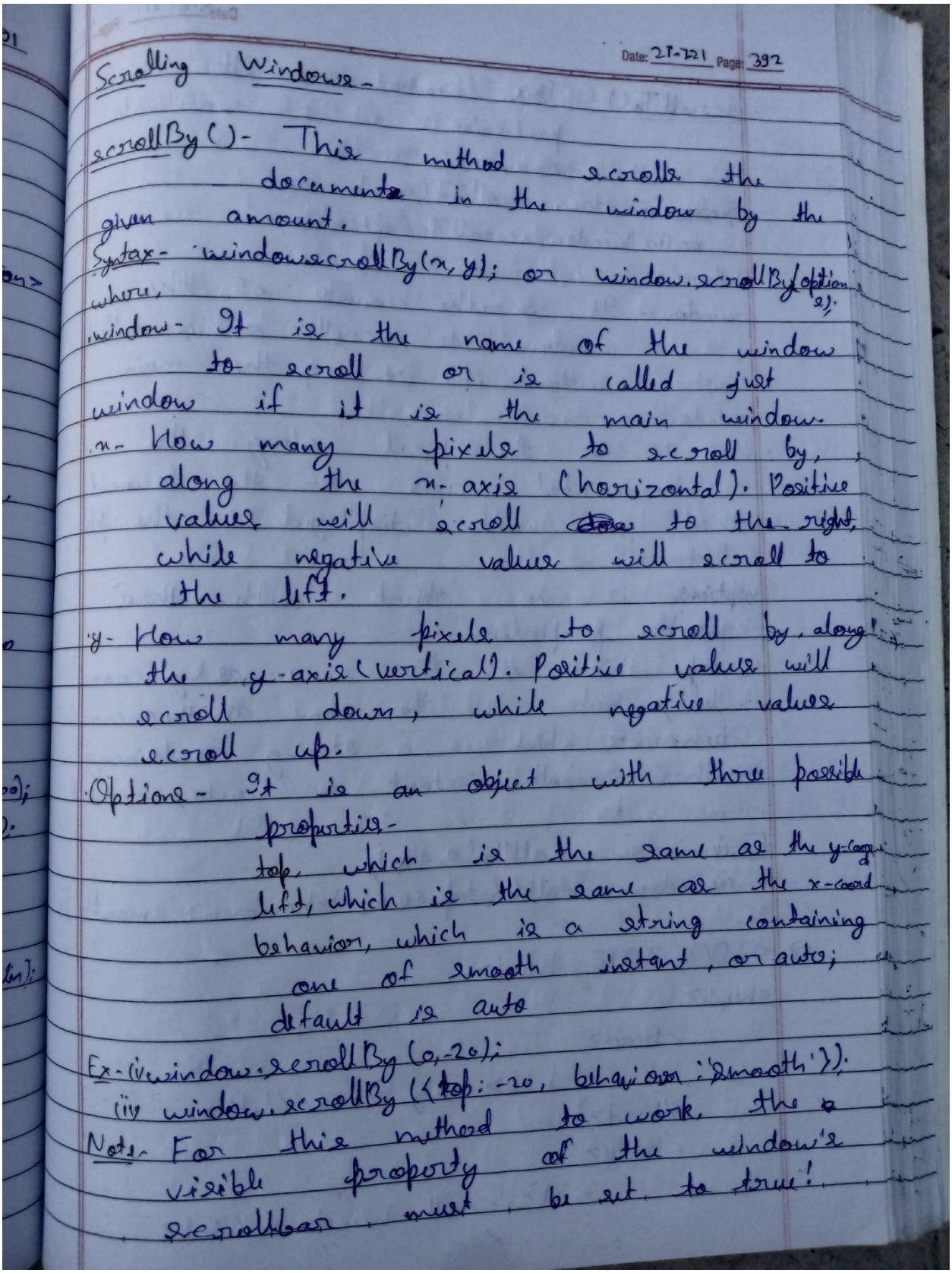
width- It is an integer value representing the new outerHeight in pixels (including scroll bars, title bars, etc.).

Ex- newWindow.resizeTo(200, 200);

```
<!DOCTYPE html>
<html>
    <head>
```



```
<title>Geeky Shows</title>
</head>
<body>
    <h1> Resizing Window </h1>
    <button type="button" id="openbtn" >
                Open Window </button>
    <button type="button" id="resizebtn" >
                Resize Window </button>
<script>
        var newWindow;
        function openWin(){
            newWindow=window.open("",
                "_blank", "width=300,
                height=200");
            newWindow.document.write(
                "New Window has
                Main");
        }
        function resizeWin(){
            newWindow.resizeBy(200,200);
            newWindow.resizeTo(200,200);
            newWindow.focus();
        }
        document.getElementById("openbtn")
            addEventListener("click", openWin
        document.getElementById("resizebtn")
            addEventListener("click",
                resizeWin);
</script>
</body>
</html>
```



Date: 27-7-21 Page: 392

Scrolling Windows -

scrollBy() - This method scrolls the documents in the window by the given amount.

Syntax - window.scrollBy(x, y); or window.scrollBy(options);

where,

window - It is the name of the window to scroll or is called just window if it is the main window.

x - How many pixels to scroll by, along the x-axis (horizontal). Positive values will scroll to the right, while negative value will scroll to the left.

y - How many pixels to scroll by, along the y-axis (vertical). Positive value will scroll down, while negative values scroll up.

Options - It is an object with three possible properties -

top, which is the same as the y-coord

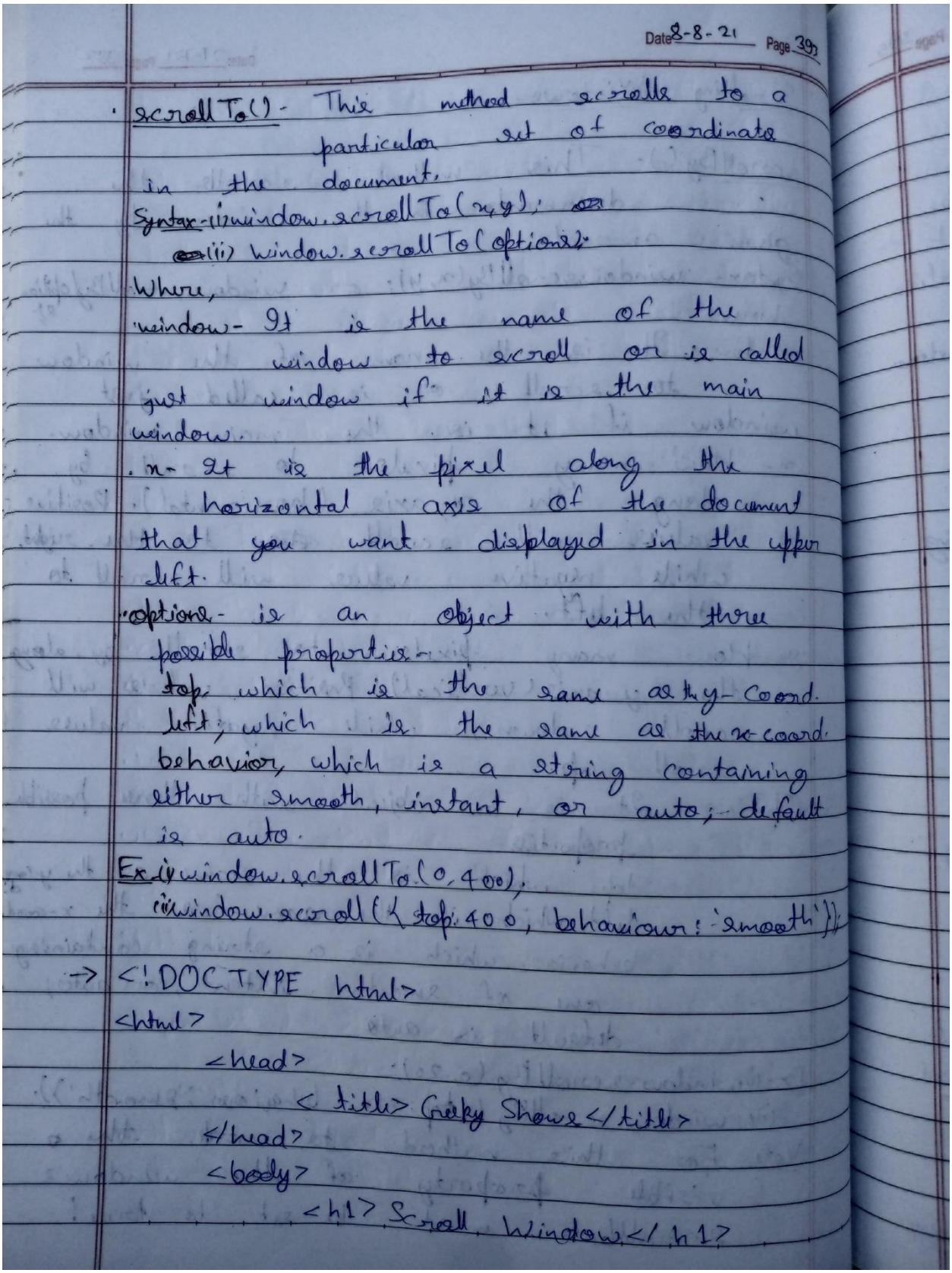
left, which is the same as the x-coord

behavior, which is a string containing one of smooth, instant, or auto; default is auto

Ex - (i) window.scrollBy(0, -20);

(ii) window.scrollBy({top: -20, behavior: 'smooth'});

Note - For this method to work, the scrollbar must be set to true!



Date 8-8-21 Page 391

- scrollTo() - This method scrolls to a particular set of coordinate in the document.

Syntax - (i) window.scrollTo(x,y);
(ii) window.scrollTo(options);

Where,

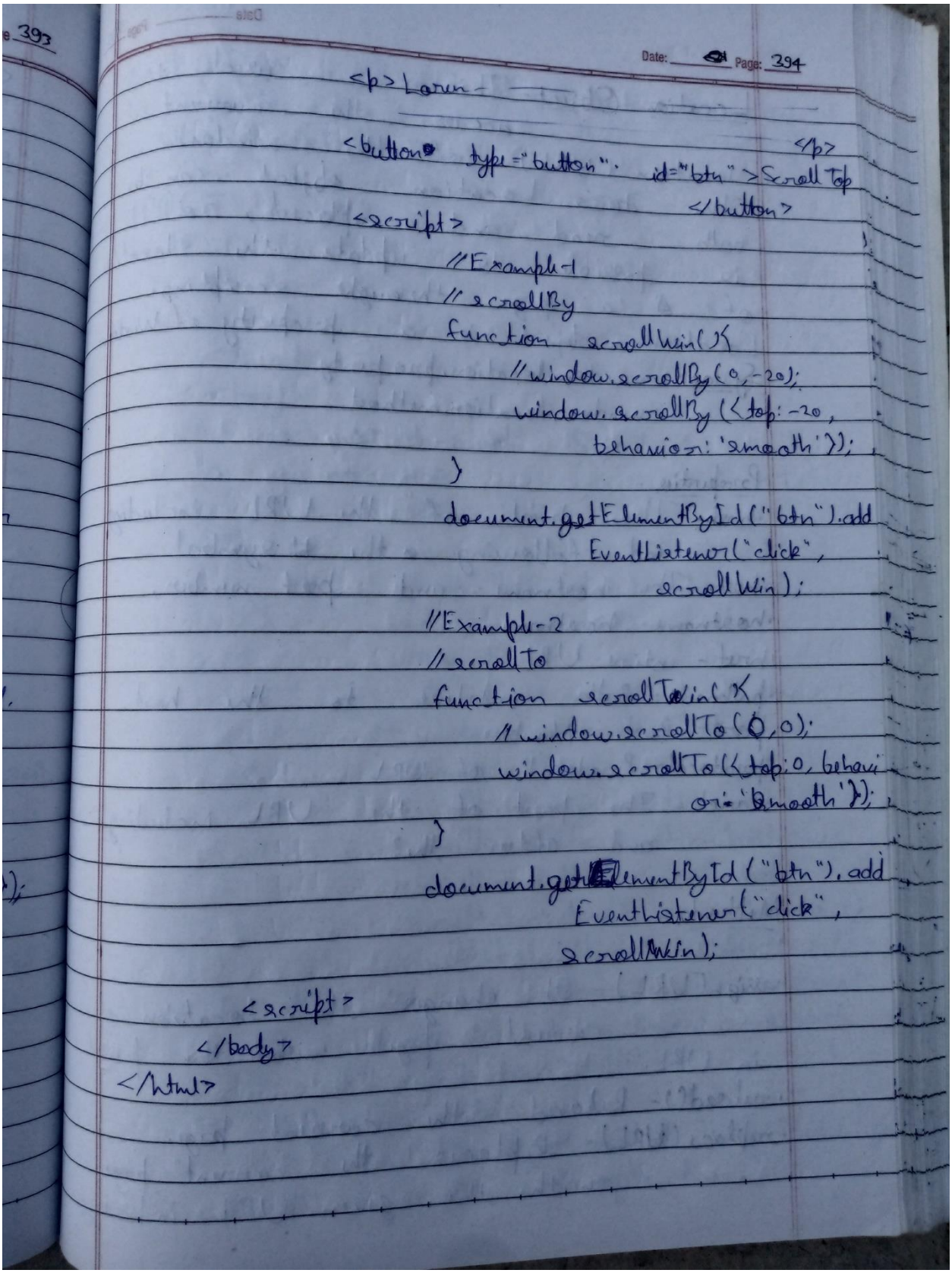
window - It is the name of the window to scroll or is called just window if it is the main window.

x - It is the pixel along the horizontal axis of the document that you want displayed in the upper left.

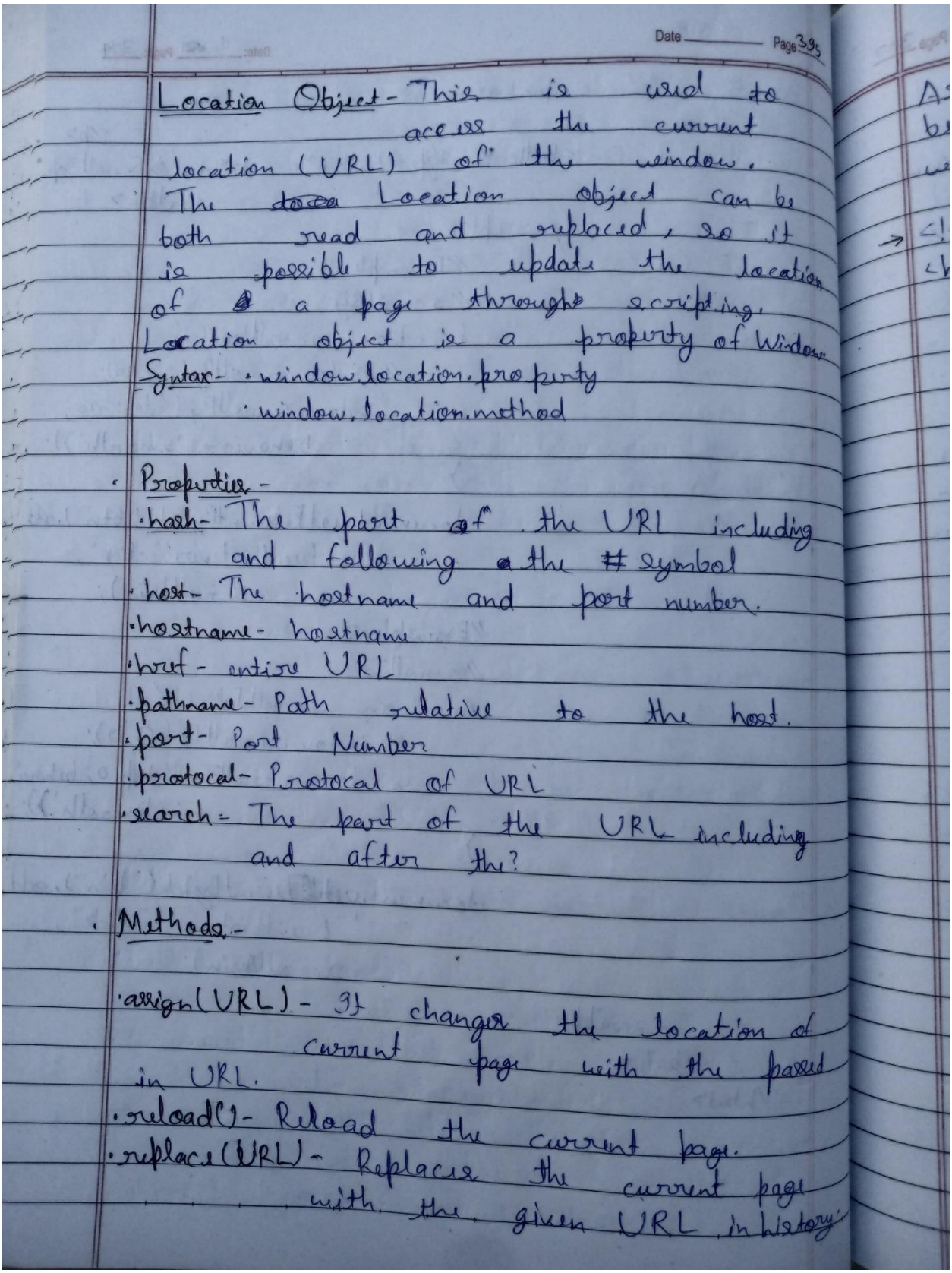
options - is an object with three possible properties -
top, which is the same as the y-coord.
left, which is the same as the x-coord.
behavior, which is a string containing either smooth, instant, or auto; default is auto.

Ex (i) window.scrollTo(0,400);
(ii) window.scroll({ top: 400, behaviour: 'smooth' });

→
```
<!DOCTYPE html>
<html>
    <head>
        <title> Geeky Shows </title>
    </head>
    <body>
        <h1> Scroll Window </h1>
```



```
<p> Lorem -
</p>
<button type="button" id="btn"> Scroll Top
</button>
<script>
    //Example-1
    //scrollBy
    function scrollWin(){
        //window.scrollBy(0,-20);
        window.scrollBy({top:-20,
            behavior:'smooth'});
    }
    document.getElementById("btn").add
        EventListener("click",
        scrollWin);
    //Example-2
    //scrollTo
    function scrollToWin(){
        //window.scrollTo(0,0);
        window.scrollTo({top:0, behavi
            or:'smooth'});
    }
    document.getElementById("btn").add
        EventListener("click",
        scrollWin);
</script>
</body>
</html>
```



Date _____ Page 395

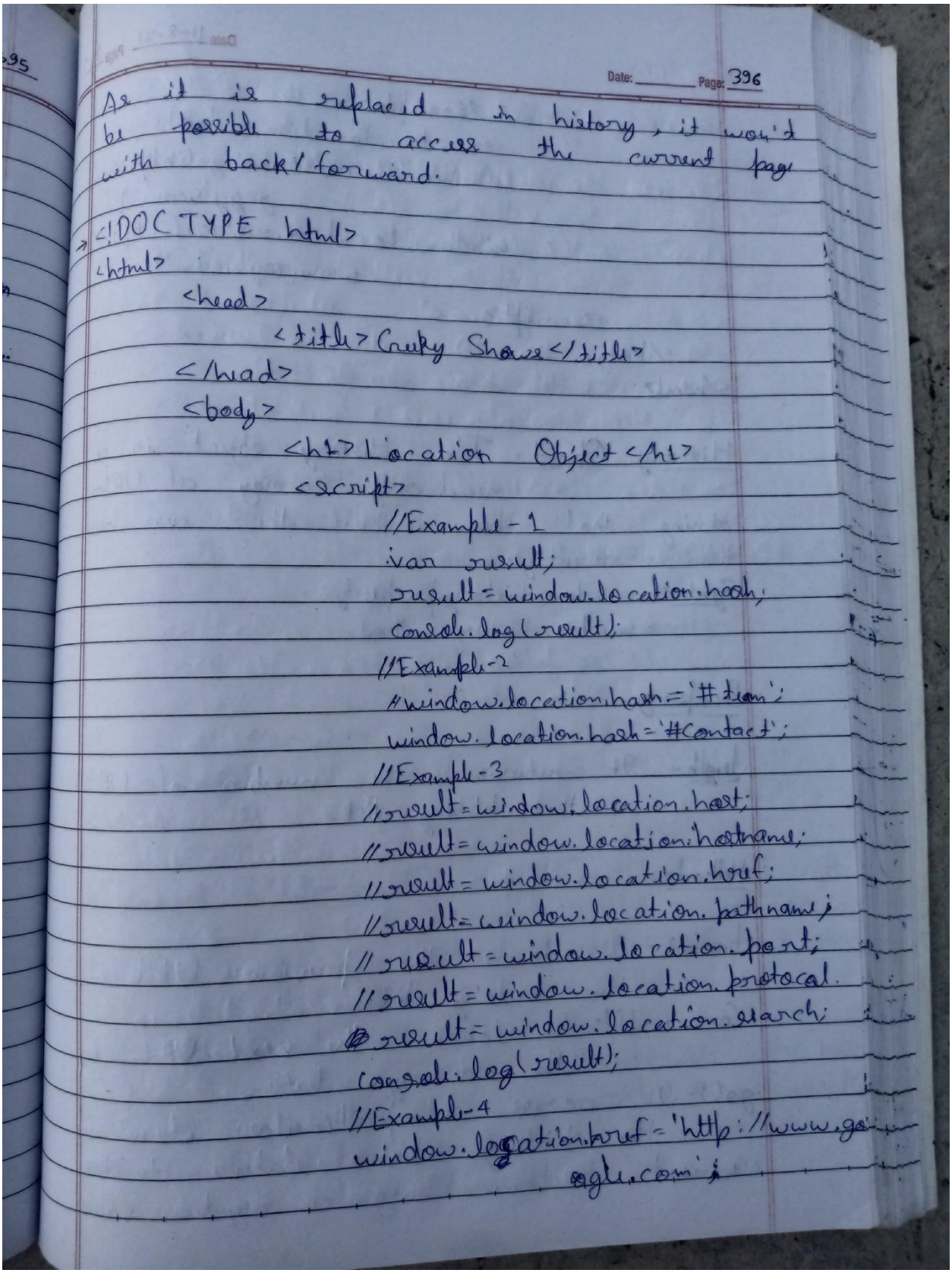
Location Object - This is used to access the current location (URL) of the window. The Location object can be both read and replaced, so it is possible to update the location of a page through scripting. Location object is a property of Window.

Syntax - · window.location.property
· window.location.method

· Properties -
· hash - The part of the URL including and following the # symbol
· host - The hostname and port number.
· hostname - hostname
· href - entire URL
· pathname - Path relative to the host.
· port - Port Number
· protocal - Protocal of URL
· search - The part of the URL including and after the ?

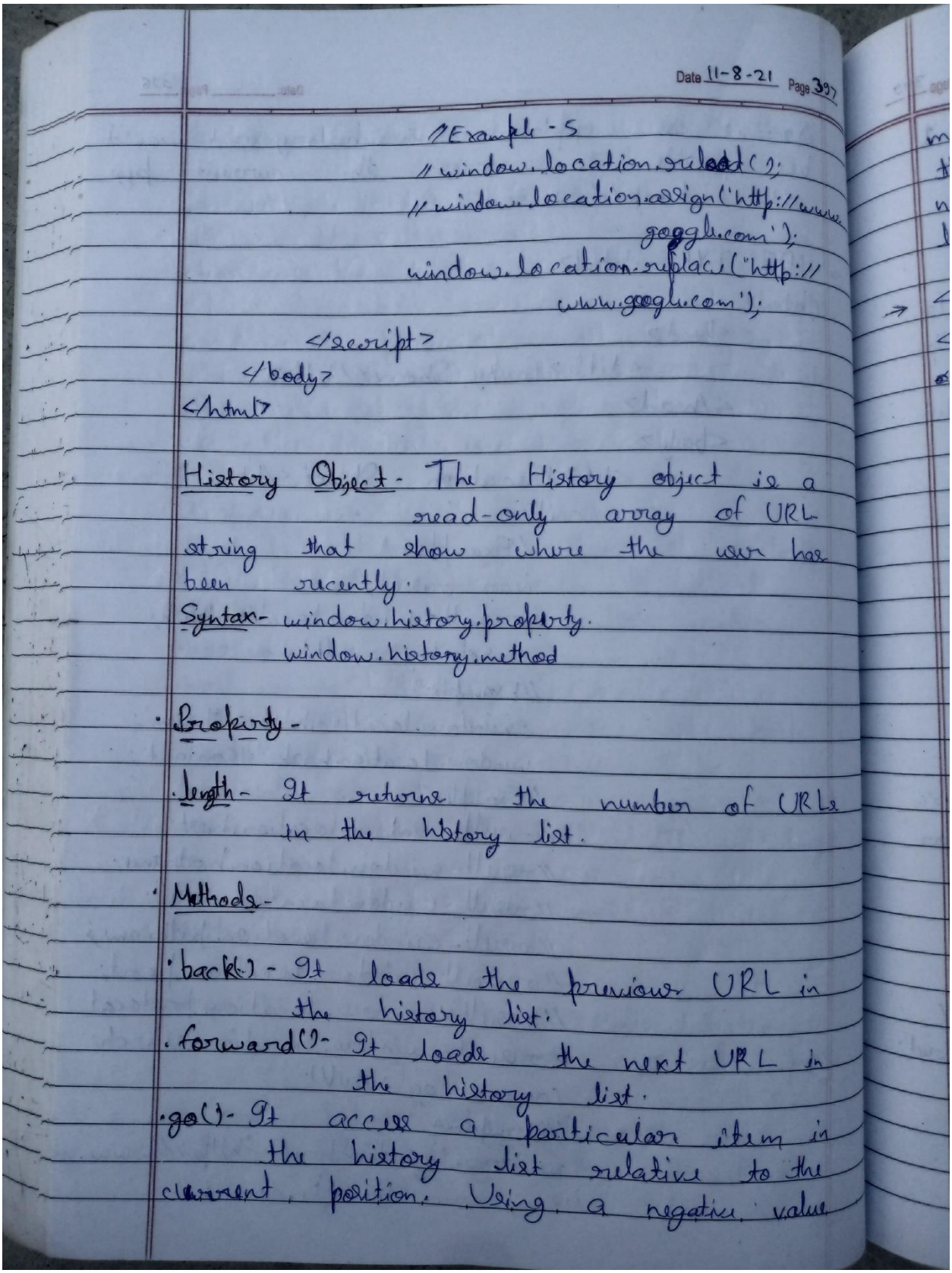
· Methods -

· assign(URL) - It changes the location of current page with the passed in URL.
· reload() - Reload the current page.
· replace(URL) - Replaces the current page with the given URL in history.



As it is replaced in history, it won't be possible to access the current page with back/forward.

```
<!DOCTYPE html>
<html>
    <head>
        <title> Cruky Shows </title>
    </head>
    <body>
        <h1> Location Object </h1>
        <script>
            //Example - 1
            var result;
            result = window.location.hash;
            console.log(result);
            //Example-2
            //window.location.hash = '#team';
            window.location.hash = '#contact';
            //Example-3
            //result = window.location.host;
            //result = window.location.hostname;
            //result = window.location.href;
            //result = window.location.pathname;
            //result = window.location.port;
            //result = window.location.protocal.
            result = window.location.search;
            console.log(result);
            //Example-4
            window.location.href = 'http://www.go
                                    ogle.com';
```



Date 11-8-21 Page 397

```
//Example - 5
// window.location.reload();
// window.location.assign('http://www.
                          google.com');
window.location.replace("http://
                    www.google.com');
    </script>
  </body>
</html>
```

History Object - The History object is a read-only array of URL string that show where the user has been recently.

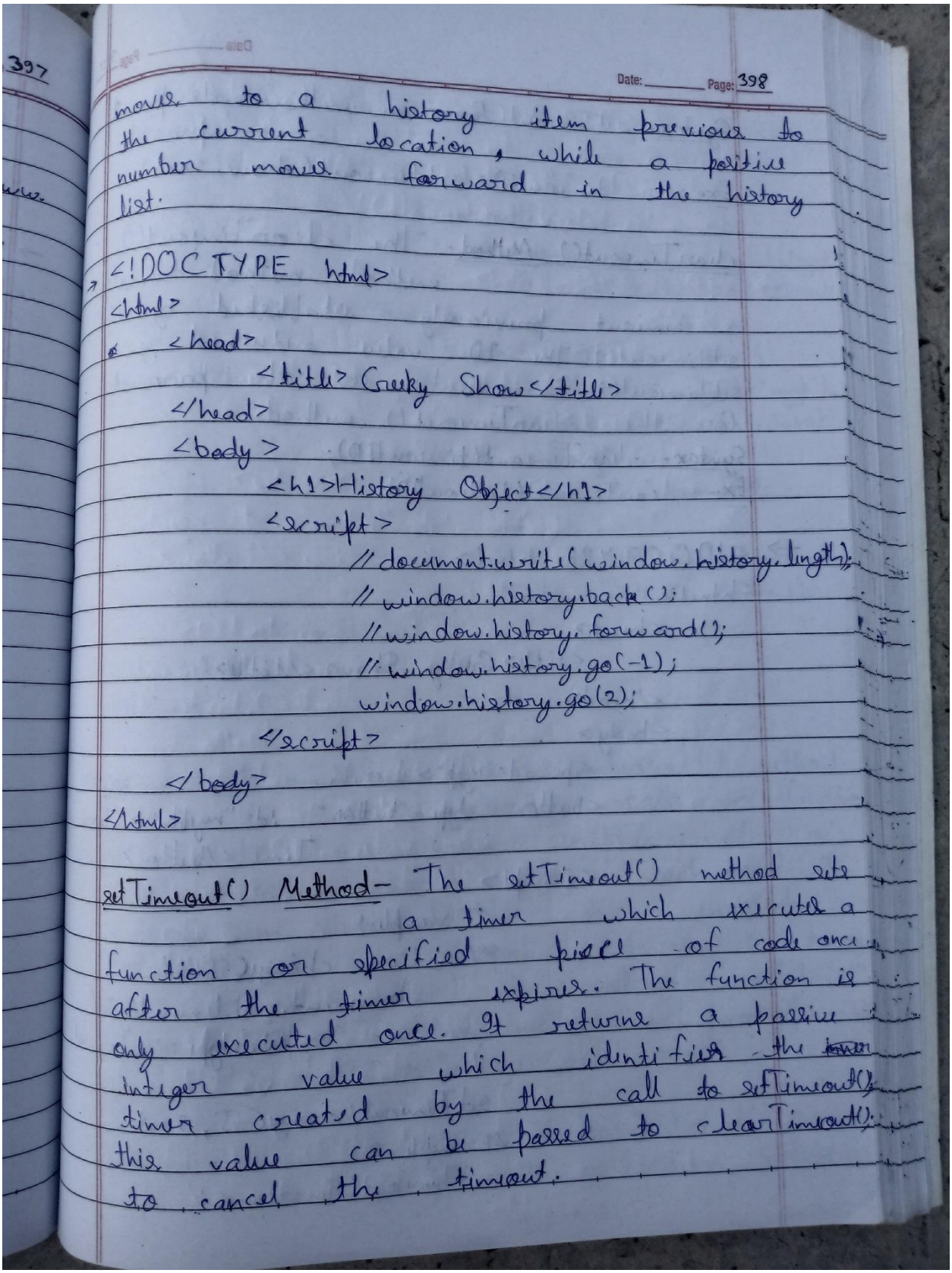
Syntax - window.history.property.
window.history.method

- Property -

- length - It returns the number of URLs in the history list.

- Methods -

- back() - It loads the previous URL in the history list.
- forward() - It loads the next URL in the history list.
- go() - It access a particular item in the history list relative to the current position. Using a negative value



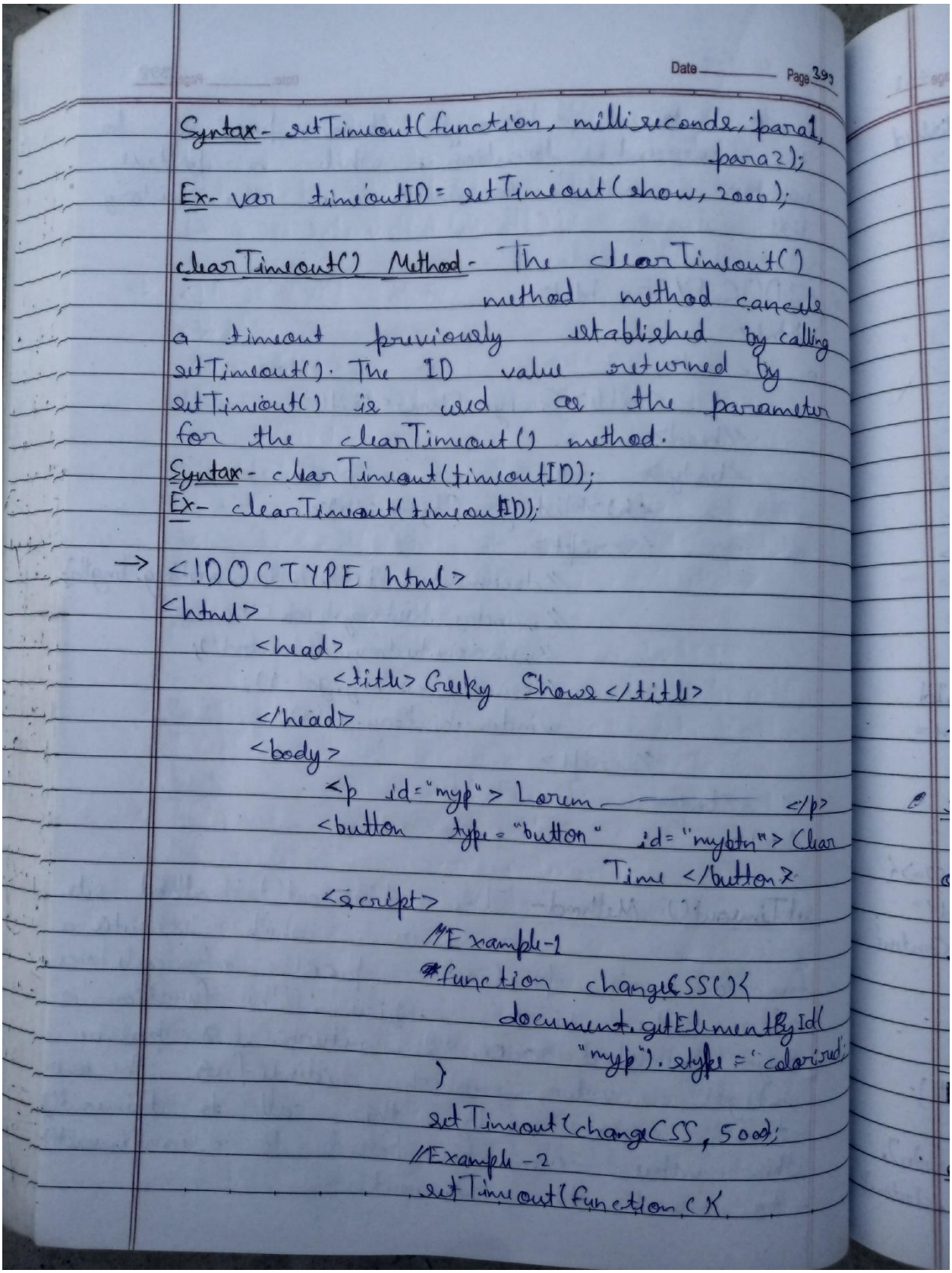
397

Date: Page: 398

moves to a history item previous to the current location, while a positive number moves forward in the history list.

```
<!DOCTYPE html>
<html>
    <head>
        <title> Geeky Shows </title>
    </head>
    <body>
        <h1>History Object</h1>
        <script>
            // document.write(window.history.length);
            // window.history.back();
            // window.history.forward();
            // window.history.go(-1);
            window.history.go(2);
        </script>
    </body>
</html>
```

setTimeout() Method- The setTimeout() method sets a timer which executes a function or specified piece of code once after the timer expires. The function is only executed once. It returns a positive integer value which identifies the timer created by the call to setTimeout(); this value can be passed to clearTimeout() to cancel the timeout.



Date Page 393

Syntax- setTimeout(function, milliseconds, para1, para2);

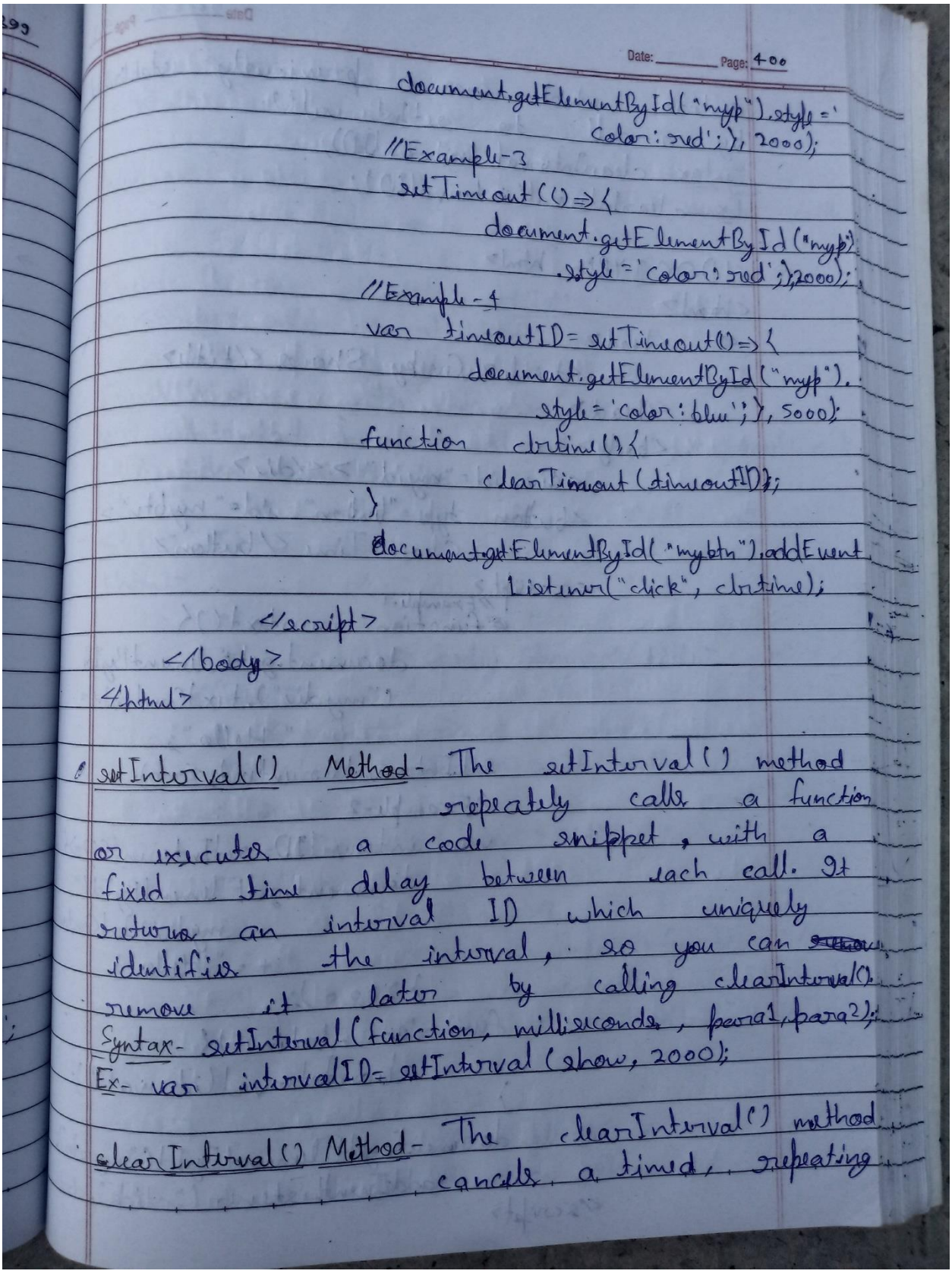
Ex- var timeoutID = setTimeout(show, 2000);

clearTimeout() Method- The clearTimeout() method method cancels a timeout previously established by calling setTimeout(). The ID value returned by setTimeout() is used as the parameter for the clearTimeout() method.

Syntax- clearTimeout(timeoutID);

Ex- clearTimeout(timeoutID);

→
```
<!DOCTYPE html>
<html>
    <head>
        <title> Geeky Shows </title>
    </head>
    <body>
        <p id="myp"> Lorem ---------- </p>
        <button type="button" id="mybtn"> Clear Time </button>
        <script>
            //Example-1
            function changeCSS(){
                document.getElementById("myp").style = 'colorized';
            }
            setTimeout(changeCSS, 5000);
            //Example-2
            setTimeout(function(){
```



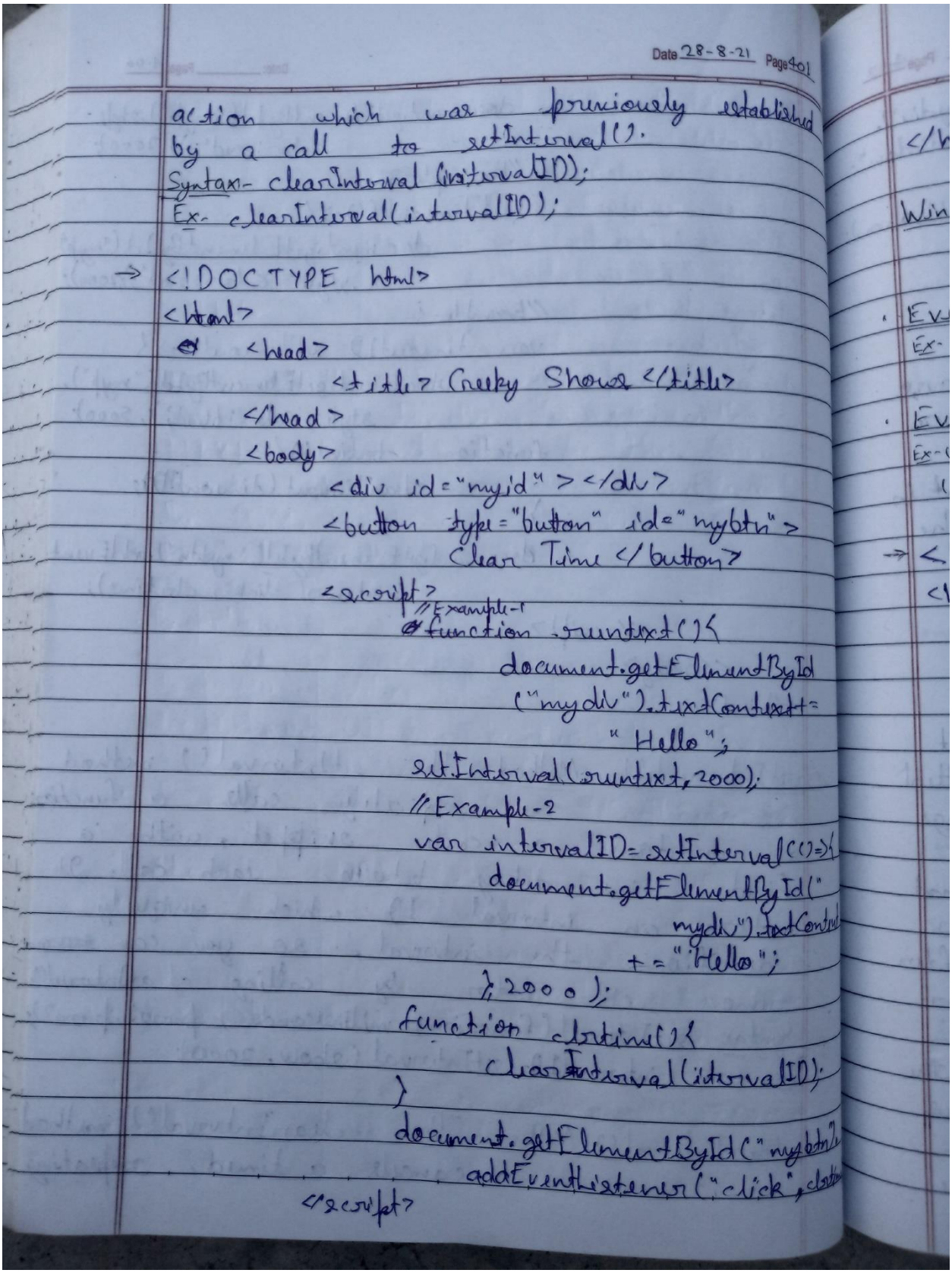
```
Date: Page: 400
document.getElementById("myp").style =
                color: red'; }, 2000);
//Example-3
setTimeout(() => {
        document.getElementById("myp")
        .style = 'color: red'; }, 2000);
//Example-4
var timeoutID = setTimeout(() => {
        document.getElementById("myp").
        style = 'color: blue'; }, 5000);
function clrtime(){
        clearTimeout(timeoutID);
}
document.getElementById("mybtn").addEvent
        Listener("click", clrtime);
</script>
</body>
</html>
```

setInterval() Method - The setInterval() method repeatedly calls a function or executes a code snippet, with a fixed time delay between each call. It returns an interval ID which uniquely identifies the interval, so you can remove it later by calling clearInterval().

