Core JavaScript

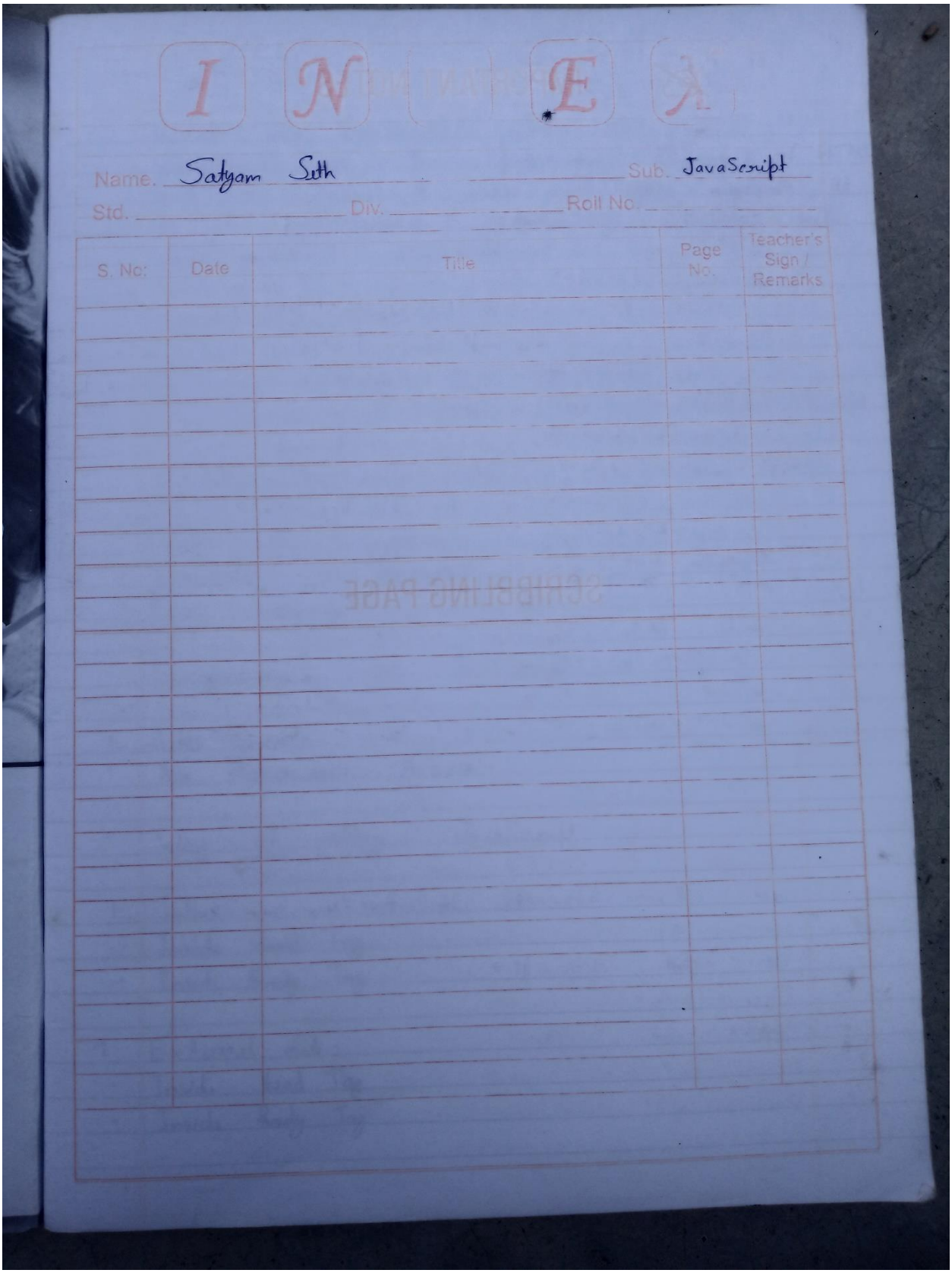
GEEKYSHOWS YOUTUBE CHANNEL LEARNING NOTES

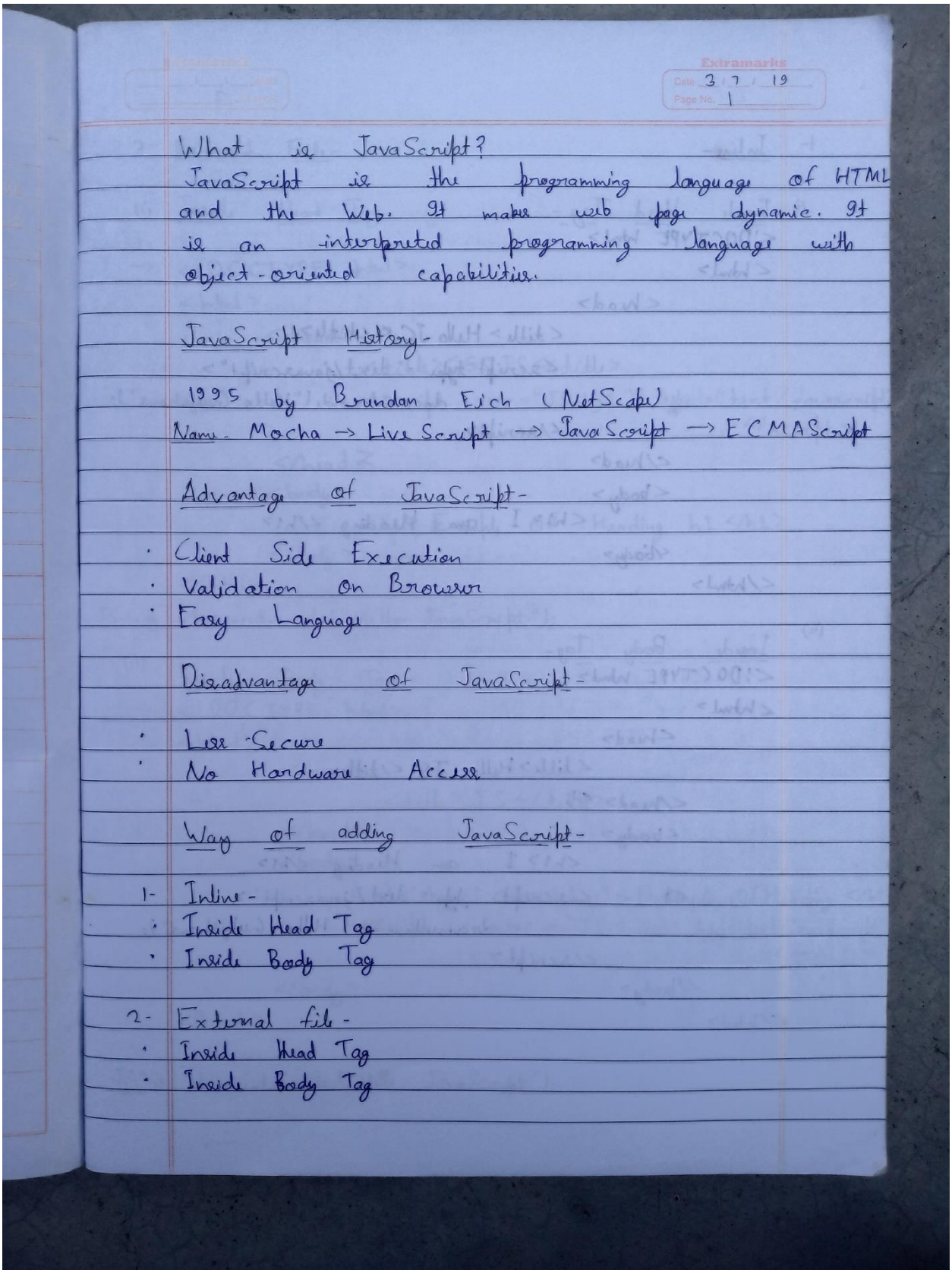
Source Code- https://github.com/satyam-seth-learnings/javascript_learning/tree/master/Geekyshows/Core%20Java%20Script

Playlist Link- https://youtube.com/playlist?list=PLbGui_ZYuhiiaQjuOfvgx_-gzVBlCxrk0

SATYAM SETH

26-09-2021


I N E
Name. Satyam Seth
Sub. JavaScript
Std.
Div.
Roll No.
S. No:
Date
Title
Page No.
Teacher's Sign / Remarks



Extramarks
Date 3/7/19
Page No. 1

What is JavaScript?
JavaScript is the programming language of HTML and the Web. It make web page dynamic. It is an interpreted programming language with object-oriented capabilities.

JavaScript History-

1995 by Brundan Eich (NetScape)
Name - Mocha → Live Script → JavaScript → ECMAScript

Advantage of JavaScript-

- Client Side Execution
- Validation on Browser
- Easy Language

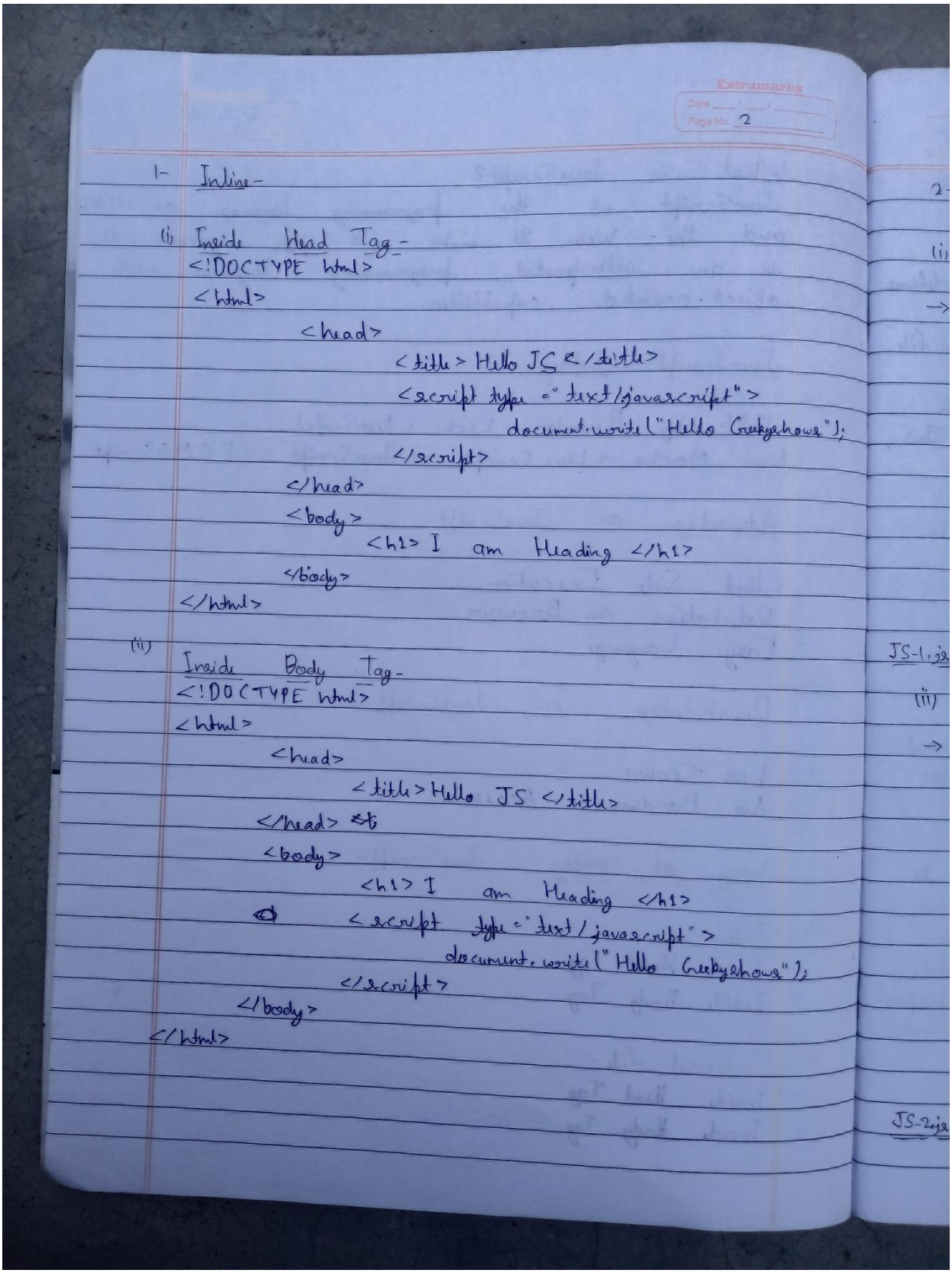
Disadvantage of JavaScript-

- Less Secure
- No Hardware Access

Way of adding JavaScript-

1- Inline -
- Inside Head Tag
- Inside Body Tag

2- External file -
- Inside Head Tag
- Inside Body Tag



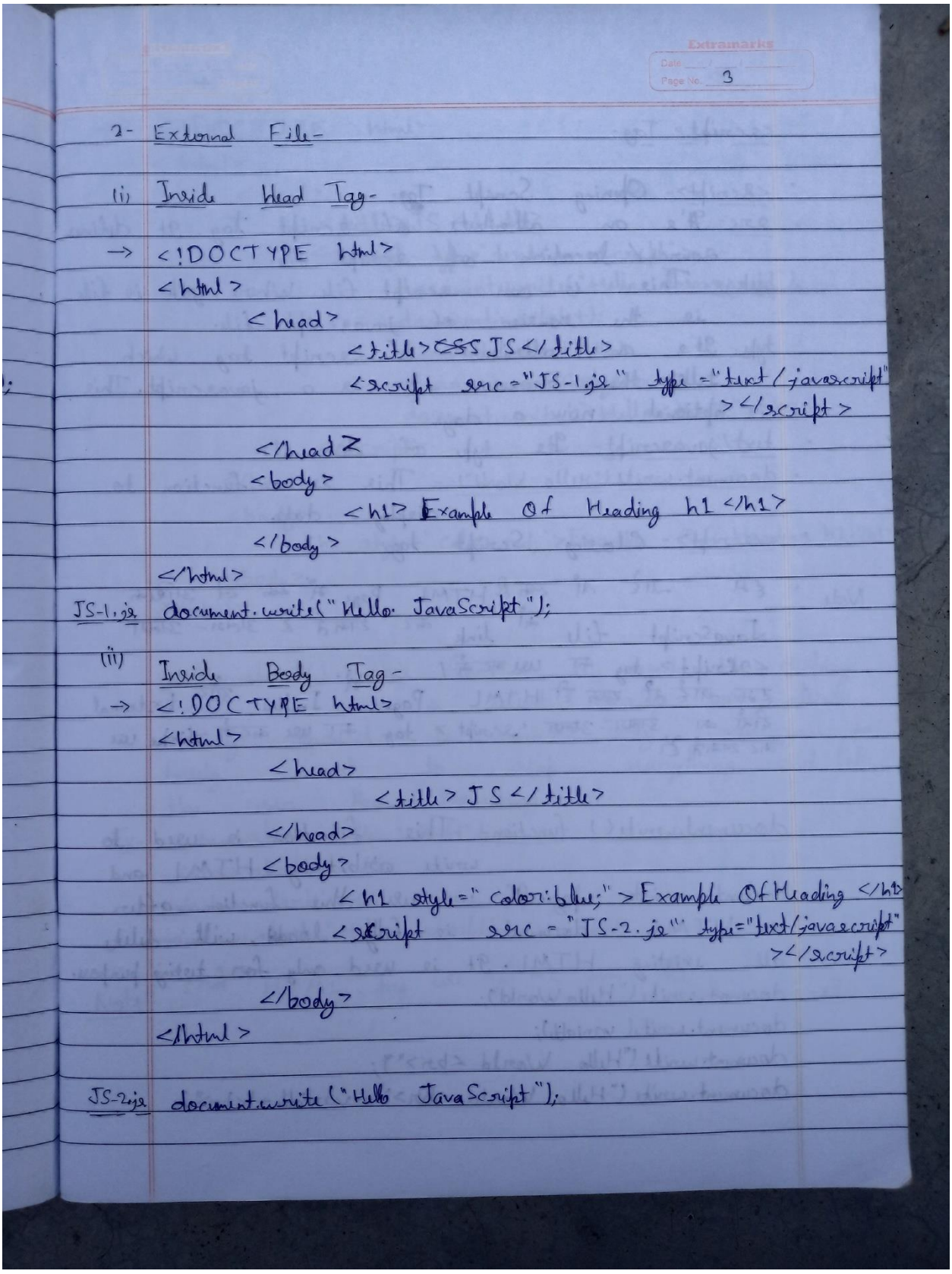
1- Inline -

(i) Inside Head Tag -

```
<!DOCTYPE html>
<html>
    <head>
        <title> Hello JS </title>
        <script type = "text/javascript">
            document.write("Hello Geekyshows");
        </script>
    </head>
    <body>
        <h1> I am Heading </h1>
    </body>
</html>
```

(ii) Inside Body Tag -

```
<!DOCTYPE html>
<html>
    <head>
        <title> Hello JS </title>
    </head>
    <body>
        <h1> I am Heading </h1>
        <script type = "text/javascript">
            document.write("Hello Geekyshows");
        </script>
    </body>
</html>
```



2- External File-

(i) Inside Head Tag-

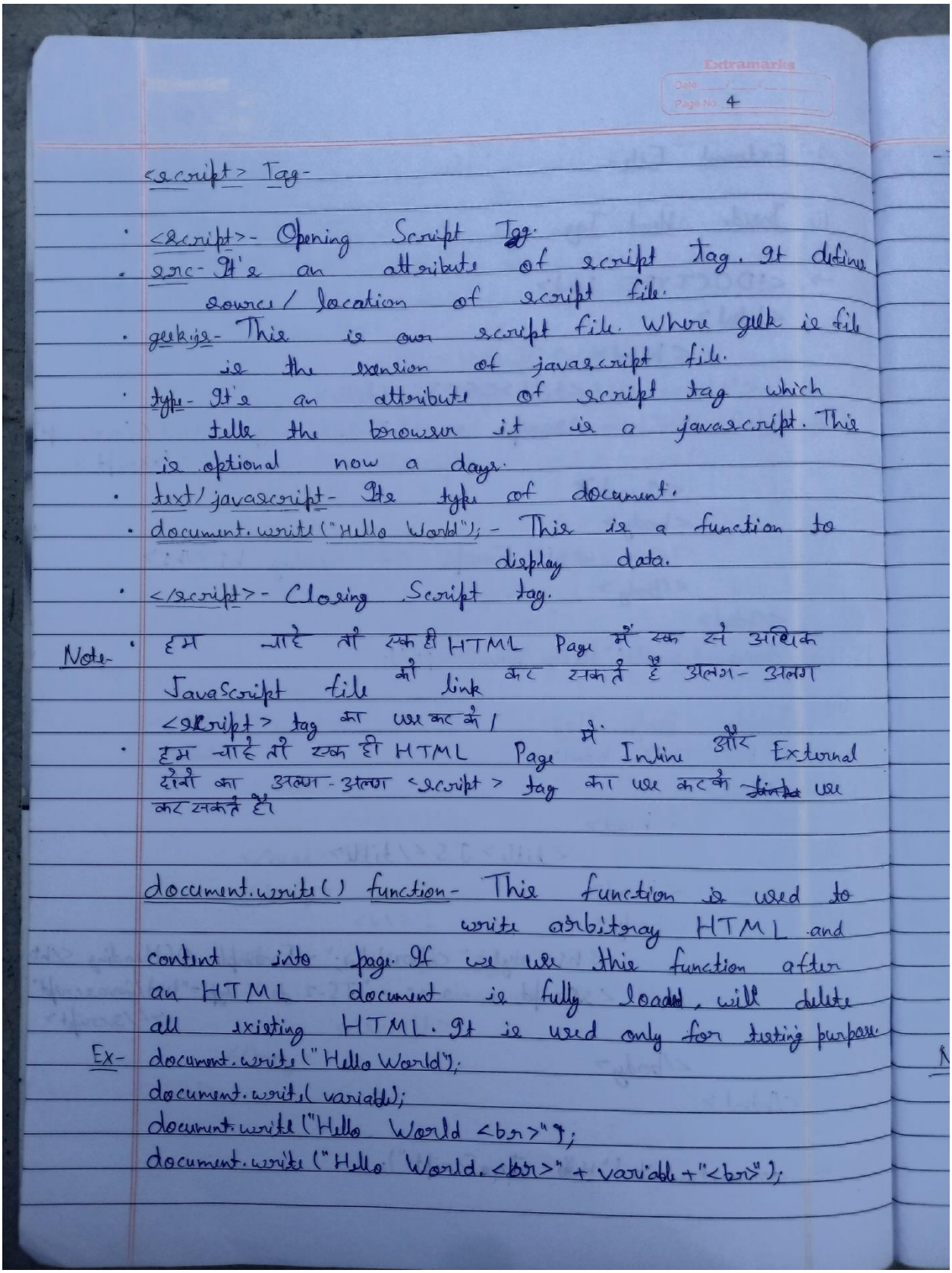
→
```
<!DOCTYPE html>
<html>
    <head>
        <title>CSS JS</title>
        <script src="JS-1.js" type="text/javascript"
        ></script>
    </head>
    <body>
        <h1> Example Of Heading h1 </h1>
    </body>
</html>
```

JS-1.js
```
document.write("Hello. JavaScript");
```

(ii) Inside Body Tag-

→
```
<!DOCTYPE html>
<html>
    <head>
        <title> JS </title>
    </head>
    <body>
        <h1 style="color:blue;"> Example Of Heading </h1>
        <script src="JS-2.js" type="text/javascript"
        ></script>
    </body>
</html>
```

JS-2.js
```
document.write("Hello JavaScript");
```



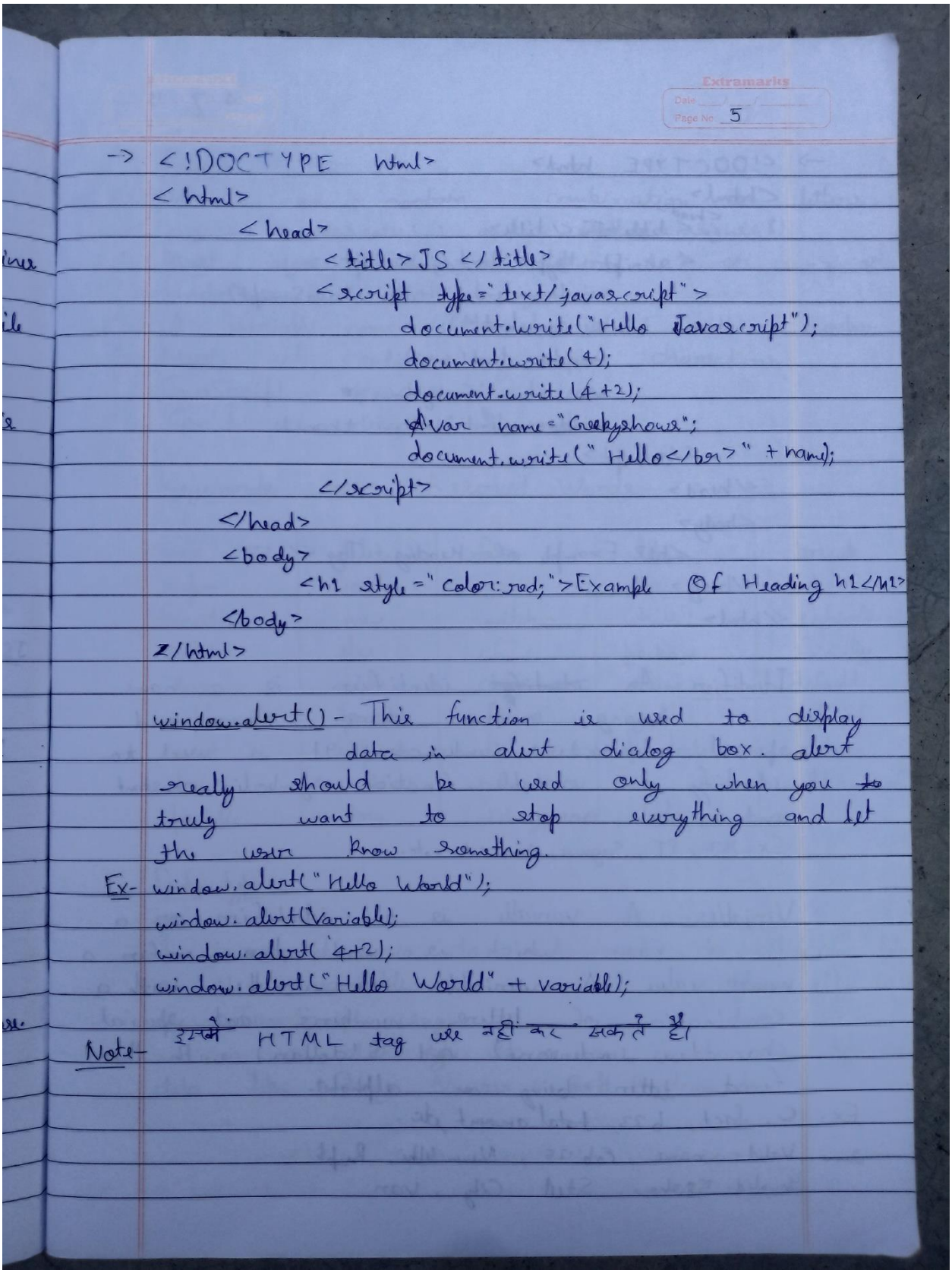
<script> Tag-

- <script>- Opening Script Tag.
- src- It's an attribute of script tag. It define source / location of script file.
- geekyjs- This is our script file. Where geek is file is the extension of javascript file.
- type- It's an attribute of script tag which tells the browser it is a javascript. This is optional now a days.
- text/javascript- Its type of document.
- document.write("Hello World");- This is a function to display data.
- </script>- Closing Script tag.

Note-
- हम चाहे तो एक ही HTML Page में एक से अधिक JavaScript file को link कर सकते है अलग-अलग <script> tag का use कर के।
- हम चाहे तो एक ही HTML Page में Inline और External दोनों का अलग-अलग <script> tag का use कर के ~~link~~ use कर सकते है।

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

Ex-
```
document.write("Hello World");
document.write(variable);
document.write("Hello World <br>");
document.write("Hello World. <br>" + variable + "<br>");
```

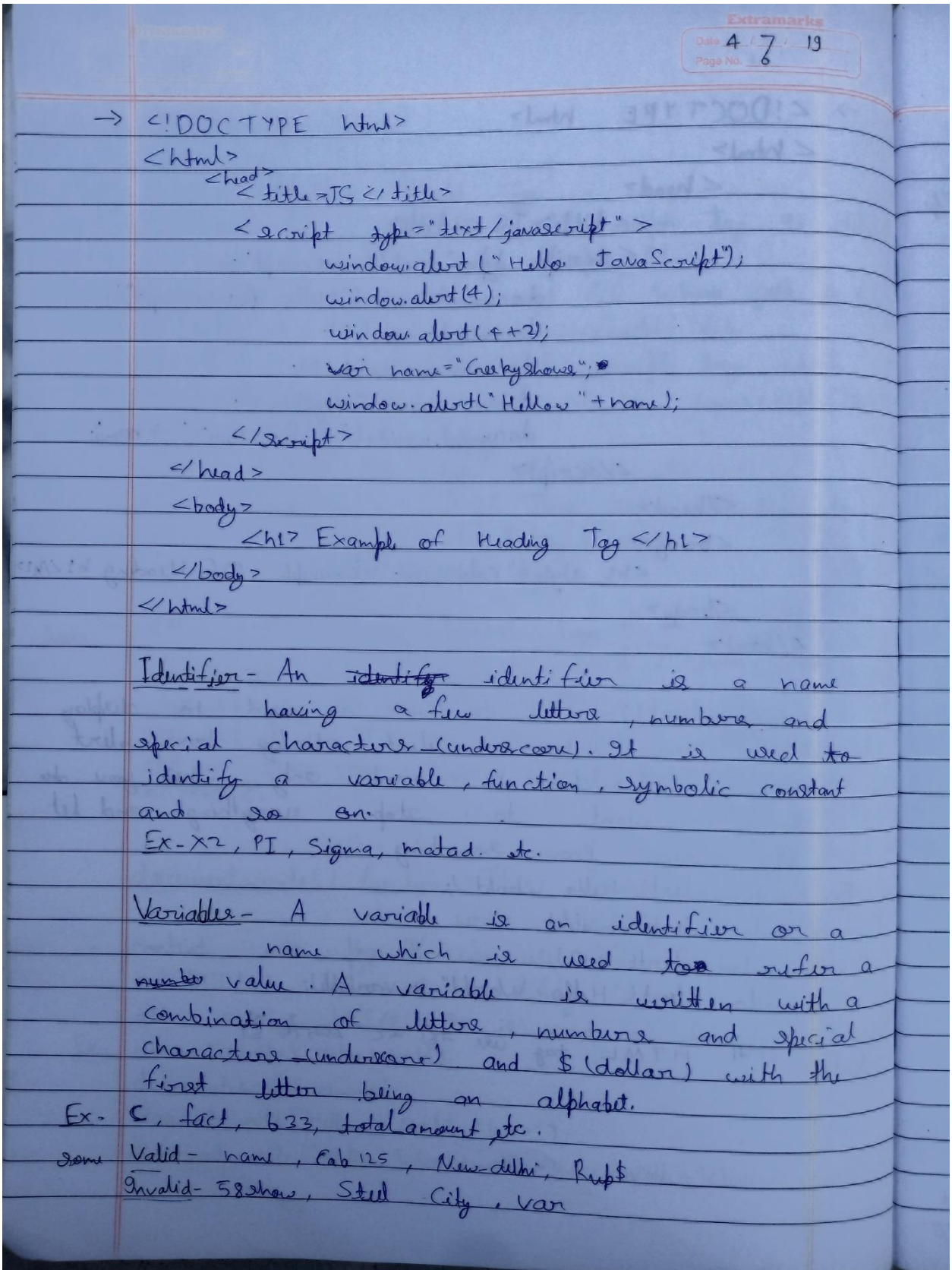


```
-> <!DOCTYPE html>
<html>
    <head>
        <title> JS </title>
        <script type="text/javascript">
            document.write("Hello Javascript");
            document.write(4);
            document.write(4+2);
            var name = "Geekyshows";
            document.write("Hello</br>" + name);
        </script>
    </head>
    <body>
        <h1 style="color:red;">Example Of Heading h1</h1>
    </body>
</html>
```

window.alert() - This function is used to display data in alert dialog box. alert really should be used only when you to truly want to stop everything and let the user know something.

Ex-
```
window.alert("Hello World");
window.alert(Variable);
window.alert(4+2);
window.alert("Hello World" + variable);
```

Note- इनको HTML tag use नहीं कर सकते हैं।



→ <!DOCTYPE html>
<html>
<head>
<title>JS</title>
<script type="text/javascript">
window.alert("Hello JavaScript");
window.alert(4);
window.alert(4+2);
var name="Geekyshows";
window.alert("Hellow "+name);
</script>
</head>
<body>
<h1> Example of Heading Tag </h1>
</body>
</html>

Identifier - An ~~identifier~~ identifier is a name having a few letters, numbers and special characters _(underscore). It is used to identify a variable, function, symbolic constant and so on.
Ex- X2, PI, Sigma, matad. etc.

Variables - A variable is an identifier or a name which is used to refer a ~~number~~ value. A variable is written with a combination of letters, numbers and special characters _(underscore) and $ (dollar) with the first letter being an alphabet.
Ex- c, fact, b33, total_amount, etc.
some Valid - name, Cab125, New_delhi, Rup$
Invalid - 58show, Steel City, var

Extramarks
Date
Page No 7

Rules-

- Variable can contain combination of letters, digits, underscore (_) and dollar sign ($).
- Must begin with a letter A-Z or a-z or underscore or dollar sign.
- A variable name cannot start with a number.
- Must not contain any space character.
- JavaScript is case-sensitive
- Can't use reserved keywords.

Keywords or Reserved Words-

var	delete	for	let	break
super	void	case	do	static
function	new	switch	while	interface
catch	else	if	package	finally
this	with	class	enum	default
implements	private	throw	yield	typedef
const	export	import	protected	return
true	continue	extends	in	instanceof
public	try	debugger	false	

Data type-

In javascript we do not need to specify type of the variable because it is dynamically used by JavaScript engine
We can var data type. It can hold any type of data like String, Number, Boolean etc.

Extramarks
Date
Page No 7

Rules-

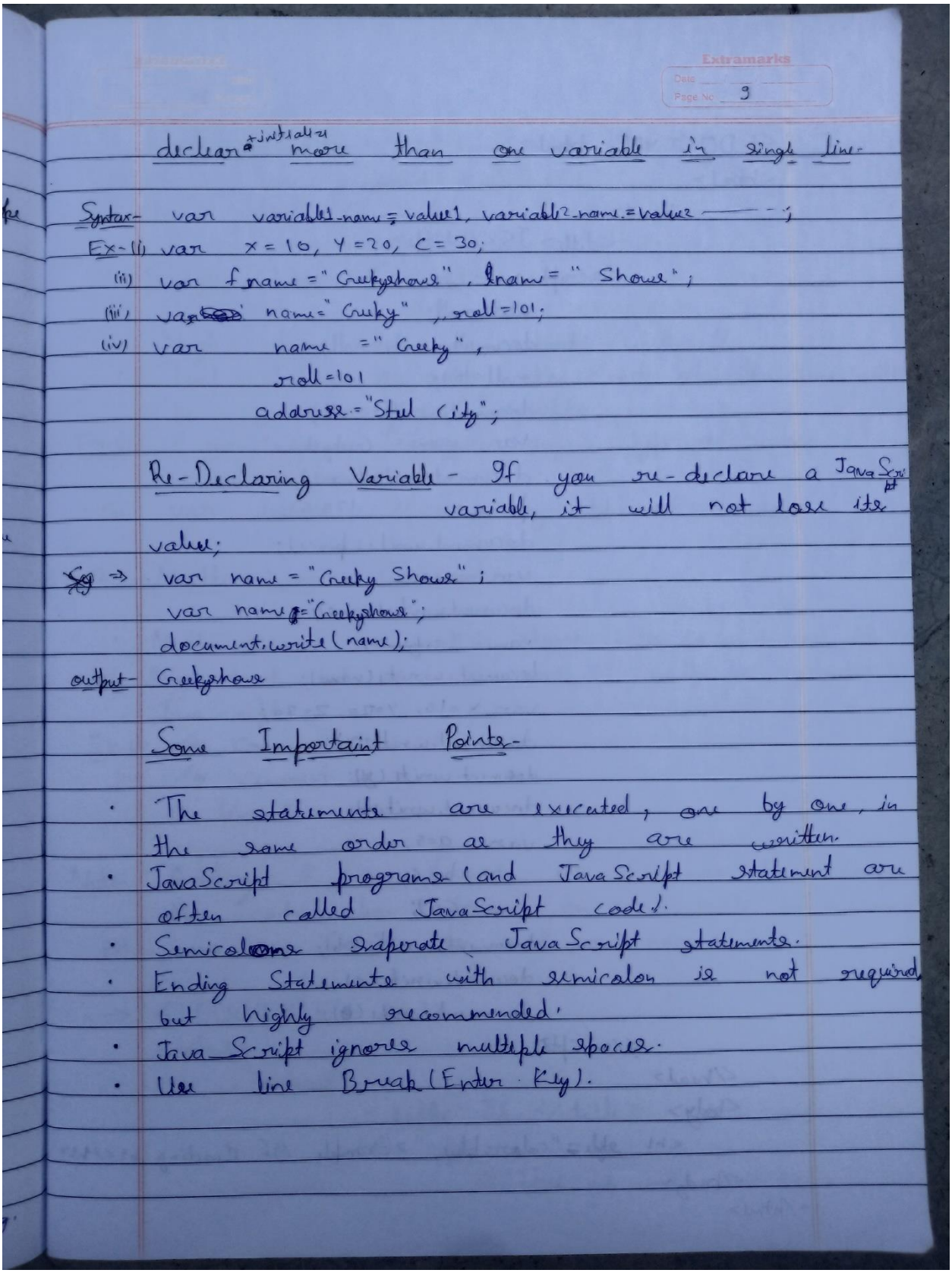
- Variable can contain combination of letters, digits, underscore (_) and dollar sign ($).
- Must begin with a letter A-Z or a-z or underscore or dollar sign.
- A variable name cannot start with a number.
- Must not contain any space character.
- JavaScript is case-sensitive
- Can't use reserved keywords.

Keywords or Reserved Words-

var	delete	for	let	break
super	void	case	do	static
function	new	switch	while	interface
catch	else	if	package	finally
this	with	class	enum	default
implements	private	throw	yield	typedef
const	export	import	protected	return
true	continue	extends	in	instanceof
public	try	debugger	false	

Data type-

In javascript we do not need to specify type of the variable because it is dynamically used by JavaScript engine
We can var data type. It can hold any type of data like String, Number, Boolean etc.



declare & initialize more than one variable in single line-

Syntax- var variable1_name = value1, variable2_name = value2 ——— ;

Ex-(i) var x = 10, Y = 20, C = 30;

(ii) var fname = "Geekyshows", lname = "Shows";

(iii) var name = "Geeky", roll = 101;

(iv) var name = "Geeky",
roll = 101
address = "Steel City";

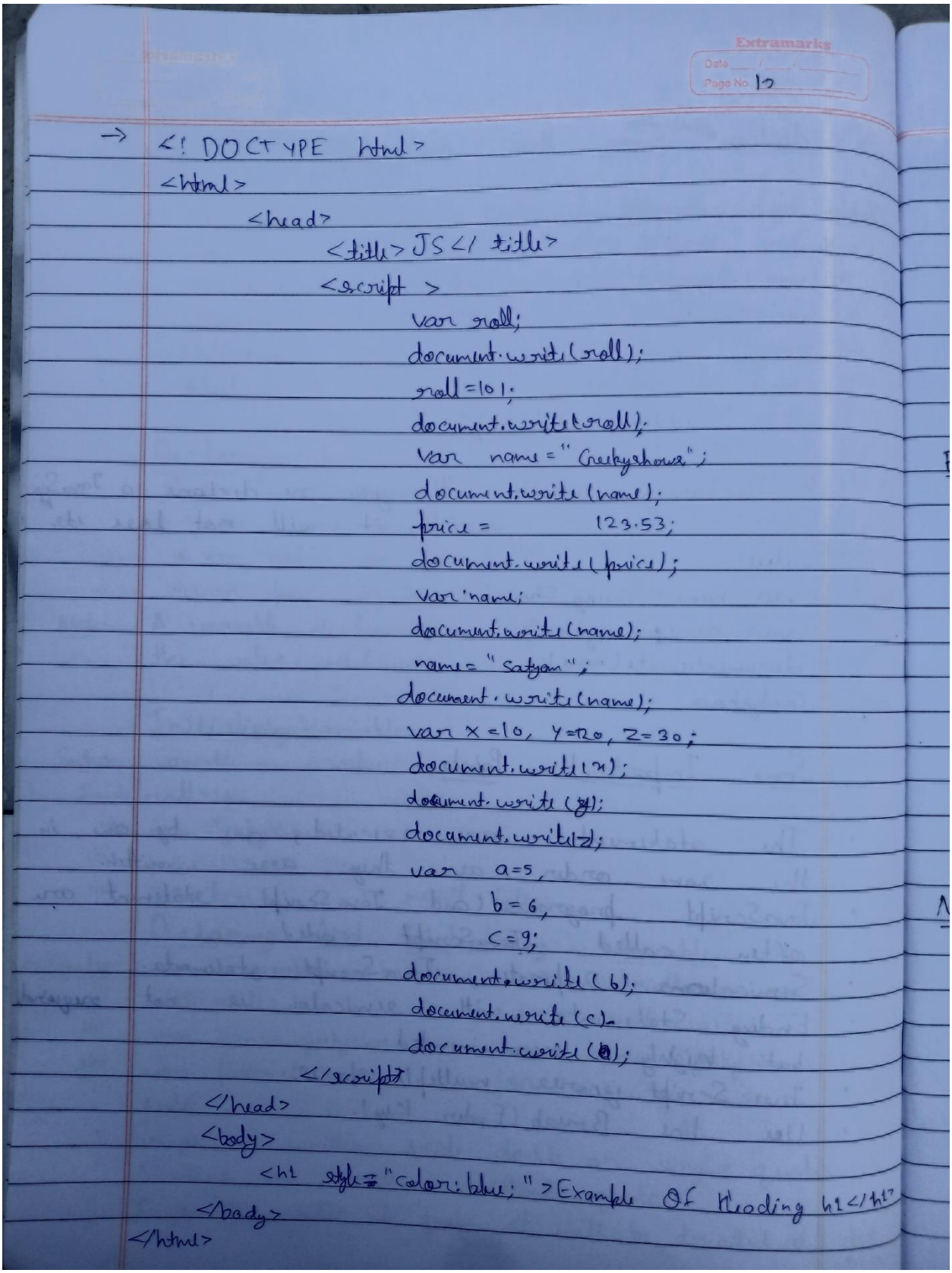
Re-Declaring Variable - If you re-declare a JavaScript variable, it will not lose its value;

Eg ⇒ var name = "Geeky Shows";
var name = "Geekyshows";
document.write(name);

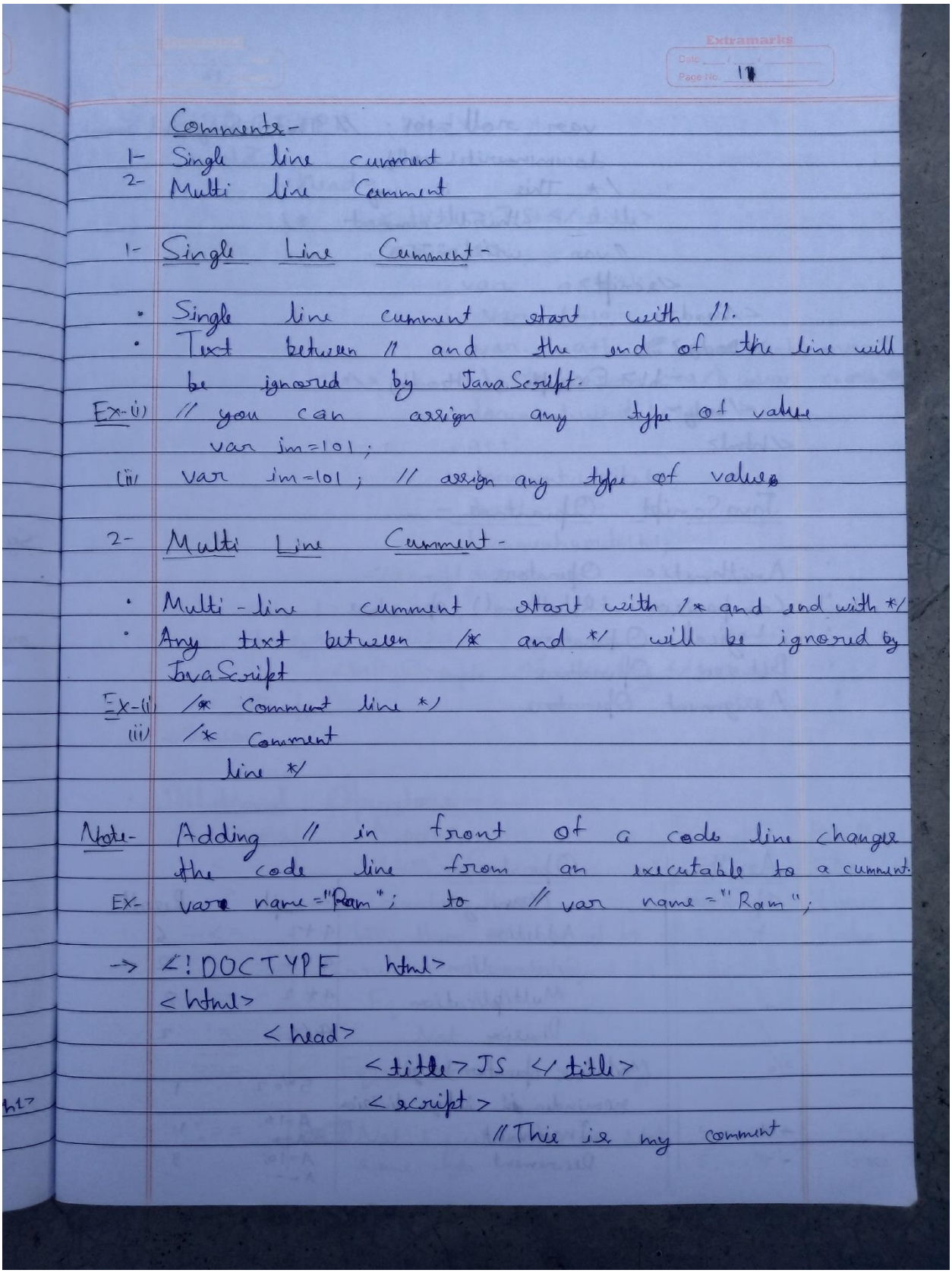
output- Geekyshows

Some Important Points-

- The statements are executed, one by one, in the same order as they are written.
- JavaScript programs (and JavaScript statement are often called JavaScript code).
- Semicolons separate JavaScript statements.
- Ending Statements with semicolon is not required, but highly recommended.
- JavaScript ignores multiple spaces.
- Use line Break (Enter Key).



```
<!DOCTYPE html>
<html>
    <head>
        <title>JS</title>
        <script>
            var roll;
            document.write(roll);
            roll=101;
            document.write(roll);
            var name="Geekyshows";
            document.write(name);
            price = 123.53;
            document.write(price);
            var name;
            document.write(name);
            name="satyam";
            document.write(name);
            var x=10, y=20, z=30;
            document.write(x);
            document.write(y);
            document.write(z);
            var a=5,
                b=6,
                c=9;
            document.write(b);
            document.write(c);
            document.write(a);
        </script>
    </head>
    <body>
        <h1 style="color:blue;">Example Of Heading h1</h1>
    </body>
</html>
```



Comments-
1- Single line comment
2- Multi line Comment

1- Single Line Comment-

- Single line comment start with //.
- Text between // and the end of the line will be ignored by JavaScript.

Ex-(i) // you can assign any type of value
var im=101;
(ii) var im=101; // assign any type of value

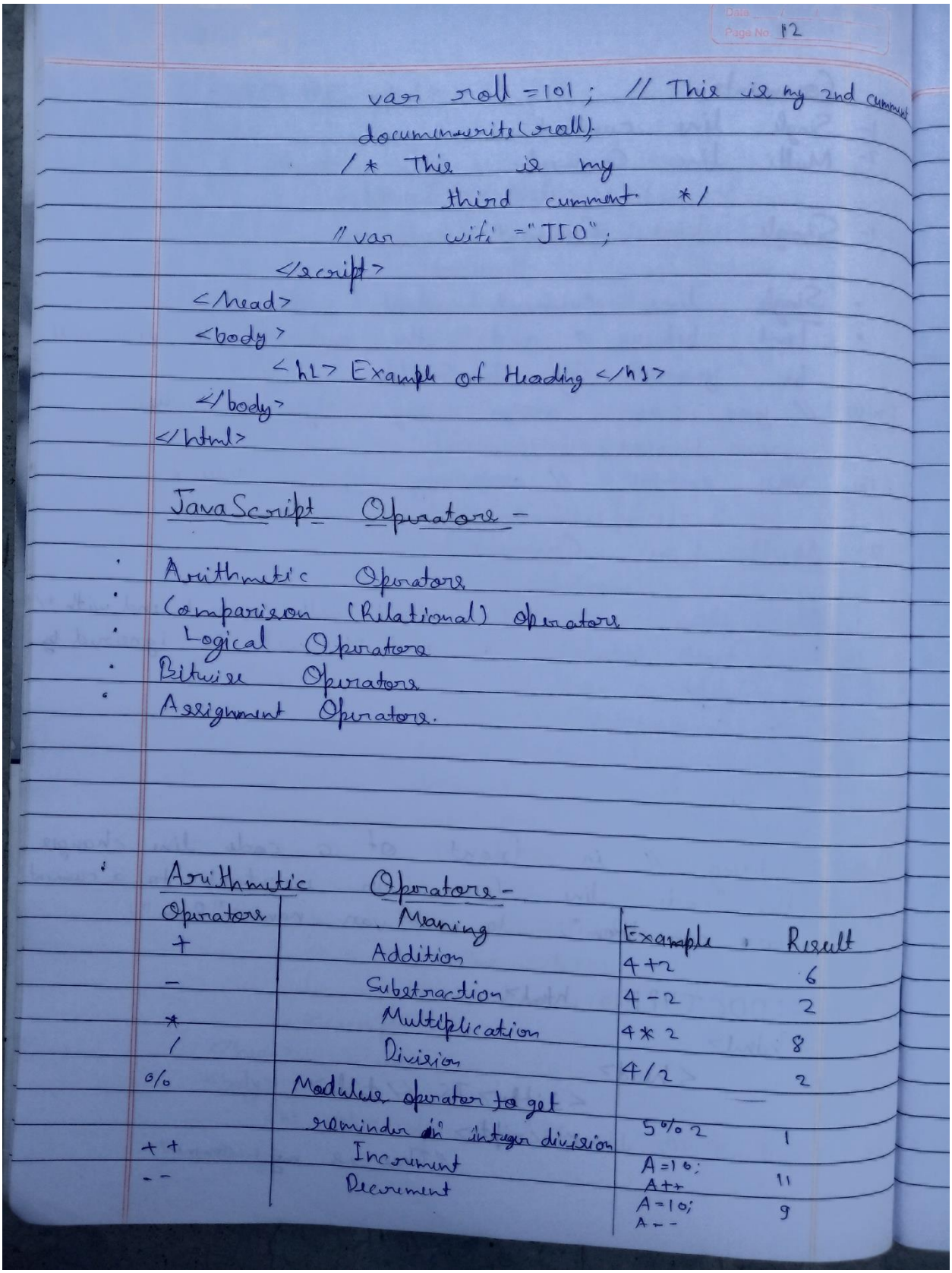
2- Multi Line Comment-

- Multi-line comment start with /* and end with */
- Any text between /* and */ will be ignored by JavaScript

Ex-(i) /* Comment line */
(ii) /* Comment
line */

Note- Adding // in front of a code line change the code line from an executable to a comment.
Ex- var name="Ram"; to // var name="Ram";

```
-> <!DOCTYPE html>
<html>
    <head>
        <title> JS </title>
        <script>
            // This is my comment
```



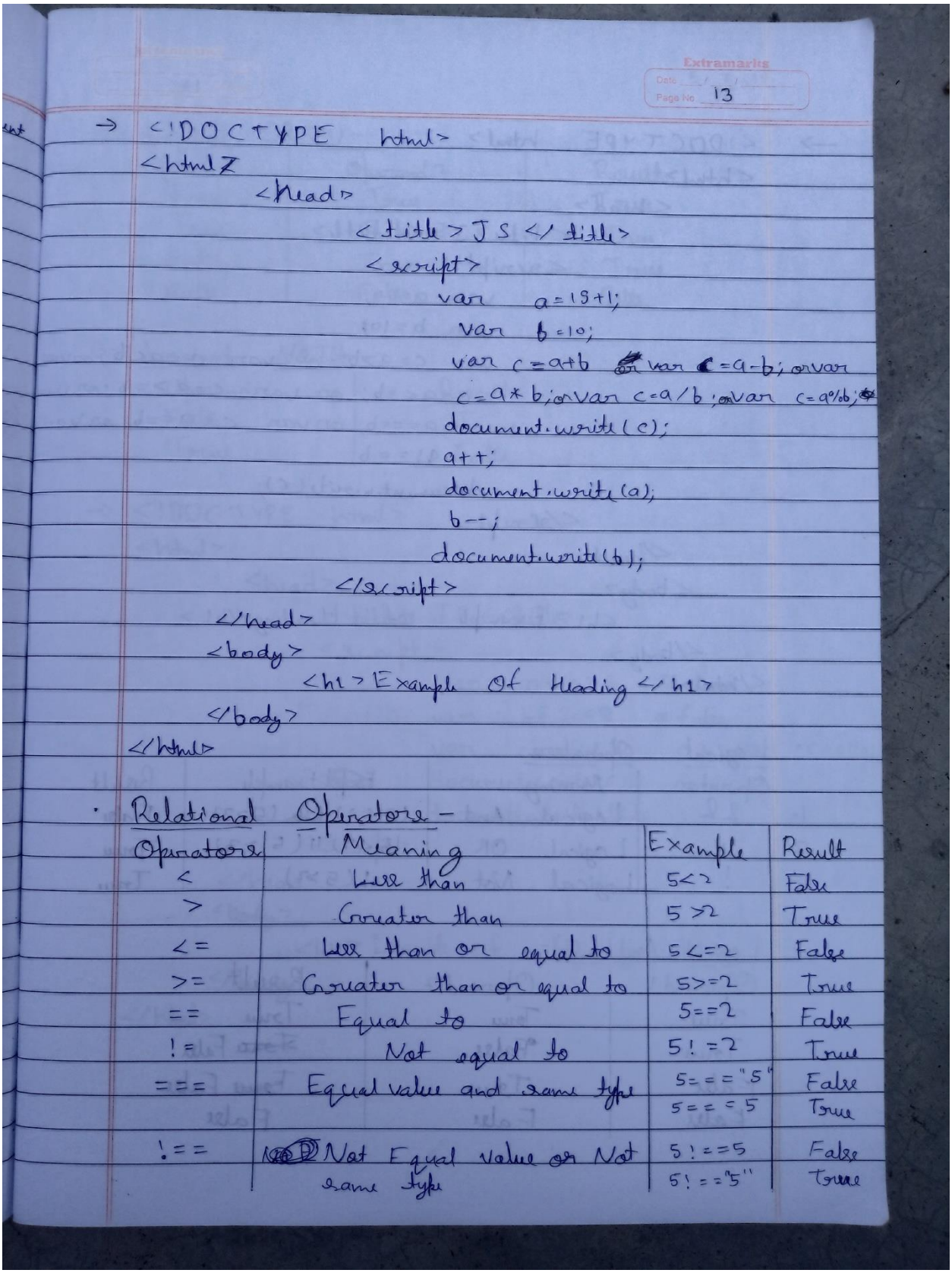
```
var roll = 101; // This is my 2nd comment
document.write(roll);
/* This is my
        third comment. */
// var wifi = "JIO";
</script>
</head>
<body>
    <h1> Example of Heading </h1>
</body>
</html>
```

JavaScript Operators -

- Arithmetic Operators
- Comparison (Relational) Operators
- Logical Operators
- Bitwise Operators
- Assignment Operators.

Arithmetic Operators -

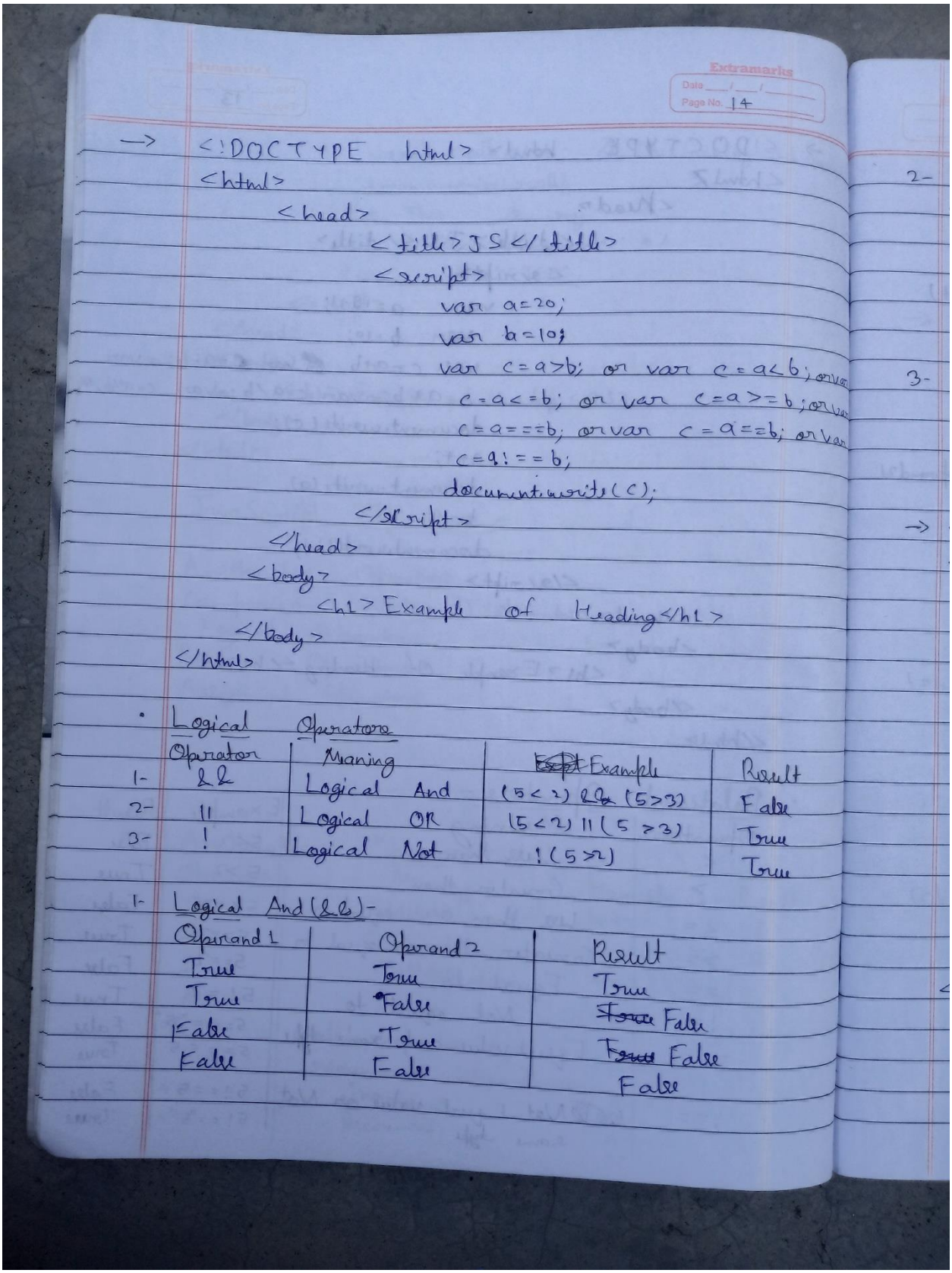
Operators	Meaning	Example	Result
+	Addition	4+2	6
-	Subtraction	4-2	2
*	Multiplication	4*2	8
/	Division	4/2	2
%	Modulus operator to get reminder in integer division	5%2	1
++	Increment	A=10; A++	11
--	Decrement	A=10; A--	9



```
<!DOCTYPE html>
<html>
    <head>
        <title> JS </title>
        <script>
            var a=19+1;
            var b=10;
            var c=a+b or var c=a-b; or var
            c=a*b; or var c=a/b; or var c=a%b;
            document.write(c);
            a++;
            document.write(a);
            b--;
            document.write(b);
        </script>
    </head>
    <body>
        <h1> Example Of Heading </h1>
    </body>
</html>
```

- **Relational Operators –**

Operators	Meaning	Example	Result
<	Less than	5<2	False
>	Greater than	5>2	True
<=	Less than or equal to	5<=2	False
>=	Greater than or equal to	5>=2	True
==	Equal to	5==2	False
!=	Not equal to	5!=2	True
===	Equal value and same type	5==="5"	False
		5===5	True
!==	Not Equal value or Not same type	5!==5	False
		5!=="5"	True



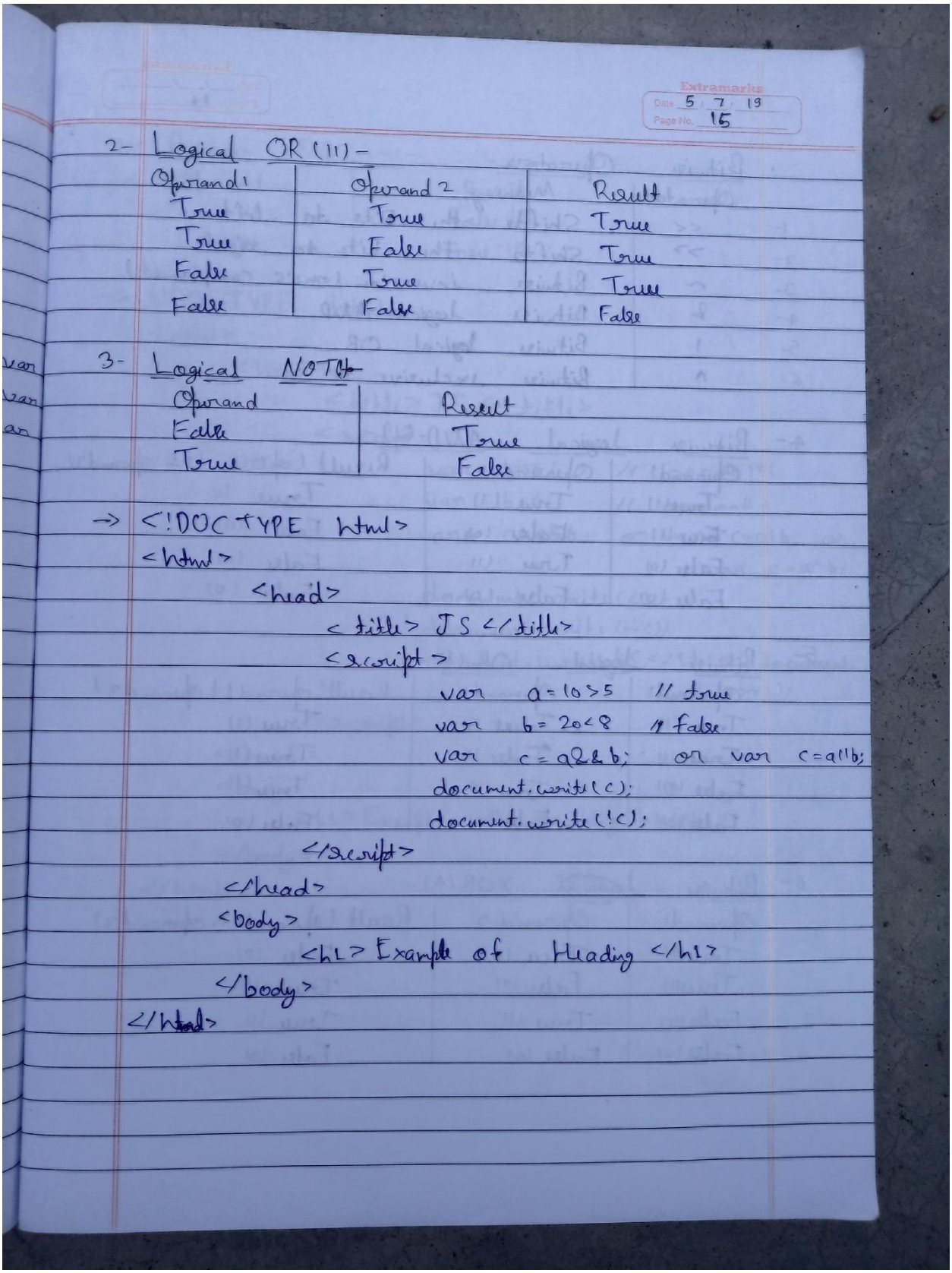
```
<!DOCTYPE html>
<html>
    <head>
        <title>JS</title>
        <script>
            var a=20;
            var b=10;
            var c=a>b; or var c=a<b; or var
             c=a<=b; or var c=a>=b; or var
             c=a===b; or var c=a==b; or var
             c=a!==b;
             document.write(c);
        </script>
    </head>
    <body>
        <h1>Example of Heading</h1>
    </body>
</html>
```

- Logical Operators

	Operator	Maning	Example	Result
1-	&&	Logical And	(5<2) && (5>3)	False
2-	||	Logical OR	(5<2) || (5>3)	True
3-	!	Logical Not	!(5>2)	True

1- Logical And (&&)-

Operand 1	Operand 2	Result
True	True	True
True	False	False
False	True	False
False	False	False



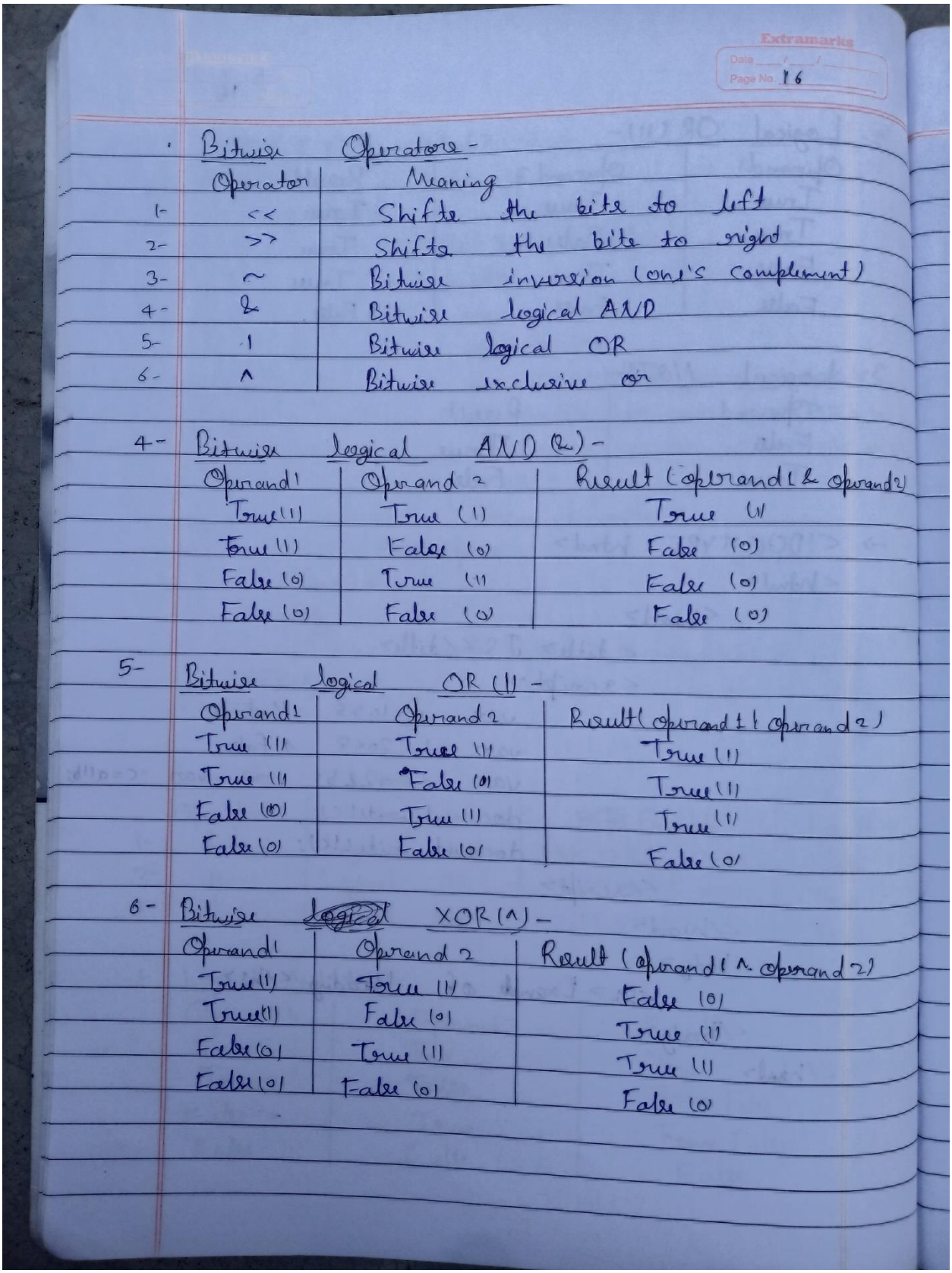
2- Logical OR (||) -

Operand1	Operand 2	Result
True	True	True
True	False	True
False	True	True
False	False	False

3- Logical NOT(!)