Syntax - setInterval(function, milliseconds, para1, para2);

Ex - var intervalID = setInterval(show, 2000);

clearInterval() Method - The clearInterval() method cancels a timed, repeating



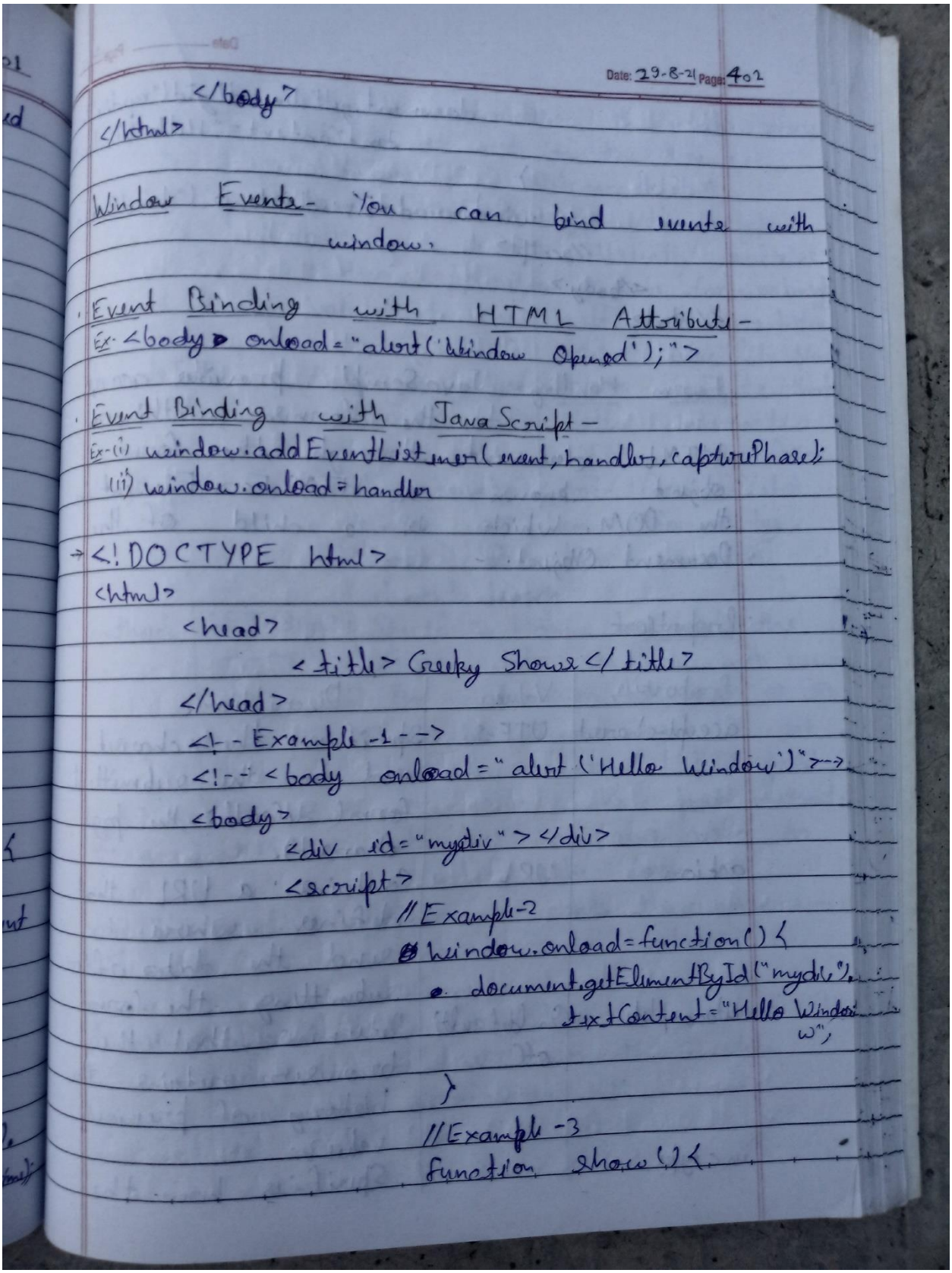
Date 28-8-21 Page to 1

action which was previously established by a call to setInterval().

Syntax- clearInterval(intervalID);

Ex- clearInterval(intervalID);

```
<!DOCTYPE html>
<html>
  <head>
    <title> Geeky Shows </title>
  </head>
  <body>
    <div id="myid" ></div>
    <button type="button" id="mybtn">
      Clear Time </button>
    <script>
      //Example-1
      function runtext(){
        document.getElementById
        ("mydiv").textContext=
          "Hello";
      }
      setInterval(runtext, 2000);
      //Example-2
      var intervalID= setInterval(()=>{
        document.getElementById("
          mydiv").textConte
          += "Hello";
      }, 2000);
      function cleartime(){
        clearInterval(intervalID);
      }
      document.getElementById("mybtn").
        addEventListener("click", cle
    </script>
```



Date: 29-8-21 Page: 402

</body>
</html>

Window Events - You can bind events with window.

1. Event Binding with HTML Attributes -
Ex. <body onload="alert('Window Opened');">

2. Event Binding with JavaScript -
Ex-(i) window.addEventListener(event, handler, capturePhase);
(ii) window.onload = handler

```
<!DOCTYPE html>
<html>
    <head>
        <title> Geeky Shows </title>
    </head>
    <!- Example -1 -->
    <!-- <body onload="alert('Hello Window')"> -->
    <body>
        <div id="mydiv"> </div>
        <script>
            //Example-2
            window.onload=function(){
            document.getElementById("mydiv").
                textContent="Hello Windo
                w";
            }
            //Example -3
            function show(){
```

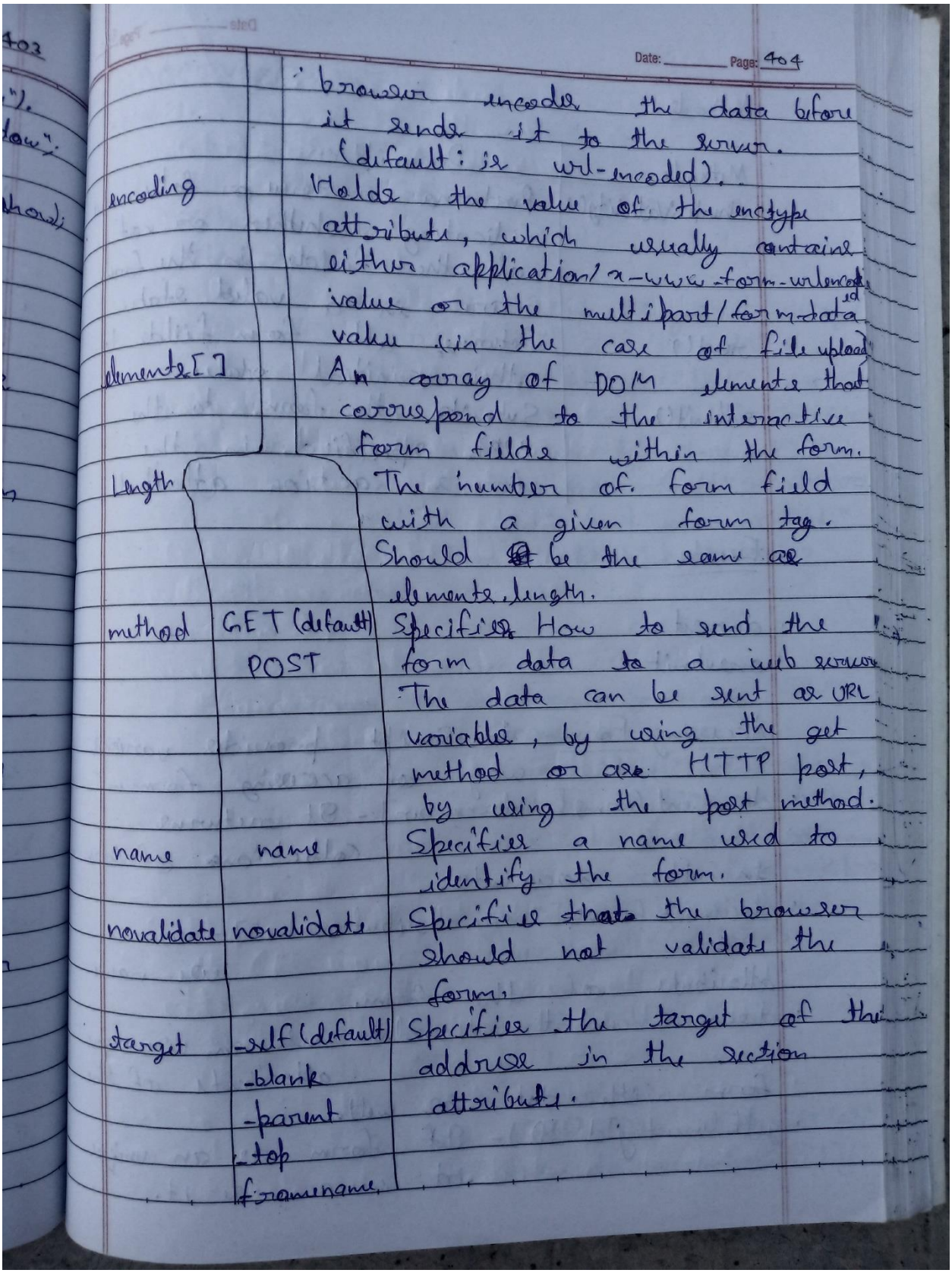
Date Page 103

```
        document.getElementById("mydiv").
            textContent = "Hello Window";
        }
        window.addEventListener("load", sho
    </script>
</body>
</html>
```

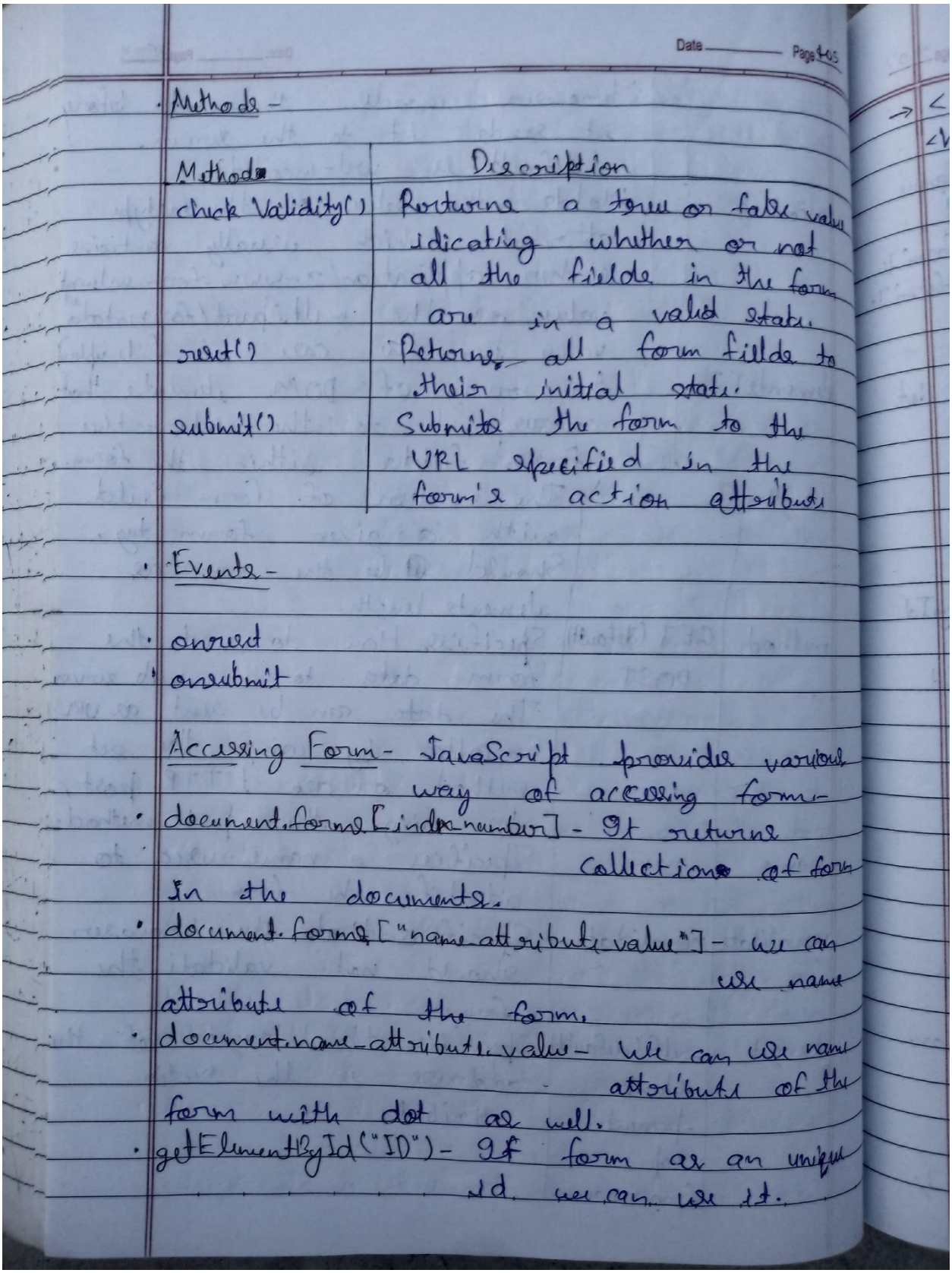
Form Handling - JavaScript provide access to the forms within an HTML document through the form object known as HTMLFormElement in the DOM. which is a child of the Document Object.

Properties -

Properties	Value	Discription.
accept-charset	UTF-8	Specifies the charset used in the submitted form (default: the page charset).
action	URL	Contains a URL that defines where to send the data after submitting the form.
autocomplete	On (default) off	Determine that the browser retains the history of previous values.
enctype		Specifies how the



		browser encodes the data before it sends it to the server. (default: is url-encoded).
encoding		Holds the value of the enctype attribute, which usually contains either application/x-www-form-urlencoded value or the multipart/form-data value in the case of file upload.
elements[]		An array of DOM elements that correspond to the interactive form fields within the form.
length		The number of form field with a given form tag. Should be the same as elements.length.
method	GET (default)	Specifies How to send the
	POST	form data to a web server. The data can be sent as URL variable, by using the get method or as HTTP post, by using the post method.
name	name	Specifies a name used to identify the form.
novalidate	novalidate	Specifies that the browser should not validate the form.
target	_self (default)	Specifies the target of the
	_blank	address in the action
	_parent	attribute.
	_top	
	framename	



Date _____ Page 105

- Methods -

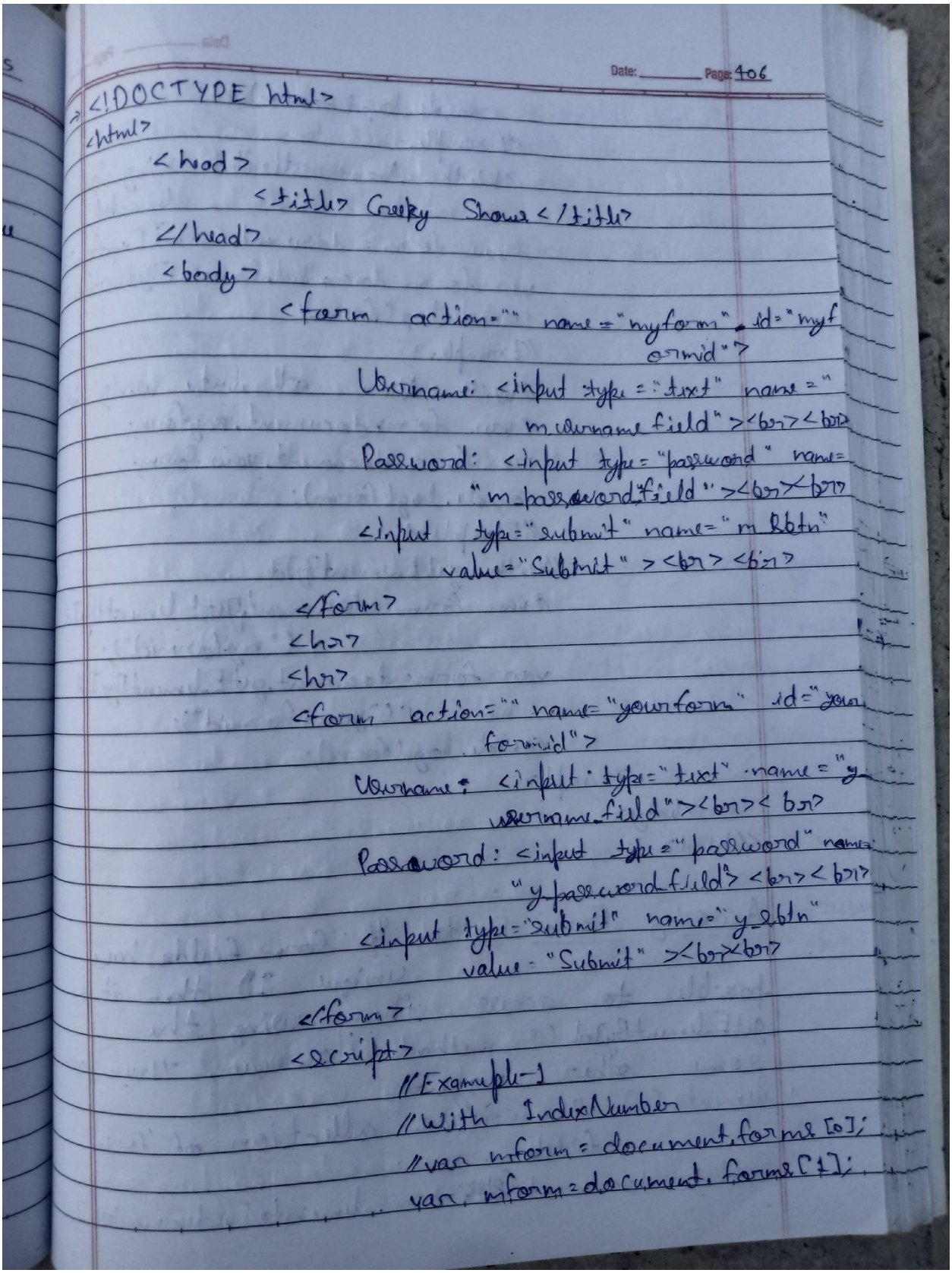
Methods	Description
checkValidity()	Returns a true or false value indicating whether or not all the fields in the form are in a valid state.
reset()	Returns all form fields to their initial state.
submit()	Submits the form to the URL specified in the form's action attribute

- Events -

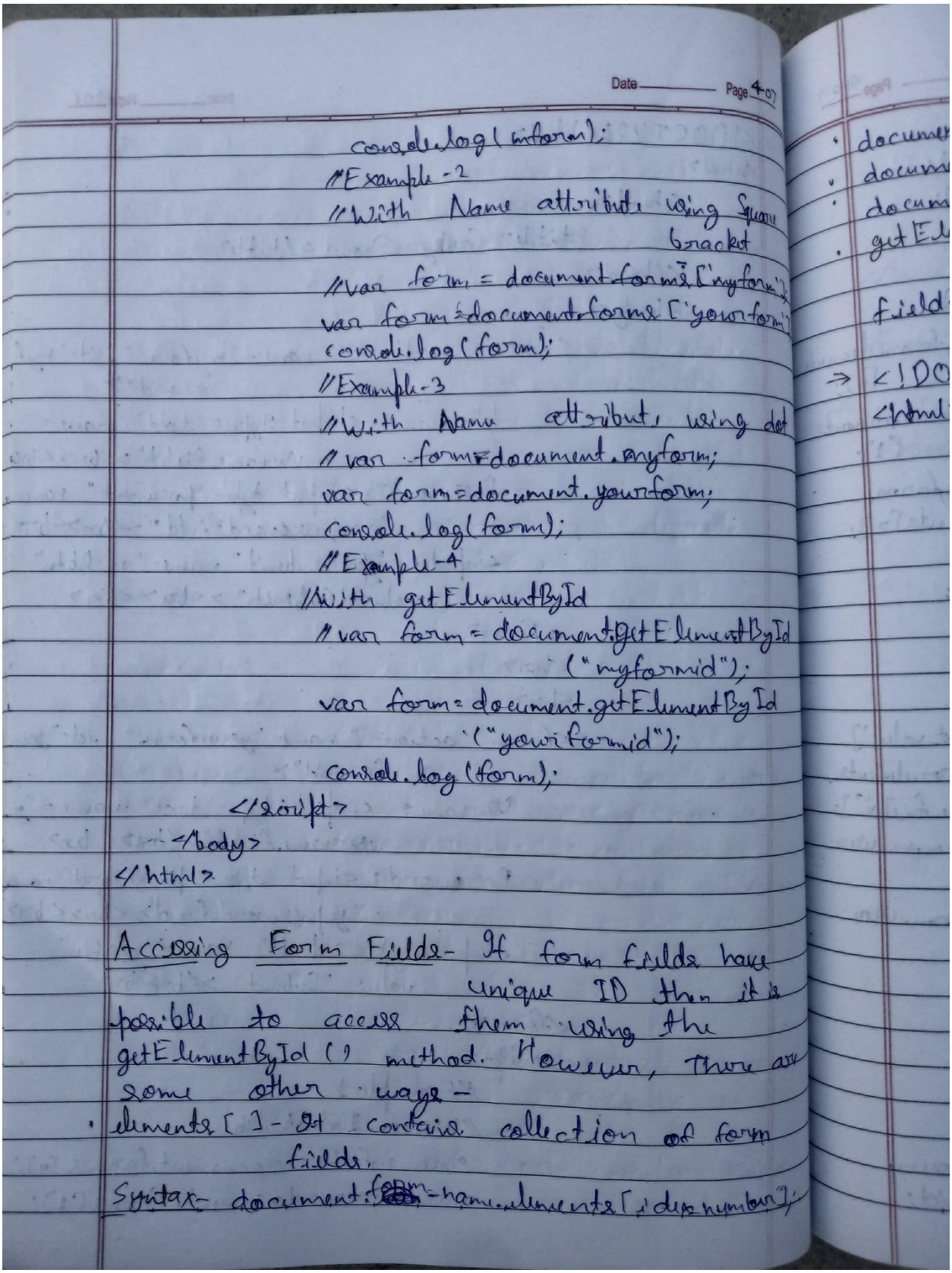
- onreset
- onsubmit

Accessing Form - JavaScript provides various way of accessing form -

- document.forms[index_number] - It returns collections of form in the documents.
- document.forms["name_attribute_value"] - we can use name attribute of the form.
- document.name_attribute.value - We can use name attribute of the form with dot as well.
- getElementById("ID") - If form as an unique id, we can use it.



```
<!DOCTYPE html>
<html>
    <head>
        <title> Geeky Shows </title>
    </head>
    <body>
        <form action="" name="myform" id="myf
                                    ormid">
            Username: <input type="text" name="
                m_username_field"><br><br>
            Password: <input type="password" name=
                "m_password_field"><br><br>
            <input type="submit" name="m_btn"
                value="Submit"><br><br>
        </form>
        <hr>
        <hr>
        <form action="" name="yourform" id="your
                                    formid">
            Username: <input type="text" name="y
                username_field"><br><br>
            Password: <input type="password" name=
                "y_password_field"><br><br>
            <input type="submit" name="y_btn"
                value="Submit"><br><br>
        </form>
        <script>
            //Example-1
            //With IndexNumber
            //var mform = document.forms[0];
            var mform = document.forms[1];
```

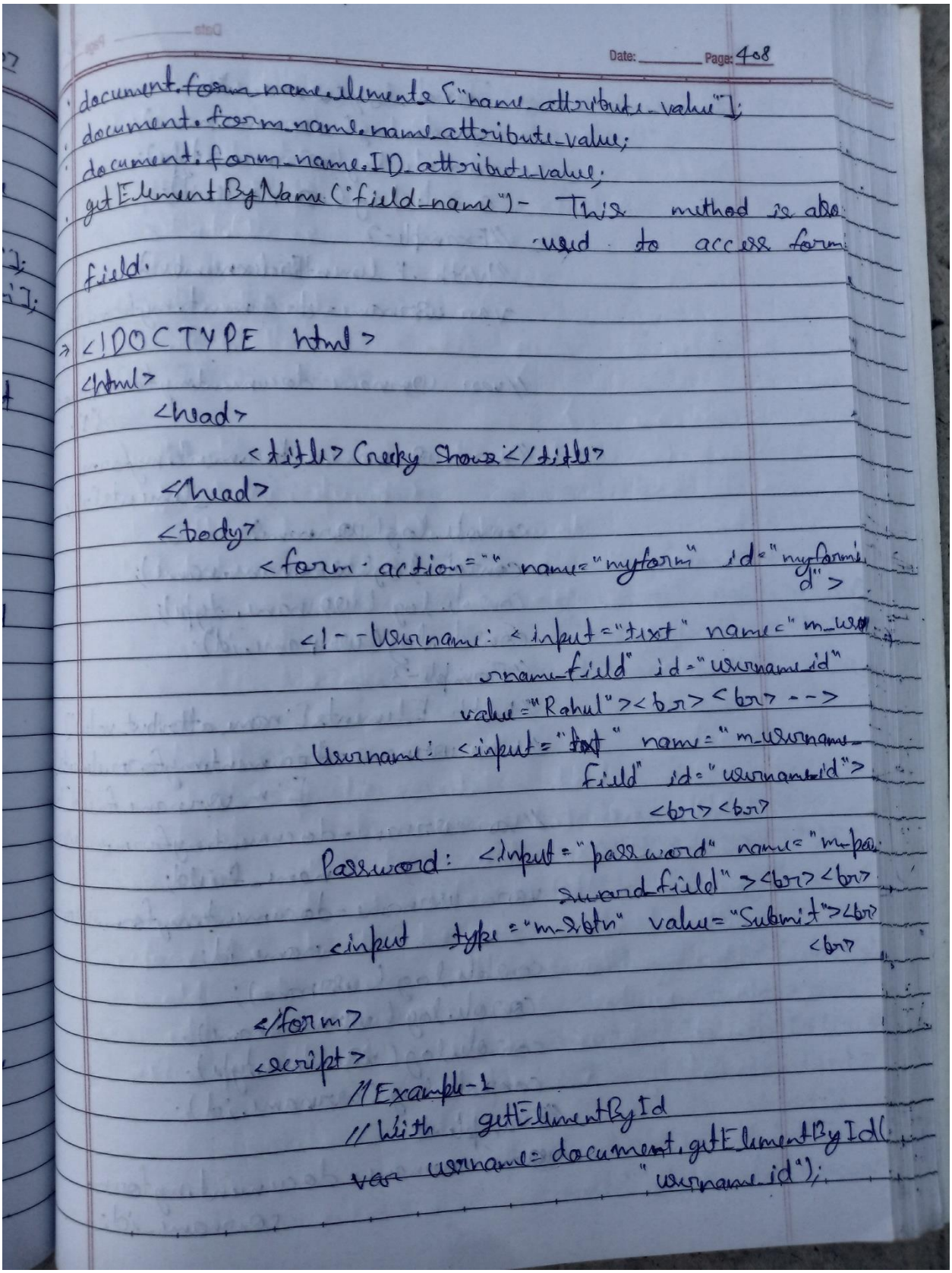


```
console.log(inform);
//Example-2
//With Name attribute using Square
                          bracket
//var form = document.forms['myform'];
var form = document.forms['yourform'];
console.log(form);
//Example-3
//With Name attribute using dot
//var form=document.myform;
var form=document.yourform;
console.log(form);
//Example-4
//With getElementById
//var form = document.getElementById
                    ("myformid");
var form = document.getElementById
                    ("yourformid");
console.log(form);
</script>
</body>
</html>
```

Accessing Form Fields- If form fields have unique ID then it is possible to access them using the getElementById() method. However, There are some other ways -

elements[] - It contains collection of form fields.

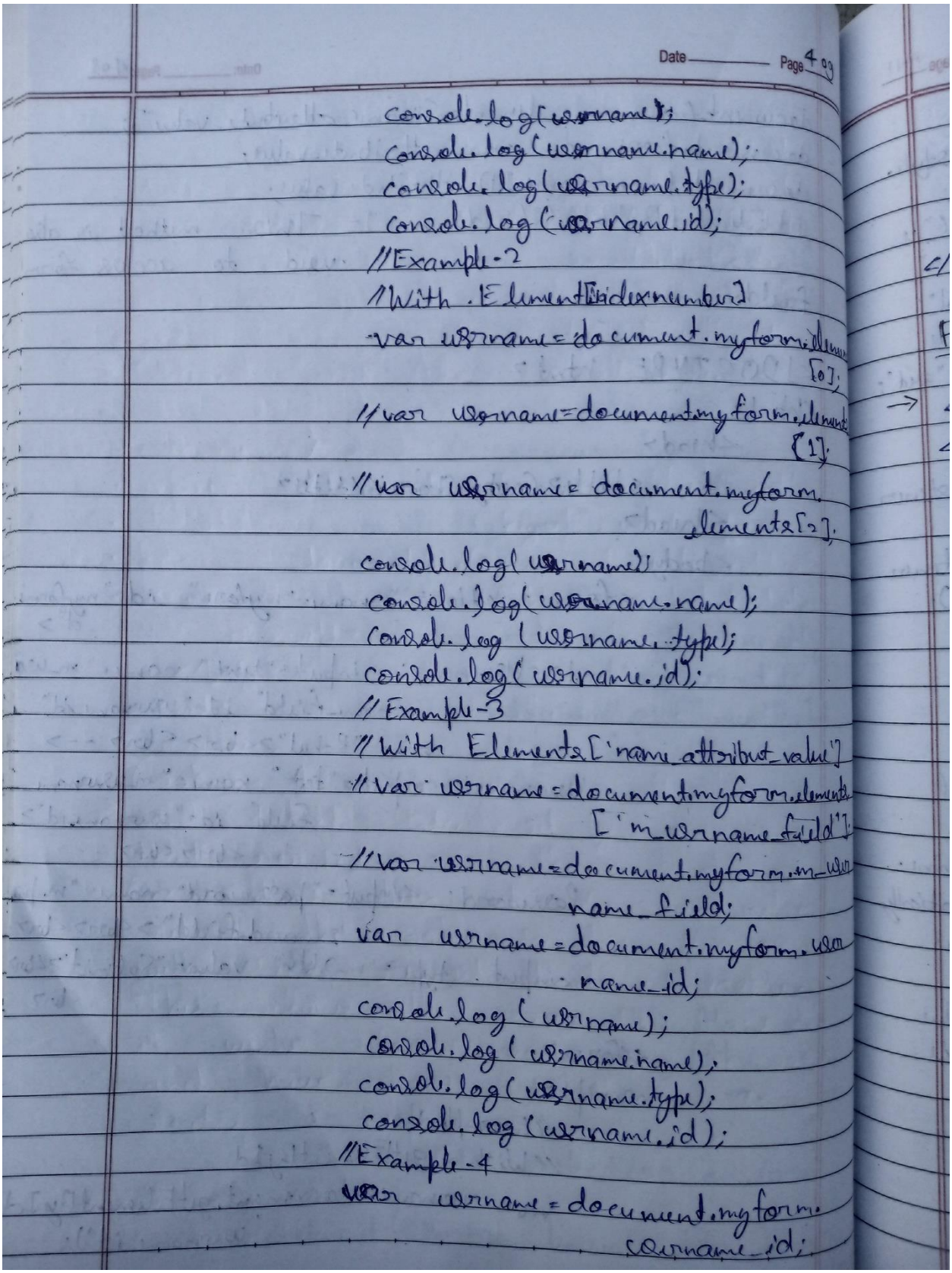
Syntax- document.form-name.elements[index number];



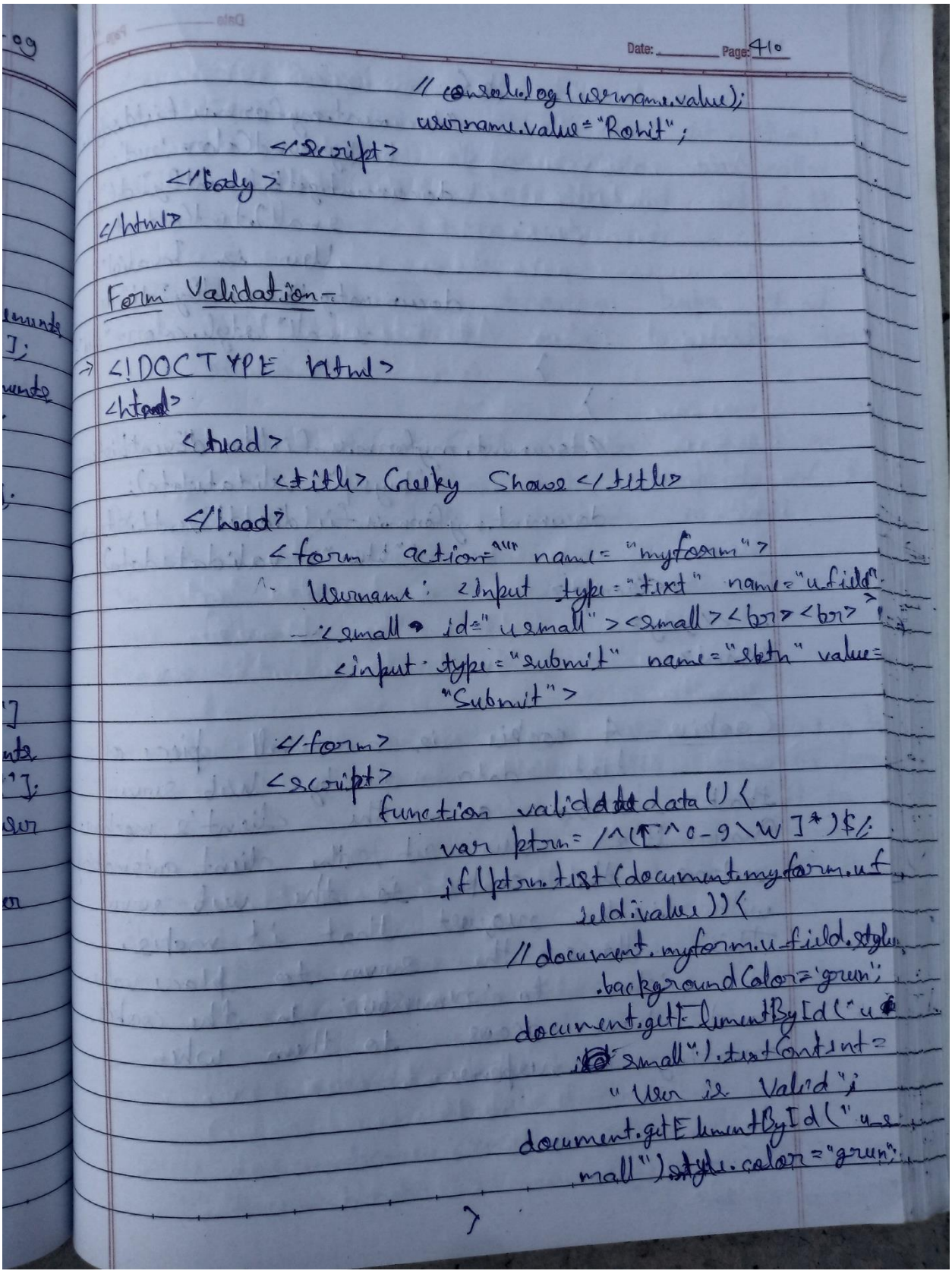
Date: Page: 408

document.form_name.elements["name_attribute_value"];
document.form_name.name_attribute_value;
document.form_name.ID_attribute_value;
getElementByName("field_name")- This method is also used to access form field.

```
<!DOCTYPE html>
<html>
    <head>
        <title> Cricky Shows </title>
    </head>
    <body>
        <form action="" name="myform" id="myformid">
            <!--Username: <input="text" name="m_username_field" id="username_id"
            value="Rahul"><br><br> --->
            Username: <input="text" name="m_username_field" id="username_id">
            <br><br>
            Password: <input="password" name="m_password_field"><br><br>
            <input type="m_btn" value="Submit"><br>
            <br>
        </form>
        <script>
            //Example-1
            //With getElementById
            var username=document.getElementById("username_id");
```



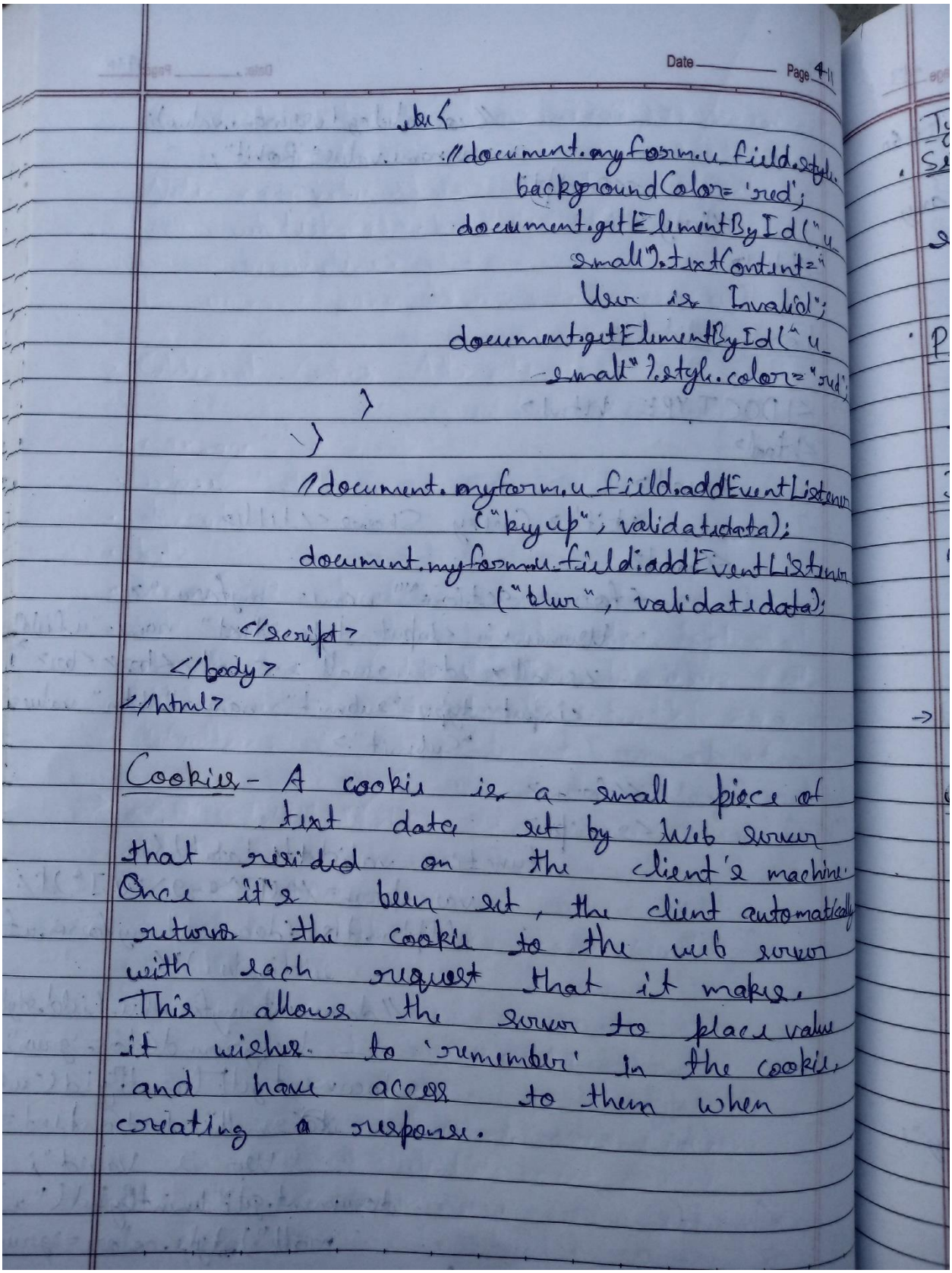
```
console.log(username);
console.log(username.name);
console.log(username.type);
console.log(username.id);
//Example-2
//With .Element[index number]
var username = document.myform.elements
                                [0];
//var username=document.myform.elements
                                [1];
//var username = document.myform.
                        elements[2];
console.log(username);
console.log(username.name);
console.log(username.type);
console.log(username.id);
//Example-3
//With Elements['name attribut value']
//var username = document.myform.elements
                ['m_username_field'];
//var username=document.myform.m_user
                    name_field;
var username=document.myform.user
                    name_id;
console.log(username);
console.log(username.name);
console.log(username.type);
console.log(username.id);
//Example-4
var username = document.myform.
                        username_id;
```



```
// console.log(username.value);
username.value = "Rohit";
</script>
</body>
</html>
```

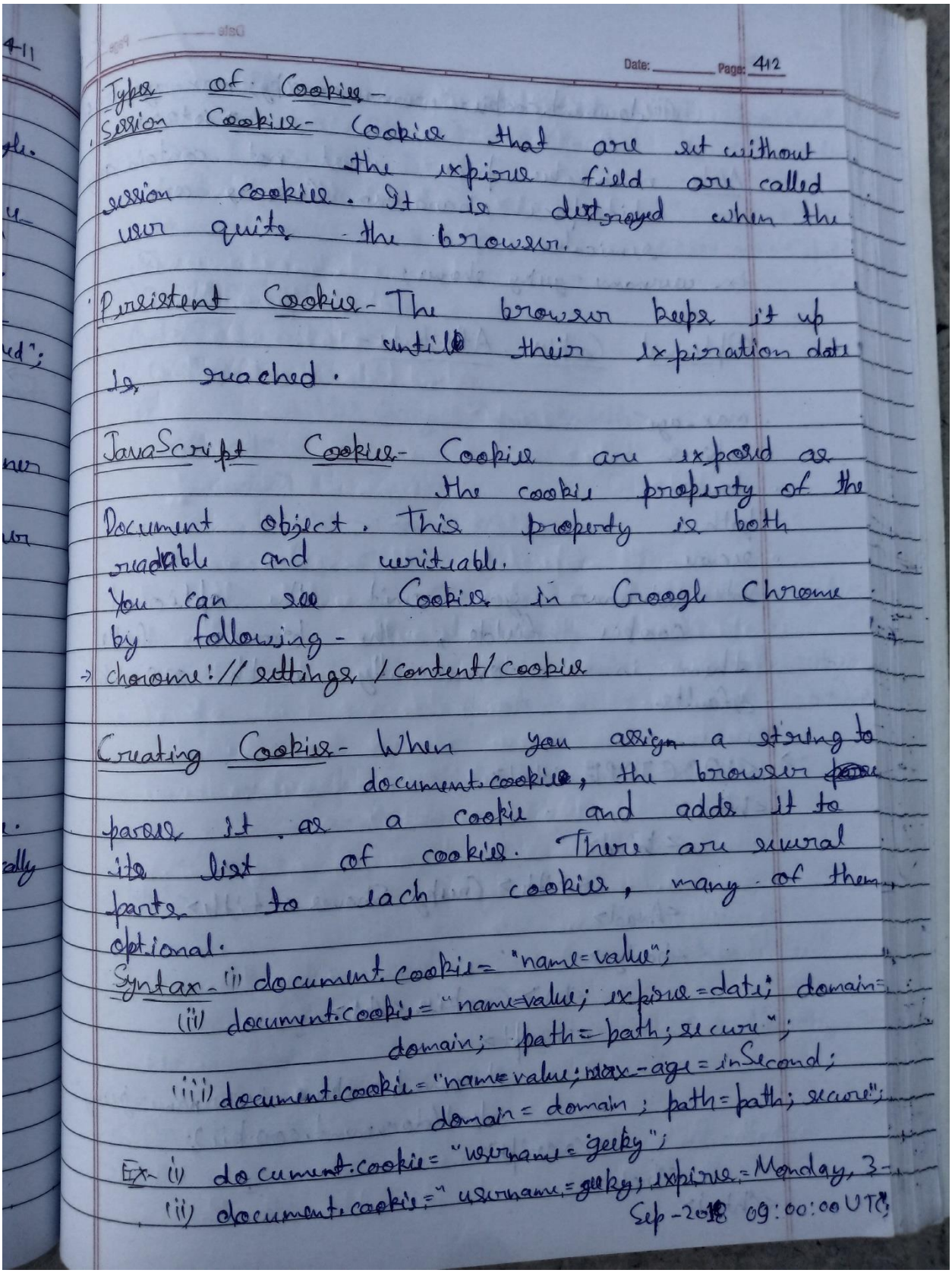
Form Validation:-

```
<!DOCTYPE html>
<html>
  <head>
    <title> Geeky Shows </title>
  </head>
  <form action="" name="myform">
    Username: <input type="text" name="u_field">
    <small id="u_small"><small><br><br>
    <input type="submit" name="sbtn" value=
      "Submit">
  </form>
  <script>
    function validdddddata(){
      var ptrn = /^([^0-9\W]*)$/;
      if(ptrn.test(document.myform.u_
        field.value)){
        // document.myform.u_field.style
        .backgroundColor='green';
        document.getElementById("u_
        small").textContent=
        "User is Valid";
        document.getElementById("u_s
        mall").style.color="green";
      }
```



```
        else{
            //document.myform.u_field.style.
                backgroundColor='red';
            document.getElementById("u_
                small").textContent="
                User is Invalid";
            document.getElementById("u_
                small").style.color="red";
        }
    }

    //document.myform.u_field.addEventListener
            ("keyup", validatedata);
    document.myform.u_field.addEventListener
            ("blur", validatedata);
</script>
</body>
</html>
```

Cookies - A cookie is a small piece of text data set by Web server that resided on the client's machine. Once it's been set, the client automatically returns the cookie to the web server with each request that it makes. This allows the server to place value it wishes to 'remember' in the cookie, and have access to them when creating a response.



Date: Page: 412

Types of Cookies -

Session Cookies - Cookies that are set without the expires field are called session cookies. It is destroyed when the user quits the browser.

Persistent Cookies - The browser keeps it up until their expiration date is reached.

JavaScript Cookies - Cookies are exposed as the cookie property of the Document object. This property is both readable and writable.

You can see Cookies in Google Chrome by following -

→ chrome://settings/content/cookies

Creating Cookies - When you assign a string to document.cookie, the browser parses it as a cookie and adds it to its list of cookies. There are several parts to each cookie, many of them optional.