Operand	Result
False	True
True	False

→
```
<!DOCTYPE html>
<html>
    <head>
        <title> JS </title>
        <script>
            var a = 10>5  // true
            var b = 20<8  // false
            var c = a&&b;  or var c=a||b;
            document.write(c);
            document.write(!c);
        </script>
    </head>
    <body>
        <h1> Example of Heading </h1>
    </body>
</html>
```



Extramarks
Page No. 16

- Bitwise Operators -

	Operator	Meaning
1-	<<	Shifts the bits to left
2-	>>	Shifts the bits to right
3-	~	Bitwise inversion (one's complement)
4-	&	Bitwise logical AND
5-	|	Bitwise logical OR
6-	^	Bitwise exclusive or

4- Bitwise logical AND (&) -

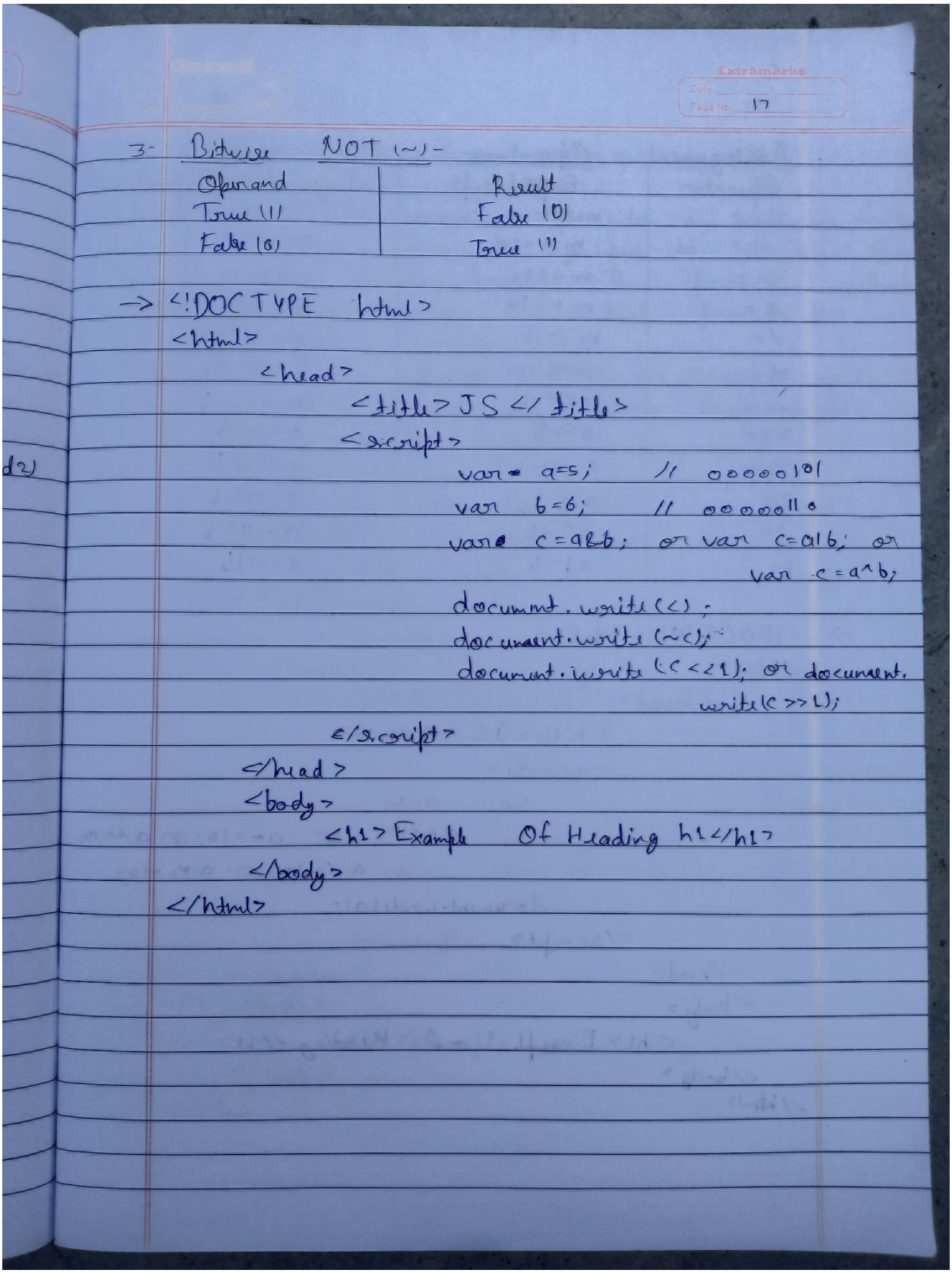
Operand1	Operand 2	Result (operand1 & operand2)
True (1)	True (1)	True (1)
True (1)	False (0)	False (0)
False (0)	True (1)	False (0)
False (0)	False (0)	False (0)

5- Bitwise logical OR (|) -

Operand1	Operand 2	Result (operand1 | operand2)
True (1)	True (1)	True (1)
True (1)	False (0)	True (1)
False (0)	True (1)	True (1)
False (0)	False (0)	False (0)

6- Bitwise logical XOR (^) -

Operand1	Operand 2	Result (operand1 ^ operand2)
True (1)	True (1)	False (0)
True (1)	False (0)	True (1)
False (0)	True (1)	True (1)
False (0)	False (0)	False (0)

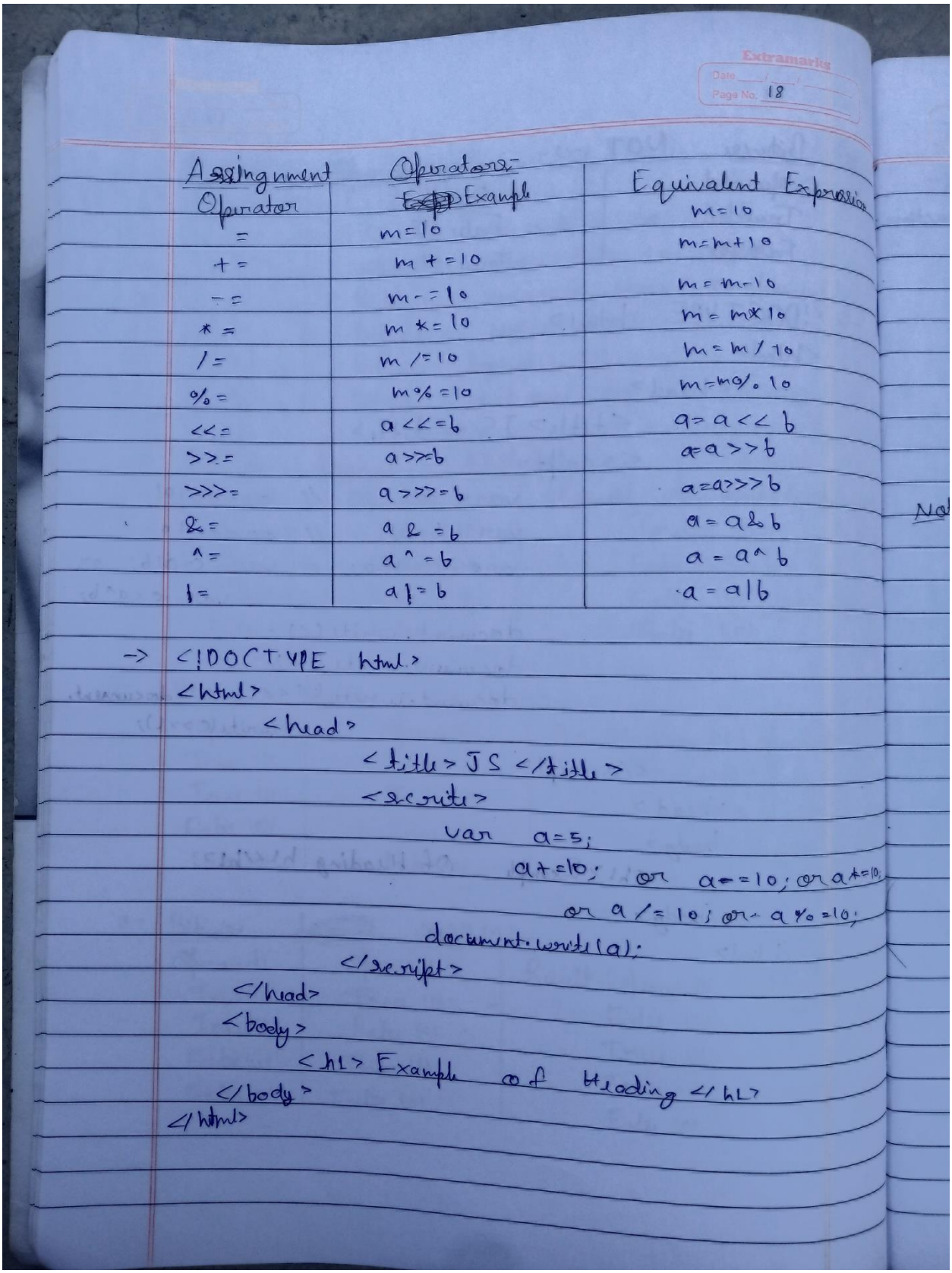


Page No. 17

3- Bitwise NOT (~) -

Operand	Result
True (1)	False (0)
False (0)	True (1)

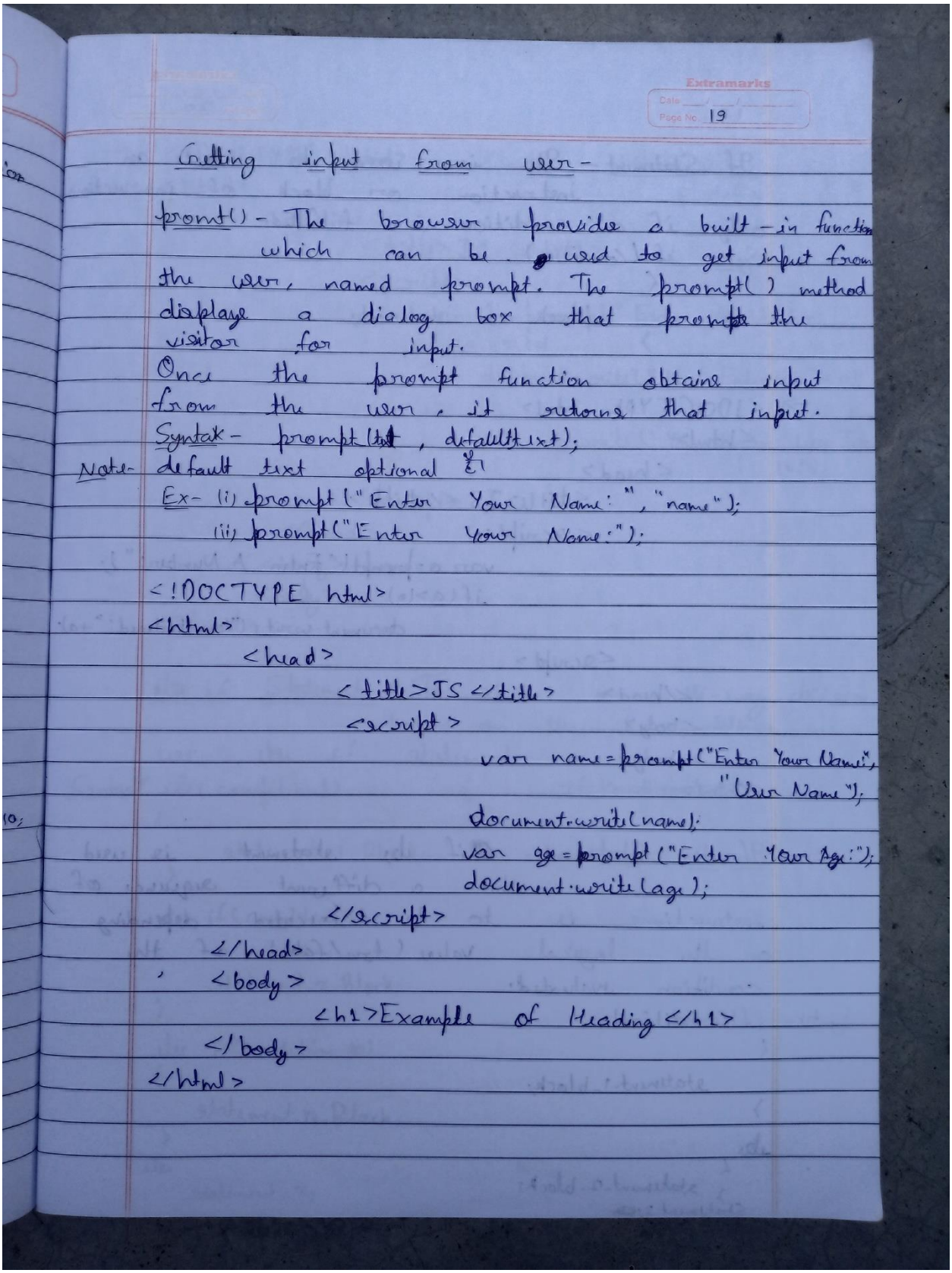
```
<!DOCTYPE html>
<html>
    <head>
        <title> JS </title>
        <script>
            var a=5;    // 00000101
            var b=6;    // 00000110
            var c=a&b; or var c=a|b; or
                                var c=a^b;
            document.write(c);
            document.write(~c);
            document.write(c<<1); or document.
                                write(c>>1);
        </script>
    </head>
    <body>
        <h1> Example Of Heading h1 </h1>
    </body>
</html>
```



Extramarks
Date / /
Page No. 18

Assignment Operator	Operator Example	Equivalent Expression
=	m=10	m=10
+=	m+=10	m=m+10
-=	m-=10	m=m-10
*=	m*=10	m=m*10
/=	m/=10	m=m/10
%=	m%=10	m=m%10
<<=	a<<=b	a=a<<b
>>=	a>>=b	a=a>>b
>>>=	a>>>=b	a=a>>>b
&=	a&=b	a=a&b
^=	a^=b	a=a^b
|=	a|=b	a=a|b

-> <!DOCTYPE html>
<html>
<head>
<title> JS </title>
<script>
var a=5;
a+=10; or a-=10; or a*=10;
or a/=10; or a%=10;
document.write(a);
</script>
</head>
<body>
<h1> Example of Heading </h1>
</body>
</html>

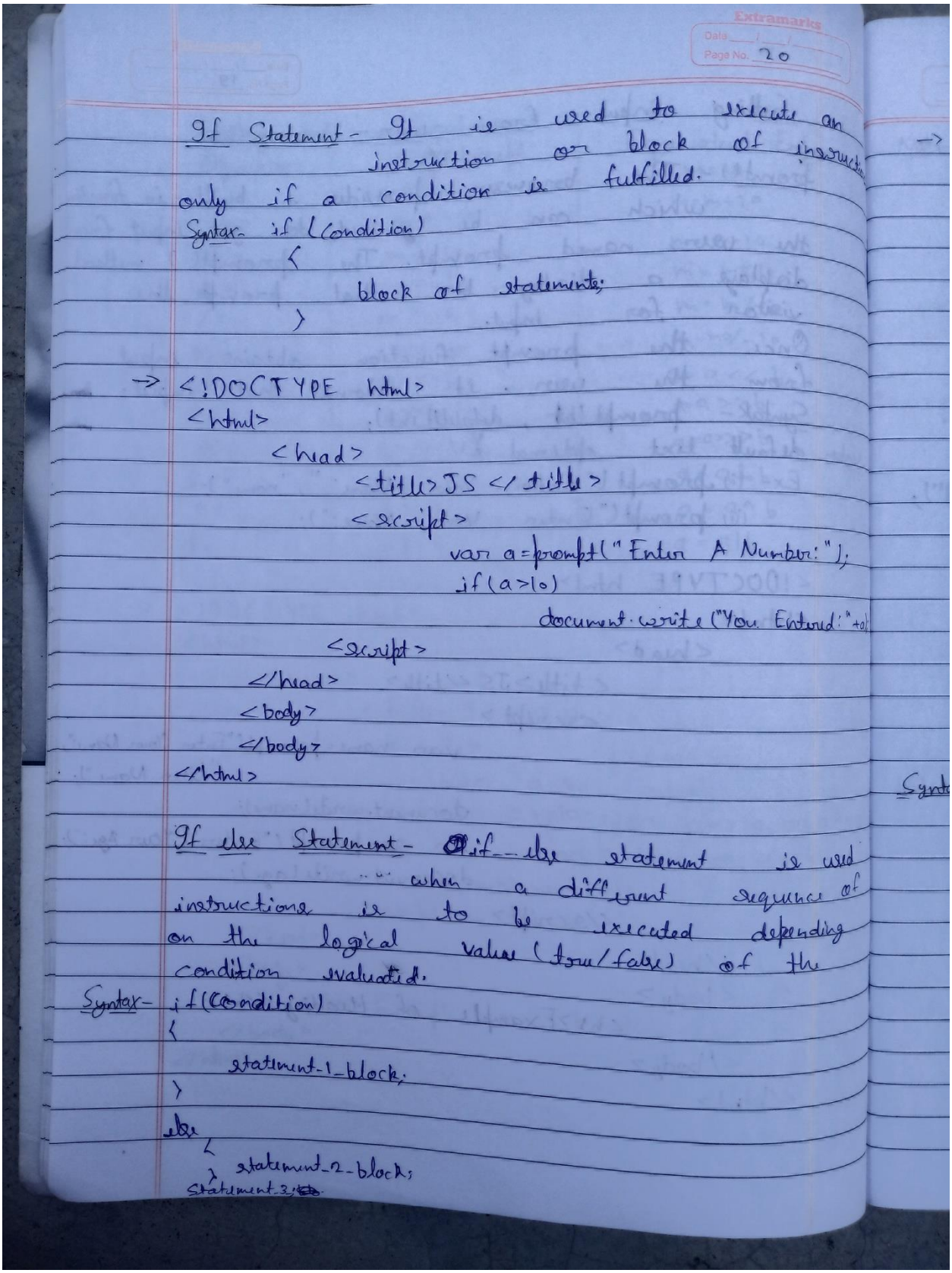


Extramarks
Date
Page No. 19

Getting input from user -

prompt() - The browser provide a built-in function which 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.
Syntax - prompt(text, defaultText);
Note - default text optional hai
Ex - (i) prompt("Enter Your Name: ", "name");
(ii) prompt("Enter Your Name:");

```
<!DOCTYPE html>
<html>
    <head>
        <title>JS</title>
        <script>
            var name = prompt("Enter Your Name",
                              "User Name");
            document.write(name);
            var age = prompt("Enter Your Age:");
            document.write(age);
        </script>
    </head>
    <body>
        <h1>Example of Heading</h1>
    </body>
</html>
```



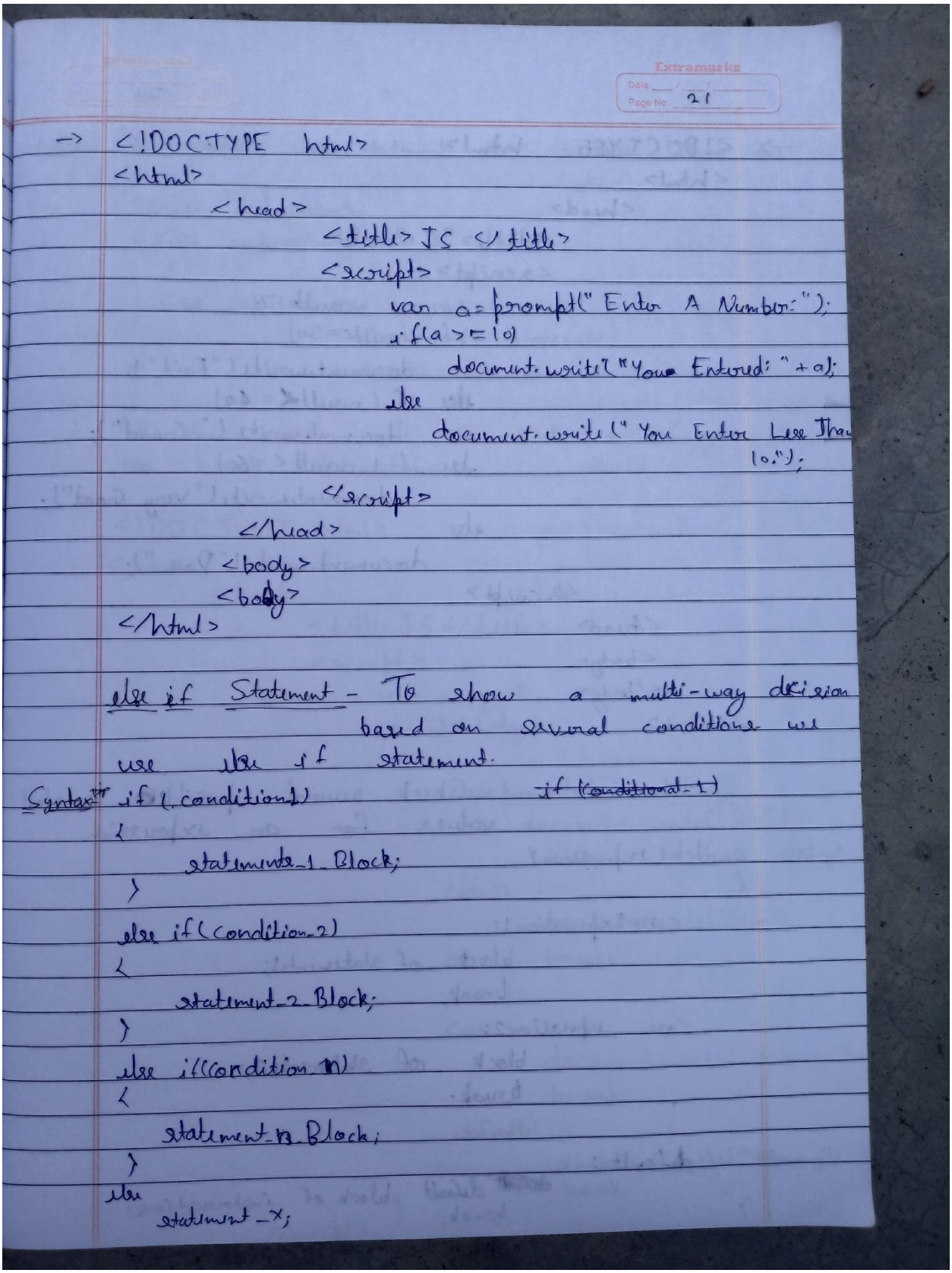
If Statement - It is used to execute an instruction or block of instruction only if a condition is fulfilled.

Syntax- if (condition)
{
block of statements;
}

```
<!DOCTYPE html>
<html>
    <head>
        <title>JS</title>
        <script>
            var a=prompt("Enter A Number:");
            if(a>10)
                document.write("You Entered:"+a)
        <script>
    </head>
    <body>
    </body>
</html>
```

If else Statement - if...else statement is used when a different sequence of instructions is to be executed depending on the logical value (true/false) of the condition evaluated.

Syntax-
```
if(Condition)
{
    statement-1-block;
}
else
{
    statement-2-block;
}
statement-3;
```



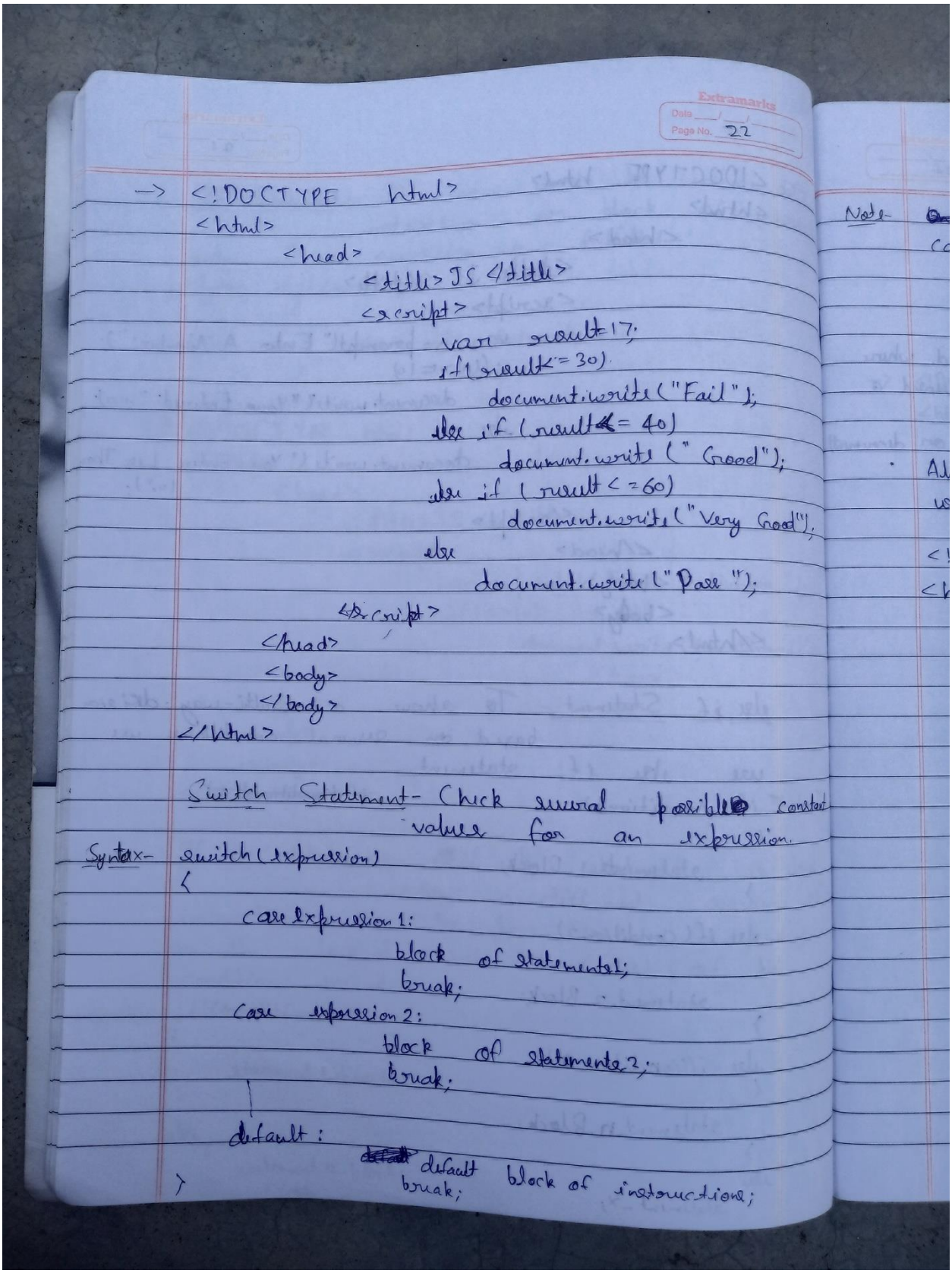
Extramarks
Date / /
Page No 21

```
-> <!DOCTYPE html>
<html>
    <head>
        <title> JS </title>
        <script>
            var a=prompt(" Enter A Number: ");
            if(a >=10)
                document.write(" Your Entered: " +a);
            else
                document.write(" You Enter Less Than
                                10.");
        </script>
    </head>
    <body>
    <body>
</html>
```

else if Statement - To show a multi-way decision based on several conditions we use the else if statement.

Syntax:

```
if (condition1)
{
    statements_1_Block;
}
else if(condition_2)
{
    statement_2_Block;
}
else if(condition_n)
{
    statement_n_Block;
}
else
    statement_x;
```

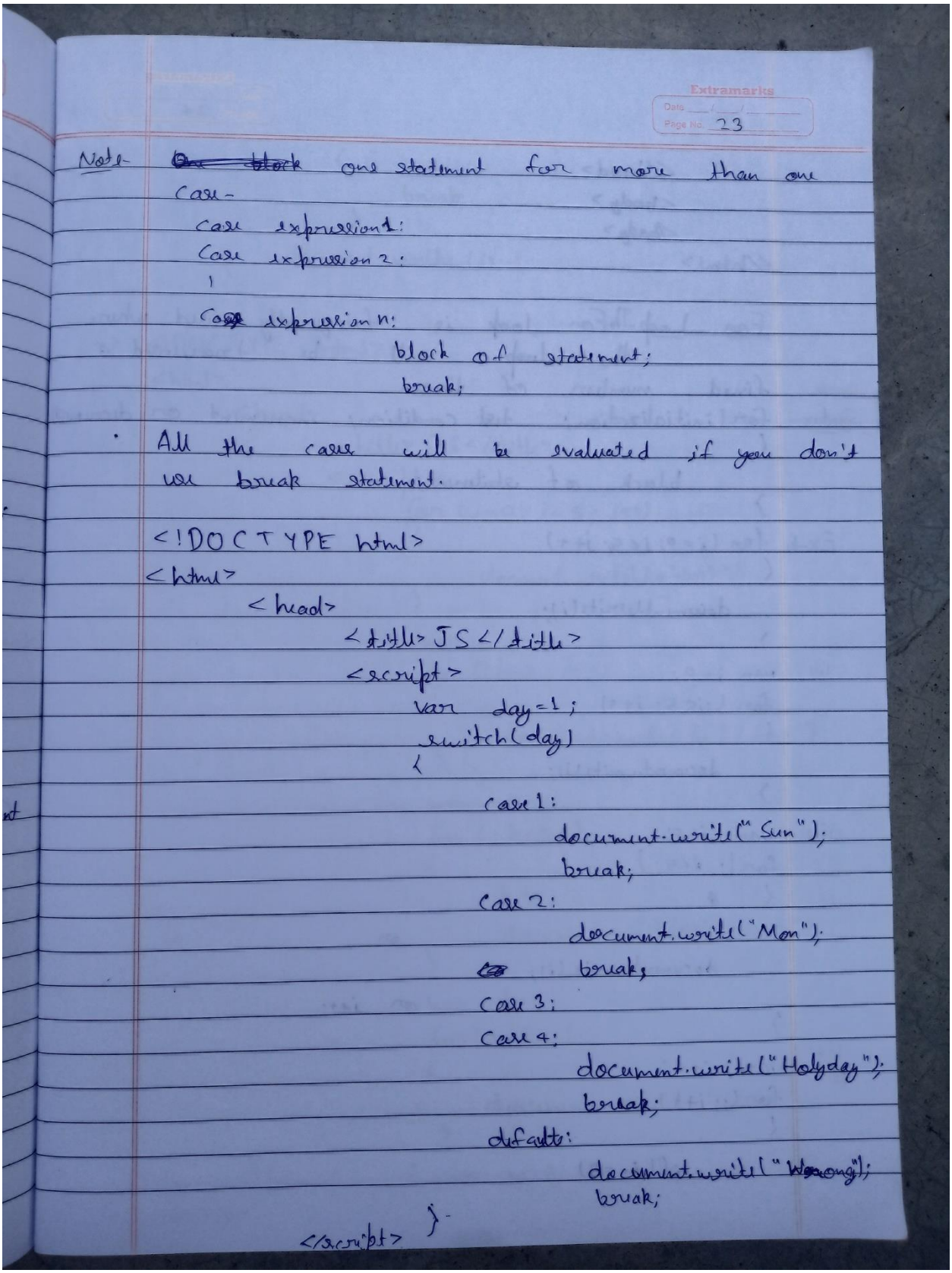


```
<!DOCTYPE html>
<html>
    <head>
        <title> JS </title>
        <script>
            var result=17;
            if(result<=30)
                document.write("Fail");
            else if (result<=40)
                document.write("Good");
            else if (result<=60)
                document.write("Very Good");
            else
                document.write("Pass");
        </script>
    </head>
    <body>
    </body>
</html>
```

Switch Statement- Check several possible constant values for an expression.

Syntax-

```
switch(expression)
{
    case expression 1:
            block of statements1;
            break;
    case expression 2:
            block of statements2;
            break;
    |
    default:
            default block of instructions;
            break;
}
```



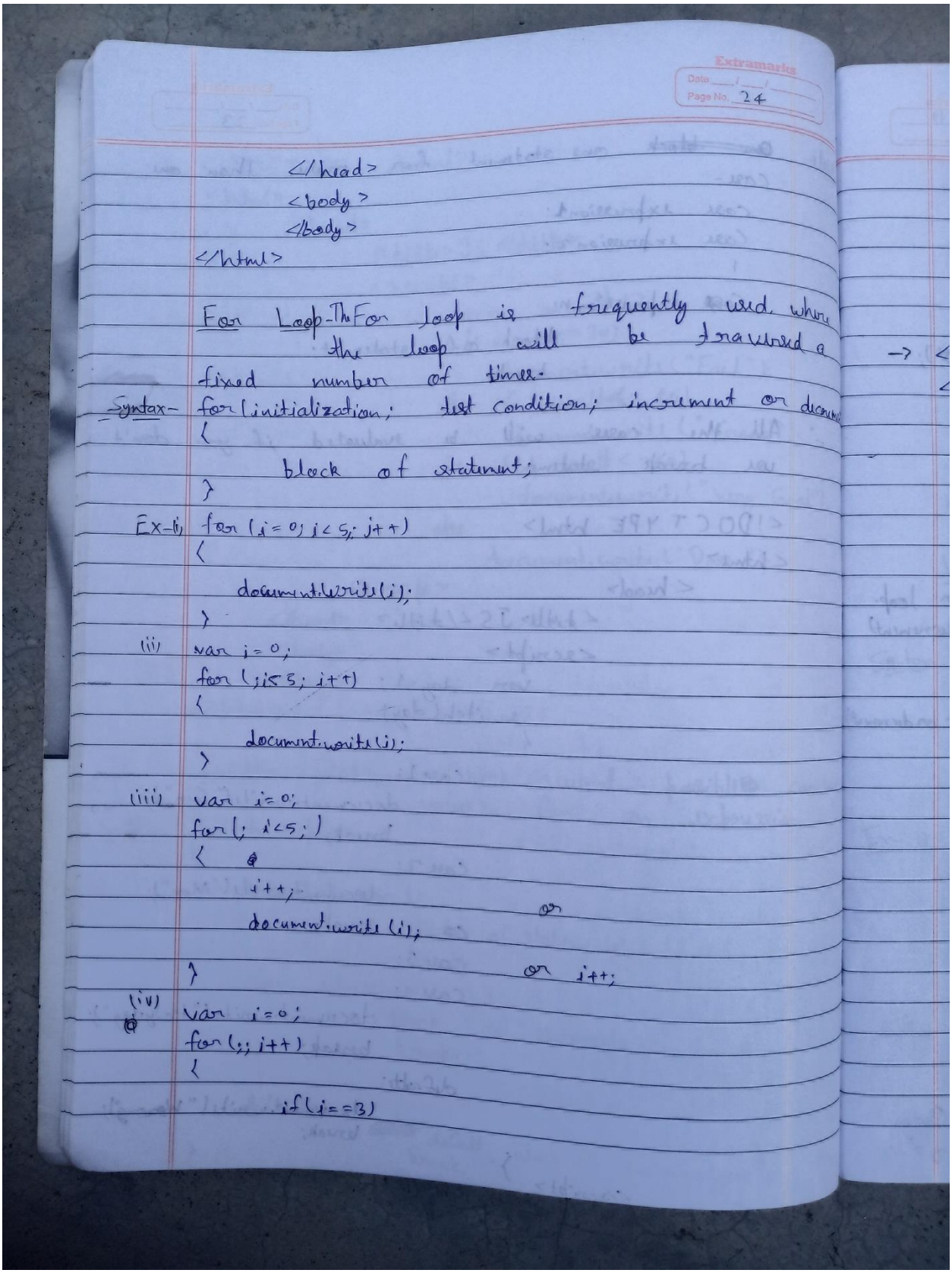
Extramarks
Date / /
Page No. 23

Note- ~~Block~~ one statement for more than one case -

```
case expression1:
case expression2:
.
.
case expression n:
        block of statement;
        break;
```

- All the case will be evaluated if you don't use break statement.

```
<!DOCTYPE html>
<html>
    <head>
        <title> JS </title>
        <script>
            var day=1;
            switch(day)
            {
                case 1:
                    document.write("Sun");
                    break;
                case 2:
                    document.write("Mon");
                    break;
                case 3:
                case 4:
                    document.write("Holyday");
                    break;
                default:
                    document.write("Wrong");
                    break;
        </script>
```



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

For Loop - The For loop is frequently used, where the loop will be traversed a fixed number of times.

Syntax-
```
for(initialization; test condition; increment or decrement)
{
    block of statement;
}
```

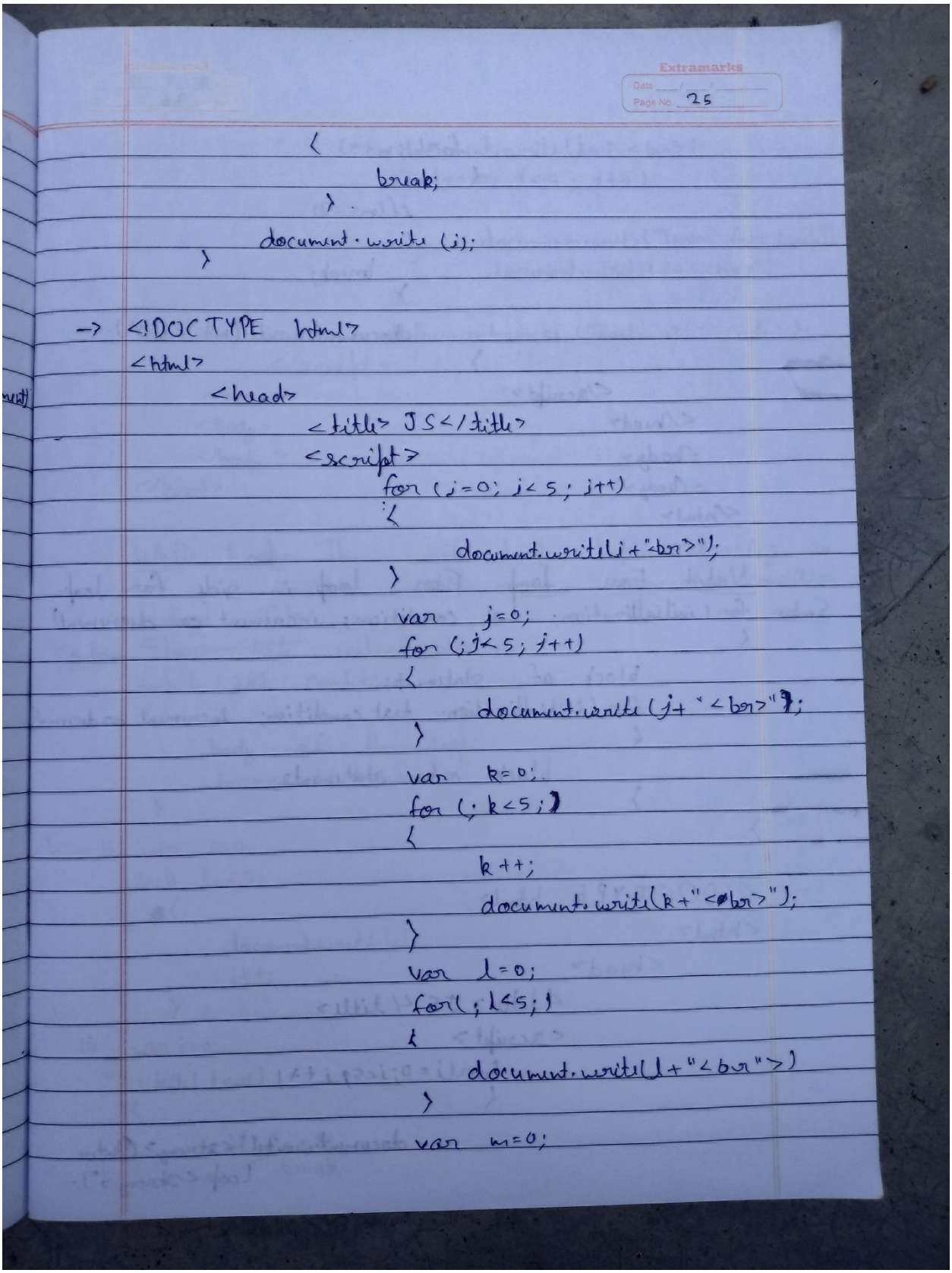
Ex-(i)
```
for (i=0; i<5; i++)
{
    document.write(i);
}
```

(ii)
```
var i=0;
for (;i<5; i++)
{
    document.write(i);
}
```

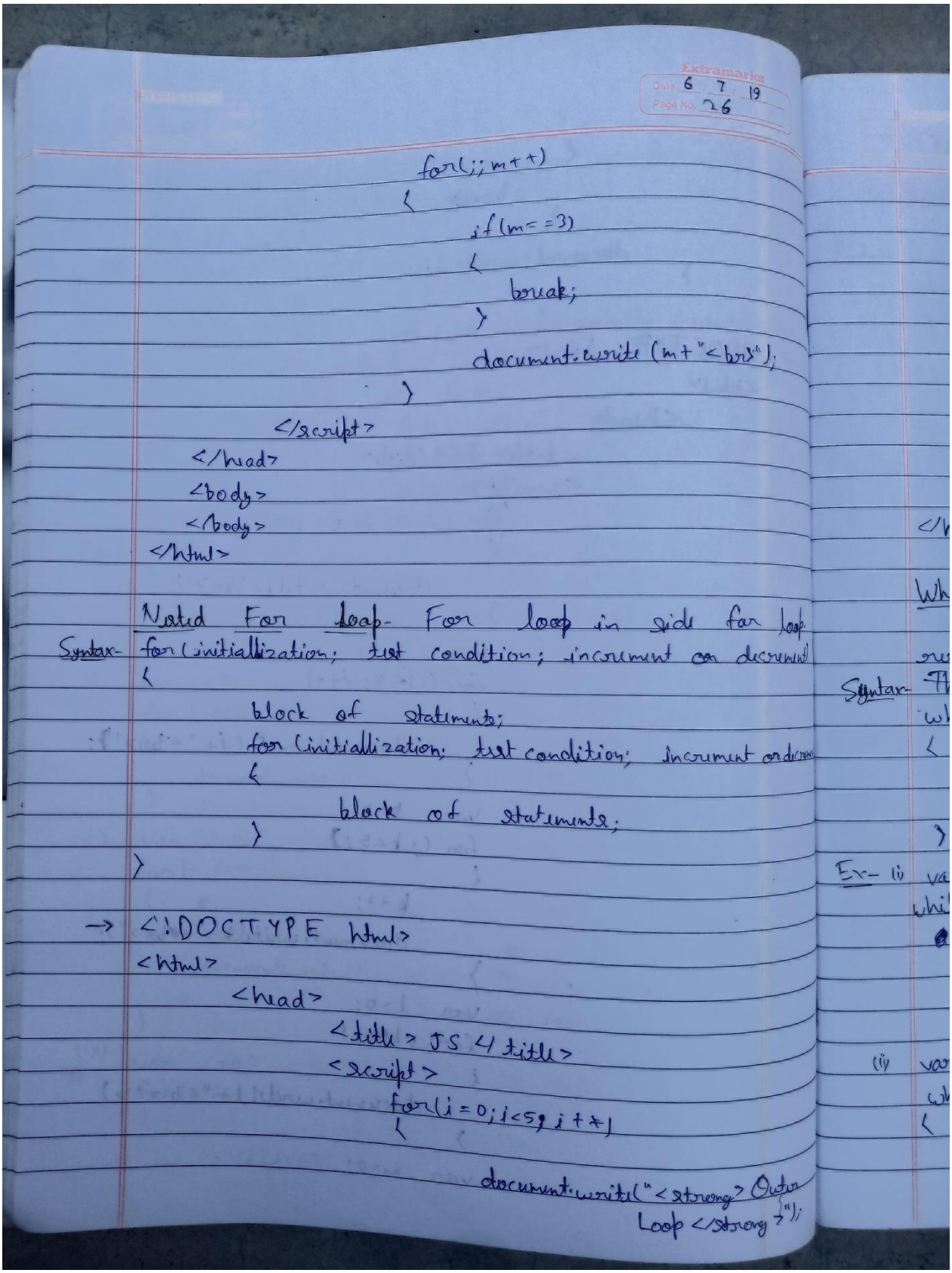
(iii)
```
var i=0;
for(; i<5; )
{
    i++;
    document.write(i);
}
```
or

or i++;

(iv)
```
var i=0;
for(;; i++)
{
    if(i==3)
```



```
        {
            break;
        }
        document.write(i);
    }

-> <!DOCTYPE html>
<html>
    <head>
        <title> JS </title>
        <script>
            for (j=0; j<5; j++)
            {
                document.write(i+"<br>");
            }
            var j=0;
            for (;j<5;j++)
            {
                document.write(j+"<br>");
            }
            var k=0;
            for (;k<5;)
            {
                k++;
                document.write(k+"<br>");
            }
            var l=0;
            for(;l<5;)
            {
                document.write(l+"<br">)
            }
            var m=0;
```

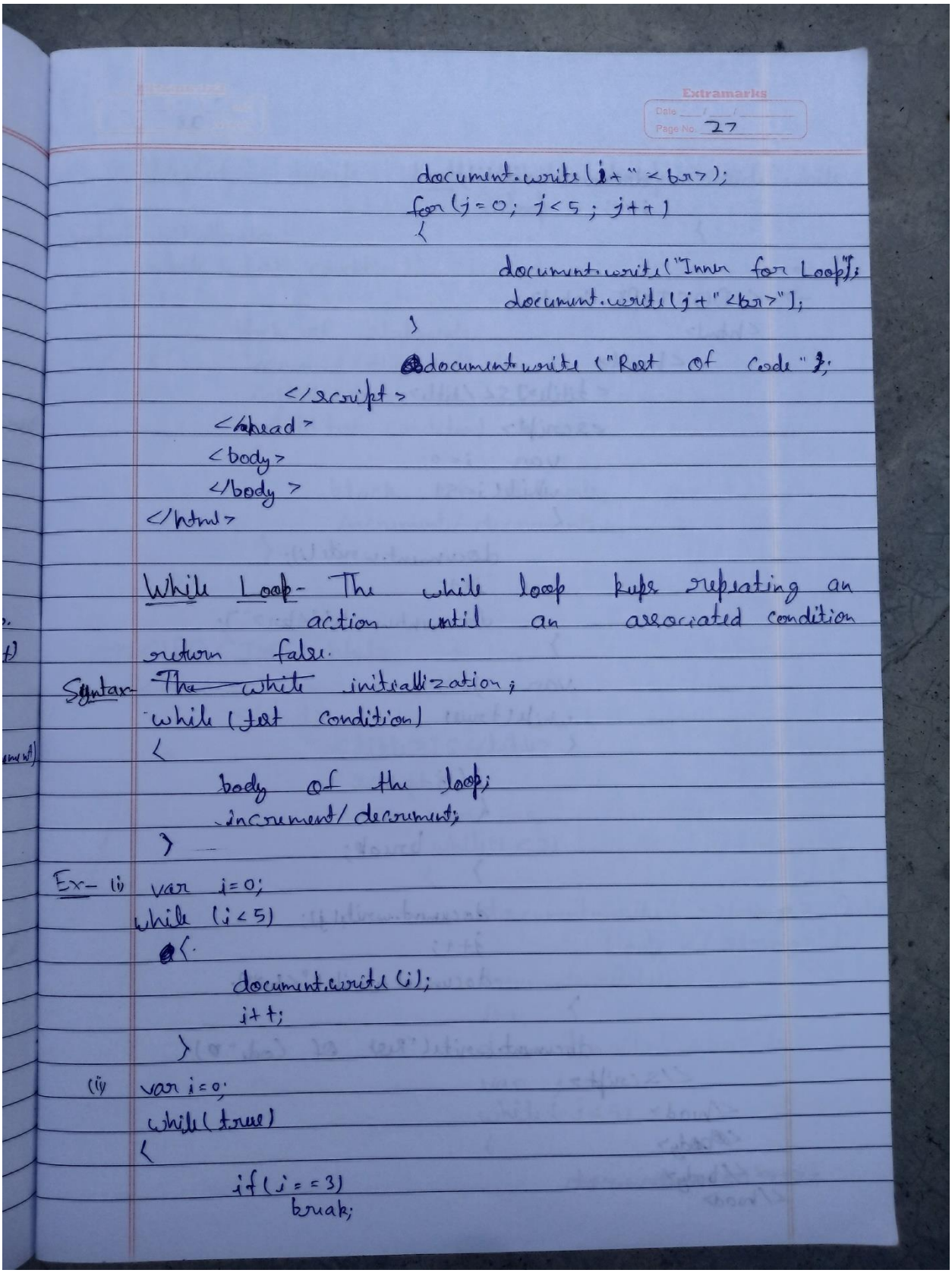


```
        for(;;m++)
        {
            if(m==3)
            {
                break;
            }
            document.write(m+"<br>");
        }
    </script>
</head>
<body>
</body>
</html>
```

Nested For loop- For loop in side for loop.

Syntax- for(initiallization; test condition; increment or decrement)
{
    block of statements;
    for(initiallization; test condition; increment or decrement)
    {
        block of statements;
    }
}

→
```
<!DOCTYPE html>
<html>
    <head>
        <title> JS </title>
        <script>
            for(i=0;i<5;i++)
            {
                document.write("<strong> Outer
                Loop </strong>");
```



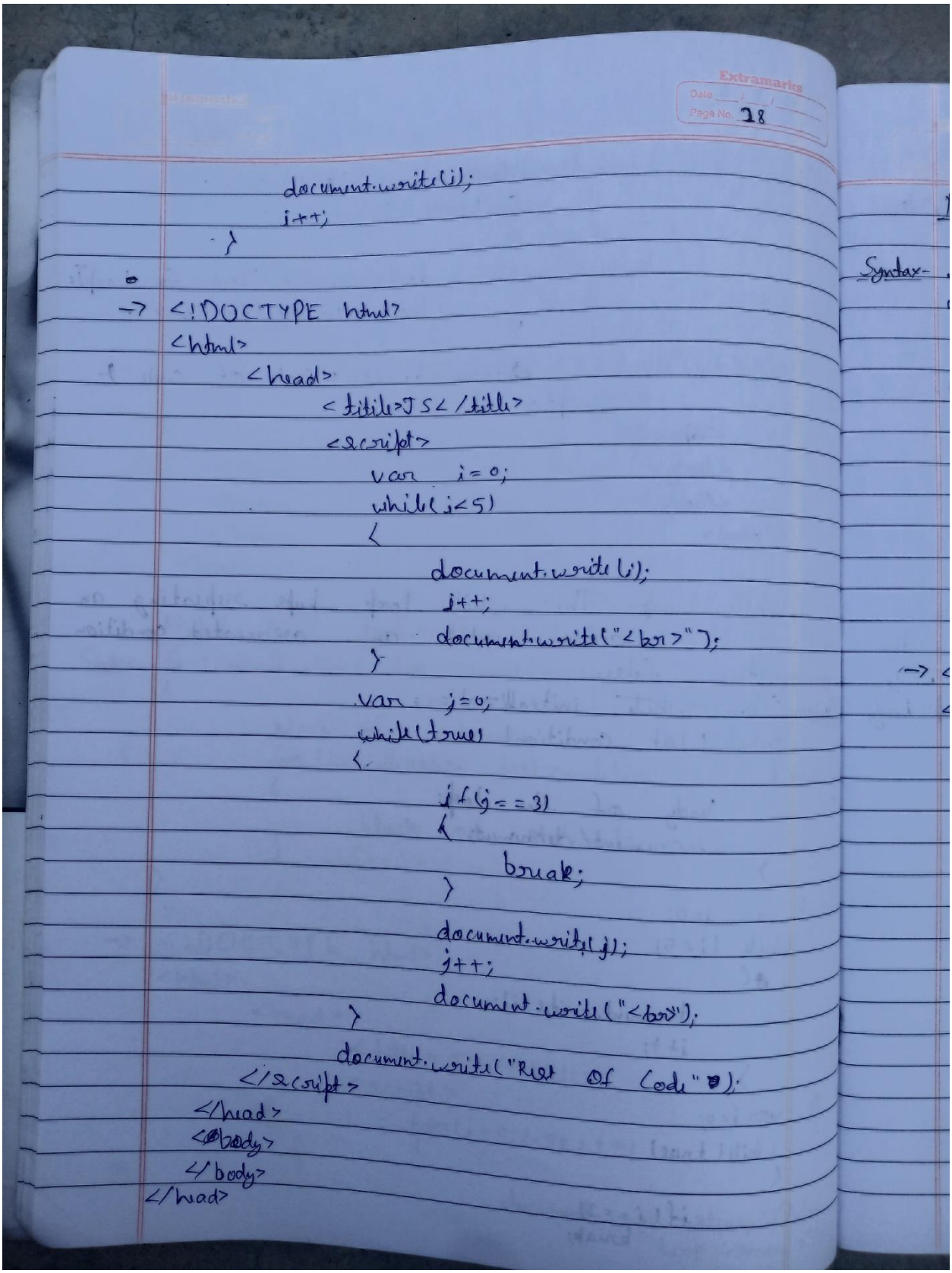
```
        document.write(i+"<br>");
        for(j=0; j<5; j++)
        {
            document.write("Inner for Loop");
            document.write(j+"<br>");
        }
        document.write("Rest of code");
    </script>
</head>
<body>
</body>
</html>
```

While Loop- The while loop keeps repeating an action until an associated condition return false.

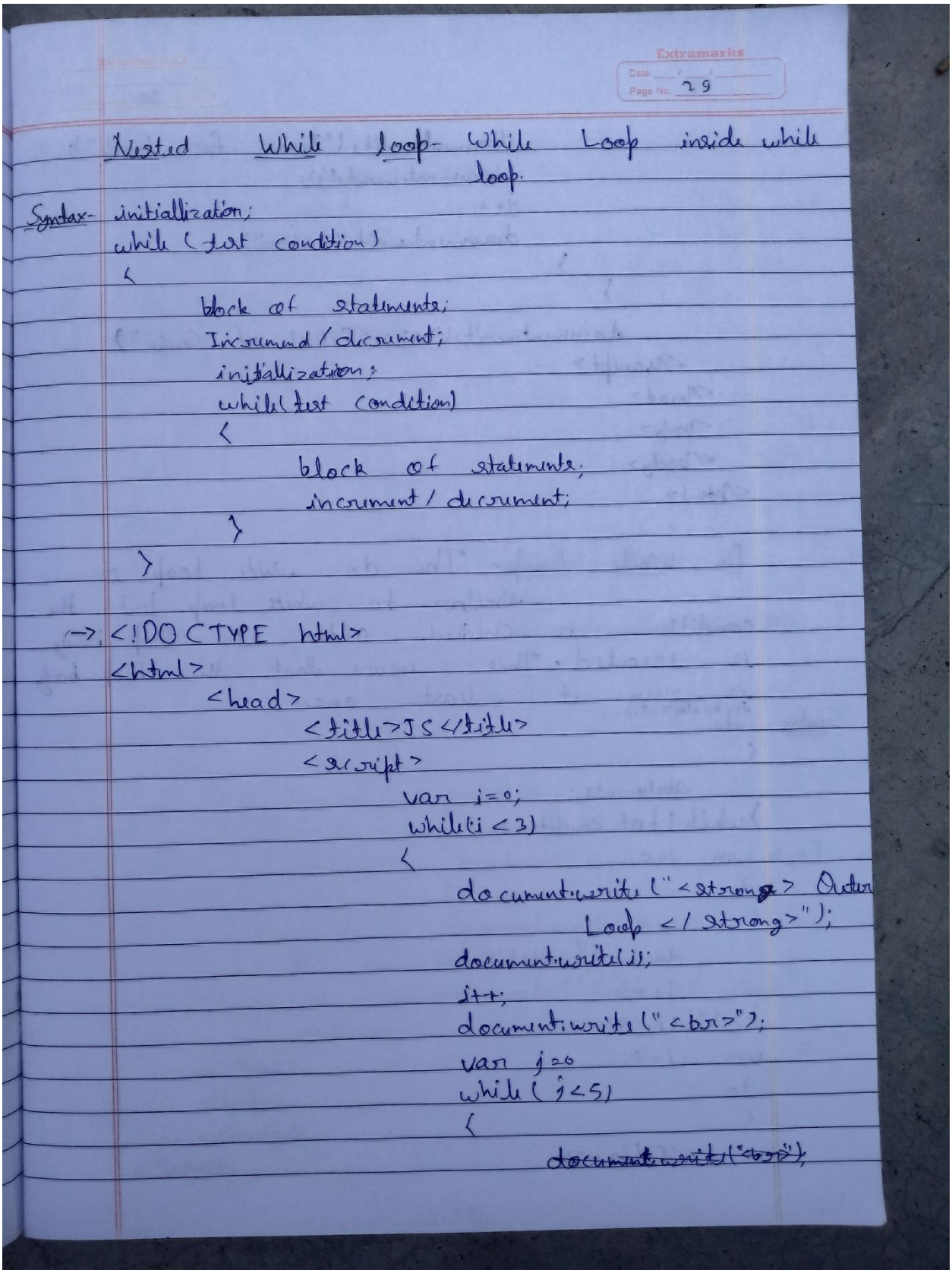
Syntax- The while initialization;
while (test condition)
{
    body of the loop;
    increment/decrement;
}

Ex- (i)
```
var i=0;
while (i<5)
{
    document.write(i);
    i++;
}
```

(ii)
```
var i=0;
while(true)
{
    if(i==3)
        break;
```



```
        document.write(i);
        i++;
    }

-> <!DOCTYPE html>
<html>
    <head>
        <title>JS</title>
        <script>
            var i=0;
            while(i<5)
            {
                document.write(i);
                i++;
                document.write("<br>");
            }
            var j=0;
            while(true)
            {
                if(j==3)
                {
                    break;
                }
                document.write(j);
                j++;
                document.write("<br>");
            }
            document.write("Rest Of Code");
        </script>
    </head>
    <body>
    </body>
</html>
```

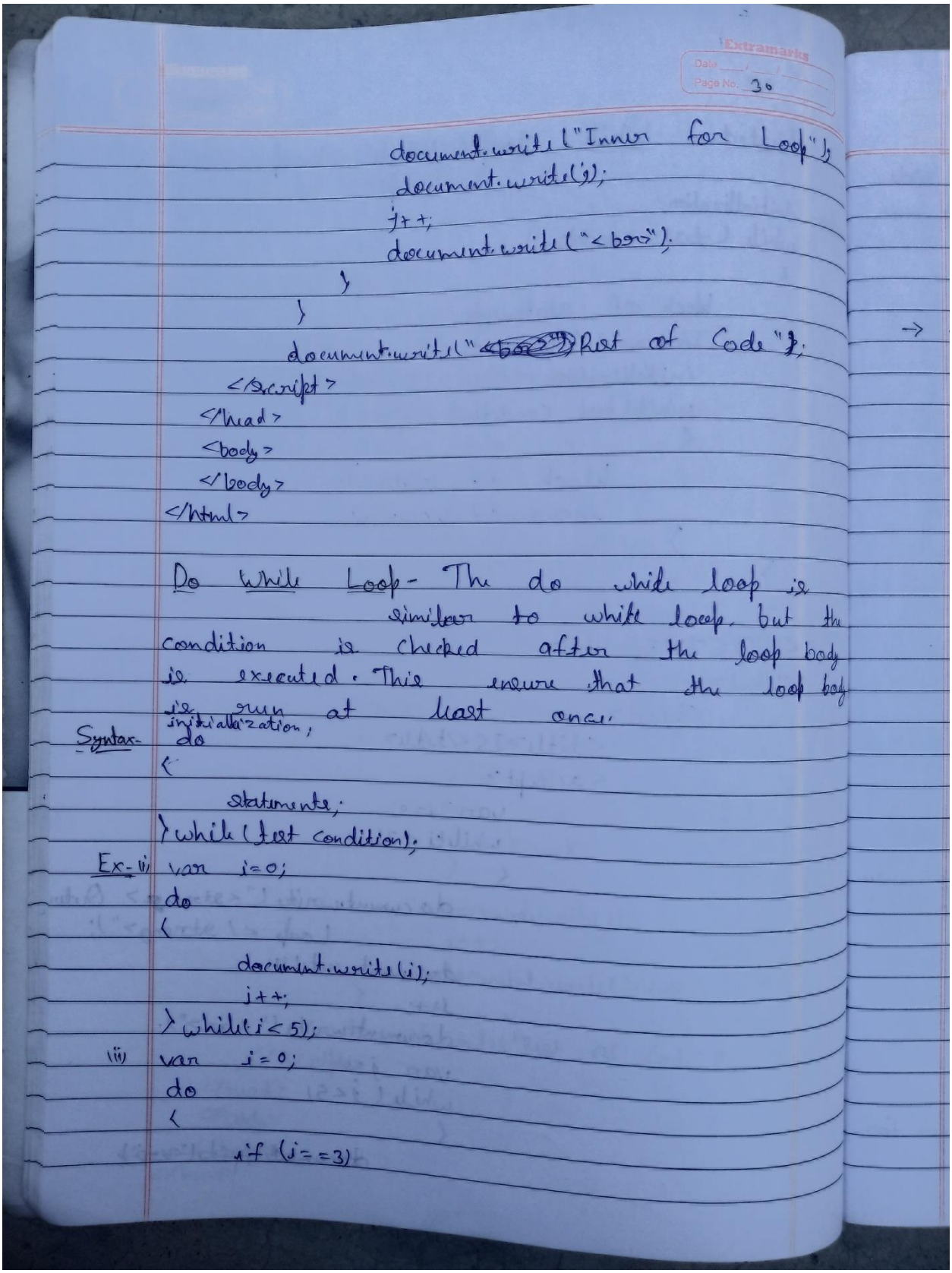


Extramarks
Date
Page No. 29

Nested While loop- While Loop inside while loop.

Syntax-
```
initiallization;
while (test condition)
{
    block of statements;
    Increment / decrement;
    initiallization;
    while(test condition)
    {
        block of statements;
        increment / decrement;
    }
}
```

→
```
<!DOCTYPE html>
<html>
    <head>
        <title>JS</title>
        <script>
            var i=0;
            while(i<3)
            {
                document.write("<strong> Outer
                    Loop </strong>");
                document.write(i);
                i++;
                document.write("<br>");
                var j=0
                while(j<5)
                {
                    document.write("<br>");
```



```
            document.write("Inner for Loop");
            document.write(j);
            j++;
            document.write("<br>");
        }
    }
    document.write("Rest of Code");
</script>
</head>
<body>
</body>
</html>
```

Do While Loop - The do while loop is similar to while loop but the condition is checked after the loop body is executed. This ensure that the loop body is run at least once.

Syntax-

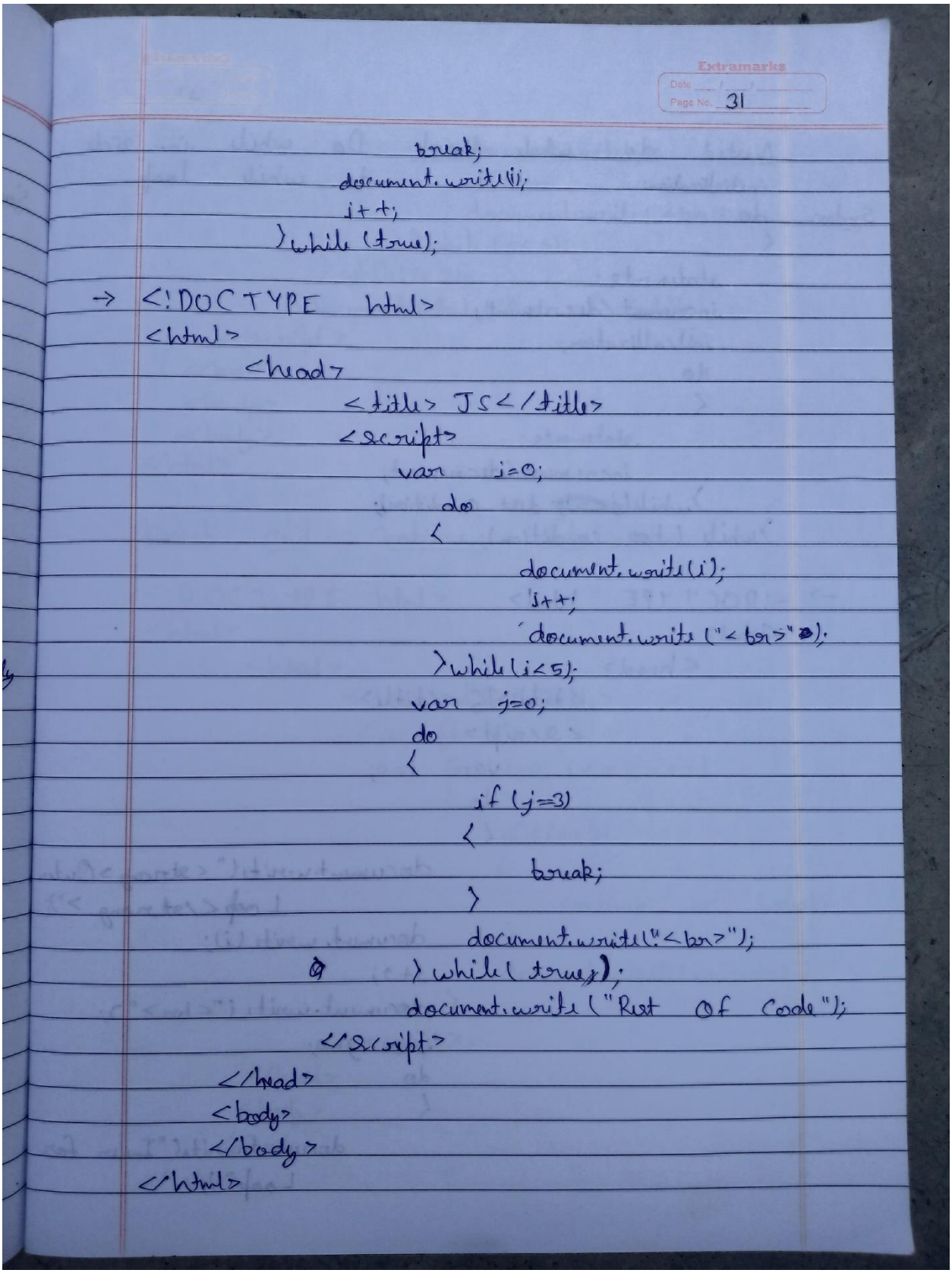
```
initialization;
do
{
    statements;
} while (test condition);
```

Ex- (i)

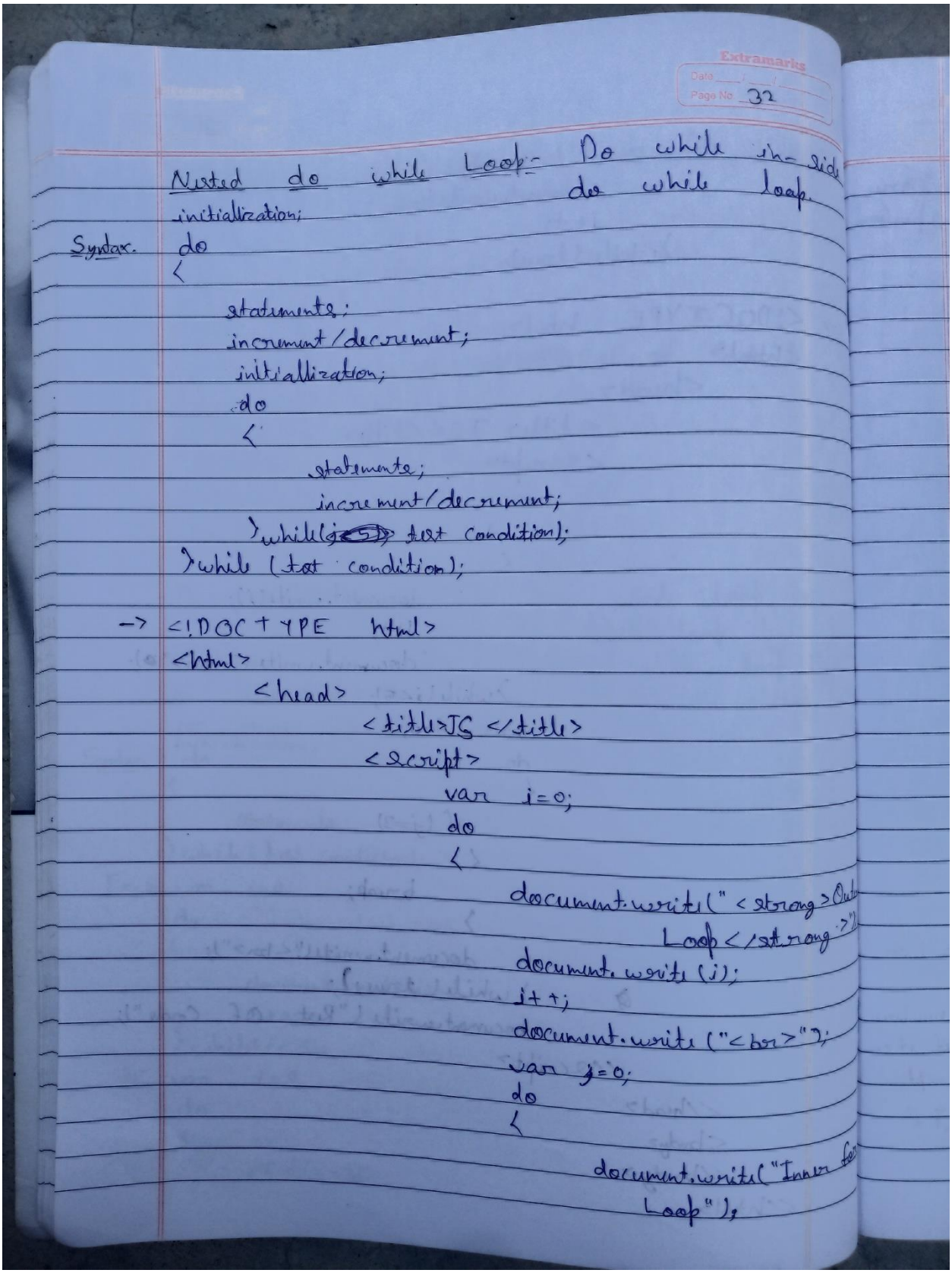
```
var i=0;
do
{
    document.write(i);
    i++;
} while(i<5);
```

(ii)

```
var i=0;
do
{
    if (i==3)
```



```
                    break;
                document.write(i);
                i++;
            }while (true);

→ <!DOCTYPE html>
<html>
    <head>
        <title> JS</title>
        <script>
            var i=0;
              do
              {
                  document.write(i);
                  i++;
                  document.write ("<br>");
              }while(i<5);
            var j=0;
            do
            {
                if (j=3)
                {
                    break;
                }
                document.write("<br>");
            }while( true);
            document.write ("Rest Of Code");
        </script>
    </head>
    <body>
    </body>
</html>
```

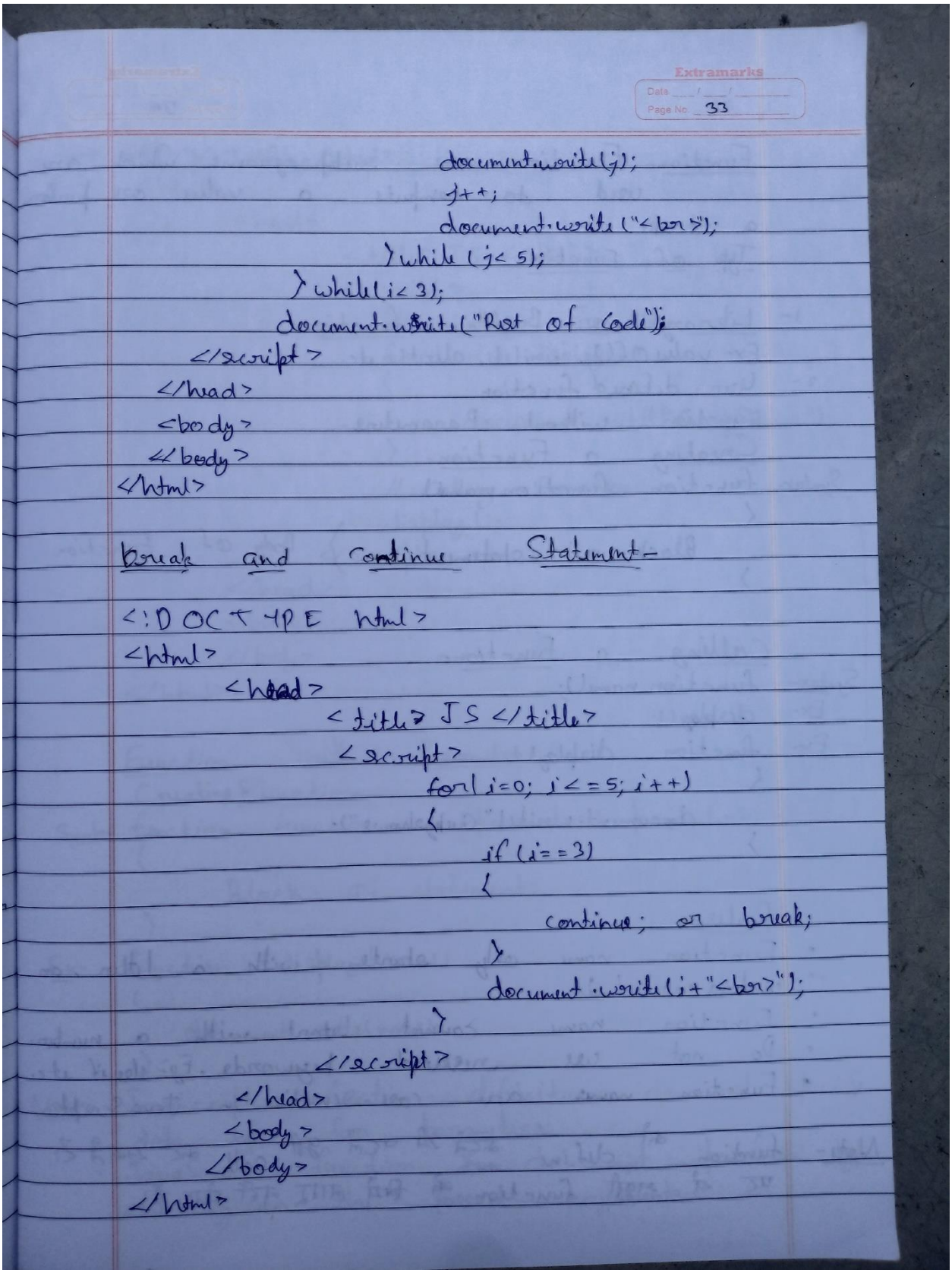


Nested do while Loop:- Do while inside do while loop.

initialization;

Syntax.
```
do
{
    statements;
    increment/decrement;
    initiallization;
    do
    {
        statements;
        increment/decrement;
    }while(test condition);
}while (test condition);
```

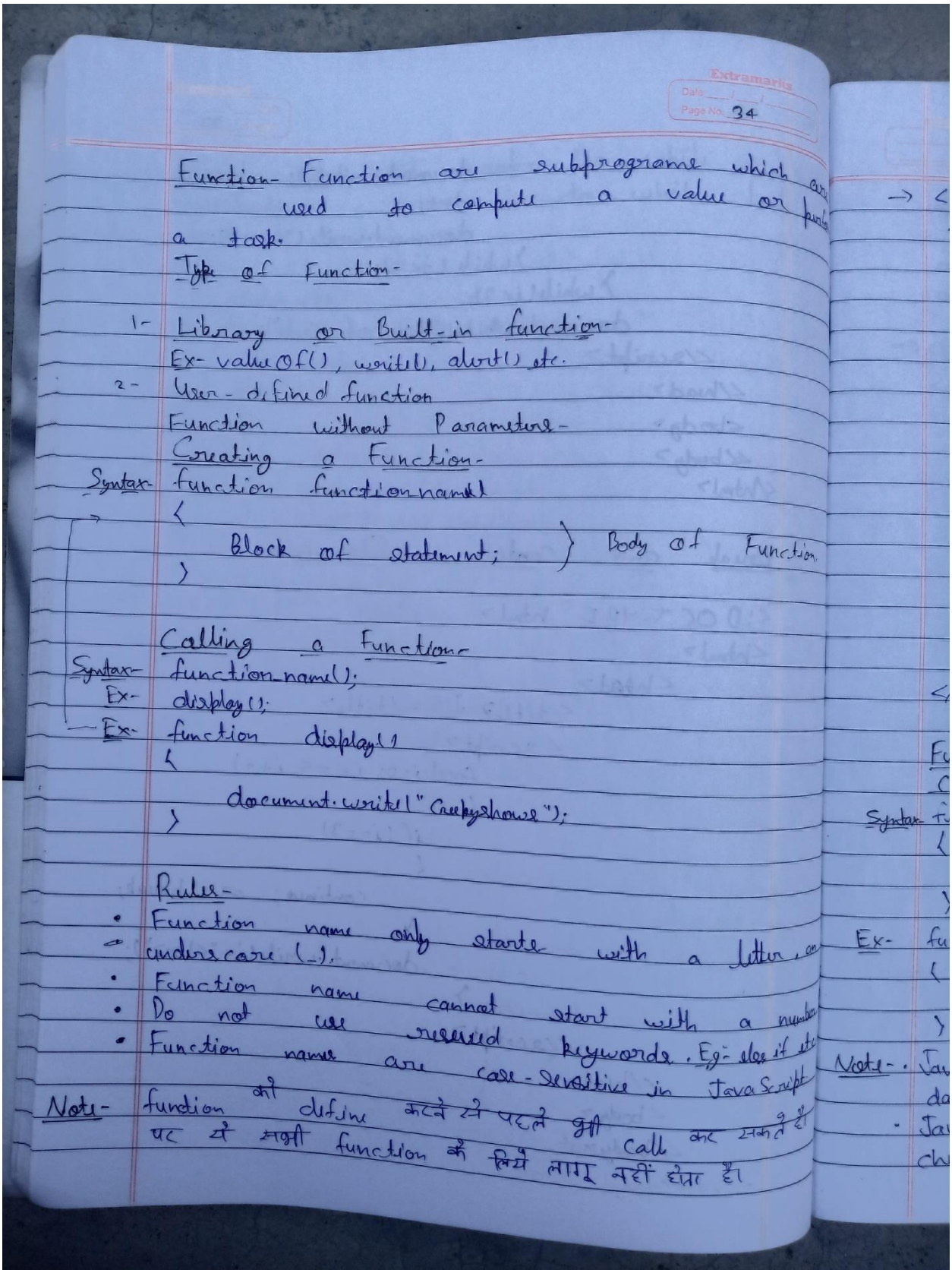
->
```
<!DOCTYPE html>
<html>
    <head>
        <title>JS</title>
        <script>
            var i=0;
            do
            {
                document.write("<strong>Outer
                    Loop</strong>")
                document.write(i);
                i++;
                document.write("<br>");
                var j=0;
                do
                {
                    document.write("Inner for
                        Loop");
```



```
                document.write(j);
                j++;
                document.write("<br>");
            }while(j<5);
        }while(i<3);
        document.write("Rest of Code");
    </script>
</head>
<body>
</body>
</html>
```

break and continue Statement-

```
<!DOCTYPE html>
<html>
    <head>
        <title> JS </title>
        <script>
            for(i=0; i<=5; i++)
            {
                if(i==3)
                {
                    continue; or break;
                }
                document.write(i+"<br>");
            }
        </script>
    </head>
    <body>
    </body>
</html>
```



Function- Function are subprograms which are used to compute a value or perform a task.

Type of Function-

1- Library or Built-in function-
Ex- value of(), write(), alert() etc.

2- User-defined function

Function without Parameters-

Creating a Function-

Syntax-
```
function functionname()
{
    Block of statement;
}
```
Body of Function

Calling a Function-

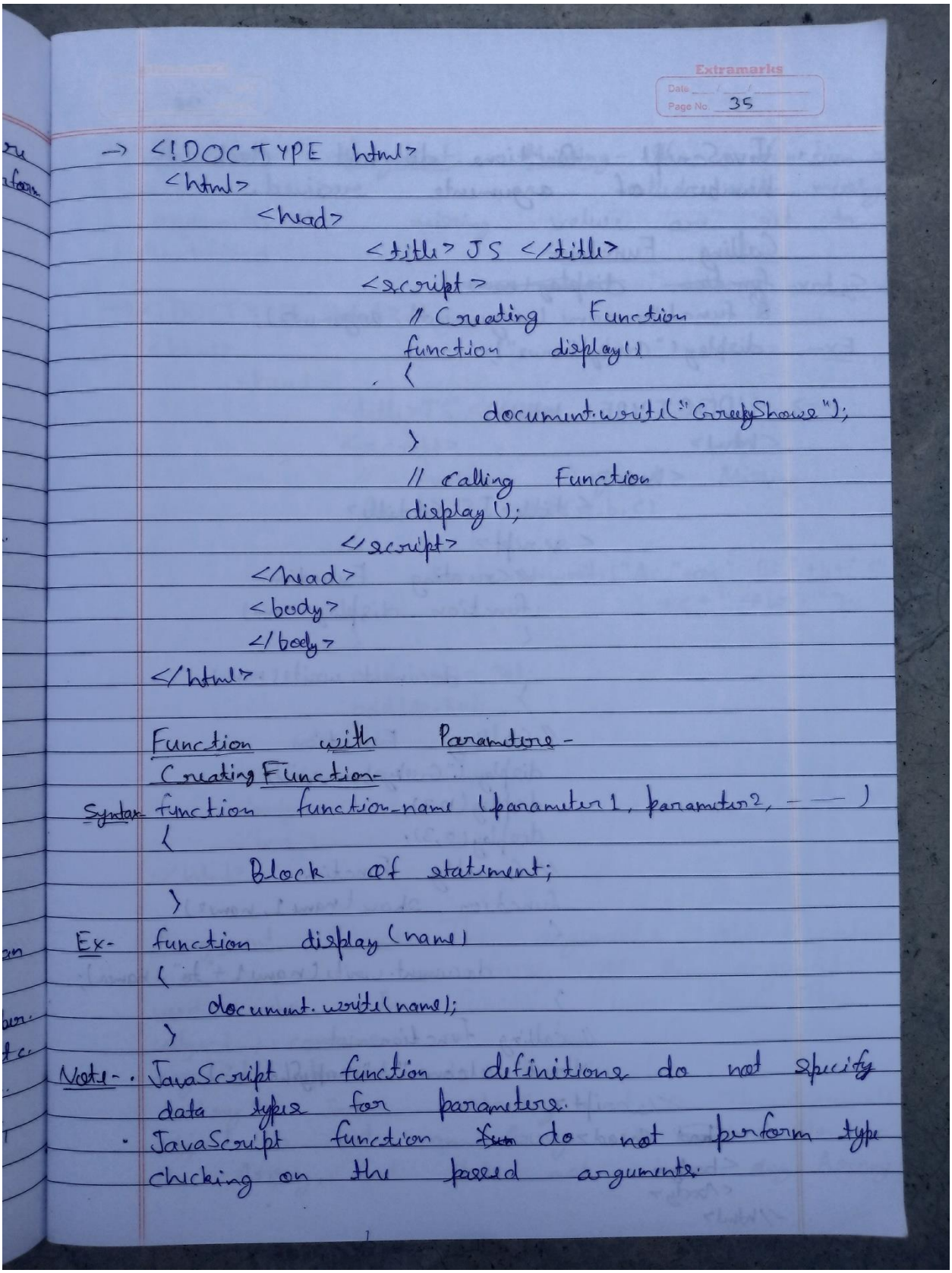
Syntax- function_name();

Ex- display();

Ex-
```
function display()
{
    document.write("Geekyshows");
}
```

Rules-
- Function name only starts with a letter, an underscore (_).
- Function name cannot start with a number
- Do not use reserved keywords. Eg- else if etc.
- Function name are case-sensitive in JavaScript

Note- function को define करने से पहले भी call कर सकते हैं पर ये सभी function के लिये लागू नहीं होता है।



Extramarks
Date
Page No. 35

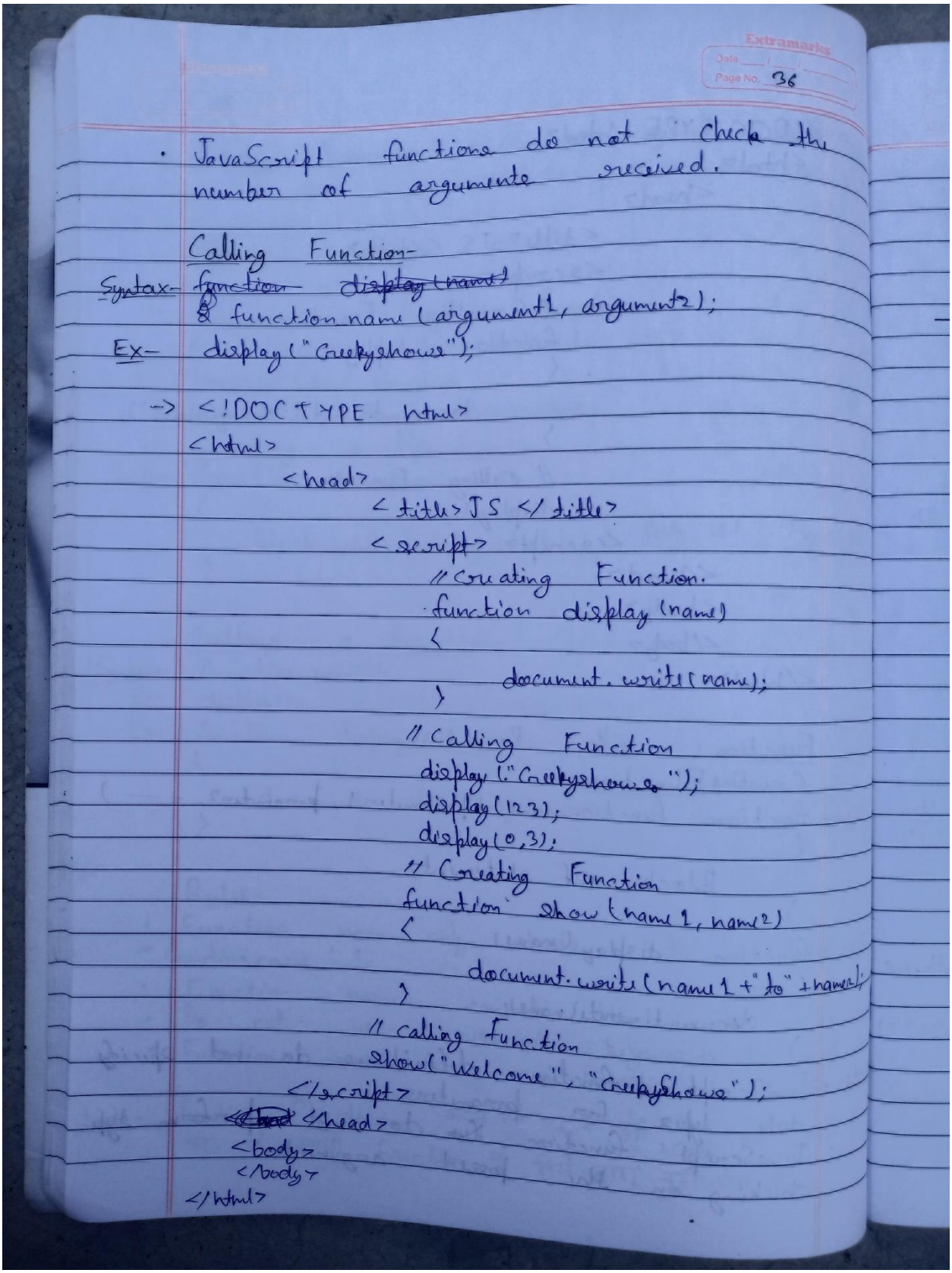
→ <!DOCTYPE html>
<html>
<head>
<title> JS </title>
<script>
// Creating Function
function display()
{
document.write("GeekyShows");
}
// calling Function
display();
</script>
</head>
<body>
</body>
</html>

Function with Parameters -
Creating Function -

Syntax- function function-name (parameter1, parameter2, ---)
{
Block of statement;
}

Ex- function display(name)
{
document.write(name);
}

Note- • JavaScript function definitions do not specify data types for parameters.
• JavaScript function do not perform type checking on the passed arguments.



- JavaScript functions do not check the number of arguments received.

Calling Function-

Syntax- ~~function display (name)~~
function name (argument1, argument2);

Ex- display ("Geekyshows");

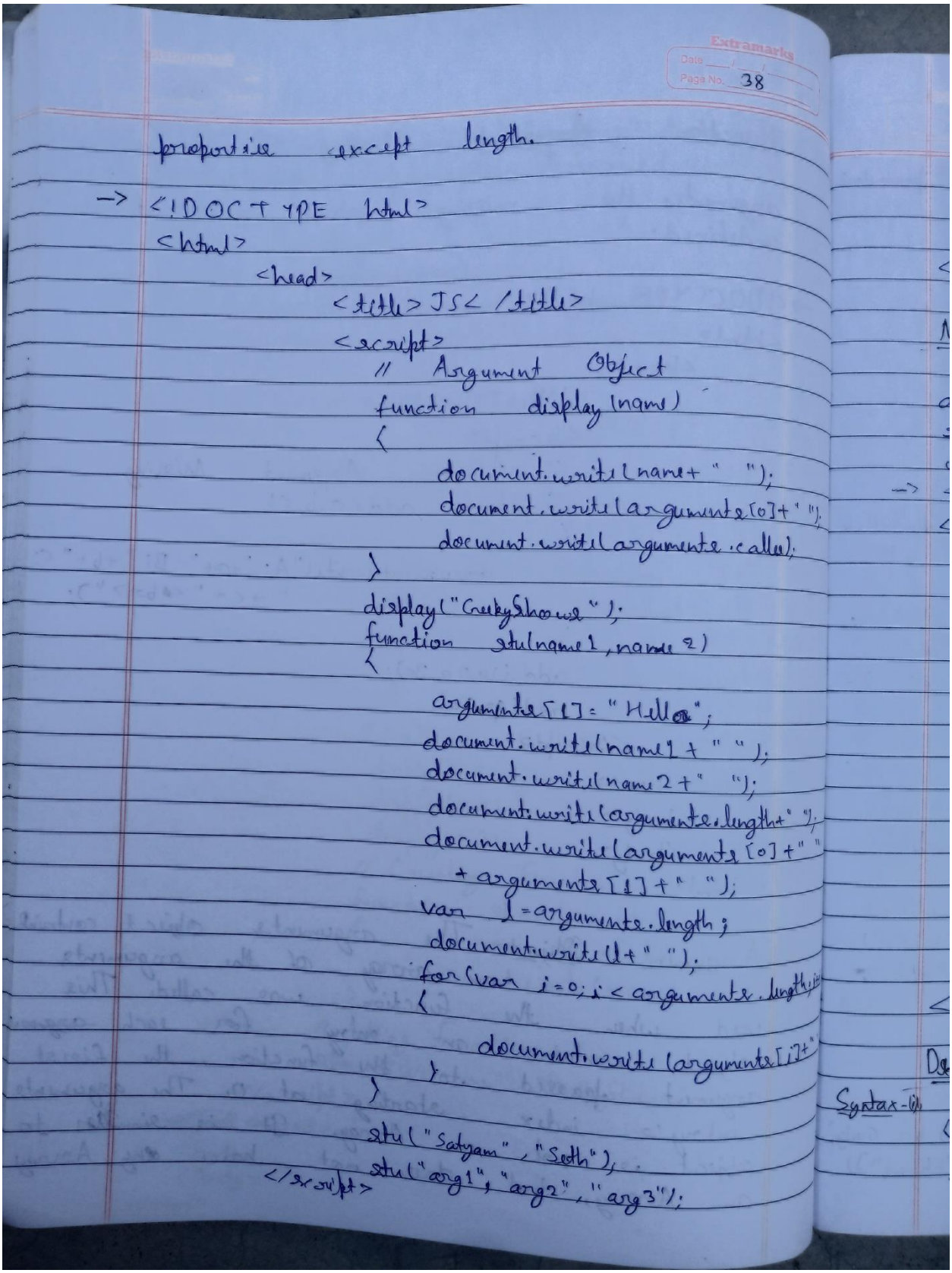
```
-> <!DOCTYPE html>
<html>
    <head>
        <title> JS </title>
        <script>
            // Creating Function.
            function display (name)
            {
                document.write(name);
            }
            // Calling Function
            display ("Geekyshows");
            display (123);
            display (0,3);
            // Creating Function
            function show (name1, name2)
            {
                document.write (name1 + "to" + name2);
            }
            // calling Function
            show ("Welcome", "Geekyshows");
        </script>
    </head>
    <body>
    </body>
</html>
```

Extramarks
Date
Page No. 37

Function Argument Missing - If a function is called with missing arguments, the missing values are set to undefined.

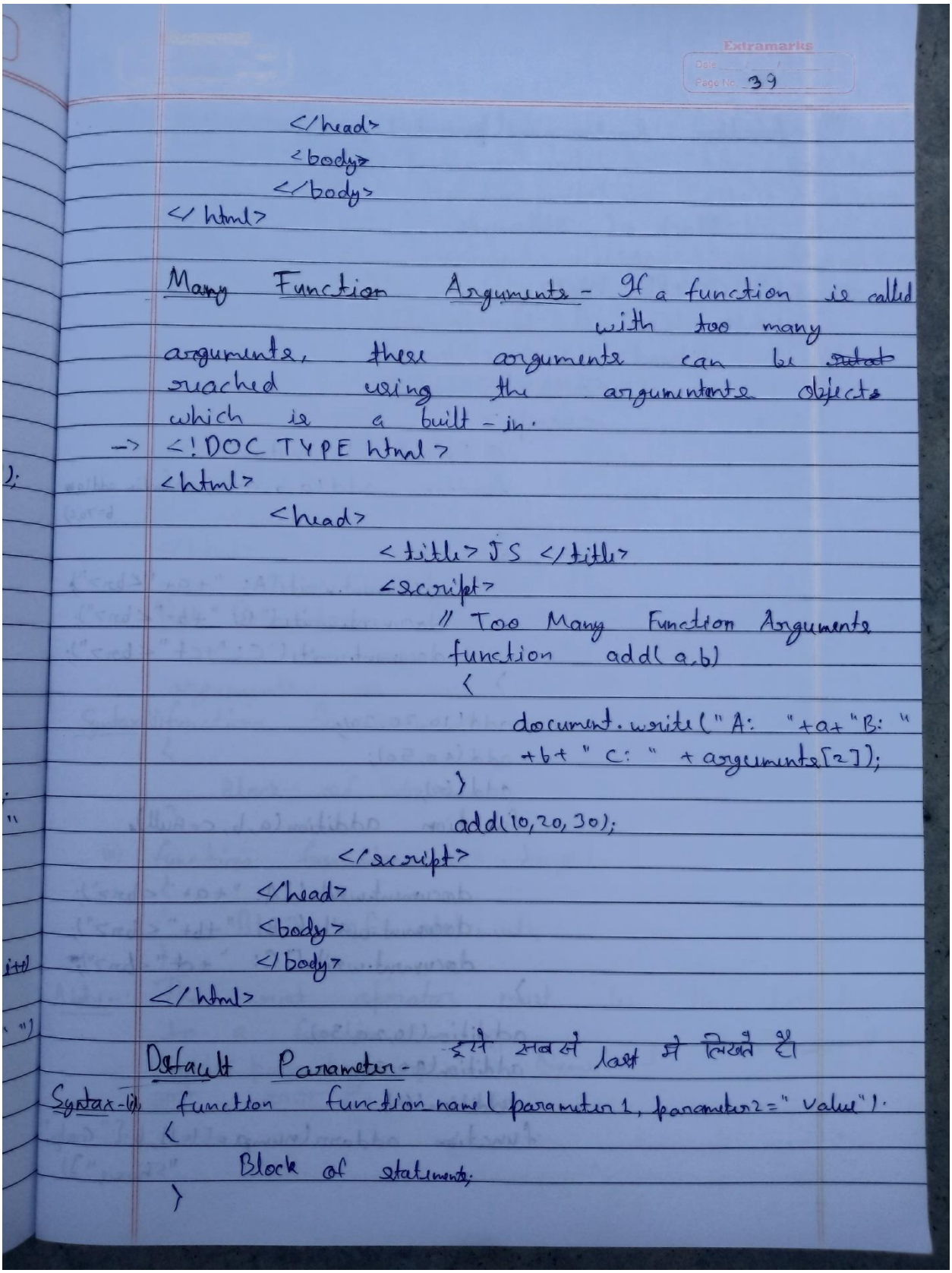
```
<!DOCTYPE html>
<html>
    <head>
        <title>JS</title>
        <script>
        // Function Argument Missing
        function add(a,b,c)
        {
            document.write("A: "+a+" B: "+b+" C: "+c+"<br>");
        }
        add(10,20,30);
        add(40,50);
        </script>
    </head>
    <body>
    </body>
</html>
```

Argument Object - The arguments object contains an array of the arguments used when the function was called. This object contains an entry for each argument passed to the function, the first entry's index starting at 0. The arguments object is not an Array. It is similar to an Array, but does not have any Array



properties except length.

```
-> <!DOCTYPE html>
<html>
    <head>
        <title> JS </title>
        <script>
            // Argument Object
            function display(name)
            {
                document.write(name+" ");
                document.write(arguments[0]+" ");
                document.write(arguments.callee);
            }
            display("GeekyShows");
            function stu(name1, name2)
            {
                arguments[1] = "Hello";
                document.write(name1 + " ");
                document.write(name2 + " ");
                document.write(arguments.length+" ");
                document.write(arguments[0]+" "
                + arguments[1] + " ");
                var l = arguments.length;
                document.write(l+" ");
                for(var i=0; i < arguments.length;i++)
                {
                    document.write(arguments[i]+" ");
                }
            }
            stu("Satyam", "Seth");
            stu("arg1", "arg2", "arg3");
        </script>
```



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

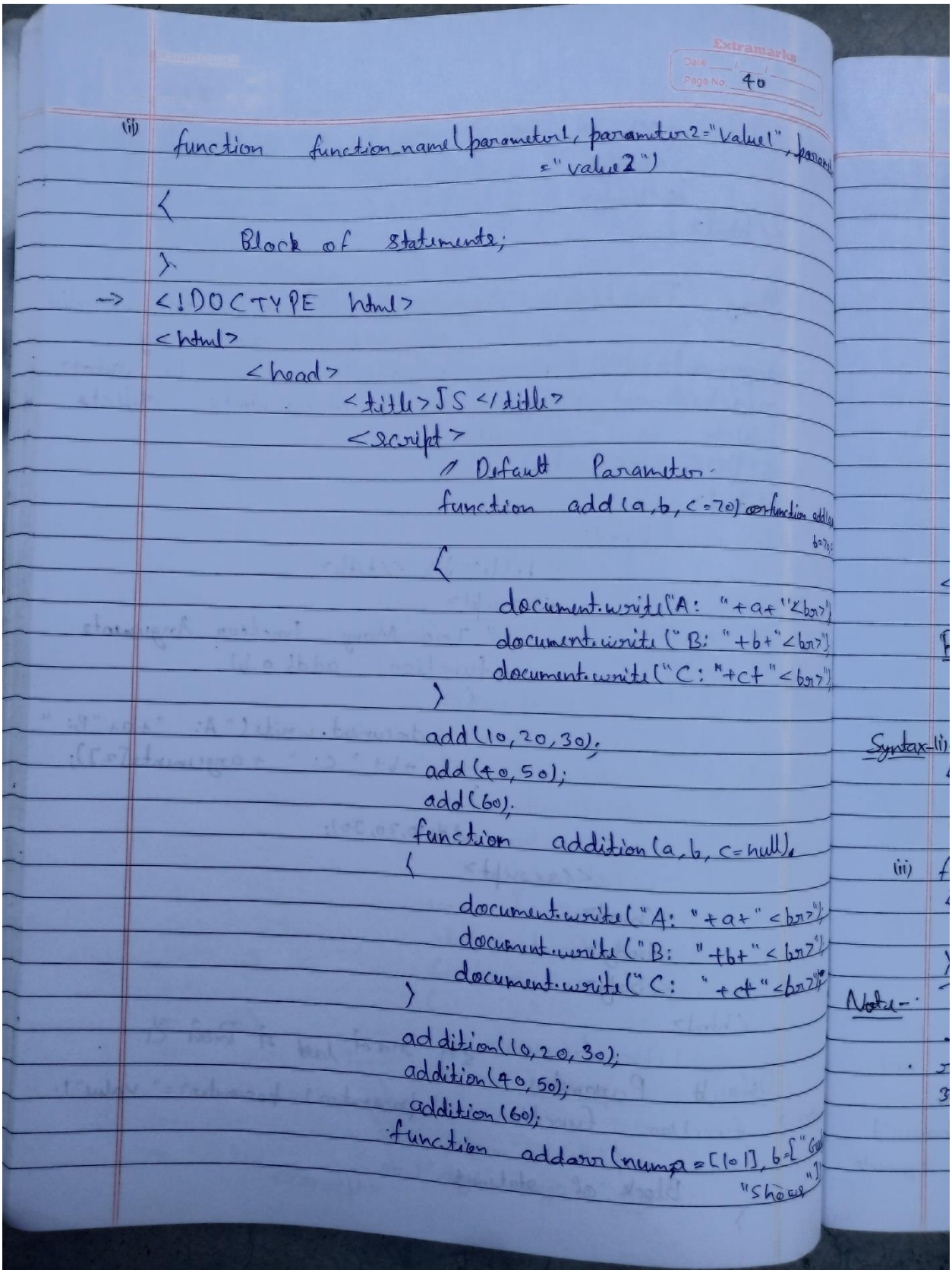
Many Function Arguments - If a function is called with too many arguments, these arguments can be ~~stat~~ reached using the argumentsobjects which is a built-in.

```
<!DOCTYPE html>
<html>
    <head>
        <title> JS </title>
        <script>
            // Too Many Function Arguments
            function add(a,b)
            {
                document.write("A: "+a+"B: "
                +b+" C: " +arguments[2]);
            }
            add(10,20,30);
        </script>
    </head>
    <body>
    </body>
</html>
```

Default Parameter - इसे सबसे last में लिखते है

Syntax-(i)

```
function functionname(parameter1, parameter2="value").
{
    Block of statements;
}
```



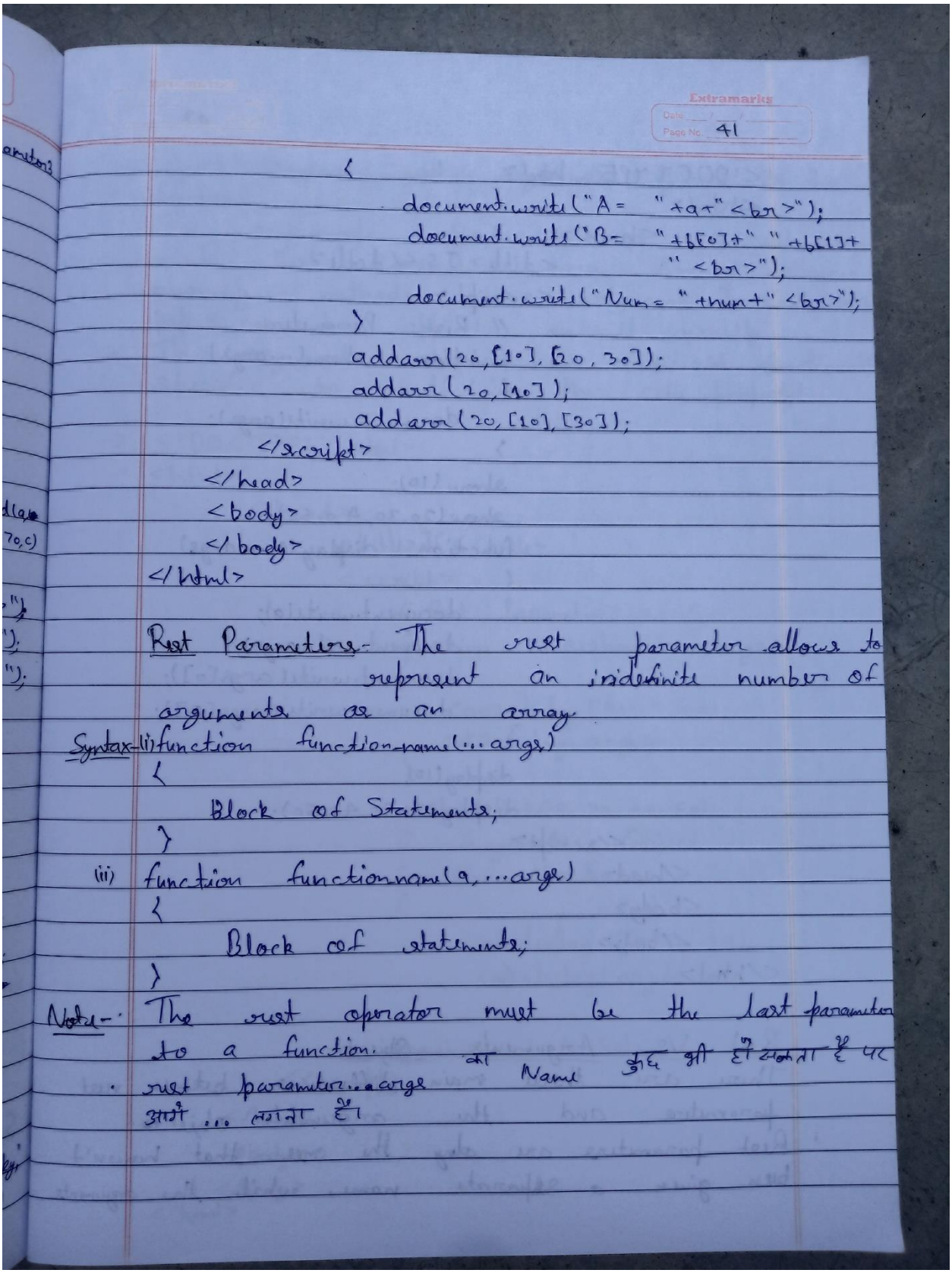
(i) function function_name(parameter1, parameter2="value1", parameter3="value2")
{
    Block of statements;
}

-> <!DOCTYPE html>
<html>
<head>
<title>JS</title>
<script>
// Default Parameter:
function add(a,b,c=70) // function addition(a,b,c=70)
{
document.write("A: "+a+"
");
document.write("B: "+b+"
");
document.write("C: "+c+"
");
}
add(10,20,30);
add(40,50);
add(60);
function addition(a,b,c=null)
{
document.write("A: "+a+"
");
document.write("B: "+b+"
");
document.write("C: "+c+"
");
}
addition(10,20,30);
addition(40,50);
addition(60);
function addarr(numa=[101], b=["Geeky "Shows"]

Syntax-(i)

(ii) f

Note:-



```
        {
            document.write("A= "+a+"<br>");
            document.write("B= "+b[0]+" "+b[1]+
                           " <br>");
            document.write("Num= "+num+"<br>");
        }
        addarr(20,[10],[20,30]);
        addarr(20,[10]);
        addarr(20,[10],[30]);
    </script>
</head>
<body>
</body>
</html>
```

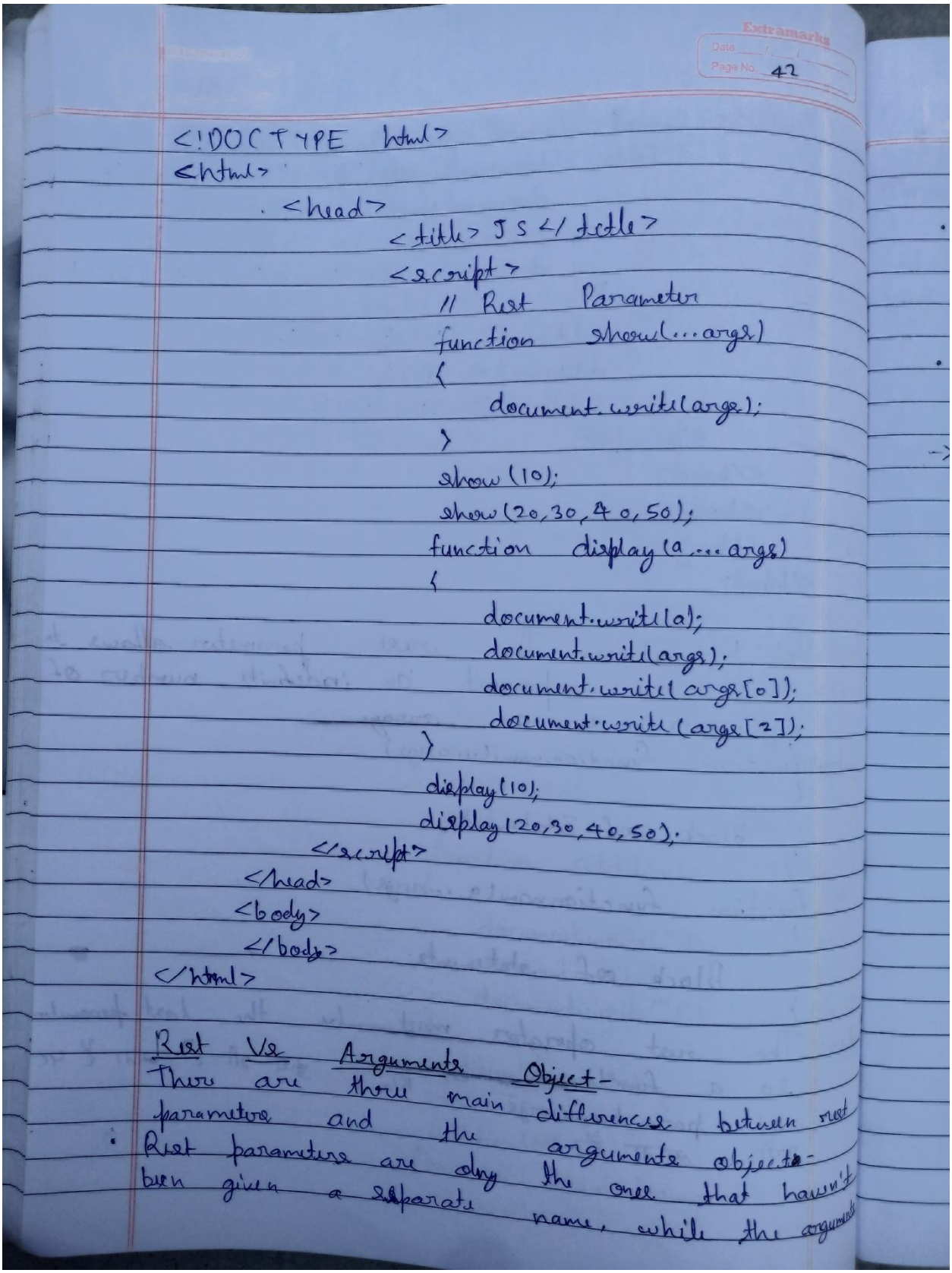
Rest Parameters- The rest parameter allows to represent an indefinite number of arguments as an array.

Syntax- (i) function function-name(...args)
{
    Block of Statements;
}

(ii) function functionname(a,...args)
{
    Block of statements;
}

Note- The rest operator must be the last paramter to a function.

- rest paramter ...args का Name कुछ भी हो सकता है पर आगे ... लगना है।

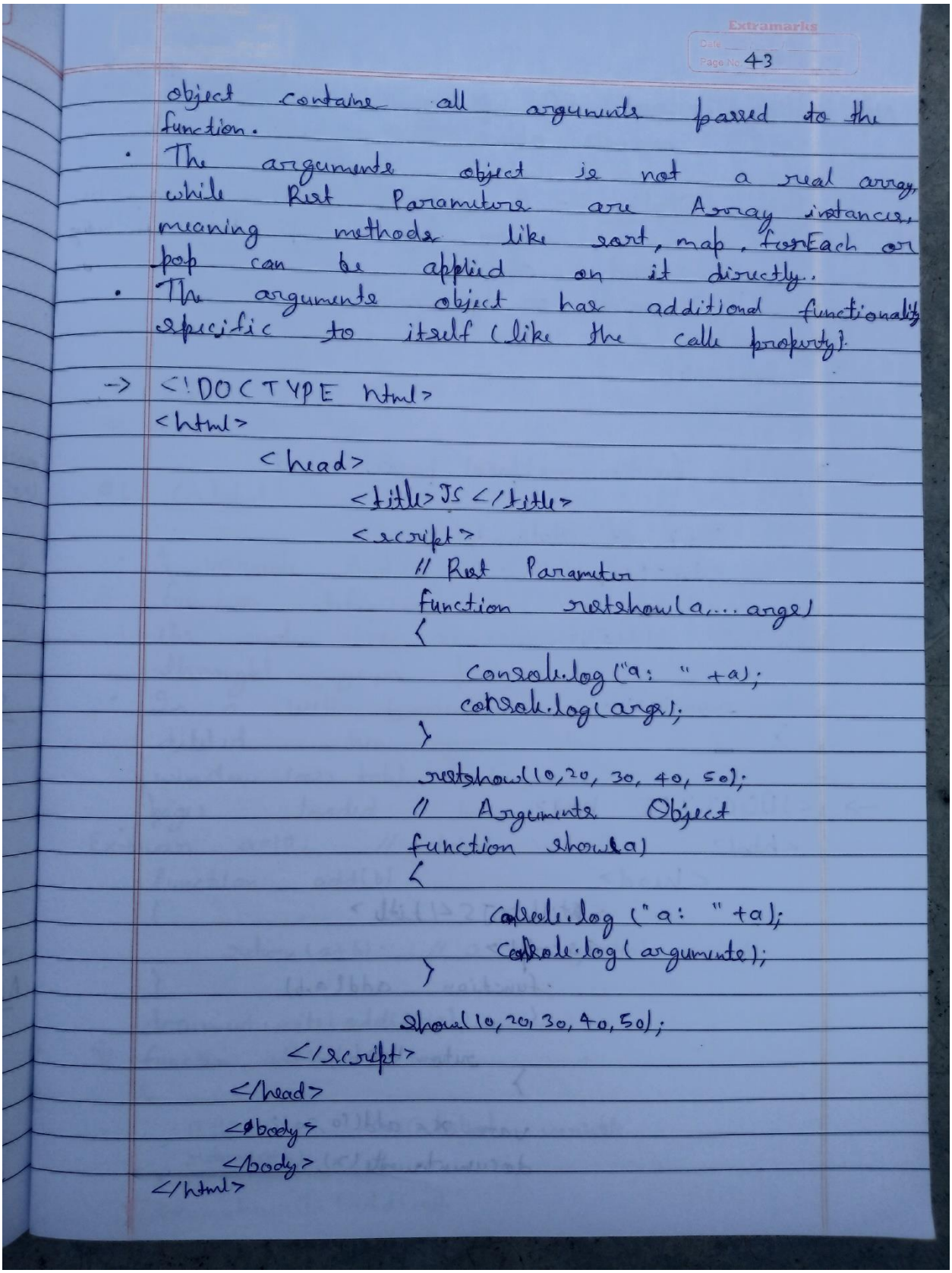


```
<!DOCTYPE html>
<html>
    <head>
        <title> JS </title>
        <script>
            // Rest Parameter
            function show(...args)
            {
                document.write(args);
            }
            show(10);
            show(20,30,40,50);
            function display(a,...args)
            {
                document.write(a);
                document.write(args);
                document.write(args[0]);
                document.write(args[2]);
            }
            display(10);
            display(20,30,40,50);
        </script>
    </head>
    <body>
    </body>
</html>
```

Rest Vs Arguments Object -

There are three main differences between rest parameters and the arguments objects -

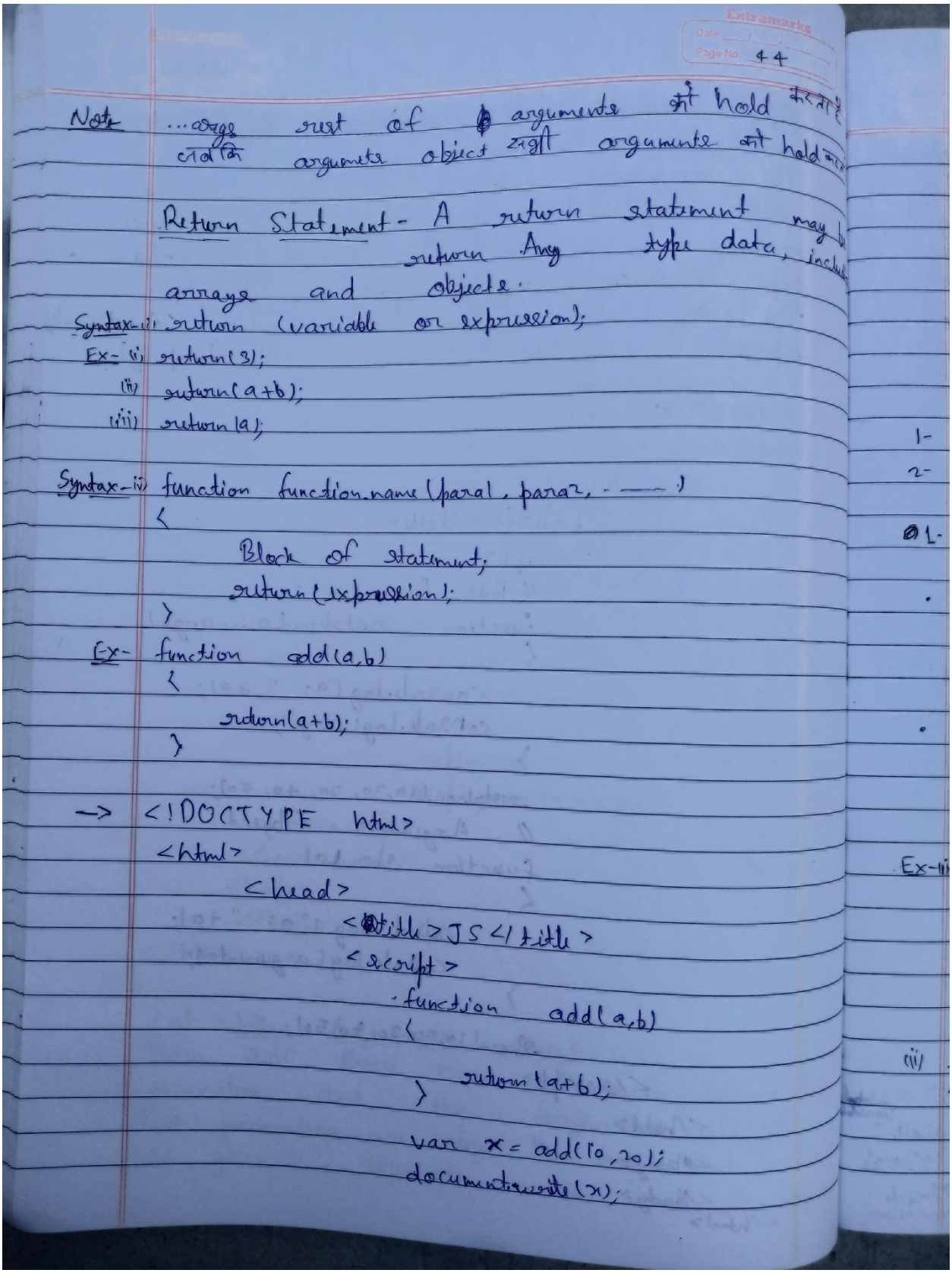
- Rest parameters are only the ones that haven't been given a separate name, while the arguments



object contains all arguments passed to the function.