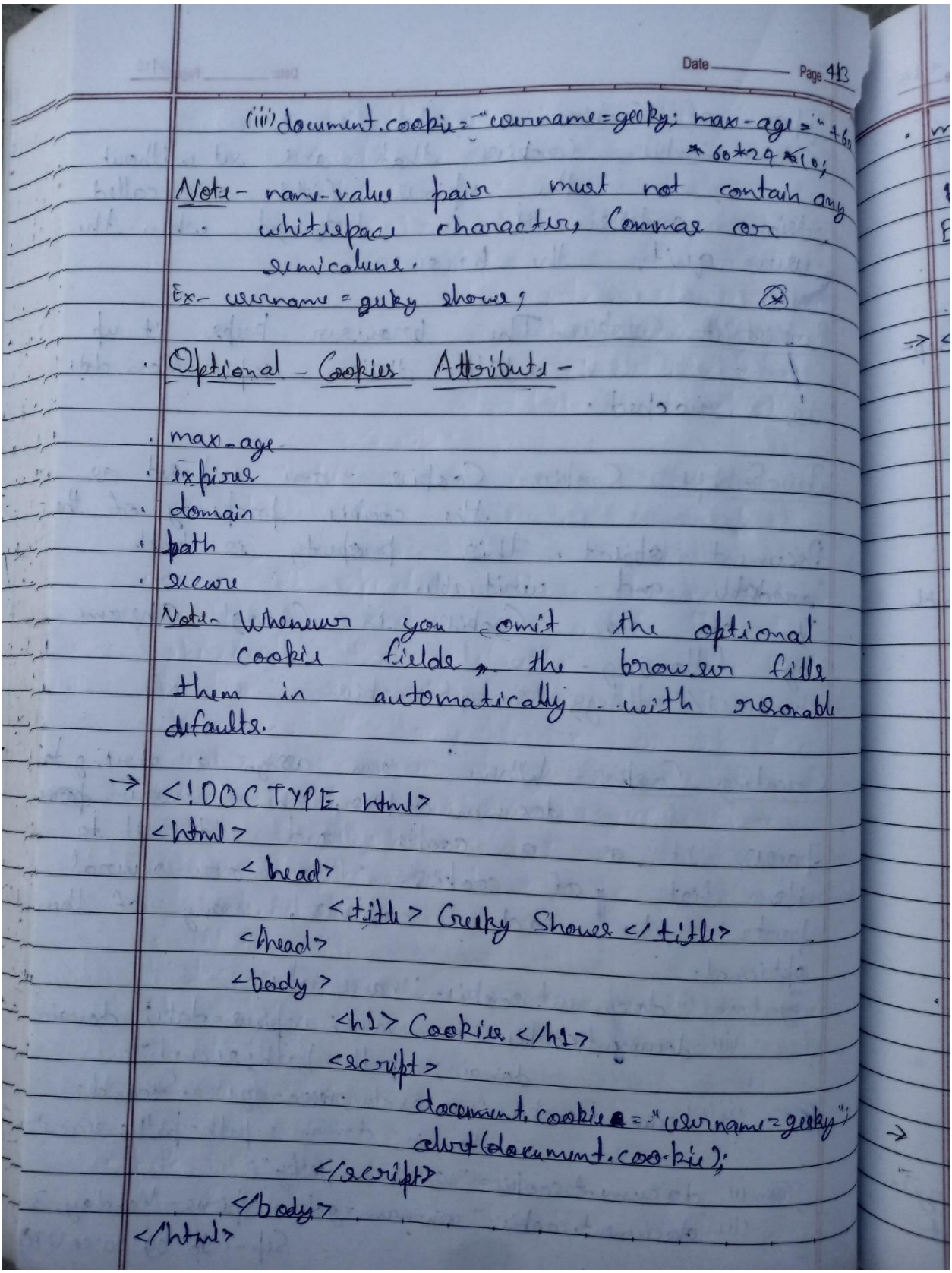
Syntax - (i) document.cookie = "name=value";

(ii) document.cookie = "name=value; expires=date; domain=domain; path=path; secure";

(iii) document.cookie = "name=value; max-age=inSecond; domain=domain; path=path; secure";

Ex - (i) document.cookie = "username=geeky";

(ii) document.cookie = "username=geeky; expires=Monday, 3-Sep-2018 09:00:00 UTC";



(iii) document.cookie = "username=geeky; max-age=" + 60*60*24*10;

Note - name-value pair must not contain any whitespace character, Commas or semicolons.

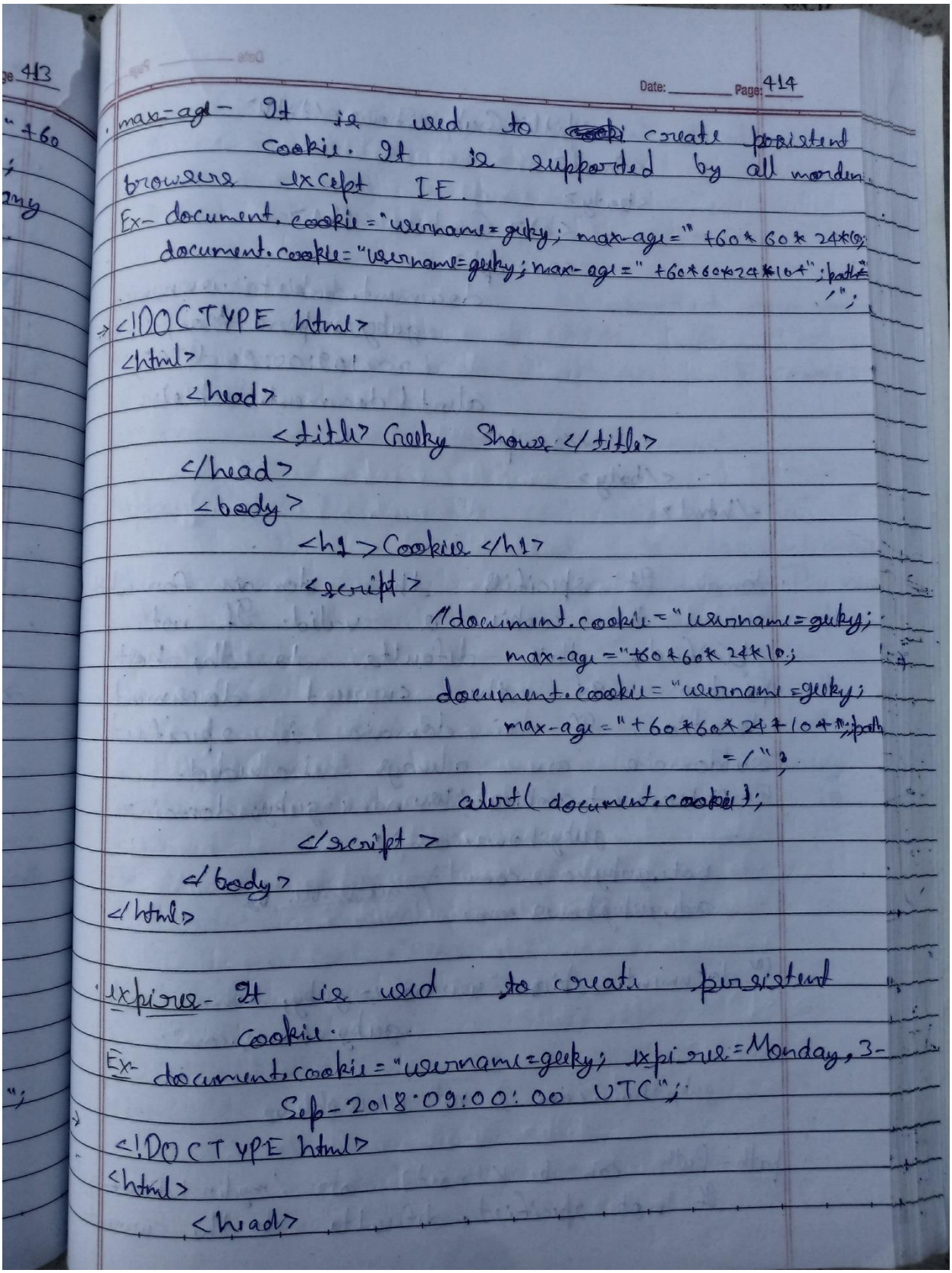
Ex - username = geeky shows; ✗

Optional - Cookies Attributs -

- max-age
- expires
- domain
- path
- secure

Note - Whenever you omit the optional cookie fields, the browser fills them in automatically with reasonable defaults.

```
<!DOCTYPE html>
<html>
    <head>
        <title> Geeky Shows </title>
    </head>
    <body>
        <h1> Cookie </h1>
        <script>
            document.cookie = "username=geeky";
            alert(document.cookie);
        </script>
    </body>
</html>
```



Date: Page: 414

- max-age - It is used to create persistent cookie. It is supported by all modern browsers except IE.

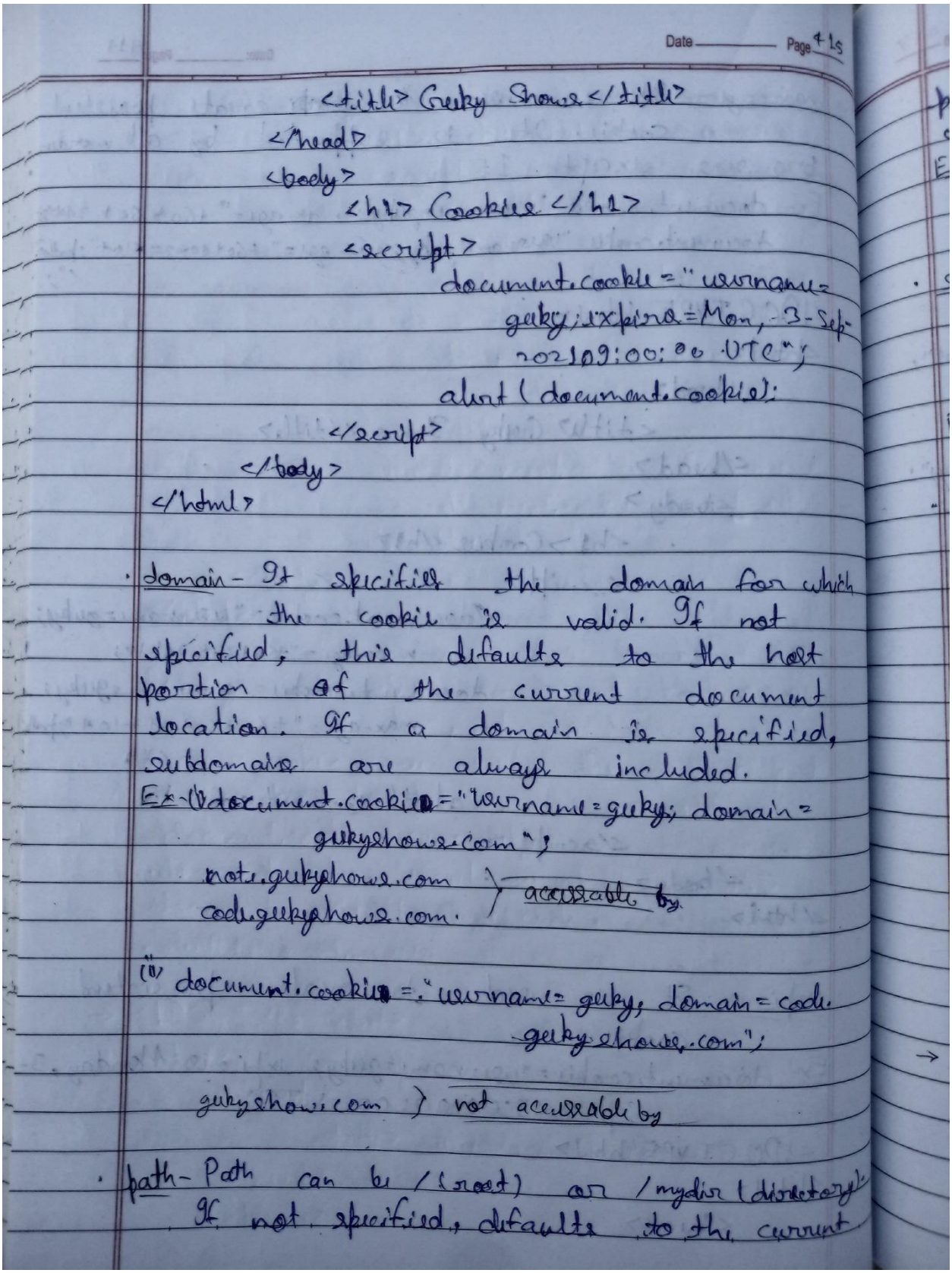
Ex- document.cookie = "username=geky; max-age=" +60*60*24*10;
document.cookie = "username=geky; max-age=" +60*60*24*10+"; path=/";

```
<!DOCTYPE html>
<html>
    <head>
        <title> Geeky Shows </title>
    </head>
    <body>
        <h1> Cookie </h1>
        <script>
            //document.cookie = "username=geky;
                max-age="+60*60*24*10;
            document.cookie = "username=geky;
                max-age="+60*60*24*10+";path
                =/";
            alert(document.cookie);
        </script>
    </body>
</html>
```

- expires - It is used to create persistent cookie.

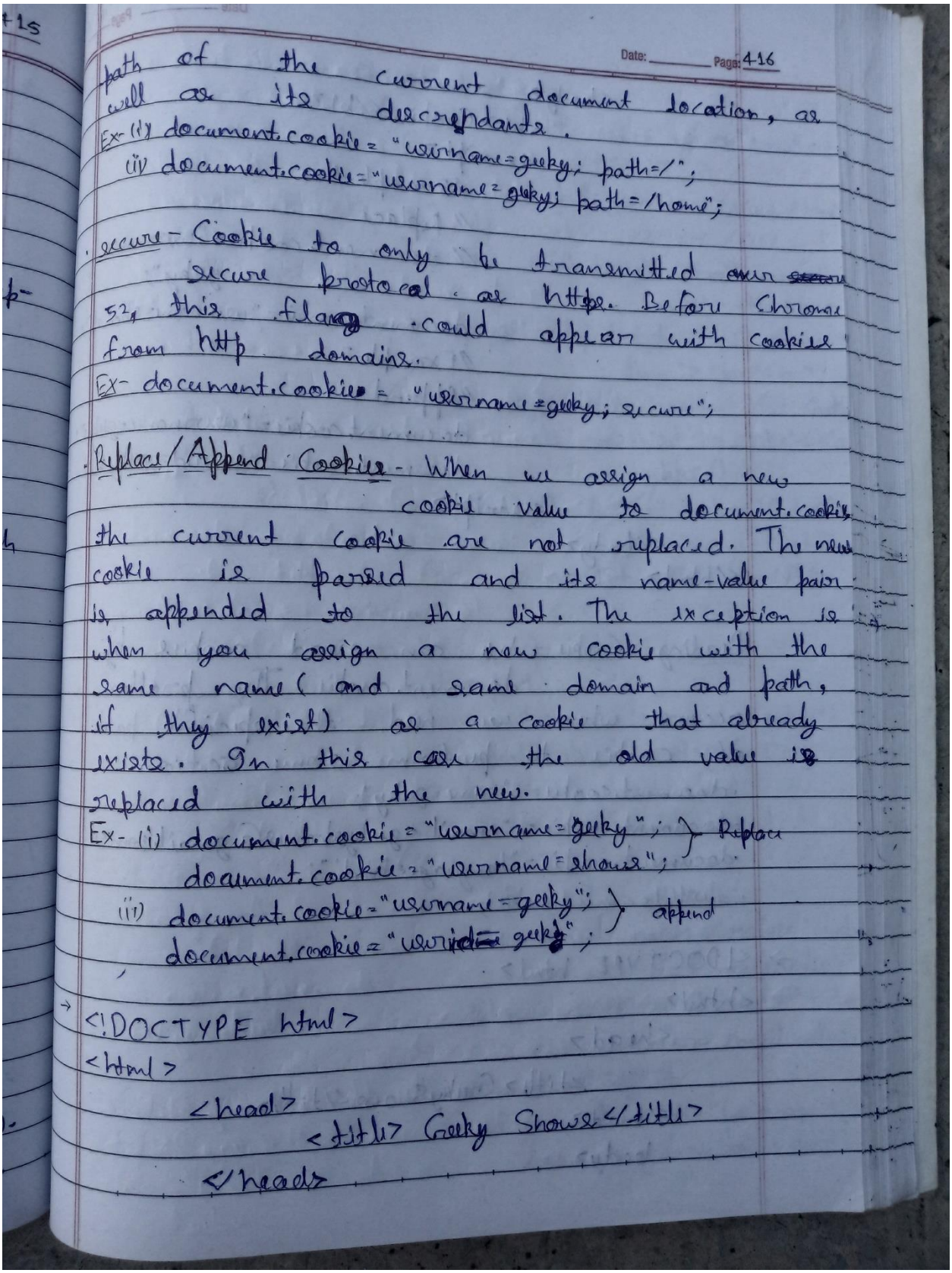
Ex- document.cookie = "username=geeky; expires=Monday, 3-Sep-2018 09:00:00 UTC";

```
<!DOCTYPE html>
<html>
    <head>
```



```
    <title> Geeky Shows </title>
</head>
<body>
    <h1> Cookies </h1>
    <script>
        document.cookie = "username=
            geeky; expires=Mon, 3-Sep-
            2021 09:00:00 UTC";
        alert (document.cookie);
    </script>
</body>
</html>
```

- domain - It specifies the domain for which the cookie is valid. If not specified, this defaults to the host portion of the current document location. If a domain is specified, subdomains are always included.
  Ex- document.cookie = "username=geeky; domain=geekyshows.com";
  note.geekyshows.com, code.geekyshows.com } accessable by
  (i) document.cookie = "username=geeky, domain=code.geekyshows.com";
  geekyshows.com } not accessable by
- path - Path can be / (root) or /mydir (directory). If not specified, defaults to the current

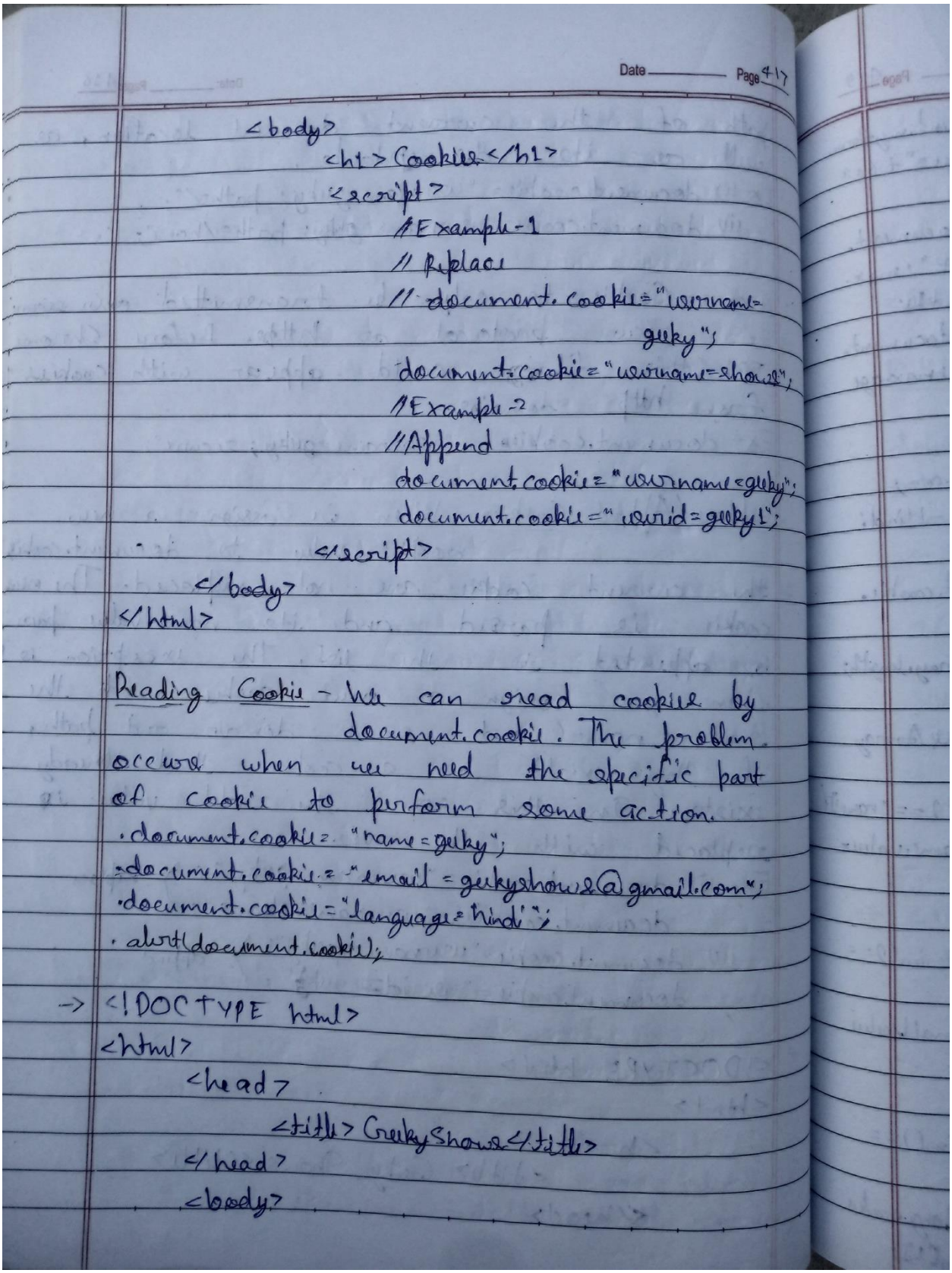


path of the current document location, as well as its descendants.
Ex- (i) document.cookie = "username=geeky; path=/";
(ii) document.cookie = "username=geeky; path=/home";

- secure - Cookie to only be transmitted over secure protocol as https. Before Chrome 52, this flag could appear with cookies from http domains.
Ex- document.cookie = "username=geeky; secure";

- Replace/Append Cookies - When we assign a new cookie value to document.cookie, the current cookie are not replaced. The new cookie is parsed and its name-value pair is appended to the list. The exception is when you assign a new cookie with the same name (and same domain and path, if they exist) as a cookie that already exists. In this case the old value is replaced with the new.
Ex- (i) document.cookie = "username=geeky"; } Replace
document.cookie = "username=shows";
(ii) document.cookie = "username=geeky"; } append
document.cookie = "userid=geek";

```
<!DOCTYPE html>
<html>
    <head>
        <title> Geeky Shows </title>
    </head>
```



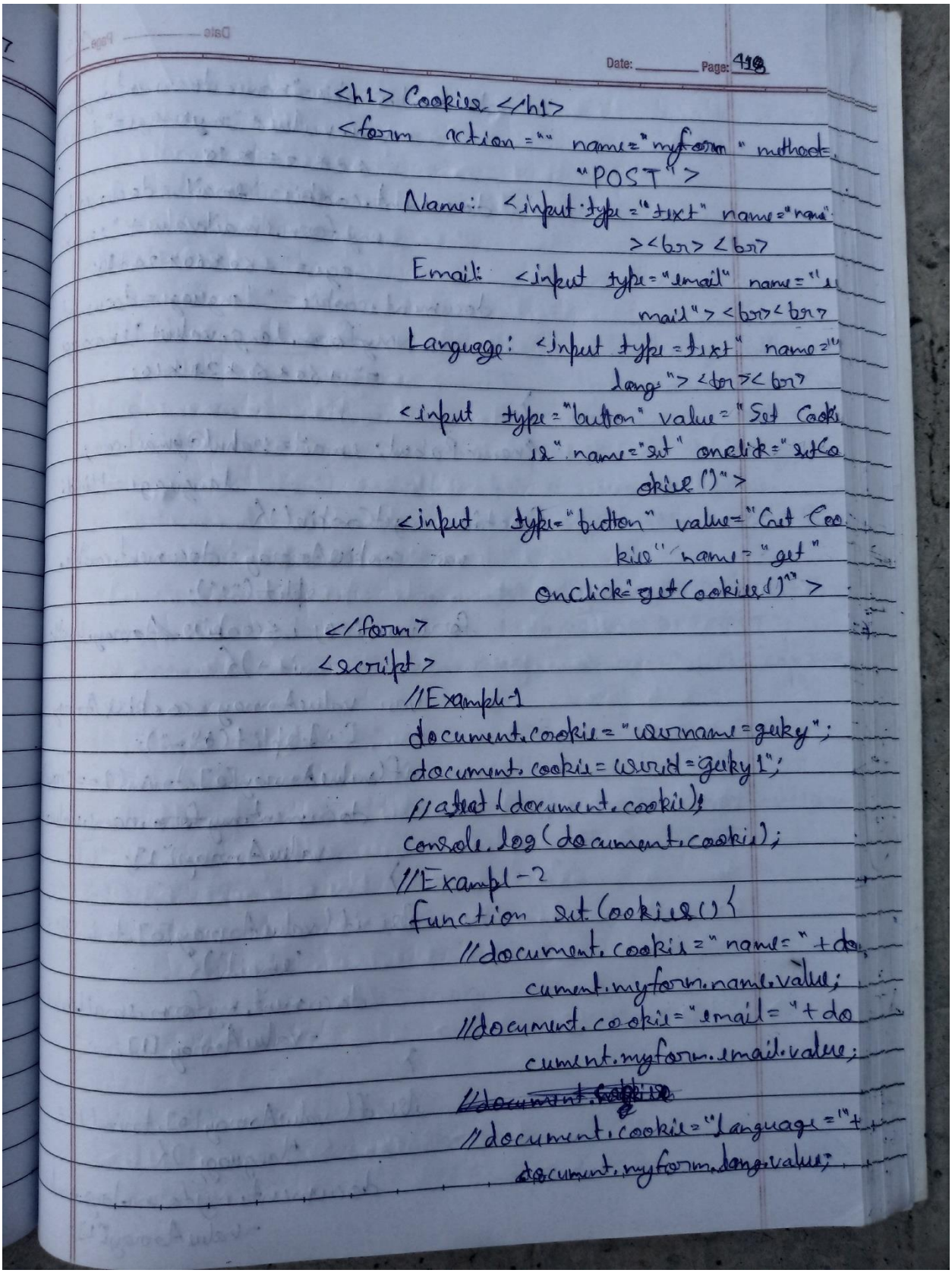
```
<body>
    <h1> Cookies </h1>
    <script>
        //Example-1
        // Replace
        // document.cookie = "username=
                                geky";
        document.cookie = "username=shows";
        //Example-2
        //Append
        document.cookie = "username=geky";
        document.cookie = "userid=geky1";
    </script>
</body>
</html>
```

Reading Cookie - We can read cookies by document.cookie. The problem occurs when we need the specific part of cookie to perform some action.

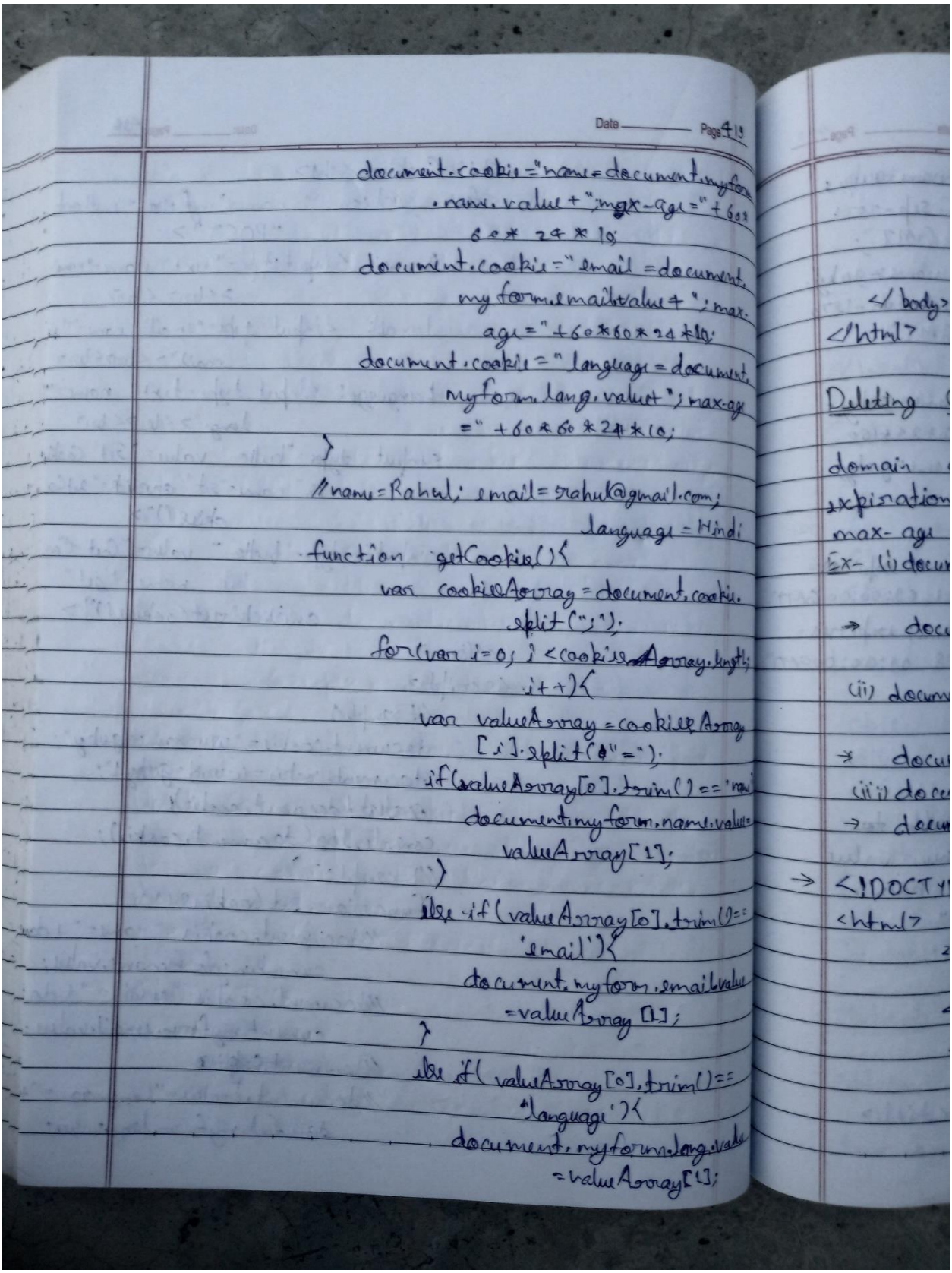
```
document.cookie = "name=geky";
document.cookie = "email=gekyshows@gmail.com";
document.cookie = "language=hindi";
alert(document.cookie);
```

→

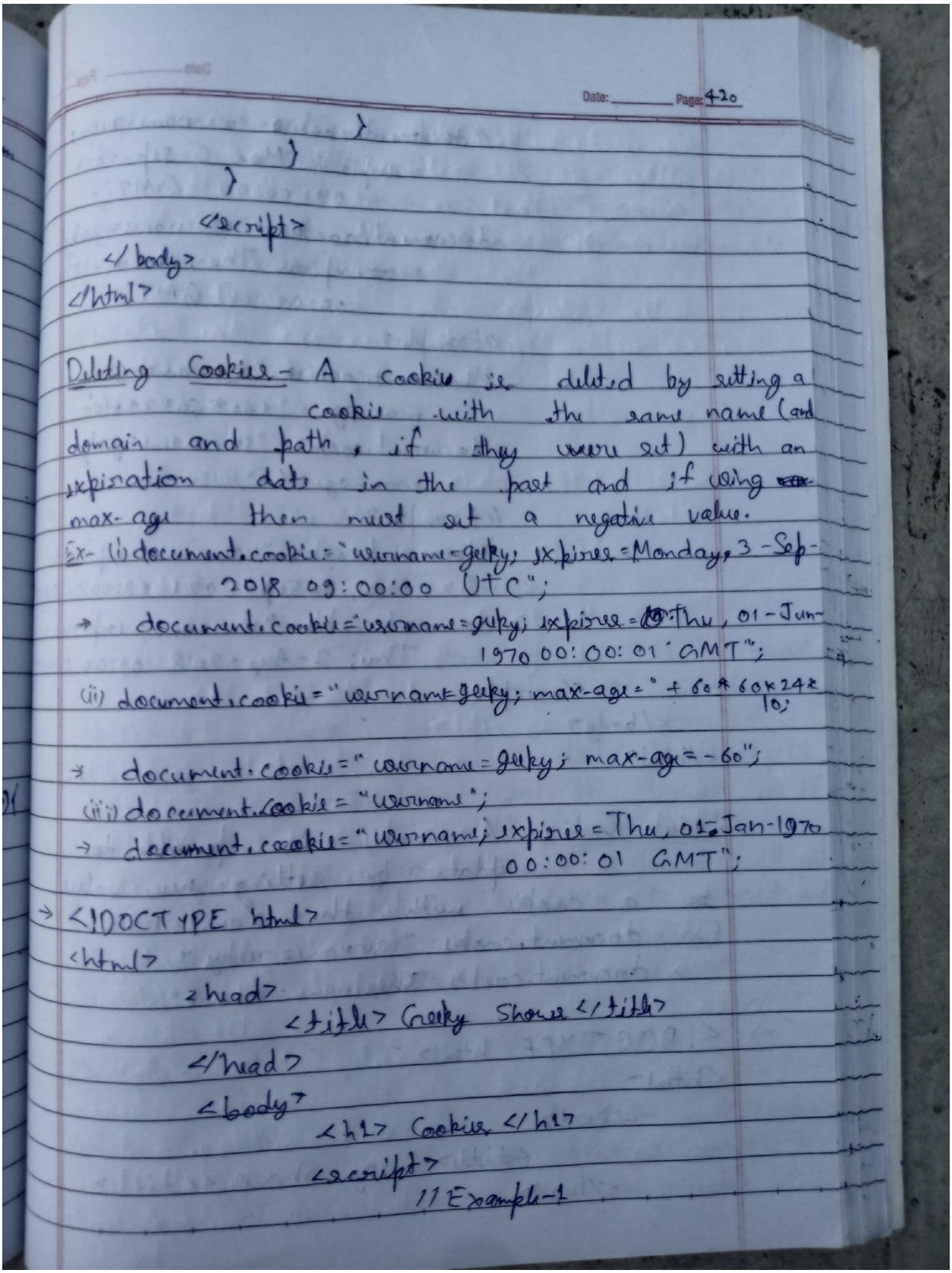
```
<!DOCTYPE html>
<html>
    <head>
        <title> GekyShows </title>
    </head>
    <body>
```



```
<h1> Cookies </h1>
<form action="" name="myform" method=
                "POST">
Name: <input type="text" name="name"
                ><br><br>
Email: <input type="email" name="e
                mail"><br><br>
Language: <input type="text" name="l
                ang"><br><br>
<input type="button" value="Set Cooki
                e" name="set" onclick="setCo
                okie()">
<input type="button" value="Get Coo
                kie" name="get"
                onclick="getCookie()">
</form>
<script>
    //Example-1
    document.cookie="username=geky";
    document.cookie="userid=geky1";
    //alert(document.cookie);
    console.log(document.cookie);
    //Example-2
    function setCookie(){
        //document.cookie="name="+do
            cument.myform.name.value;
        //document.cookie="email="+do
            cument.myform.email.value;
        //document.cookie
        //document.cookie="language="+
            document.myform.lang.value;
```



```
document.cookie="name=document.myform
.name.value+";max-age="+60*
60*24*10;
document.cookie="email=document.
myform.email.value+";max-
age="+60*60*24*10;
document.cookie="language=document.
myform.lang.value";max-age
="+60*60*24*10;
}
//name=Rahul; email=rahul@gmail.com;
                language=Hindi
function getCookie(){
var cookieArray=document.cookie.
split(";");
for(var i=0; i<cookieArray.length;
i++){
var valueArray=cookieArray
[i].split("=");
if(valueArray[0].trim()=='na
document.myform.name.value=
valueArray[1];
}
else if(valueArray[0].trim()==
'email'){
document.myform.email.value
=valueArray[1];
}
else if(valueArray[0].trim()==
'language'){
document.myform.lang.value
=valueArray[1];
```



Date: Page: 420

```
    }
  }
</script>
</body>
</html>
```

Deleting Cookies - A cookie is deleted by setting a cookie with the same name (and domain and path, if they were set) with an expiration date in the past and if using max-age then must set a negative value.

Ex- (i) document.cookie = "username=geky; expires=Monday, 3-Sep-2018 09:00:00 UTC";

→ document.cookie = "username=geky; expires=Thu, 01-Jun-1970 00:00:01 GMT";

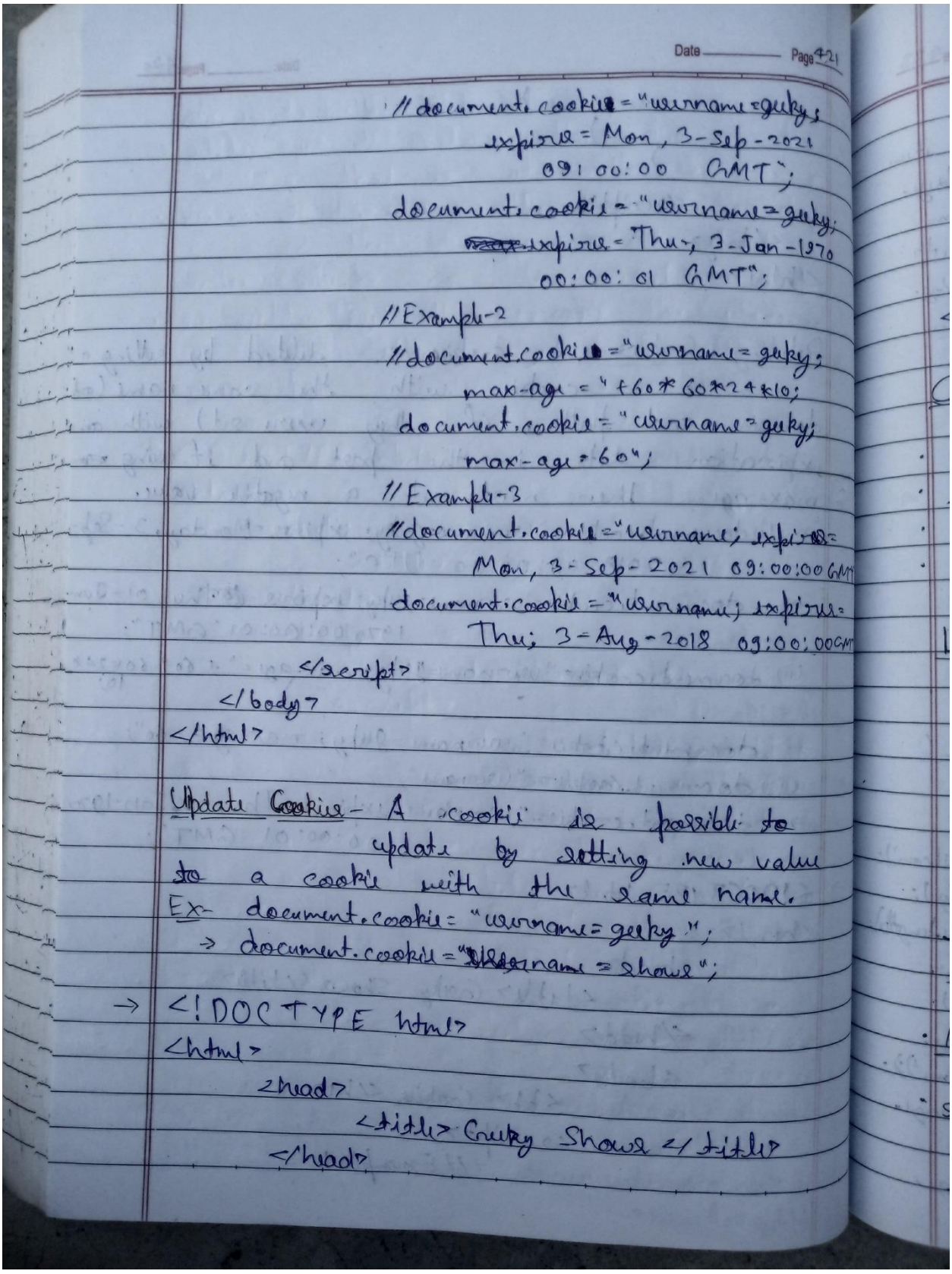
(ii) document.cookie = "username=geky; max-age=" + 60*60*24*10;

→ document.cookie = "username=geky; max-age=-60";

(iii) document.cookie = "username";

→ document.cookie = "username; expires=Thu, 01-Jan-1970 00:00:01 GMT";

```
→ <!DOCTYPE html>
<html>
  <head>
    <title> Geeky Shows </title>
  </head>
  <body>
    <h1> Cookies </h1>
    <script>
      // Example-1
```

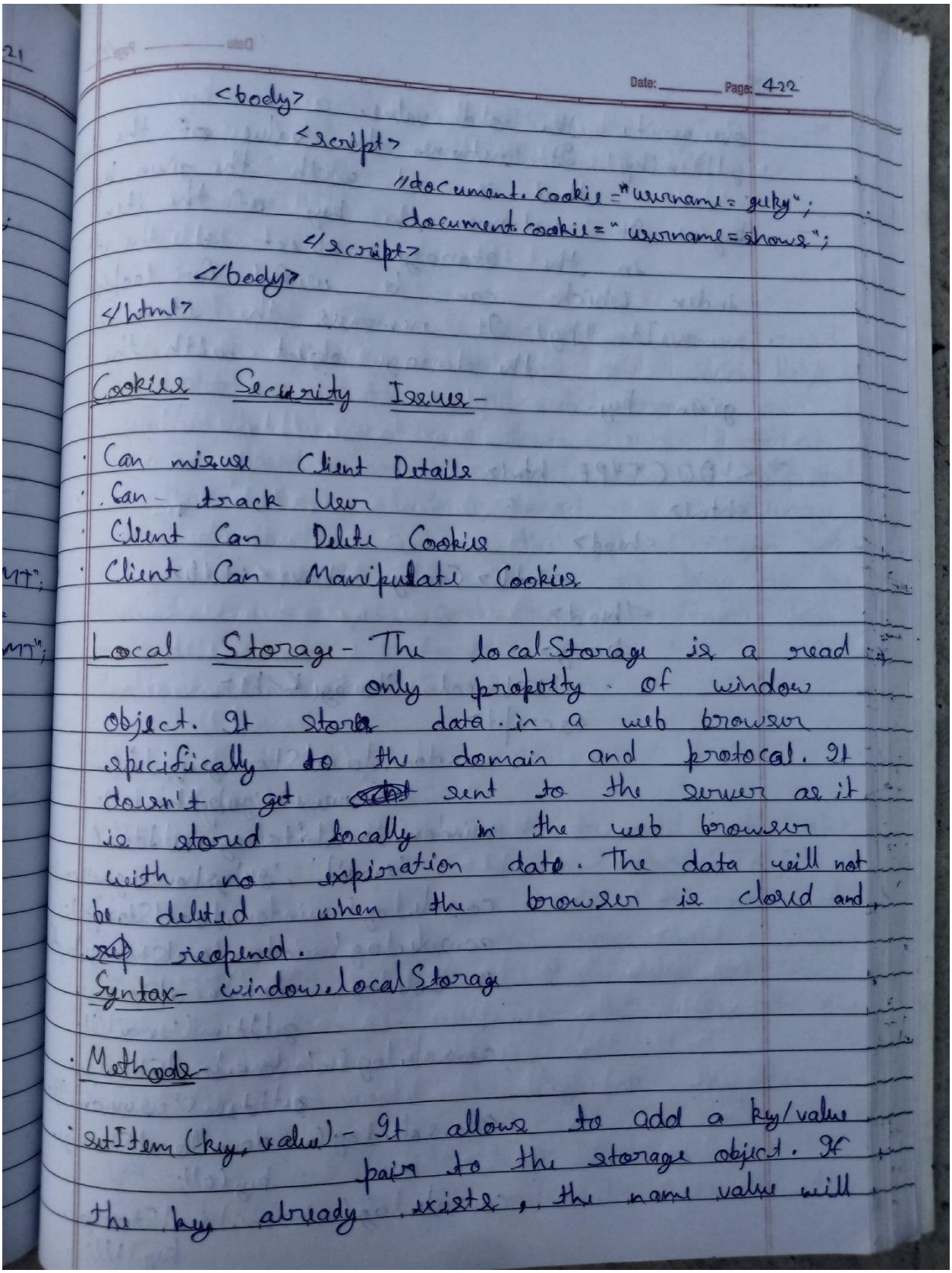


```
// document.cookie = "username=geky;
    expires = Mon, 3-Sep-2021
    09:00:00 GMT";
document.cookie = "username=geky;
    expires = Thu, 3-Jan-1970
    00:00:01 GMT";
// Example-2
// document.cookie = "username=geky;
    max-age = " + 60*60*24*10;
document.cookie = "username=geky;
    max-age=60";
// Example-3
// document.cookie = "username; expires=
    Mon, 3-Sep-2021 09:00:00 GMT
document.cookie = "username; expires=
    Thu, 3-Aug-2018 09:00:00GMT
</script>
</body>
</html>
```

Update Cookies - A cookie is possible to update by setting new value to a cookie with the same name.