- The arguments object is not a real array, while Rest Parameters are Array instances, meaning methods like sort, map, forEach or pop can be applied on it directly.
- The arguments object has additional functionality specific to itself (like the calle property).

-> <!DOCTYPE html>
<html>
<head>
<title>JS</title>
<script>
// Rest Parameter
function restshow(a,...args)
{
console.log("a: "+a);
console.log(args);
}
restshow(10,20,30,40,50);
// Arguments Object
function show(a)
{
console.log("a: "+a);
console.log(arguments);
}
show(10,20,30,40,50);
</script>
</head>
<body>
</body>
</html>



Note ...args rest of arguments को hold करता है
वाला arguments object सभी arguments को hold करता

Return Statement - A return statement may be return Any type data, including arrays and objects.

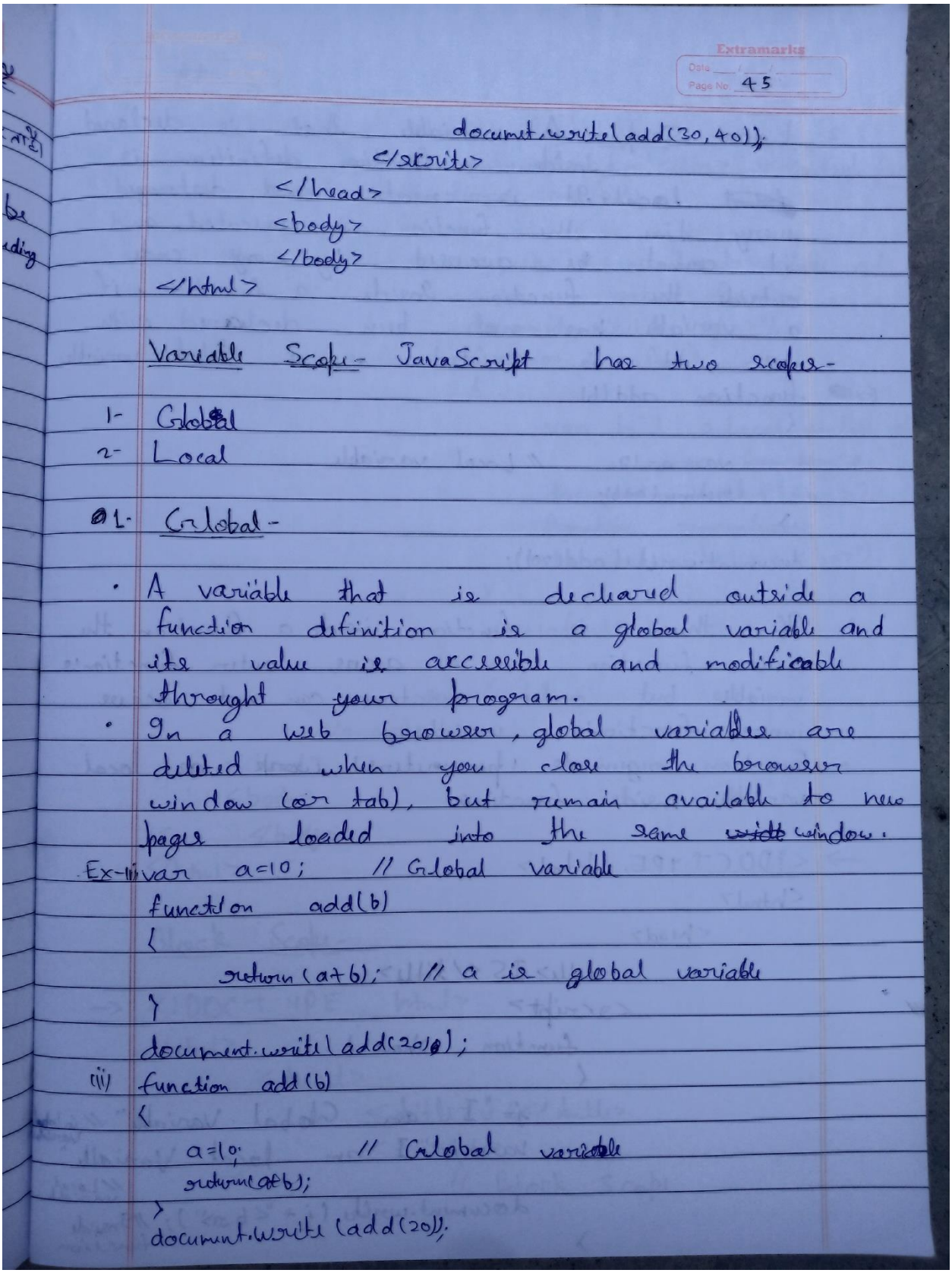
Syntax-i) return (variable or expression);

Ex- i) return(3);
ii) return(a+b);
iii) return(a);

Syntax-ii) function function_name(para1, para2, ----)
{
Block of statement;
return(expression);
}

Ex- function add(a,b)
{
return(a+b);
}

```
--> <!DOCTYPE html>
<html>
    <head>
        <title>JS</title>
        <script>
            function add(a,b)
            {
                return(a+b);
            }
            var x = add(10,20);
            document.write(x);
```



```
            document.write(add(30,40));
        </script>
    </head>
    <body>
    </body>
</html>
```

Variable Scope - JavaScript has two scopes -

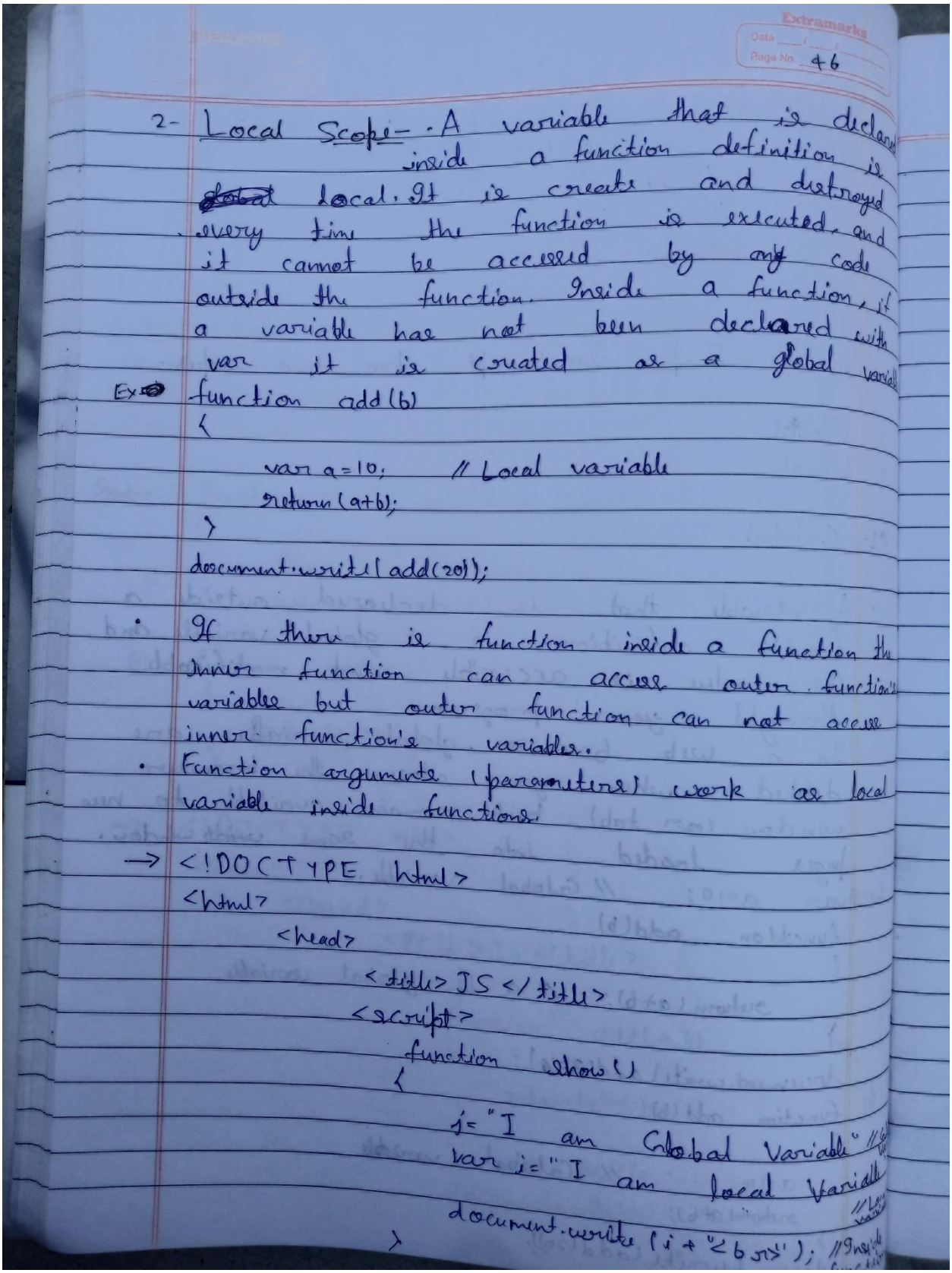
1- Global
2- Local

1. Global -

- A variable that is declared outside a function definition is a global variable and its value is accessible and modifiable throughout your program.
- In a web browser, global variables are deleted when you close the browser window (or tab), but remain available to new pages loaded into the same ~~with~~ window.

Ex-i)
```
var a=10;   // Global variable
function add(b)
{
    return(a+b);   // a is global variable
}
document.write(add(20));
```

(ii)
```
function add(b)
{
    a=10;   // Global variable
    return(a+b);
}
document.write(add(20));
```

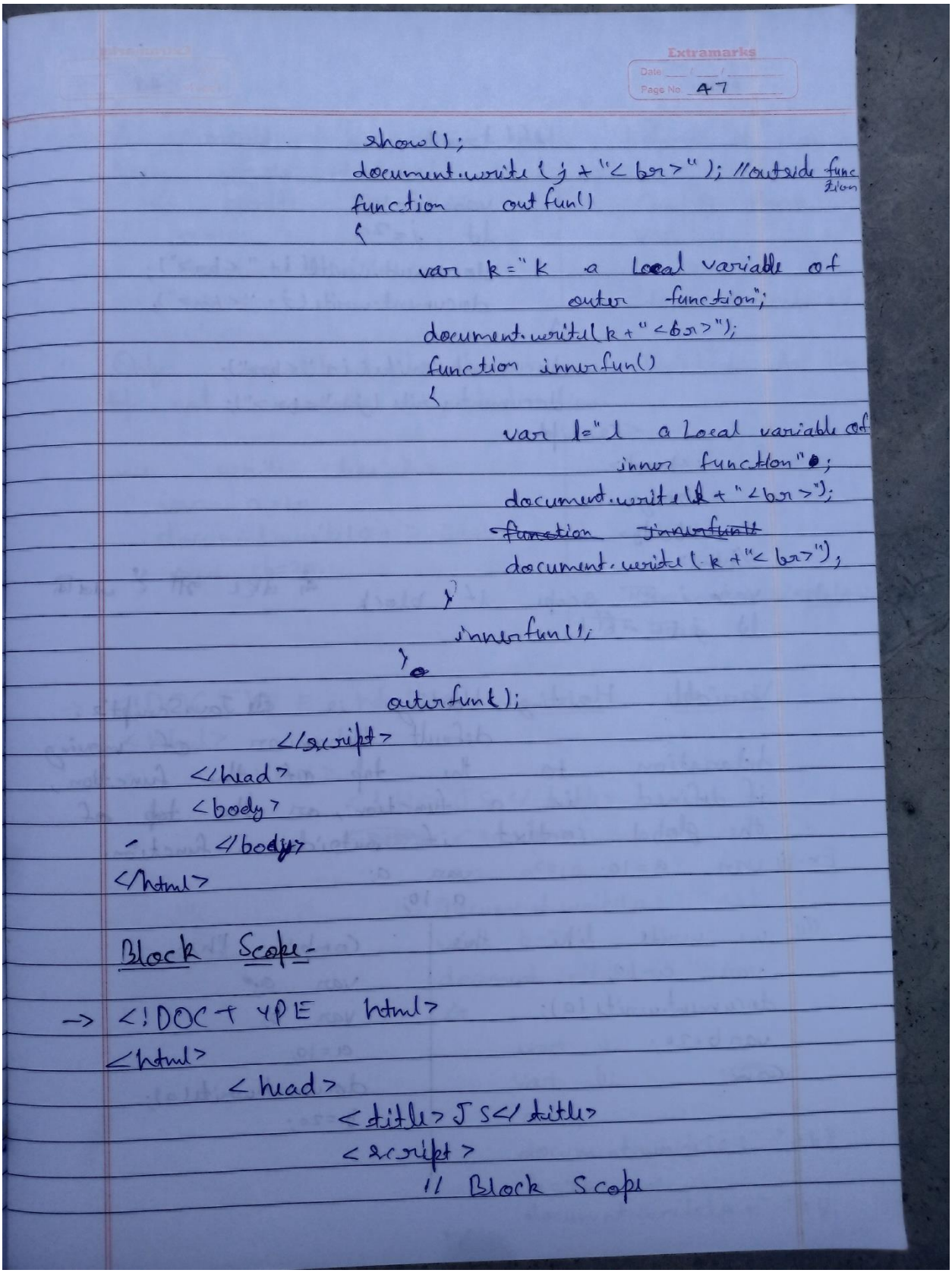


2- Local Scope - A variable that is declared inside a function definition is ~~global~~ local. It is created and destroyed every time the function is executed, and it cannot be accessed by any code outside the function. Inside a function, if a variable has not been declared with var it is created as a global variable.

Ex
```
function add(b)
{
    var a=10;    // Local variable
    return(a+b);
}
document.write(add(20));
```

- If there is function inside a function the inner function can access outer function's variables but outer function can not access inner function's variables.
- Function arguments (parameters) work as local variable inside functions.

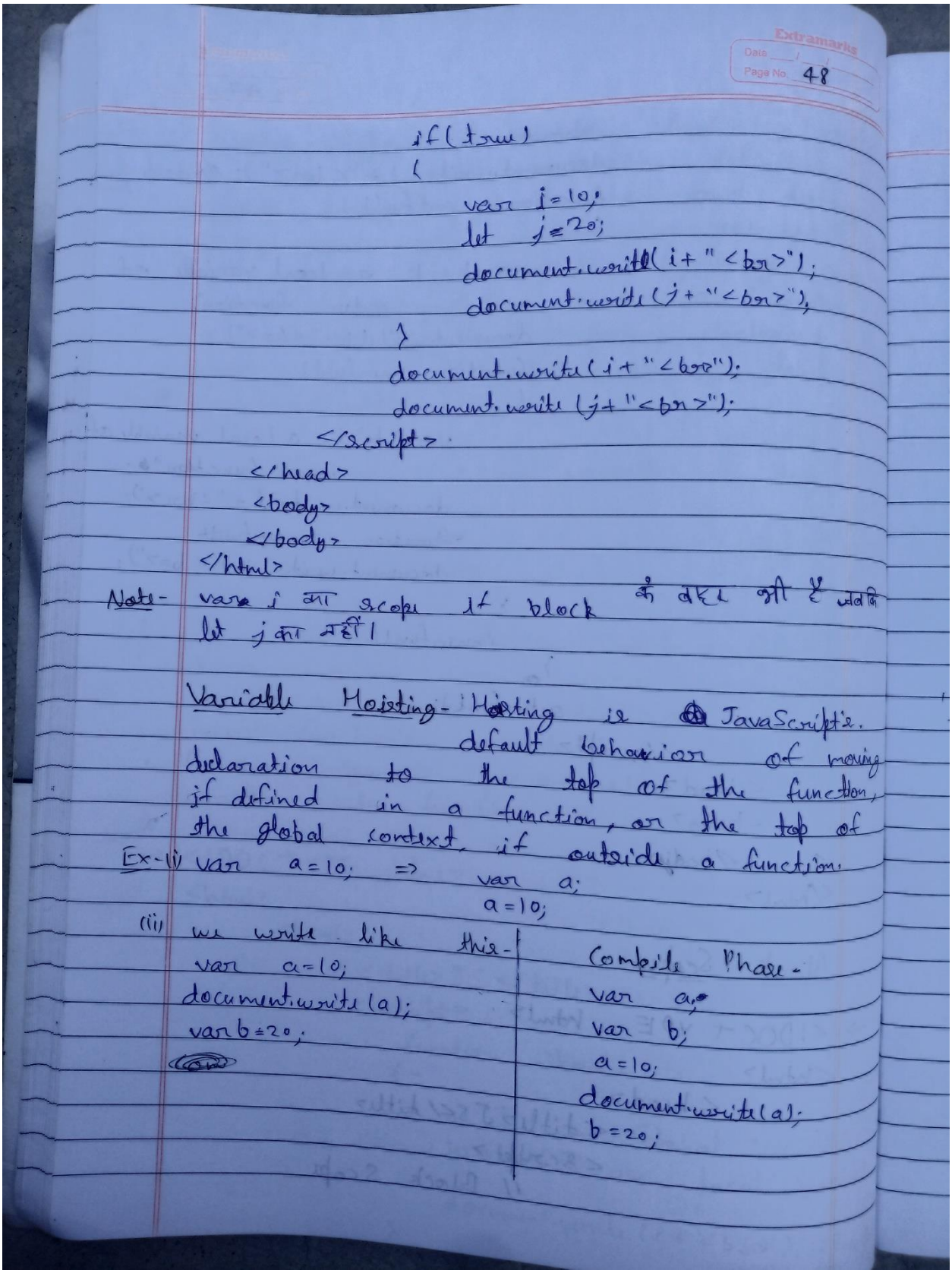
→
```
<!DOCTYPE html>
<html>
    <head>
        <title> JS </title>
        <script>
            function show()
            {
                i= "I am Global Variable" //Global
                var i= "I am local Variable"
                //Local variable
                document.write(i + "<br>"); //Inside
            }
```



```
show();
document.write(j + "<br>"); //outside function
function outfun()
{
    var k = "k a Local variable of
             outer function";
    document.write(k + "<br>");
    function innerfun()
    {
        var l = "l a Local variable of
                 inner function";
        document.write(k + "<br>");
        function innerfun1
        document.write(k + "<br>");
    }
    innerfun();
}
outerfun();
</script>
</head>
<body>
</body>
</html>
```

Block Scope-

```
-> <!DOCTYPE html>
<html>
    <head>
        <title> JS </title>
        <script>
            // Block Scope
```



```
if(true)
{
    var i=10;
    let j=20;
    document.write(i+"<br>");
    document.write(j+"<br>");
}
document.write(i+"<br>");
document.write(j+"<br>");
</script>
</head>
<body>
</body>
</html>
```

Note- var i का scope if block के बाहर भी है जबकि let j का नहीं।

Variable Hoisting- Hoisting is JavaScript's default behaviour of moving declaration to the top of the function, if defined in a function, or the top of the global context, if outside a function.

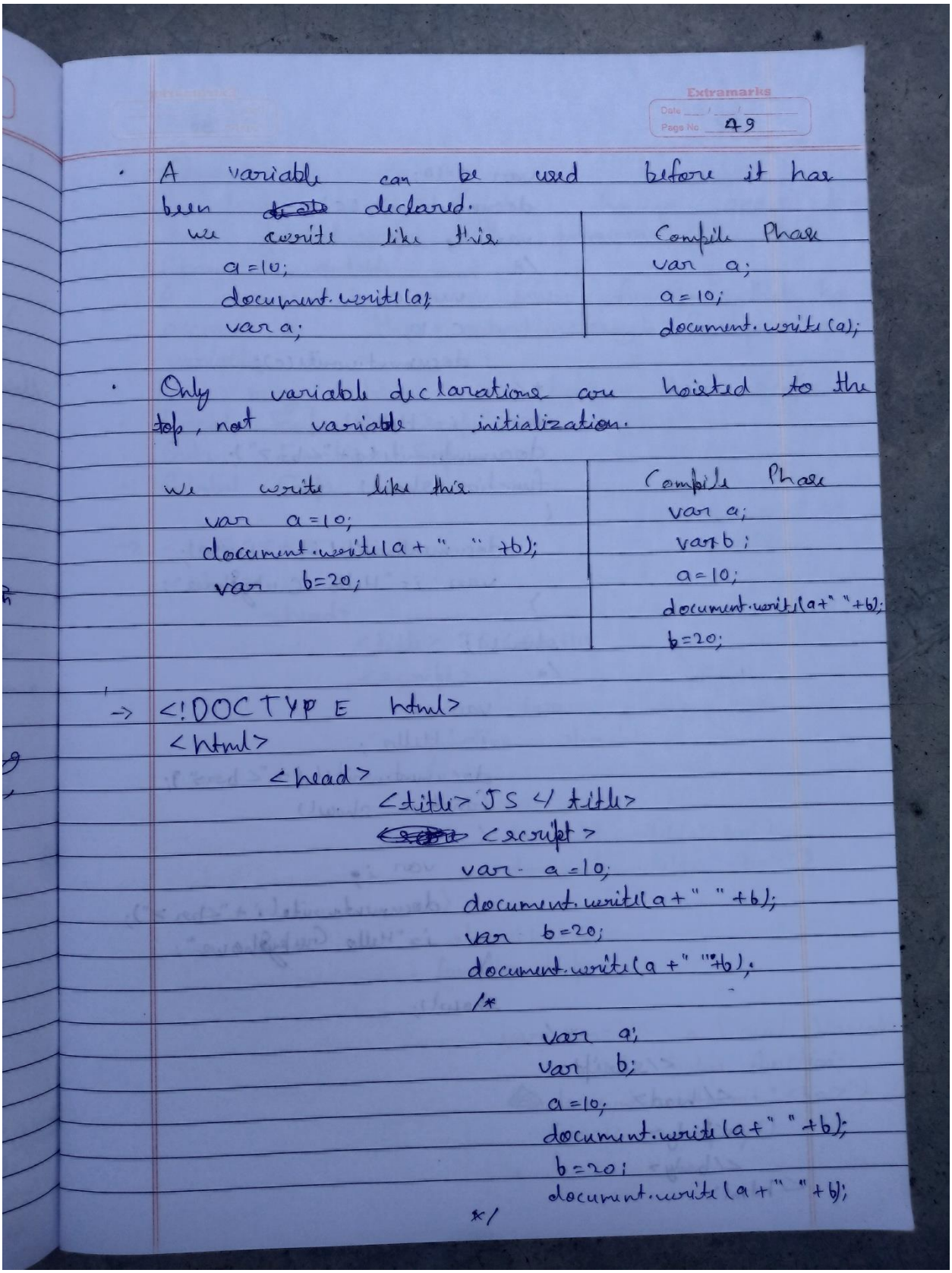
Ex-(i) var a=10; => var a;
a=10;

(ii) We write like this-

```
var a=10;
document.write(a);
var b=20;
```

Compile Phase-

```
var a;
var b;
a=10;
document.write(a);
b=20;
```



Extramarks
Page No. 49

- A variable can be used before it has been declared.

we write like this	Compile Phase

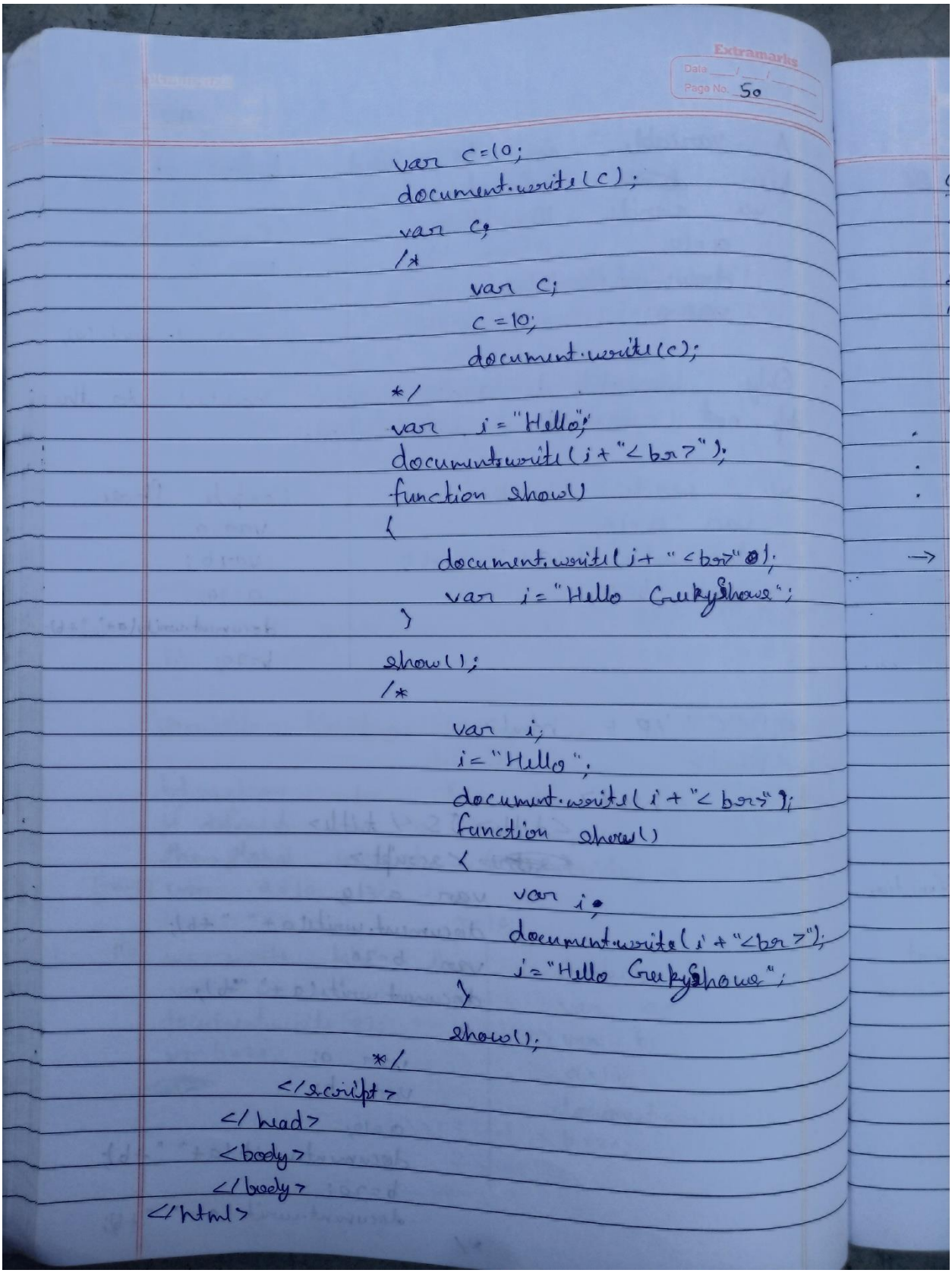
a=10;
document.write(a);
var a;
	var a;
	a=10;
	document.write(a);

- Only variable declarations are hoisted to the top, not variable initialization.

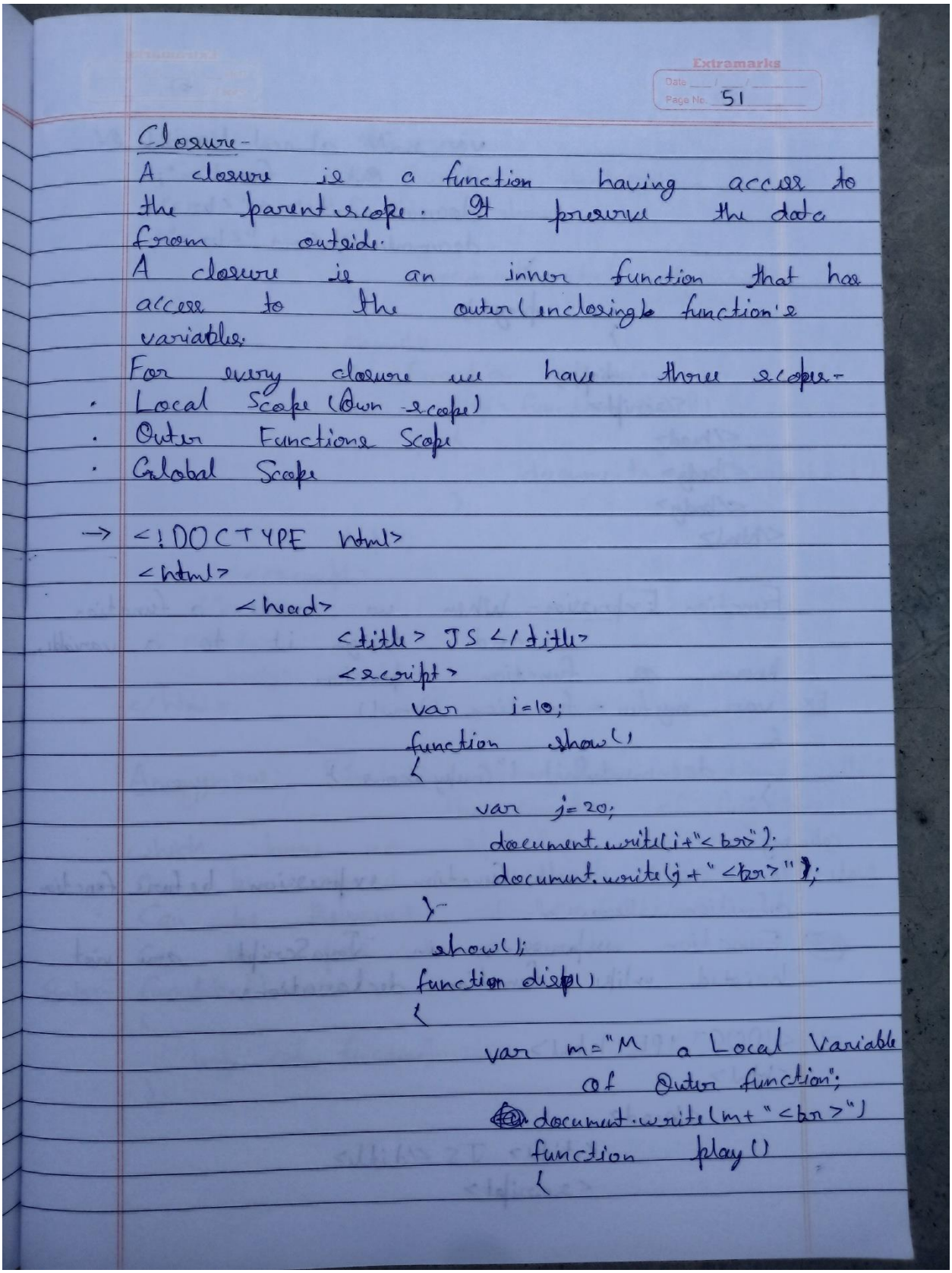
We write like this	Compile Phase

var a=10;
document.write(a+" "+b);
var b=20;
	var a;
	var b;
	a=10;
	document.write(a+" "+b);
	b=20;

->
```
<!DOCTYPE html>
<html>
    <head>
        <title> JS </title>
        <script>
            var a=10;
            document.write(a+" "+b);
            var b=20;
            document.write(a+" "+b);
            /*
                var a;
                var b;
                a=10;
                document.write(a+" "+b);
                b=20;
                document.write(a+" "+b);
            */
```



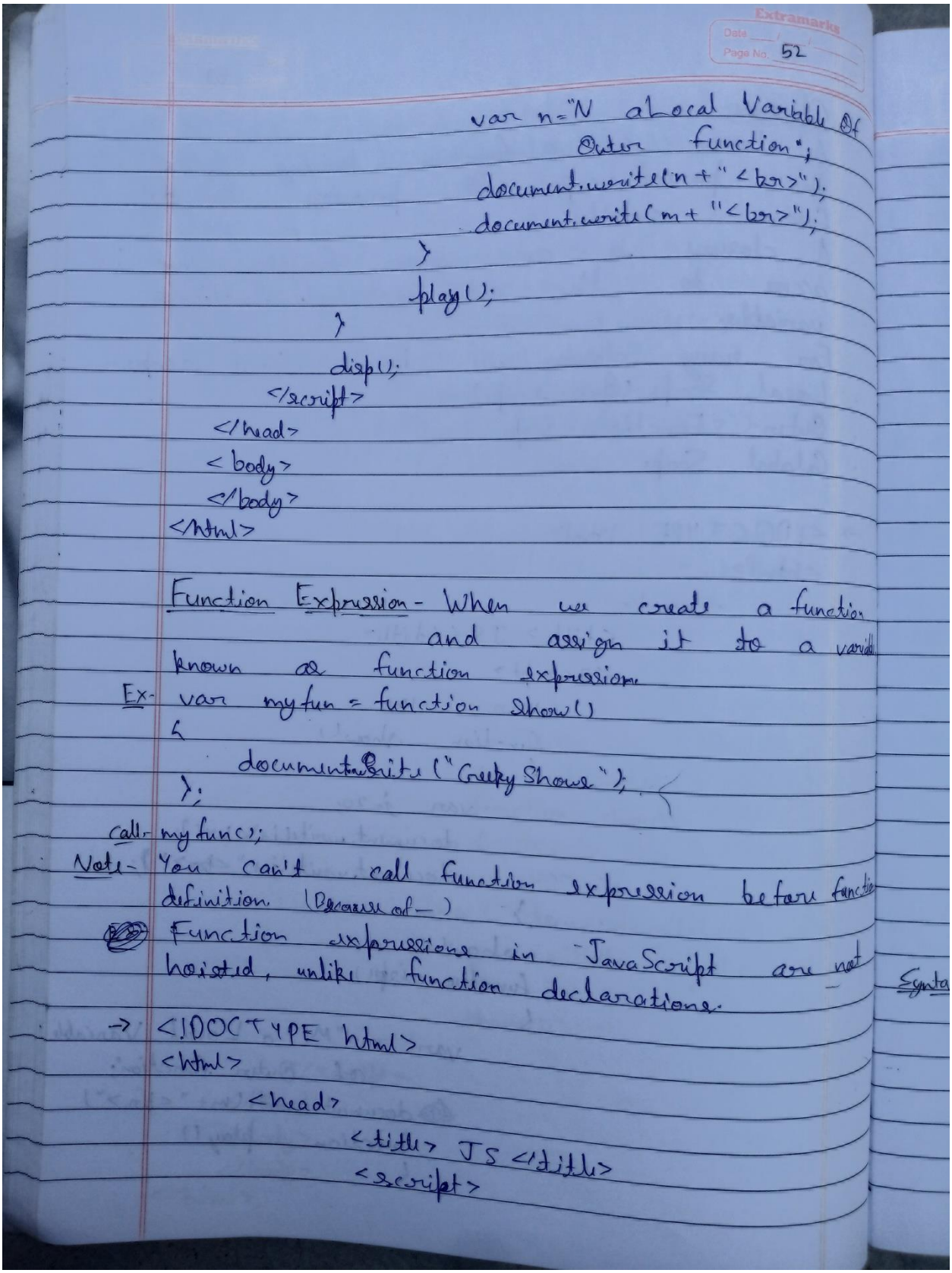
```
var c=10;
document.write(c);
var c;
/*
    var c;
    c=10;
    document.write(c);
*/
var i="Hello";
document.write(i+"<br>");
function show()
{
    document.write(i+"<br>");
    var i="Hello Geekyshows";
}
show();
/*
    var i;
    i="Hello";
    document.write(i+"<br>");
    function show()
    {
        var i;
        document.write(i+"<br>");
        i="Hello Geekyshows";
    }
    show();
*/
</script>
</head>
<body>
</body>
</html>
```



Extramarks
Date / /
Page No. 51

Closure-
A closure is a function having access to the parent scope. It preserve the data from outside.
A closure is an inner function that has access to the outer (enclosing) function's variables.
For every closure we have three scopes-
- Local Scope (Own scope)
- Outer Functions Scope
- Global Scope

```
<!DOCTYPE html>
<html>
    <head>
        <title> JS </title>
        <script>
            var i=10;
            function show()
            {
                var j=20;
                document.write(i+"<br>");
                document.write(j+"<br>");
            }
            show();
            function disp()
            {
                var m="M a Local Variable
                    of Outer function";
                document.write(m+"<br>")
                function play()
                {
```



```
                var n="N a Local Variable Of
                        Outer function";
                document.write(n+"<br>");
                document.write(m+"<br>");
            }
            play();
        }
        disp();
    </script>
</head>
<body>
</body>
</html>
```

Function Expression - When we create a function and assign it to a variable known as function expression.

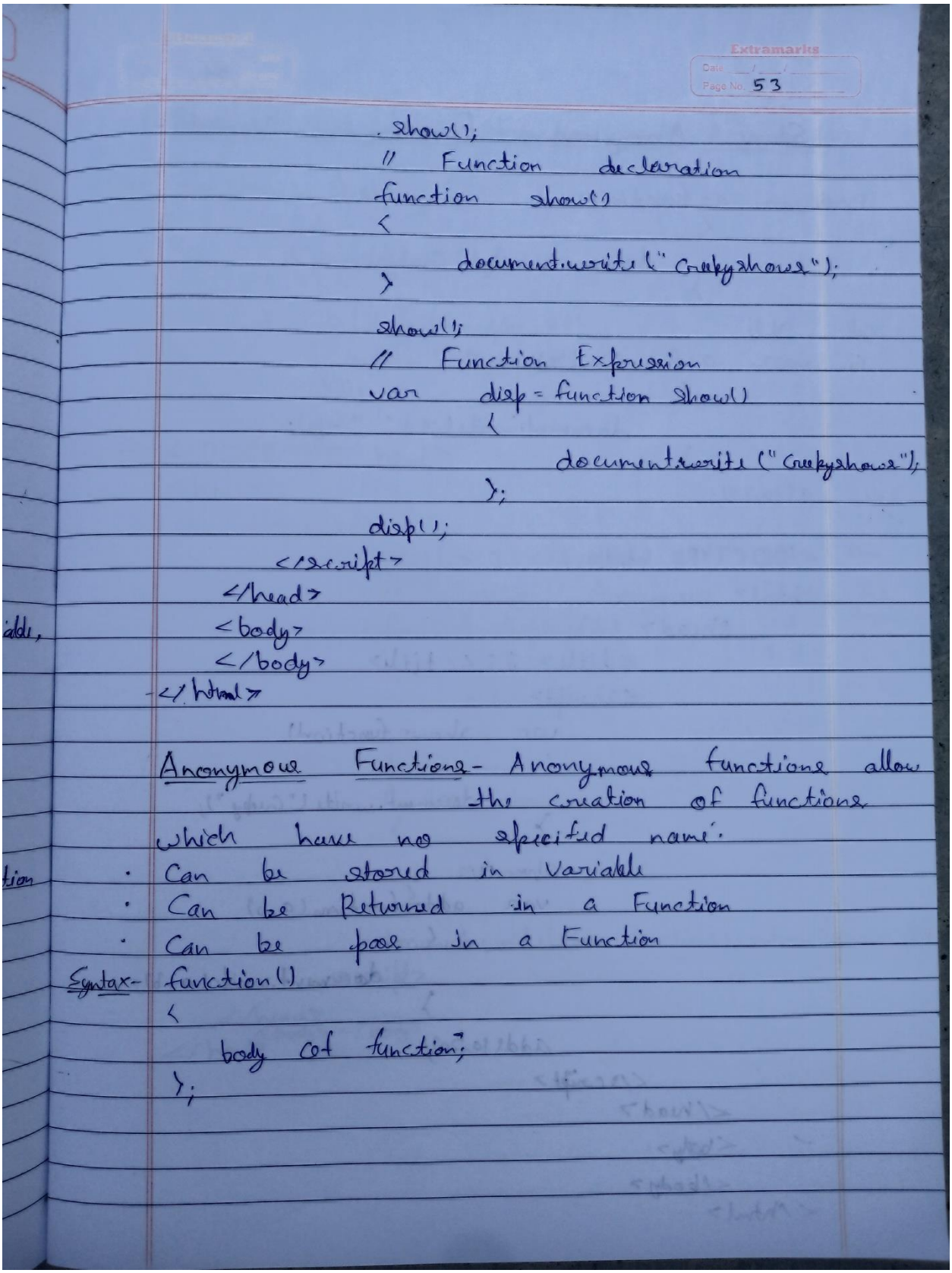
Ex- var myfun = function show()
{
    document.write("Geeky Shows");
};

call- myfunc();

Note- You can't call function expression before function definition. (Because of -)

Function expressions in JavaScript are not hoisted, unlike function declarations.

```
→ <!DOCTYPE html>
<html>
    <head>
        <title> JS </title>
        <script>
```



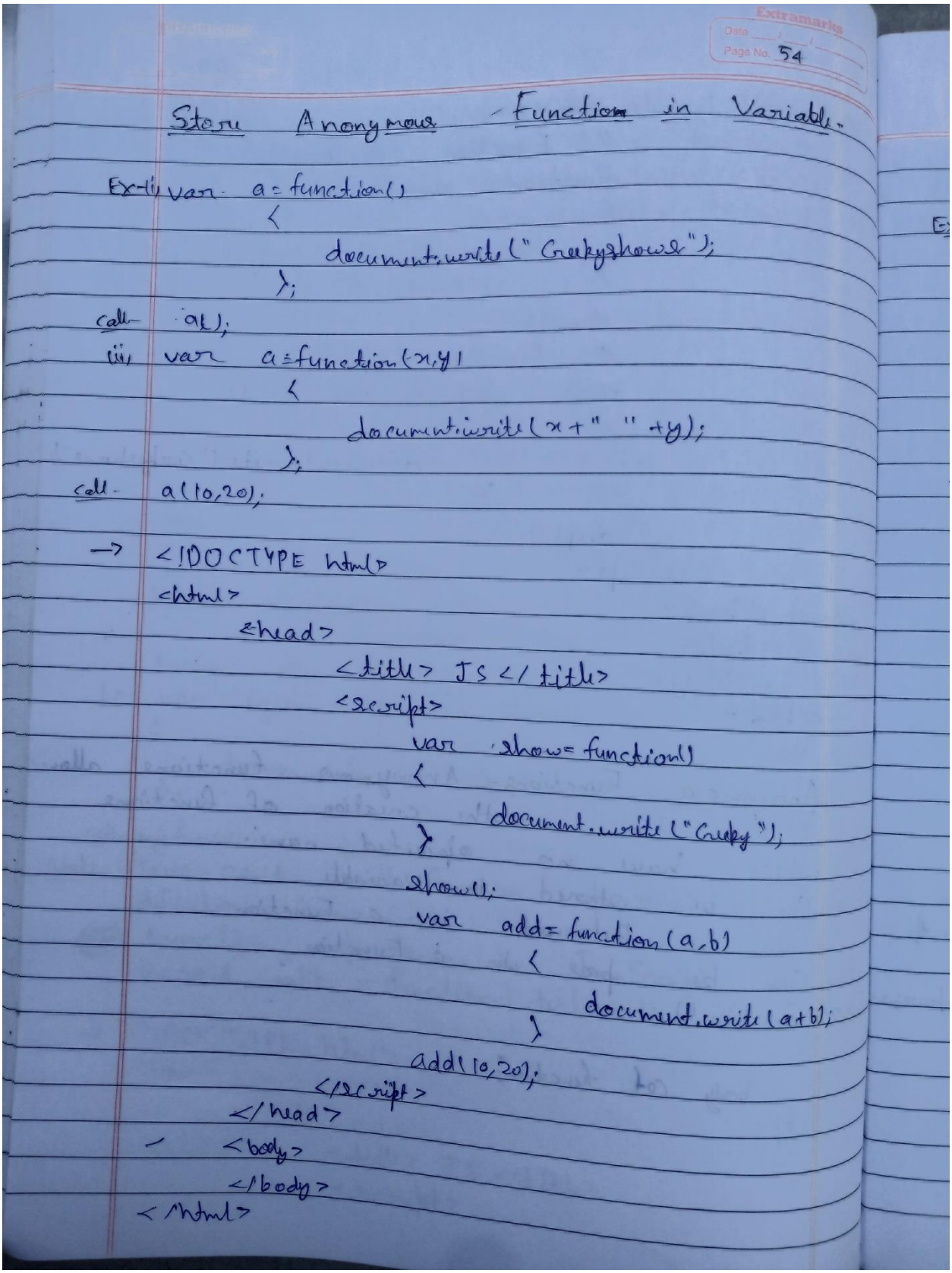
```
.show();
// Function declaration
function show()
{
    document.write("Geekyshows");
}
show();
// Function Expression
var disp = function show()
{
    document.write("Geekyshows");
};
disp();
</script>
</head>
<body>
</body>
</html>
```

Anonymous Functions- Anonymous functions allow the creation of functions which have no specified name.

- Can be stored in Variable
- Can be Returned in a Function
- Can be pass in a Function

Syntax-

```
function()
{
    body of function;
};
```



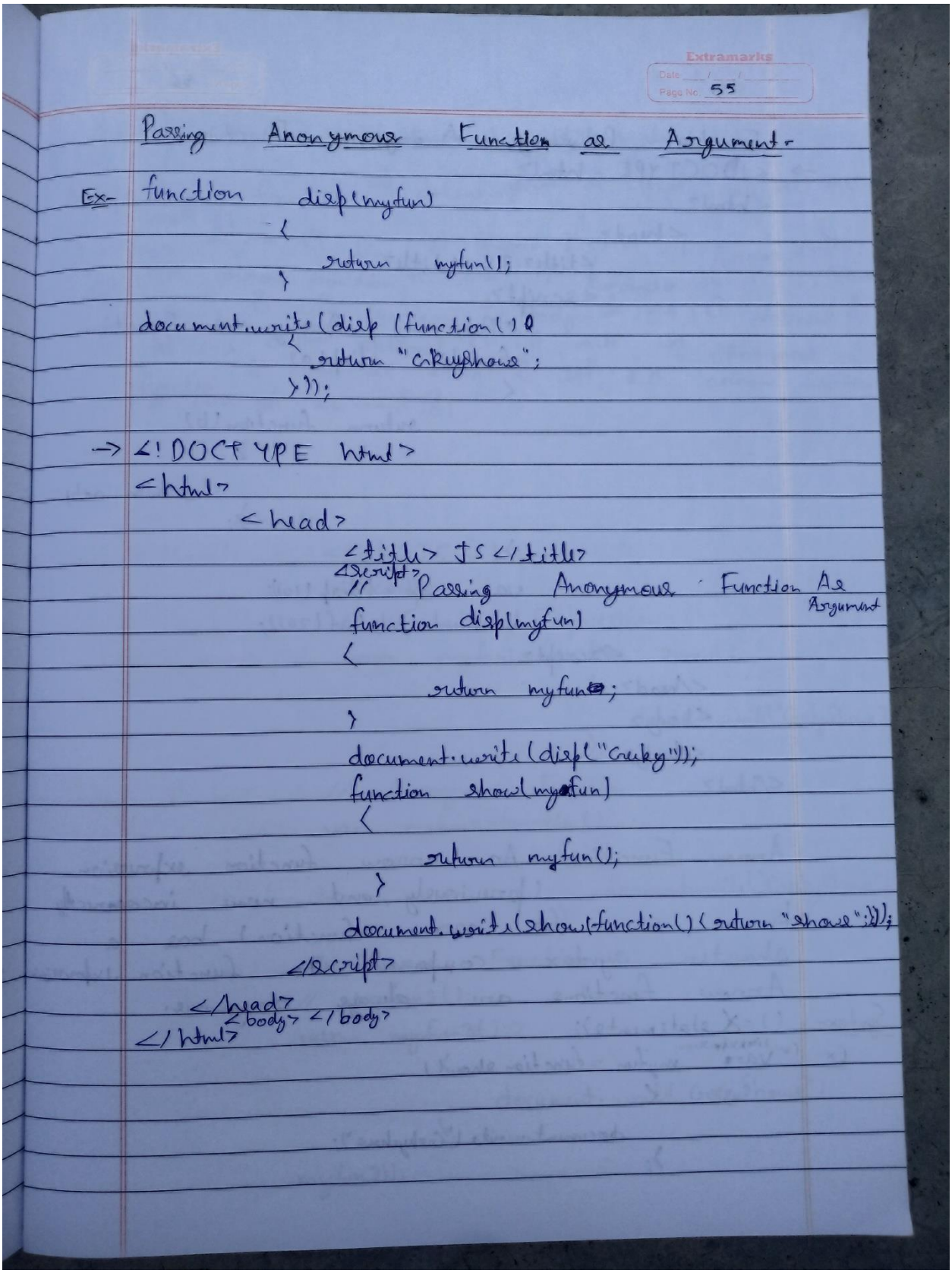
Page No. 54

Store Anonymous Function in Variable.

Ex-i) var a = function()
{
document.write("Geekyshows");
};
call- a();

ii) var a = function(x,y)
{
document.write(x+" "+y);
};
call- a(10,20);

→
```
<!DOCTYPE html>
<html>
<head>
<title> JS </title>
<script>
var show = function()
{
document.write("Cricky");
}
show();
var add = function(a,b)
{
document.write(a+b);
}
add(10,20);
</script>
</head>
<body>
</body>
</html>
```

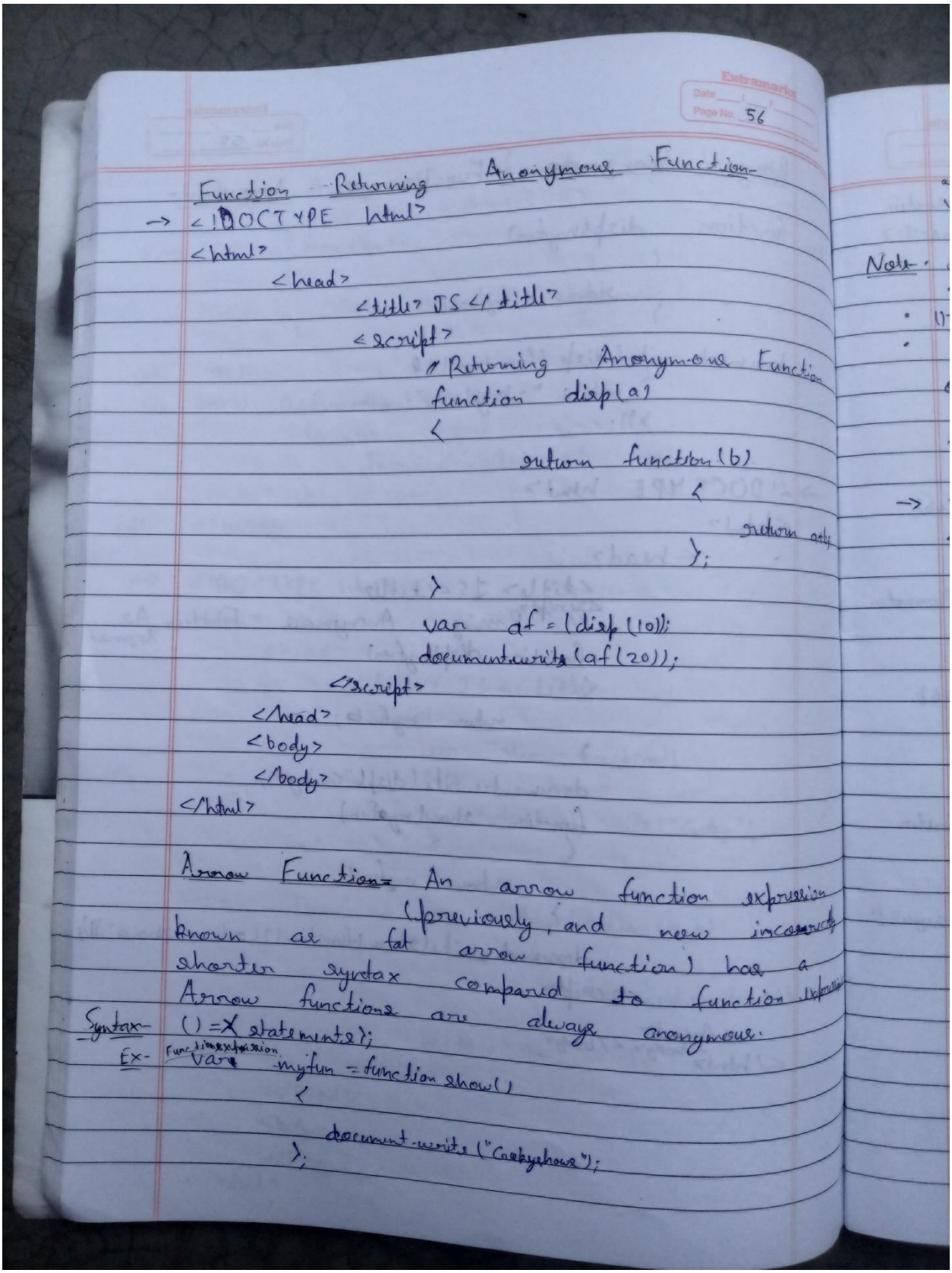


Extramarks
Date / /
Page No. 55

Passing Anonymous Function as Argument.

Ex- function disp(myfun)
{
return myfun();
}
document.write(disp(function(){
return "Geekyshows";
}));

→ <!DOCTYPE html>
<html>
<head>
<title> JS </title>
<script>
// Passing Anonymous Function As Argument
function disp(myfun)
{
return myfun();
}
document.write(disp("Cricky"));
function show(myfun)
{
return myfun();
}
document.write(show(function(){return "show";}));
</script>
</head>
<body> </body>
</html>



Function Returning Anonymous Function-

```
<!DOCTYPE html>
<html>
    <head>
        <title> JS </title>
        <script>
            // Returning Anonymous Function
            function disp(a)
            {
                return function(b)
                {
                    return a+b;
                };
            }
            var af = disp(10);
            document.write(af(20));
        </script>
    </head>
    <body>
    </body>
</html>
```

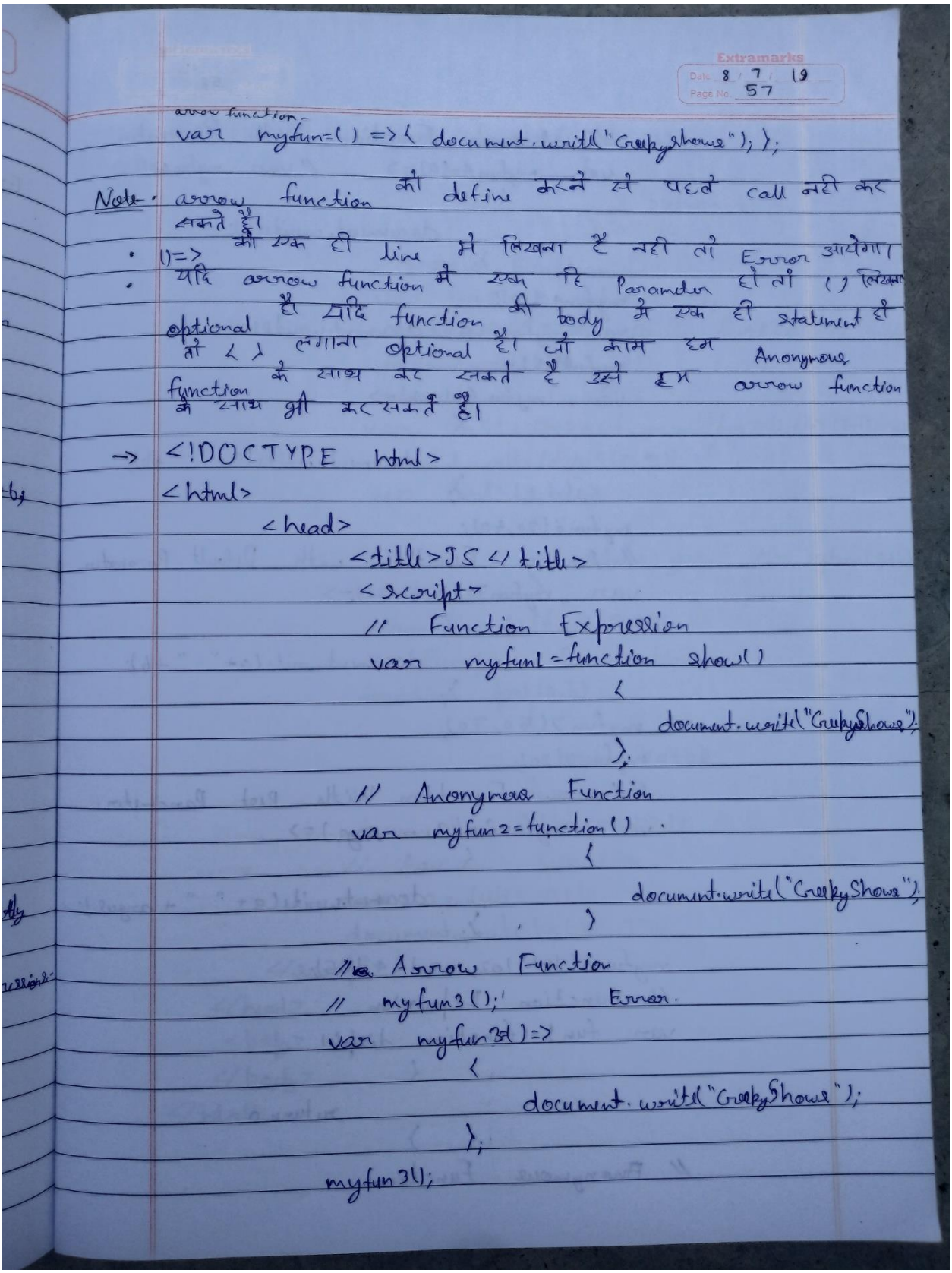
Note.

Arrow Function= An arrow function expression (previously, and now incorrectly known as fat arrow function) has a shorter syntax compared to function expressions. Arrow functions are always anonymous.