Ex- document.cookie = "username=geeky";
→ document.cookie = "username=shows";

```
→ <!DOCTYPE html>
<html>
    <head>
        <title> Geeky Shows </title>
    </head>
```



```
<body>
    <script>
        //document.cookie = "username = geky";
        document.cookie = "username = shows";
    </script>
</body>
</html>
```

Cookies Security Issues -

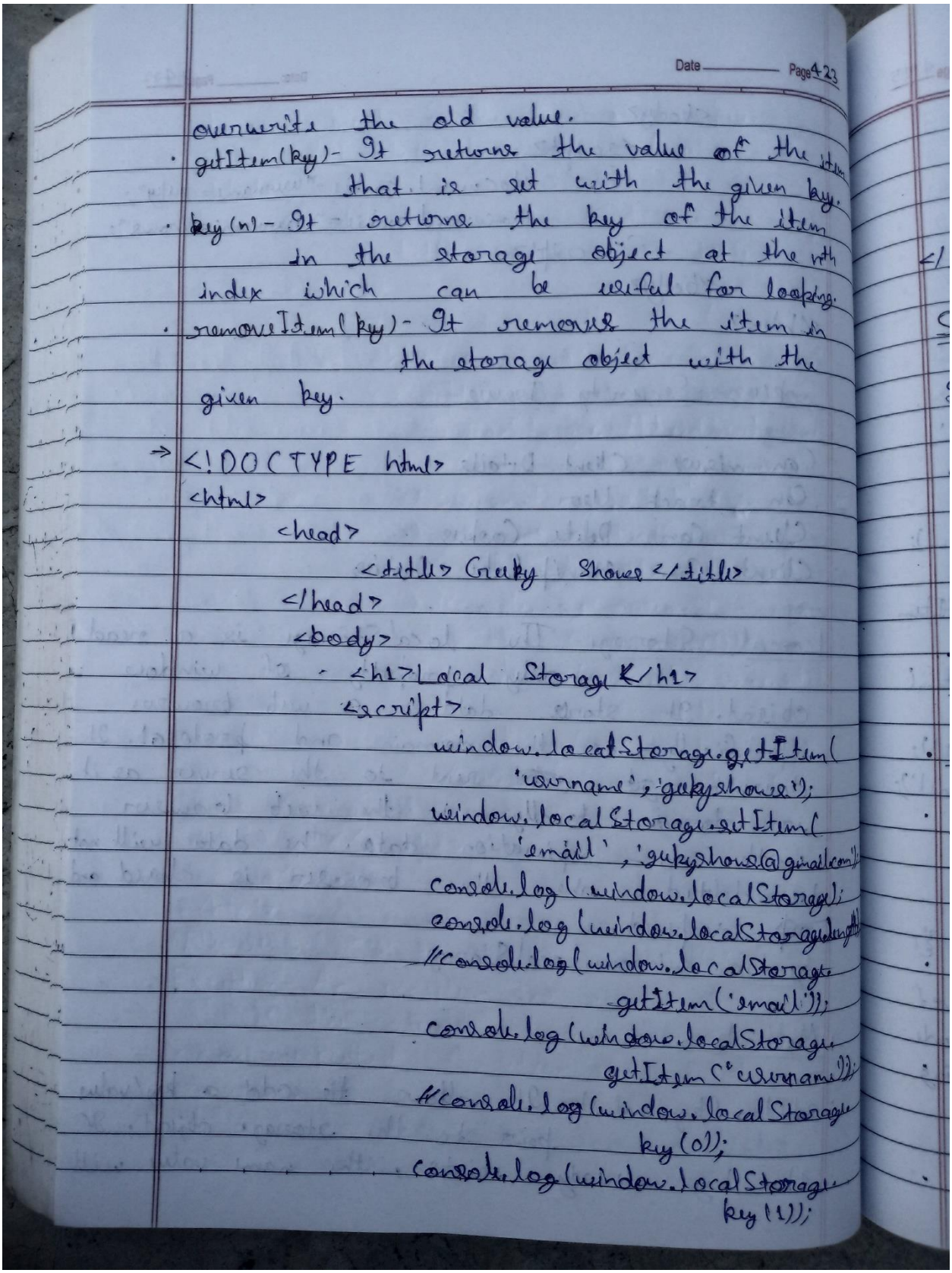
- Can misuse Client Details
- Can track User
- Client Can Delete Cookies
- Client Can Manipulate Cookies

Local Storage - The localStorage is a read only property of window object. It stores data in a web browser specifically to the domain and protocal. It doesn't get sent to the server as it is stored locally in the web browser with no expiration date. The data will not be deleted when the browser is closed and reopened.

Syntax - window.localStorage

- Methods -

setItem (key, value) - It allows to add a key/value pair to the storage object. If the key already exists, the name value will



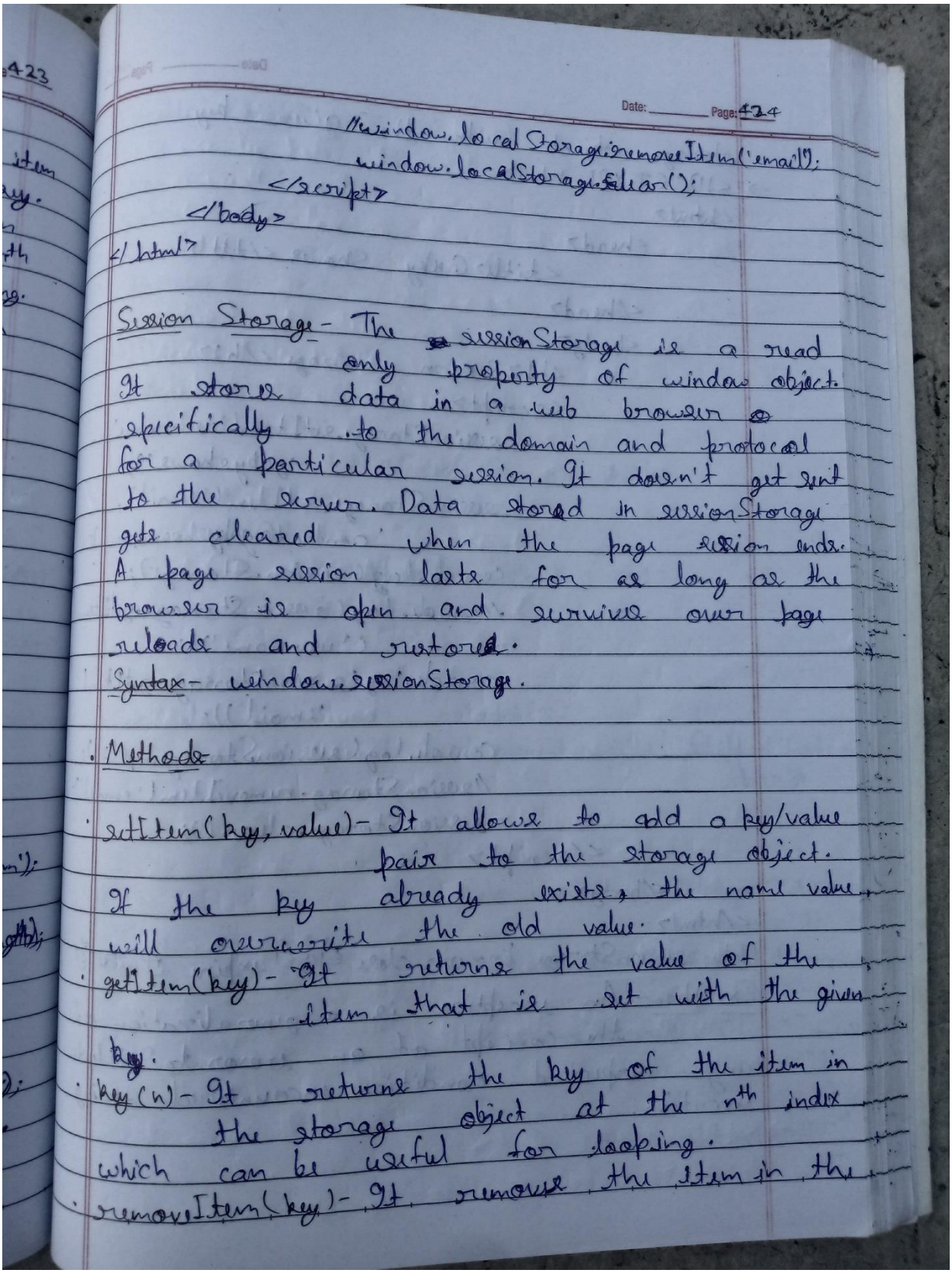
Date Page 423

overwrite the old value.

- getItem(key) - It returns the value of the item that is set with the given key.
- key(n) - It returns the key of the item in the storage object at the nth index which can be useful for looping.
- removeItem(key) - It removes the item in the storage object with the given key.

→

```
<!DOCTYPE html>
<html>
    <head>
        <title> Geeky Shows </title>
    </head>
    <body>
        <h1> Local Storage </h1>
        <script>
            window.localStorage.setItem(
                'username', 'gekyshows');
            window.localStorage.setItem(
                'email', 'gekyshows@gmail.com');
            console.log(window.localStorage);
            console.log(window.localStorage.length);
            //console.log(window.localStorage.
                getItem('email'));
            console.log(window.localStorage.
                getItem('username'));
            //console.log(window.localStorage.
                key(0));
            console.log(window.localStorage.
                key(1));
```



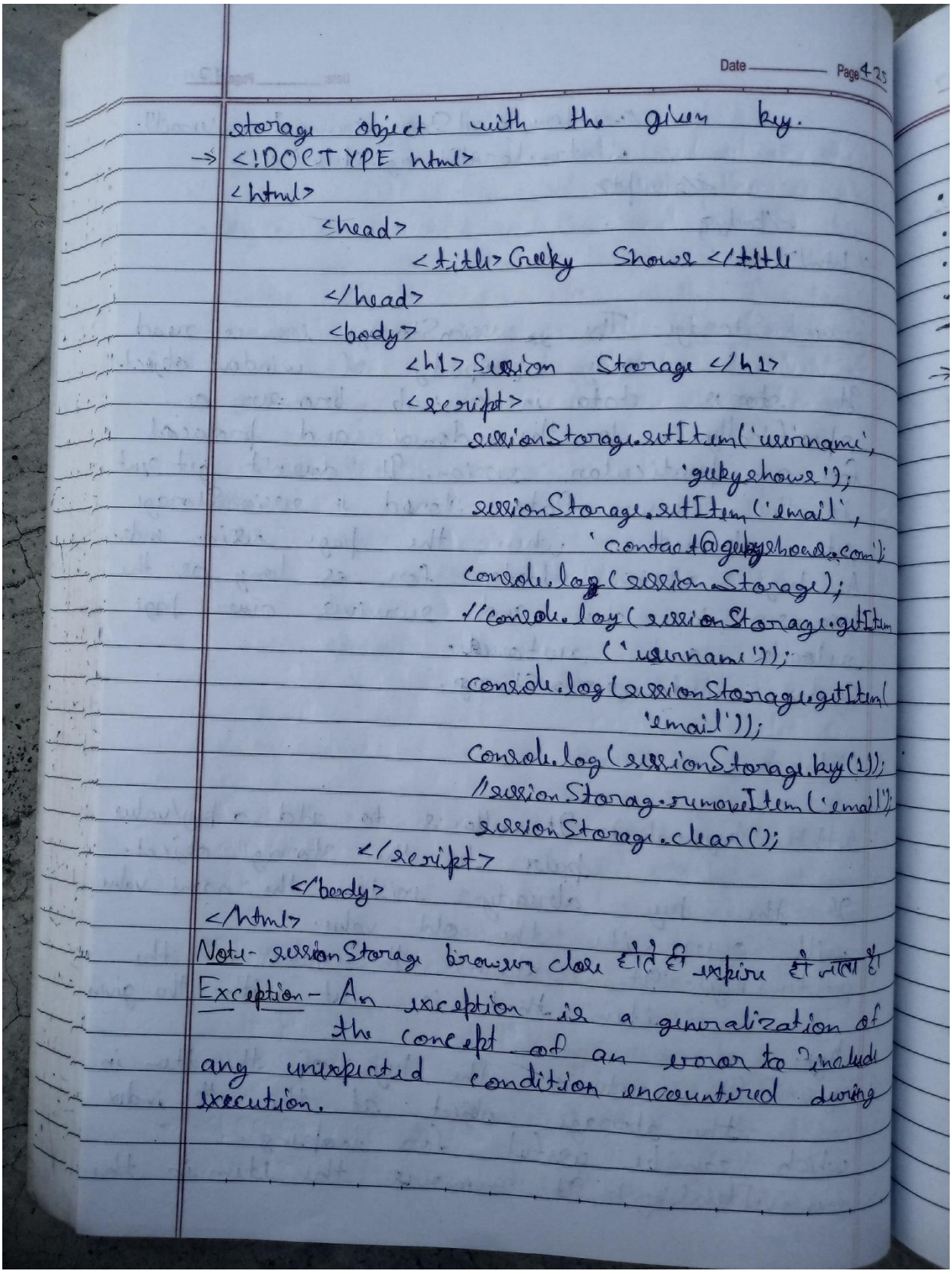
//window.localStorage.removeItem('email');
window.localStorage.clear();
</script>
</body>
</html>

Session Storage - The sessionStorage is a read only property of window object. It stores data in a web browser specifically to the domain and protocol for a particular session. It doesn't get sent to the server. Data stored in sessionStorage gets cleared when the page session ends. A page session lasts for as long as the browser is open and survives over page reloads and restores.

Syntax - window.sessionStorage.

- Methods

- setItem(key, value) - It allows to add a key/value pair to the storage object. If the key already exists, the name value will overwrite the old value.
- getItem(key) - It returns the value of the item that is set with the given key.
- key(n) - It returns the key of the item in the storage object at the nth index which can be useful for looping.
- removeItem(key) - It removes the item in the

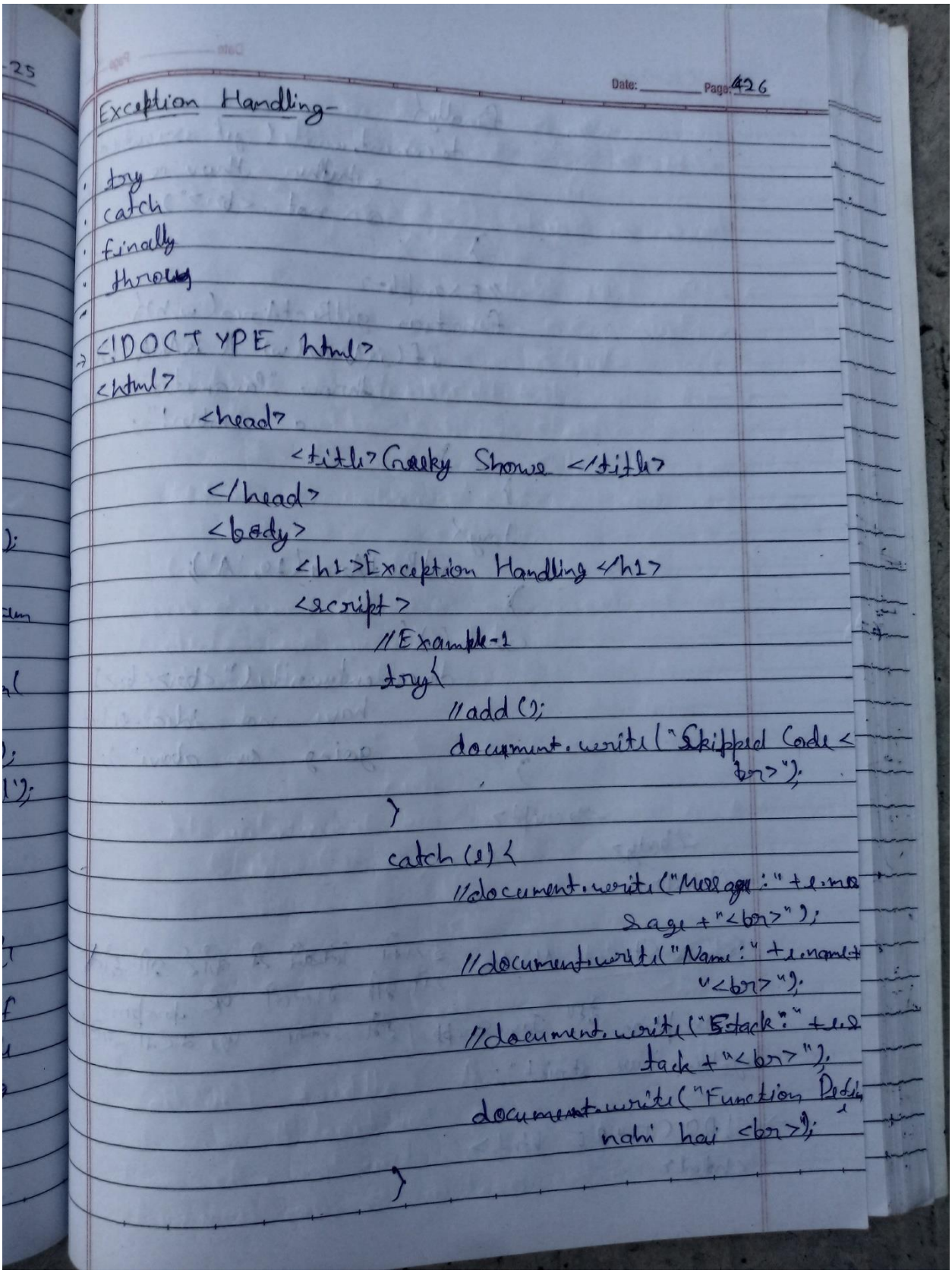


storage object with the given key.

→
```
<!DOCTYPE html>
<html>
    <head>
        <title> Geeky Shows </title>
    </head>
    <body>
        <h1> Session Storage </h1>
        <script>
            sessionStorage.setItem('username',
                'gekyshows');
            sessionStorage.setItem('email',
                'contact@gekyshows.com');
            console.log(sessionStorage);
            //console.log(sessionStorage.getItem
                ('username'));
            console.log(sessionStorage.getItem(
                'email'));
            Console.log(sessionStorage.key(1));
            //sessionStorage.removeItem('email');
            sessionStorage.clear();
        </script>
    </body>
</html>
```

Note- sessionStorage browser close होते ही expire हो जाता है

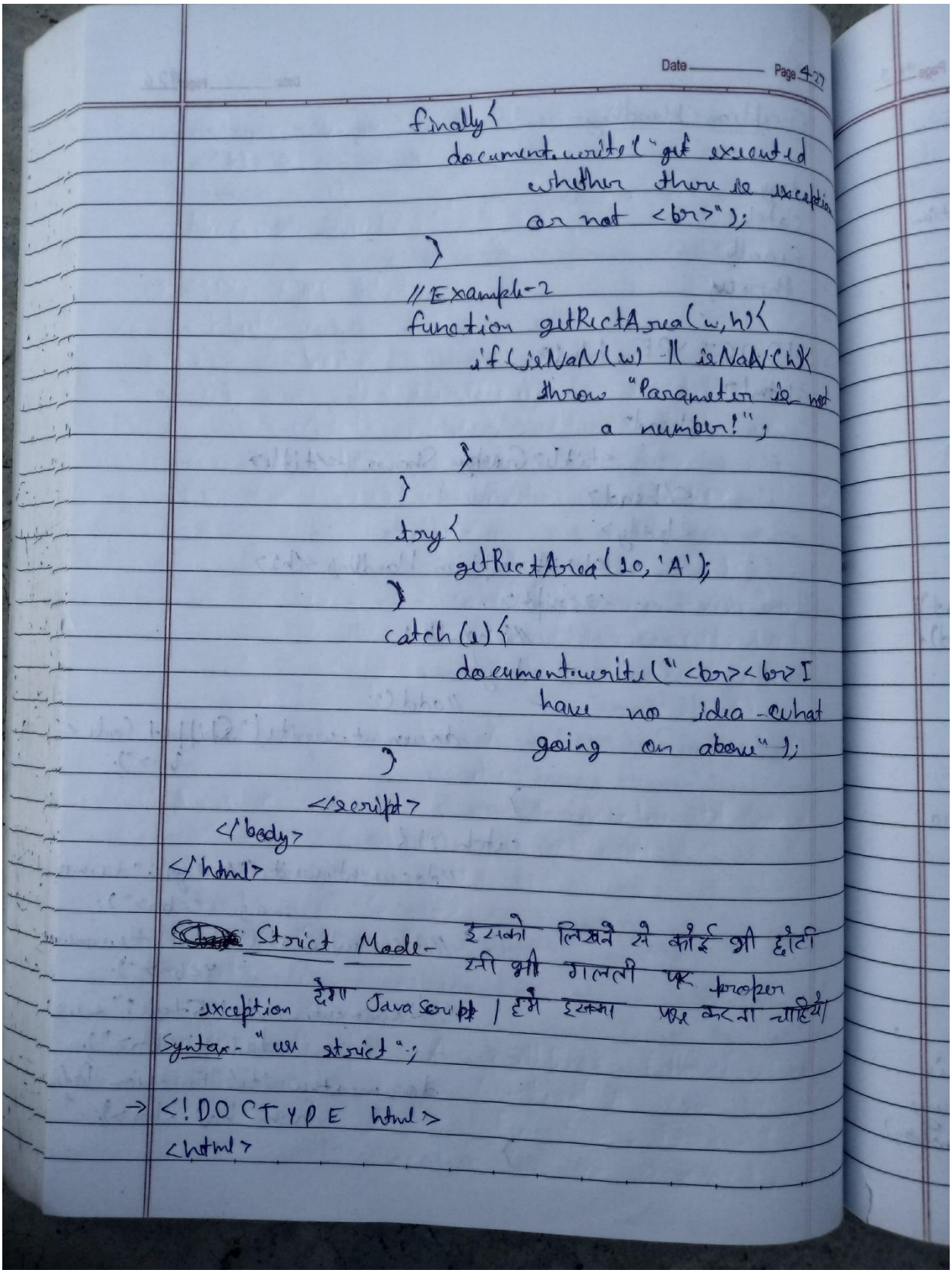
Exception- An exception is a generalization of the concept of an error to include any unexpected condition encountered during execution.



Exception Handling-

- try
- catch
- finally
- throw

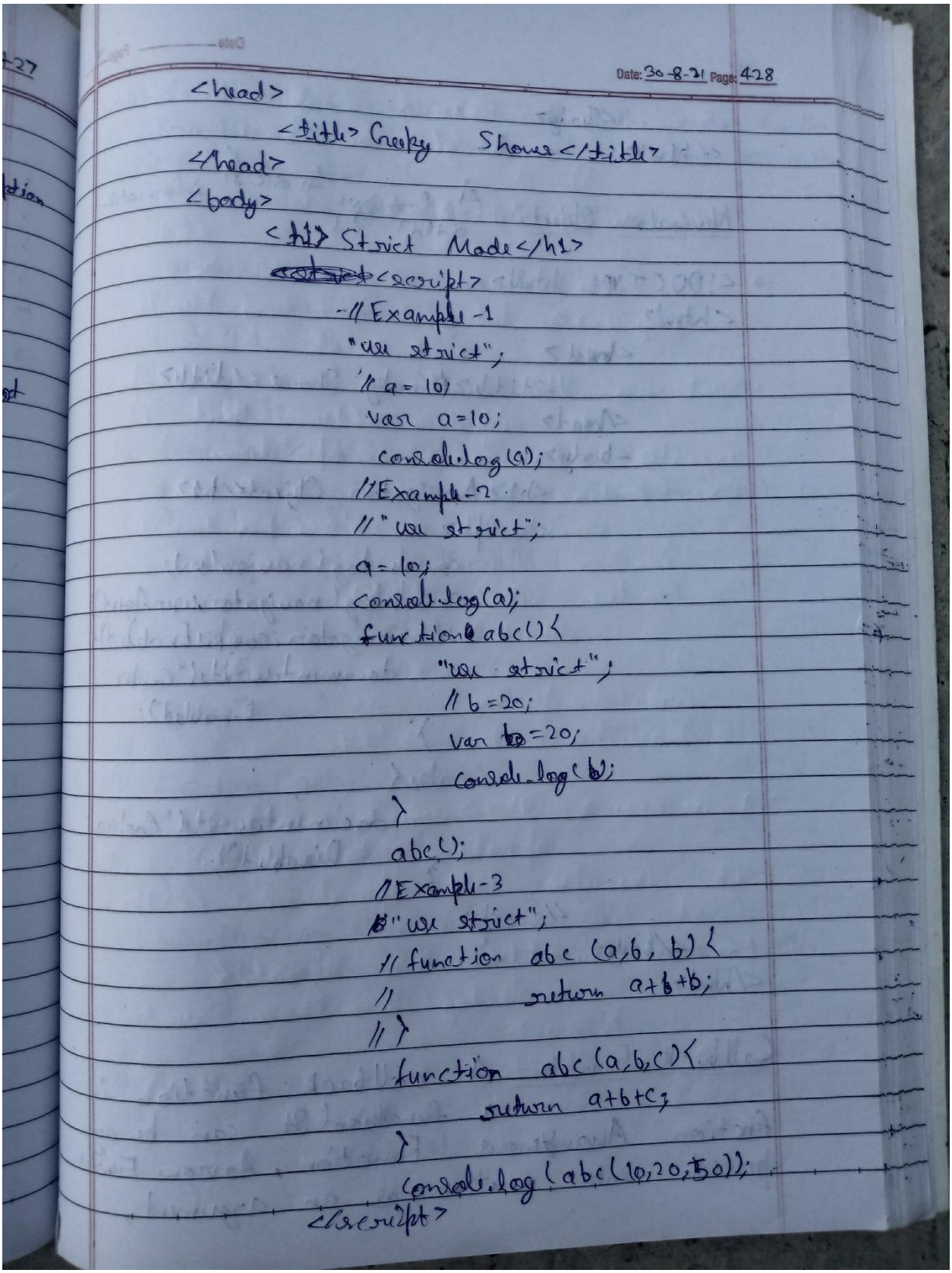
```
<!DOCTYPE html>
<html>
    <head>
        <title> Geeky Shows </title>
    </head>
    <body>
        <h1> Exception Handling </h1>
        <script>
            //Example-1
            try{
                //add();
                document.write("Skipped Code <
                    br>");
            }
            catch(e){
                //document.write("Message :" +e.mes
                    sage +"<br>");
                //document.write("Name :" +e.name+
                    "<br>");
                //document.write("Stack :" +e.
                    stack +"<br>");
                document.write("Function Defi
                    nahi hai <br>");
            }
```



```
finally{
    document.write("get executed
        whether there is exception
        or not <br>");
}
//Example-2
function getRectArea(w,h){
    if(isNaN(w) || isNaN(h)){
        throw "Parameter is not
            a number!";
    }
}
try{
    getRectArea(10,'A');
}
catch(e){
    document.write("<br><br>I
        have no idea what
        going on above");
}
</script>
</body>
</html>
```

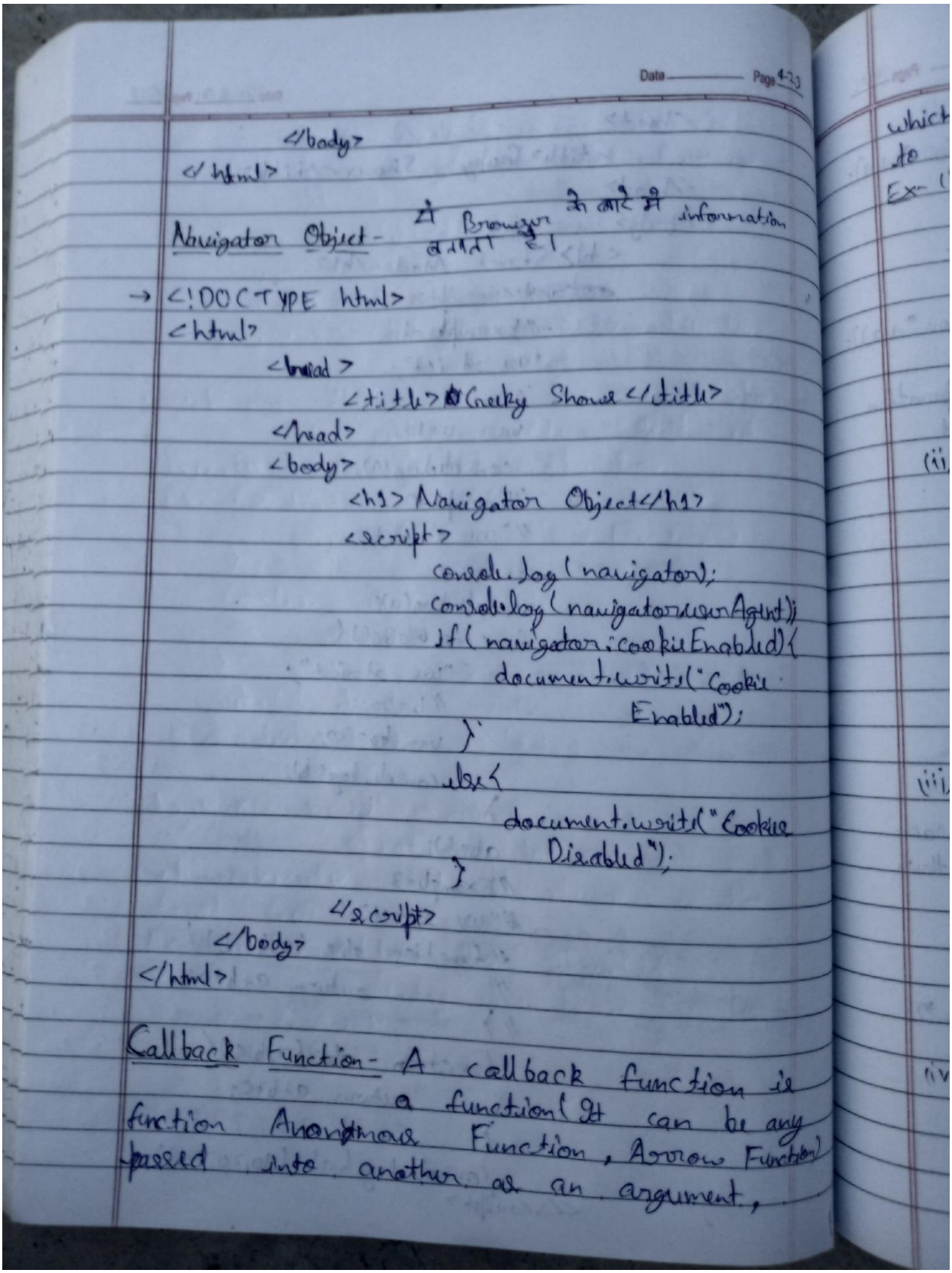
Strict Mode- इसको लिखने से कोई भी होती
इसी भी गलती पर proper
exception देगा JavaScript / हमें इसका use करना चाहिये।
Syntax- "use strict";

```
→ <!DOCTYPE html>
<html>
```



Date: 30-8-21 Page: 428

```
<head>
    <title> Geeky Shows </title>
</head>
<body>
    <h1> Strict Mode </h1>
    <script>
        // Example -1
        "use strict";
        // a = 10;
        var a=10;
        console.log(a);
        //Example-2
        // "use strict";
        a=10;
        console.log(a);
        function abc(){
            "use strict";
            // b=20;
            var b=20;
            console.log(b);
        }
        abc();
        //Example-3
        "use strict";
        // function abc(a,b,b){
        //     return a+b+b;
        // }
        function abc(a,b,c){
            return a+b+c;
        }
        console.log(abc(10,20,30));
    </script>
```

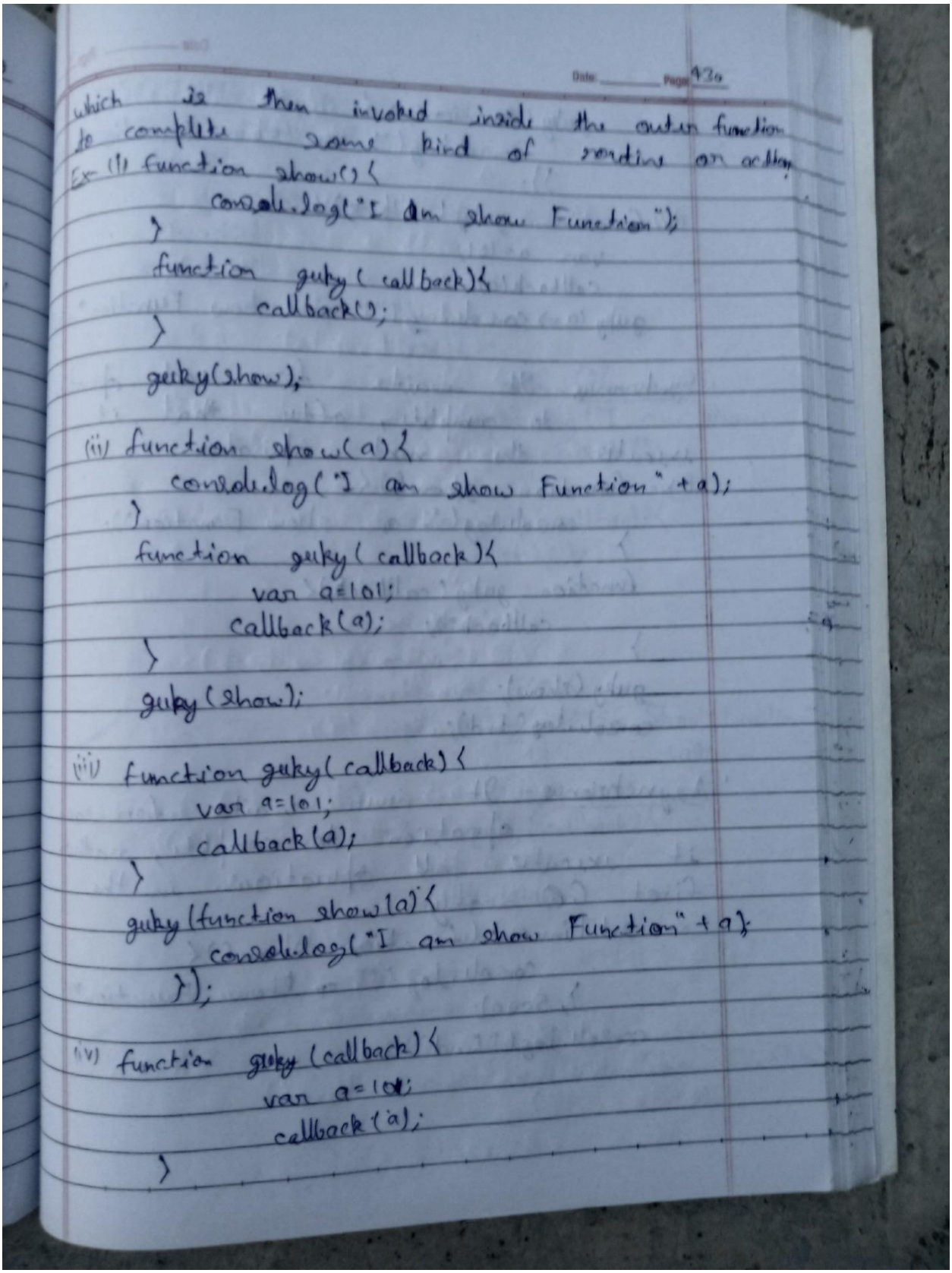


```
</body>
</html>
```

Navigator Object - Browser के बारे में information देता है।

```
<!DOCTYPE html>
<html>
    <head>
        <title> Geeky Shows </title>
    </head>
    <body>
        <h1> Navigator Object </h1>
        <script>
            console.log(navigator);
            console.log(navigator.userAgent);
            if(navigator.cookieEnabled){
                document.write("Cookie
                    Enabled");
            }
            else{
                document.write("Cookie
                    Disabled");
            }
        </script>
    </body>
</html>
```

Callback Function - A callback function is a function (It can be any function Anonymous Function, Arrow Function) passed into another as an argument,



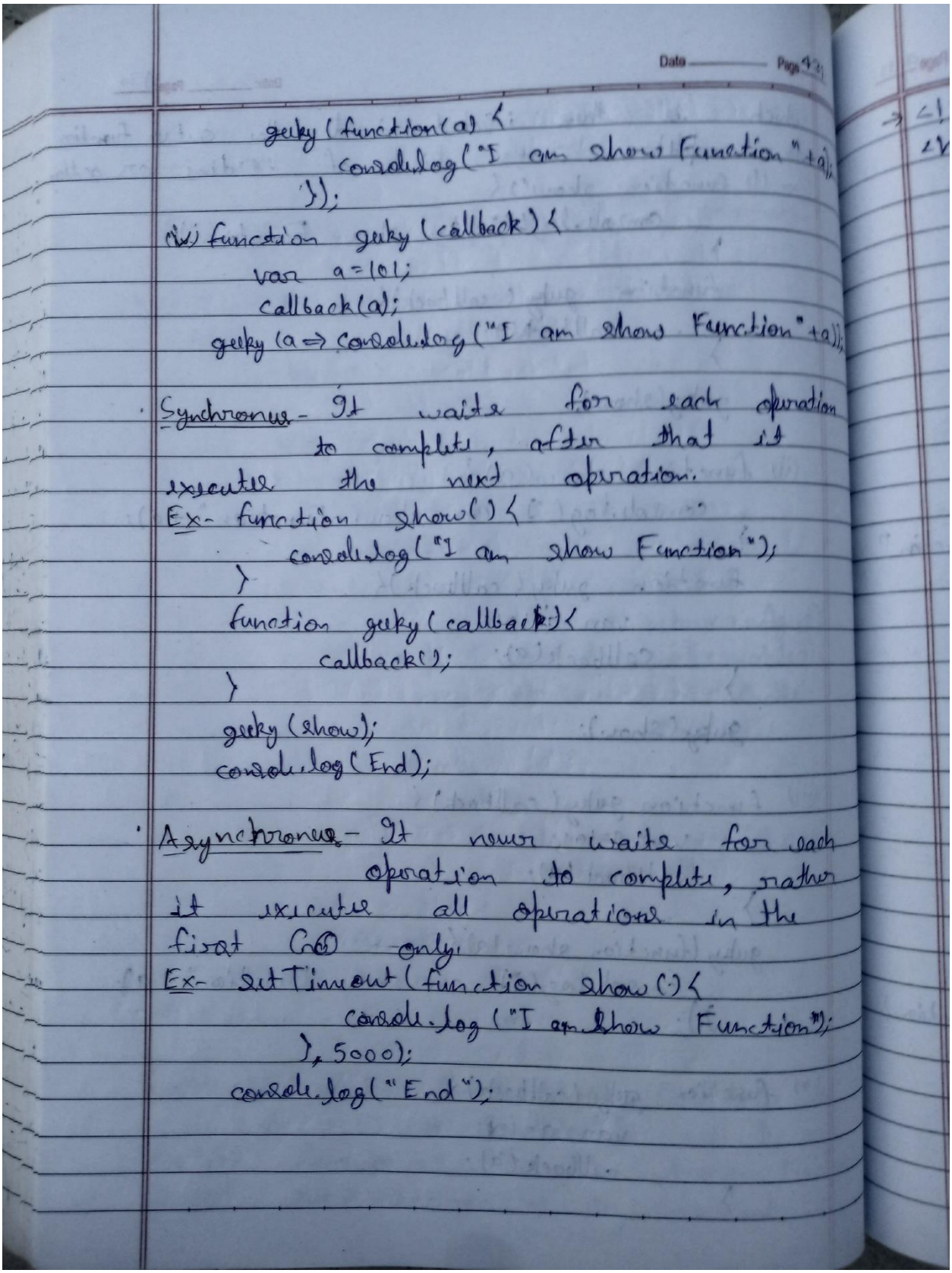
which is then invoked inside the outer function to complete some kind of routine or action.

Ex- (i)
```
function show(){
    console.log("I am show Function");
}
function geeky(callback){
    callback();
}
geeky(show);
```

(ii)
```
function show(a){
    console.log("I am show Function" + a);
}
function geeky(callback){
    var a=101;
    callback(a);
}
geeky(show);
```

(iii)
```
function geeky(callback){
    var a=101;
    callback(a);
}
geeky(function show(a){
    console.log("I am show Function" + a);
});
```

(iv)
```
function geeky(callback){
    var a=101;
    callback(a);
}
```



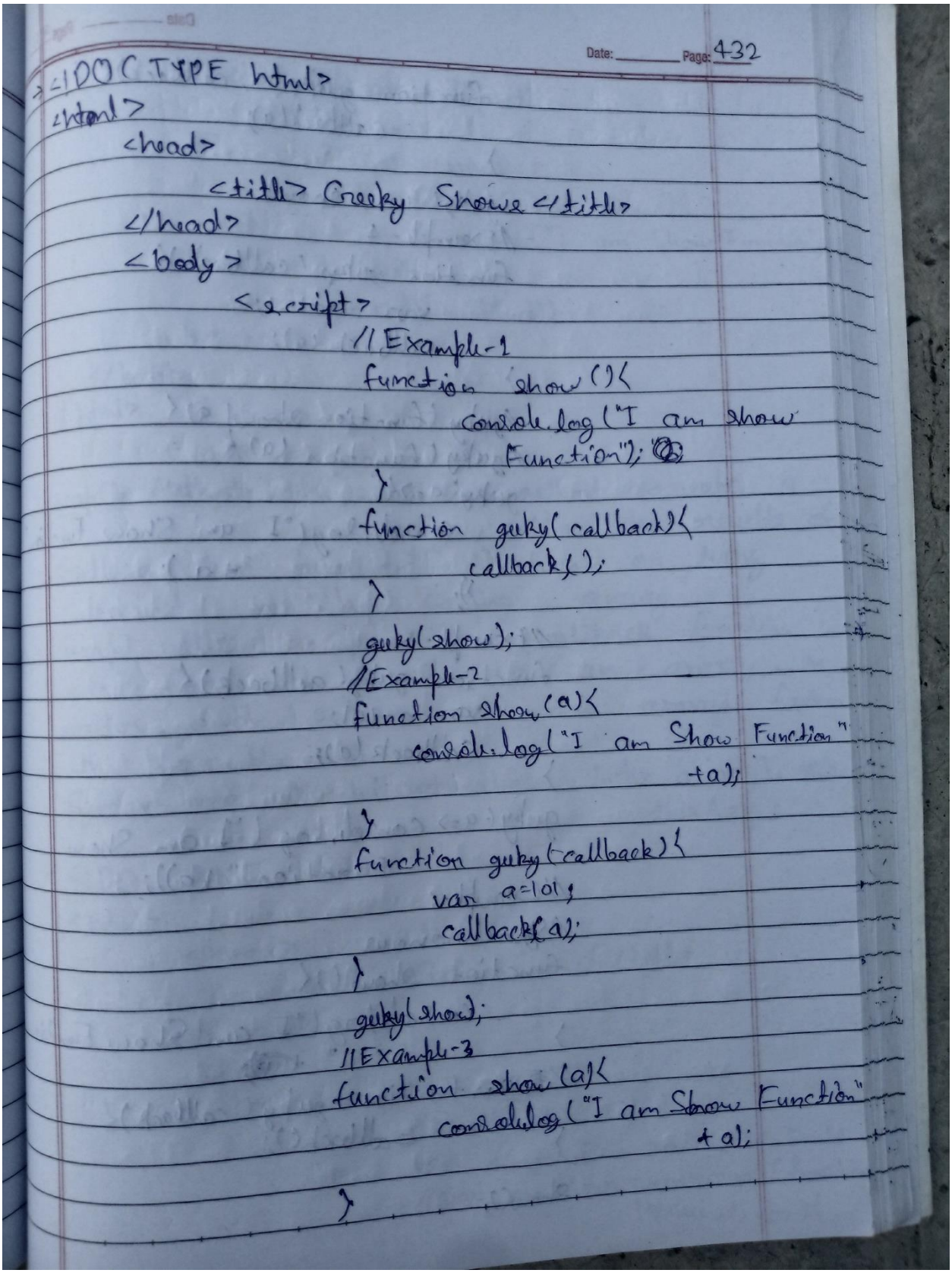
```
geeky(function(a){
    console.log("I am show Function"+a);
});
(iv) function geeky(callback){
    var a=101;
    callback(a);
geeky(a=> console.log("I am show Function"+a));
```

Synchronous - It waits for each operation to complete, after that it executes the next operation.

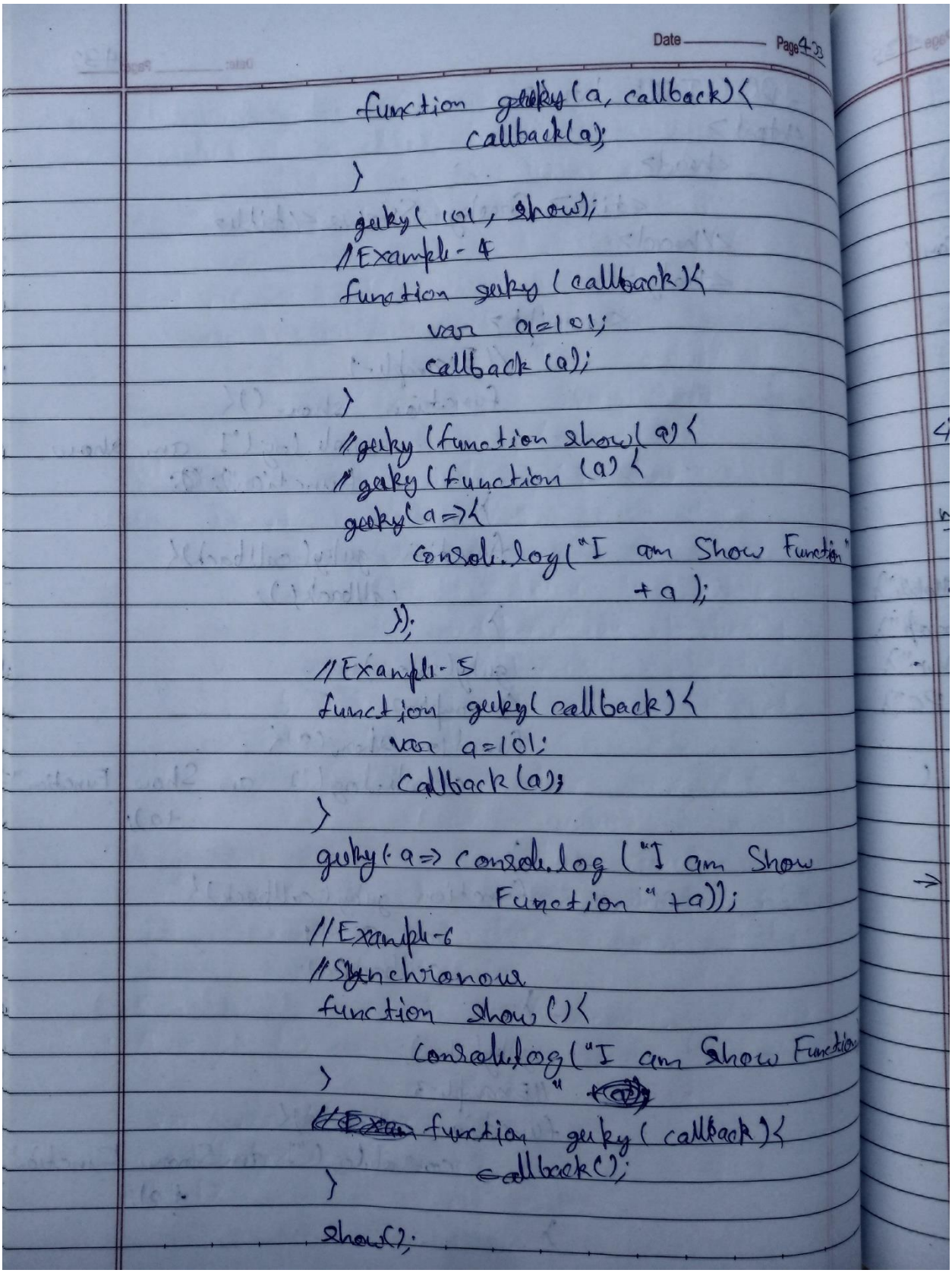
Ex- function show(){
    console.log("I am show Function");
}
function geeky(callback){
    callback();
}
geeky(show);
console.log(End);

Asynchronous - It never waits for each operation to complete, rather it executes all operations in the first GEO only.

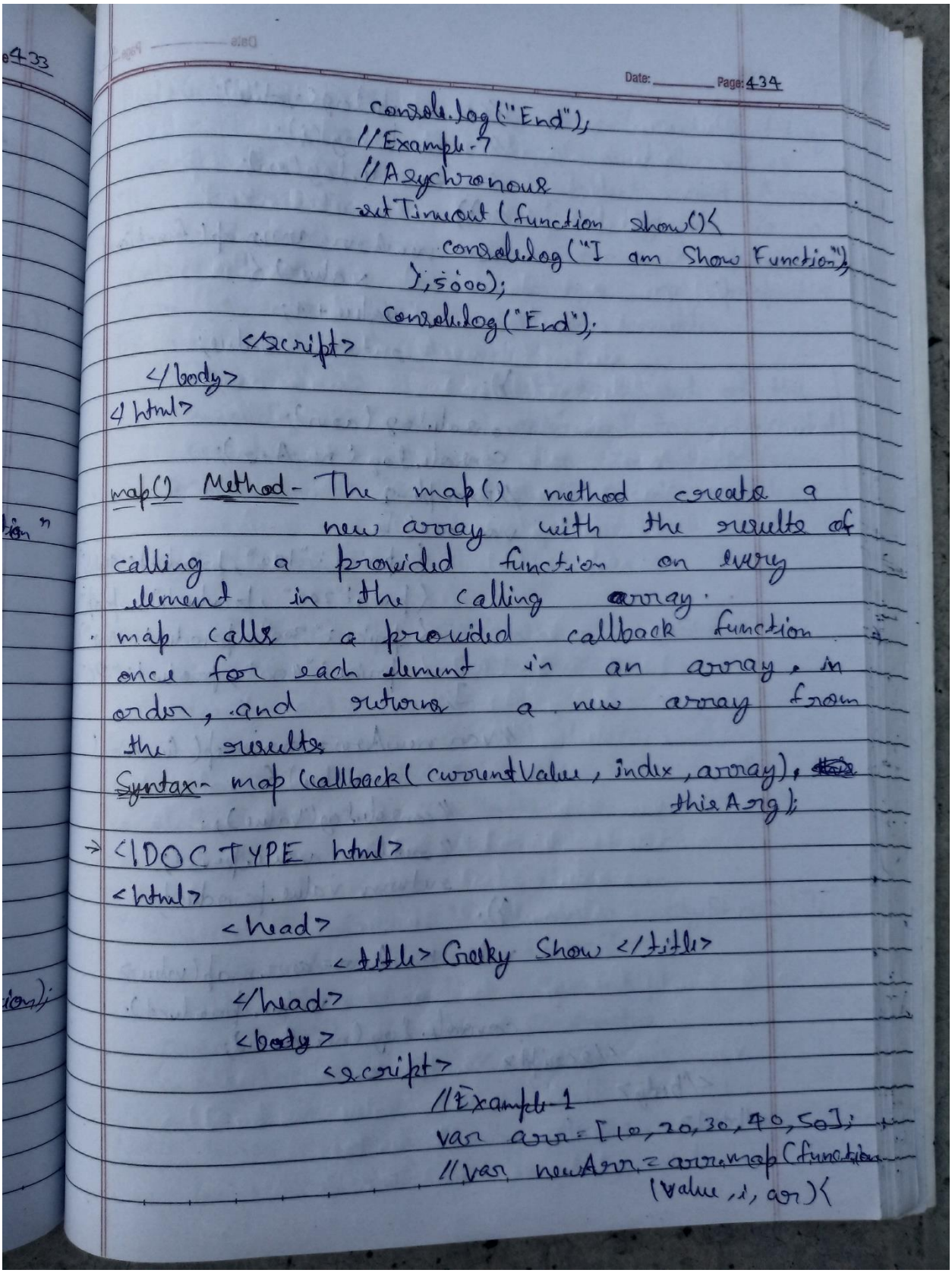
Ex- setTimeout(function show(){
    console.log("I am show Function");
}, 5000);
console.log("End");



```
Date: Page: 432
<!DOCTYPE html>
<html>
    <head>
        <title> Geeky Shows </title>
</head>
<body>
        <script>
            //Example-1
            function show(){
                console.log("I am show
                    Function");
            }
            function geky(callback){
                callback();
            }
            geky(show);
            //Example-2
            function show(a){
                console.log("I am Show Function"
                    +a);
            }
            function geky(callback){
                var a=101;
                callback(a);
            }
            geky(show);
            //Example-3
            function show(a){
                console.log("I am Show Function"
                    +a);
            }
```



```
function geeky(a, callback){
    callback(a);
}
geeky(101, show);
//Example-4
function geeky(callback){
    var a=101;
    callback(a);
}
//geeky(function show(a){
//geeky(function (a){
geeky(a=>{
    console.log("I am Show Function"
        +a);
});
//Example-5
function geeky(callback){
    var a=101;
    Callback(a);
}
geeky(a=> console.log("I am Show
    Function "+a));
//Example-6
//Synchronous
function show(){
    console.log("I am Show Function
    "
function geeky(callback){
    callback();
}
show();
```

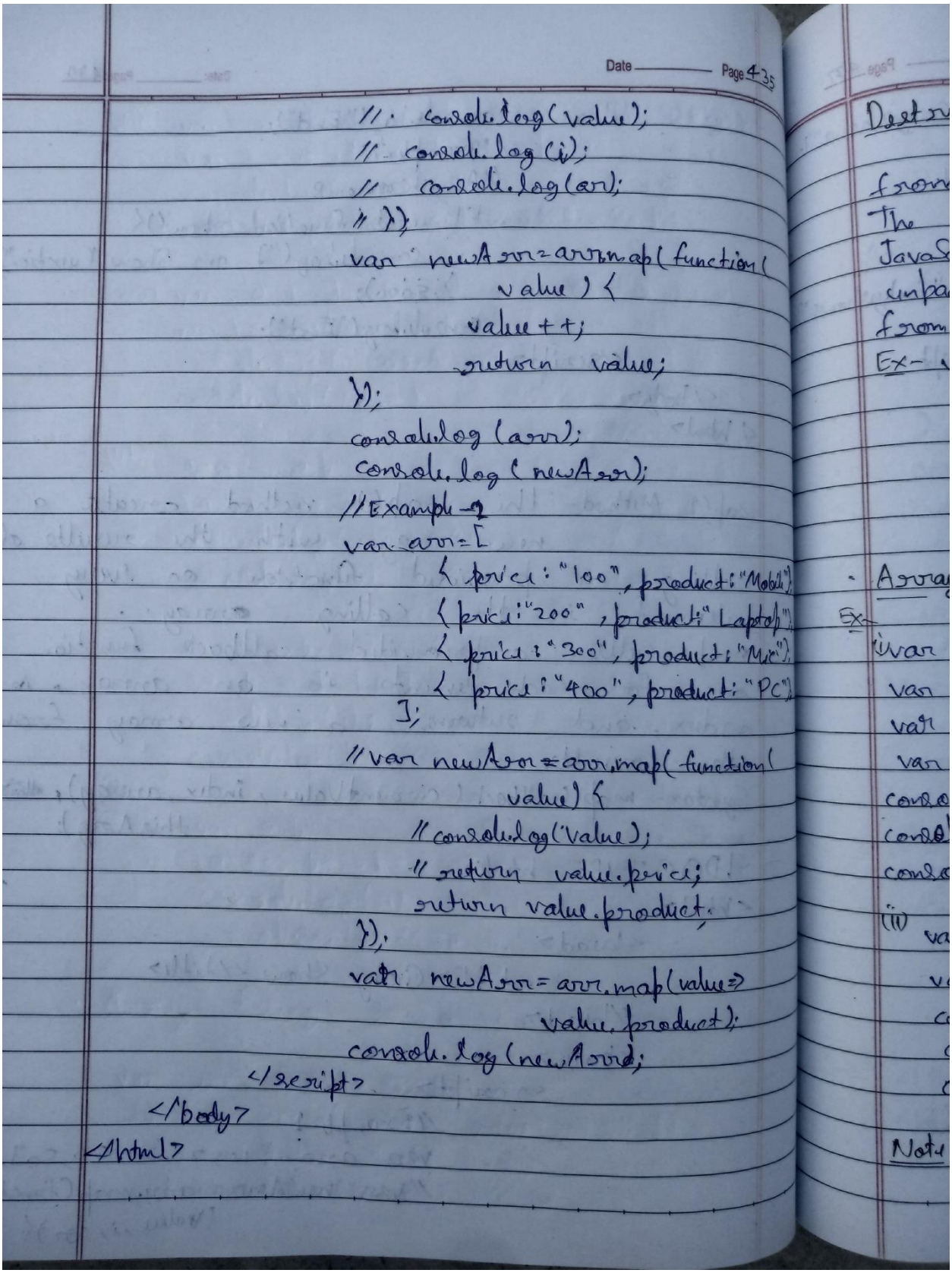


```
                console.log("End");
                //Example-7
                //Asychronous
                setTimeout(function show(){
                    console.log("I am Show Function");
                },5000);
                console.log("End");
            </script>
        </body>
    </html>
```

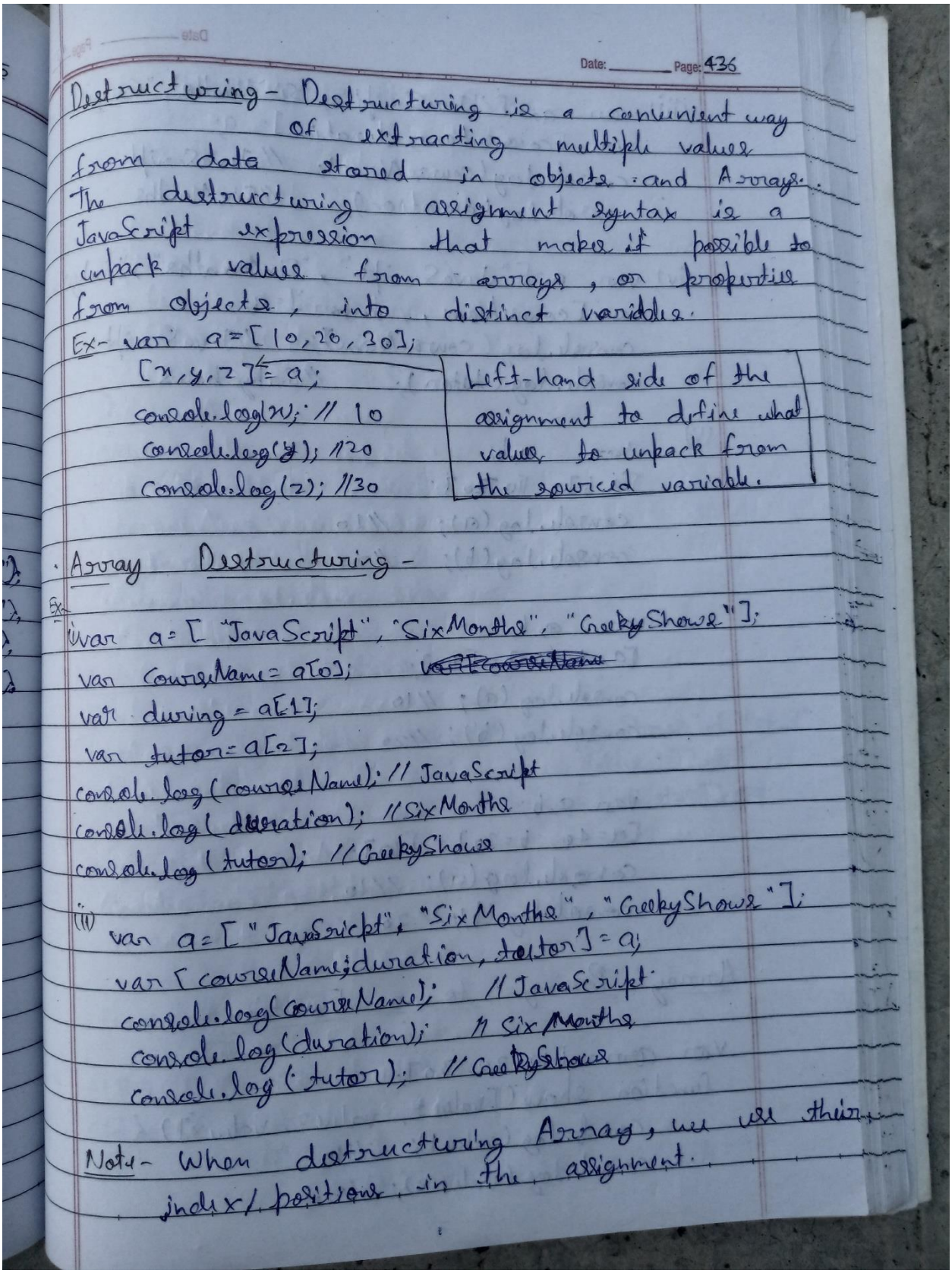
map() Method- The map() method creates a new array with the results of calling a provided function on every element in the calling array.
map calls a provided callback function once for each element in an array, in order, and returns a new array from the results.

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

```
<!DOCTYPE html>
<html>
    <head>
        <title> Geeky Show </title>
    </head>
    <body>
        <script>
            //Example-1
            var arr=[10,20,30,40,50];
            //var newArr=arr.map(function
                (value,i,arr){
```



```
// console.log(value);
// console.log(i);
// console.log(arr);
// });
var newArr = arr.map(function(
        value) {
    value++;
    return value;
});
console.log(arr);
console.log(newArr);
//Example-2
var arr = [
    { price: "100", product: "Mobile"},
    { price: "200", product: "Laptop"},
    { price: "300", product: "Mic"},
    { price: "400", product: "PC"},
];
//var newArr = arr.map(function(
        value) {
    //console.log(value);
    //return value.price;
    return value.product;
});
var newArr = arr.map(value=>
        value.product);
console.log(newArr);
</script>
</body>
</html>
```



Date: Page: 436

Destructuring - Destructuring is a convenient way of extracting multiple values from data stored in objects and Arrays. The destructuring assignment syntax is a JavaScript expression that makes it possible to unpack values from arrays, or properties from objects, into distinct variables.

Ex- var a = [10, 20, 30];

```
[x, y, z] = a;
console.log(x); // 10
console.log(y); // 20
console.log(z); // 30
```

Left-hand side of the assignment to define what values to unpack from the sourced variable.

- **Array Destructuring** -

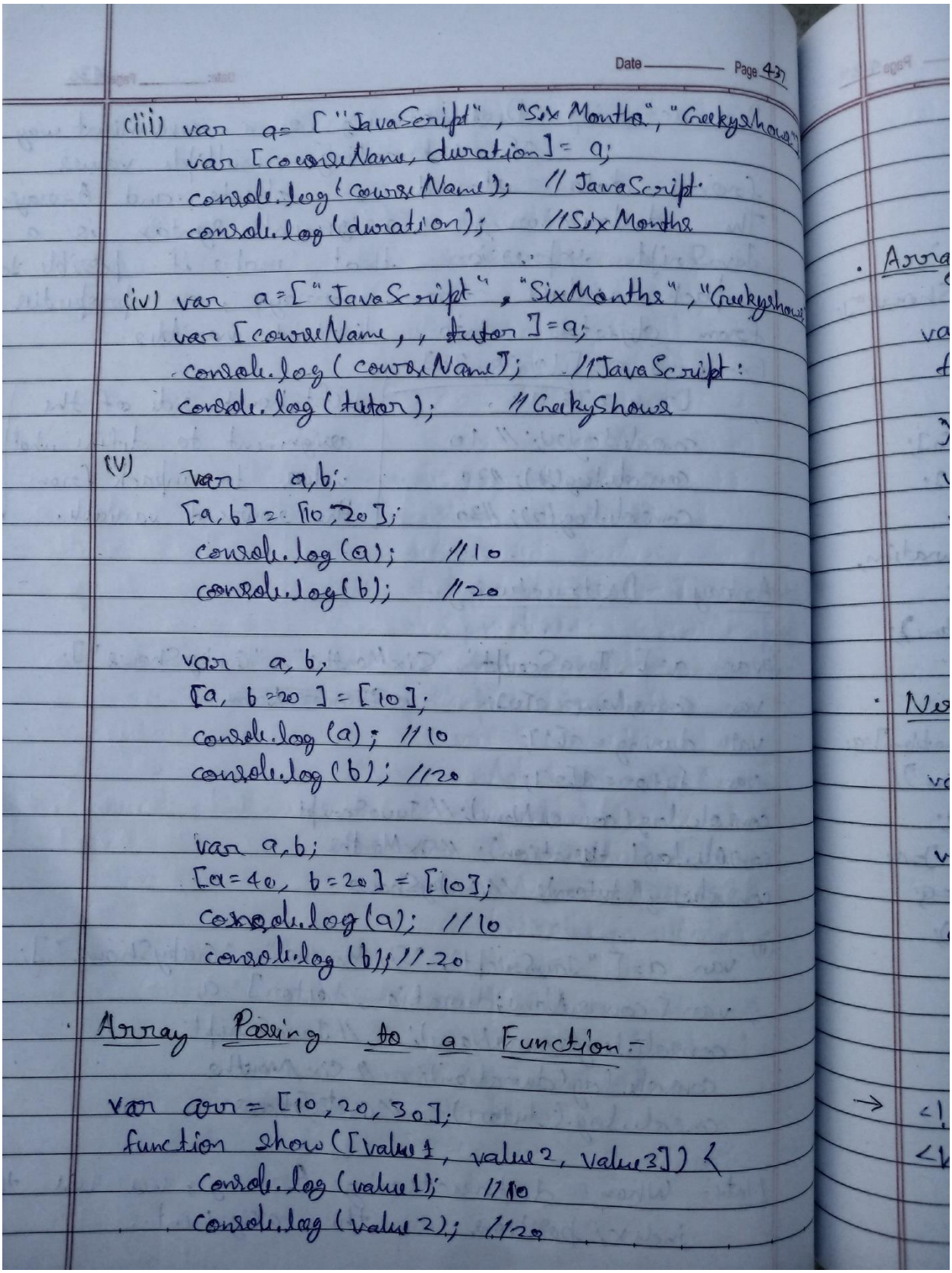
Ex-

```
var a = ["JavaScript", "SixMonths", "GeekyShows"];
var courseName = a[0];
var during = a[1];
var tutor = a[2];
console.log(courseName); // JavaScript
console.log(duration); // SixMonths
console.log(tutor); // GeekyShows
```

(ii)

```
var a = ["JavaScript", "SixMonths", "GeekyShows"];
var [courseName, duration, tutor] = a;
console.log(courseName); // JavaScript
console.log(duration); // SixMonths
console.log(tutor); // GeekyShows
```

Note- When destructuring Arrays, we use their indix/positions in the assignment.



(iii) var a= ["JavaScript", "Six Months", "Geekyshows"]
var [courseName, duration] = a;
console.log(courseName); // JavaScript
console.log(duration); // Six Months

(iv) var a=["JavaScript", "SixMonths", "Geekyshows"]
var [courseName, , tutor] = a;
console.log(courseName); // JavaScript
console.log(tutor); // Geekyshows

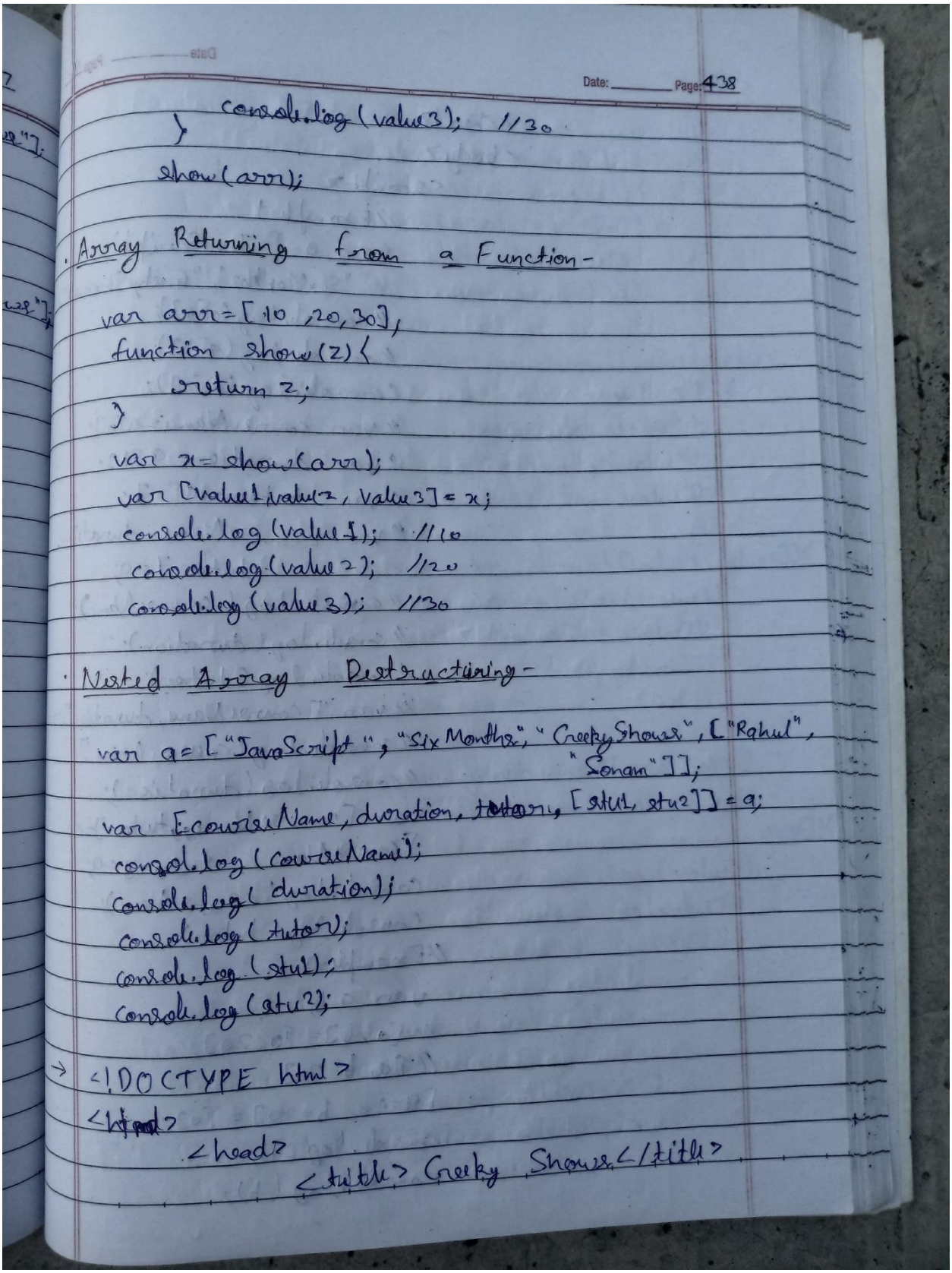
(v)
var a,b;
[a,b] = [10,20];
console.log(a); //10
console.log(b); //20

var a, b;
[a, b=20] = [10];
console.log(a); //10
console.log(b); //20

var a,b;
[a=40, b=20] = [10];
console.log(a); //10
console.log(b); //20

Array Passing to a Function =

var arr = [10,20,30];
function show([value1, value2, value3]) {
console.log(value1); //10
console.log(value2); //20



```
        console.log(value3); //30
    }
    show(arr);
```

. Array Returning from a Function -

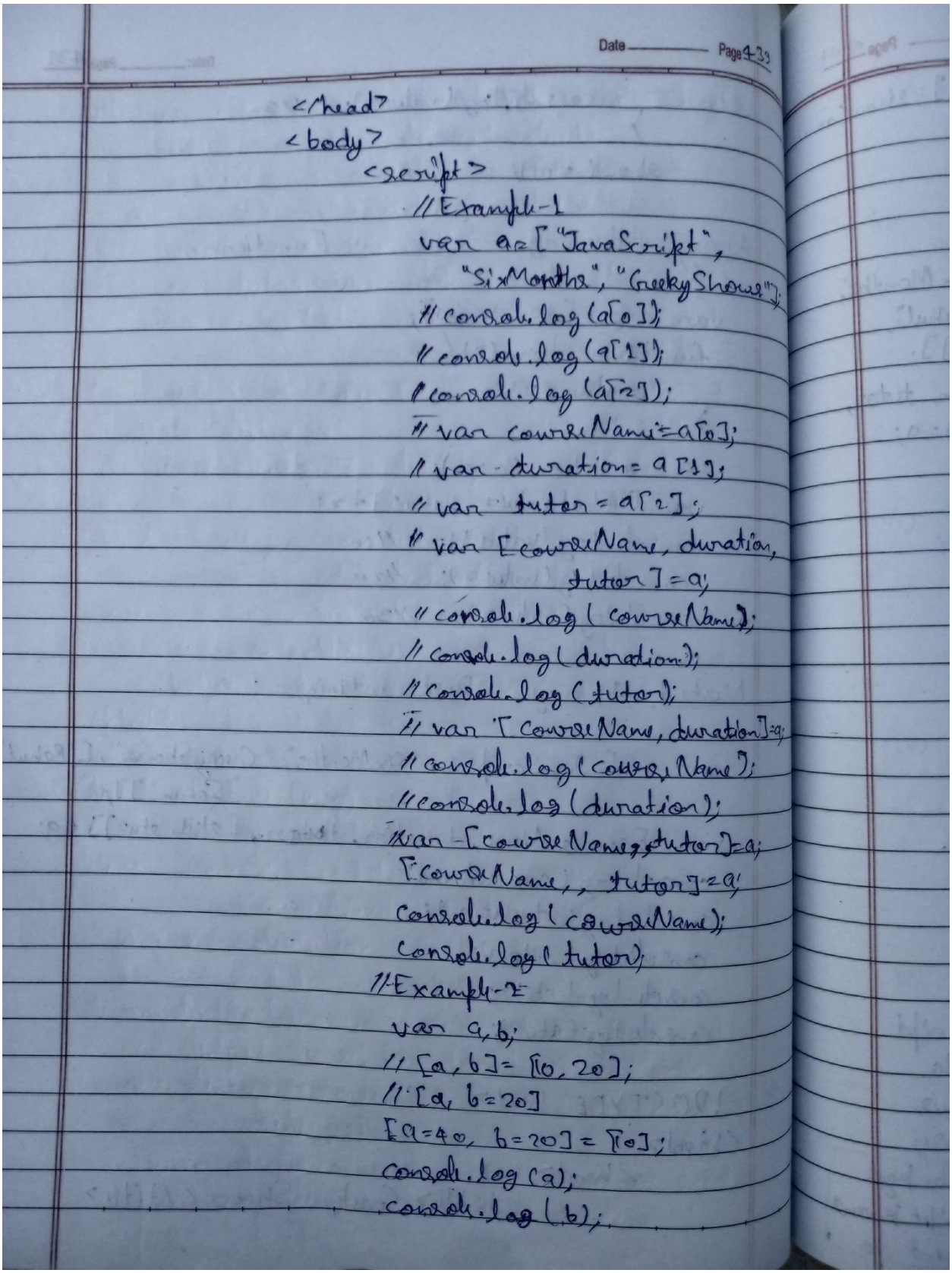
```
var arr = [10, 20, 30];
function show(z){
    return z;
}
var x = show(arr);
var [value1, value2, value3] = x;
console.log(value1); //10
console.log(value2); //20
console.log(value3); //30
```

. Nested Array Destructuring -

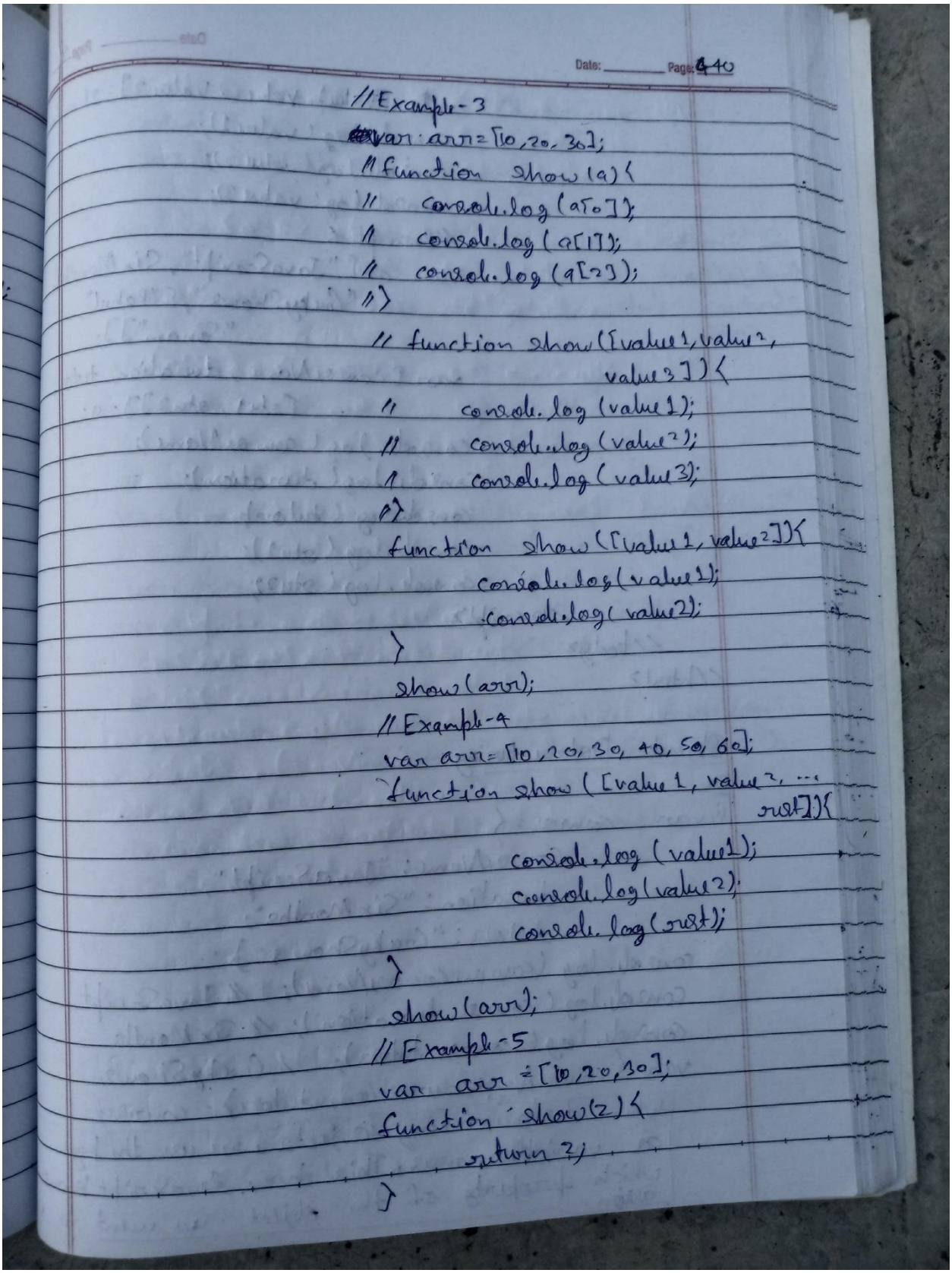
```
var a = ["JavaScript", "Six Months", "Geeky Shows", ["Rahul",
                                                     "Sonam"]];
var [courseName, duration, tutor, [stu1, stu2]] = a;
console.log(courseName);
console.log(duration);
console.log(tutor);
console.log(stu1);
console.log(stu2);
```

→

```
<!DOCTYPE html>
<html>
    <head>
        <title> Geeky Shows </title>
```



```
</head>
<body>
    <script>
        // Example-1
        var a=["JavaScript",
            "Six Months", "GeekyShows"];
        // console.log(a[0]);
        // console.log(a[1]);
        // console.log(a[2]);
        // var courseName=a[0];
        // var duration=a[1];
        // var tutor=a[2];
        // var [courseName, duration,
                tutor]=a;
        // console.log(courseName);
        // console.log(duration);
        // console.log(tutor);
        // var [courseName, duration]=a;
        // console.log(courseName);
        // console.log(duration);
        // var [courseName, tutor]=a;
        [courseName, , tutor]=a;
        console.log(courseName);
        console.log(tutor);
        // Example-2
        var a, b;
        // [a, b]=[10, 20];
        // [a, b=20]
        [a=40, b=20]=[10];
        console.log(a);
        console.log(b);
```



```
Date: _____ Page: 110
//Example-3
var arr=[10,20,30];
//function show(a){
//  console.log(a[0]);
//  console.log(a[1]);
//  console.log(a[2]);
//}
// function show([value1,value2,
                  value3]){
//     console.log(value1);
//     console.log(value2);
//     console.log(value3);
//}
function show([value1,value2]){
     console.log(value1);
     console.log(value2);
}
show(arr);
//Example-4
var arr=[10,20,30,40,50,60];
function show([value1,value2,...
                  rest]){
     console.log(value1);
     console.log(value2);
     console.log(rest);
}
show(arr);
//Example-5
var arr=[10,20,30];
function show(z){
     return z;
}
```

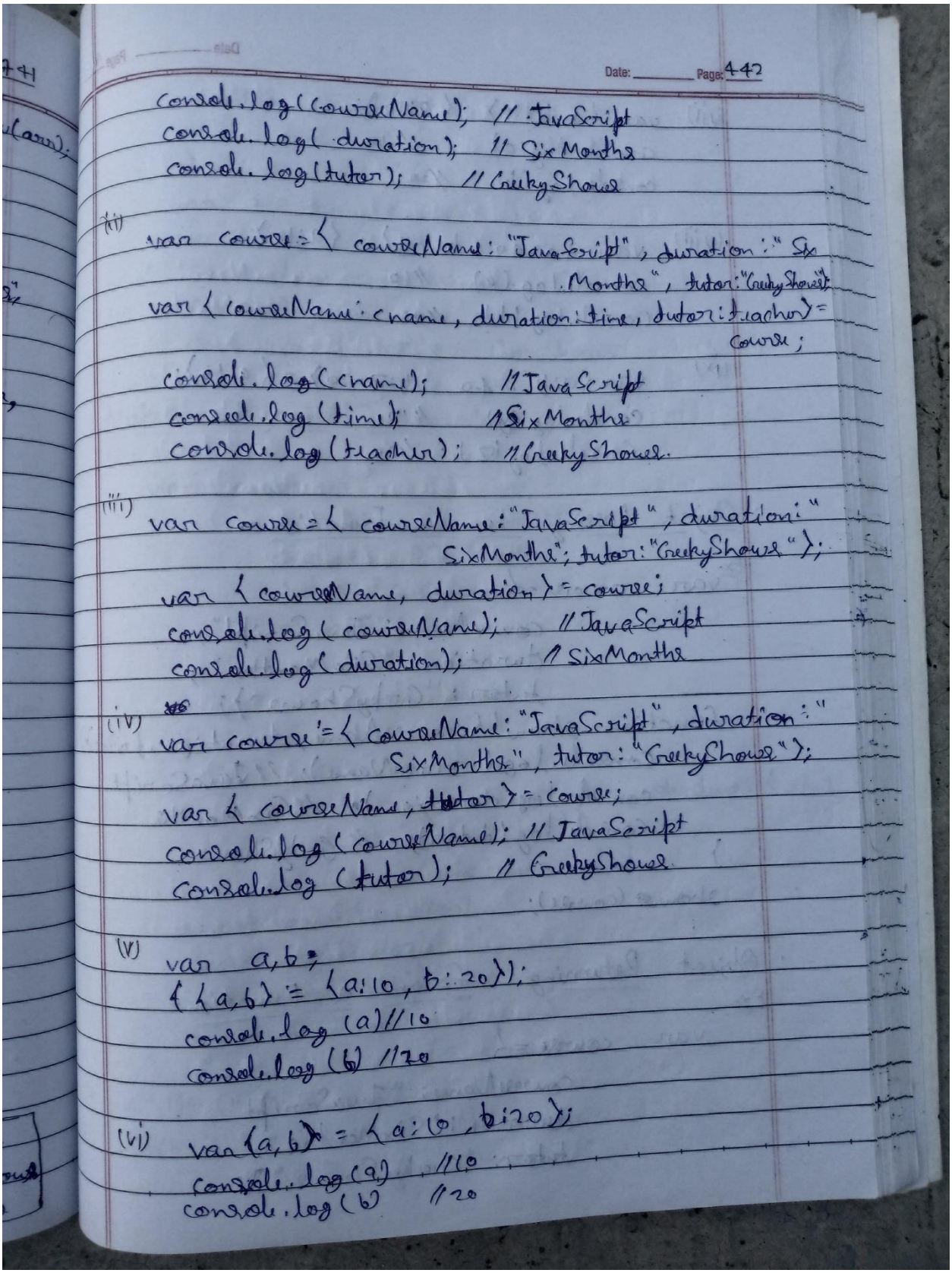
```
var [value1, value2, value3] = show(...);
console.log(value1);
console.log(value2);
console.log(value3);
//Example-6
var a=["JavaScript", "Six Months",
       "GeekyShows", ["Rahul",
                      "Sonam"]];
var [courseName, duration, tutor,
                 [stu1, stu2]] = a;
console.log(courseName);
console.log(duration);
console.log(tutor);
console.log(stu1);
console.log(stu2);
    </script>
  </body>
</html>
```

Object Destructuring -

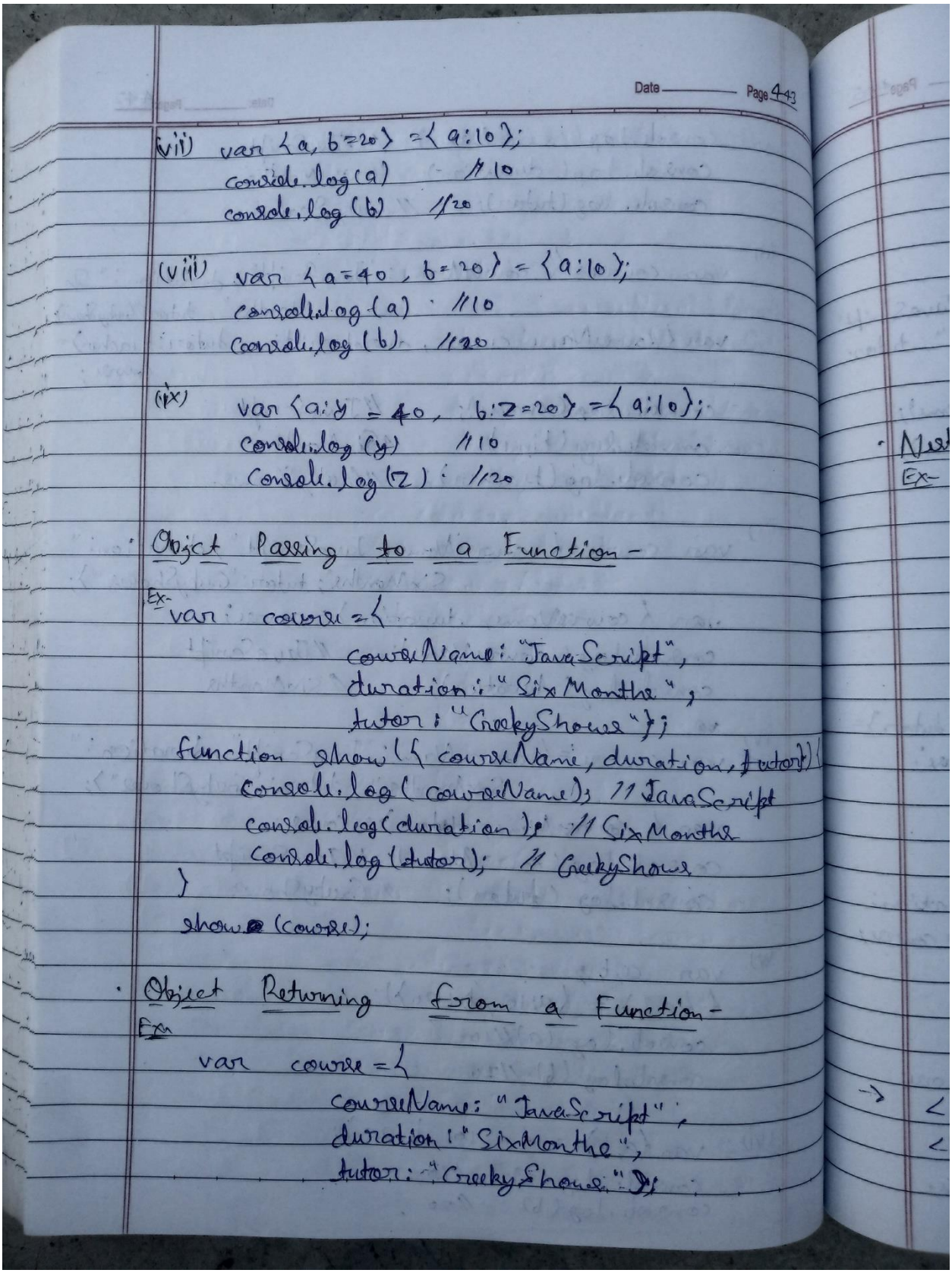
Ex-

```
(i) var course = {
        CourseName: "JavaScript",
        duration: "Six Months",
        tutor: "GeekyShows"};
console.log(course.courseName); // JavaScript
console.log(course.duration); // Six Months
console.log(course.tutor); // GeekyShows
var {courseName, duration, tutor} = course;
```

When destructuring objects, we use the keys as variable names. This how JavaScript knows which property of the object we want to assign.



```
console.log(courseName); // JavaScript
console.log(duration); // Six Months
console.log(tutor); // GeekyShows

(ii)
var course = { courseName: "JavaScript", duration: "Six
                Months", tutor: "GeekyShows" };
var { courseName: cname, duration: time, tutor: teacher } =
                course;

console.log(cname); // JavaScript
console.log(time); // Six Months
console.log(teacher); // GeekyShows.

(iii)
var course = { courseName: "JavaScript", duration: "
            SixMonths", tutor: "GeekyShows" };
var { courseName, duration } = course;
console.log(courseName); // JavaScript
console.log(duration); // SixMonths

(iv)
var course = { courseName: "JavaScript", duration: "
            SixMonths", tutor: "GeekyShows" };
var { courseName, tutor } = course;
console.log(courseName); // JavaScript
console.log(tutor); // GeekyShows

(v)
var a, b;
({a, b} = {a:10, b:20});
console.log(a) //10
console.log(b) //20

(vi)
var {a, b} = {a:10, b:20};
console.log(a) //10
console.log(b) //20
```



(vii) var {a, b=20} = {a:10};
console.log(a) //10
console.log(b) //20

(viii) var {a=40, b=20} = {a:10};
console.log(a) //10
console.log(b) //20

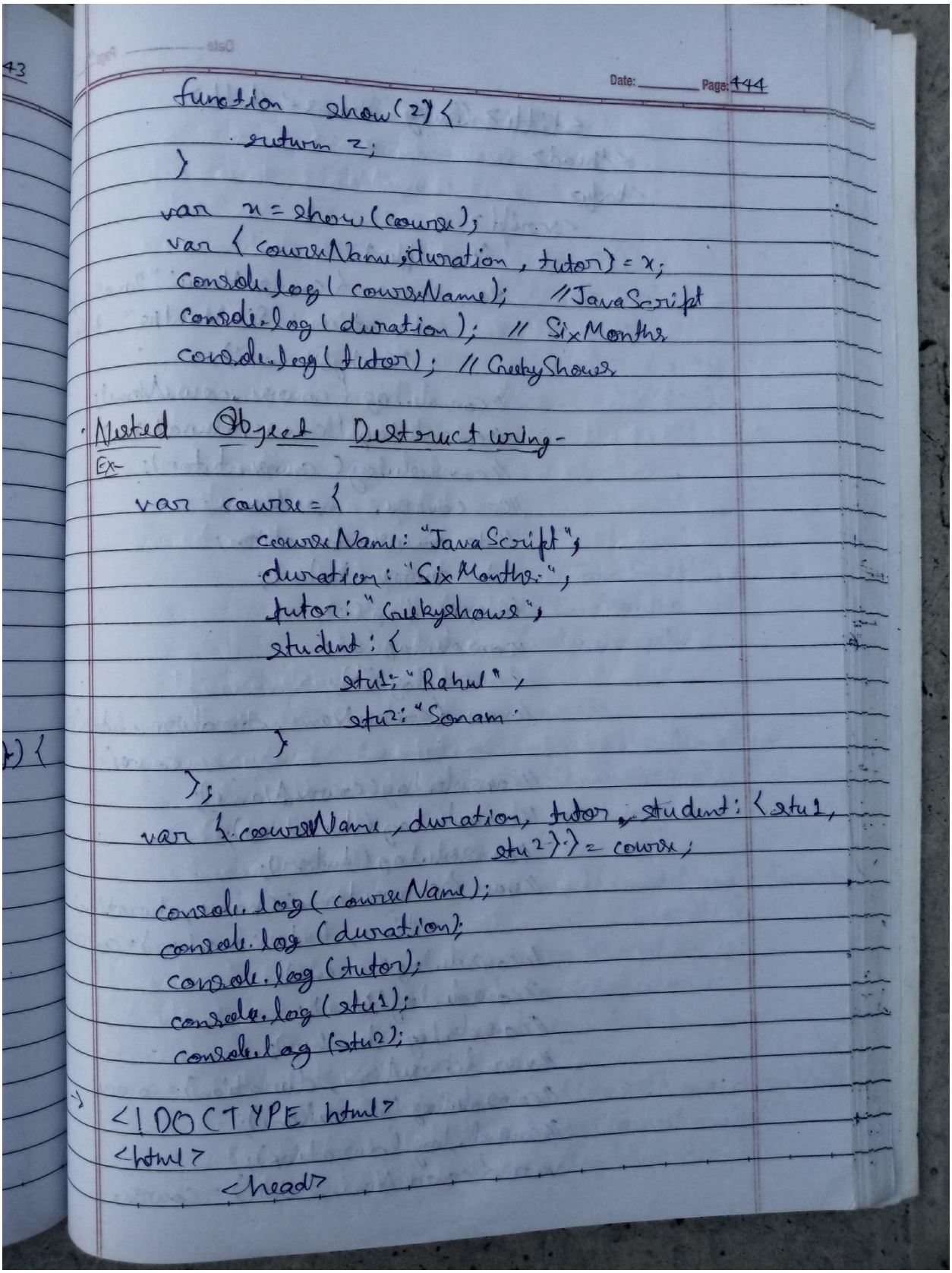
(ix) var {a:y = 40, b:z=20} = {a:10};
console.log(y) //10
console.log(z) //20