Syntax- () => { statements };

Ex- Function expression

```
var myfun = function show()
{
    document.write("Geekyshows");
};
```



Extramarks
Date 8/7/19
Page No. 57

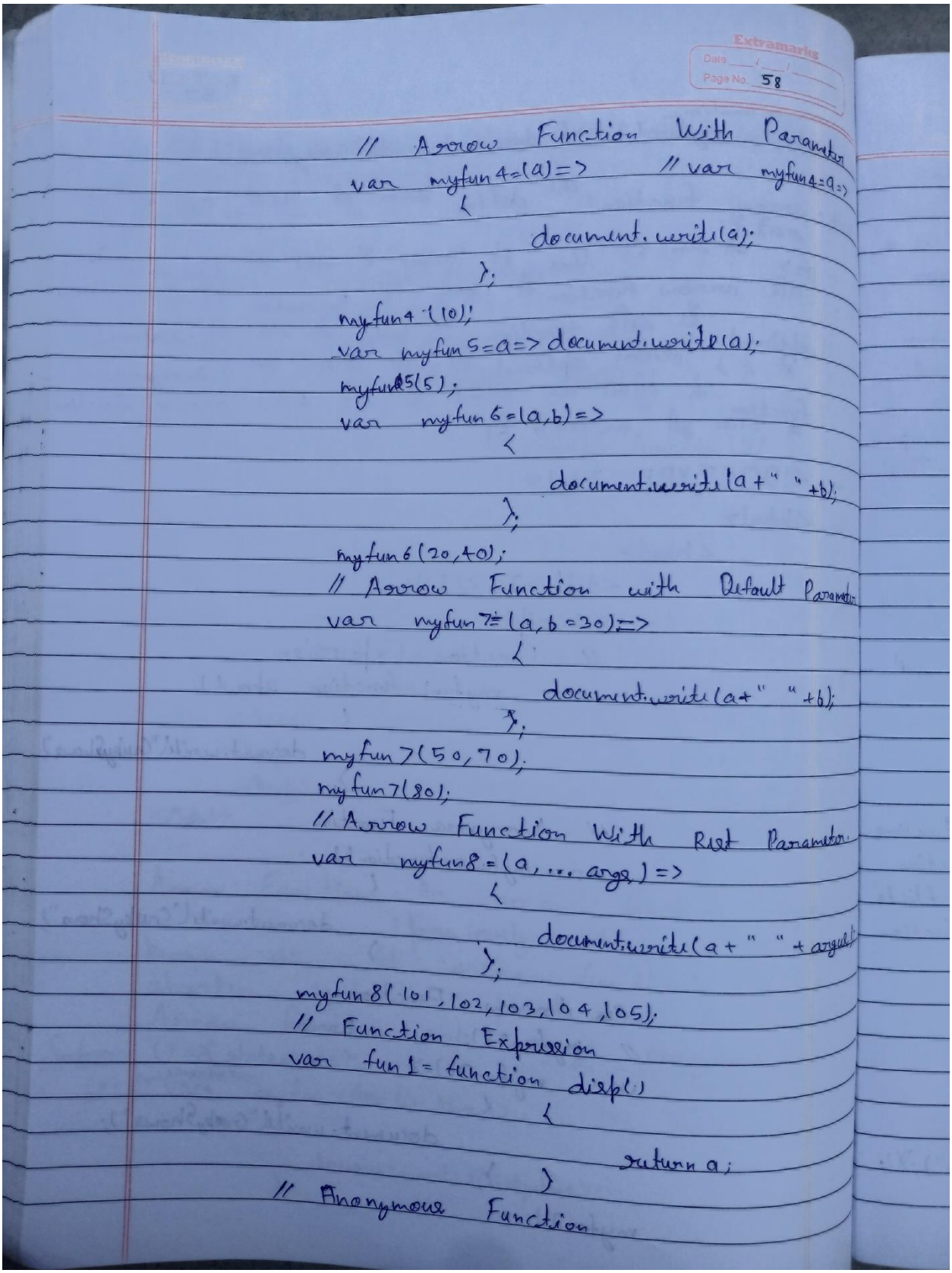
arrow function -

```
var myfun=()=>{ document.write("GeekyShows"); };
```

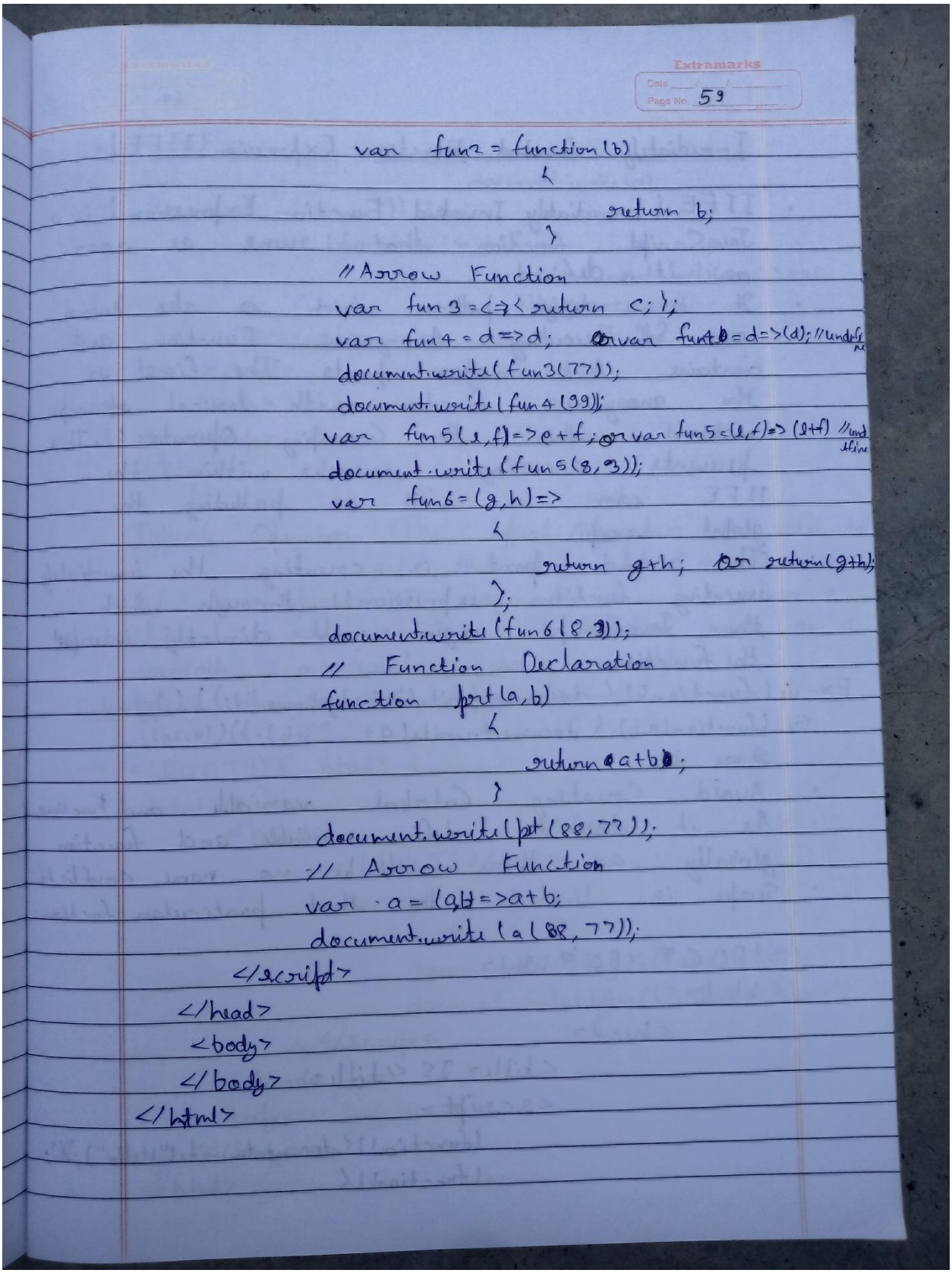
Note: arrow function को define करने से पहले call नहीं कर सकते है।

- ()=> को एक ही line में लिखना है नहीं तो Error आयेगा।
- यदि arrow function में एक ही Parameter हो तो () लिखना optional है यदि function की body में एक ही statement हो तो { } लगाना optional है। जो काम हम Anonymous function के साथ कर सकते है उसी हम arrow function के साथ भी कर सकते है।

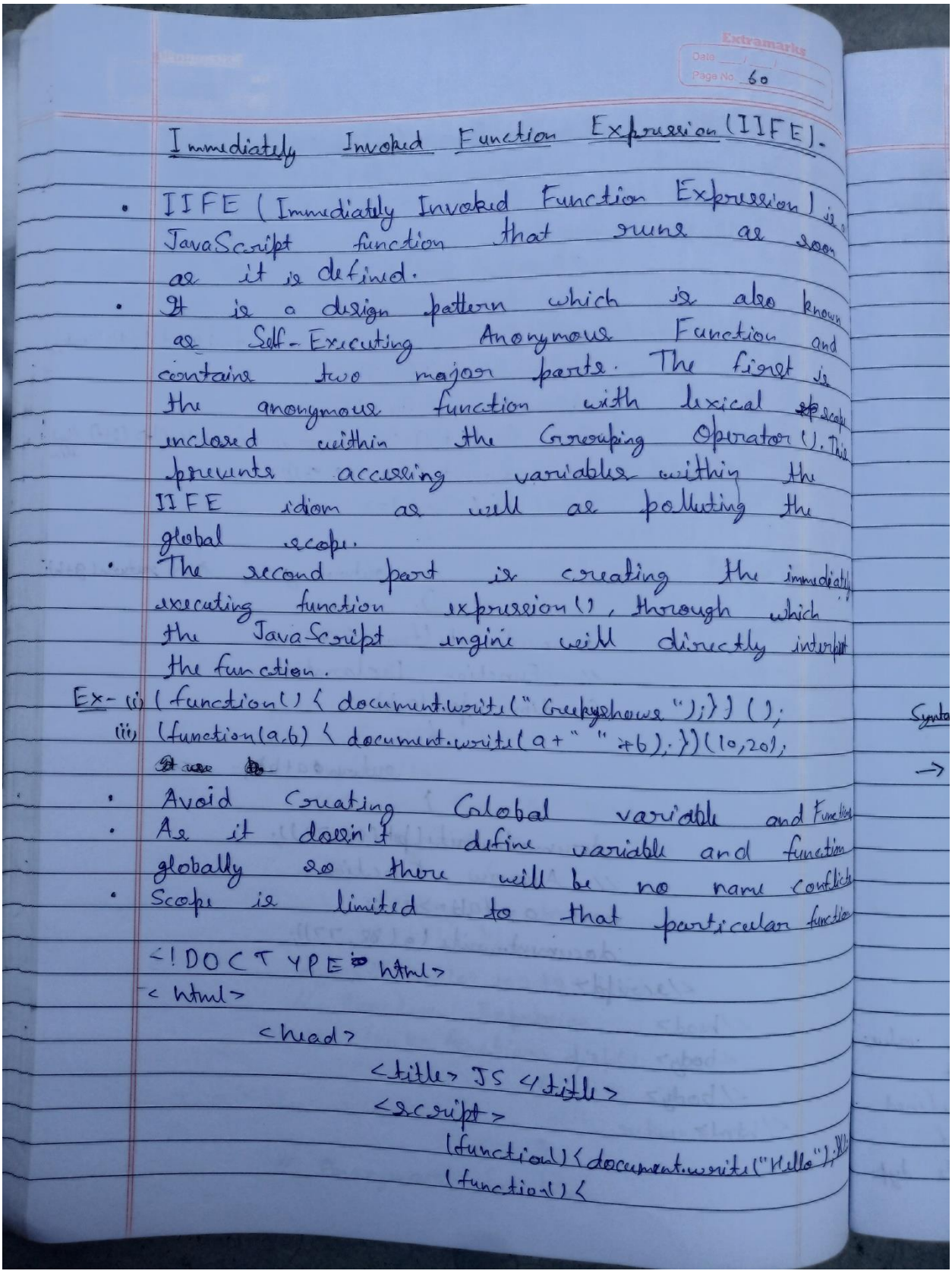
```
-> <!DOCTYPE html>
<html>
    <head>
        <title>JS</title>
        <script>
        // Function Expression
        var myfun1=function show()
                {
                    document.write("GeekyShows");
                };
        // Anonymous Function
        var myfun2=function()
                {
                    document.write("GeekyShows");
                }
        // Arrow Function
        // myfun3();    Error.
        var myfun3()=>
                {
                    document.write("GeekyShows");
                };
        myfun3();
```



```
// Arrow Function With Parameter
var myfun4=(a)=>     // var myfun4=a=>
{
    document.write(a);
};
myfun4(10);
var myfun5=a=>document.write(a);
myfun5(5);
var myfun6=(a,b)=>
{
    document.write(a+" "+b);
};
myfun6(20,40);
// Arrow Function with Default Parameter
var myfun7=(a,b=30)=>
{
    document.write(a+" "+b);
};
myfun7(50,70);
myfun7(80);
// Arrow Function With Rest Parameter
var myfun8=(a,...args)=>
{
    document.write(a+" "+args);
};
myfun8(101,102,103,104,105);
// Function Expression
var fun1=function disp()
{
    return a;
}
// Anonymous Function
```



```
var fun2 = function(b)
{
    return b;
}
// Arrow Function
var fun3 = ()=>{ return c; };
var fun4 = d=>d;  or var fun4 = d=>(d); // undefined
document.write(fun3(77));
document.write(fun4(99));
var fun5 = (e,f)=>e+f; or var fun5 = (e,f)=>(e+f) // undefined
document.write(fun5(8,3));
var fun6 = (g,h)=>
{
    return g+h; or return(g+h);
};
document.write(fun6(8,3));
// Function Declaration
function fnt(a,b)
{
    return a+b;
}
document.write(fnt(88,77));
// Arrow Function
var a = (a,b)=>a+b;
document.write(a(88,77));
</script>
</head>
<body>
</body>
</html>
```



Extramarks
Date
Page No 60

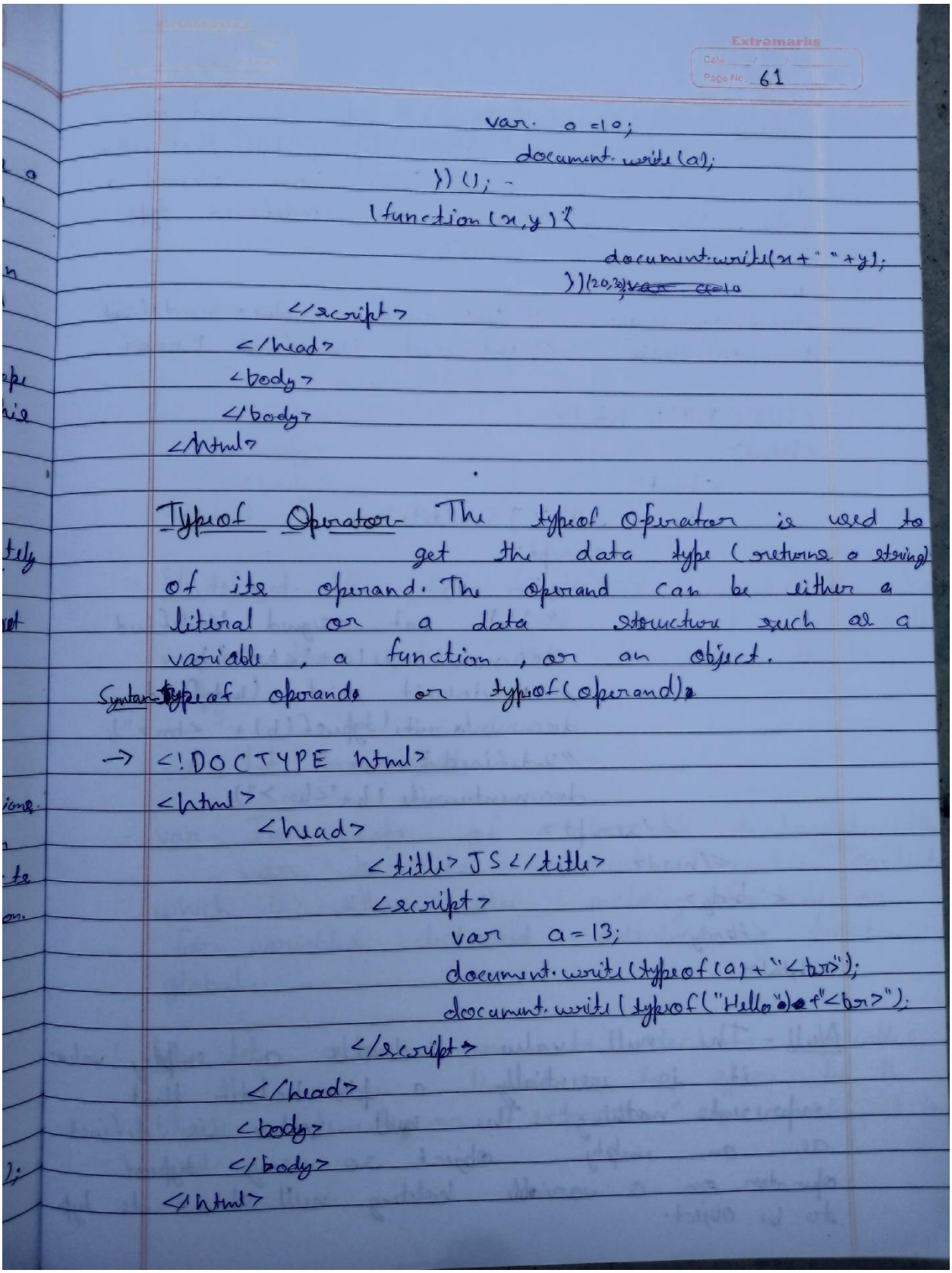
Immediately Invoked Function Expression (IIFE).

- IIFE (Immediately Invoked Function Expression) is a JavaScript function that runs as soon as it is defined.
- It is a design pattern which is also known as Self-Executing Anonymous Function and contains two major parts. The first is the anonymous function with lexical scope enclosed within the Grouping Operator (). This prevents accessing variables within the IIFE idiom as well as polluting the global scope.
- The second part is creating the immediately executing function expression (), through which the JavaScript engine will directly interpret the function.

Ex- (i) (function() { document.write("Geekyshows "); })();
(ii) (function(a,b) { document.write(a+" "+b); })(10,20);

- Avoid creating Global variable and Function.
- As it doesn't define variable and function globally so there will be no name conflict.
- Scope is limited to that particular function.

```
<!DOCTYPE html>
<html>
    <head>
        <title> JS </title>
        <script>
            (function(){ document.write("Hello"); })();
            (function(){
```



Extramarks
Page No. 61

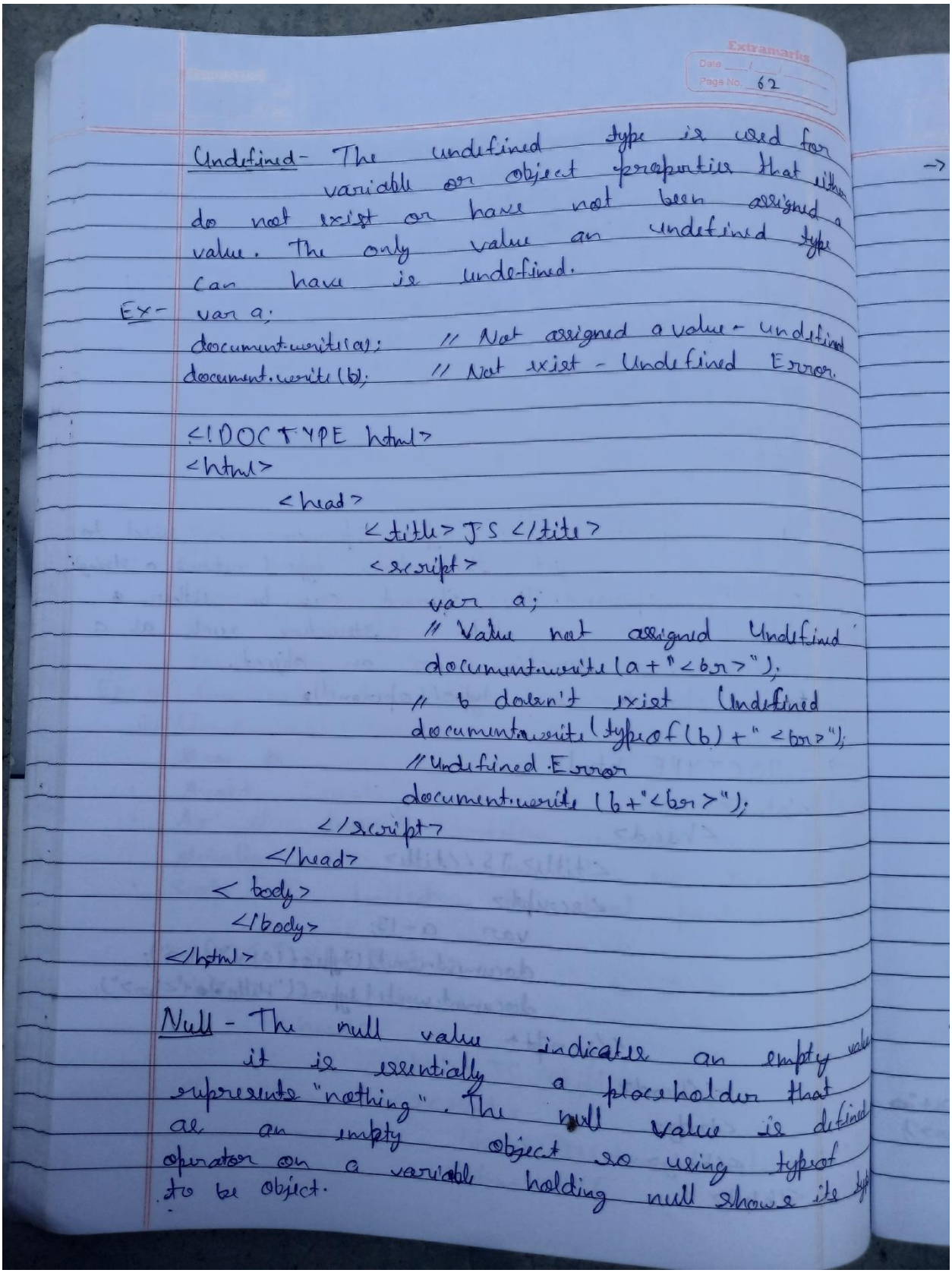
```
        var a = 10;
        document.write(a);
    })();
    (function(x,y){
        document.write(x+" "+y);
    })(20,30);
</script>
</head>
<body>
</body>
</html>
```

Typeof Operator- The typeof operator is used to get the data type (returns a string) of its operand. The operand can be either a literal or a data structure such as a variable, a function, or an object.

Syntax- typeof operand or typeof(operand)

→

```
<!DOCTYPE html>
<html>
    <head>
        <title>JS</title>
        <script>
            var a=13;
            document.write(typeof(a)+"<br>");
            document.write(typeof("Hello")+"<br>");
        </script>
    </head>
    <body>
    </body>
</html>
```

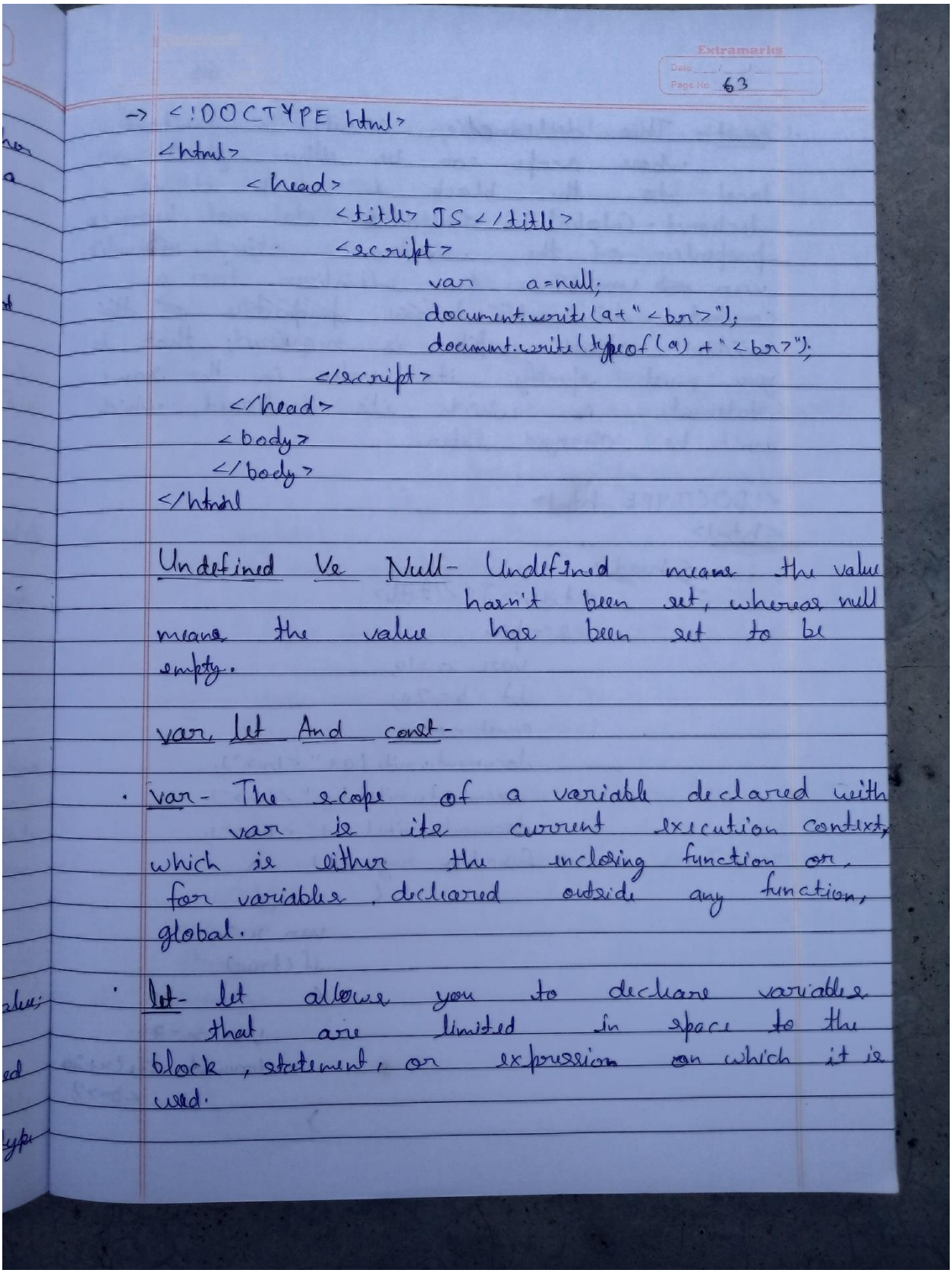


Undefined - The undefined type is used for variable or object properties that either do not exist or have not been assigned a value. The only value an undefined type can have is undefined.

Ex-
```
var a;
document.write(a);    // Not assigned a value - undefined
document.write(b);    // Not exist - Undefined Error.
```

```
<!DOCTYPE html>
<html>
    <head>
        <title> JS </title>
        <script>
            var a;
            // Value not assigned Undefined
            document.write(a + "<br>");
            // b doesn't exist (Undefined
            document.write(typeof(b) + "<br>");
            // Undefined Error
            document.write(b + "<br>");
        </script>
    </head>
    <body>
    </body>
</html>
```

Null - The null value indicates an empty value it is essentially a placeholder that represents "nothing". The null value is defined as an empty object so using typeof operator on a variable holding null shows its type to be object.



Extramarks
Date
Page No. 63

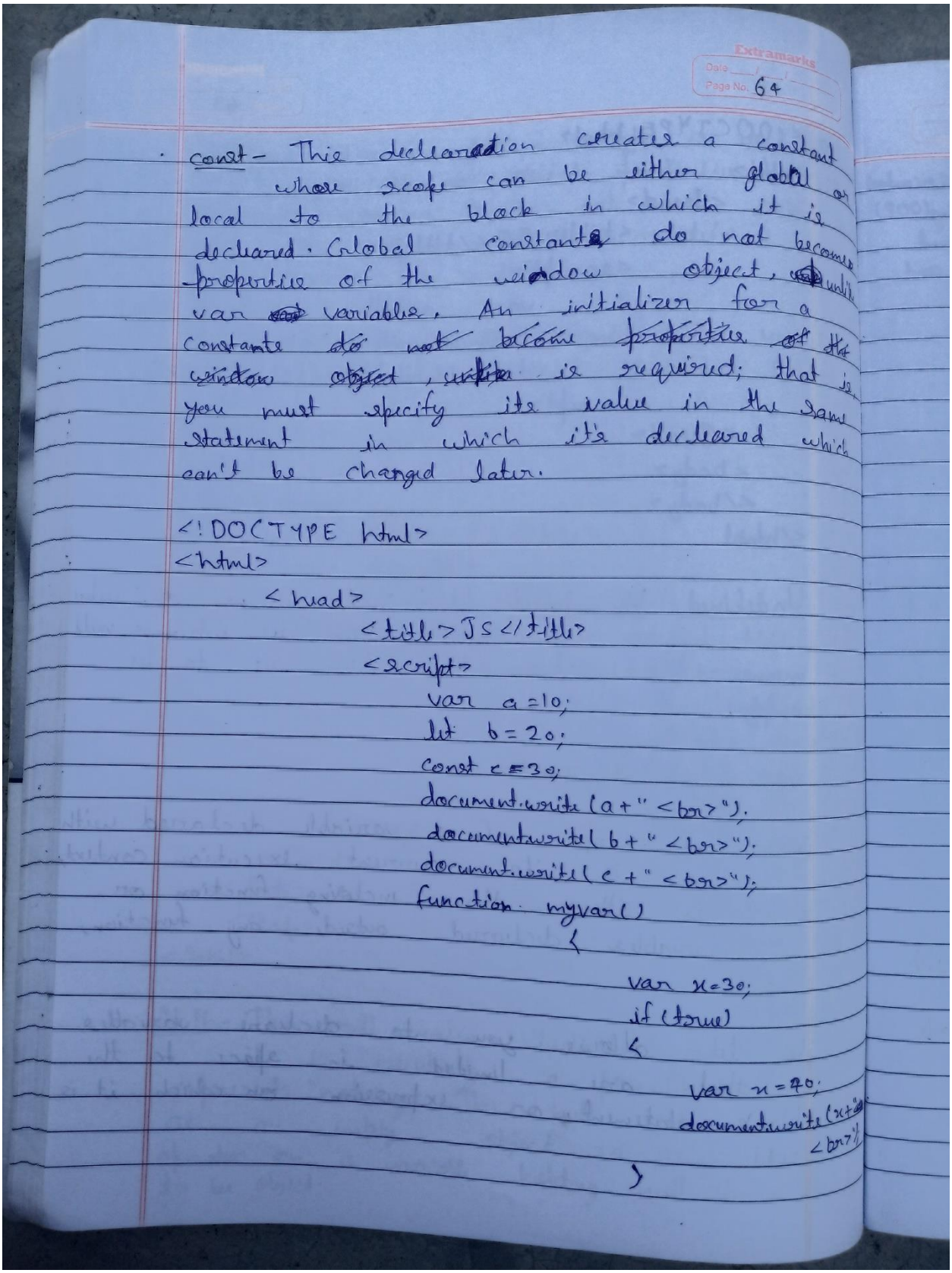
-> <!DOCTYPE html>
<html>
<head>
<title> JS </title>
<script>
var a=null;
document.write(a+"
");
document.write(typeof(a)+"
");
</script>
</head>
<body>
</body>
</html>

Undefined Vs Null- Undefined means the value hasn't been set, whereas null means the value has been set to be empty.

var, let And const-

- var- The scope of a variable declared with var is its current execution context, which is either the enclosing function or, for variables declared outside any function, global.

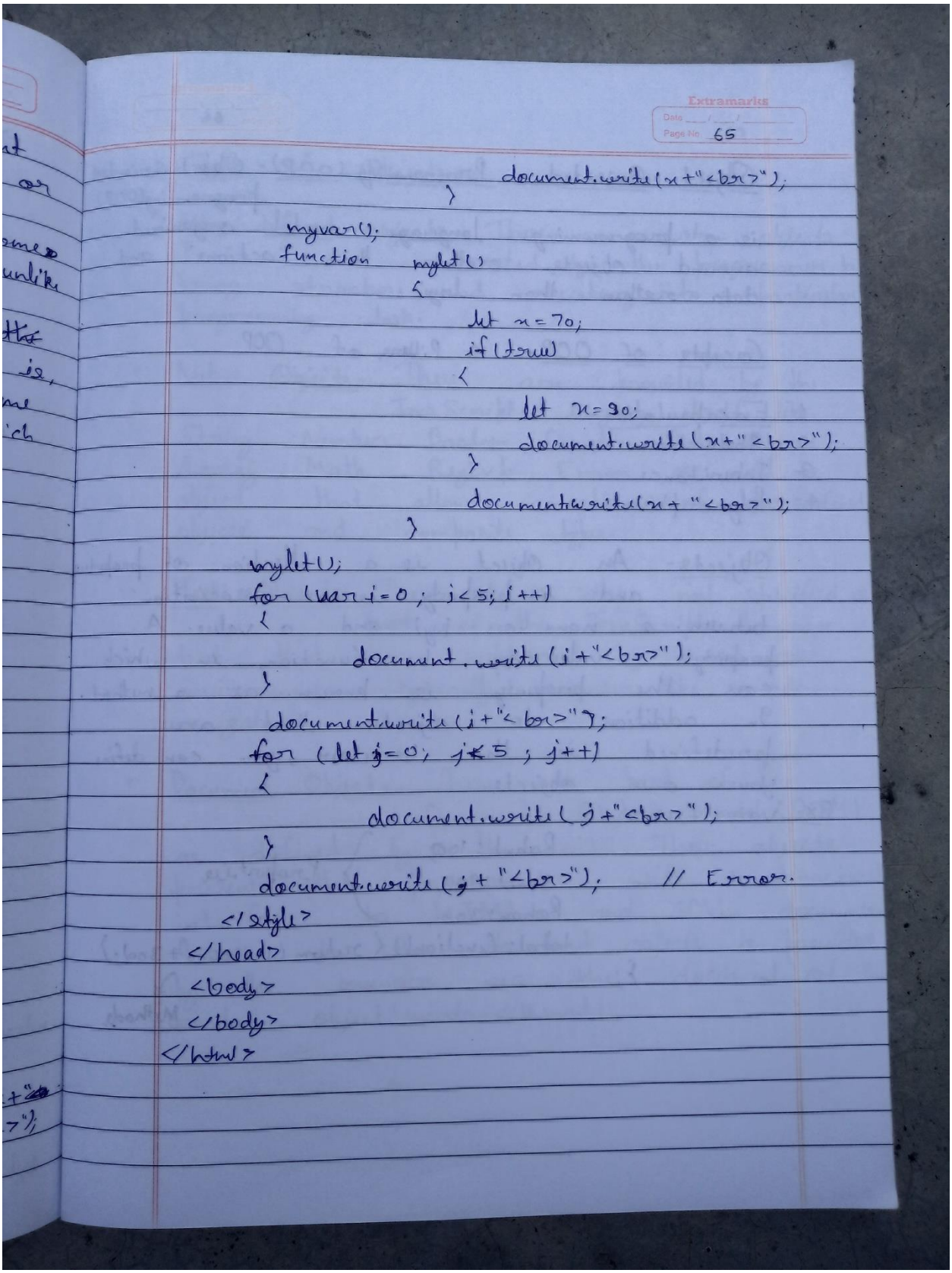
- let- let allows you to declare variables that are limited in space to the block, statement, or expression on which it is used.



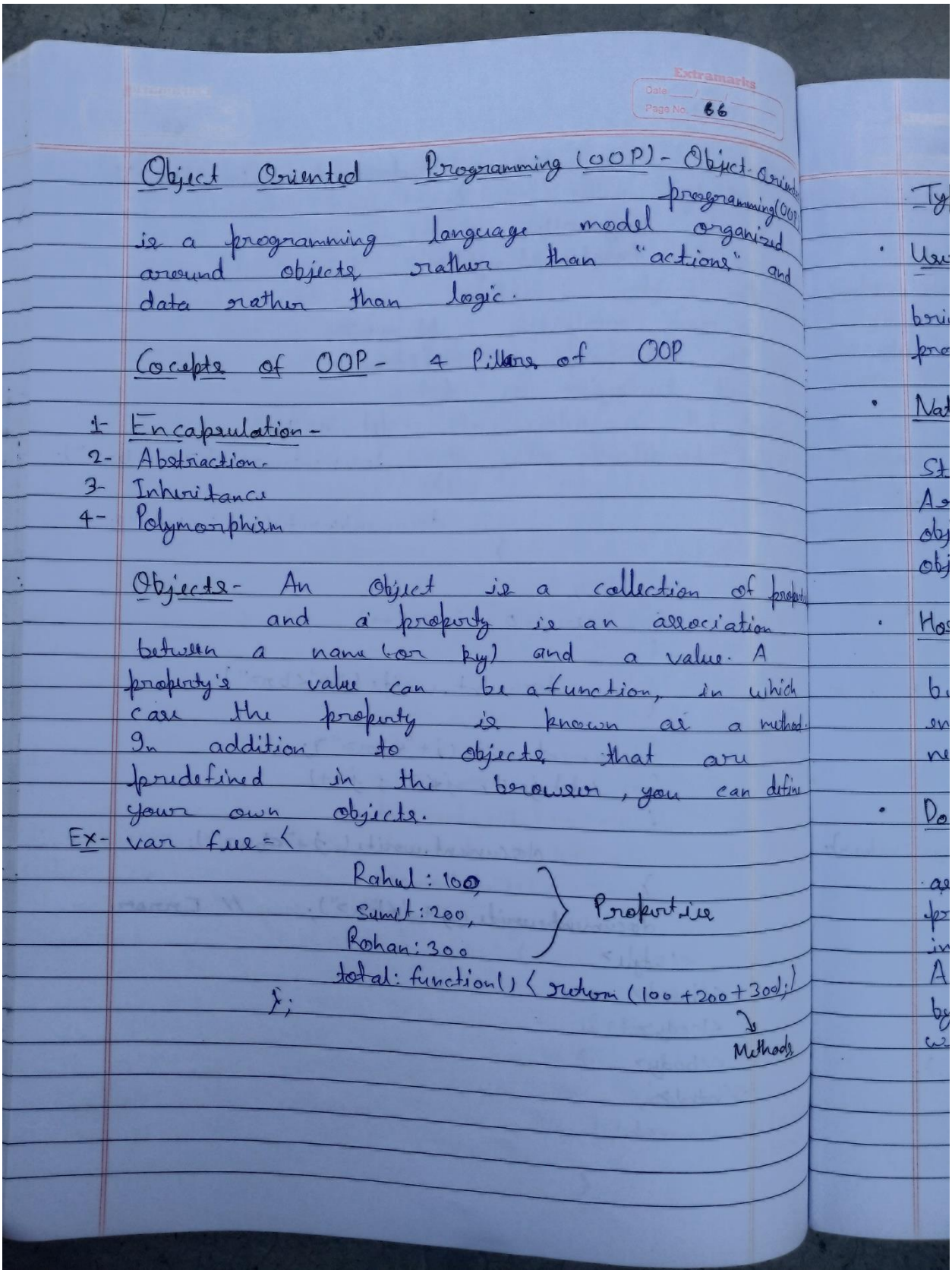
Extramarks
Page No. 64

- const - This declaration creates a constant whose scope can be either global or local to the block in which it is declared. Global constants do not become properties of the window object, unlike var variables. An initializer for a constant is required; that is you must specify its value in the same statement in which it's declared which can't be changed later.

```
<!DOCTYPE html>
<html>
    <head>
        <title> JS </title>
        <script>
            var a=10;
            let b=20;
            const c=30;
            document.write(a+"<br>");
            document.write(b+"<br>");
            document.write(c+"<br>");
            function myvar()
            {
                var x=30;
                if(true)
                {
                    var x=40;
                    document.write(x+"<br>");
                }
```



```
                document.write(x+"<br>");
            }
myvar();
function mylet()
{
    let x=70;
    if(true)
    {
        let x=90;
        document.write(x+"<br>");
    }
    document.write(x+"<br>");
}
mylet();
for(var i=0; i<5; i++)
{
    document.write(i+"<br>");
}
document.write(i+"<br>");
for(let j=0; j<5; j++)
{
    document.write(j+"<br>");
}
document.write(j+"<br>");    // Error.
</style>
</head>
<body>
</body>
</html>
```



Extramarks
Date
Page No 66

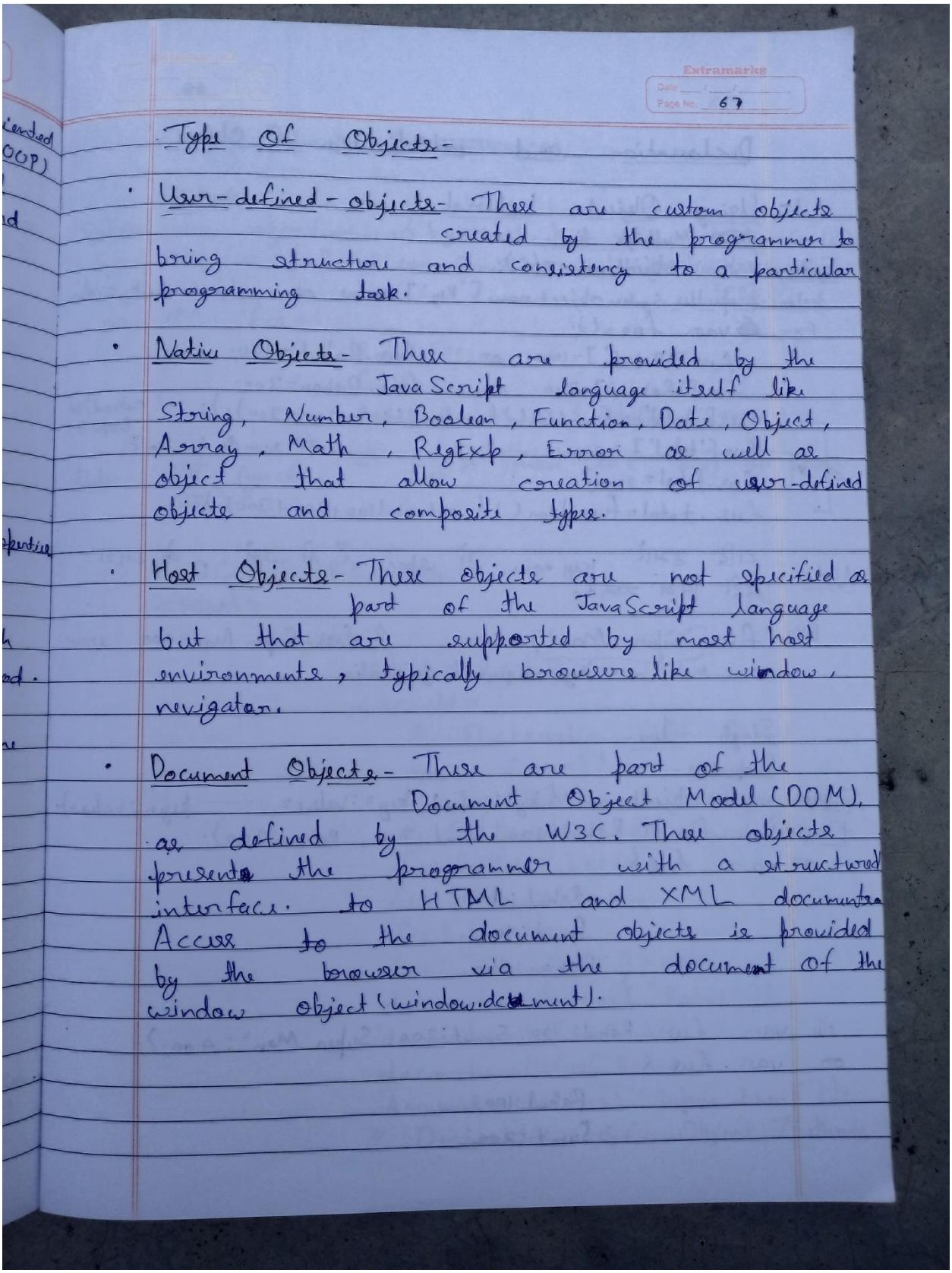
Object Oriented Programming (OOP) - Object Oriented programming (OOP) is a programming language model organized around objects rather than "actions" and data rather than logic.

Cocepts of OOP - 4 Pillars of OOP

1- Encapsulation -
2- Abstraction.
3- Inheritance
4- Polymorphism

Objects - An Object is a collection of property and a property is an association between a name (or key) and a value. A property's value can be a function, in which case the property is known as a method. In addition to objects that are predefined in the browser, you can define your own objects.

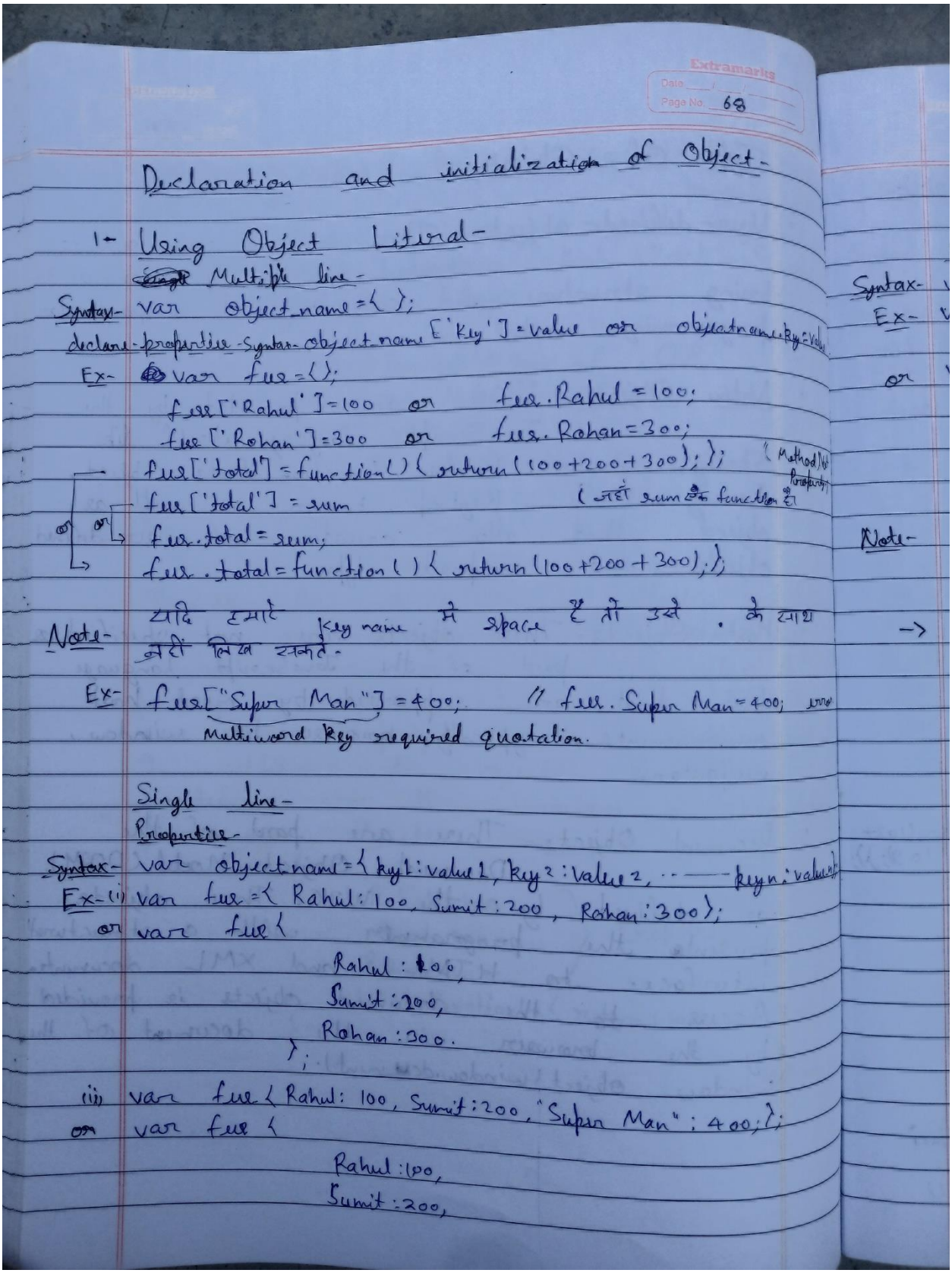
Ex- var fee = {
Rahul : 100,
Sumit : 200,
Rohan : 300
} Properties
total : function() { return (100 + 200 + 300); }
↓
Methods
};



Extramarks
Date
Page No. 67

Type of Objects-

- User-defined-objects- These are custom objects created by the programmer to bring structure and consistency to a particular programming task.
- Native Objects- These are provided by the JavaScript language itself like String, Number, Boolean, Function, Date, Object, Array, Math, RegExp, Error as well as object that allow creation of user-defined objects and composite types.
- Host Objects- These objects are not specified as part of the JavaScript language but that are supported by most host environments, typically browsers like window, navigator.
- Document Objects- These are part of the Document Object Model (DOM), as defined by the W3C. These objects presents the programmer with a structured interface to HTML and XML documents. Access to the document objects is provided by the browser via the document of the window object (window.document).



Extramarks
Date / /
Page No. 68

Declaration and initialization of Object-

1- Using Object Literal-

Multiple line-

Syntax- var objectname={ };

declare-properties-Syntax- objectname['key']=value or objectname.key=value

Ex- var fun={};

fun['Rahul']=100 or fun.Rahul=100;

fun['Rohan']=300 or fun.Rohan=300;

fun['total']=function(){return(100+200+300);}; (Method/Property)

or fun['total']=sum (यहाँ sum एक function है)

or fun.total=sum;

fun.total=function(){return(100+200+300);};

Note- यदि हमारे key name में space है तो उसे . के द्वारा नहीं लिख सकते।

Ex- fun["Super Man"]=400; // fun.Super Man=400; error

Multiword key required quotation.

Single line-

Properties-

Syntax- var objectname={key1:value1, key2:value2, ----- keyn:valuen};

Ex-(i) var fun={Rahul:100, Sumit:200, Rohan:300};

or var fun{

Rahul:100,

Sumit:200,

Rohan:300.

};

(ii) var fun{Rahul:100, Sumit:200, "Super Man":400;};

or var fun{

Rahul:100,

Sumit:200,

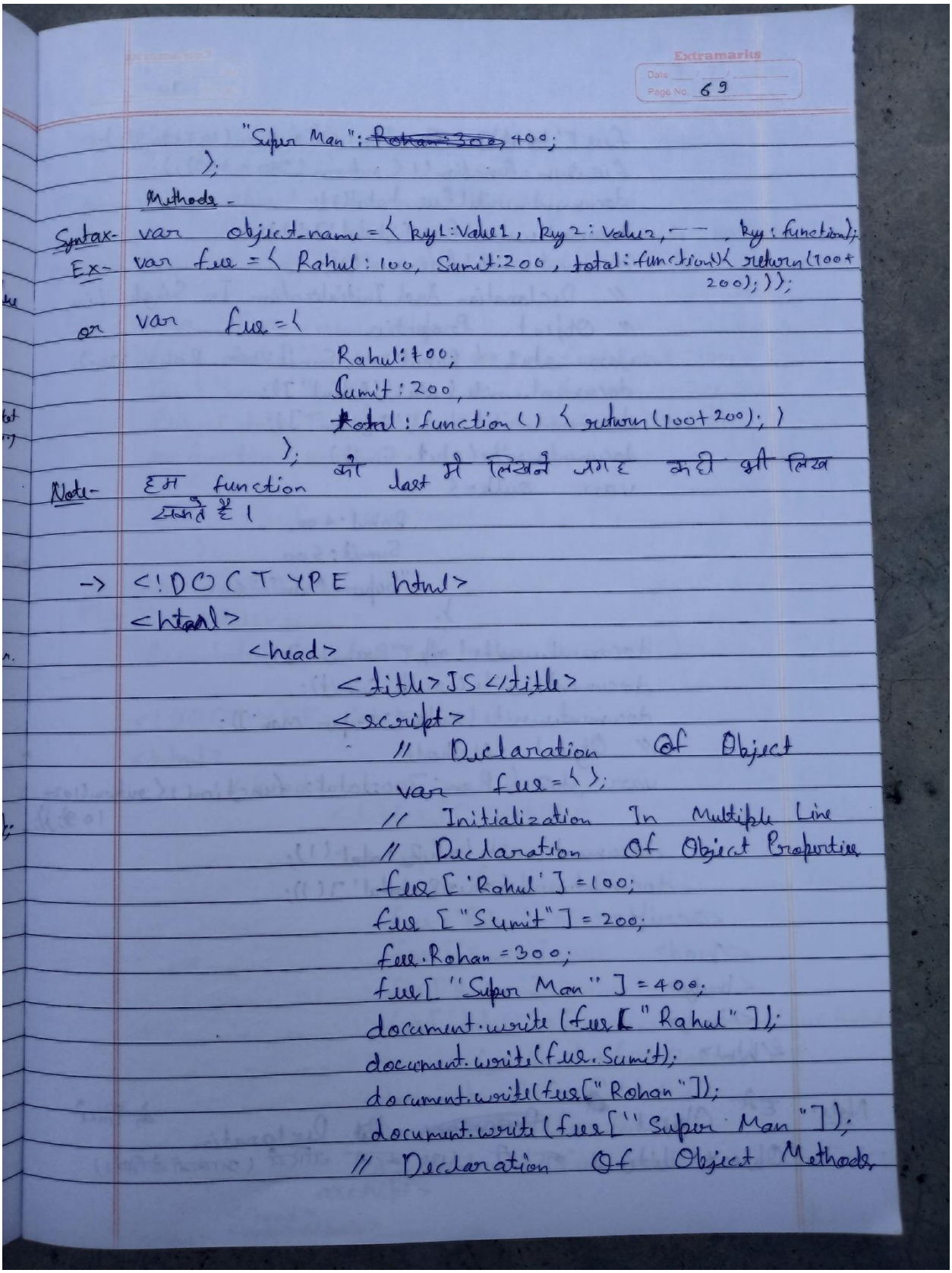
Syntax-

Ex-

or

Note-

->



"Super Man": ~~Rohan: 300~~ 400;
};

Methods -

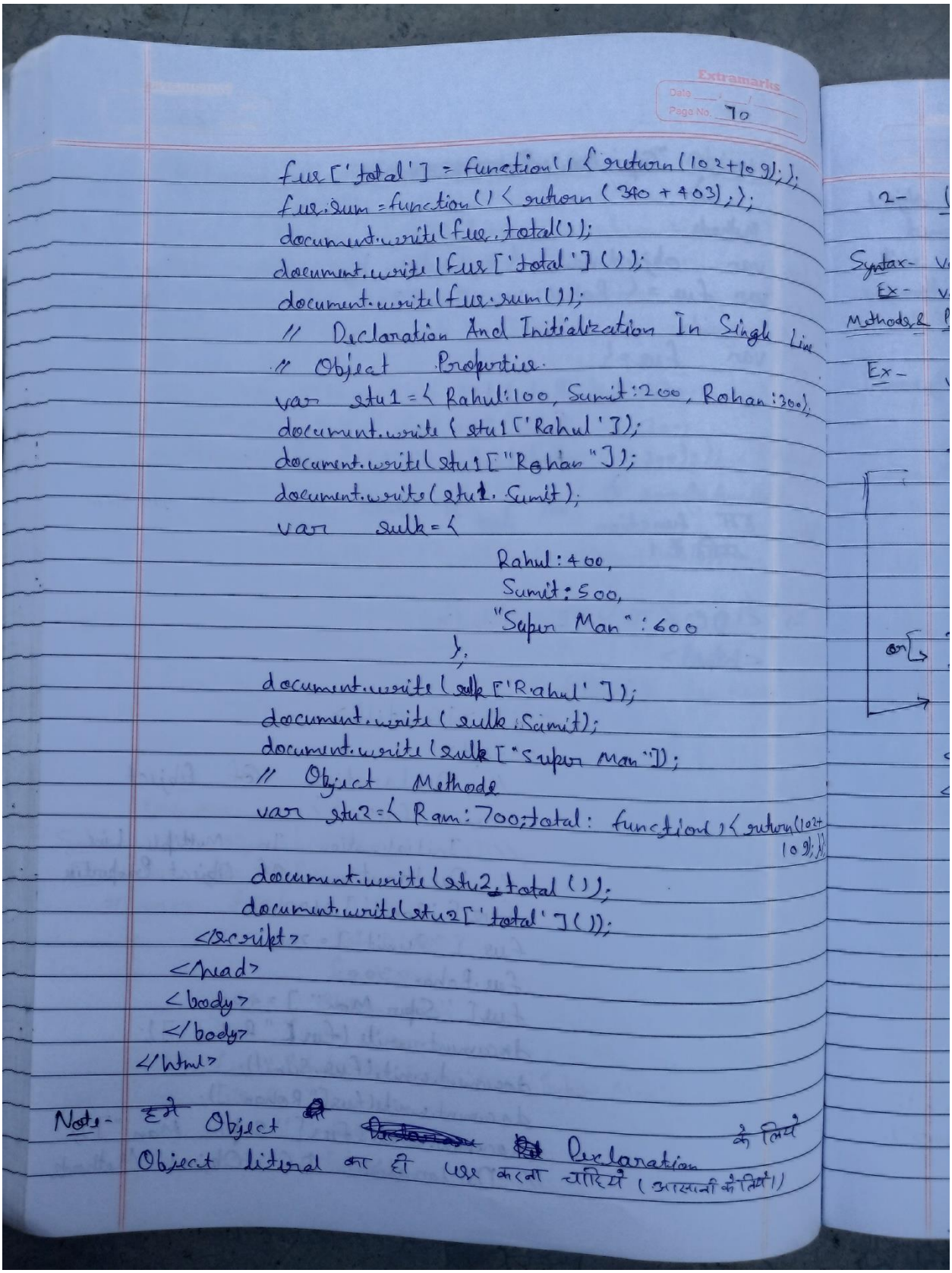
Syntax- var object_name = { key1: value1, key2: value2, ---, key: function};

Ex- var fun = { Rahul: 100, Sumit: 200, total: function(){ return(100+200); }};

or var fun = {
Rahul: 100,
Sumit: 200,
total: function() { return(100+200); }
};

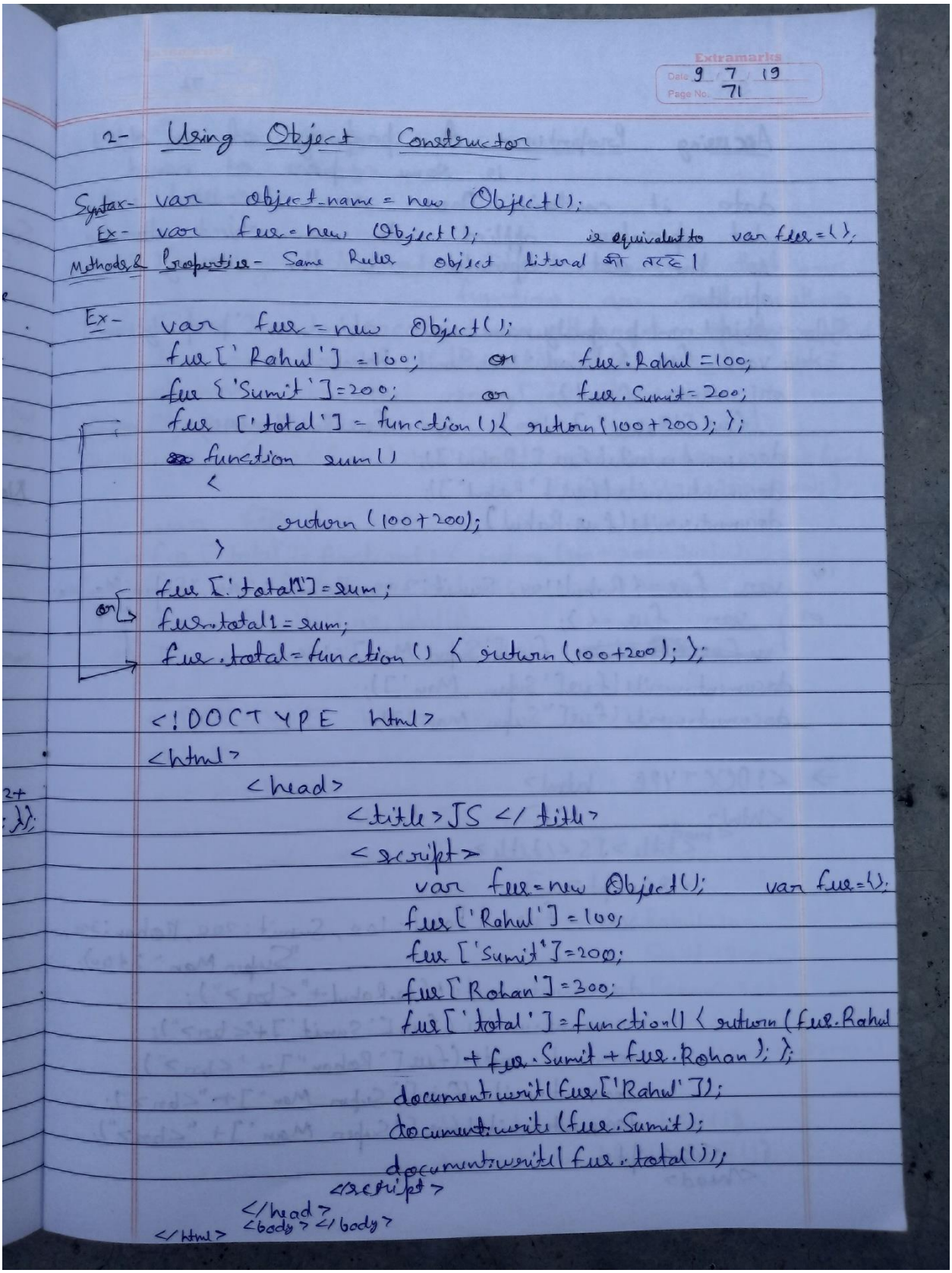
Note- ES function को last में लिखने लगे कहीं भी लिख सकते हैं।

-> <!DOCTYPE html>
<html>
<head>
<title> JS </title>
<script>
// Declaration of Object
var fun = {};
// Initialization In Multiple Line
// Declaration Of Object Properties
fun['Rahul'] = 100;
fun["Sumit"] = 200;
fun.Rohan = 300;
fun["Super Man"] = 400;
document.write(fun["Rahul"]);
document.write(fun.Sumit);
document.write(fun["Rohan"]);
document.write(fun["Super Man"]);
// Declaration Of Object Methods,



```
fus['total'] = function(){return(102+109);};
fus.sum = function(){return(340+403);};
document.write(fus.total());
document.write(fus['total']());
document.write(fus.sum());
// Declaration And Initialization In Singh Line
// Object Properties
var stu1 = { Rahul:100, Sumit:200, Rohan:300};
document.write(stu1['Rahul']);
document.write(stu1["Rohan"]);
document.write(stu1.Sumit);
var sulk={
            Rahul:400,
            Sumit:500,
            "Super Man":600
          };
document.write(sulk['Rahul']);
document.write(sulk.Sumit);
document.write(sulk["Super Man"]);
// Object Methods
var stu2={ Ram:700;total: function(){return(102+109);}
document.write(stu2.total());
document.write(stu2['total']());
</script>
</head>
<body>
</body>
</html>
```

Note- ~~ई~~ Object ~~[redacted]~~ Declaration के लिये Object literal का ही use करना चाहिये (आसानी के लिये)

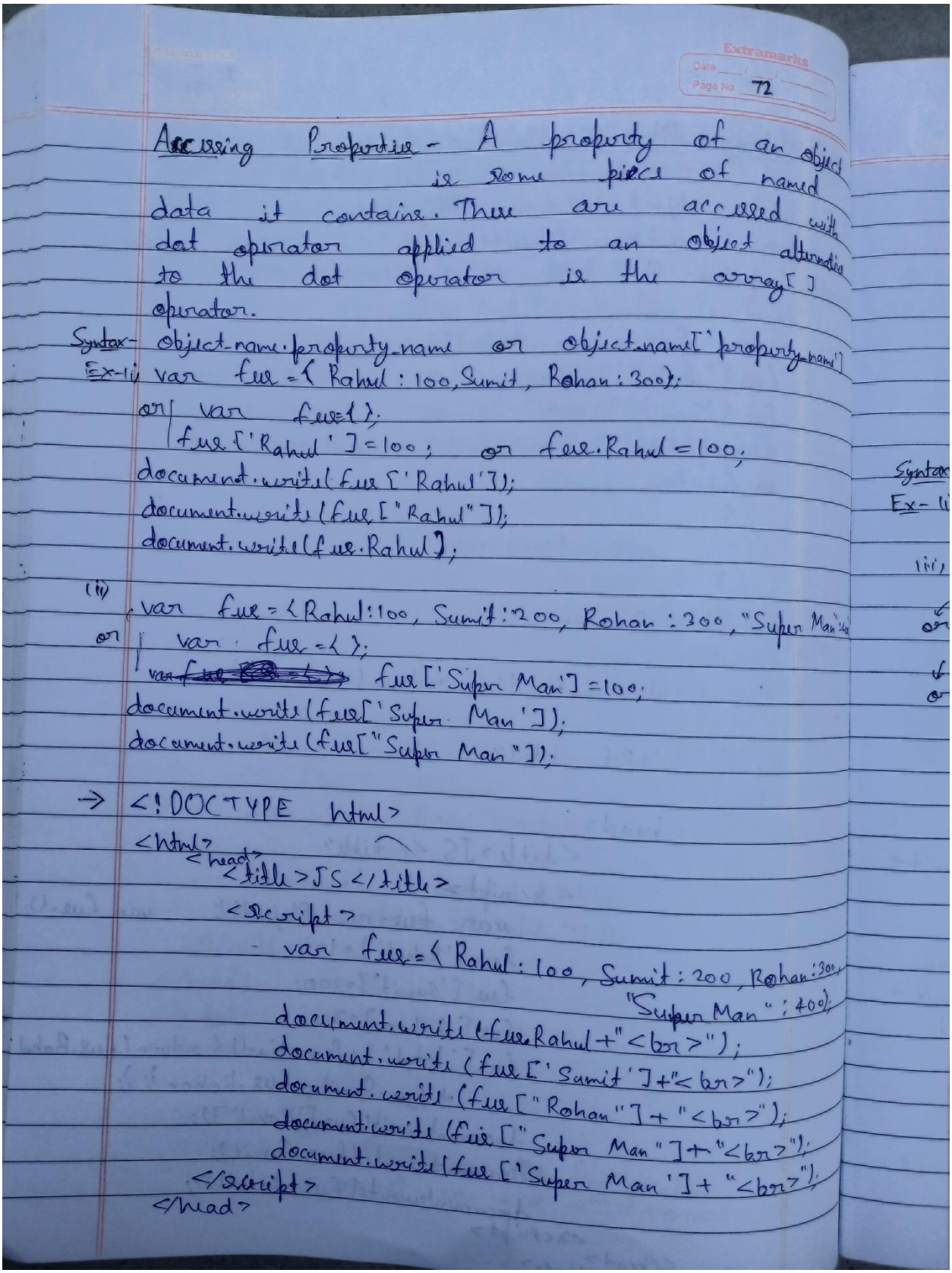


Extramarks
Date 9/7/19
Page No. 71

2- Using Object Constructor

Syntax- var object_name = new Object();
Ex- var fun = new Object(); is equivalent to var fun = {};
Methods & Properties - Same Rules object literal की तरह ।

Ex-
```
var fun = new Object();
fun['Rahul'] = 100;        or   fun.Rahul = 100;
fun['Sumit'] = 200;        or   fun.Sumit = 200;
fun['total'] = function(){ return (100+200); };
function sum()
{
    return (100+200);
}
fun['total1'] = sum;
or
fun.total1 = sum;
fun.total = function(){ return (100+200); };

<!DOCTYPE html>
<html>
    <head>
        <title>JS</title>
        <script>
            var fun = new Object();   var fun = {};
            fun['Rahul'] = 100;
            fun['Sumit'] = 200;
            fun['Rohan'] = 300;
            fun['total'] = function(){ return (fun.Rahul
                + fun.Sumit + fun.Rohan); };
            document.write(fun['Rahul']);
            document.write(fun.Sumit);
            document.write(fun.total());
        </script>
    </head>
    <body></body>
</html>
```



Accessing Properties - A property of an object is some piece of named data it contains. These are accessed with dot operator applied to an object alternative to the dot operator is the array [ ] operator.