· Object Passing to a Function -

Ex- var course = {
courseName: "JavaScript",
duration: "Six Months",
tutor: "GeekyShows"};
function show({courseName, duration, tutor}){
console.log(courseName); // JavaScript
console.log(duration); // Six Months
console.log(tutor); // GeekyShows
}
show(course);

· Object Returning from a Function -

Ex
var course = {
courseName: "JavaScript",
duration: "Six Months",
tutor: "GeekyShows"};



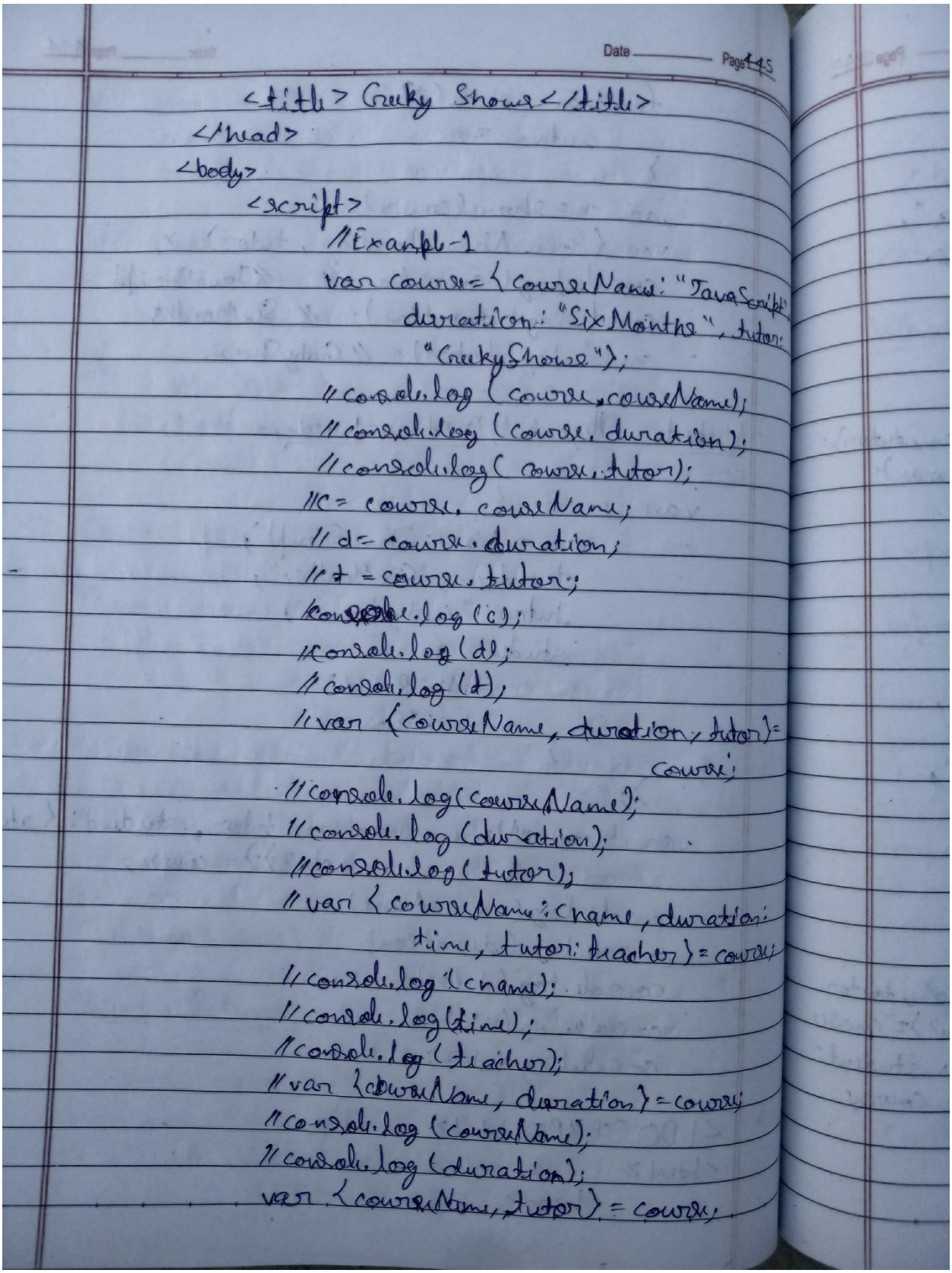
```
function show(z){
    return z;
}
var x = show(course);
var { courseName, duration, tutor } = x;
console.log(courseName); // JavaScript
console.log(duration); // SixMonths
console.log(tutor); // GeekyShows
```

Nested Object Destructuring -

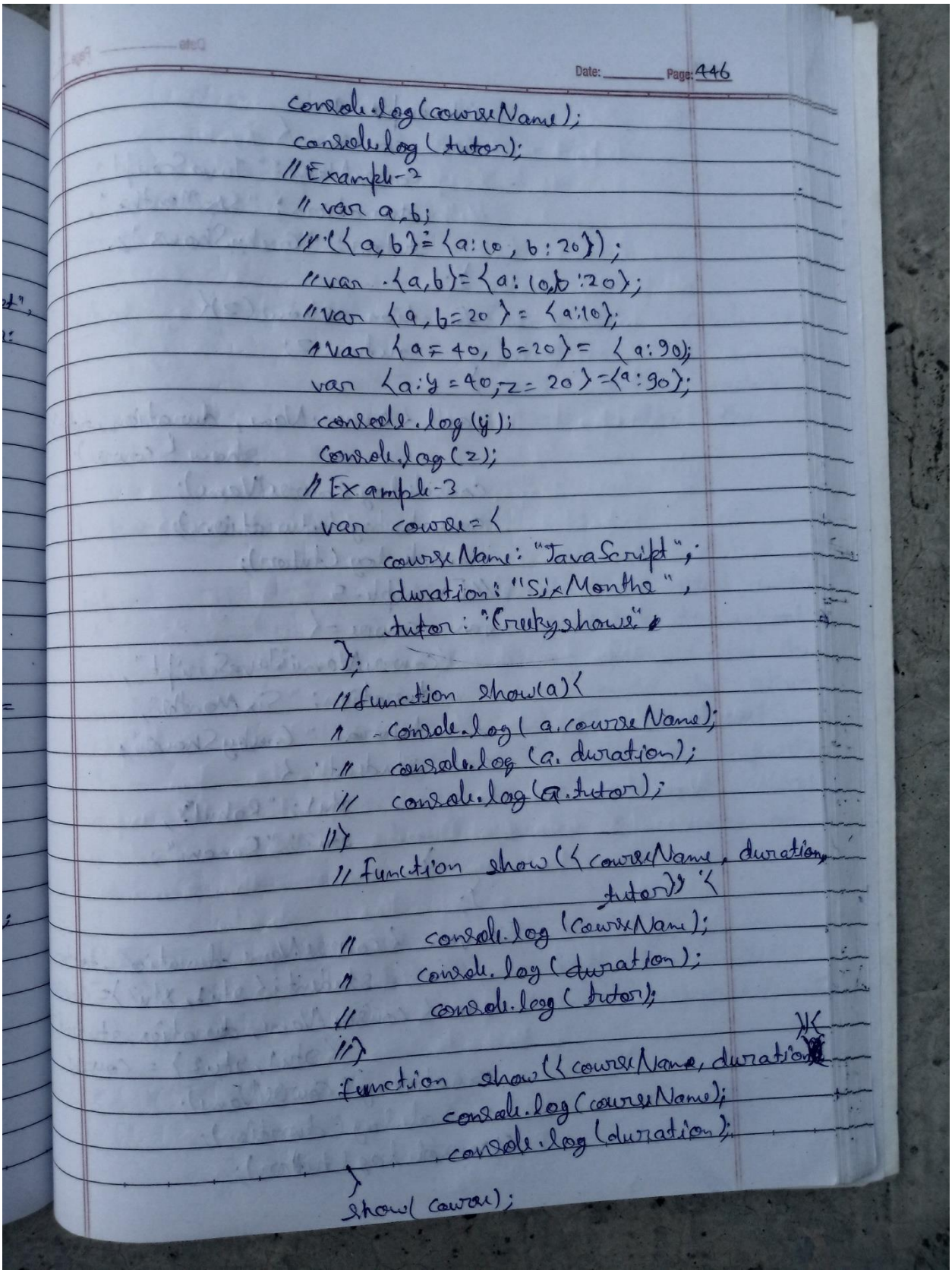
Ex-

```
var course = {
    courseName: "JavaScript",
    duration: "SixMonths",
    tutor: "Geekyshows",
    student: {
        stu1: "Rahul",
        stu2: "Sonam"
    }
};
var { courseName, duration, tutor, student: { stu1, stu2 } } = course;

console.log(courseName);
console.log(duration);
console.log(tutor);
console.log(stu1);
console.log(stu2);
```

```
<!DOCTYPE html>
<html>
    <head>
```

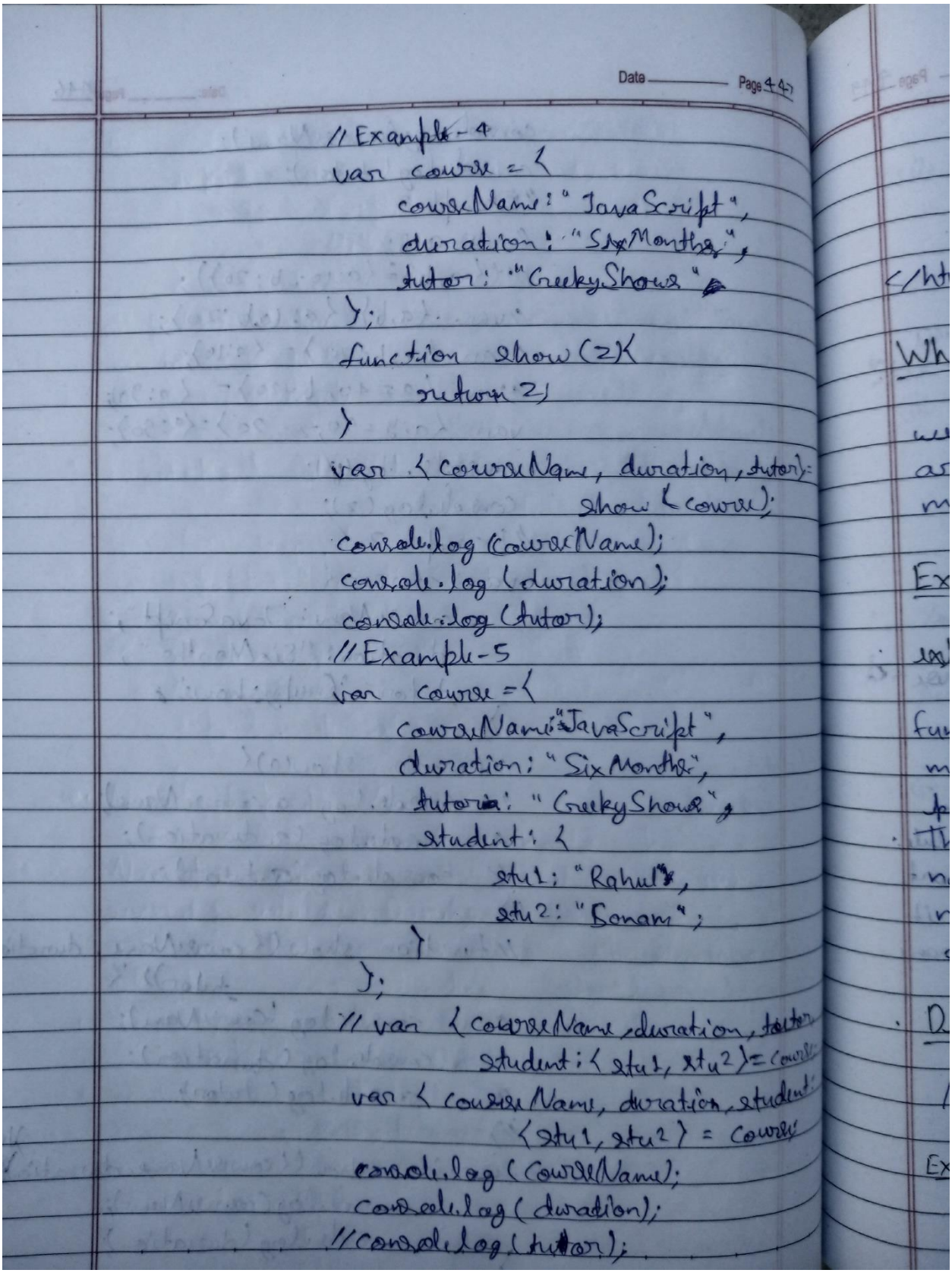


```
<title> Geeky Shows </title>
</head>
<body>
<script>
    //Example-1
    var course = { courseName: "JavaScript",
        duration: "Six Months", tutor:
        "GeekyShows" };
    //console.log ( course, courseName);
    //console.log ( course. duration);
    //console.log ( course. tutor);
    //c = course. courseName;
    //d = course. duration;
    //t = course. tutor;
    //console.log (c);
    //console.log (d);
    //console.log (t);
    //var {courseName, duration, tutor} =
                                course;
    //console.log (courseName);
    //console. log (duration);
    //console.log ( tutor);
    //var { courseName: cname, duration:
            time, tutor: teacher } = course;
    //console.log (cname);
    //console.log (time);
    //console.log ( teacher);
    //var {courseName, duration} = course;
    //console.log (courseName);
    //console.log (duration);
    var {courseName, tutor} = course;
```

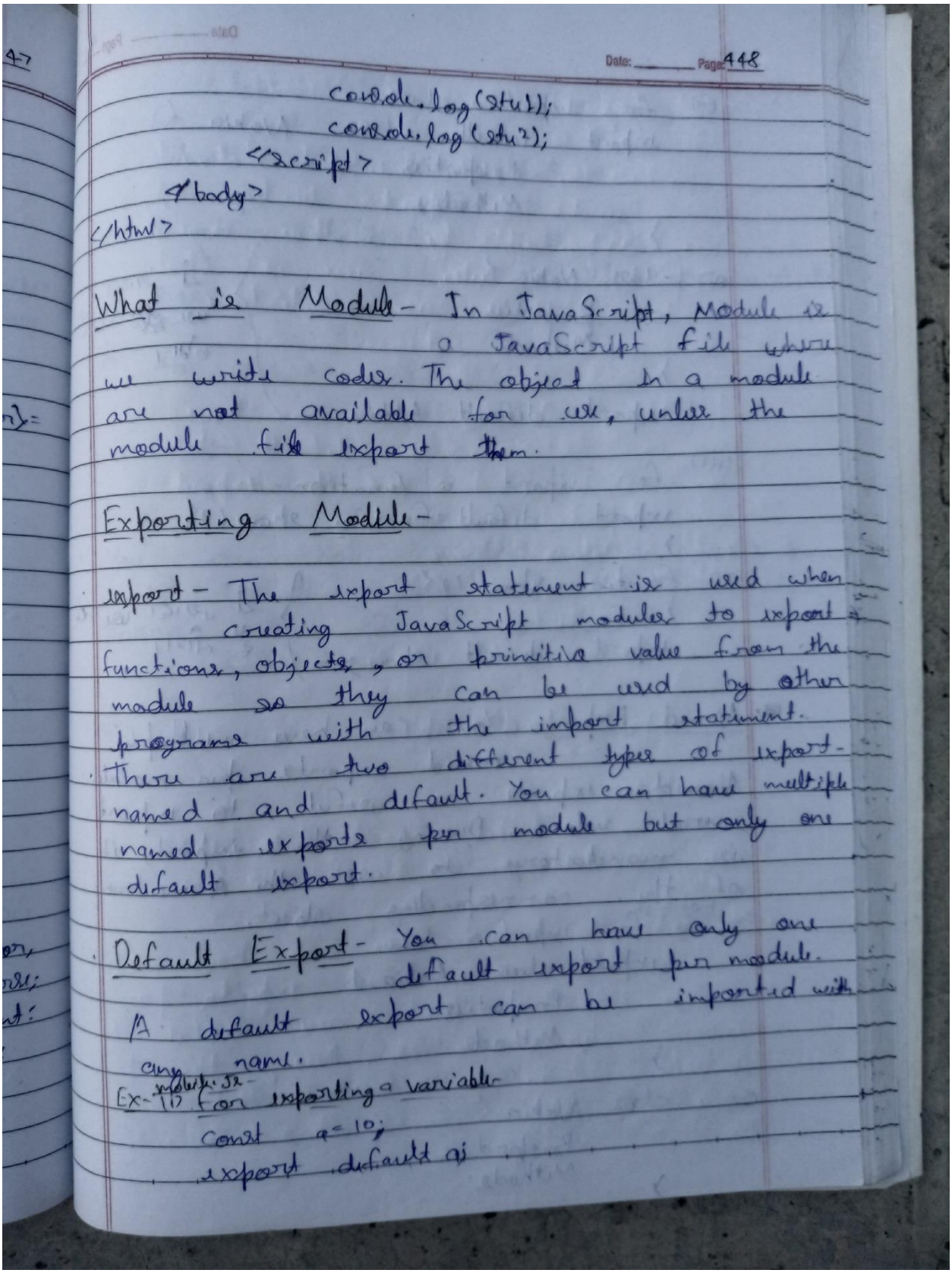


Date: Page: 446

```
console.log(courseName);
console.log(tutor);
// Example-2
// var a,b;
// ({a,b}={a:10,b:20});
// var {a,b}={a:10,b:20};
// var {a,b=20}={a:10};
// var {a=40,b=20}={a:90};
var {a:y=40,z=20}={a:90};
console.log(y);
console.log(z);
// Example-3
var course={
    courseName:"JavaScript",
    duration:"SixMonths",
    tutor:"Geekyshows"
};
// function show(a){
//     console.log(a.courseName);
//     console.log(a.duration);
//     console.log(a.tutor);
// }
// function show({courseName,duration,
                tutor}){
//     console.log(courseName);
//     console.log(duration);
//     console.log(tutor);
// }
function show({courseName,duration}){
    console.log(courseName);
    console.log(duration);
}
show(course);
```



```
// Example-4
var course = {
    courseName: "JavaScript",
    duration: "Six Months",
    tutor: "GeekyShows"
};
function show(z){
    return z;
}
var { courseName, duration, tutor } =
                show(course);
console.log(courseName);
console.log(duration);
console.log(tutor);
// Example-5
var course = {
    courseName: "JavaScript",
    duration: "Six Months",
    tutorial: "GeekyShows",
    student: {
        stu1: "Rahul",
        stu2: "Sonam";
    }
};
// var { courseName, duration, tutor
        student: { stu1, stu2 } = course;
var { courseName, duration, student:
            { stu1, stu2 } = course;
console.log(courseName);
console.log(duration);
//console.log(tutor);
```



```
        console.log(stu1);
        console.log(stu2);
    </script>
</body>
</html>
```

What is Module - In JavaScript, Module is a JavaScript file where we write codes. The objects in a module are not available for use, unless the module file export them.

Exporting Module -

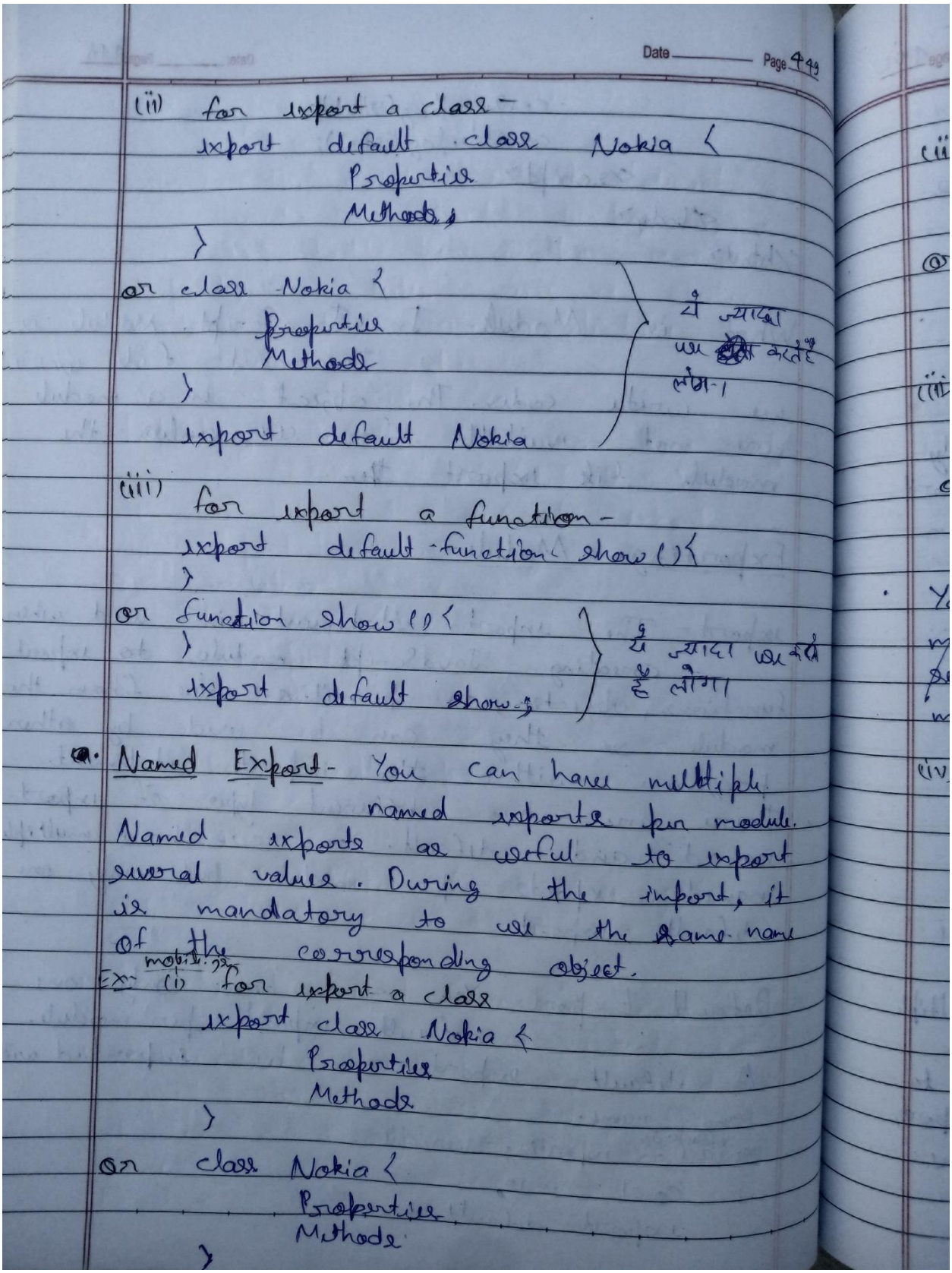
export - The export statement is used when creating JavaScript modules to export functions, objects, or primitive value from the module so they can be used by other programs with the import statement. There are two different types of export - named and default. You can have multiple named exports per module but only one default export.

Default Export - You can have only one default export per module. A default export can be imported with any name.

Ex- index.js

// For exporting a variable

```
const a = 10;
export default a;
```



(ii) for export a class -

```
export default class Nokia {
    Properties
    Methods;
}
```

or

```
class Nokia {
    Properties
    Methods
}
export default Nokia
```

} ये ज्यादा use करते हैं लोग।

(iii) for export a function -

```
export default function show(){
}
```

or

```
function show(){
}
export default show;
```

} ये ज्यादा use करते हैं लोग।

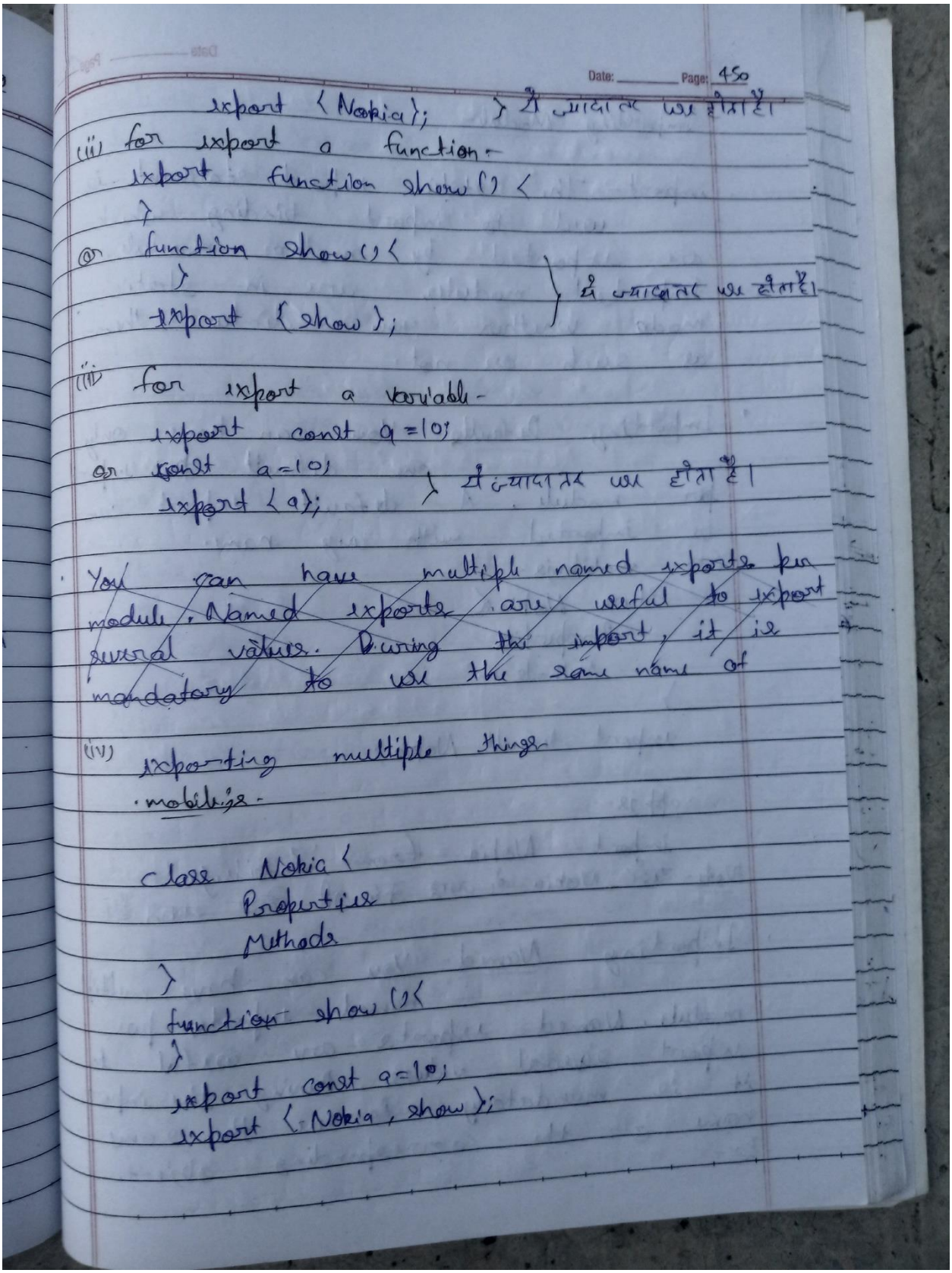
2. Named Export - You can have multiple named exports per module. Named exports are useful to export several values. During the import, it is mandatory to use the same name of the corresponding object.

Ex: (i) for export a class

```
export class Nokia {
    Properties
    Methods
}
```

or

```
class Nokia {
    Properties
    Methods
}
```



Date: Page: 450

export {Nokia}; } ये ज्यादातर use होता है।

(ii) for export a function-

export function show() {
}

or function show() {
}
export {show}; } ये ज्यादातर use होता है।

(iii) for export a variable-

export const a=10;

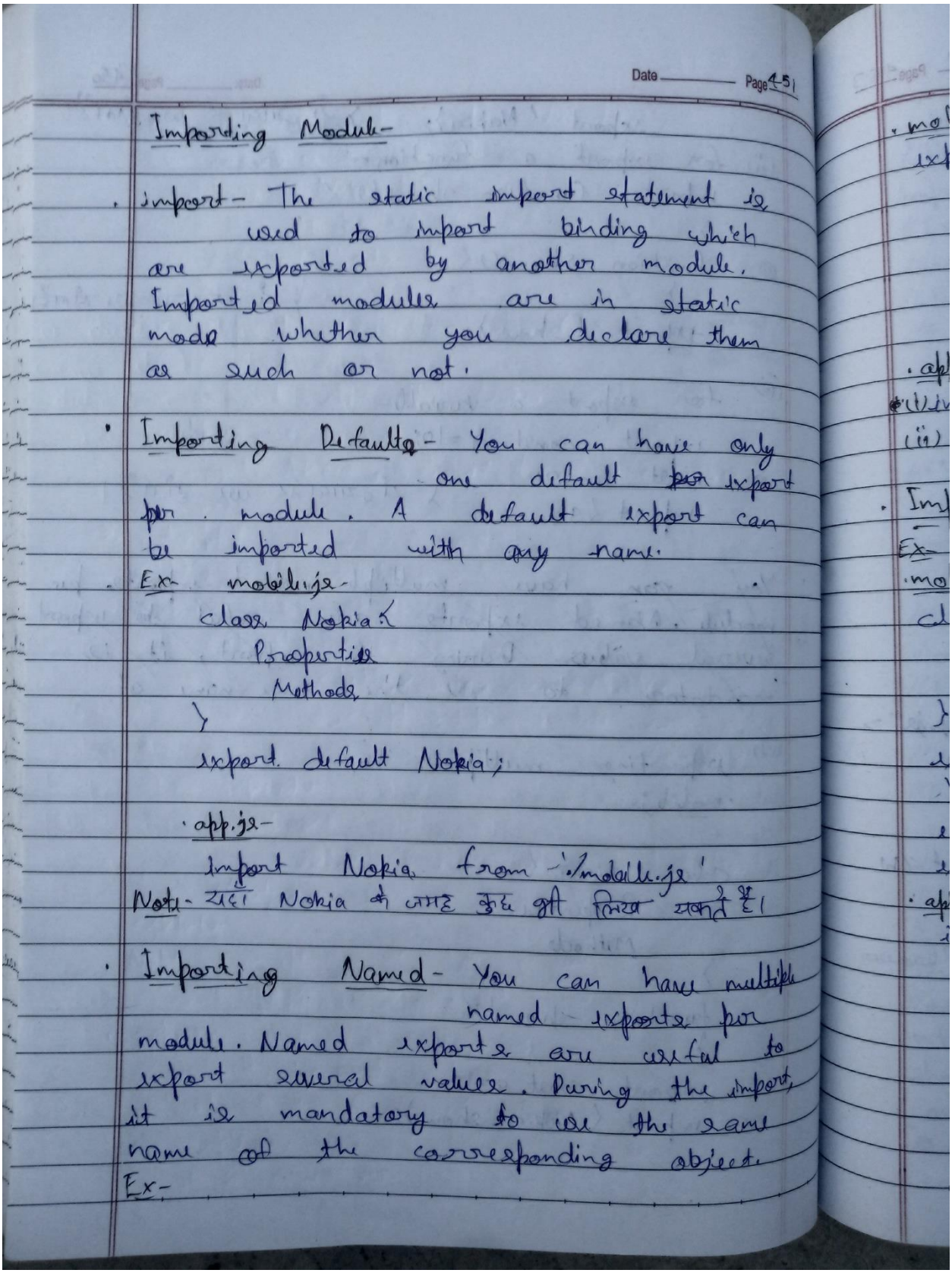
or const a=10;
export {a}; } ये ज्यादातर use होता है।

You can have multiple named exports in module. Named exports are useful to export several values. During the import, it is mandatory to use the same name of

(iv) exporting multiple things-

mobile.js-

class Nokia {
Properties
Methods
}
function show() {
}
export const a=10;
export {Nokia, show};



Importing Module-

- import - The static import statement is used to import binding which are exported by another module. Imported modules are in static mode whether you declare them as such or not.

- Importing Defaults - You can have only one default export per module. A default export can be imported with any name.

Ex- mobile.js-

```
class Nokia {
    Properties
    Methods
}
export default Nokia;
```

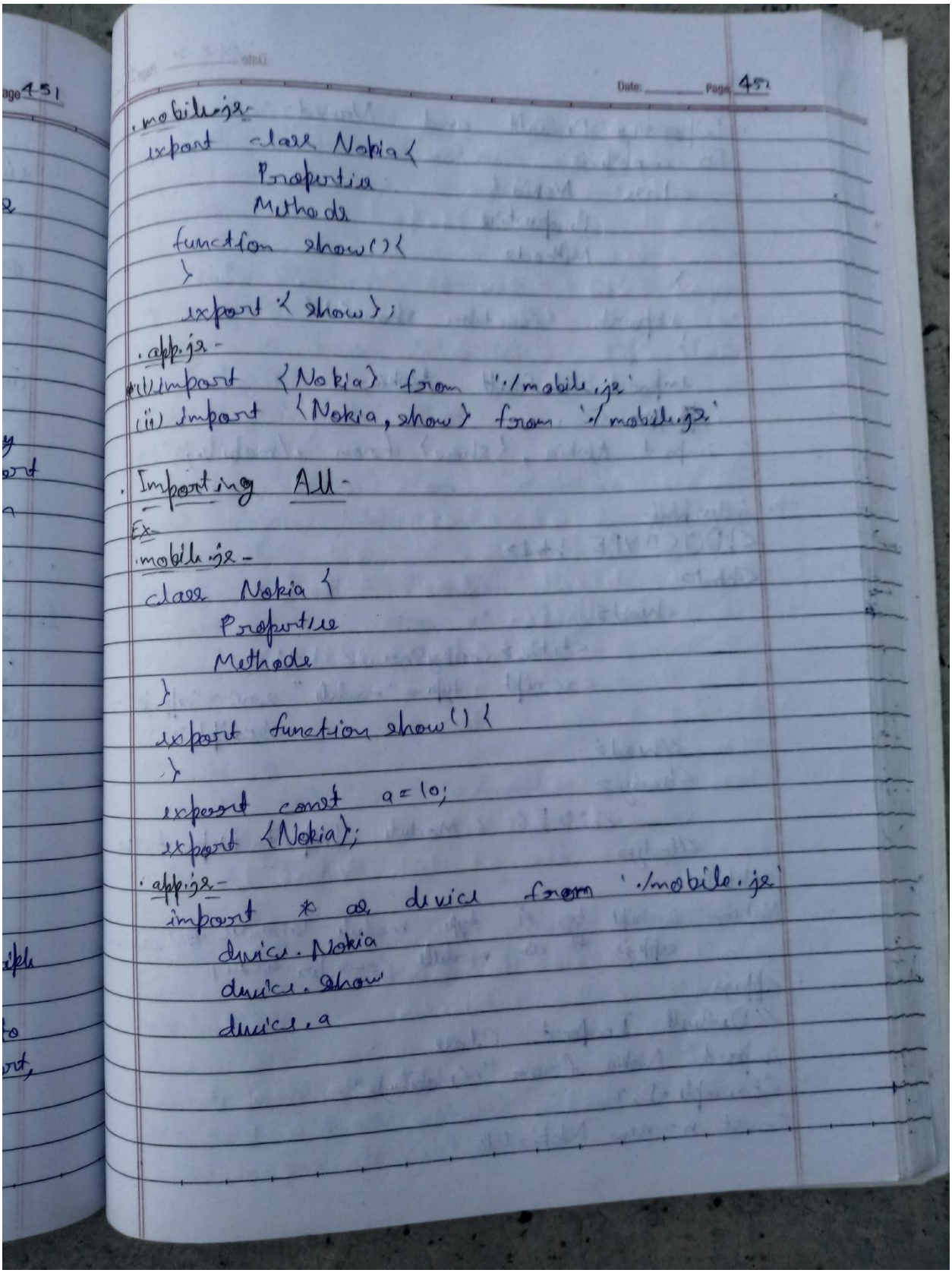
app.js-

```
import Nokia from './mobile.js'
```

Note- यहां Nokia के जगह कुछ भी लिख सकते हैं।

- Importing Named - You can have multiple named exports per module. Named exports are useful to export several values. During the import, it is mandatory to use the same name of the corresponding object.

Ex-



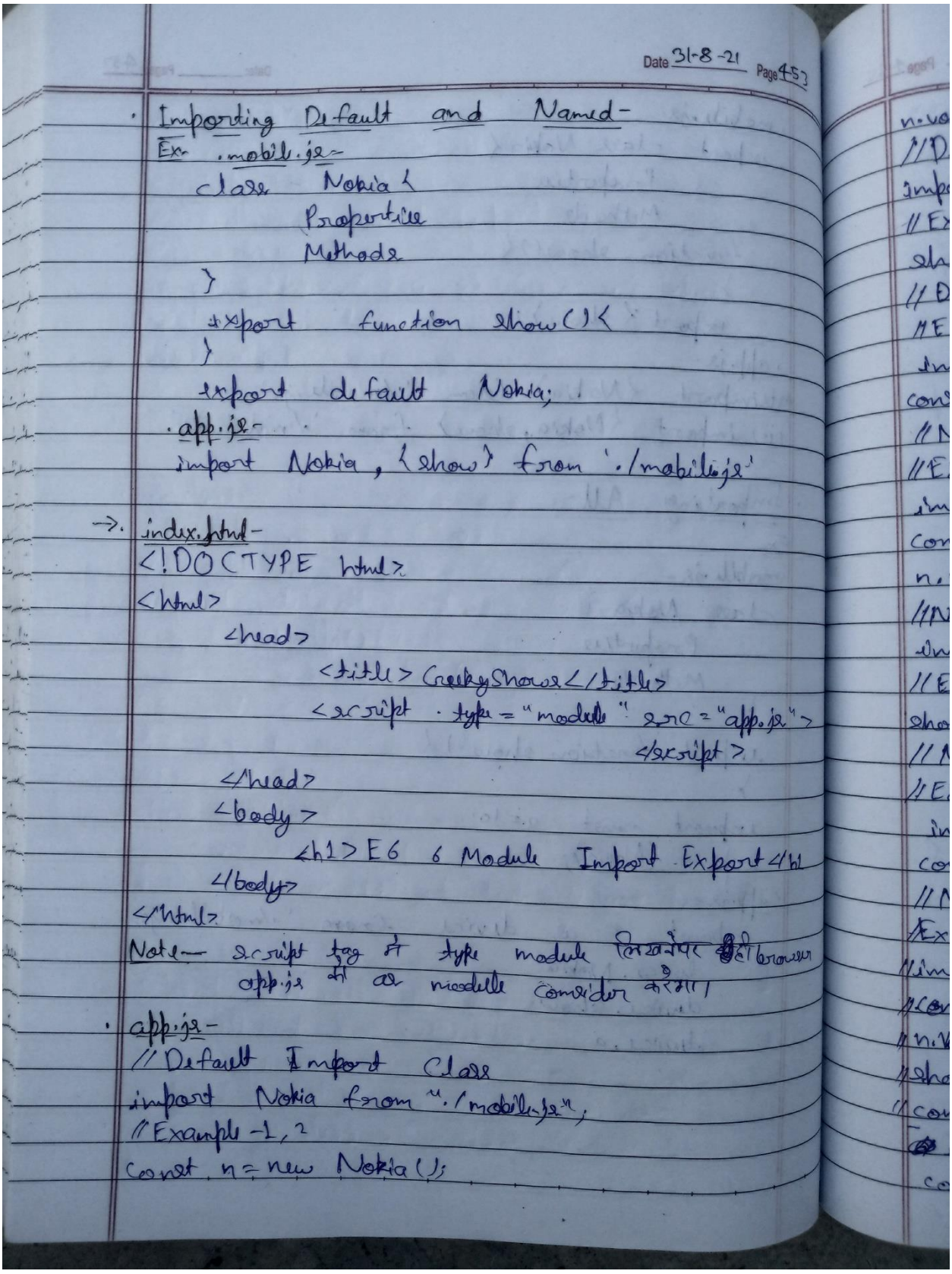
Date: Page 45

• mobile.js -
export class Nokia {
Properties
Methods
function show() {
}
export { show };

• app.js -
(i) import {Nokia} from './mobile.js'
(ii) import {Nokia, show} from './mobile.js'

• Importing All -
Ex-
• mobile.js -
class Nokia {
Properties
Methods
}
export function show() {
}
export const a = 10;
export {Nokia};

• app.js -
import * as device from './mobile.js'
device.Nokia
device.show
device.a



Date 31-8-21 Page 453

- Importing Default and Named-

Ex. mobile.js-

```
class Nokia {
    Properties
    Methods
}
export function show() {
}
export default Nokia;
```

app.js-

```
import Nokia, {show} from './mobile.js'
```

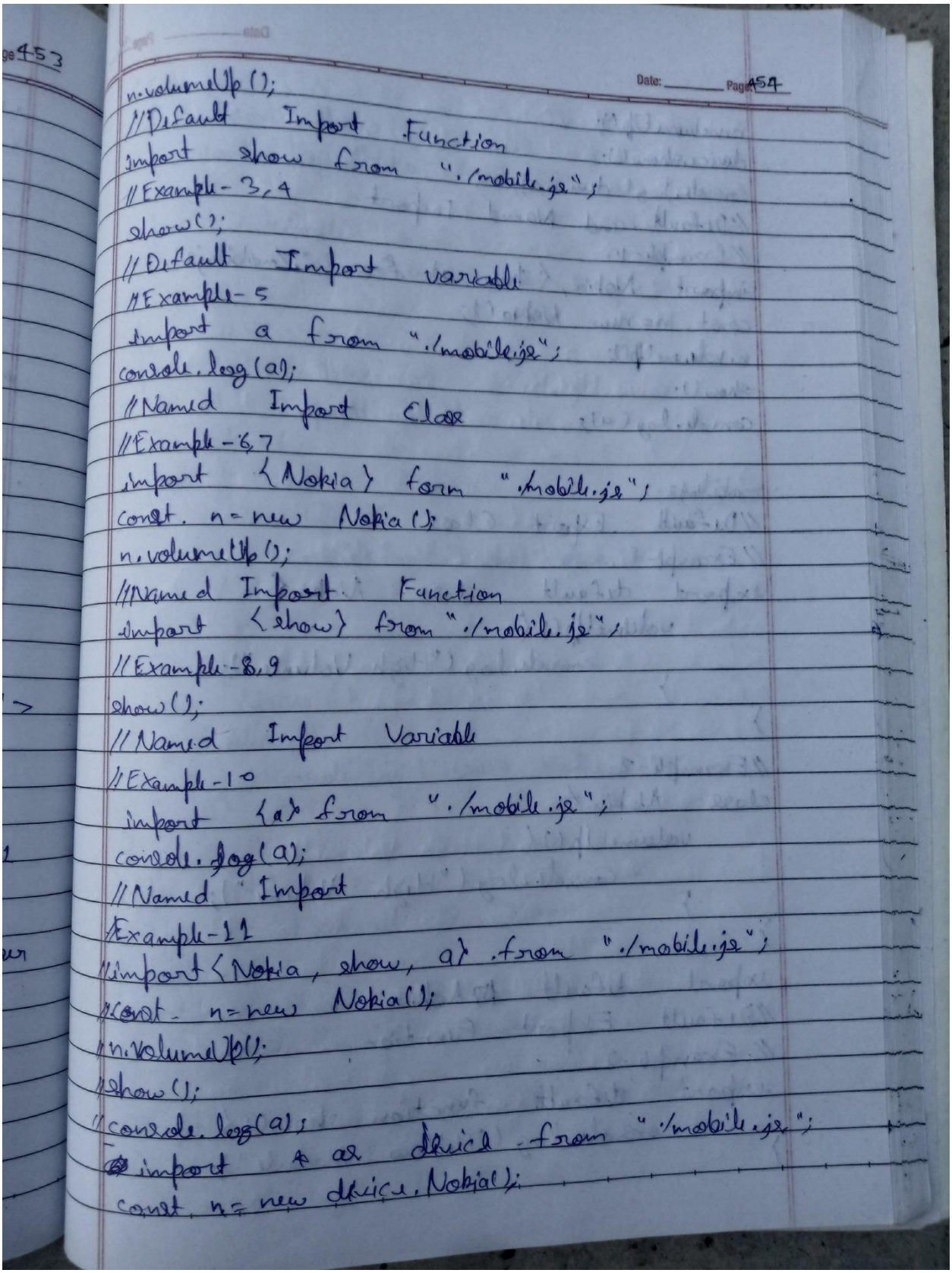
→. index.html-

```
<!DOCTYPE html>
<html>
    <head>
        <title> GeekyShows </title>
        <script type="module" src="app.js">
        </script>
    </head>
    <body>
        <h1> E6 : Module Import Export </h1>
    </body>
</html>
```