Syntax- object_name.property_name or object_name['property_name']

Ex-(i) var fun = { Rahul : 100, Sumit, Rohan : 300};

or var fun={};

fun['Rahul'] = 100; or fun.Rahul = 100;

document.write(fun['Rahul']);

document.write(fun["Rahul"]);

document.write(fun.Rahul);

(ii)

var fun = {Rahul:100, Sumit:200, Rohan : 300, "Super Man":400};

or var fun = {};

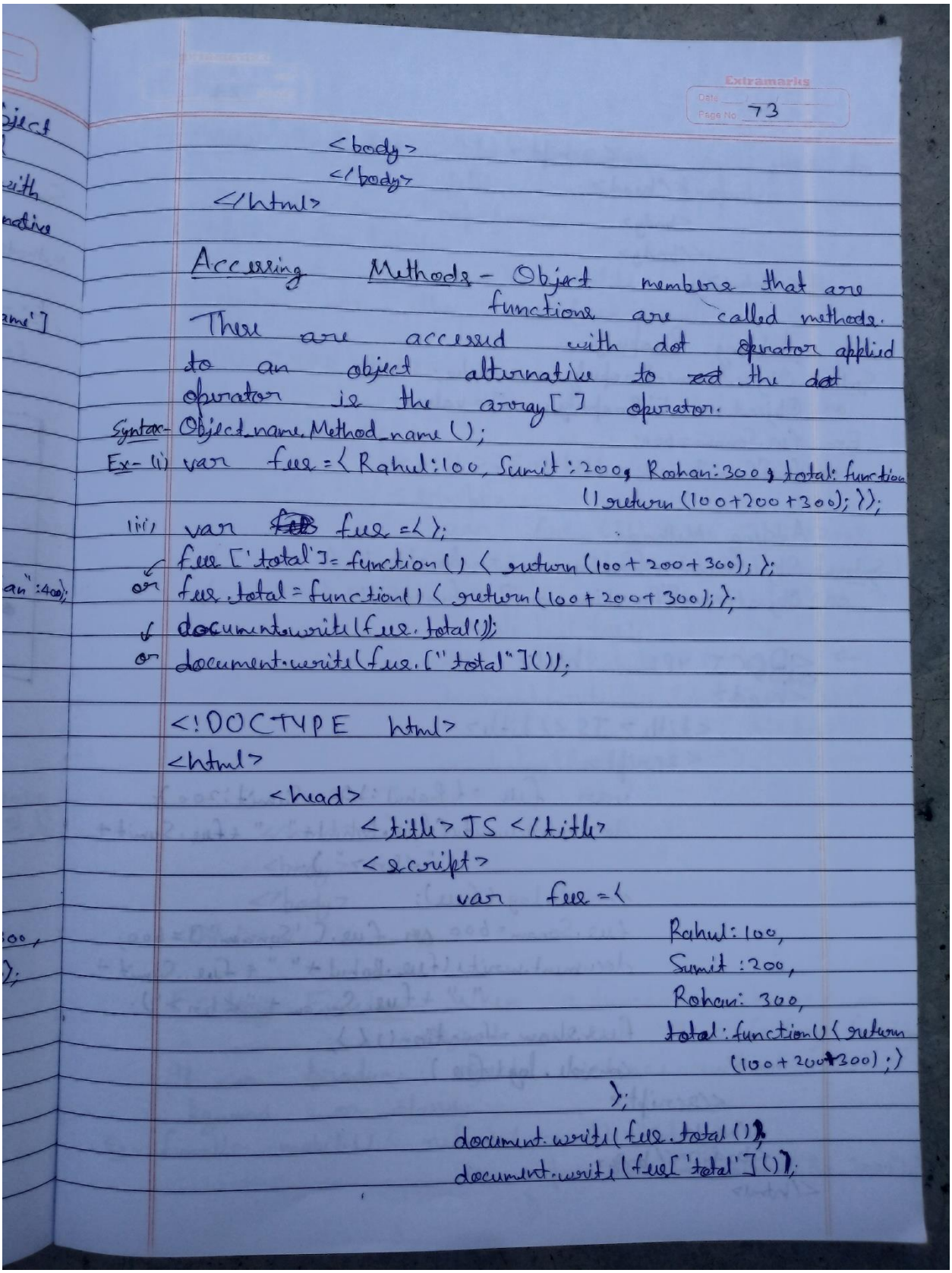
fun['Super Man'] = 100;

document.write(fun['Super Man']);

document.write(fun["Super Man"]);

→

```
<!DOCTYPE html>
<html>
  <head>
    <title>JS</title>
    <script>
      var fun = { Rahul : 100, Sumit : 200, Rohan : 300,
                  "Super Man" : 400};
      document.write(fun.Rahul + "<br>");
      document.write(fun['Sumit'] + "<br>");
      document.write(fun["Rohan"] + "<br>");
      document.write(fun["Super Man"] + "<br>");
      document.write(fun['Super Man'] + "<br>");
    </script>
  </head>
```



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

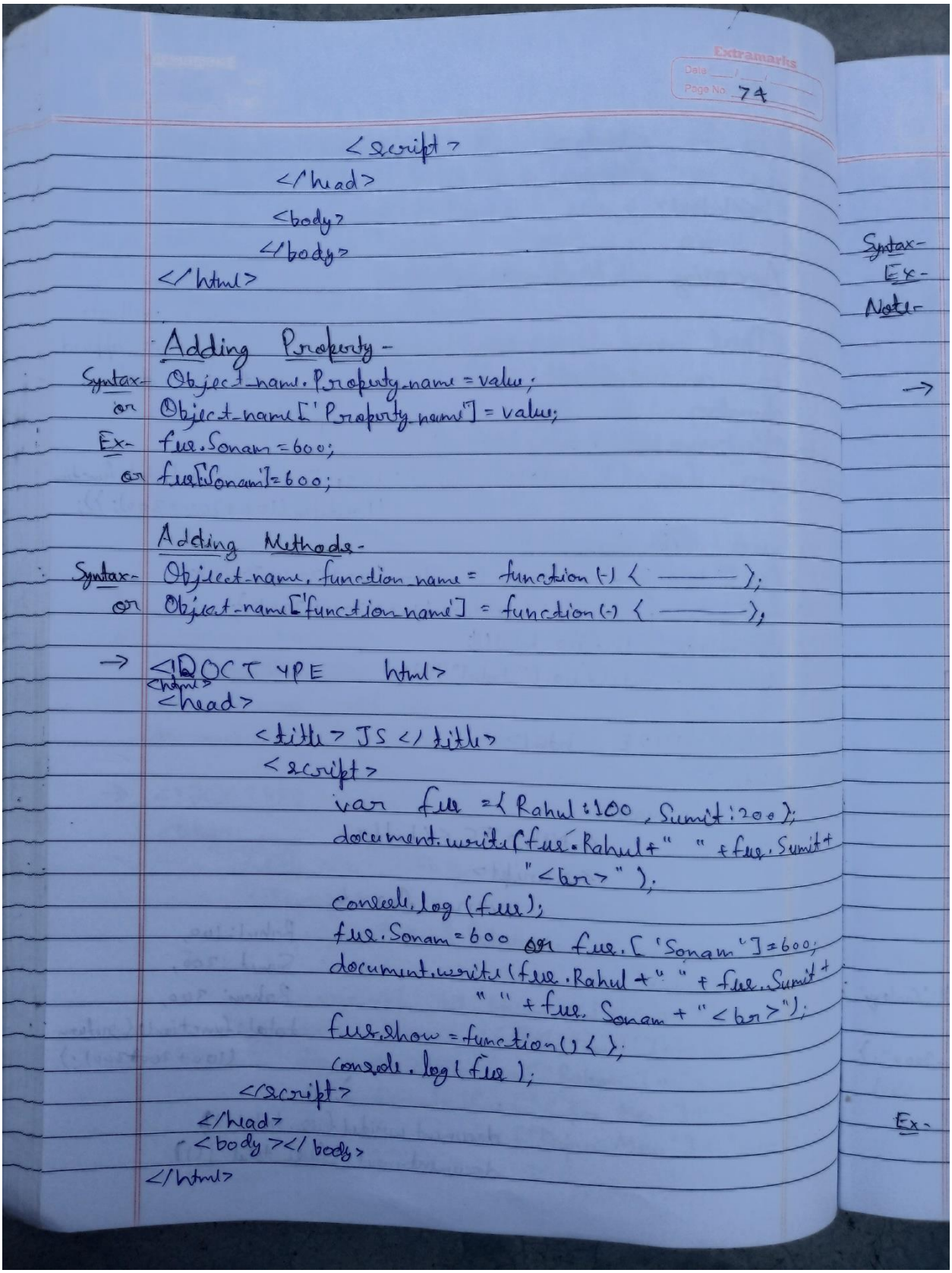
Accessing Methods - Object members that are functions are called methods. These are accessed with dot operator applied to an object alternative to the dot operator is the array[] operator.

Syntax- Object_name.Method_name();

Ex- (i) var fee = { Rahul:100, Sumit:200, Rohan:300, total: function() { return (100+200+300); } };

(ii) var fee = { };
fee['total'] = function() { return (100+200+300); };
or fee.total = function() { return (100+200+300); };
document.write(fee.total());
or document.write(fee["total"]());

```
<!DOCTYPE html>
<html>
    <head>
        <title> JS </title>
        <script>
            var fee = {
                Rahul: 100,
                Sumit : 200,
                Rohan: 300,
                total: function() { return
                    (100+200+300); }
            };
            document.write(fee.total());
            document.write(fee['total']());
```



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

Syntax-
Ex-
Note-

Adding Property -

Syntax- Object_name.Property_name = value;
or Object_name['Property_name'] = value;
Ex- fun.Sonam = 600;
or fun['Sonam'] = 600;

Adding Methods -

Syntax- Object_name.function_name = function() { ——— };
or Object_name['function_name'] = function() { ——— };

→

```
<!DOCTYPE html>
<html>
<head>
    <title> JS </title>
    <script>
        var fun = { Rahul:100, Sumit:200 };
        document.write(fun.Rahul + " " + fun.Sumit +
                       "<br>");
        console.log(fun);
        fun.Sonam = 600 or fun.['Sonam'] = 600;
        document.write(fun.Rahul + " " + fun.Sumit +
                       " " + fun.Sonam + "<br>");
        fun.show = function() { };
        console.log(fun);
    </script>
</head>
<body></body>
</html>
```

Ex-

Extramarks
Date
Page No 75

Delete Properties - Delete operator is used to delete instance properties.

Syntax- delete Objectname.Propertyname;

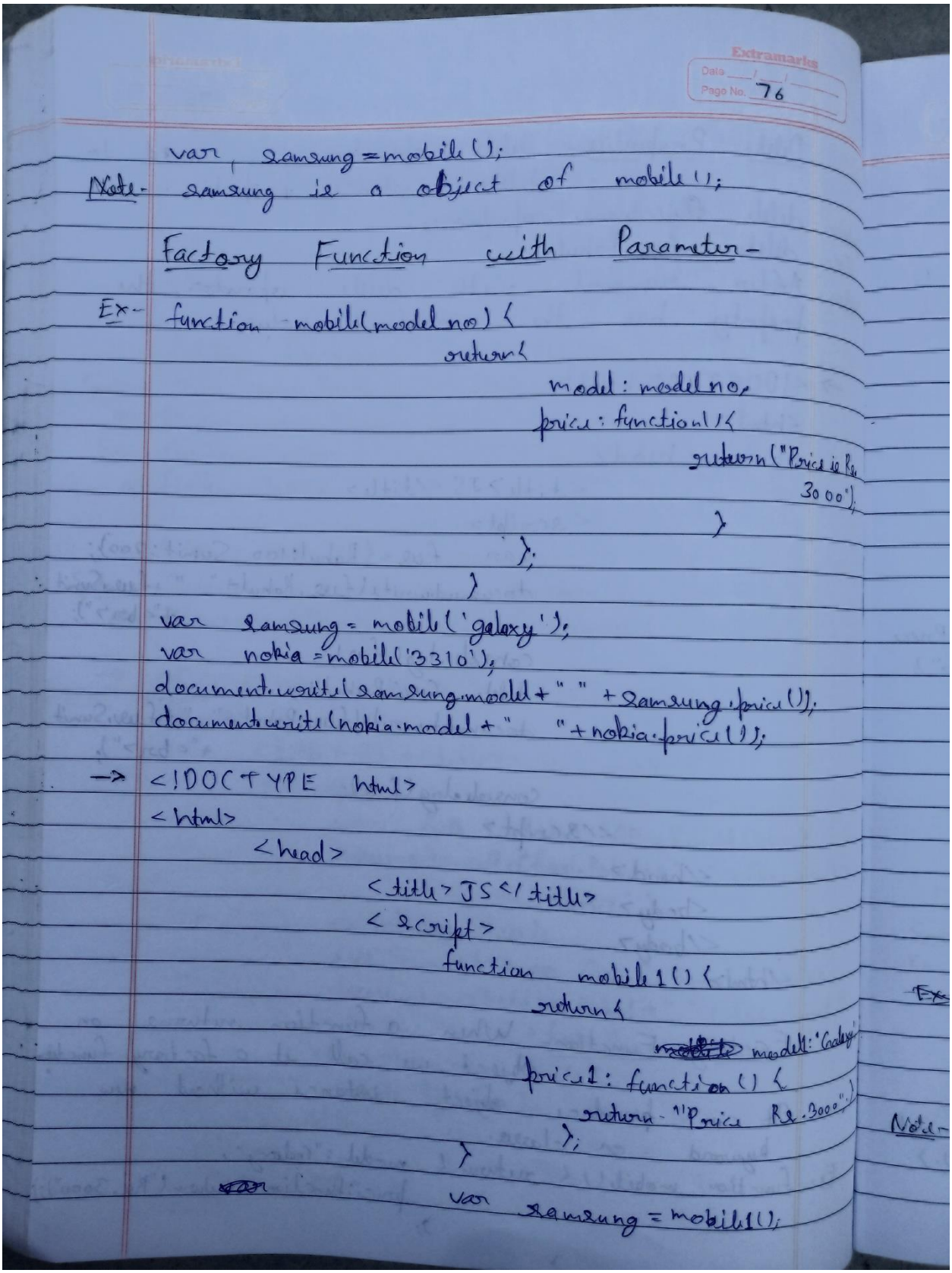
Ex- delete fee.Rahul;

Note- After removal with delete operator, the property has the undefined value.

→
```
<!DOCTYPE html>
<html>
    <head>
        <title>JS</title>
        <script>
            var fee={Rahul:100, Sumit:200};
            document.write(fee.Rahul+" "+fee.Sumit
                +"<br>");
            console.log(fee);
            delete fee.Rahul;
            document.write(fee.Rahul+" "+fee.Sumit
                +"<br>");
            console.log(fee);
        </script>
    </head>
    <body>
    </body>
</html>
```

Factory Function - When a function returns an object, we call it a factory function. It can produce object instance without new keyword or classes.

Ex-
```
function mobile(){ return { model:"Galaxy",
                            price:function(){return("Rs. 3000");}
                          };
}
```



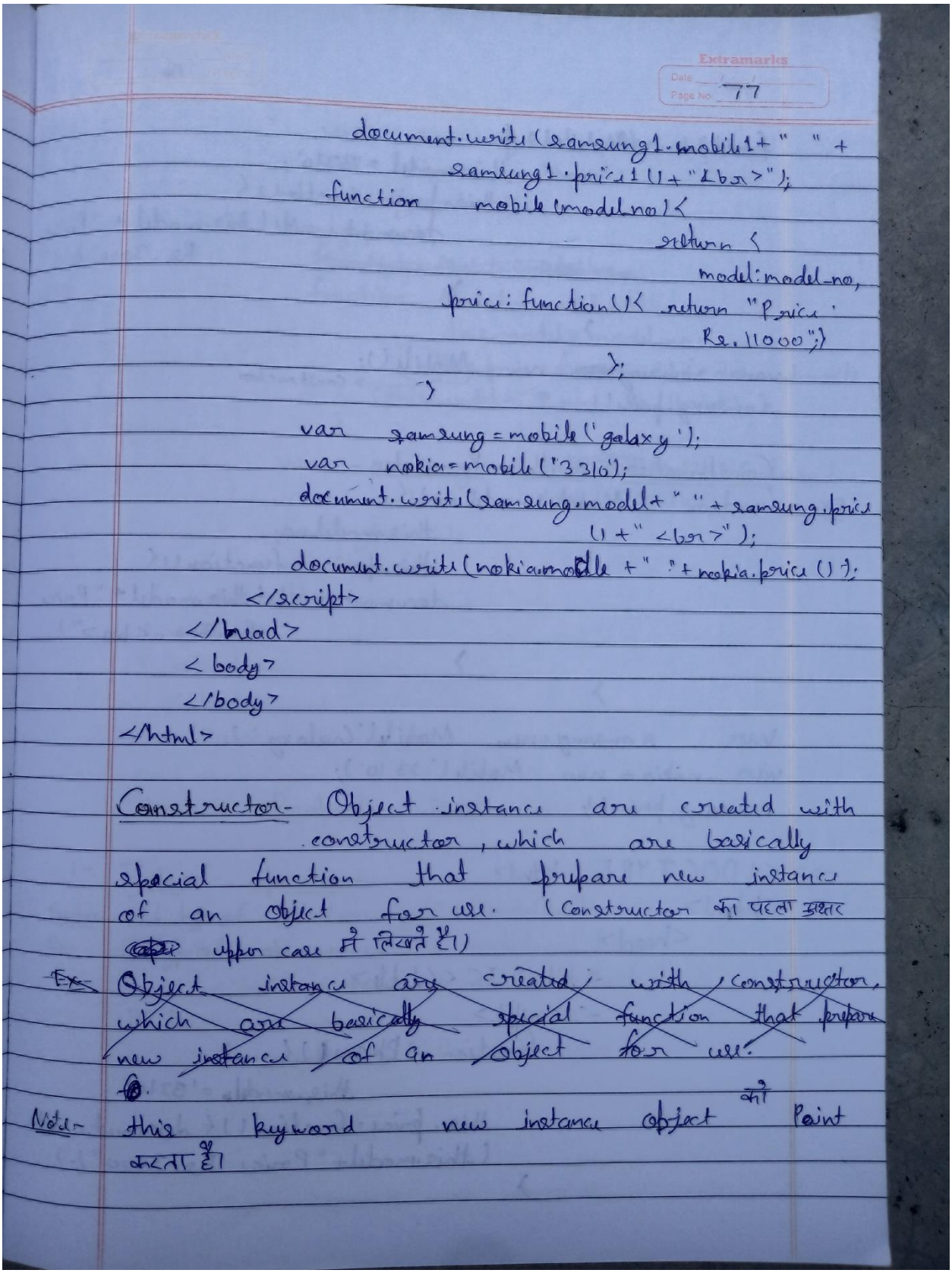
var samsung = mobile();

Note- samsung is a object of mobile();

Factory Function with Parameter-

Ex-
```
function mobile(model no) {
    return {
        model: model no,
        price: function() {
            return ("Price is Rs 3000");
        }
    };
}
var samsung = mobile('galaxy');
var nokia = mobile('3310');
document.write(samsung.model + " " + samsung.price());
document.write(nokia.model + " " + nokia.price());
```

→
```
<!DOCTYPE html>
<html>
    <head>
        <title> JS </title>
        <script>
            function mobile1() {
                return {
                    model: 'Galaxy'
                    price1: function() {
                        return "Price Rs. 3000";
                    }
                }
            var samsung = mobile1();
```

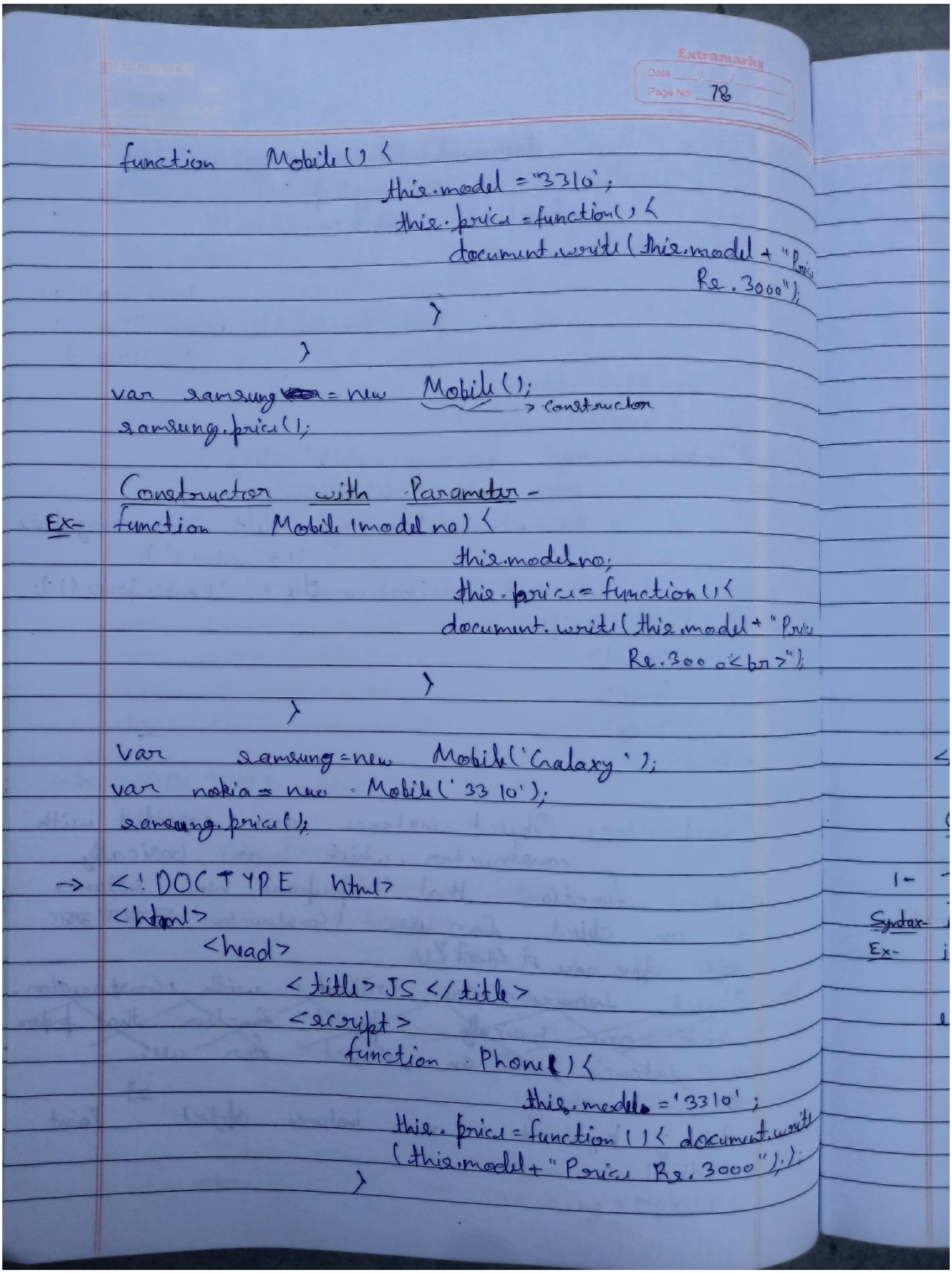


```
document.write(samsung1.mobile1+" "+
    samsung1.price1()+"<br>");
function mobile(model_no){
    return {
        model:model_no,
        price:function(){return "Price:
            Rs.11000";}
    };
}
var samsung=mobile('galaxy');
var nokia=mobile('3310');
document.write(samsung.model+" "+samsung.price
    ()+"<br>");
document.write(nokia.mobile+" "+nokia.price());
</script>
</head>
<body>
</body>
</html>
```

Constructor- Object instance are created with constructor, which are basically special function that prepare new instance of an object for use. (Constructor का पहला अक्षर upper case में लिखते हैं)

Ex- Object instance are created with constructor, which are basically special function that prepare new instance of an object for use.

Note- this keyword new instance object को Point करता है।



```
function Mobile() {
        this.model = '3310';
        this.price = function() {
            document.write(this.model + "Price
                Rs. 3000");
        }
    }
var samsung = new Mobile();
samsung.price();
```

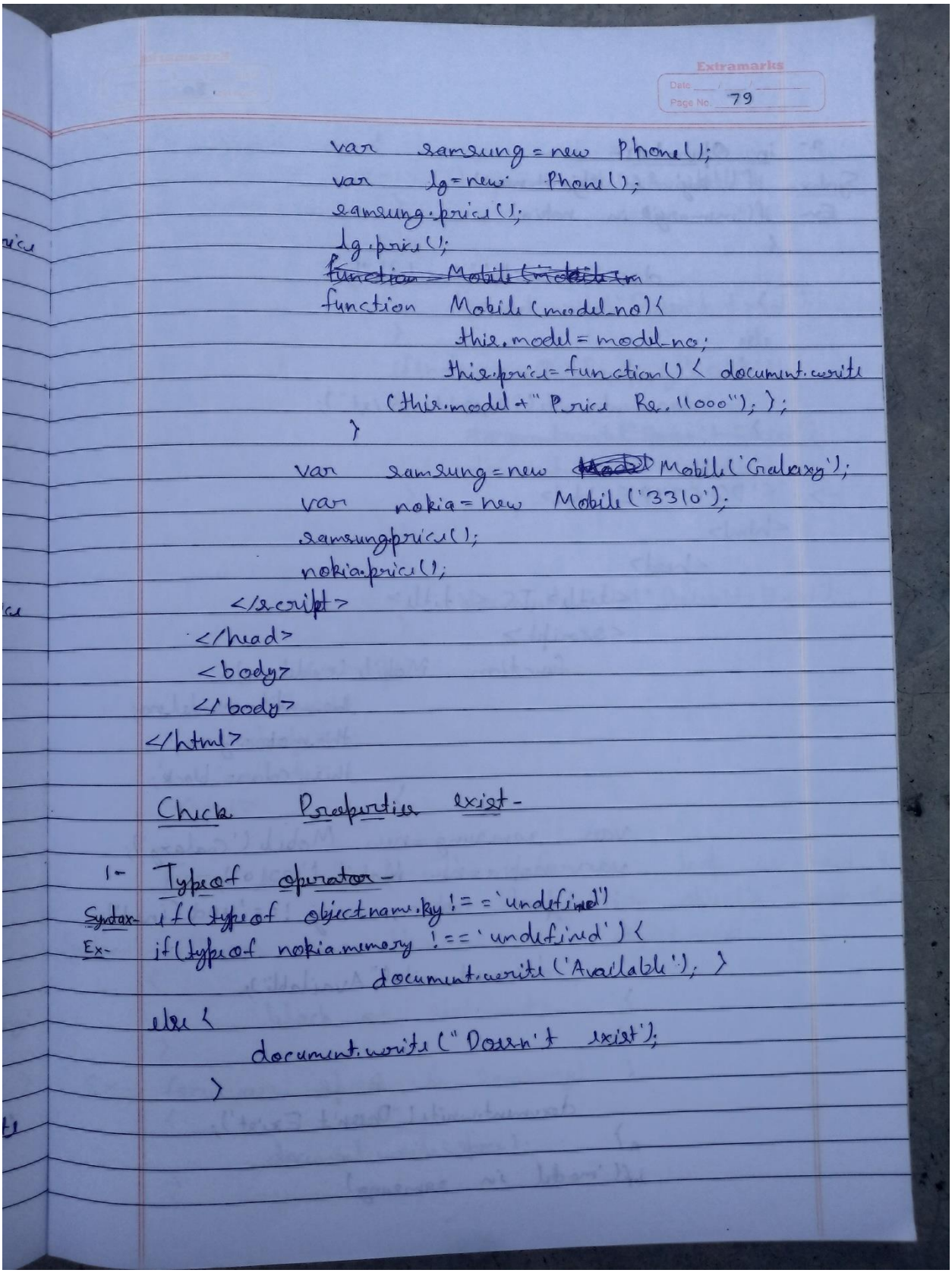
Constructor with Parameter -

Ex-

```
function Mobile(model no) {
        this.model_no;
        this.price = function() {
        document.write(this.model + "Price
            Rs. 3000 <br>");
        }
    }
var samsung = new Mobile('Galaxy');
var nokia = new Mobile('3310');
samsung.price();
```

→

```
<!DOCTYPE html>
<html>
    <head>
        <title> JS </title>
        <script>
            function Phone() {
                this.model = '3310';
            this.price = function() { document.write
            (this.model + "Price Rs. 3000");};
            }
```



```
var samsung = new Phone();
var lg = new Phone();
samsung.price();
lg.price();
function Mobile(model_no){
    this.model = model_no;
    this.price = function(){ document.write
(this.model + " Price Rs. 11000"); };
}
var samsung = new Mobile('Galaxy');
var nokia = new Mobile('3310');
samsungprice();
nokia.price();
</script>
</head>
<body>
</body>
</html>
```

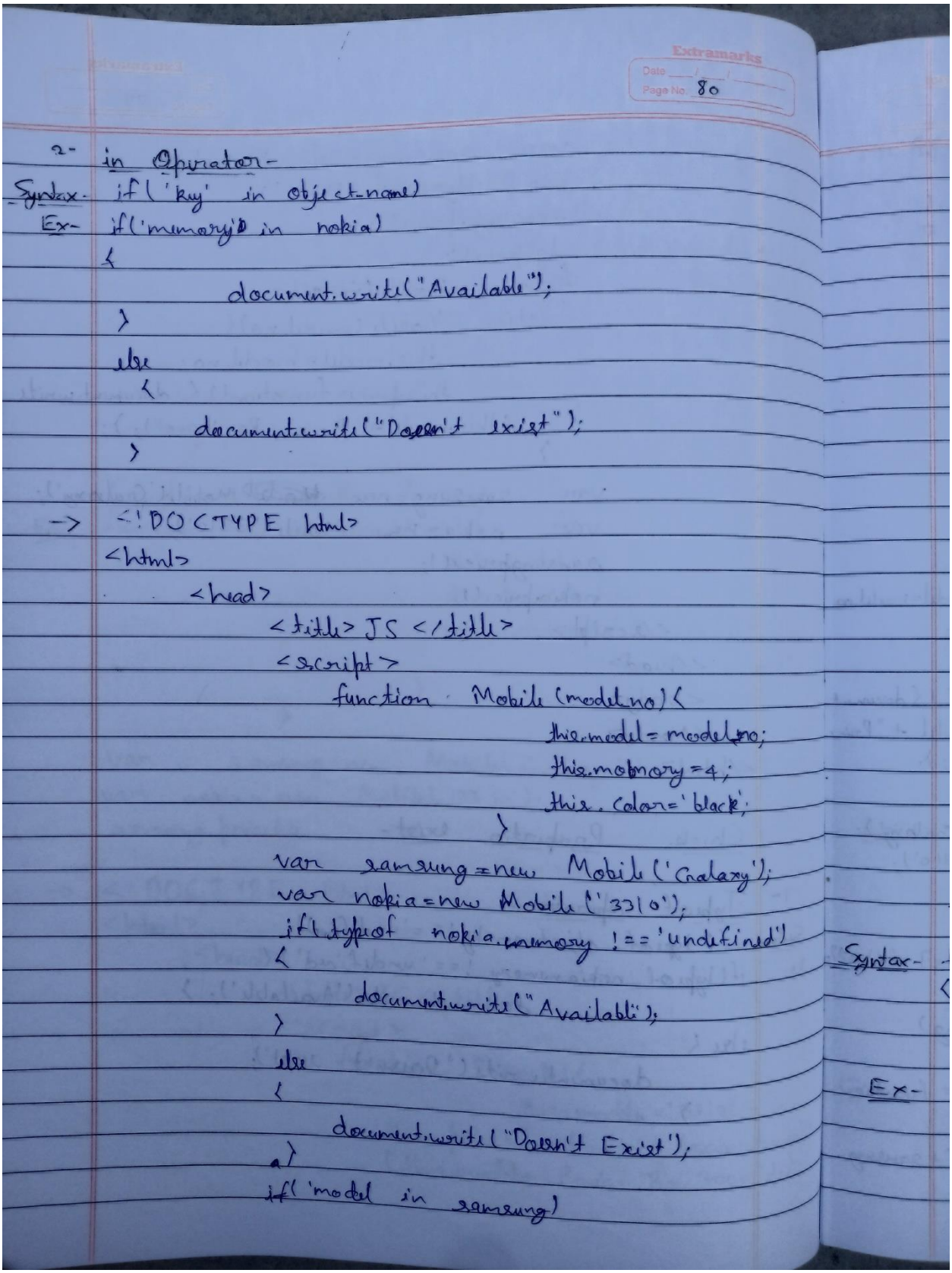
Check Properties exist -

1- Typeof operator -

Syntax- if(typeof objectname.key !== 'undefined')

Ex- if(typeof nokia.memory !== 'undefind'){
document.write('Available'); }

else {
document.write("Doesn't exist');
}



2- in Operator-

Syntax- if('key' in object-name)

Ex- if('memory' in nokia)

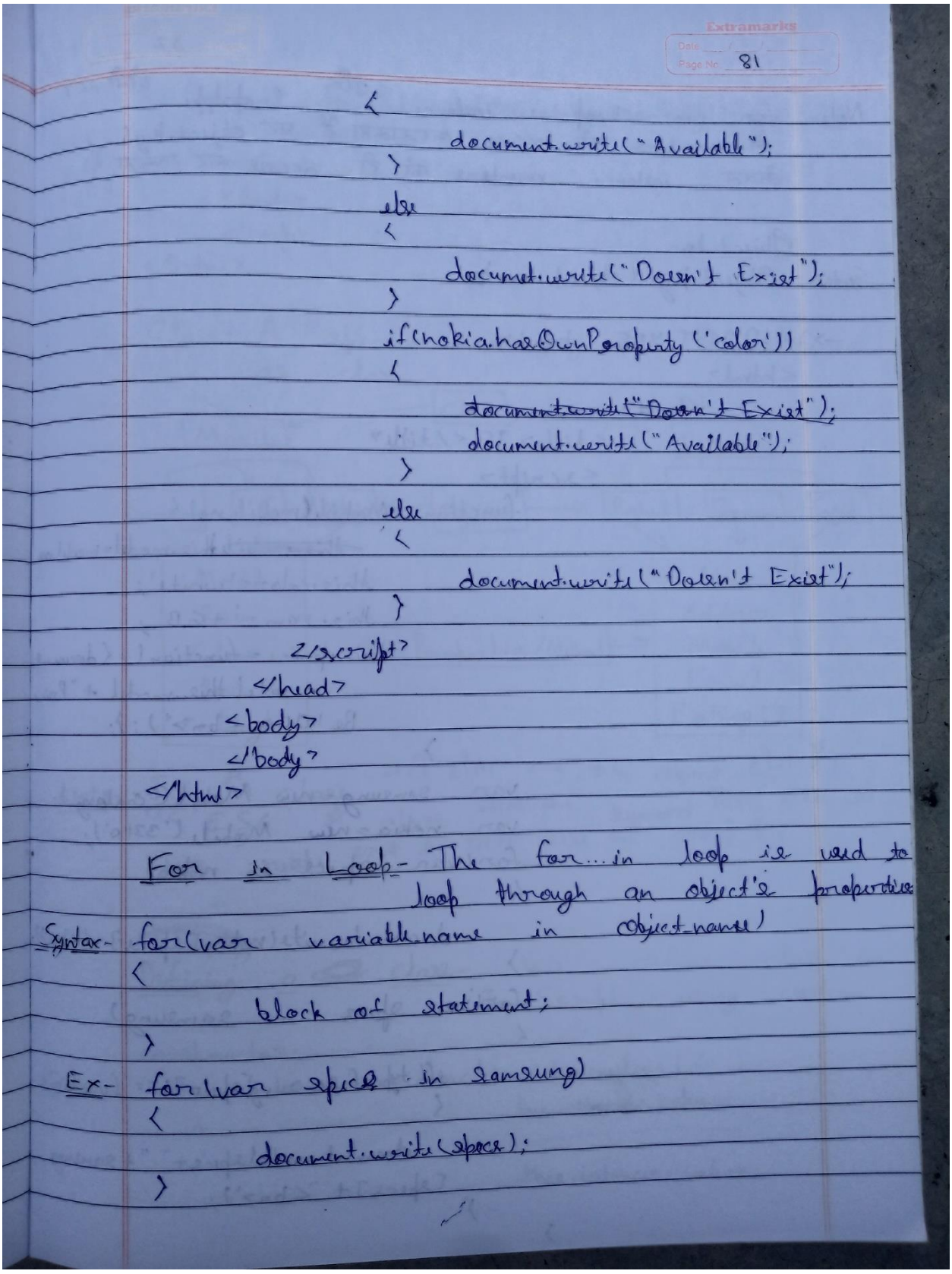
```
{
        document.write("Available");
}
else
{
    document.write("Doesn't exist");
}
```

->

```
<!DOCTYPE html>
<html>
    <head>
        <title> JS </title>
        <script>
            function Mobile(model_no){
                this.model = model_no;
                this.memory = 4;
                this.color = 'black';
            }
            var samsung = new Mobile('Galaxy');
            var nokia = new Mobile('3310');
            if(typeof nokia.memory !== 'undefined')
            {
                document.write("Available");
            }
            else
            {
                document.write("Doesn't Exist");
            }
            if('model in samsung)
```

Syntax-

Ex-

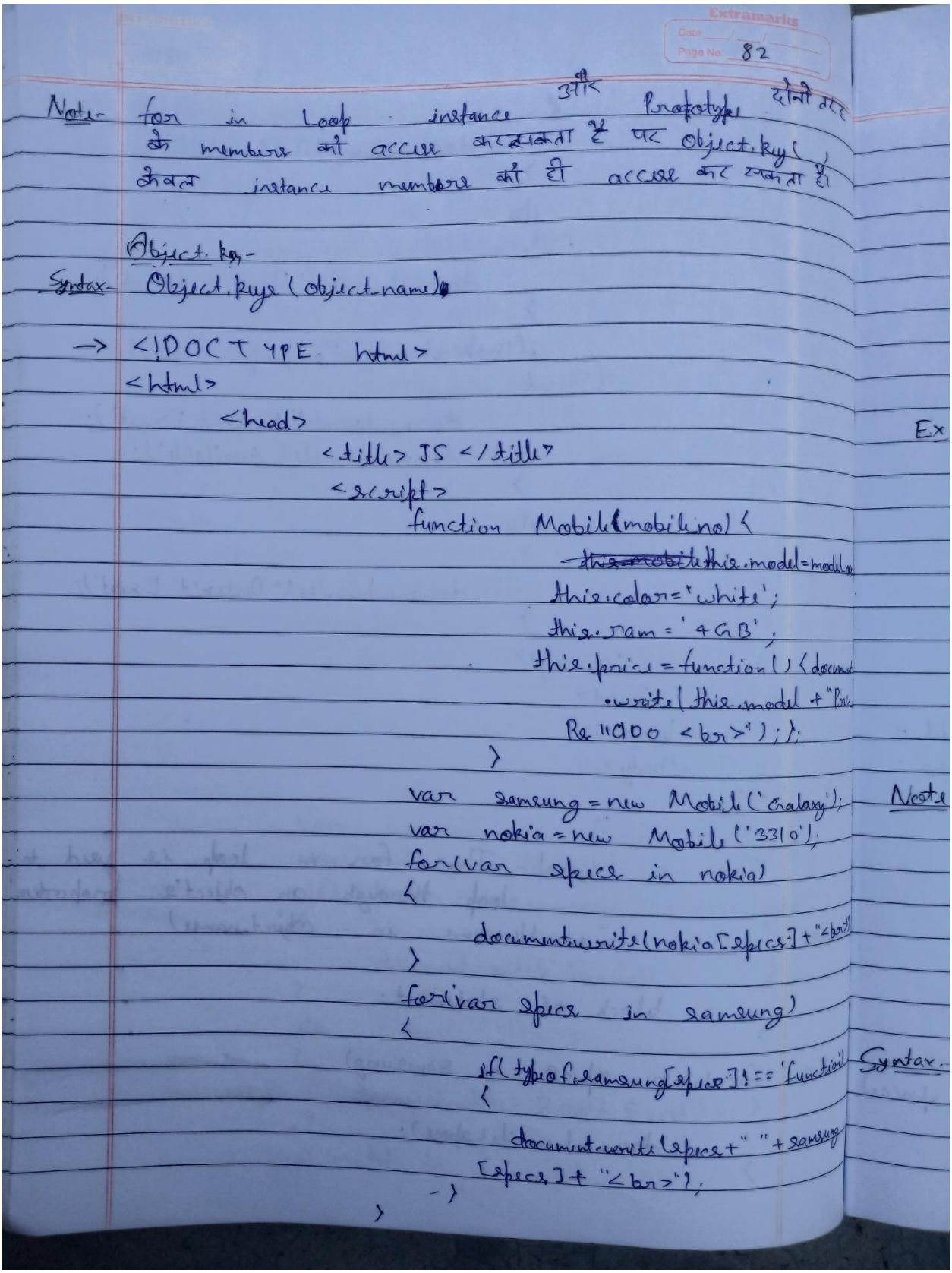


```
        {
            document.write("Available");
        }
        else
        {
            document.write("Doesn't Exist");
        }
        if(nokia.hasOwnProperty('color'))
        {
            document.write("Doesn't Exist");
            document.write("Available");
        }
        else
        {
            document.write("Doesn't Exist");
        }
    </script>
</head>
<body>
</body>
</html>
```

For in Loop- The for...in loop is used to loop through an object's properties

Syntax-
```
for(var variable_name in object_name)
{
    block of statement;
}
```

Ex-
```
for(var specs in samsung)
{
    document.write(specs);
}
```



Extramarks
Date
Page No. 82

Note:- for in Loop instance और Prototype दोनों के members को access कर सकता है पर Object.keys( ) केवल instance members को ही access कर पाता है

Object.keys -

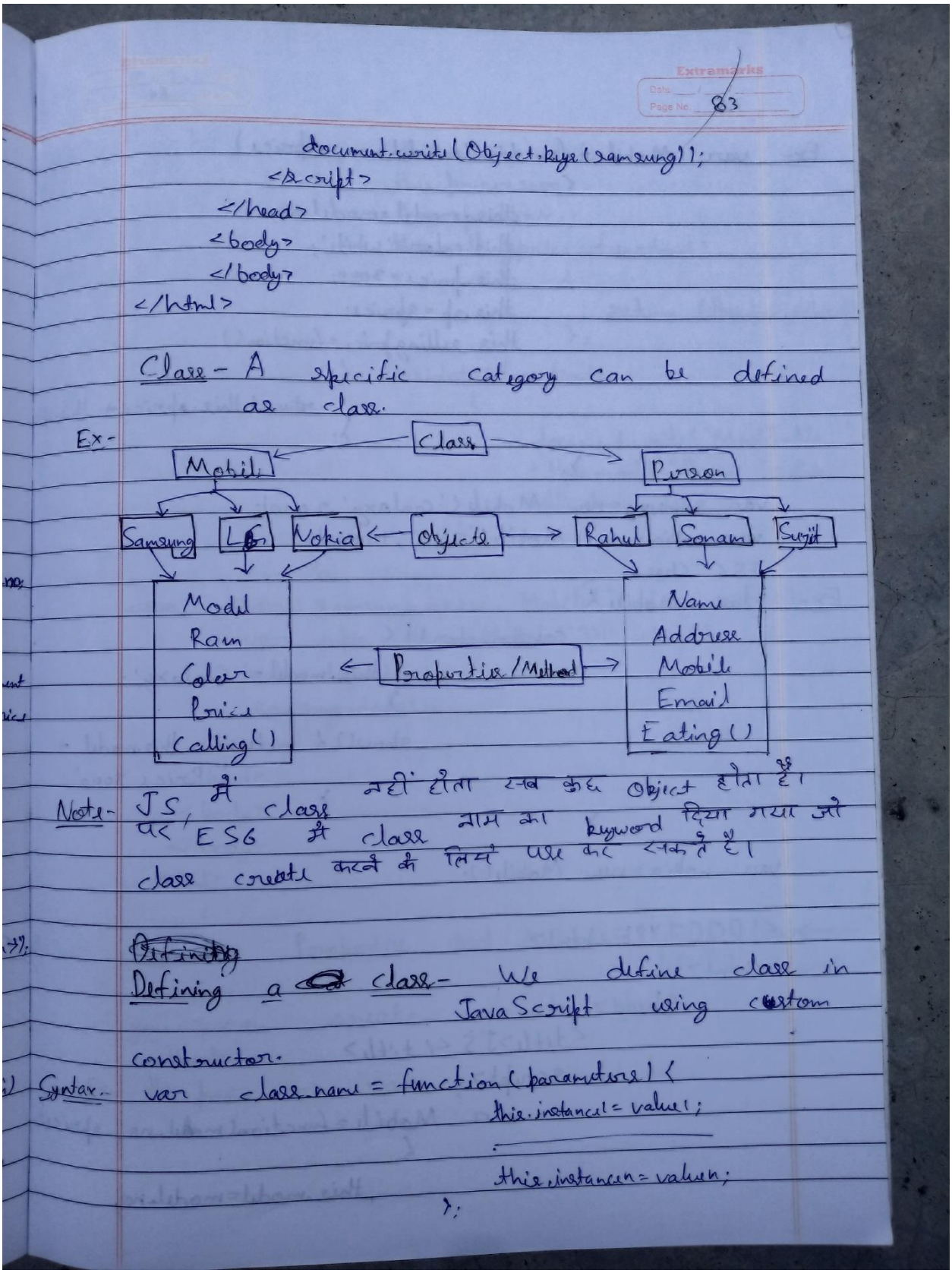
Syntax- Object.keys(object_name)

```
-> <!DOCTYPE html>
<html>
    <head>
        <title> JS </title>
        <script>
            function Mobile(mobile_no){
                this.model=model_no
                this.color='white';
                this.ram='4GB';
                this.price=function(){document
                .write(this.model+"Price
                Rs 1000 <br>");};
            }
            var samsung=new Mobile('Galaxy');
            var nokia=new Mobile('3310');
            for(var specs in nokia)
            {
                document.write(nokia[specs]+"<br>");
            }
            for(var specs in samsung)
            {
                if(typeof samsung[specs]!=='function')
                {
                    document.write(specs+" "+samsung
                [specs]+"<br>");
                }
            }
```

Ex

Note

Syntax



Extramarks
Date
Page No. 83

```
document.write(Object.keys(samsung));
</script>
</head>
<body>
</body>
</html>
```

Class - A specific category can be defined as class.

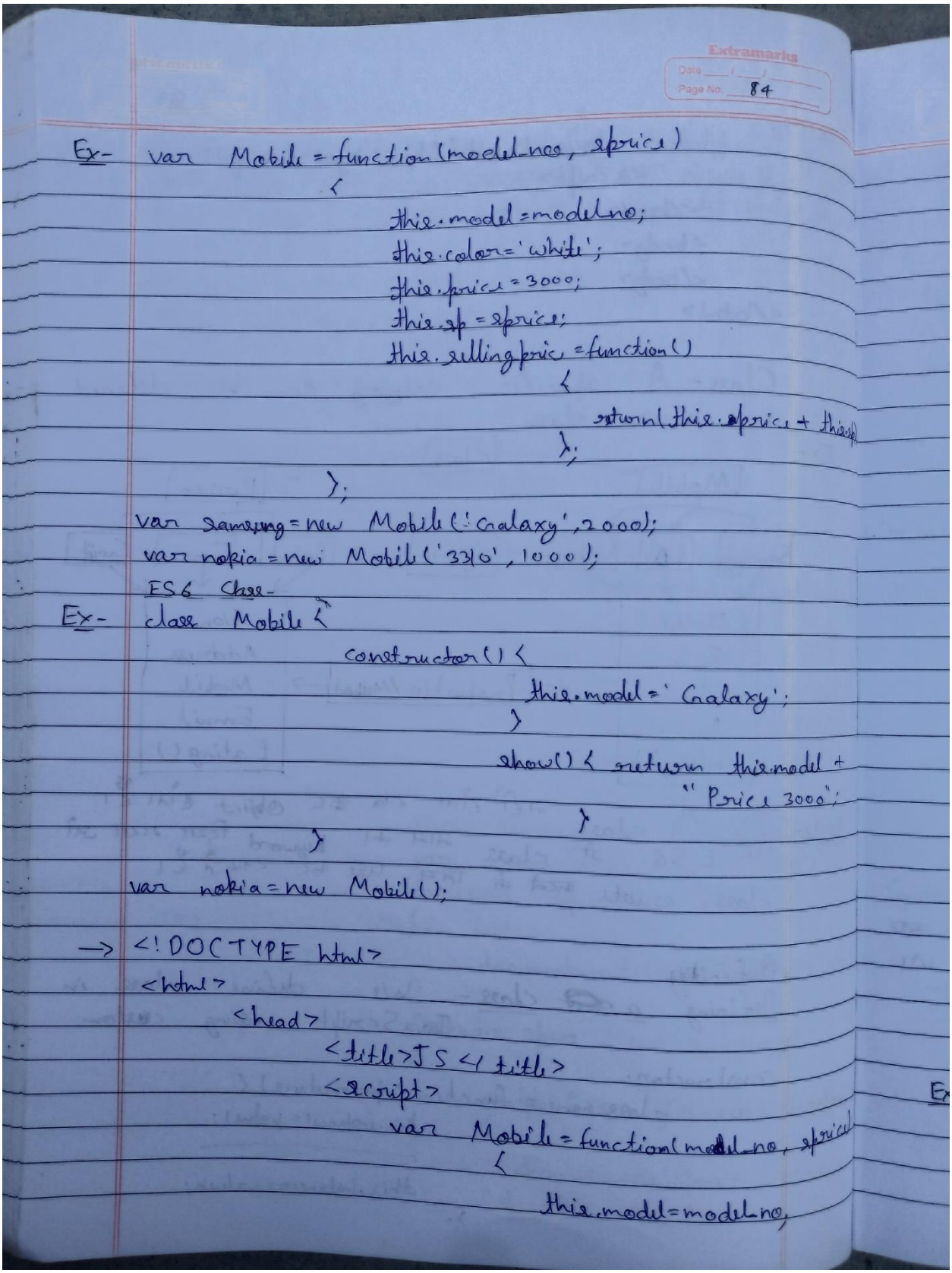
Ex-

Note- JS में class नहीं होता सब कुछ Object होता है। पर ES6 में class नाम का keyword दिया गया जो class create करने के लिये use कर सकते है।

Defining a class - We define class in JavaScript using custom constructor.

Syntax.

```
var class_name = function(parameters){
    this.instance1 = value1;
    .
    .
    this.instancen = valuen;
};
```



Extramarks
Date ___/___/___
Page No. 84

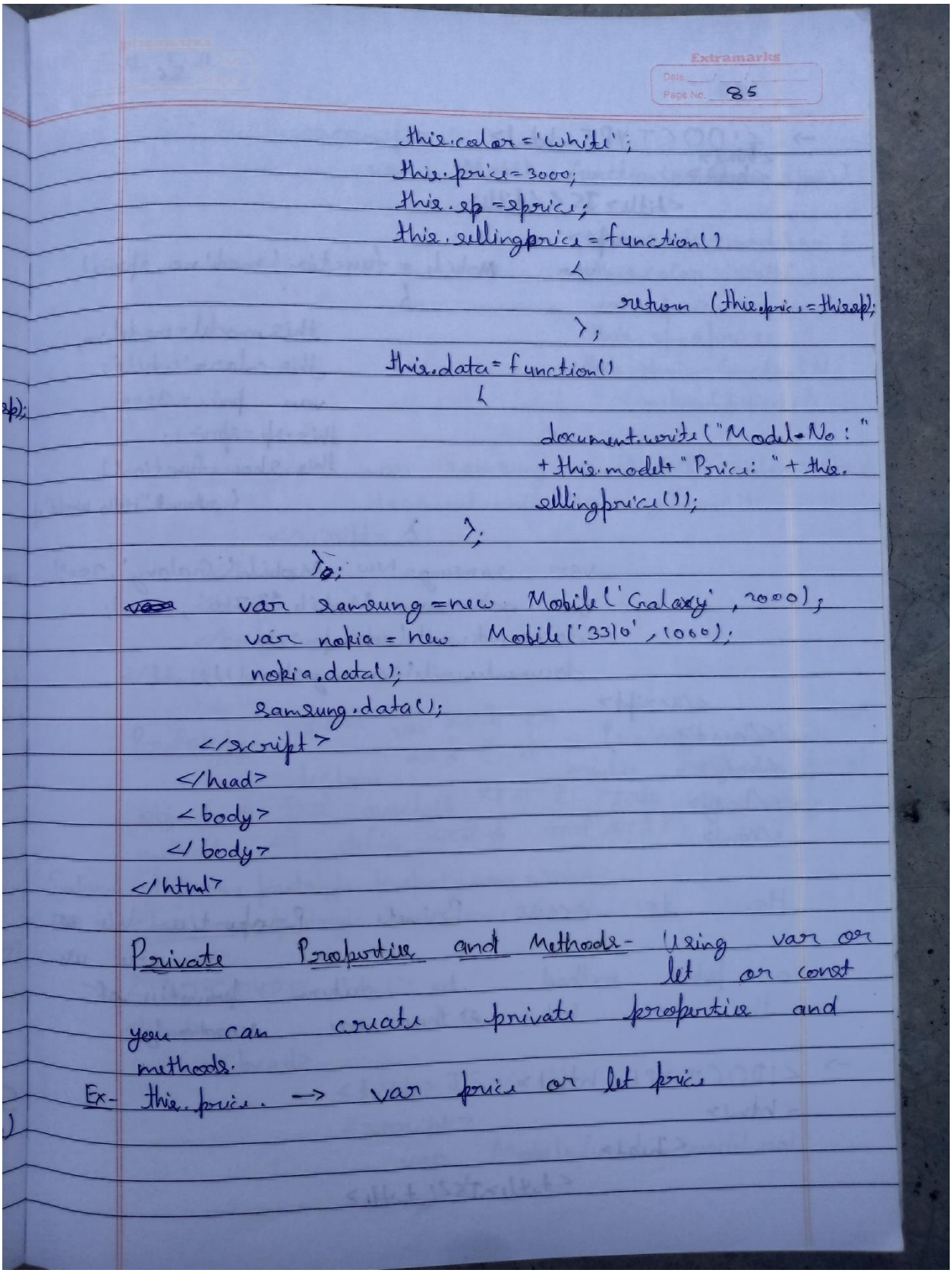
Ex- var Mobile = function (model_no, sprice)
{
this.model = model_no;
this.color = 'white';
this.price = 3000;
this.sp = sprice;
this.sellingprice = function()
{
return this.sprice + this.sp
};
};
var samsung = new Mobile ('Galaxy', 2000);
var nokia = new Mobile ('3310', 1000);

ES6 Class-

Ex- class Mobile {
constructor() {
this.model = 'Galaxy';
}
show() { return this.model +
" Price 3000";
}
}
var nokia = new Mobile();

→ <!DOCTYPE html>
<html>
<head>
<title>JS</title>
<script>
var Mobile = function(model_no, sprice)
{
this.model=model_no;

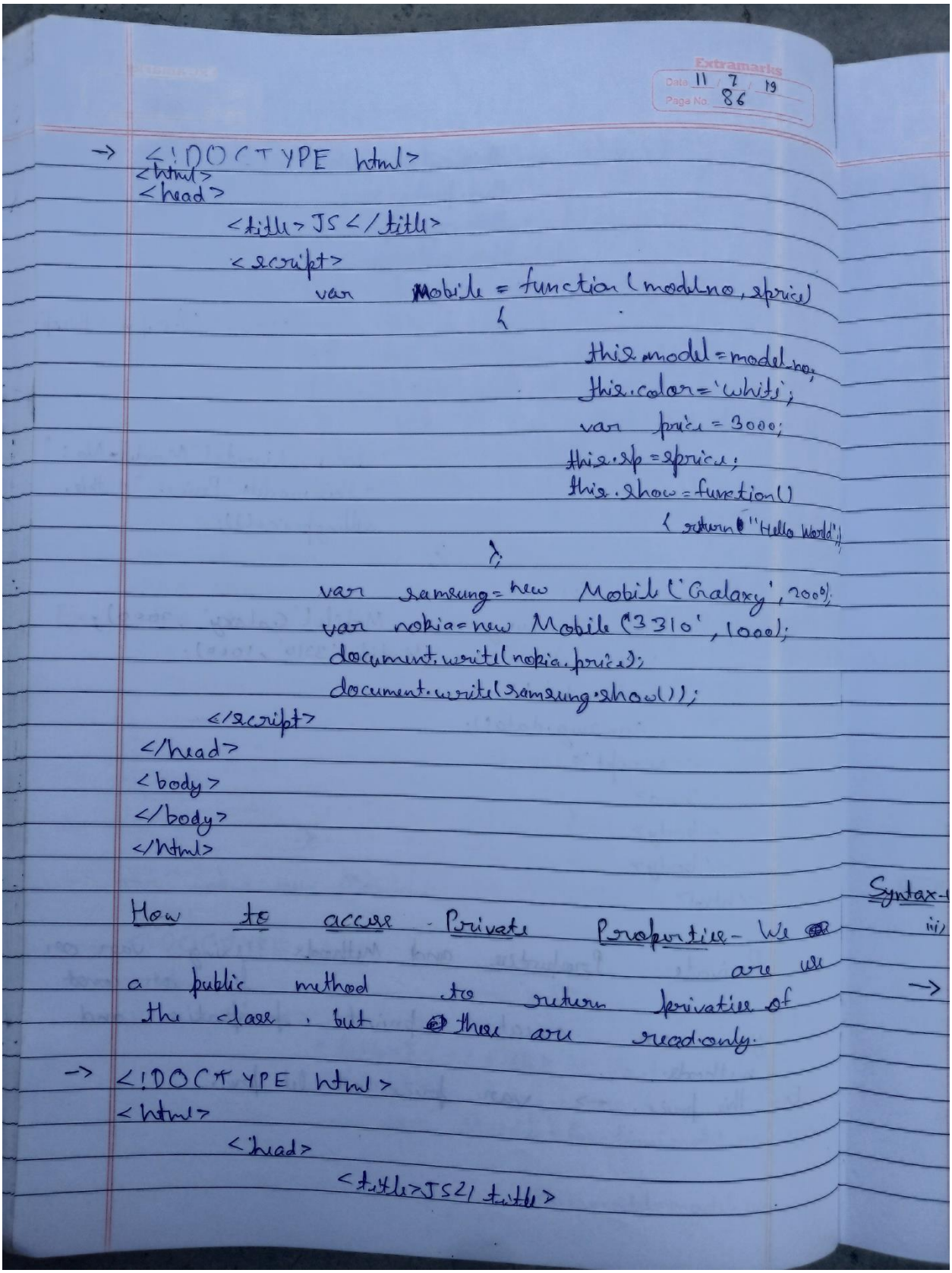
Ex



```
        this.color = 'white';
        this.price = 3000;
        this.sp = sprice;
        this.sellingprice = function()
                {
                    return (this.price = this.sp);
                };
        this.data = function()
                {
                    document.write("Model No: "
                    + this.model + "Price: " + this.
                    sellingprice());
                };
    };
    var samsung = new Mobile('Galaxy', 2000);
    var nokia = new Mobile('3310', 1000);
    nokia.data();
    samsung.data();
</script>
</head>
<body>
</body>
</html>
```

Private Properties and Methods - Using var or let or const you can create private properties and methods.

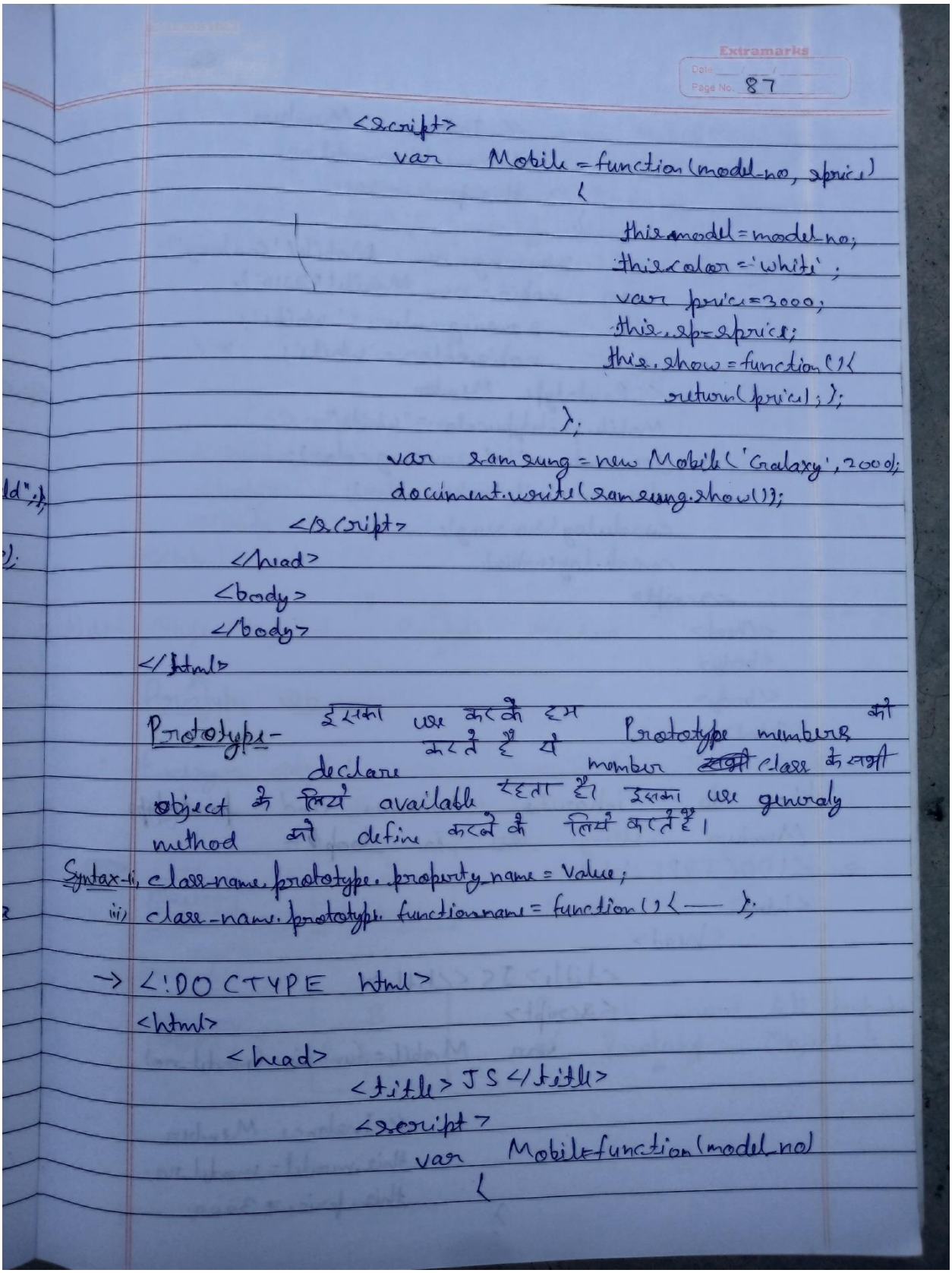
Ex- this.price. --> var price or let price



```
<!DOCTYPE html>
<html>
<head>
    <title>JS</title>
    <script>
        var Mobile = function(modelno, sprice)
        {
            this.model = model_no;
            this.color = 'white';
            var price = 3000;
            this.sp = sprice;
            this.show = function()
            { return "Hello World"; }
        };
        var samsung = new Mobile('Galaxy', 2000);
        var nokia = new Mobile('3310', 1000);
        document.write(nokia.price);
        document.write(samsung.show());
    </script>
</head>
<body>
</body>
</html>
```

How to access Private Properties - We are use a public method to return privaties of the class. but these are read only.

```
<!DOCTYPE html>
<html>
    <head>
        <title>JS2</title>
```

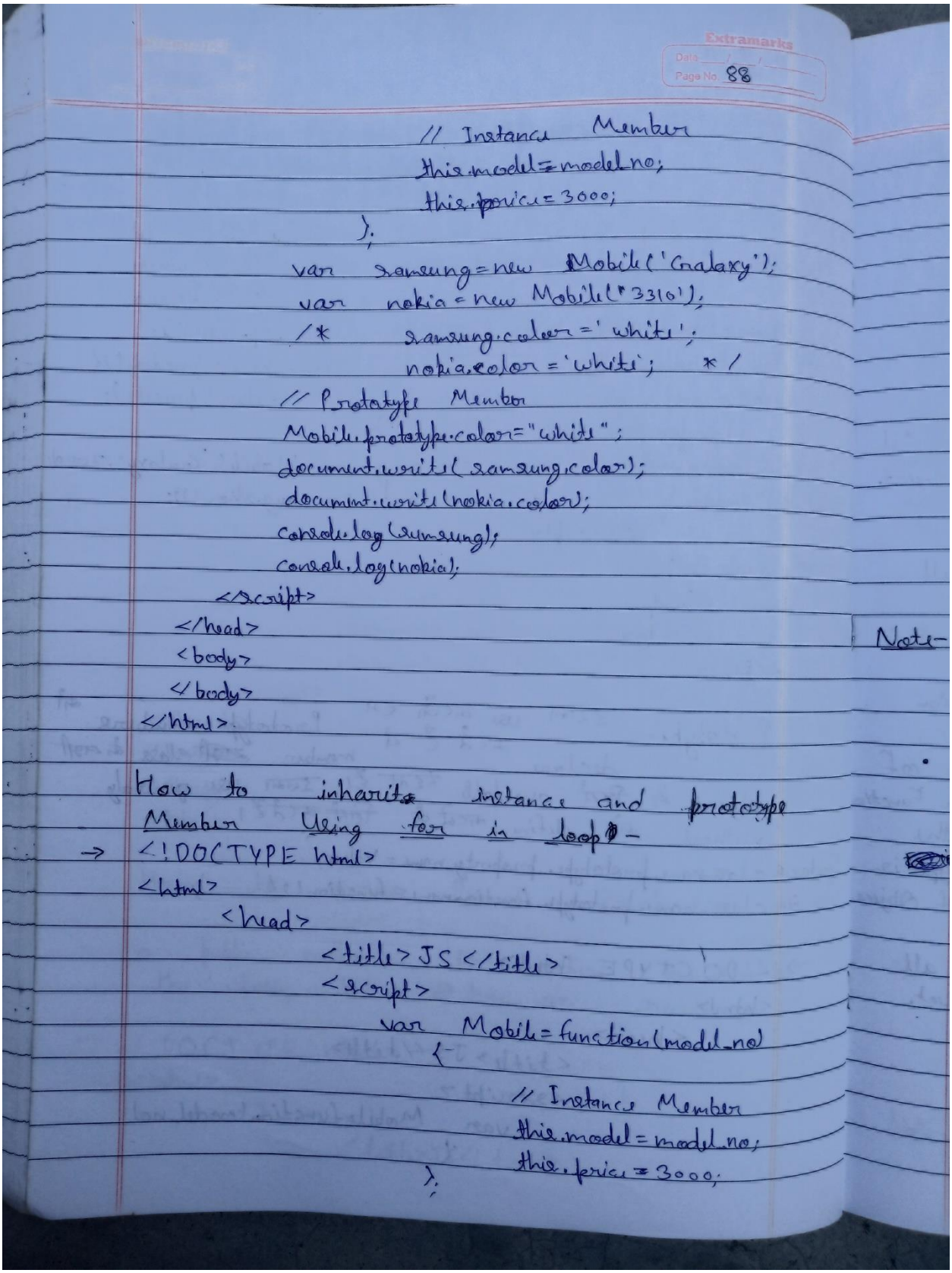


```
<script>
    var Mobile = function(model_no, sprice)
                {
                    this.model = model_no;
                    this.color = 'white';
                    var price = 3000;
                    this.sp = sprice;
                    this.show = function(){
                        return(price); };
                };
    var samsung = new Mobile('Galaxy', 2000);
    document.write(samsung.show());
</script>
</head>
<body>
</body>
</html>
```

Prototype:- इसका use करके हम Prototype members को declare करते हैं ये member class के सभी object के लिये available रहता है। इसका use generaly method को define करने के लिये करते हैं।

Syntax- i) class_name.prototype.property_name = value;
ii) class_name.prototype.functionname = function(){ ---- };

```
-> <!DOCTYPE html>
<html>
    <head>
        <title> JS </title>
        <script>
            var Mobile=function(model_no)
                {
```



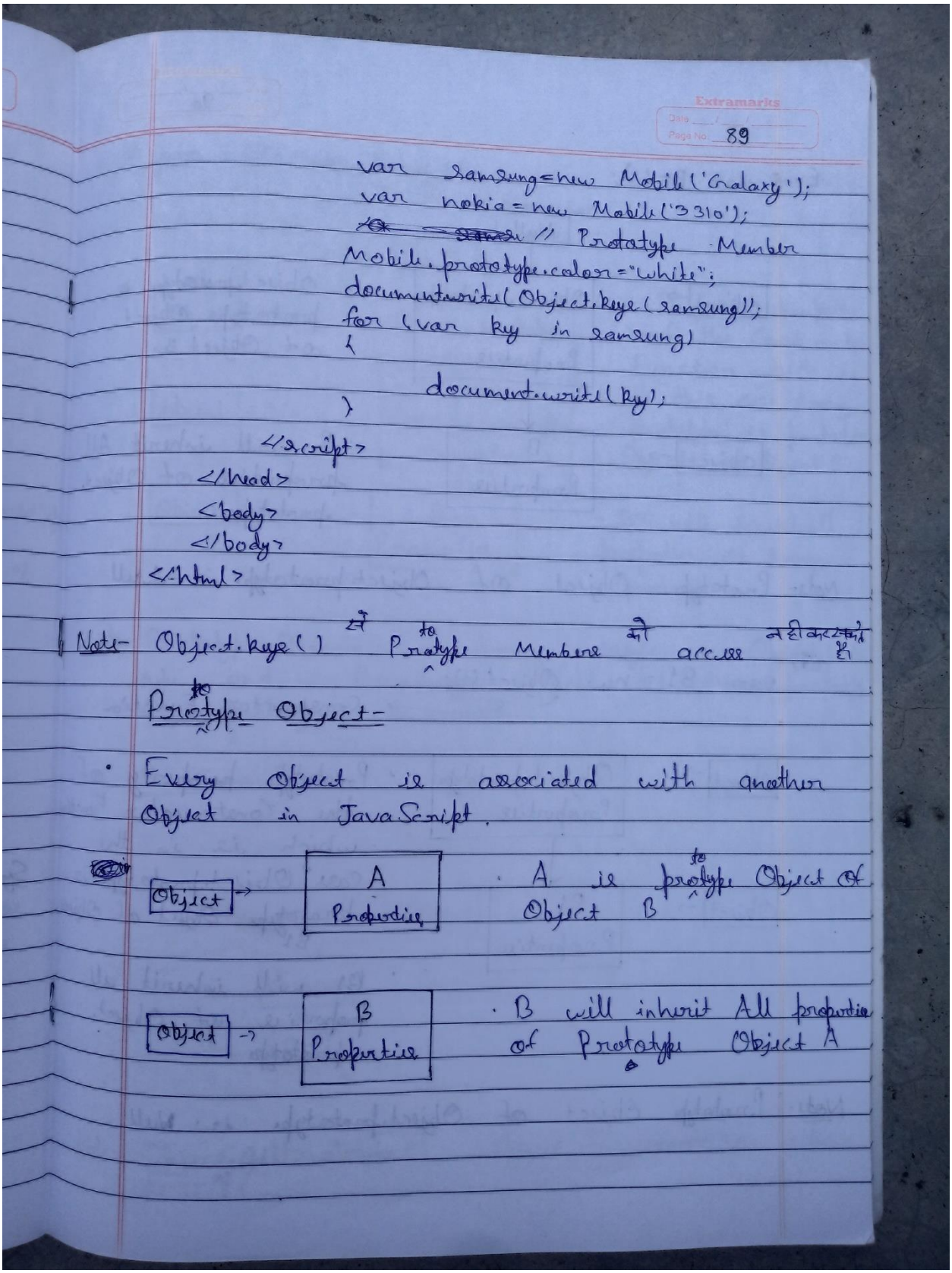
```
        // Instance Member
        this.model = model_no;
        this.price = 3000;
    };
var samsung = new Mobile('Galaxy');
var nokia = new Mobile('3310');
/*  samsung.color = 'white';
    nokia.color = 'white';  */
// Prototype Member
Mobile.prototype.color = "white";
document.write(samsung.color);
document.write(nokia.color);
console.log(samsung);
console.log(nokia);
</script>
</head>
<body>
</body>
</html>
```

Note-

How to inherit instance and prototype Member Using for in loop -

→

```
<!DOCTYPE html>
<html>
    <head>
        <title> JS </title>
        <script>
            var Mobile = function(model_no)
            {
                // Instance Member
                this.model = model_no;
                this.price = 3000;
            };
```



```
var samsung=new Mobile('Galaxy');
var nokia=new Mobile('3310');
// Prototype Member
Mobile.prototype.color="white";
document.write(Object.keys(samsung));
for(var ky in samsung)
{
    document.write(ky);
}
</script>
</head>
<body>
</body>
</html>
```

Note- Object.keys() से Prototype Members को access नहीं कर सकते हैं

Prototype Object-

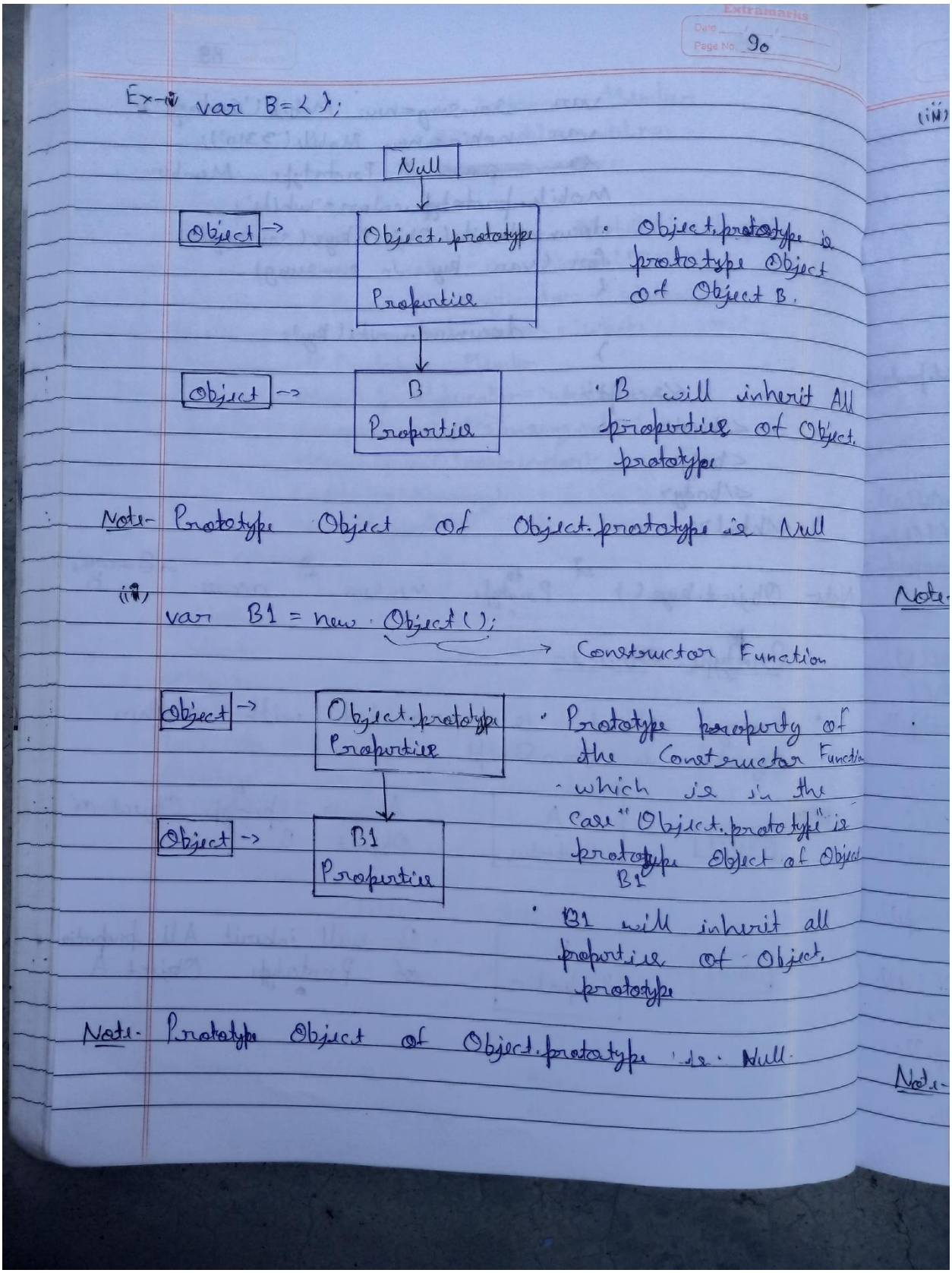
- Every Object is associated with another Object in JavaScript.

Object → [A Properties]

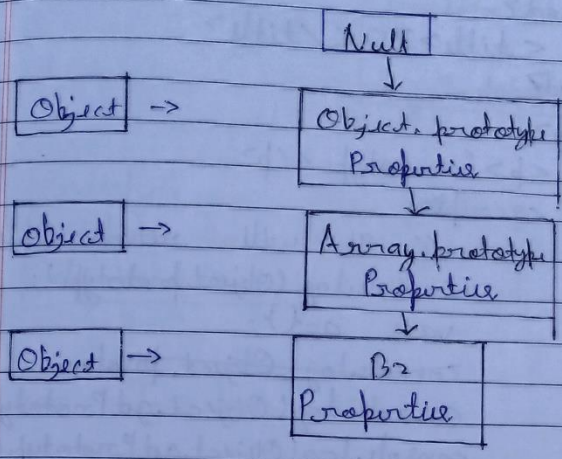
- A is prototype Object of Object B

Object → [B Properties]

- B will inherit All properties of Prototype Object A


Extramarks
Date
Page No. 90
Ex- var B={};
Null
Object →
Object.prototype
Properties
• Object.prototype is prototype Object of Object B.
Object →
B
Properties
• B will inherit All properties of Object. prototype
Note- Prototype Object of Object.prototype is Null
(ii) var B1 = new Object();
Constructor Function
Object →
Object.prototype
Properties
• Prototype property of the Constructor Functio which is in the case "Object.prototype" is prototype Object of Object B1
Object →
B1
Properties
• B1 will inherit all properties of Object. prototype
Note- Prototype Object of Object.prototype is Null.

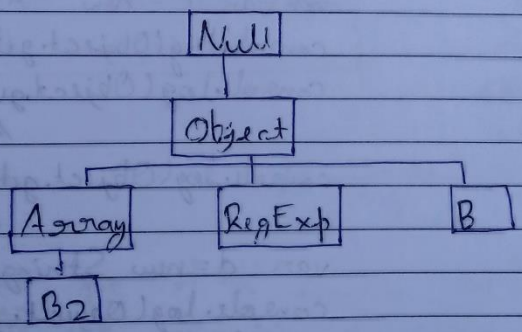
(iii) var B2 = new Array();
constructor



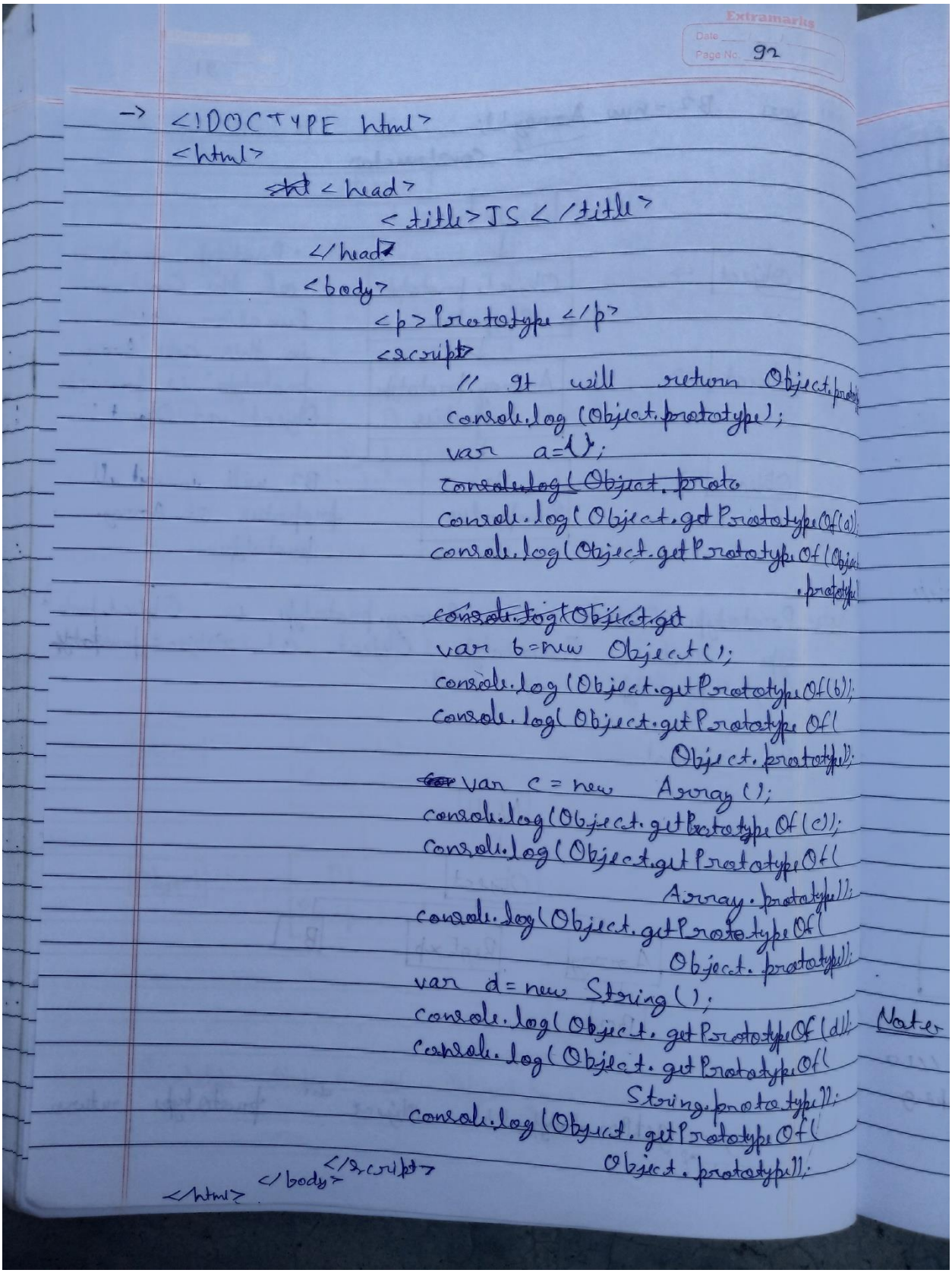
- Prototype property of the Constructor Function which is in this case 'Array-prototype' is prototype Object of Object B2
- B2 will inherit all properties of Array-prototype.

Note- Prototype Object of Array.prototype is Object.prototype and Prototype Object of Object.prototype is null.

In General -

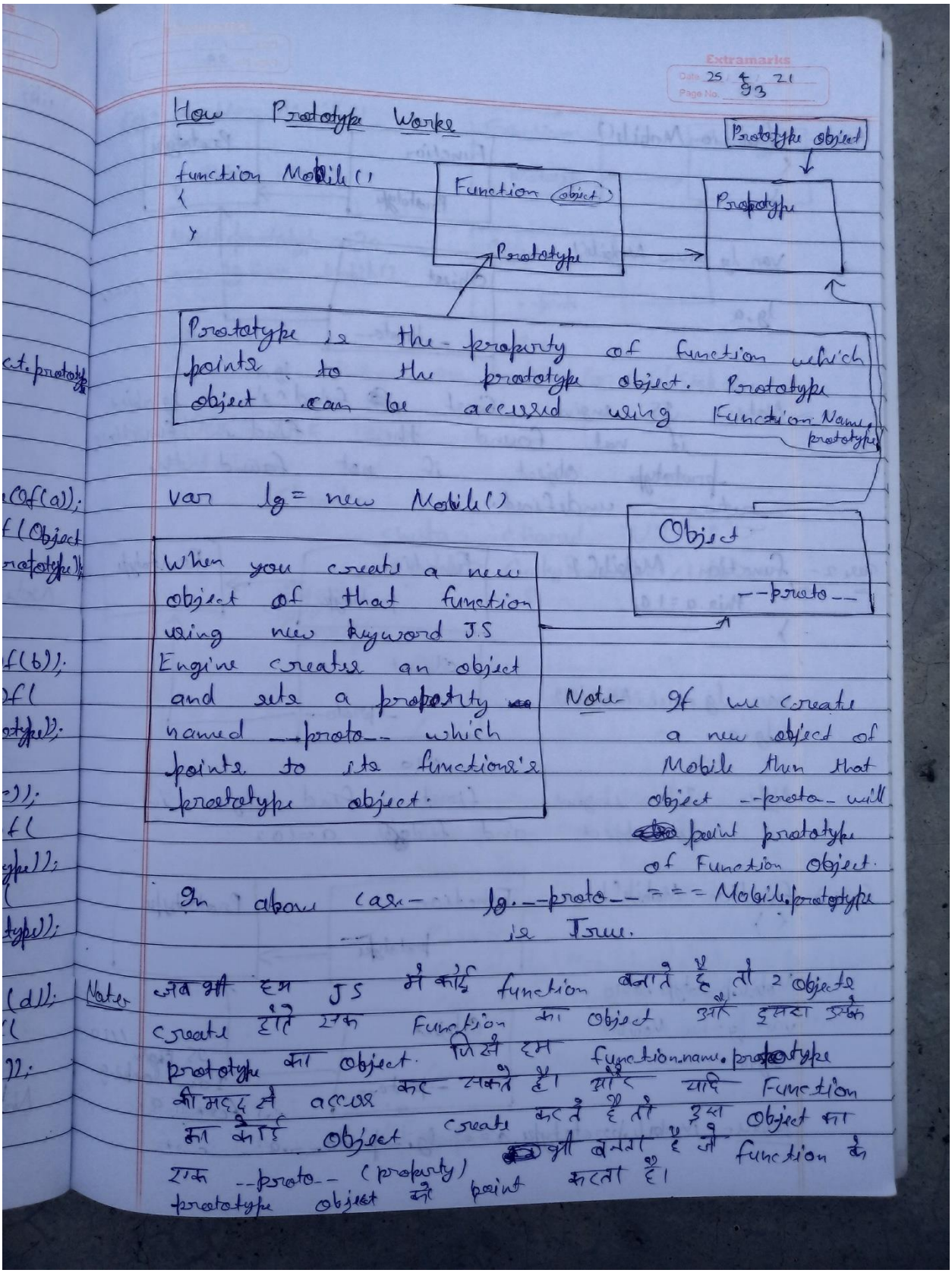


Note- Object.getPrototype() Object की prototype return करता है।



```
<!DOCTYPE html>
<html>
    <head>
        <title>JS</title>
    </head>
    <body>
        <p>Prototype</p>
        <script>
            // It will return Object.prototype
            console.log(Object.prototype);
            var a={};
            console.log(Object.getPrototypeOf(a));
            console.log(Object.getPrototypeOf(Object.prototype));

            var b=new Object();
            console.log(Object.getPrototypeOf(b));
            console.log(Object.getPrototypeOf(Object.prototype));

            var c = new Array();
            console.log(Object.getPrototypeOf(c));
            console.log(Object.getPrototypeOf(Array.prototype));
            console.log(Object.getPrototypeOf(Object.prototype));

            var d=new String();
            console.log(Object.getPrototypeOf(d));
            console.log(Object.getPrototypeOf(String.prototype));
            console.log(Object.getPrototypeOf(Object.prototype));
        </script>
    </body>
</html>
```

Note



How Prototype Works

function Mobile()
{
}

Function (object)
Prototype

Prototype object
Prototype

Prototype is the property of function which points to the prototype object. Prototype object can be accessed using Function Name.prototype

var lg = new Mobile()

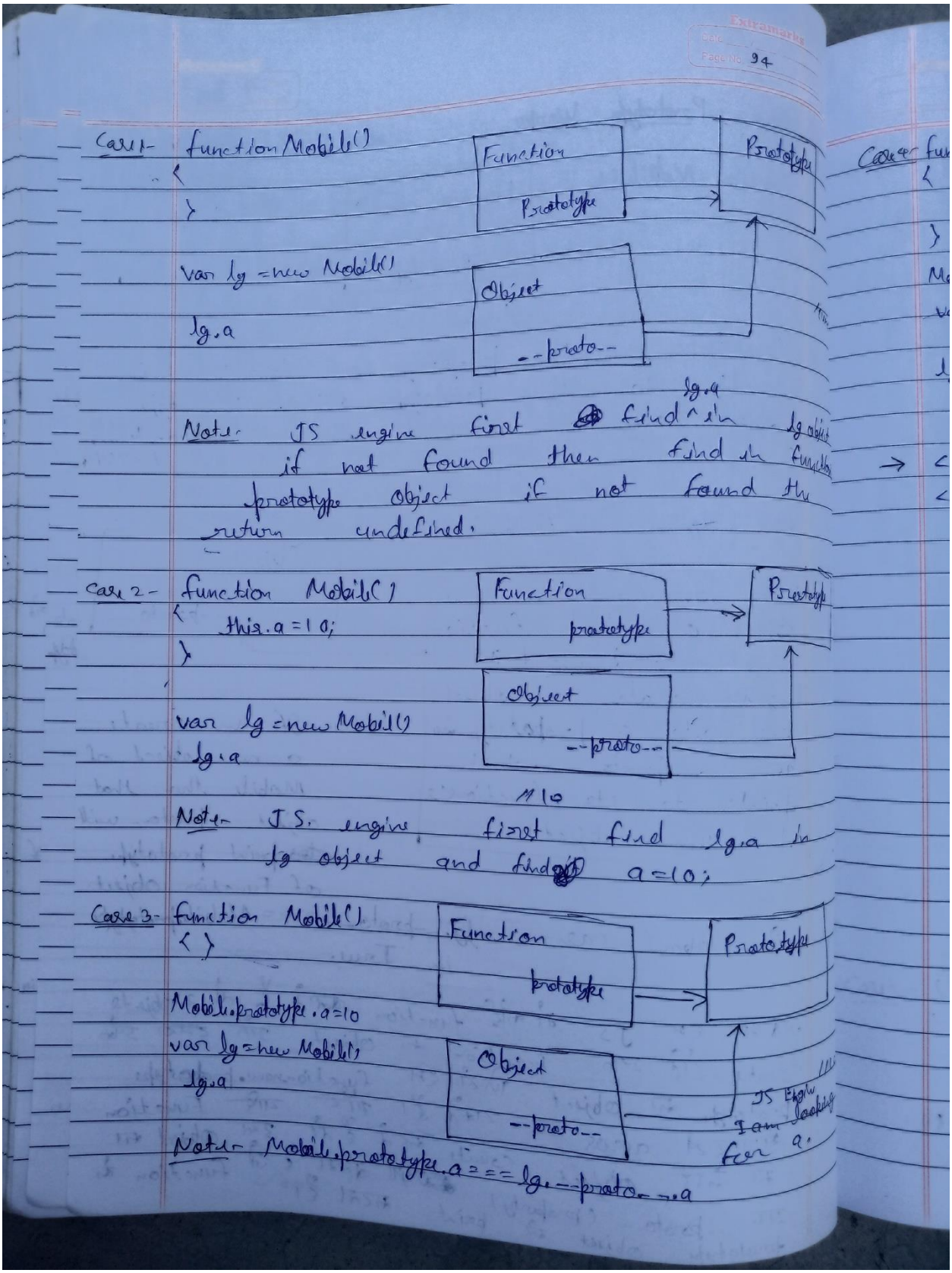
Object
--proto--

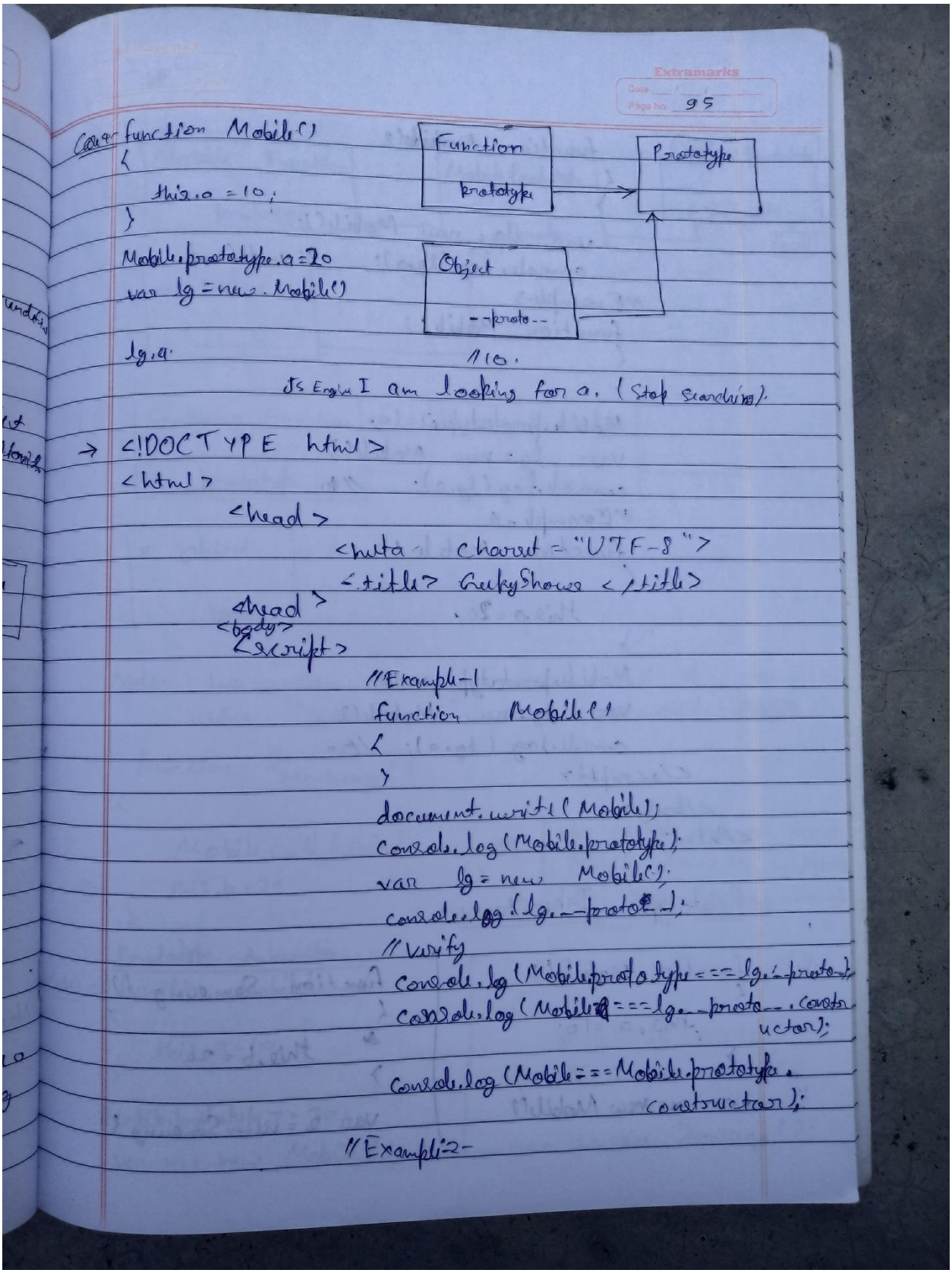
When you create a new object of that function using new keyword JS Engine creates an object and sets a property named --proto-- which points to its function's prototype object.

Note: If we create a new object of Mobile then that object --proto-- will point prototype of Function object.

In above case - lg.--proto-- === Mobile.prototype is True.

Note: जब भी हम JS में कोई function बनाते हैं तो 2 objects create होते एक Function का object और दूसरा उसके prototype का object. जिसे हम function.name.prototype की मदद से access कर सकते हैं। और यदि Function का कोई object create करते हैं तो उस object का एक --proto-- (property) भी बनता है जो function के prototype object को point करता है।


Case 1-
function Mobile()
{
}
var lg = new Mobile()
lg.a
Function
Prototype
Prototype
Object
--proto--
lg.a
Note- JS engine first find in lg object
if not found then find in function
prototype object if not found then
return undefined.
Case 2-
function Mobile()
{
this.a = 10;
}
var lg = new Mobile()
lg.a
Function
prototype
Prototype
Object
--proto--
//10
Note- JS. engine first find lg.a in
lg object and find a=10;
Case 3-
function Mobile()
{ }
Mobile.prototype.a=10
var lg = new Mobile()
lg.a
Function
prototype
Prototype
Object
--proto--
JS Engine looking for a.
Note- Mobile.prototype.a == lg.__proto__.a



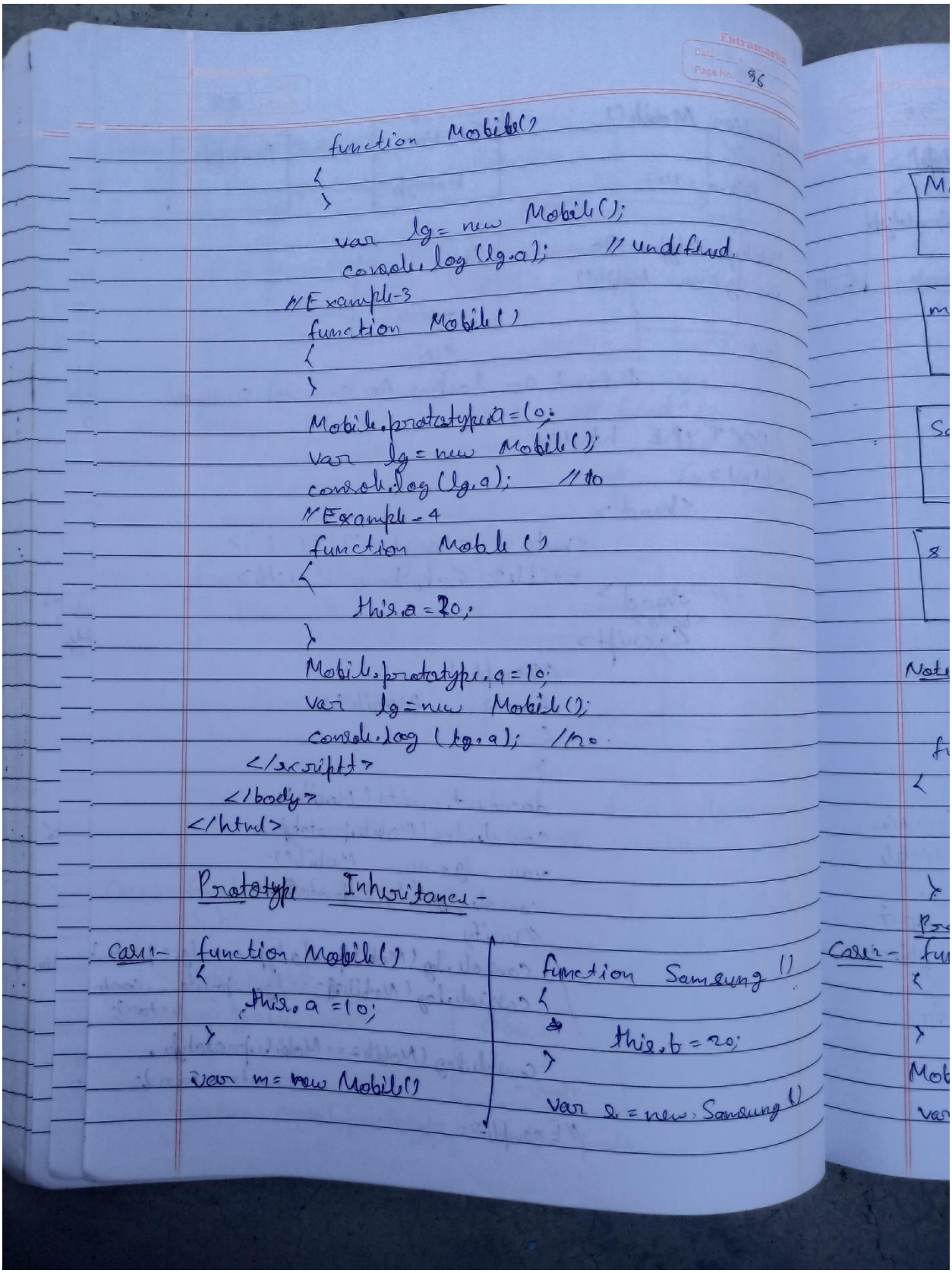
Extramarks
Date
Page No. 95

Car function Mobile()
{
this.a = 10;
}
Mobile.prototype.a=20
var lg = new Mobile()
lg.a.

Function
prototype
Prototype
Object
--proto--

//10.
JS Engine I am looking for a. (Stop searching).

→ <!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title> GeekyShows </title>
</head>
<body>
<script>
//Example-1
function Mobile()
{
}
document.write(Mobile);
Console.log(Mobile.prototype);
var lg = new Mobile();
console.log(lg.__proto__);
//verify
Console.log(Mobile.prototype === lg.__proto__);
Console.log(Mobile === lg.__proto__.constructor);
Console.log(Mobile === Mobile.prototype.constructor);
//Example-2-



```
function Mobile()
{
}
var lg = new Mobile();
console.log(lg.a);   // undefined.
//Example-3
function Mobile()
{
}
Mobile.prototype.a = 10;
var lg = new Mobile();
console.log(lg.a);   //10
//Example-4
function Mobile()
{
    this.a = 20;
}
Mobile.prototype.a = 10;
var lg = new Mobile();
console.log(lg.a);   //20.
</script>
</body>
</html>
```

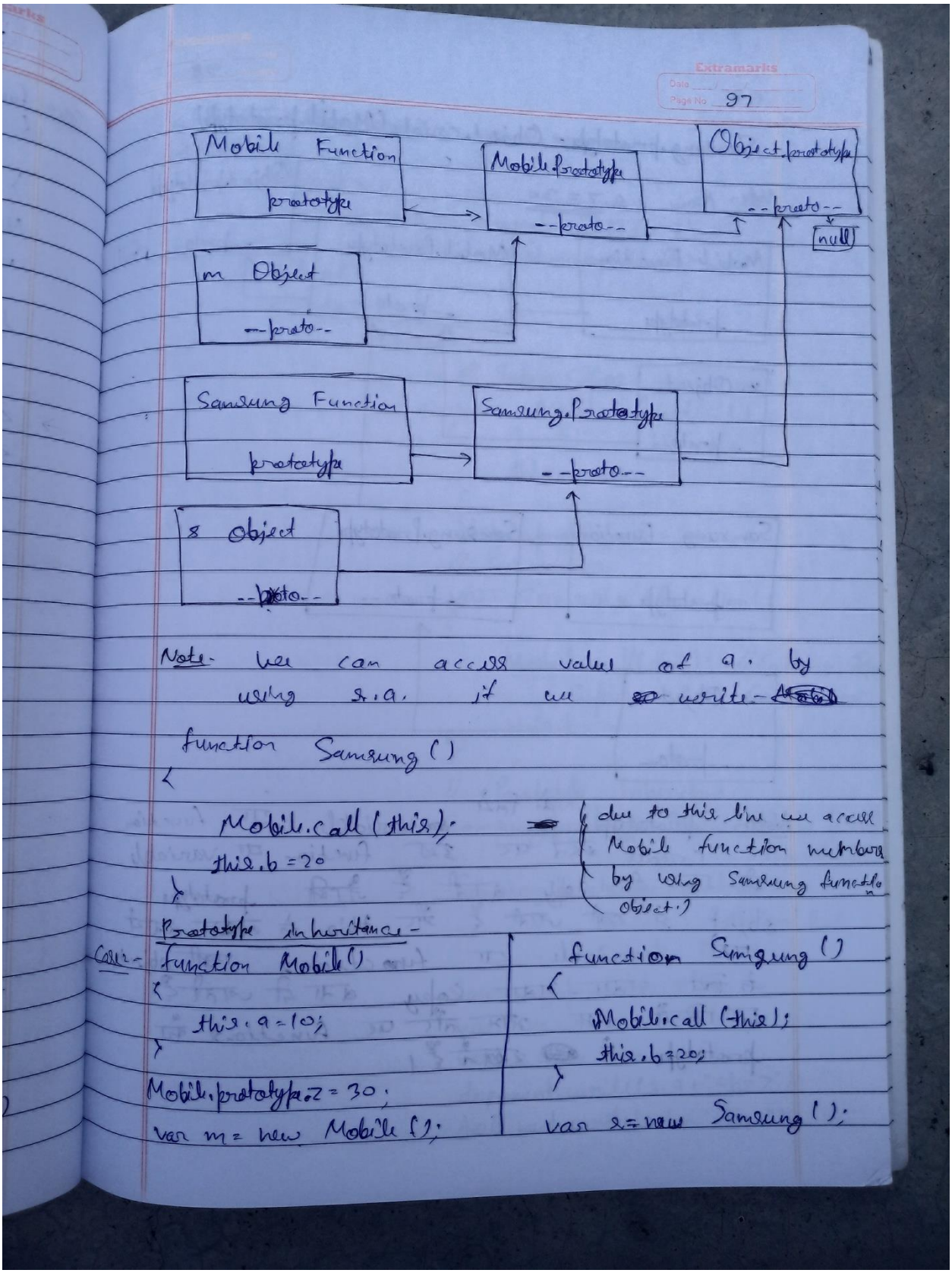
Prototype Inheritance -

Case1 -

```
function Mobile()
{
    this.a = 10;
}
var m = new Mobile()
```

```
function Samsung()
{
    this.b = 20;
}
var s = new Samsung()
```

Case2 -



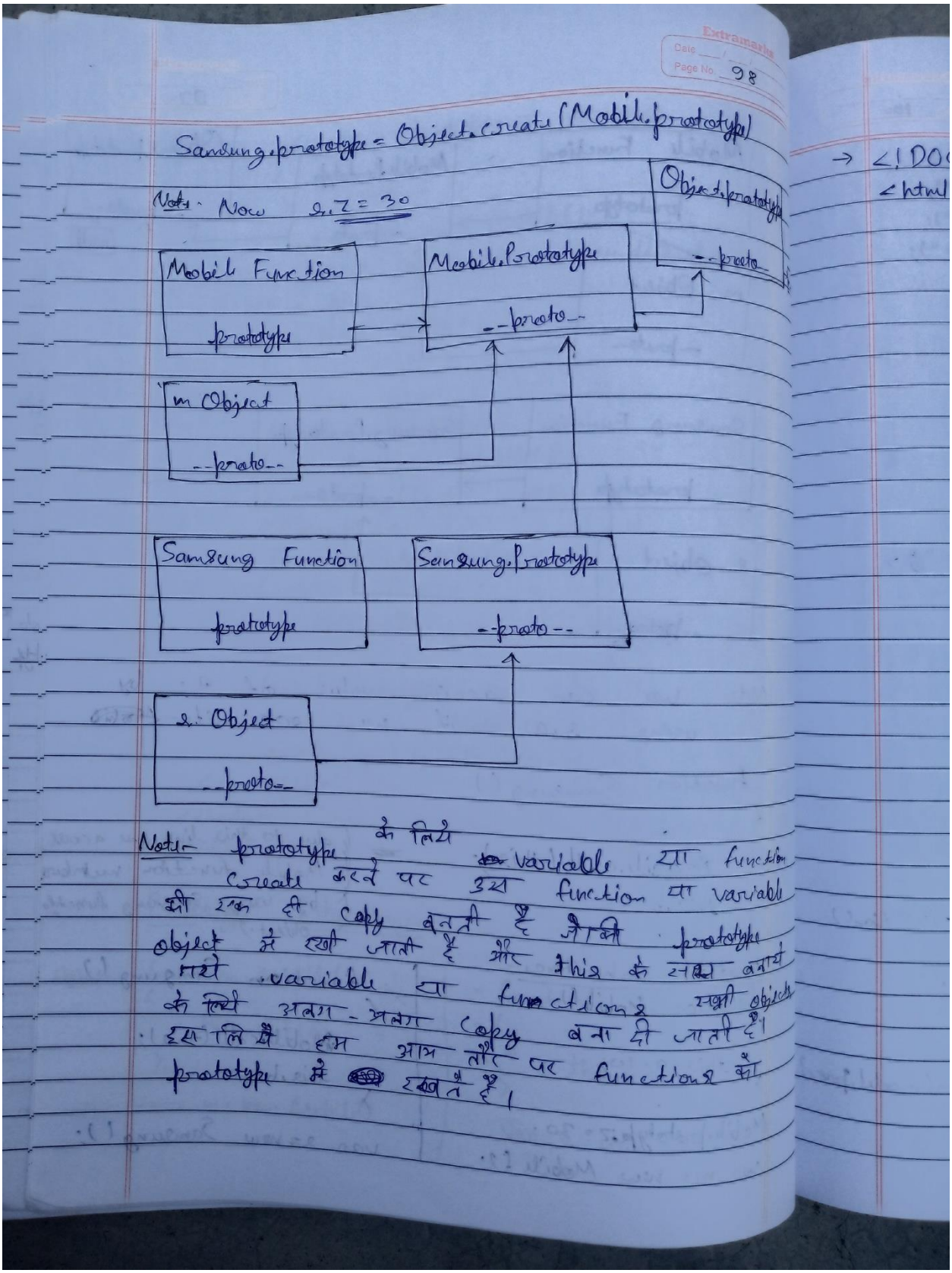
Note- We can access value of a. by using s.a. if we write-

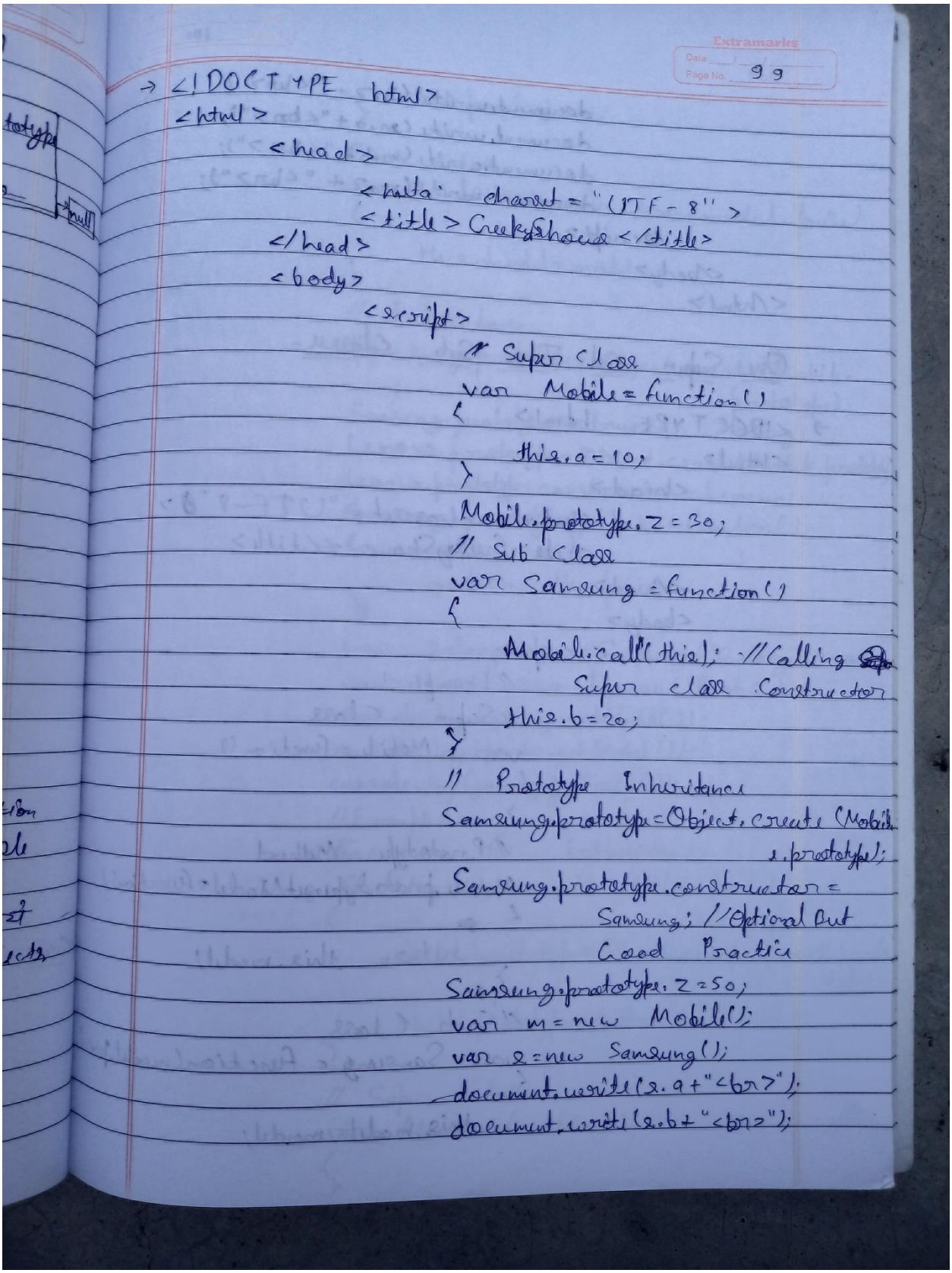
```
function Samsung()
{
    Mobile.call(this);   // due to this line we access Mobile function members by using Samsung function object.
    this.b = 20
}
```

Prototype Inheritance -

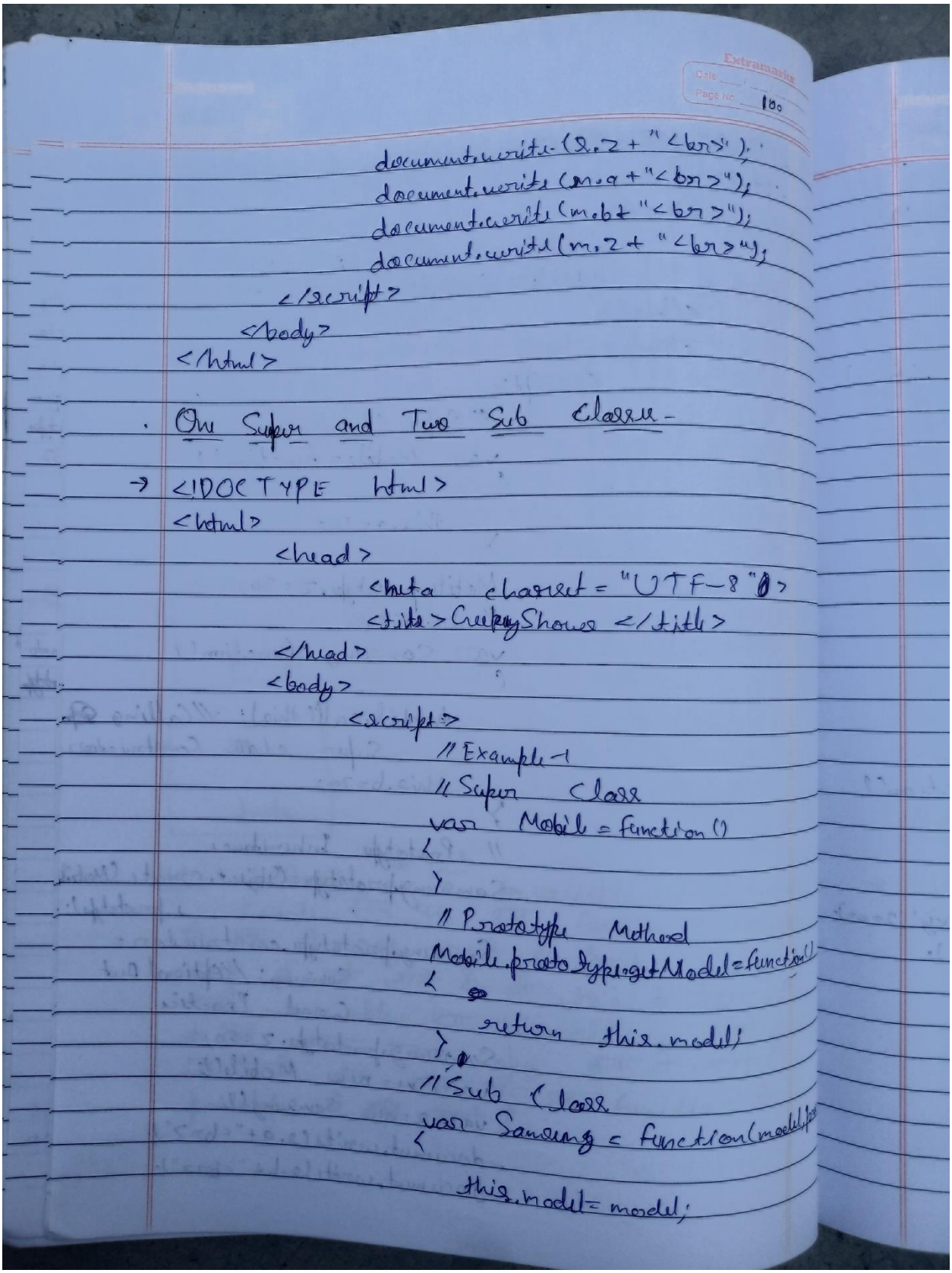
```
Call:- function Mobile()
{
    this.a = 10;
}
Mobile.prototype.z = 30;
var m = new Mobile();
```

```
function Simgung()
{
    Mobile.call(this);
    this.b = 20;
}
var s = new Samsung();
```


Samsung.prototype = Object.create(Mobile.prototype)
Note: Now s.z = 30
Object.prototype
--proto--
Mobile Function
prototype
Mobile.Prototype
--proto--
m Object
--proto--
Samsung Function
prototype
Samsung.Prototype
--proto--
s Object
--proto--
Note- prototype के लिये variable या function create करने पर उस function या variable की एक ही copy बनती है जैसी की prototype object में रखी जाती है और this के द्वारा बनाये गये variable या functions सभी objects के लिये अलग-अलग copy बना दी जाती है। इस लिये हम आम तौर पर functions को prototype में रखते हैं।



```
<!DOCTYPE html>
<html>
    <head>
        <meta charset = "UTF-8">
        <title> Geekyshows </title>
    </head>
    <body>
        <script>
            // Super Class
            var Mobile = function()
            {
                this.a = 10;
            }
            Mobile.prototype.z = 30;
            // Sub Class
            var Samsung = function()
            {
                Mobile.call(this); // Calling Super class Constructor
                this.b = 20;
            }
            // Prototype Inheritance
            Samsung.prototype = Object.create(Mobile.prototype);
            Samsung.prototype.constructor = Samsung; // Optional But Good Practice
            Samsung.prototype.z = 50;
            var m = new Mobile();
            var s = new Samsung();
            document.write(s.a + "<br>");
            document.write(s.b + "<br>");
```