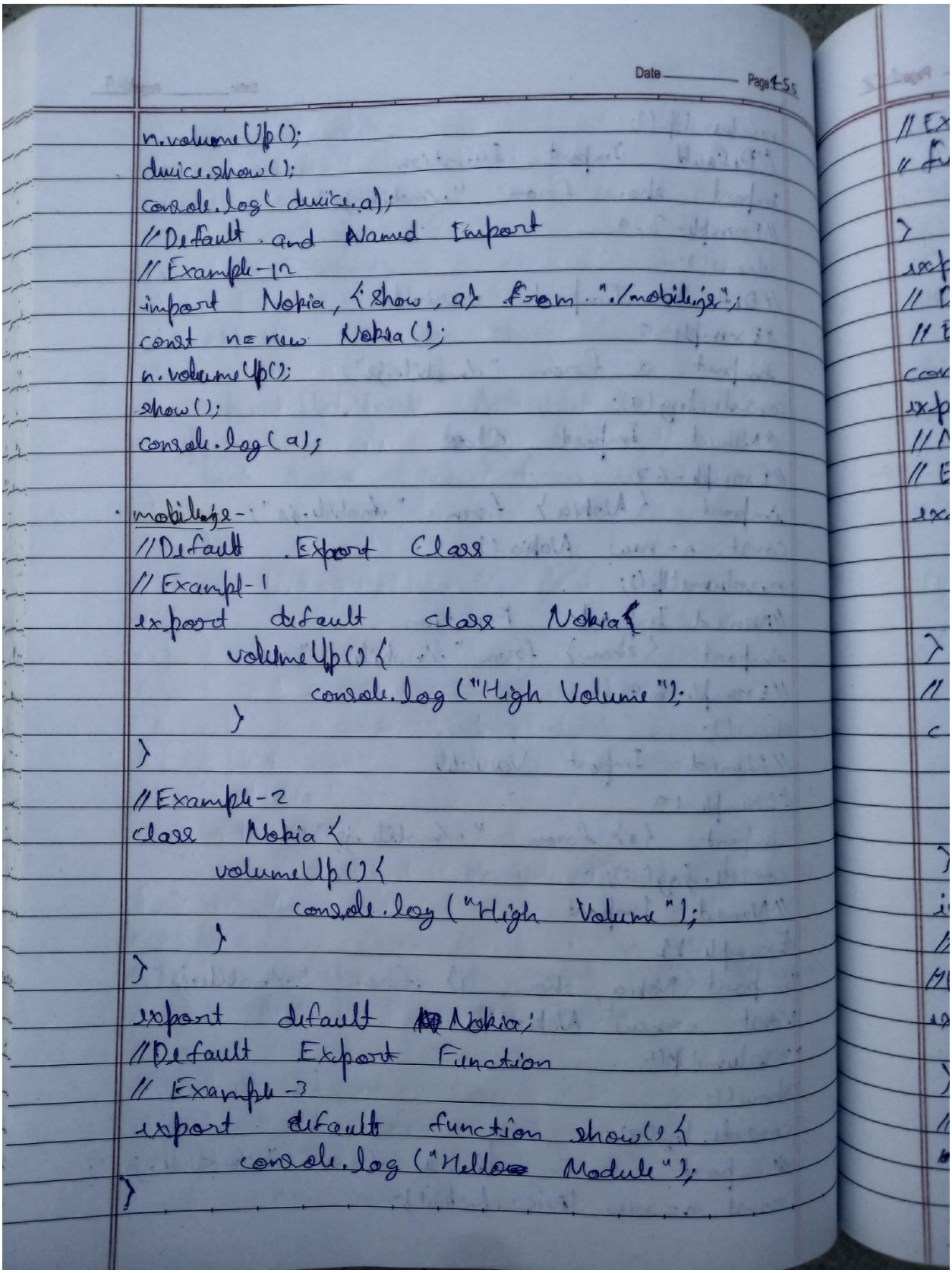
Note— script tag में type module लिखने पर browser app.js को as module consider करेगा।

- app.js-

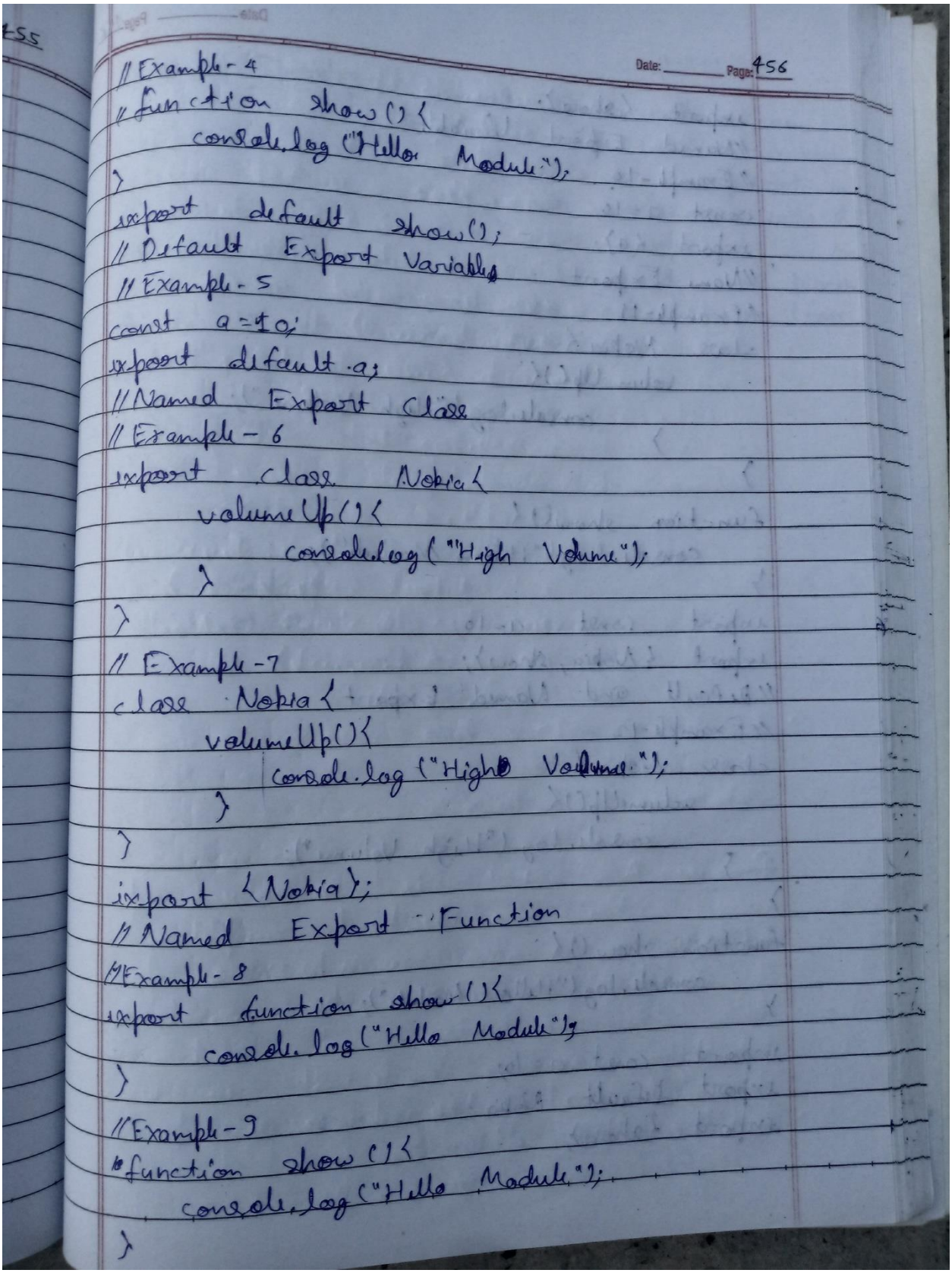
```
// Default Import Class
import Nokia from "./mobile.js";
// Example -1, 2
const n = new Nokia();
```



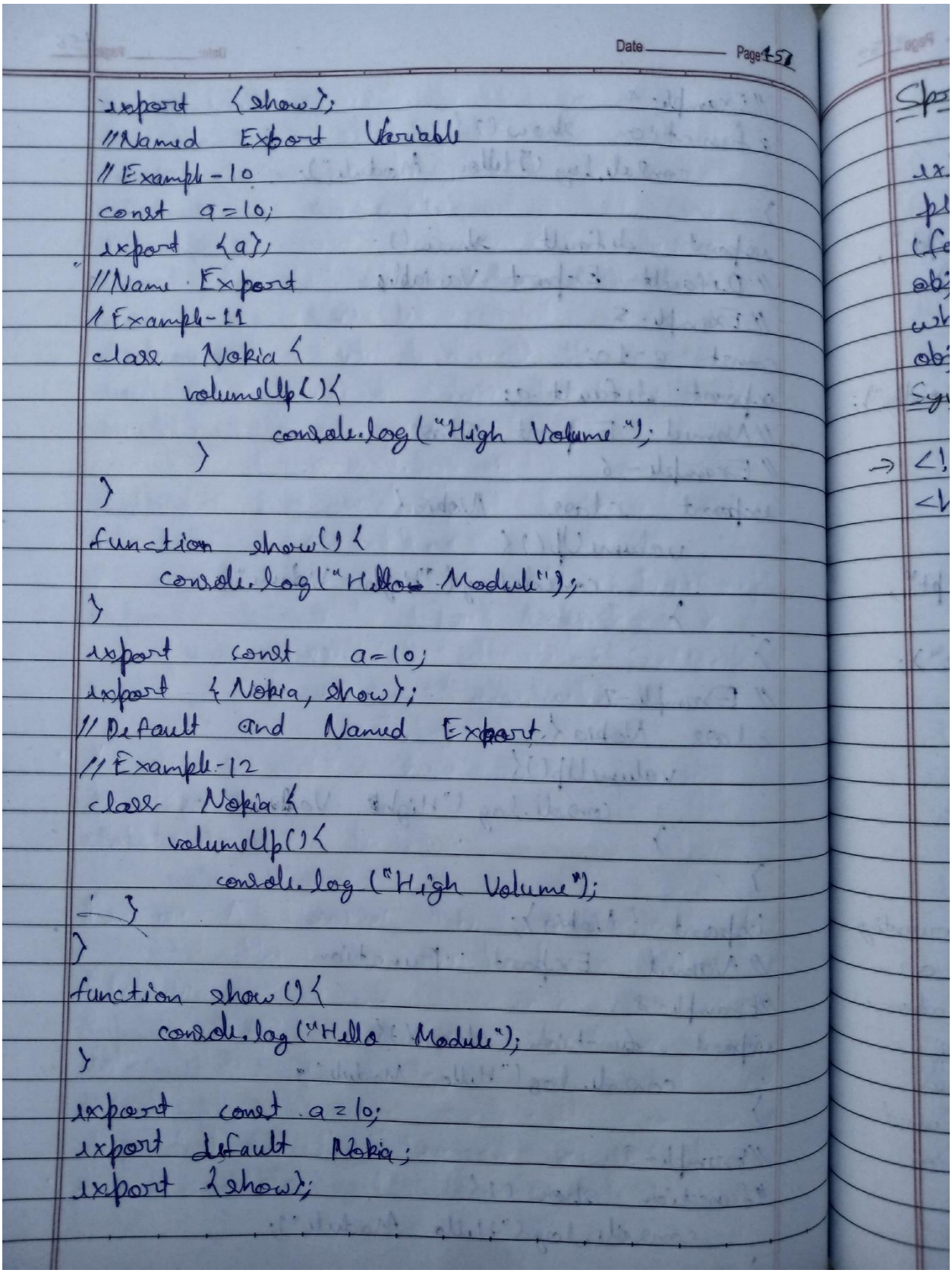
```
n.volumeUp();
//Default Import Function
import show from "./mobile.js";
//Example-3,4
show();
//Default Import variable
//Example-5
import a from "./mobile.js";
console.log(a);
//Named Import Class
//Example-6,7
import {Nokia} form "./mobile.js";
const n=new Nokia();
n.volumeUp();
//Named Import Function
import {show} from "./mobile.js";
//Example-8,9
show();
//Named Import Variable
//Example-10
import {a} from "./mobile.js";
console.log(a);
//Named Import
Example-11
//import {Nokia, show, a} from "./mobile.js";
//const n=new Nokia();
//n.volumeUp();
//show();
//console.log(a);
import * as device from "./mobile.js";
const n=new device.Nokia();
```



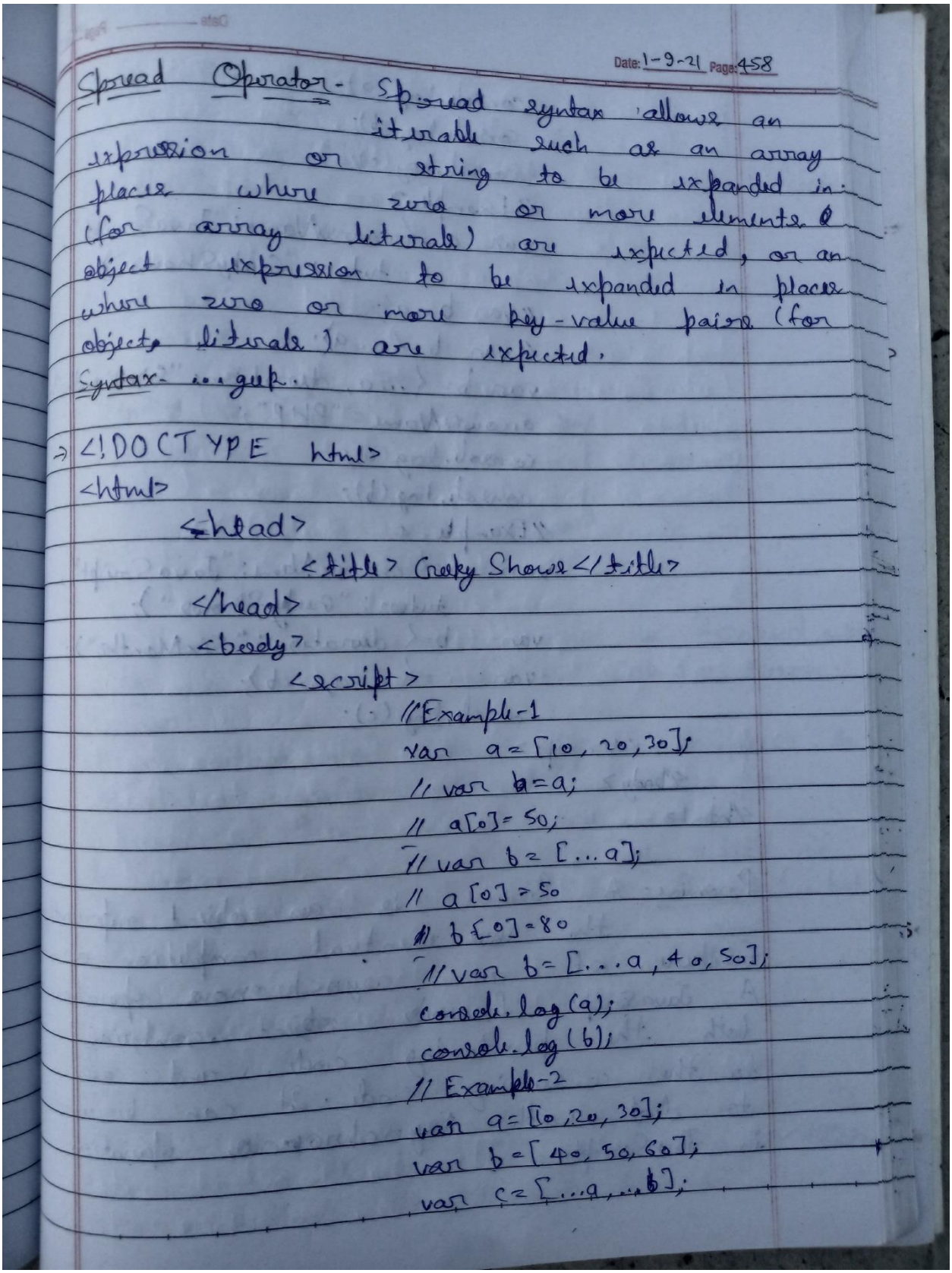
```
n.volumeUp();
device.show();
console.log(device.a);
// Default and Named Import
// Example-12
import Nokia, {show, a} from "./mobile12";
const n = new Nokia();
n.volumeUp();
show();
console.log(a);

mobile12-
// Default Export Class
// Example-1
export default class Nokia{
    volumeUp(){
        console.log("High Volume");
    }
}
// Example-2
class Nokia{
    volumeUp(){
        console.log("High Volume");
    }
}
export default Nokia;
// Default Export Function
// Example-3
export default function show(){
    console.log("Hello Module");
}
```



```
// Example - 4
// function show(){
    console.log("Hello Module");
}
export default show();
// Default Export Variable
// Example - 5
const a = 10;
export default a;
// Named Export Class
// Example - 6
export class Nokia{
    volumeUp(){
        console.log("High Volume");
    }
}
// Example - 7
class Nokia{
    volumeUp(){
        console.log("High Volume");
    }
}
export {Nokia};
// Named Export Function
// Example - 8
export function show(){
    console.log("Hello Module");
}
// Example - 9
function show(){
    console.log("Hello Module");
}
```



```
export {show};
//Named Export Variable
//Example-10
const a=10;
export {a};
//Name Export
//Example-11
class Nokia {
    volumeUp(){
        console.log("High Volume");
    }
}
function show(){
    console.log("Hello Module");
}
export const a=10;
export {Nokia, show};
//Default and Named Export
//Example-12
class Nokia {
    volumeUp(){
        console.log("High Volume");
    }
}
function show(){
    console.log("Hello Module");
}
export const a=10;
export default Nokia;
export {show};
```

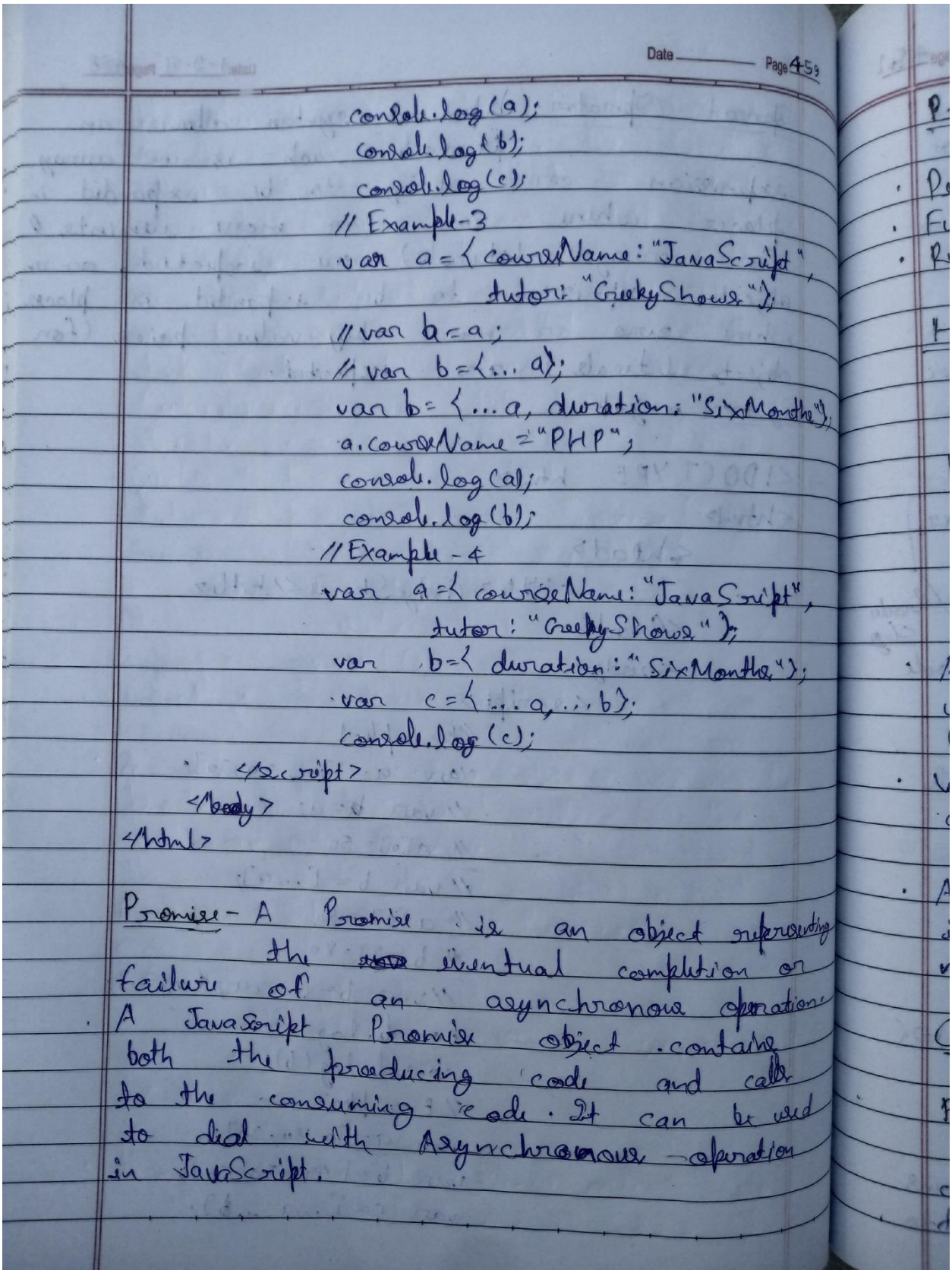


Date: 1-9-21 Page: 458

Spread Operator - Spread syntax allows an iterable such as an array expression or string to be expanded in places where zero or more elements (for array literals) are expected, or an object expression to be expanded in places where zero or more key-value pairs (for object literals) are expected.

Syntax - ...geek.

```
<!DOCTYPE html>
<html>
    <head>
        <title> Geeky Shows </title>
    </head>
    <body>
        <script>
            //Example-1
            var a = [10, 20, 30];
            // var b=a;
            // a[0]= 50;
            // var b = [...a];
            // a[0] = 50
            // b[0]=80
            // var b=[...a, 40, 50];
            console.log(a);
            console.log(b);
            // Example-2
            var a=[10,20,30];
            var b=[40, 50, 60];
            var c=[...a,...b];
```



```
console.log(a);
console.log(b);
console.log(c);
// Example-3
var a={courseName: "JavaScript",
       tutor: "GeekyShows"};
// var b=a;
// var b={...a};
var b={...a, duration: "SixMonths"};
a.courseName="PHP";
console.log(a);
console.log(b);
// Example-4
var a={courseName: "JavaScript",
       tutor: "GeekyShows"};
var b={duration: "SixMonths"};
var c={...a, ...b};
console.log(c);
</script>
</body>
</html>
```

Promise - A Promise is an object representing the eventual completion or failure of an asynchronous operation. A JavaScript Promise object contains both the producing code and calls to the consuming code. It can be used to deal with Asynchronous operation in JavaScript.

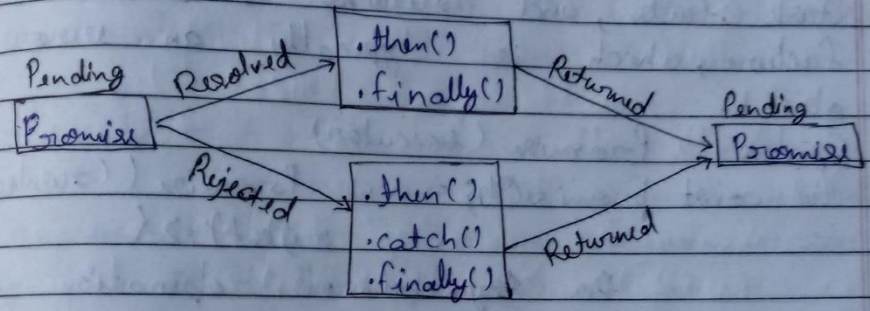
459

Date: Page: 460

Promise State -

- Pending - Initial State, Not yet Fulfilled or Rejected.
- Fulfilled/Resolved - Promise Completed
- Rejected - Promise Failed

How Promise Works -



- A pending promise can either be Resolved with a value or Rejected with a reason (error).
- When either of these options happens, the associated handlers queued up by a promise's then method are called.
- A promise is said to be settled if it is either Resolved or Rejected, but not Pending.

Creating Promise -

Promise() - A Promise object is created using the new keyword and its constructor. This constructor takes a function, called the "executor function", as its

parameter. This function should take two functions as parameters. The first of these function (resolve) is called when the asynchronous task completes successfully and returns the results of the task as a value. The second (reject) is called when the task fails, and returns the reason for failure, which is typically an error object.

Syntax - (i) Promise (executor)

(ii) const promiseObj = new Promise ((resolve, reject) => {
    Do Asynchronous operation
    resolve(value)
    reject(Error)
});

Producing Code

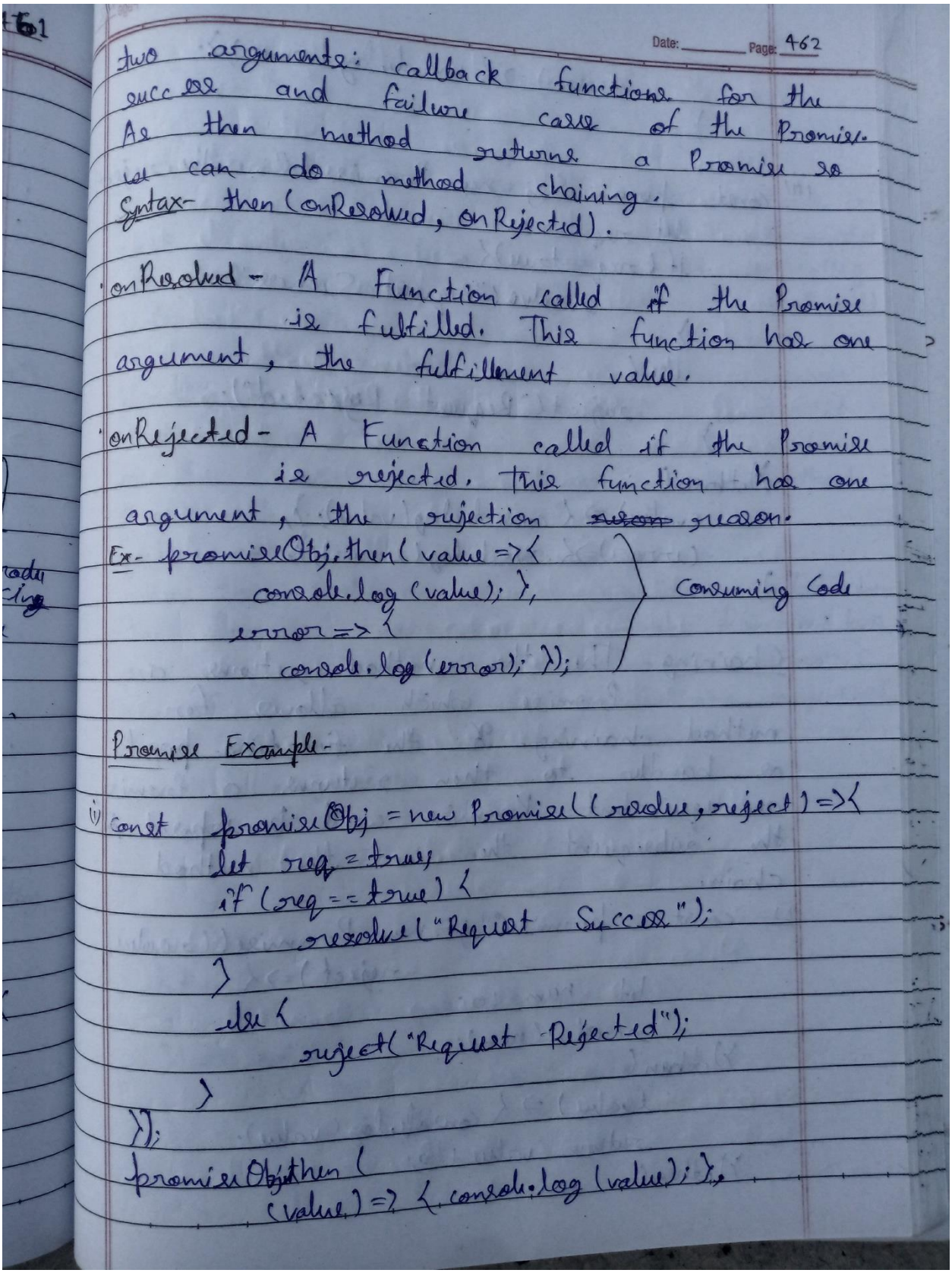
- A JavaScript Promise object contains the producing code and calls to the consuming code.

Function Returning a Promise -

Syntax -

```
function myFunction() {
    return new Promise ((resolve, reject) => {
    });
}
```

- then() Method - The then() method returns a Promise. It takes up to



two arguments: callback functions for the success and failure case of the Promise. As then method returns a Promise so we can do method chaining.

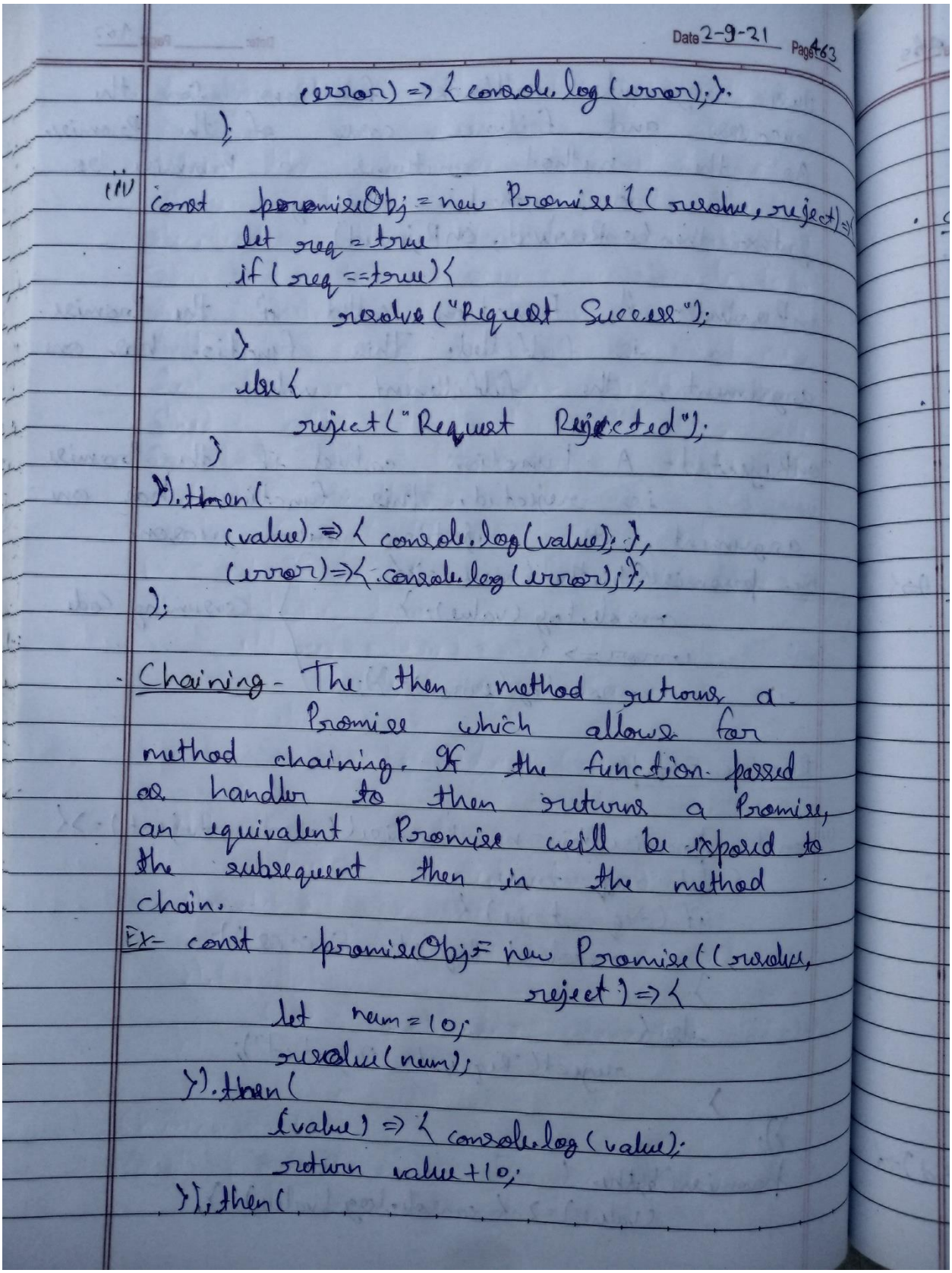
Syntax- then(onResolved, onRejected).

- onResolved - A Function called if the Promise is fulfilled. This function has one argument, the fulfillment value.
- onRejected - A Function called if the Promise is rejected. This function has one argument, the rejection reason.

Ex-
```
promiseObj.then(value => {
    console.log(value); },
error => {
    console.log(error); });
```
} Consuming code

Promise Example-

i)
```
const promiseObj = new Promise((resolve, reject) => {
    let req = true;
    if (req == true) {
        resolve("Request Success");
    }
    else {
        reject("Request Rejected");
    }
});
promiseObj.then(
    (value) => { console.log(value); });
```

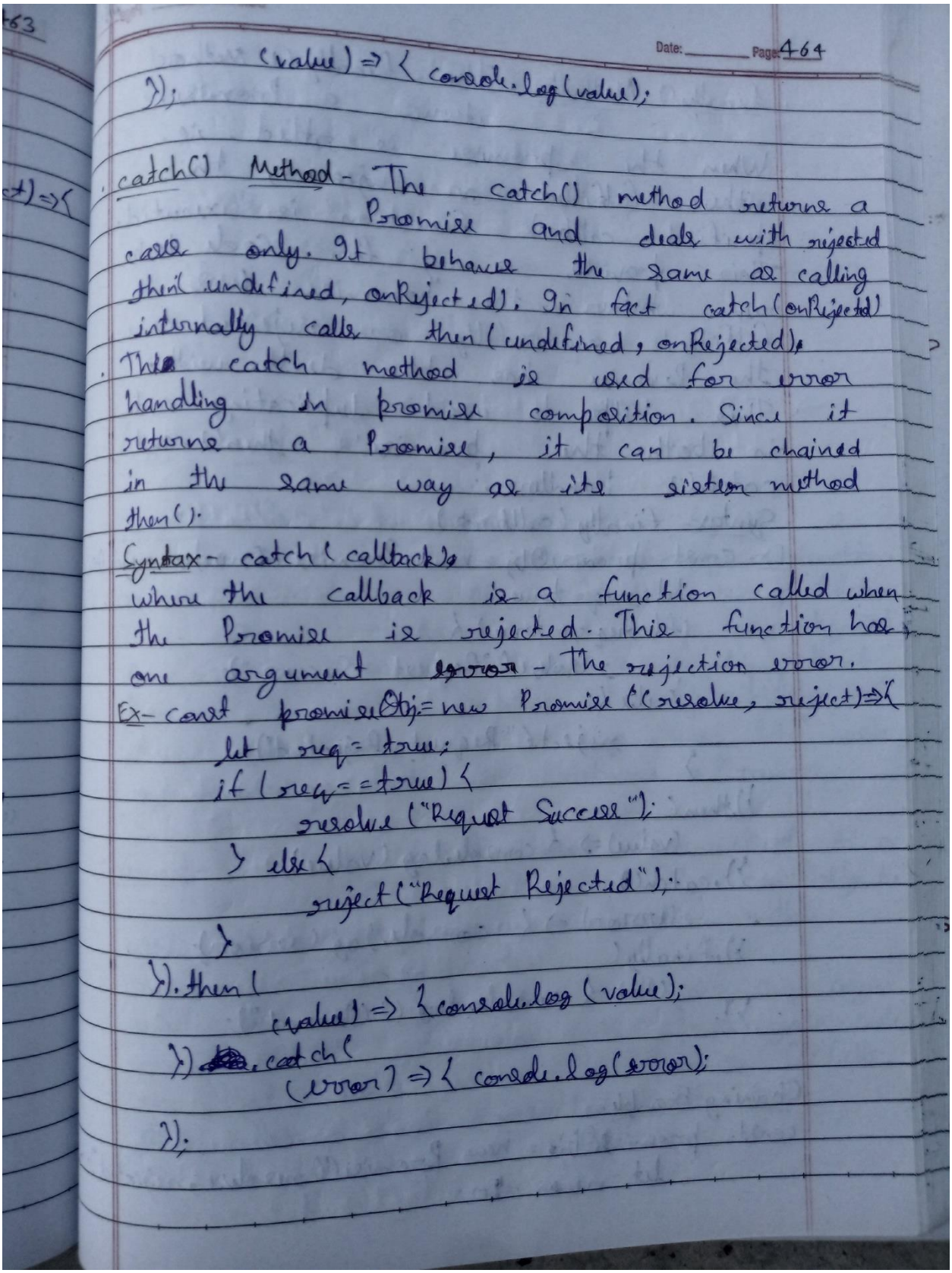


Date 2-9-21 Page 163

```
        (error) => { console.log(error); }
    );

(iii) const promiseObj = new Promise((resolve, reject) => {
        let req = true
        if(req == true){
            resolve("Request Success");
        }
        else{
            reject("Request Rejected");
        }
    }).then(
        (value) => { console.log(value); },
        (error) => { console.log(error); },
    );
```

Chaining - The then method returns a Promise which allows for method chaining. If the function passed as handler to then returns a Promise, an equivalent Promise will be exposed to the subsequent then in the method chain.

Ex- const promiseObj = new Promise((resolve, reject) => {
        let num = 10;
        resolve(num);
    }).then(
        (value) => { console.log(value);
        return value + 10;
    }).then(

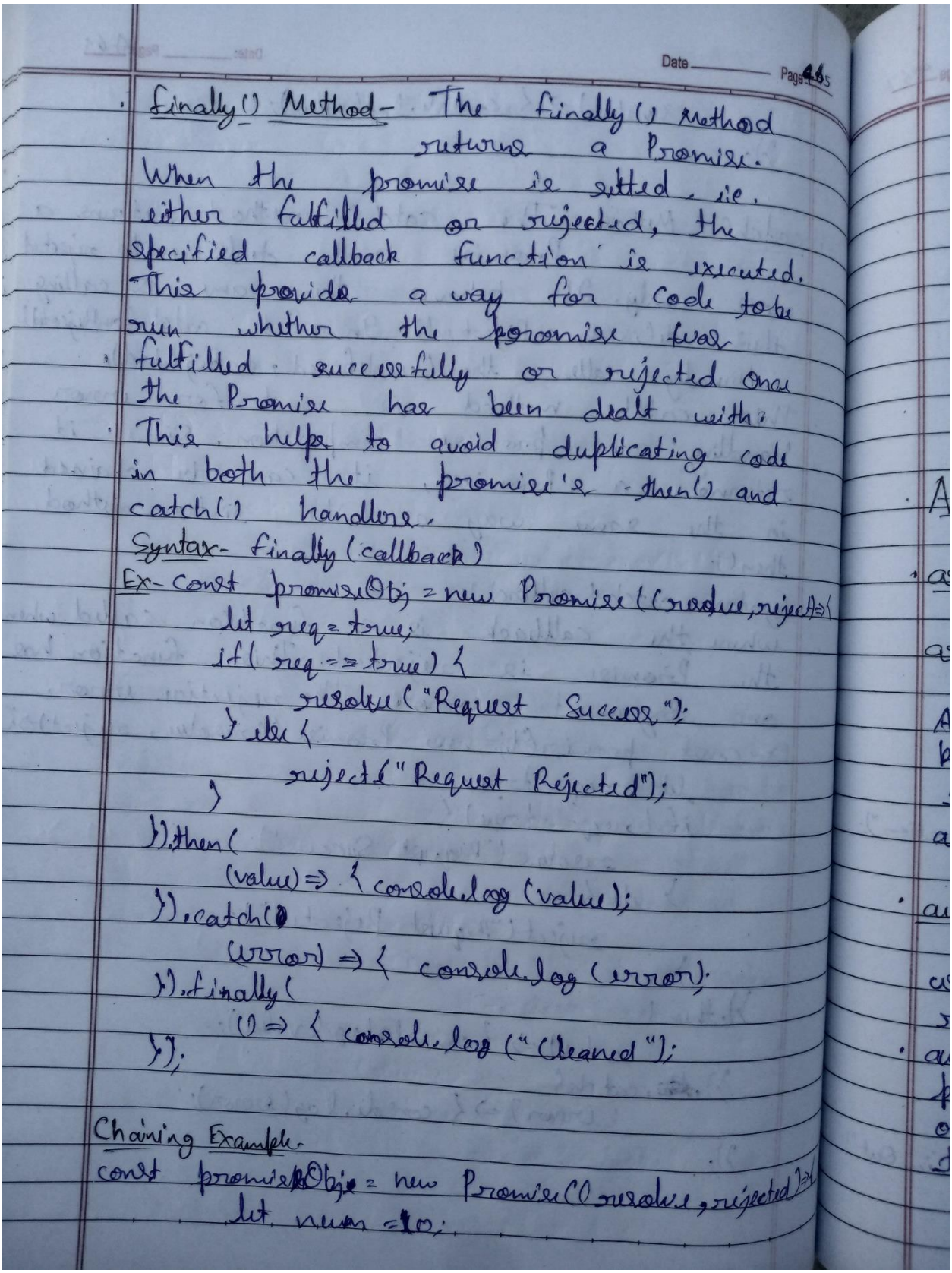


(value) => { console.log(value);
});

catch() Method - The catch() method returns a Promise and deals with rejected cases only. It behaves the same as calling then(undefined, onRejected). In fact catch(onRejected) internally calls then(undefined, onRejected).
This catch method is used for error handling in promise composition. Since it returns a Promise, it can be chained in the same way as its sister method then().

Syntax - catch(callback)
where the callback is a function called when the Promise is rejected. This function has one argument error - The rejection error.

Ex - const promiseObj = new Promise((resolve, reject)=>{
let req = true;
if(req==true){
resolve("Request Success");
} else {
reject("Request Rejected");
}
}).then(
(value) => {console.log(value);
}).catch(
(error) => { console.log(error);
});

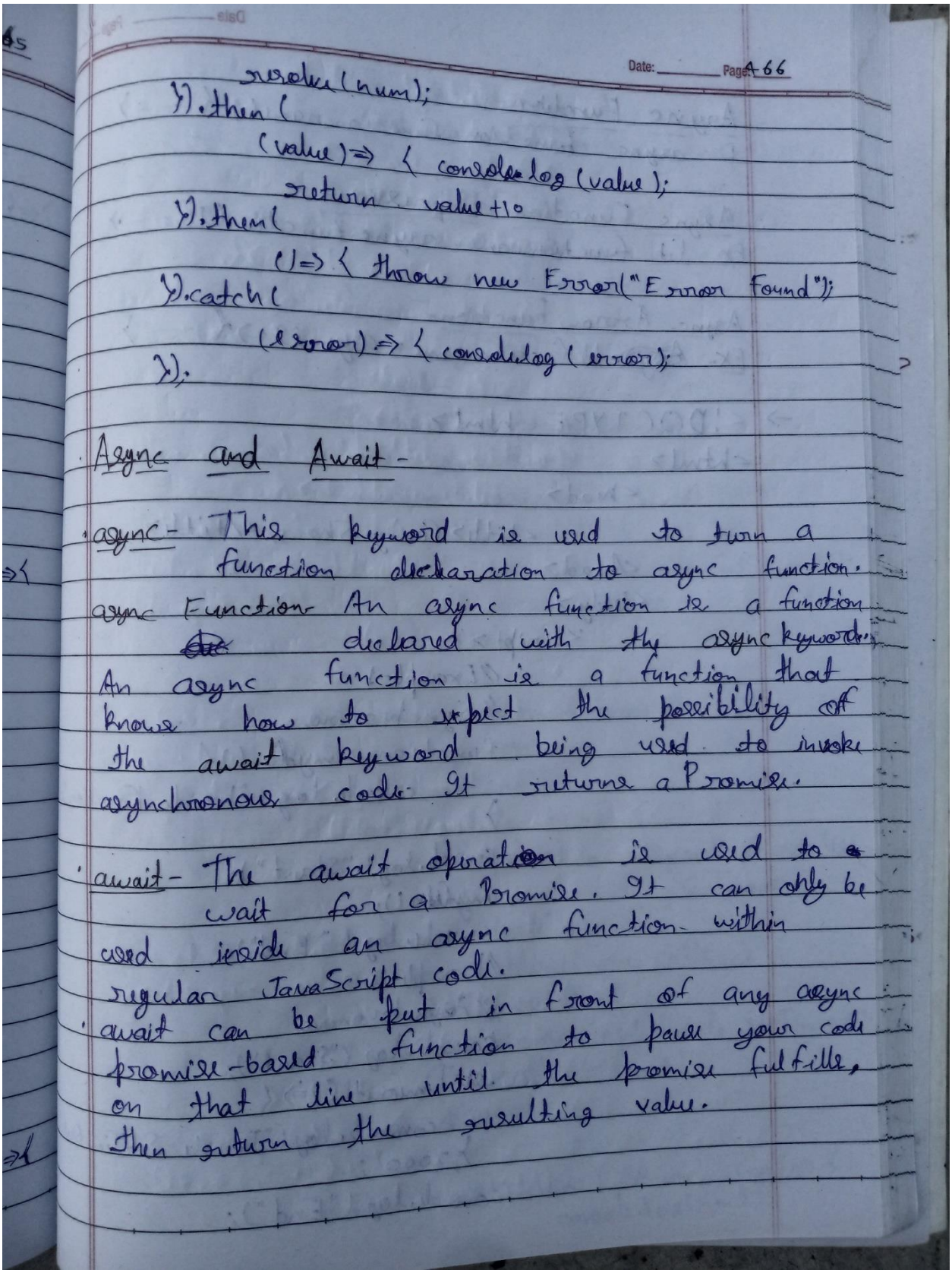


Finally() Method- The finally() method returns a Promise. When the promise is settled, ie. either fulfilled or rejected, the specified callback function is executed. This provides a way for code to be run whether the promise was fulfilled successfully or rejected once the Promise has been dealt with. This helps to avoid duplicating code in both the promise's then() and catch() handlers.

Syntax- finally(callback)

Ex-
```
const promiseObj = new Promise((resolve, reject)=>{
    let req = true;
    if(req == true){
        resolve("Request Success");
    }else{
        reject("Request Rejected");
    }
}).then(
    (value) => { console.log(value);
}).catch(
    (error) => { console.log(error);
}).finally(
    () => { console.log("Cleaned");
});
```

Chaining Example.
```
const promiseObj = new Promise((resolve, reject)=>{
    let num = 10;
```



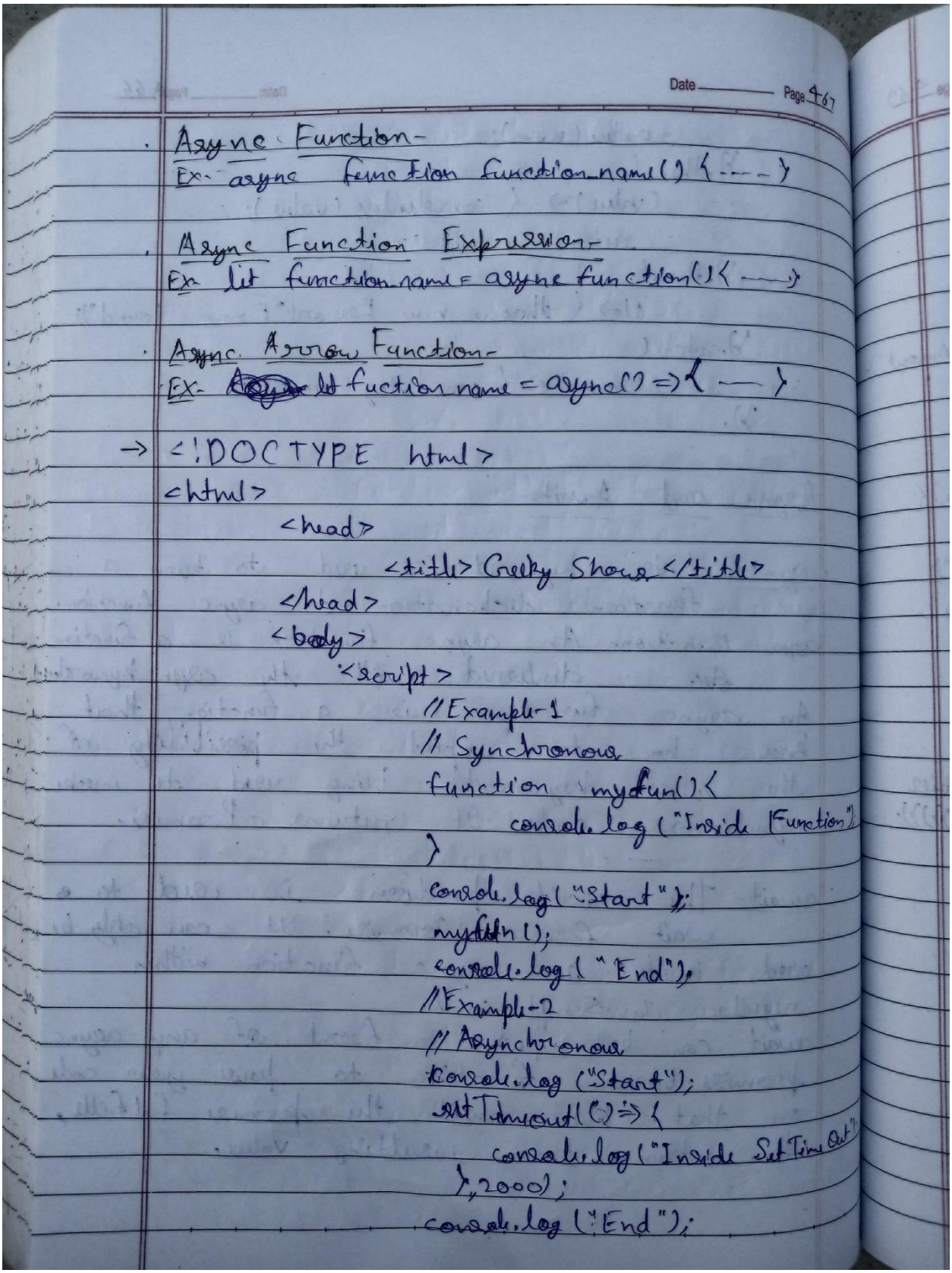
```
    resolve(num);
}).then(
    (value) => { console.log(value);
        return value+10
}).then(
    () => { throw new Error("Error Found");
}).catch(
    (error) => { console.log(error);
});
```

Async and Await -

- async - This keyword is used to turn a function declaration to async function.

async Function - An async function is a function declared with the async keyword. An async function is a function that knows how to expect the possibility of the await keyword being used to invoke asynchronous code. It returns a Promise.

- await - The await operation is used to wait for a Promise. It can only be used inside an async function within regular JavaScript code.
- await can be put in front of any async promise-based function to pause your code on that line until the promise fulfills, then return the resulting value.



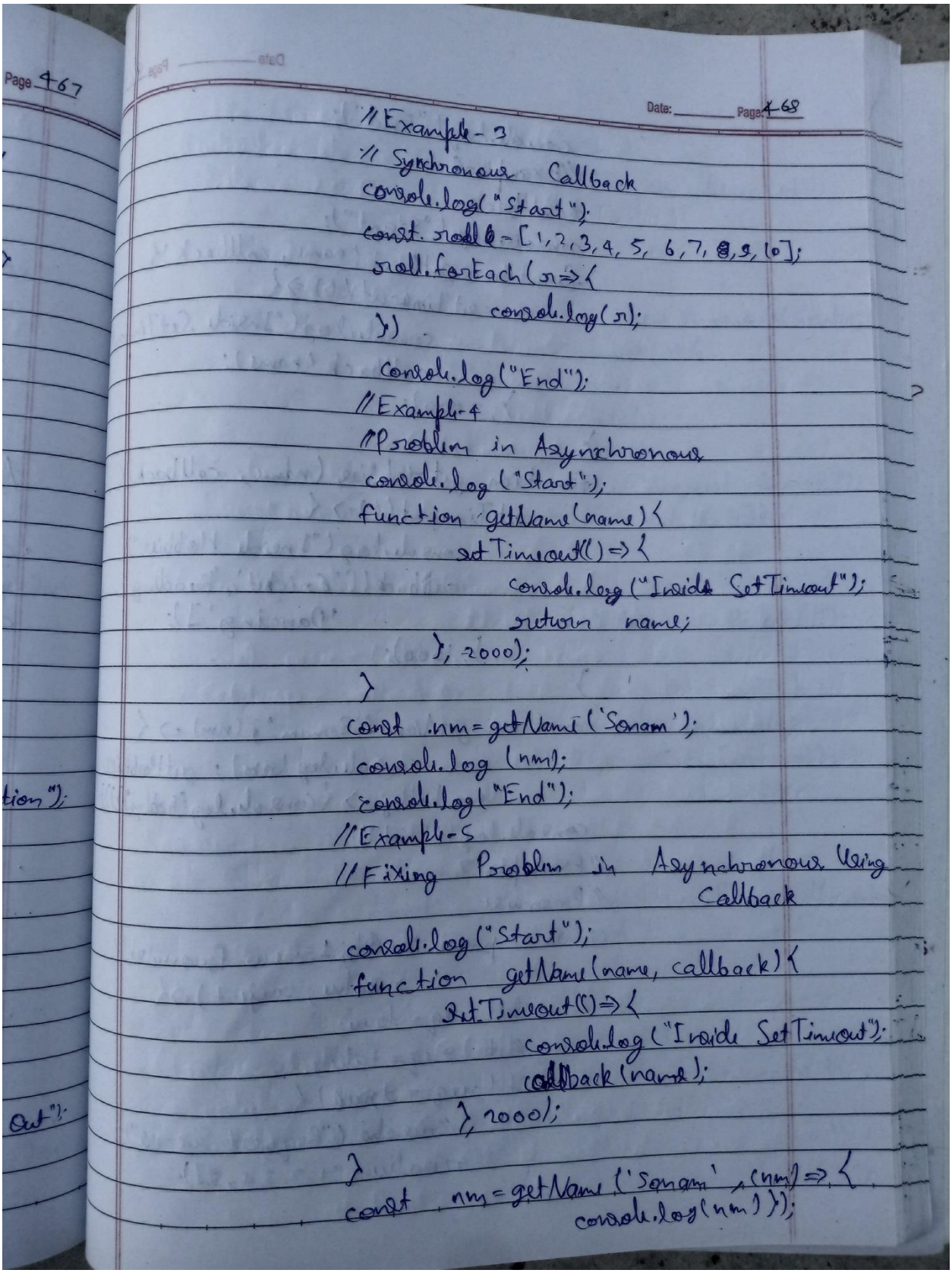
Date Page 467

. Async Function-
Ex- async function function_name() {---}

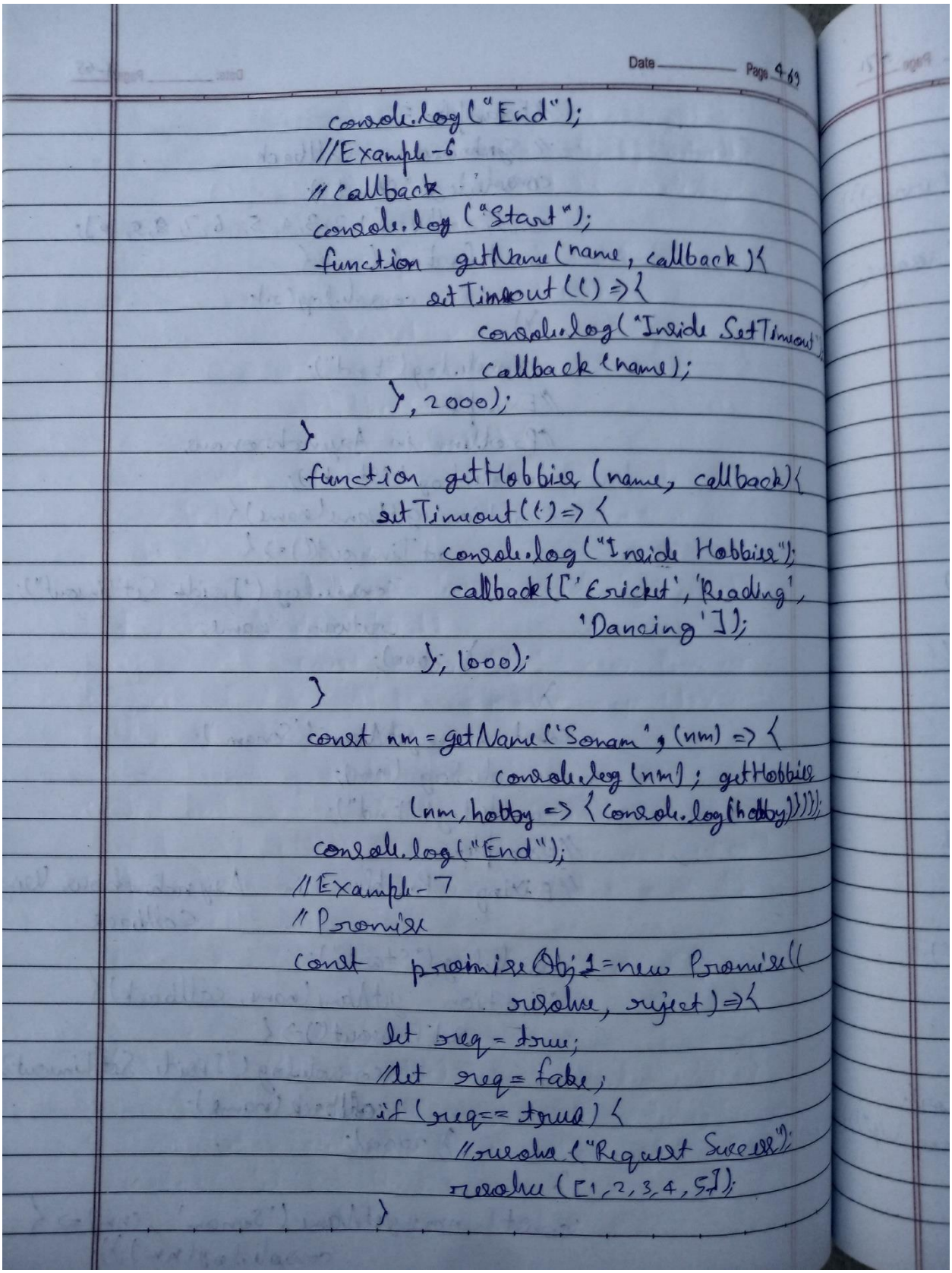
. Async Function Expression-
Ex- let function_name = async function(){---}

. Async Arrow Function-
Ex- let function_name = async()=>{---}

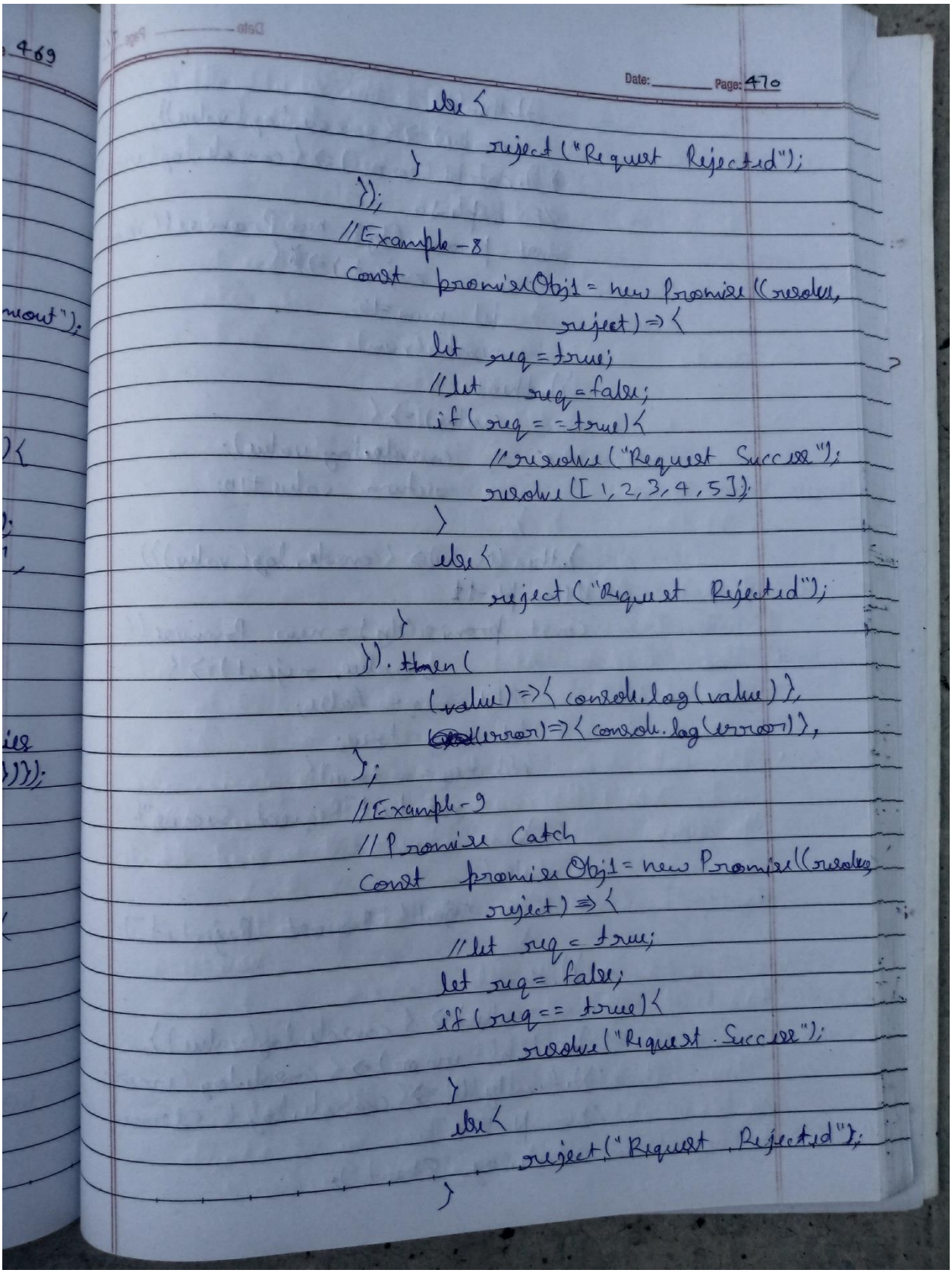
```
<!DOCTYPE html>
<html>
    <head>
        <title> Geeky Shows </title>
    </head>
    <body>
        <script>
            //Example-1
            // Synchronous
            function myfun(){
                console.log("Inside Function")
            }
            console.log("Start");
            myfun();
            console.log("End");
            //Example-2
            // Asynchronous
            console.log("Start");
            setTimeout(()=>{
                console.log("Inside SetTimeOut")
            },2000);
            console.log("End");
```



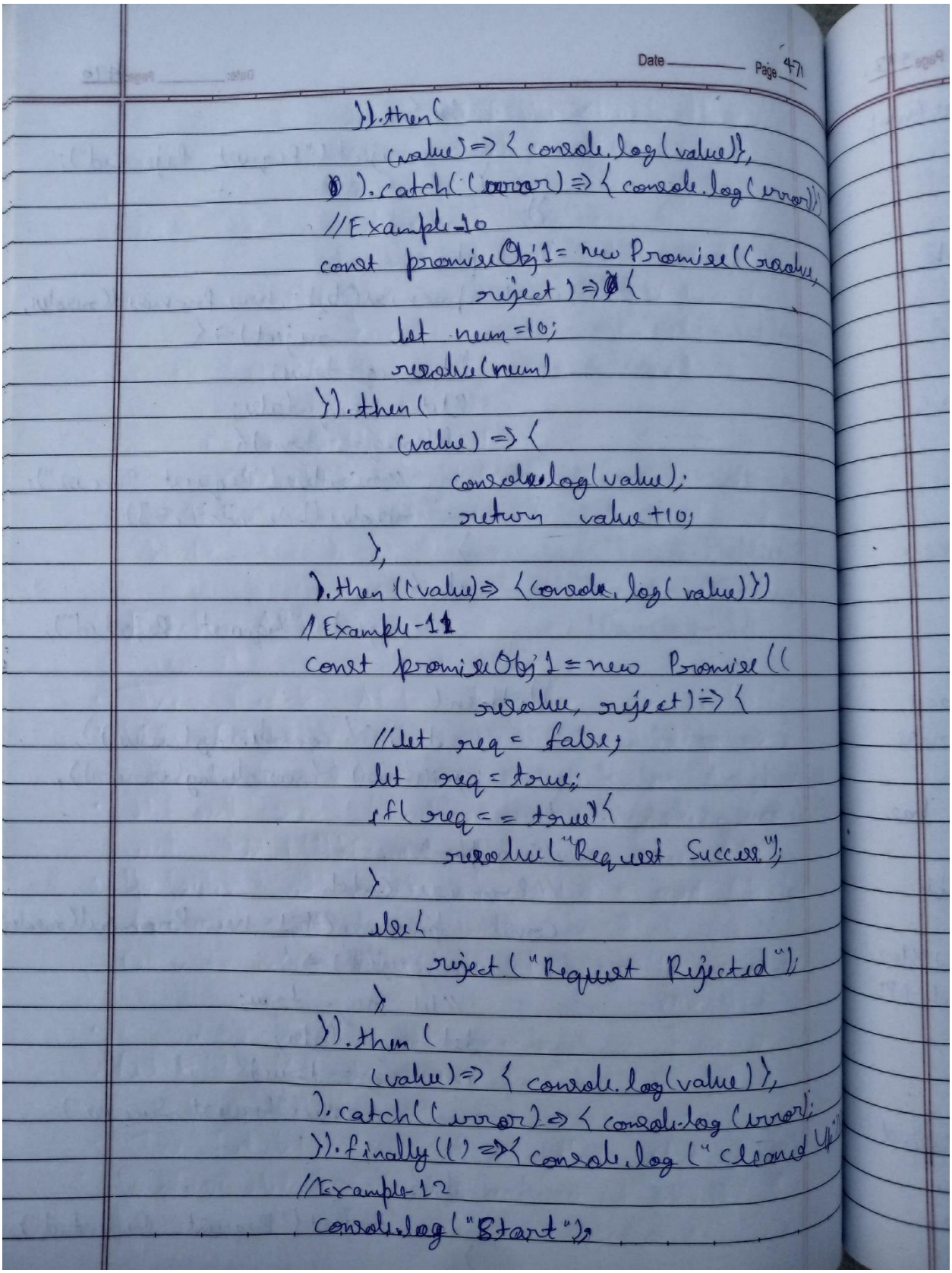
```
//Example-3
// Synchronous Callback
console.log("Start");
const nos = [1,2,3,4,5,6,7,8,9,10];
nos.forEach(n=>{
        console.log(n);
})
console.log("End");
//Example-4
//Problem in Asynchronous
console.log("Start");
function getName(name){
    setTimeout(()=>{
        console.log("Inside SetTimeout");
        return name;
    },2000);
}
const nm=getName('Sonam');
console.log(nm);
console.log("End");
//Example-5
//Fixing Problem in Asynchronous Using
                    Callback

console.log("Start");
function getName(name, callback){
    setTimeout(()=>{
        console.log("Inside SetTimeout");
        callback(name);
    },2000);
}
const nm=getName('Sonam',(nm)=>{
        console.log(nm)});
```



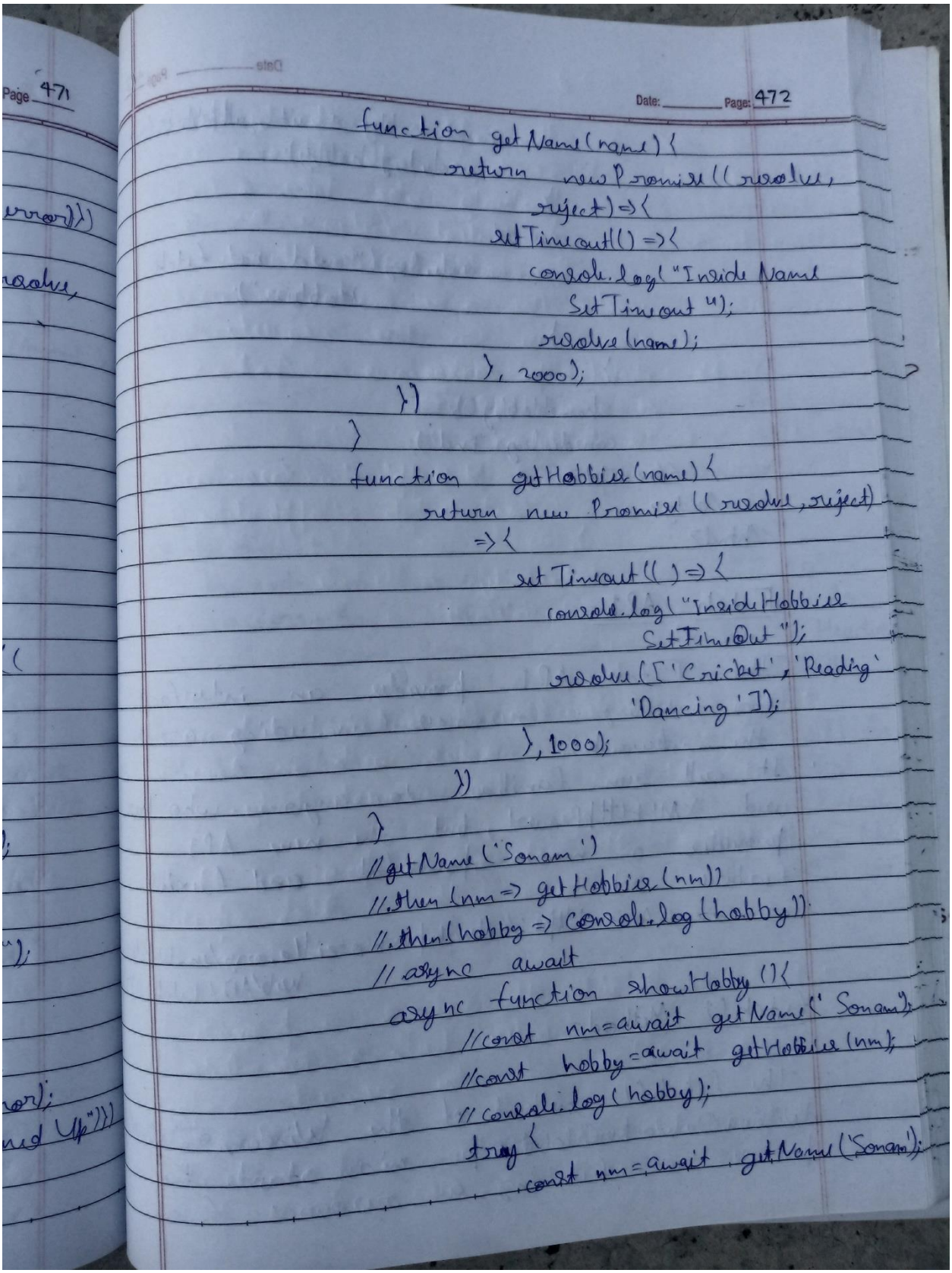
```
console.log("End");
//Example-6
// callback
console.log("Start");
function getName(name, callback){
    setTimeout(() => {
        console.log("Inside SetTimeout");
        callback(name);
    }, 2000);
}
function getHobbies(name, callback){
    setTimeout(() => {
        console.log("Inside Hobbies");
        callback(['Cricket', 'Reading',
            'Dancing']);
    }, 1000);
}
const nm = getName('Sonam', (nm) => {
        console.log(nm); getHobbies
    (nm, hobby => {console.log(hobby)}));
console.log("End");
//Example-7
// Promise
const promiseObj1 = new Promise((
        resolve, reject) => {
    let req = true;
    //let req = false;
    if(req == true){
        //resolve("Request Success");
        resolve([1,2,3,4,5]);
    }
```



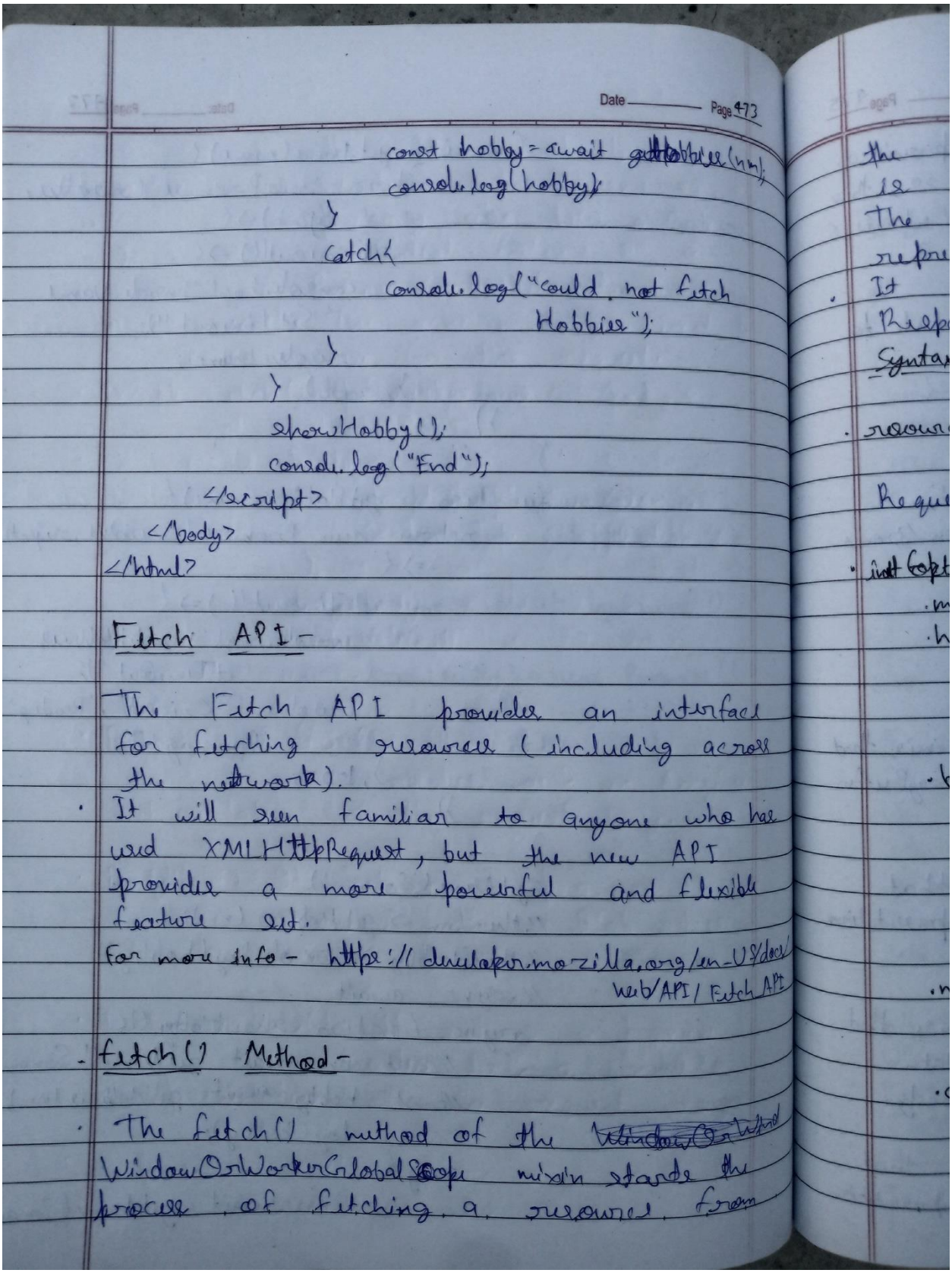
```
    else {
        reject("Request Rejected");
    }
});
//Example-8
const promiseObj1 = new Promise((resolve,
                    reject) => {
    let req = true;
    //let req = false;
    if(req == true){
        //resolve("Request Success");
        resolve([1,2,3,4,5]);
    }
    else{
        reject("Request Rejected");
    }
}).then(
    (value) => { console.log(value) },
    (error) => { console.log(error) },
);
//Example-9
//Promise Catch
const promiseObj1 = new Promise((resolve,
                    reject) => {
    //let req = true;
    let req = false;
    if(req == true){
        resolve("Request Success");
    }
    else{
        reject("Request Rejected");
    }
```



```
}).then(
    (value) => { console.log(value)},
).catch((error) => { console.log(error)})
//Example-10
const promiseObj1 = new Promise((resolve,
        reject) => {
    let num = 10;
    resolve(num)
}).then(
    (value) => {
        console.log(value);
        return value+10;
    },
).then((value) => {console.log(value)})
//Example-11
const promiseObj1 = new Promise((
        resolve, reject) => {
    //let req = false;
    let req = true;
    if(req == true){
        resolve("Request Success.");
    }
    else{
        reject("Request Rejected");
    }
}).then(
    (value) => { console.log(value)},
).catch((error) => { console.log(error);
}).finally(() => {console.log("cleared up")
//Example-12
console.log("Start");
```



```
function getName(name){
    return new Promise((resolve,
        reject)=>{
        setTimeout(()=>{
            console.log("Inside Name
                SetTimeout");
            resolve(name);
        }, 2000);
    })
}
function getHobbies(name){
    return new Promise((resolve,reject)
        =>{
        setTimeout(()=>{
            console.log("Inside Hobbies
                SetTimeOut");
            resolve(['Cricket','Reading'
                'Dancing']);
        },1000);
    })
}
//getName('Sonam')
//.then(nm=>getHobbies(nm))
//.then(hobby=>console.log(hobby))
//async await
async function showHobby(){
    //const nm=await getName("Sonam");
    //const hobby=await getHobbies(nm);
    //console.log(hobby);
    try{
        const nm=await getName('Sonam');
```



```
            const hobby = await getHobbies(nm);
            console.log(hobby);
        }
        catch{
            console.log("could not fetch
                        Hobbies");
        }
    }
    showHobby();
    console.log("End");
</script>
</body>
</html>
```

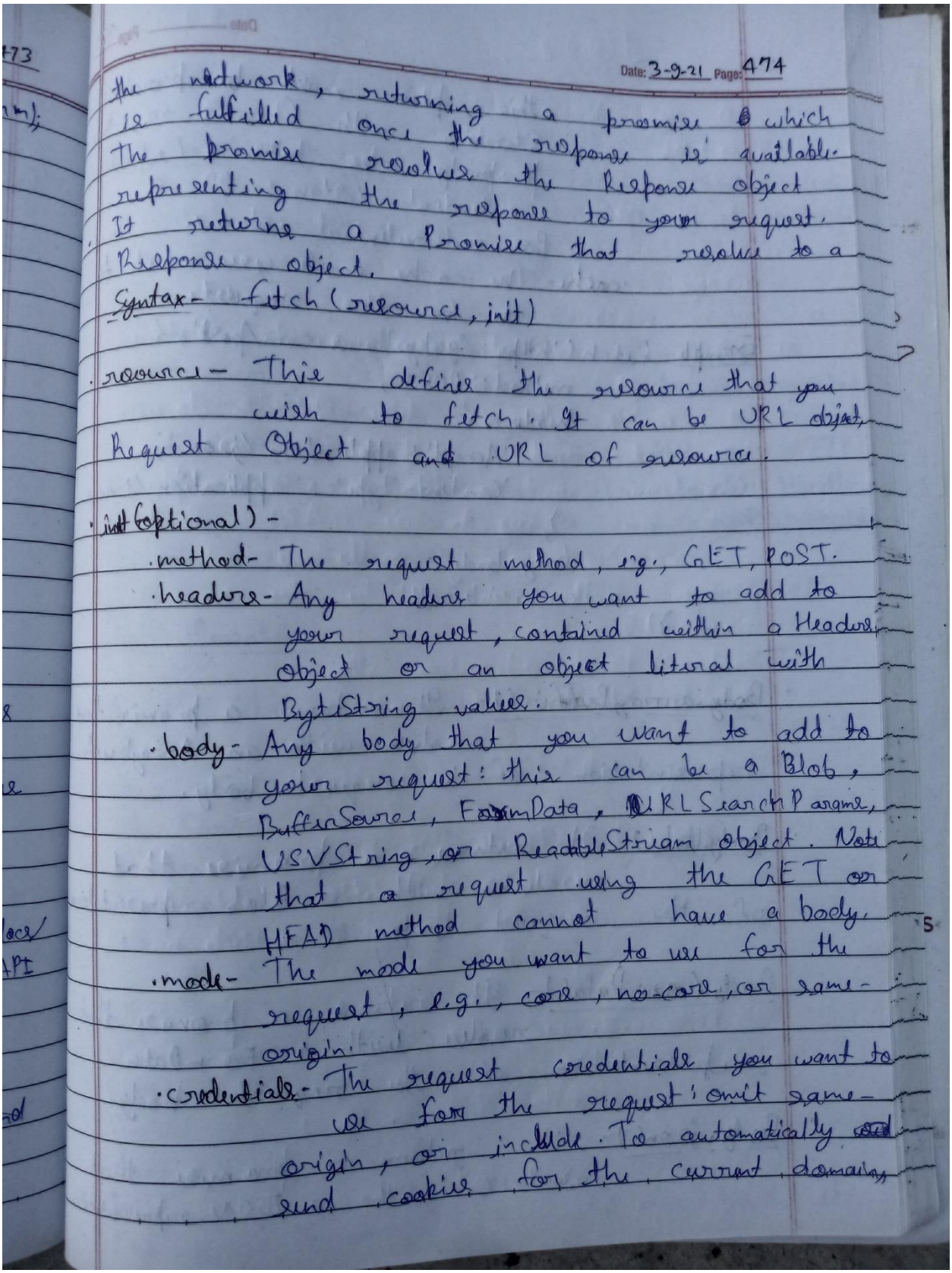
Fetch API -

- The Fetch API provides an interface for fetching resources (including across the network).
- It will seem familiar to anyone who has used XMLHttpRequest, but the new API provides a more powerful and flexible feature set.

For more info - https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API

fetch() Method -

- The fetch() method of the ~~WindowOrWorker~~ WindowOrWorkerGlobalScope mixin starts the process of fetching a resource from

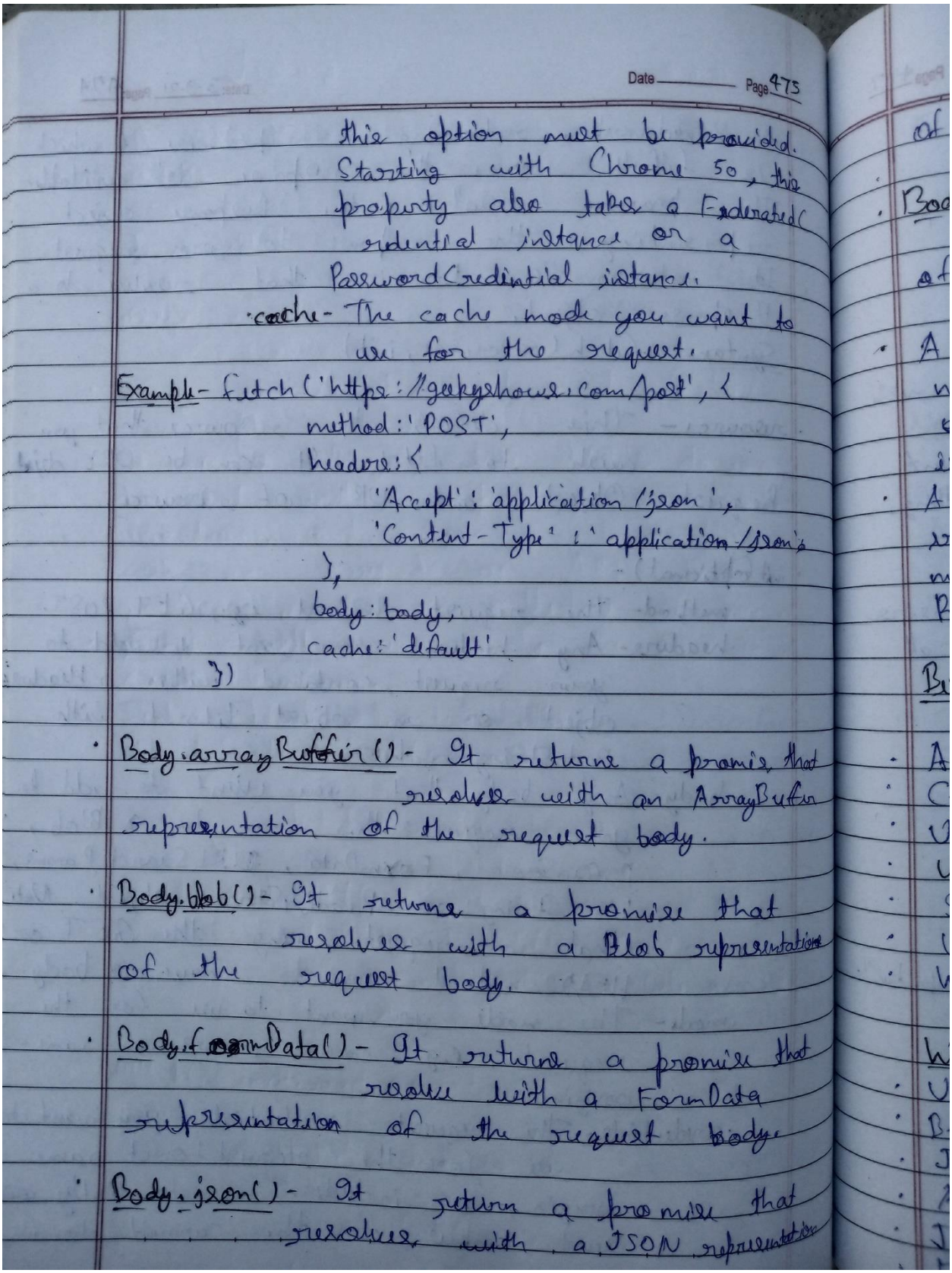


Date: 3-9-21 Page: 474

the network, returning a promise which is fulfilled once the response is available. The promise resolves the Response object representing the response to your request. It returns a Promise that resolves to a Response object.

Syntax- fetch(resource, init)

- resource - This defines the resource that you wish to fetch. It can be URL object, Request Object and URL of resource.
- init (optional) -
  - method - The request method, e.g., GET, POST.
  - headers - Any headers you want to add to your request, contained within a Headers object or an object literal with ByteString values.
  - body - Any body that you want to add to your request: this can be a Blob, BufferSource, FormData, URLSearchParams, USVString, or ReadableStream object. Note that a request using the GET or HEAD method cannot have a body.
  - mode - The mode you want to use for the request, e.g., cors, no-cors, or same-origin.
  - credentials - The request credentials you want to use for the request: omit, same-origin, or include. To automatically send cookies for the current domain,



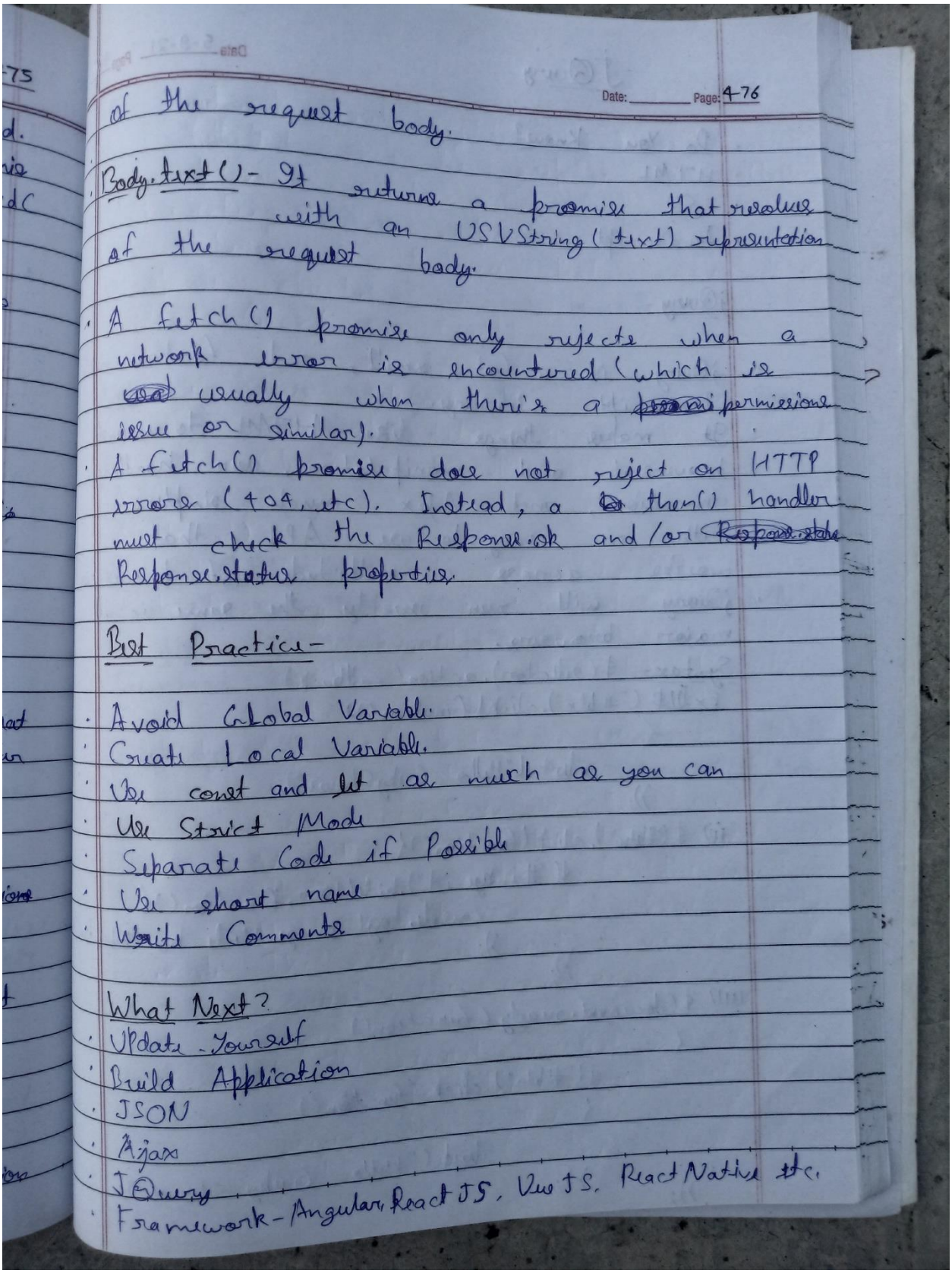
Date _____ Page 475

this option must be provided. Starting with Chrome 50, this property also takes a FederatedCredential instance or a PasswordCredential instance.

- cache - The cache mode you want to use for the request.

Example -
```
fetch('https://geekyshows.com/post', {
    method: 'POST',
    headers: {
        'Accept': 'application/json',
        'Content-Type': 'application/json'
    },
    body: body,
    cache: 'default'
})
```

- Body.arrayBuffer() - It returns a promise that resolves with an ArrayBuffer representation of the request body.
- Body.blob() - It returns a promise that resolves with a Blob representation of the request body.
- Body.formData() - It returns a promise that resolves with a FormData representation of the request body.
- Body.json() - It return a promise that resolves with a JSON representation



of the request body.

Body.text() - It returns a promise that resolves with an USVString (text) representation of the request body.

- A fetch() promise only rejects when a network error is encountered (which is usually when there's a permissions issue or similar).
- A fetch() promise does not reject on HTTP errors (404, etc). Instead, a then() handler must check the Response.ok and/or Response.status properties.

Best Practice -

- Avoid Global Variable.
- Create Local Variable.
- Use const and let as much as you can
- Use Strict Mode
- Separate Code if Possible
- Use short name
- Write Comments

What Next?

- Update Yourself
- Build Application
- JSON
- Ajax
- JQuery
- Framework - Angular, React JS, Vue JS, React Native etc.