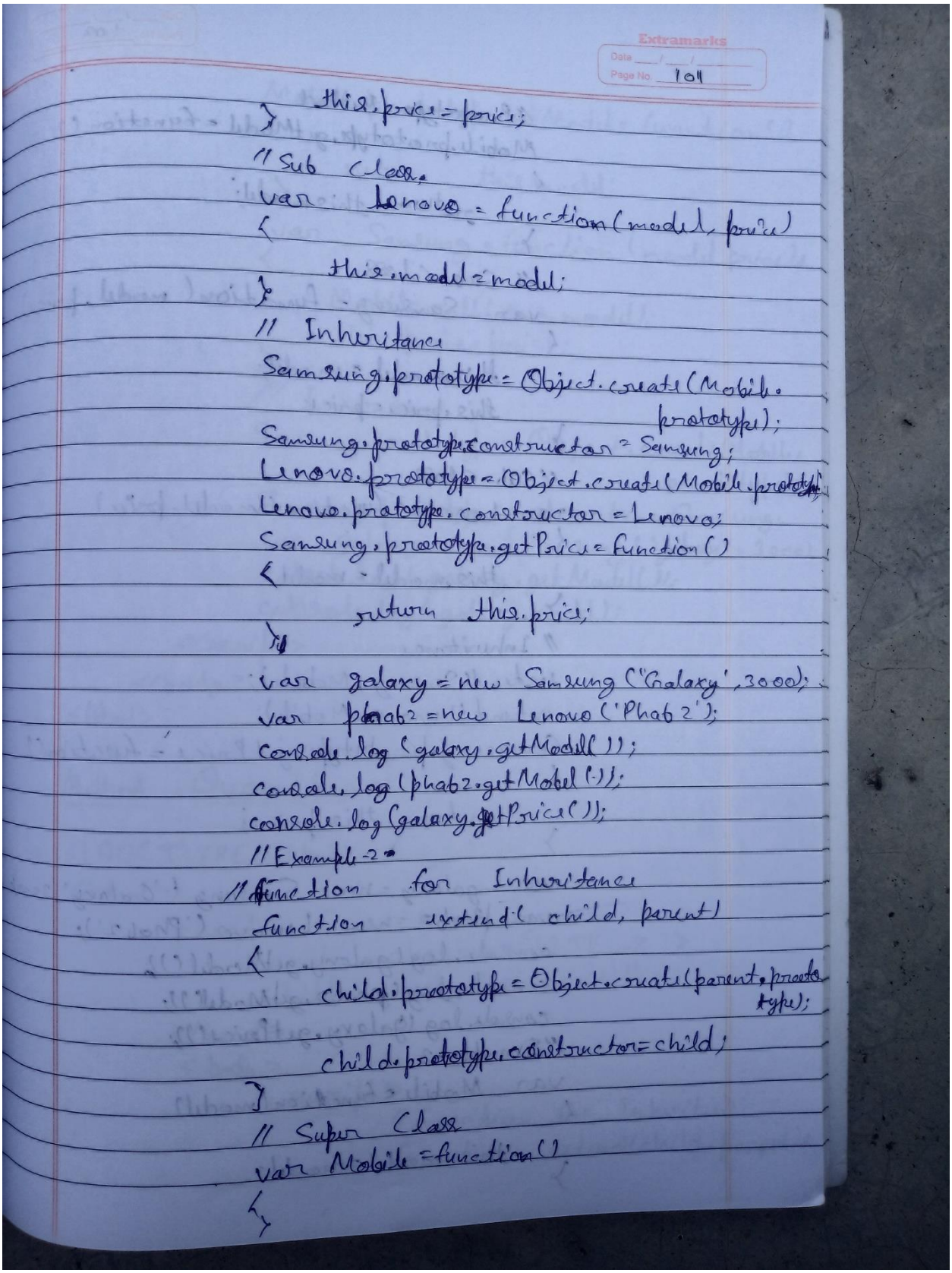


```
        document.write(s.z + "<br>");
        document.write(m.a + "<br>");
        document.write(m.b + "<br>");
        document.write(m.z + "<br>");
    </script>
</body>
</html>
```

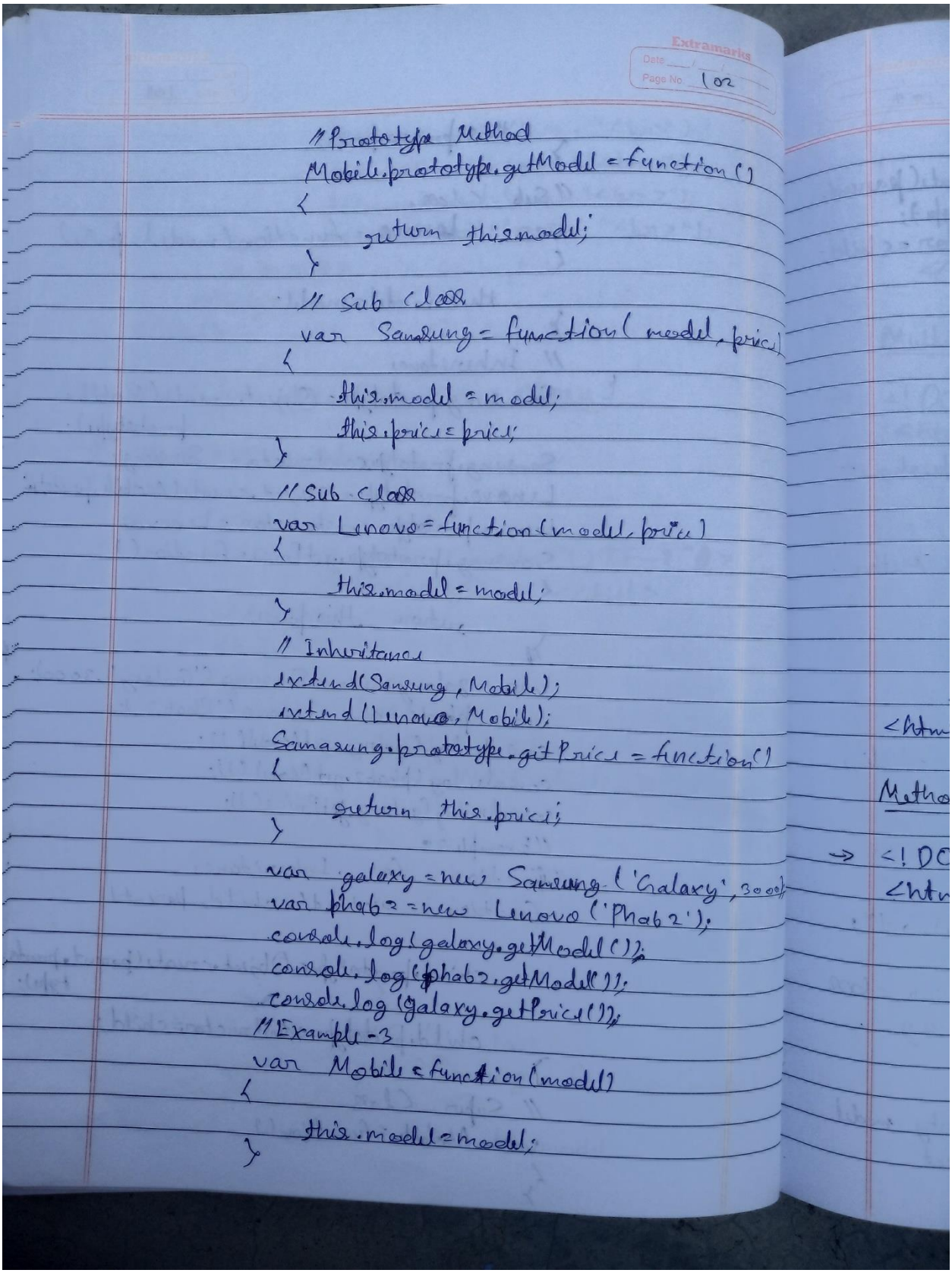
. One Super and Two Sub Classes -

→

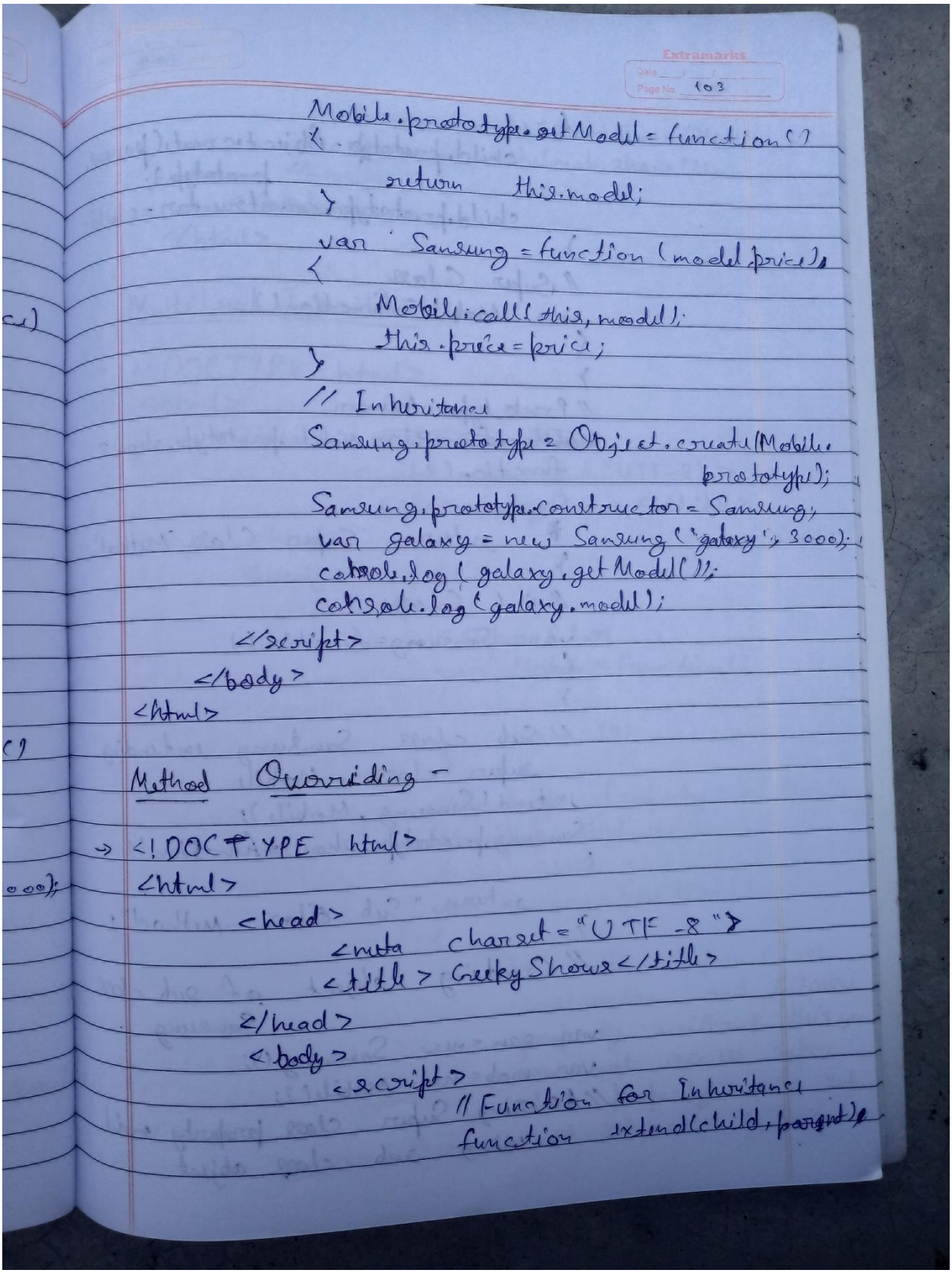
```
<!DOCTYPE html>
<html>
    <head>
        <meta charset = "UTF-8">
        <title> GeekyShows </title>
    </head>
    <body>
        <script>
            // Example-1
            // Super Class
            var Mobile = function()
            {
            }
            // Prototype Method
            Mobile.prototype.getModel = function()
            {
                return this.model;
            }
            // Sub Class
            var Samsung = function(model)
            {
                this.model = model;
```



```
    this.price = price;
}
// Sub Class.
var Lenovo = function(modul, price)
{
    this.modul = modul;
}
// Inheritance
Samsung.prototype = Object.create(Mobile.
                                  prototype);
Samsung.prototype.constructor = Samsung;
Lenovo.prototype = Object.create(Mobile.prototype);
Lenovo.prototype.constructor = Lenovo;
Samsung.prototype.getPrice = function()
{
    return this.price;
}
var galaxy = new Samsung('Galaxy', 3000);
var phab2 = new Lenovo('Phab 2');
console.log(galaxy.getModel());
console.log(phab2.getModel());
console.log(galaxy.getPrice());
// Example -2
// function for Inheritance
function extend(child, parent)
{
    child.prototype = Object.create(parent.prototype);
    child.prototype.constructor = child;
}
// Super Class
var Mobile = function()
{
}
```



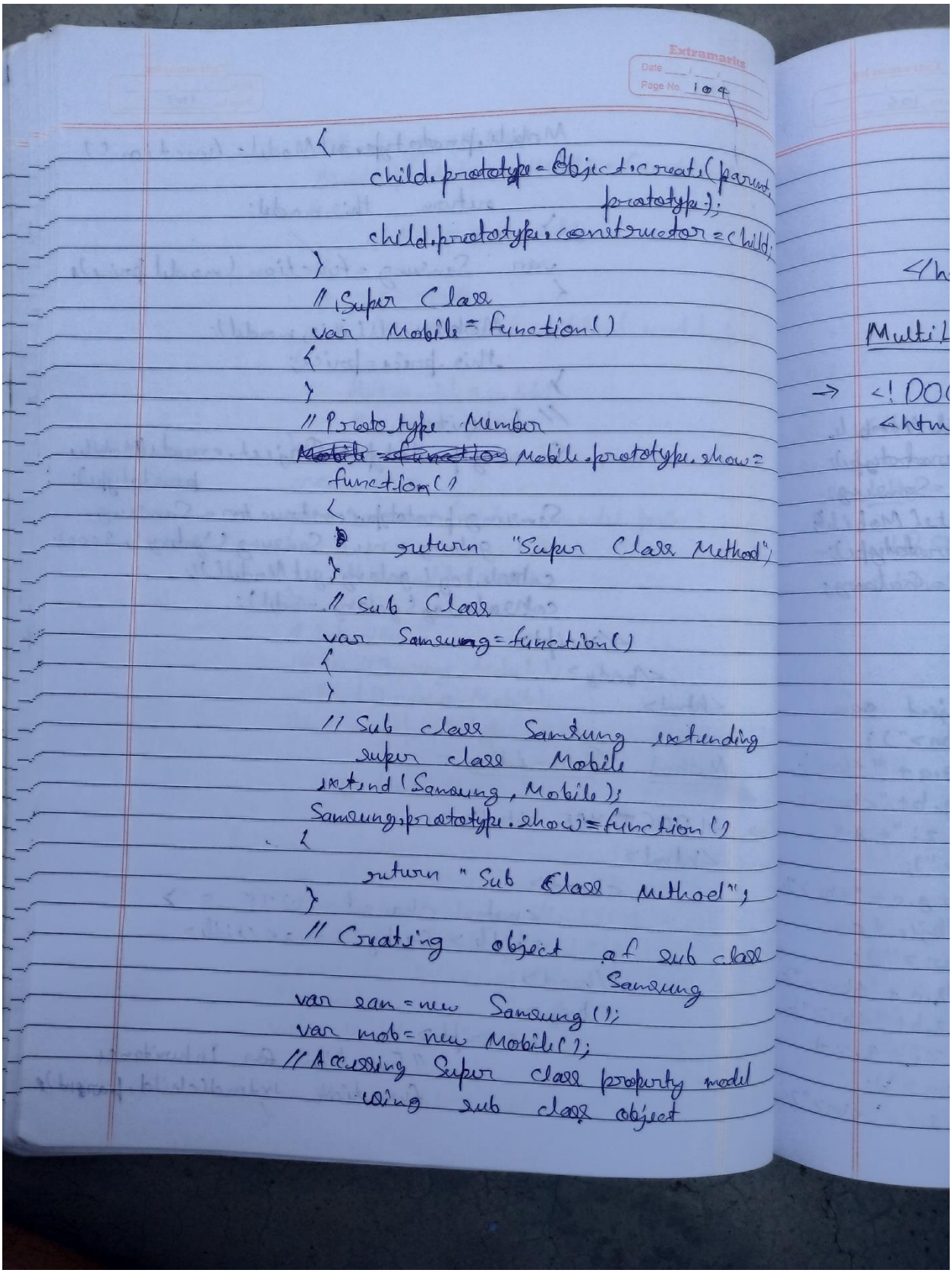
```
// Prototype Method
Mobile.prototype.getModel = function()
{
    return this.model;
}
// Sub Class
var Samsung = function(model, price)
{
    this.model = model;
    this.price = price;
}
// Sub Class
var Lenovo = function(model, price)
{
    this.model = model;
}
// Inheritance
extend(Samsung, Mobile);
extend(Lenovo, Mobile);
Samsung.prototype.getPrice = function()
{
    return this.price;
}
var galaxy = new Samsung('Galaxy', 3000);
var phab2 = new Lenovo('Phab2');
console.log(galaxy.getModel());
console.log(phab2.getModel());
console.log(galaxy.getPrice());
// Example-3
var Mobile = function(model)
{
    this.model = model;
}
```



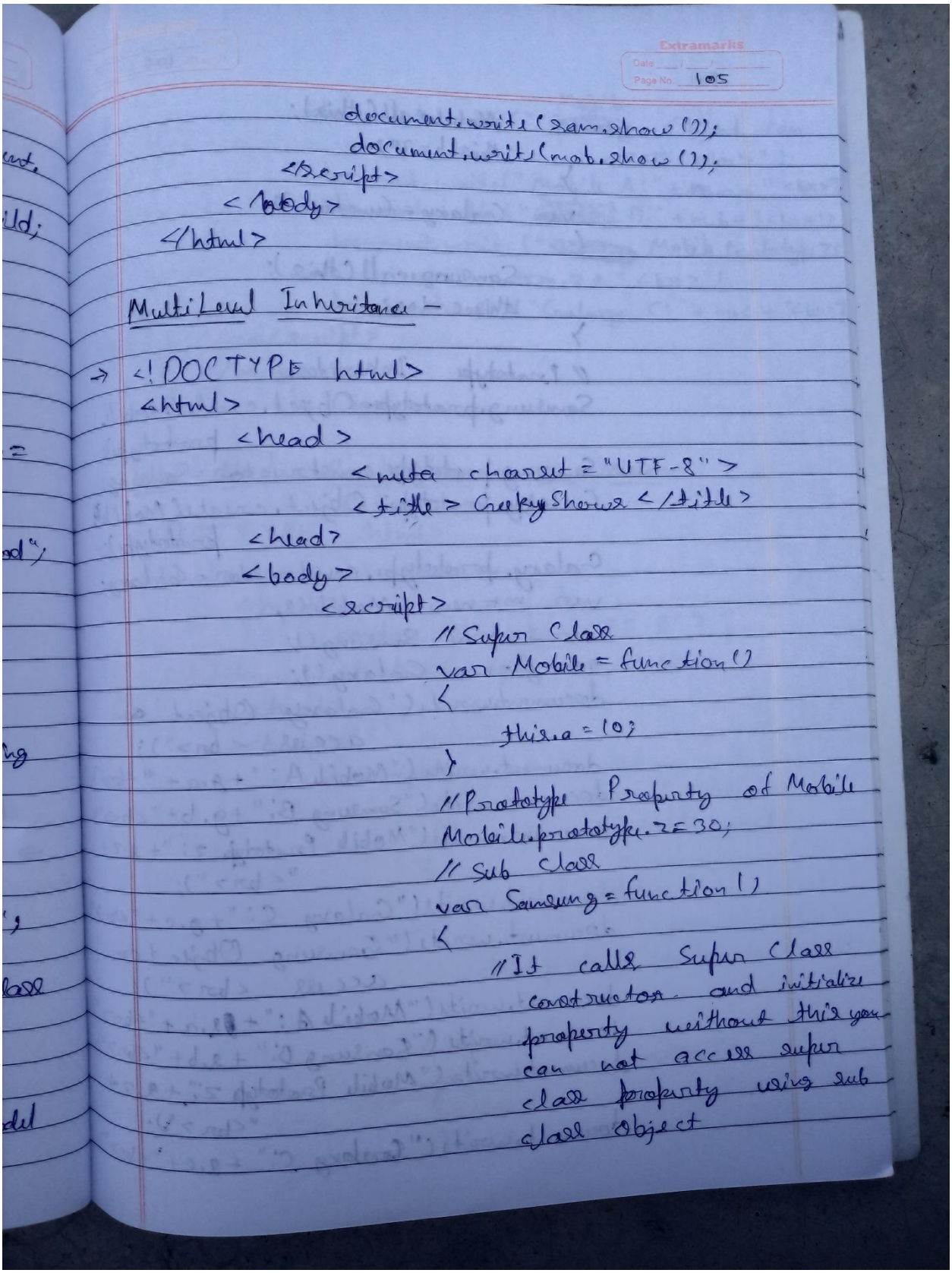
```
Mobile.prototype.getModel = function()
{
    return this.model;
}
var Samsung = function(model, price)
{
    Mobile.call(this, model);
    this.price = price;
}
// Inheritance
Samsung.prototype = Object.create(Mobile.
                                  prototype);
Samsung.prototype.constructor = Samsung;
var galaxy = new Samsung('galaxy', 3000);
console.log(galaxy.getModel());
console.log(galaxy.model);
</script>
</body>
</html>
```

Method Overriding -

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="UTF-8">
        <title>Geeky Shows</title>
    </head>
    <body>
        <script>
            // Function for Inheritance
            function extend(child, parent)
```



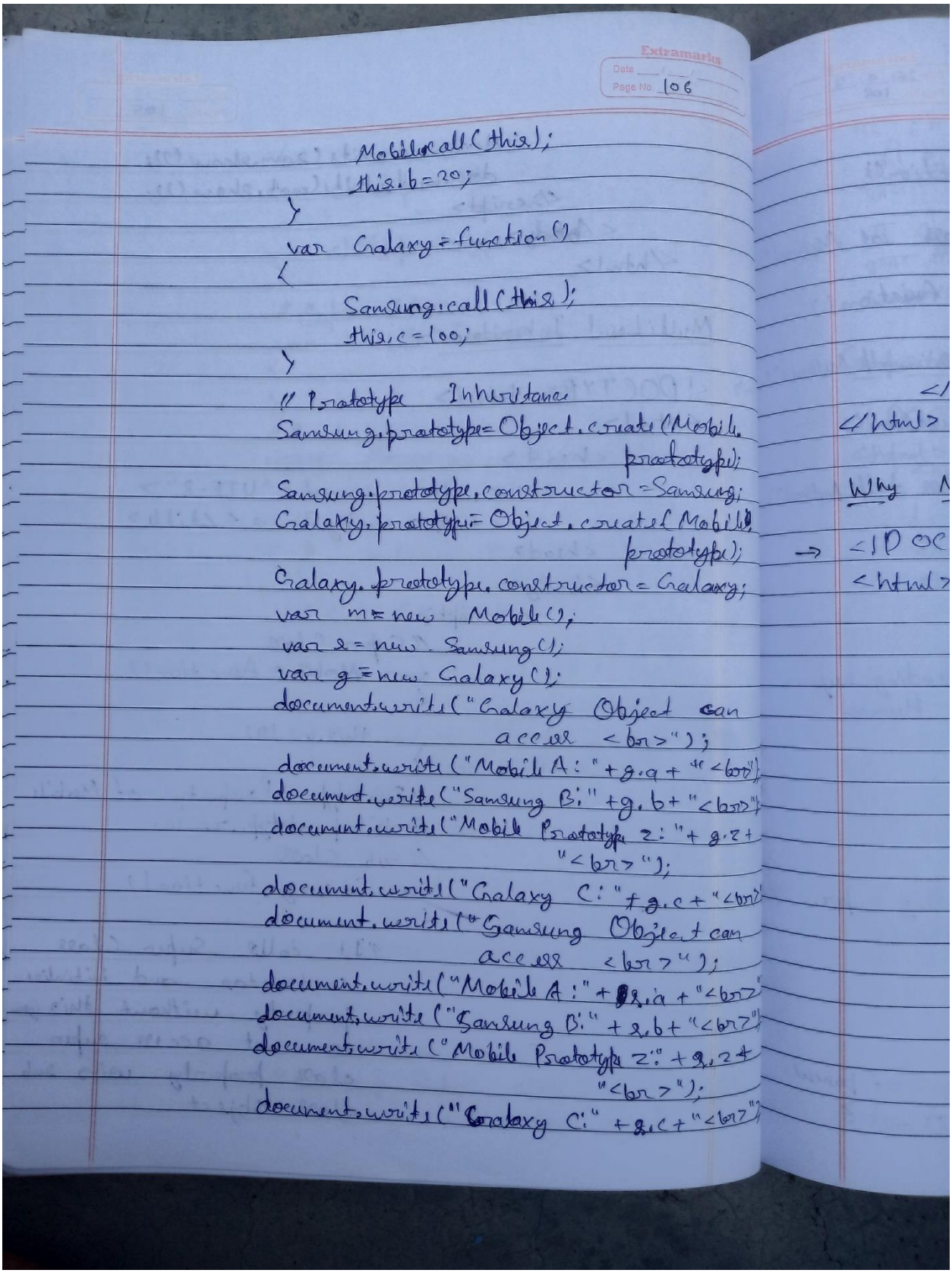
```
{
    child.prototype = Object.create(parent.
                                    prototype);
    child.prototype.constructor = child;
}
// Super Class
var Mobile = function()
{
}
// Prototype Member
Mobile.prototype.show =
 function()
 {
    return "Super Class Method";
 }
// Sub Class
var Samsung = function()
{
}
// Sub class Samsung extending
   super class Mobile
extend('Samsung', 'Mobile');
Samsung.prototype.show = function()
{
    return "Sub Class Method";
}
// Creating object of sub class
                        Samsung
var san = new Samsung();
var mob = new Mobile();
// Accessing Super class property modul
      using sub class object
```



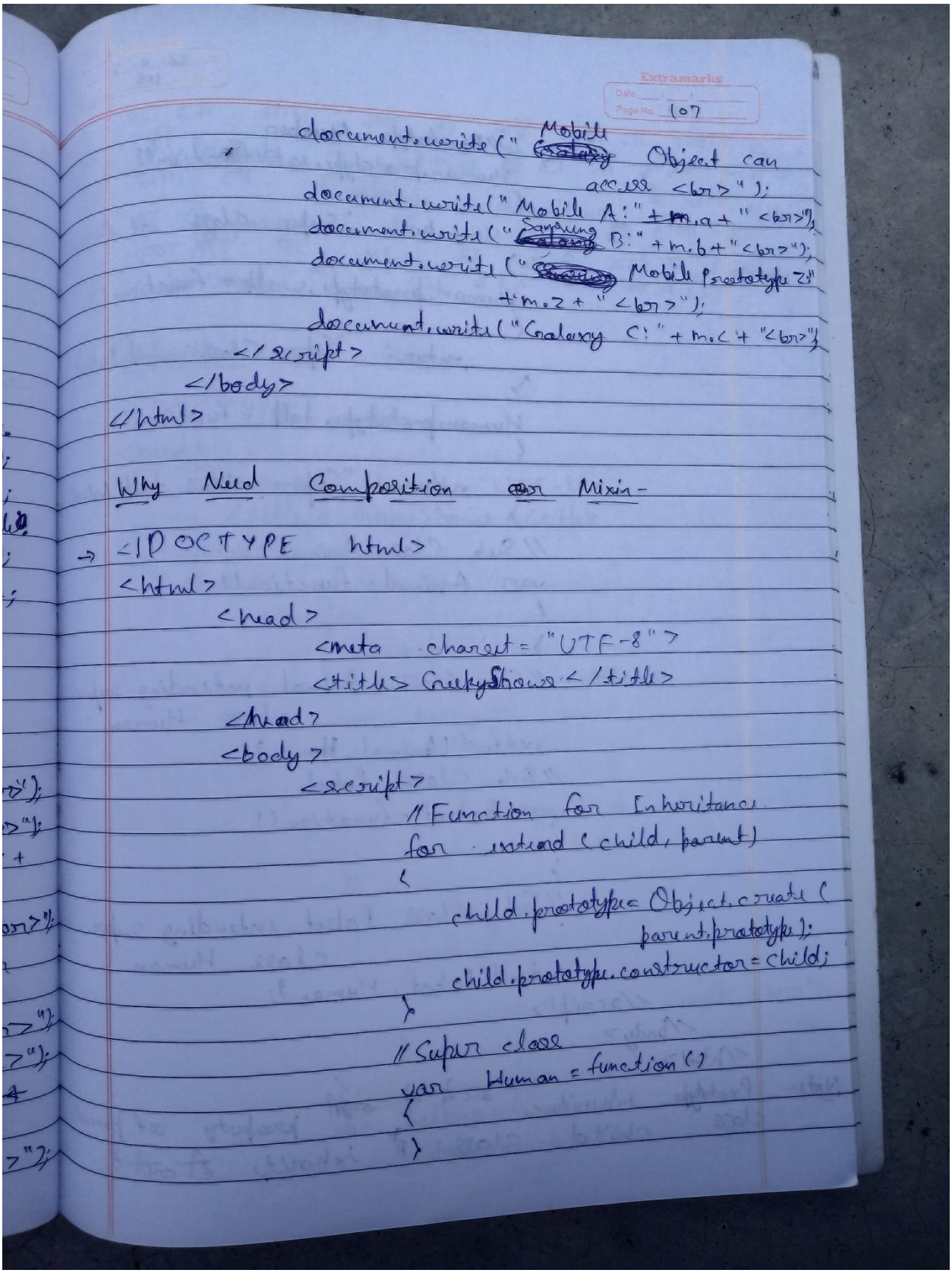
```
        document.write("Sam.show()");
        document.write("mob.show()");
    </script>
  </body>
</html>
```

MultiLevel Inheritance –

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="UTF-8">
        <title> Geekyshows </title>
    </head>
    <body>
        <script>
            // Super Class
            var Mobile = function()
            {
                this.a = 10;
            }
            // Prototype Property of Mobile
            Mobile.prototype.z = 30;
            // Sub Class
            var Samsung = function()
            {
                // It calls Super Class
                constructor and initialize
                property without this you
                can not access super
                class property using sub
                class object
```



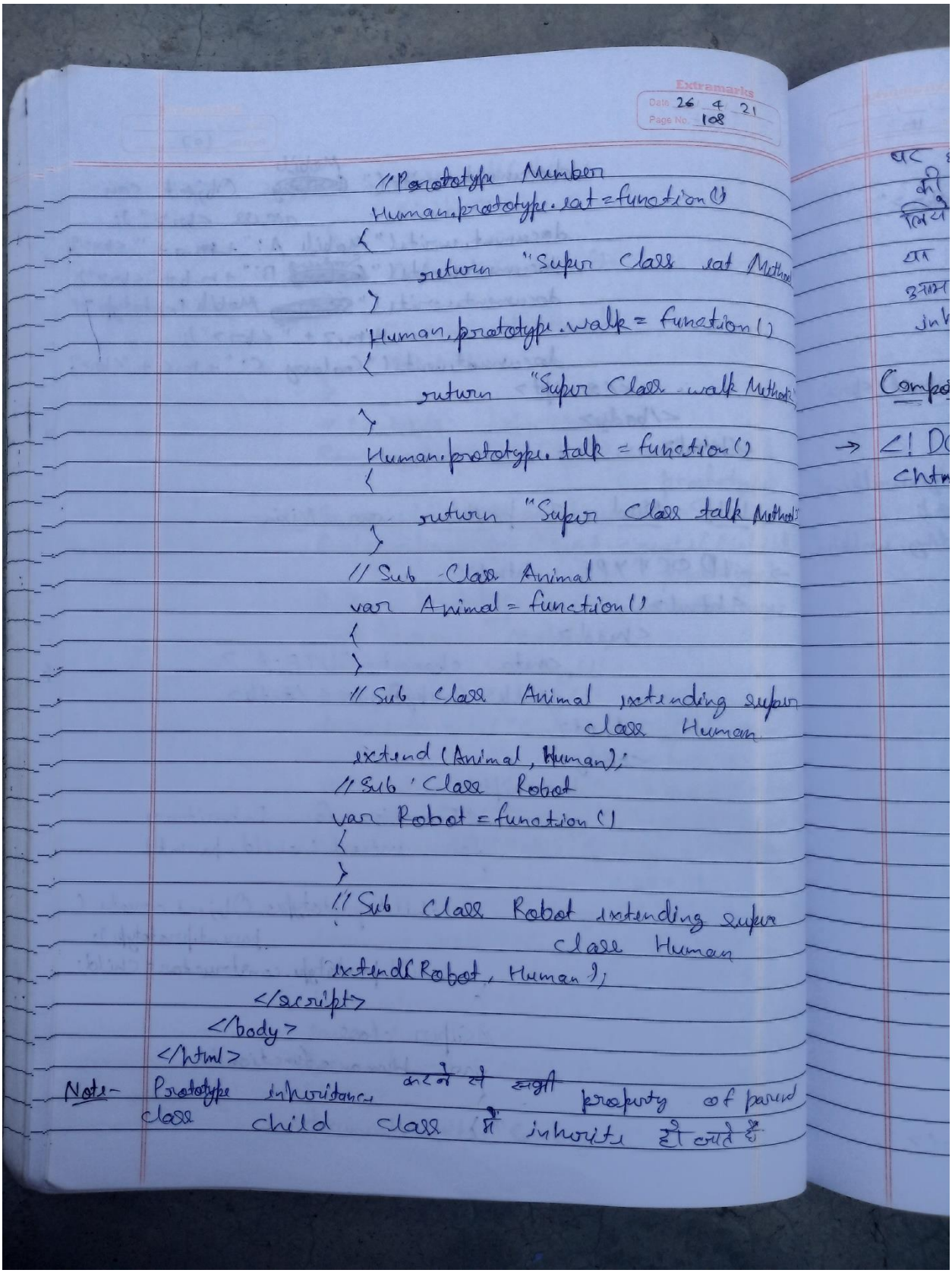
```
    Mobile.call(this);
    this.b=20;
}
var Galaxy=function()
{
    Samsung.call(this);
    this.c=100;
}
// Prototype Inheritance
Samsung.prototype=Object.create(Mobile.
                    prototype);
Samsung.prototype.constructor=Samsung;
Galaxy.prototype=Object.create(Mobile.
                    prototype);
Galaxy.prototype.constructor=Galaxy;
var m=new Mobile();
var s=new Samsung();
var g=new Galaxy();
document.write("Galaxy Object can
                access <br>");
document.write("Mobile A:"+g.a+"<br>");
document.write("Samsung B:"+g.b+"<br>");
document.write("Mobile Prototype z:"+g.z+
                "<br>");
document.write("Galaxy C:"+g.c+"<br>");
document.write("Samsung Object can
                access <br>");
document.write("Mobile A:"+s.a+"<br>");
document.write("Samsung B:"+s.b+"<br>");
document.write("Mobile Prototype z:"+s.z+
                "<br>");
document.write("Galaxy C:"+s.c+"<br>");
```



```
document.write("Mobile Object can access <br>");
document.write("Mobile A:" + m.a + "<br>");
document.write("Samsung B:" + m.b + "<br>");
document.write("Mobile Prototype Z:" + m.z + "<br>");
document.write("Galaxy C:" + m.c + "<br>");
</script>
</body>
</html>
```

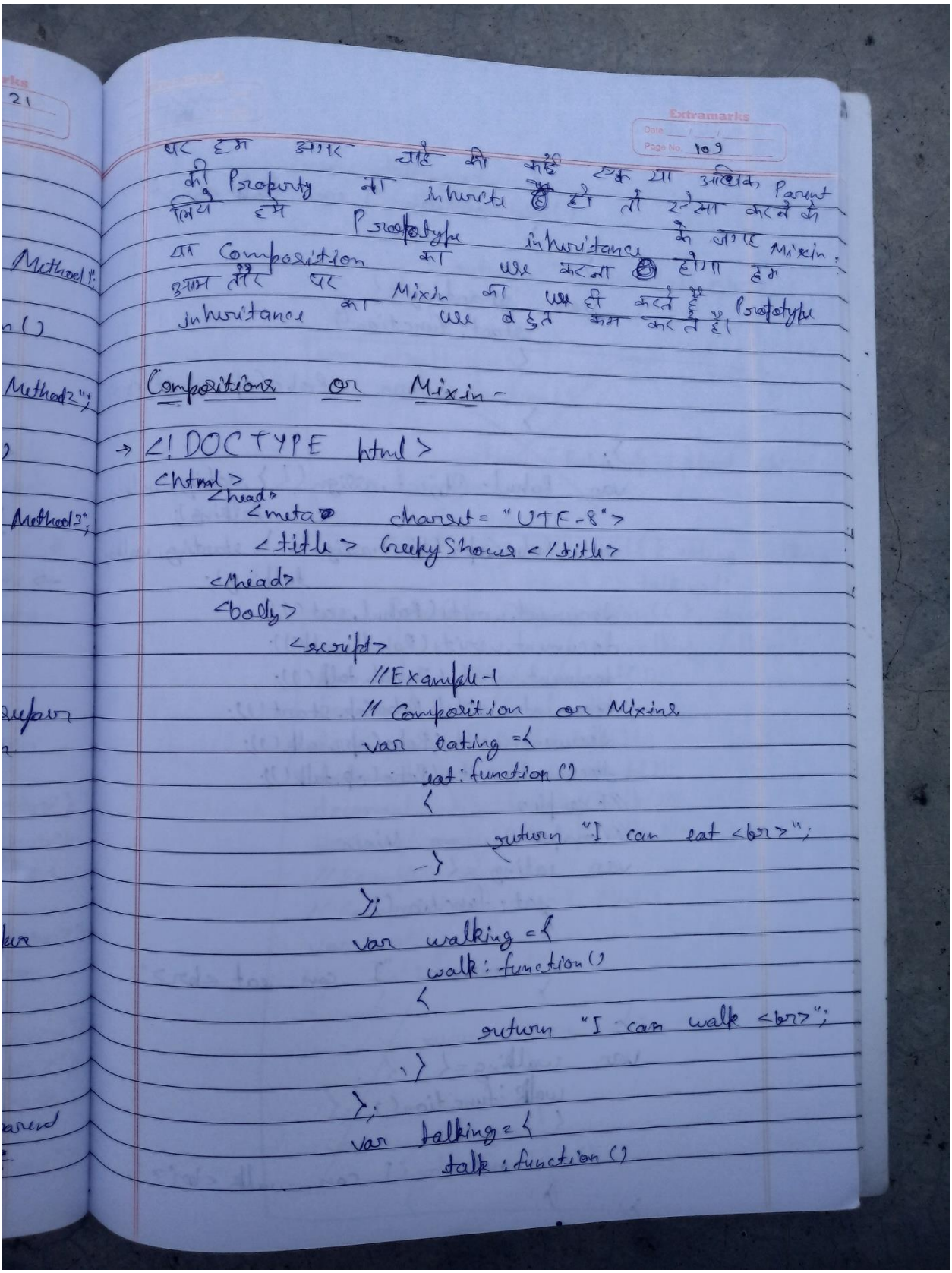
Why Need Comparition or Mixin -

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="UTF-8">
        <title> Geekyshows </title>
    </head>
    <body>
        <script>
            // Function for Inheritance
            for extend (child, parent)
            {
                child.prototype = Object.create(
                                    parent.prototype);
                child.prototype.constructor = child;
            }
            // Super class
            var Human = function()
            {
            }
```



```
//Prototype Number
Human.prototype.eat = function()
{
    return "Super class eat Method";
}
Human.prototype.walk = function()
{
    return "Super Class walk Method";
}
Human.prototype.talk = function()
{
    return "Super Class talk Method";
}
// Sub Class Animal
var Animal = function()
{
}
// Sub Class Animal extending super
                    class Human
extend(Animal, Human);
// Sub Class Robot
var Robot = function()
{
}
// Sub Class Robot extending super
                    class Human
extend(Robot, Human);
</script>
</body>
</html>
```

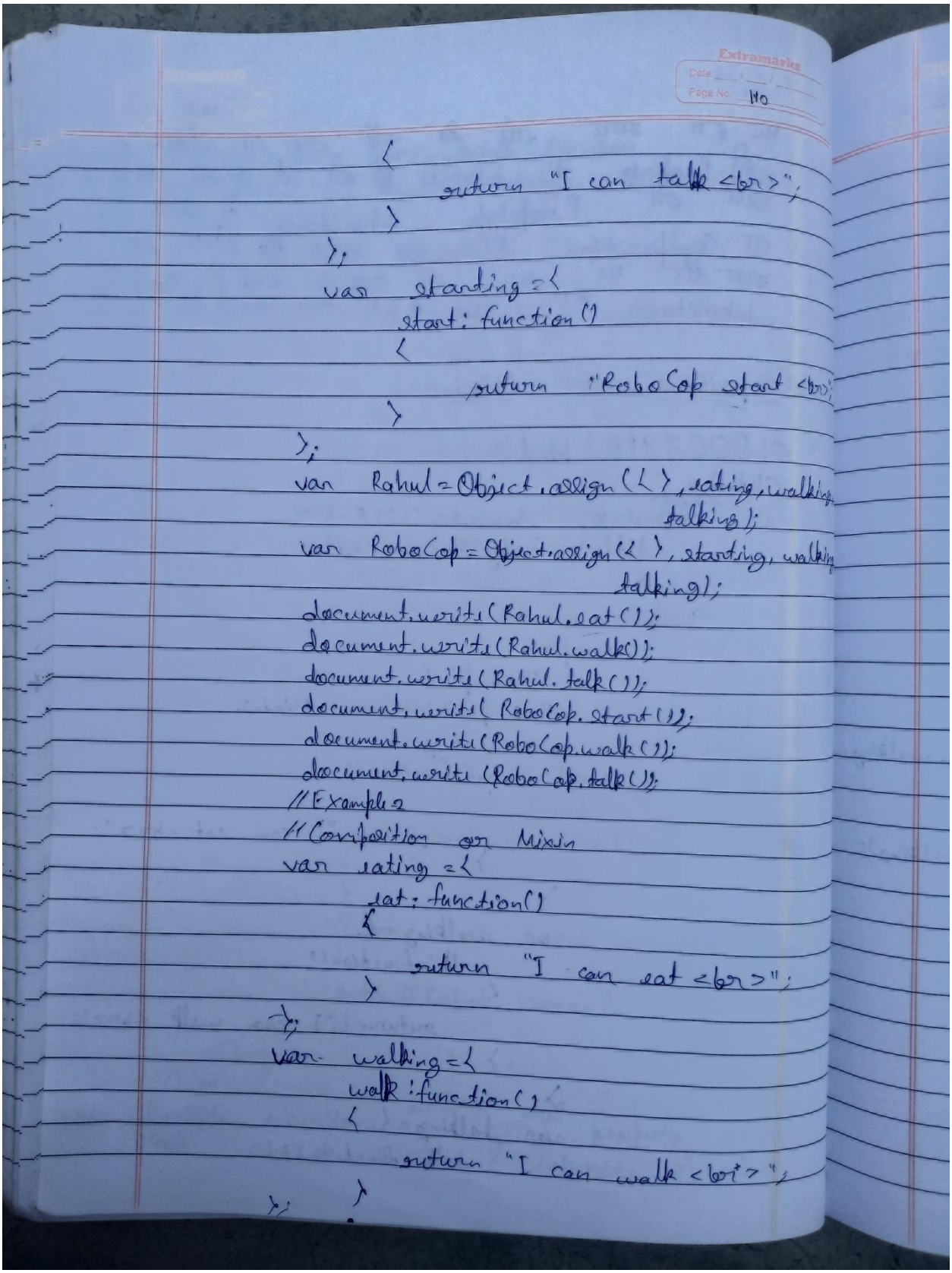
Note- Prototype inheritance ... property of parent class child class में inherit हो जाते हैं



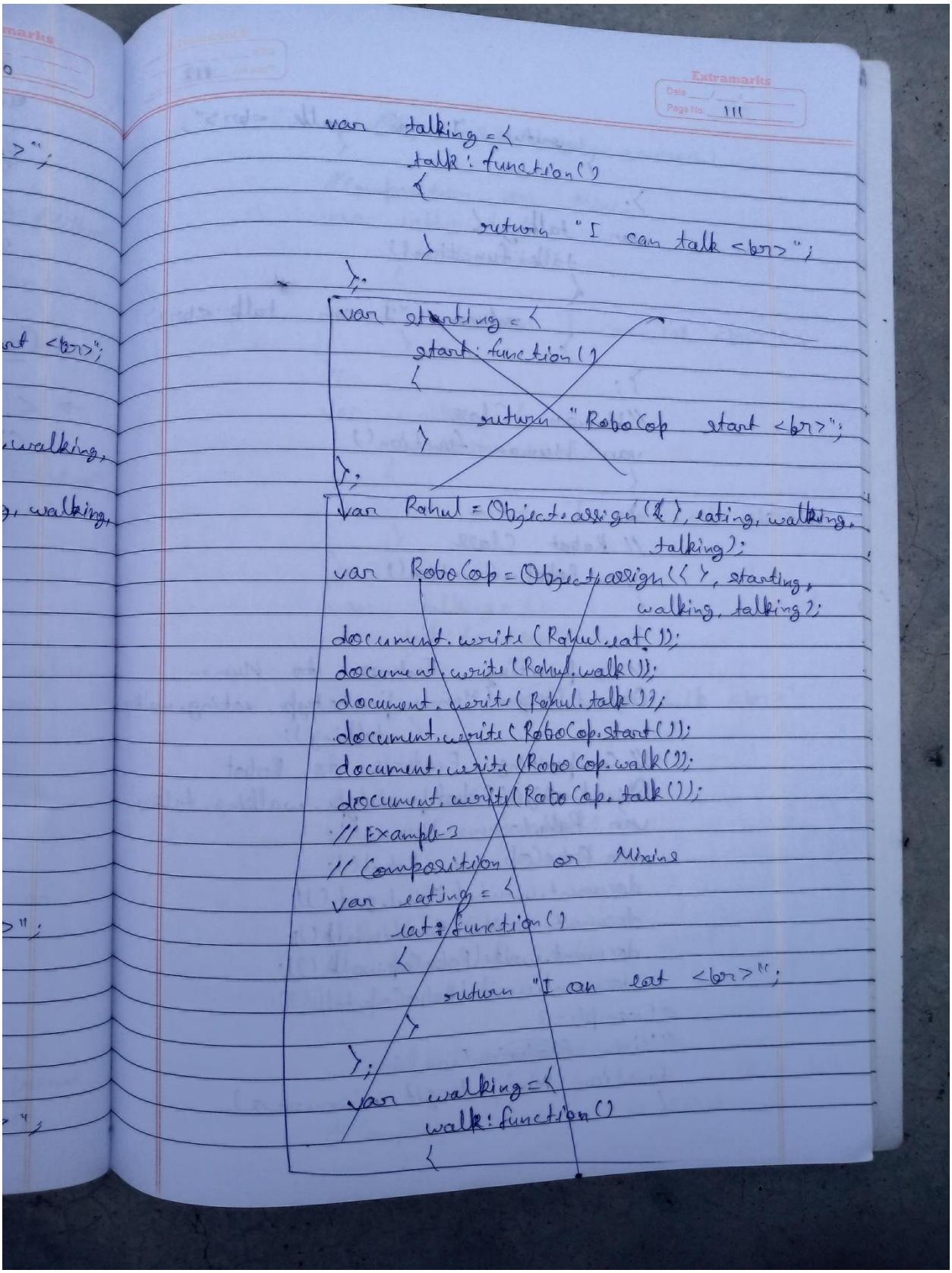
अगर हम अगर चाहें की कहीं एक या अधिक Parent की Property का inherit हो तो ऐसा करने के लिये हमे Prototype inheritance के जगह Mixin या Composition का use करना होगा। हम ज्यादा तर पर Mixin का use ही करते हैं Prototype inheritance का use कुछ कम करते हैं।

Compositions or Mixin -

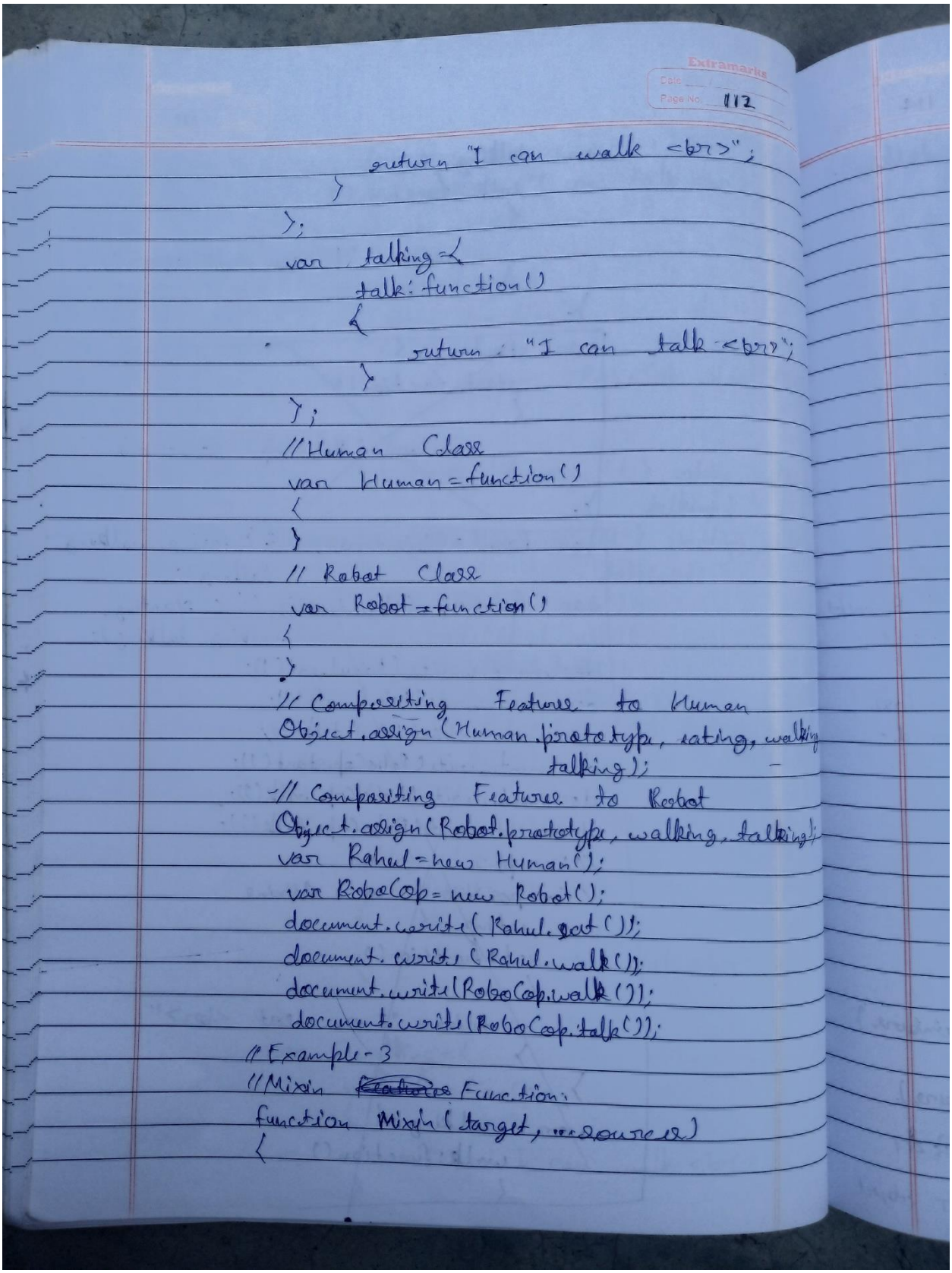
```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
    <title> GeekyShows </title>
  </head>
  <body>
    <script>
      //Example-1
      // Composition or Mixins
      var eating ={
        eat: function()
        {
          return "I can eat <br>";
        }
      };
      var walking ={
        walk: function()
        {
          return "I can walk <br>";
        }
      };
      var talking ={
        talk: function()
```



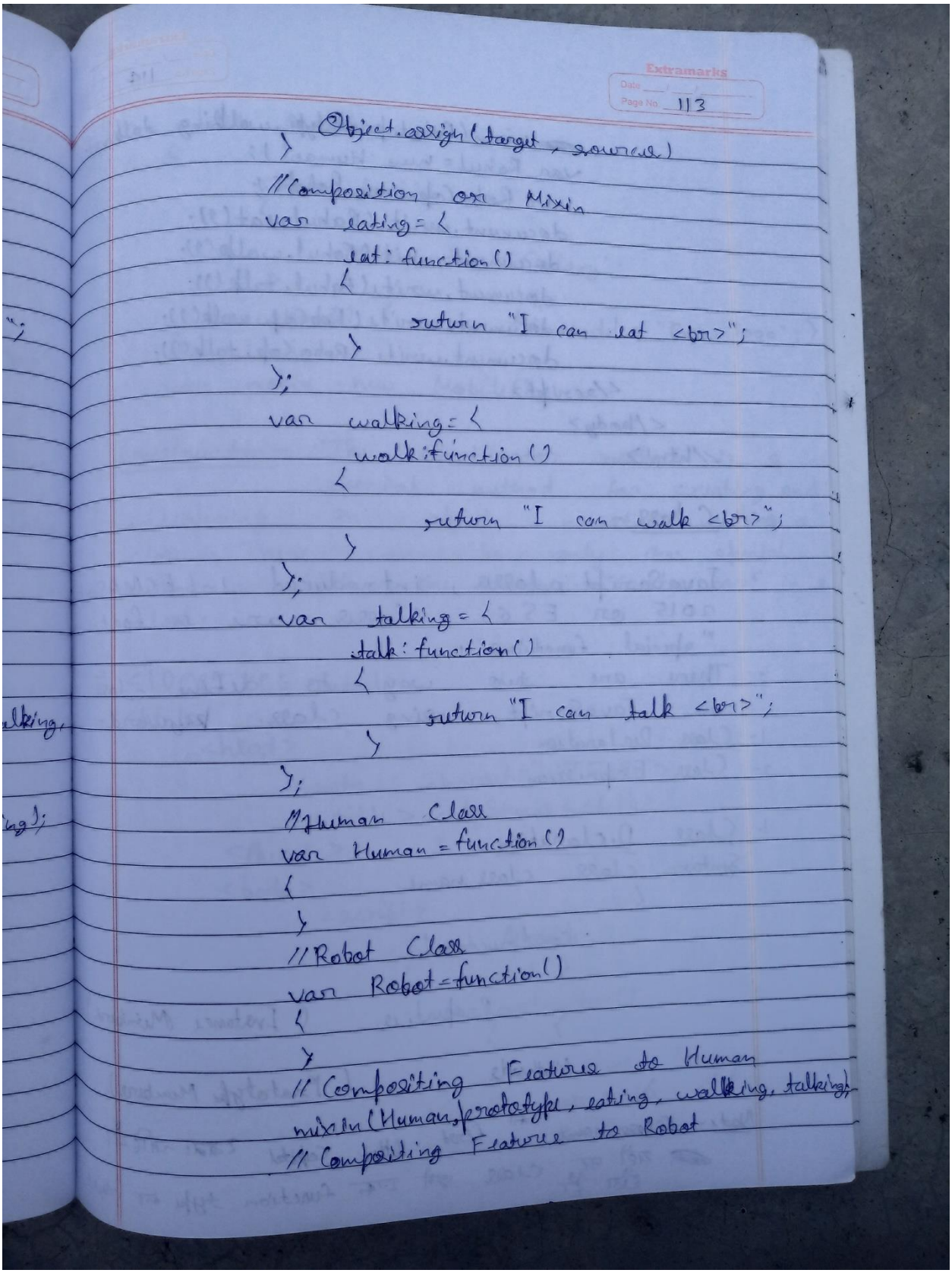
```
        {
            return "I can talk <br>";
        }
    };
    var starting = {
        start: function()
        {
            return "RoboCop start <br>";
        }
    };
    var Rahul = Object.assign({}, eating, walking,
                              talking);
    var RoboCop = Object.assign({}, starting, walking,
                                talking);
    document.write(Rahul.eat());
    document.write(Rahul.walk());
    document.write(Rahul.talk());
    document.write(RoboCop.start());
    document.write(RoboCop.walk());
    document.write(RoboCop.talk());
    //Example 2
    //Composition or Mixin
    var eating = {
        eat: function()
        {
            return "I can eat <br>";
        }
    };
    var walking = {
        walk: function()
        {
            return "I can walk <br>";
        }
    };
```



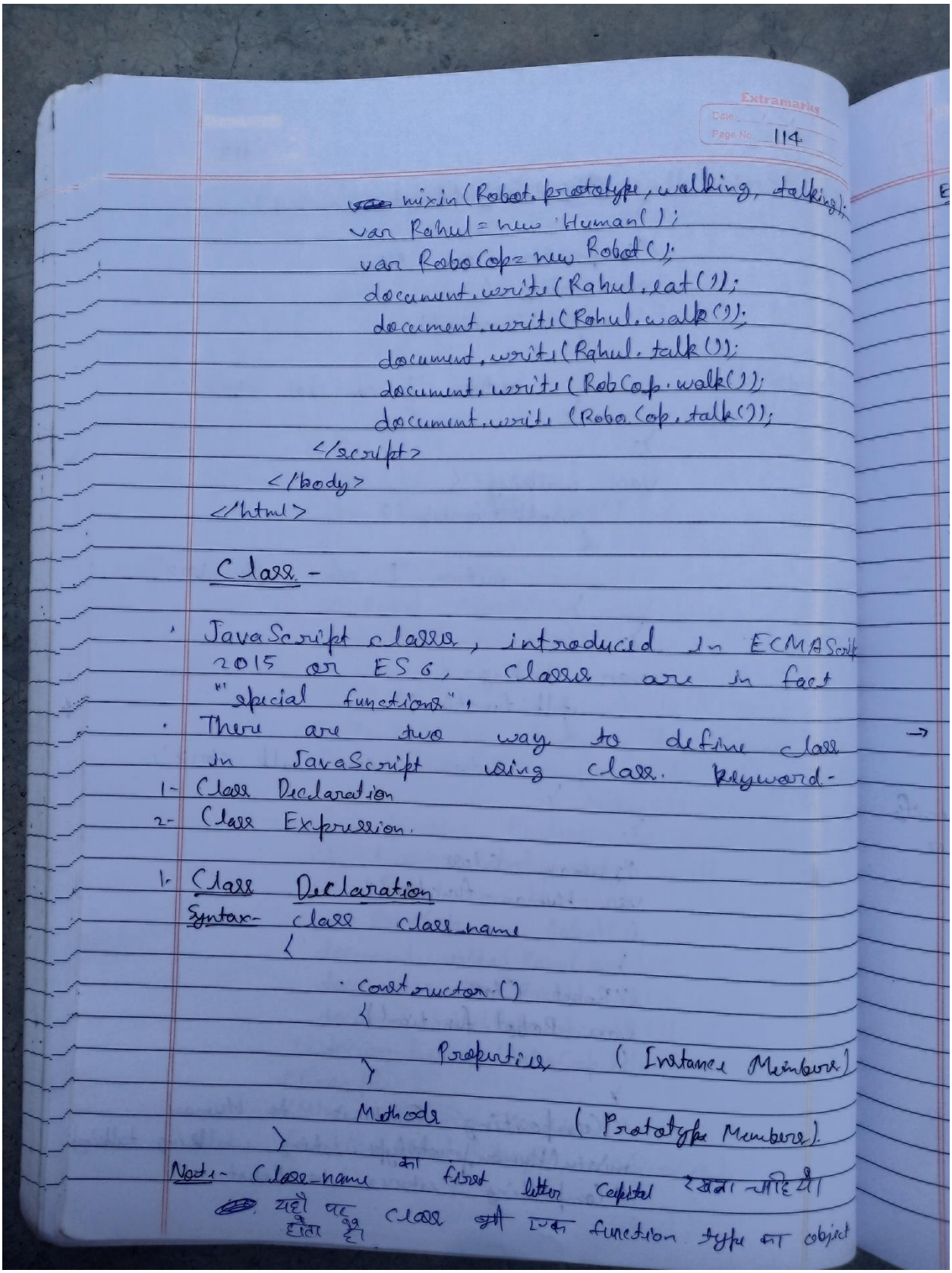
```
var talking = {
    talk: function()
    {
        return "I can talk <br>";
    }
};
var starting = {
    start: function()
    {
        return "RoboCop start <br>";
    }
};
var Rahul = Object.assign({}, eating, walking, talking);
var RoboCop = Object.assign({}, starting, walking, talking);
document.write(Rahul.eat());
document.write(Rahul.walk());
document.write(Rahul.talk());
document.write(RoboCop.start());
document.write(RoboCop.walk());
document.write(RoboCop.talk());
// Example-3
// Composition or Mixins
var eating = {
    eat: function()
    {
        return "I can eat <br>";
    }
};
var walking = {
    walk: function()
    {
```



```
        return "I can walk <br>";
    }
};
var talking={
    talk: function()
    {
        return "I can talk <br>";
    }
};
//Human Class
var Human=function()
{
}
// Robot Class
var Robot=function()
{
}
// Composeting Features to Human
Object.assign(Human.prototype, eating, walking
                                    talking);
// Composeting Features to Robot
Object.assign(Robot.prototype, walking, talking);
var Rahul=new Human();
var RoboCop=new Robot();
document.write(Rahul.eat());
document.write(Rahul.walk());
document.write(RoboCop.walk());
document.write(RoboCop.talk());
// Example-3
// Mixin ~~Factories~~ Function:
function Mixin(target, ...sources)
{
```



```
Extramarks
Date
Page No. 113

@Object.assign(target, sources)
}
//Composition or Mixin
var eating = {
    eat: function()
    {
        return "I can eat <br>";
    }
};
var walking = {
    walk: function()
    {
        return "I can walk <br>";
    }
};
var talking = {
    talk: function()
    {
        return "I can talk <br>";
    }
};
//Human Class
var Human = function()
{
}
//Robot Class
var Robot = function()
{
}
// Compositing Features to Human
mixin(Human.prototype, eating, walking, talking);
// Compositing Features to Robot
```



```
mixin(Robot.prototype, walking, talking);
var Rahul = new Human();
var RoboCop = new Robot();
document.write(Rahul.eat());
document.write(Rahul.walk());
document.write(Rahul.talk());
document.write(RobCop.walk());
document.write(Robo.Cop.talk());
</script>
</body>
</html>
```

Class -

- JavaScript class, introduced in ECMAScript 2015 or ES6, classes are in fact "special functions".
- There are two way to define class in JavaScript using class. keyword -

1- Class Declaration
2- Class Expression.

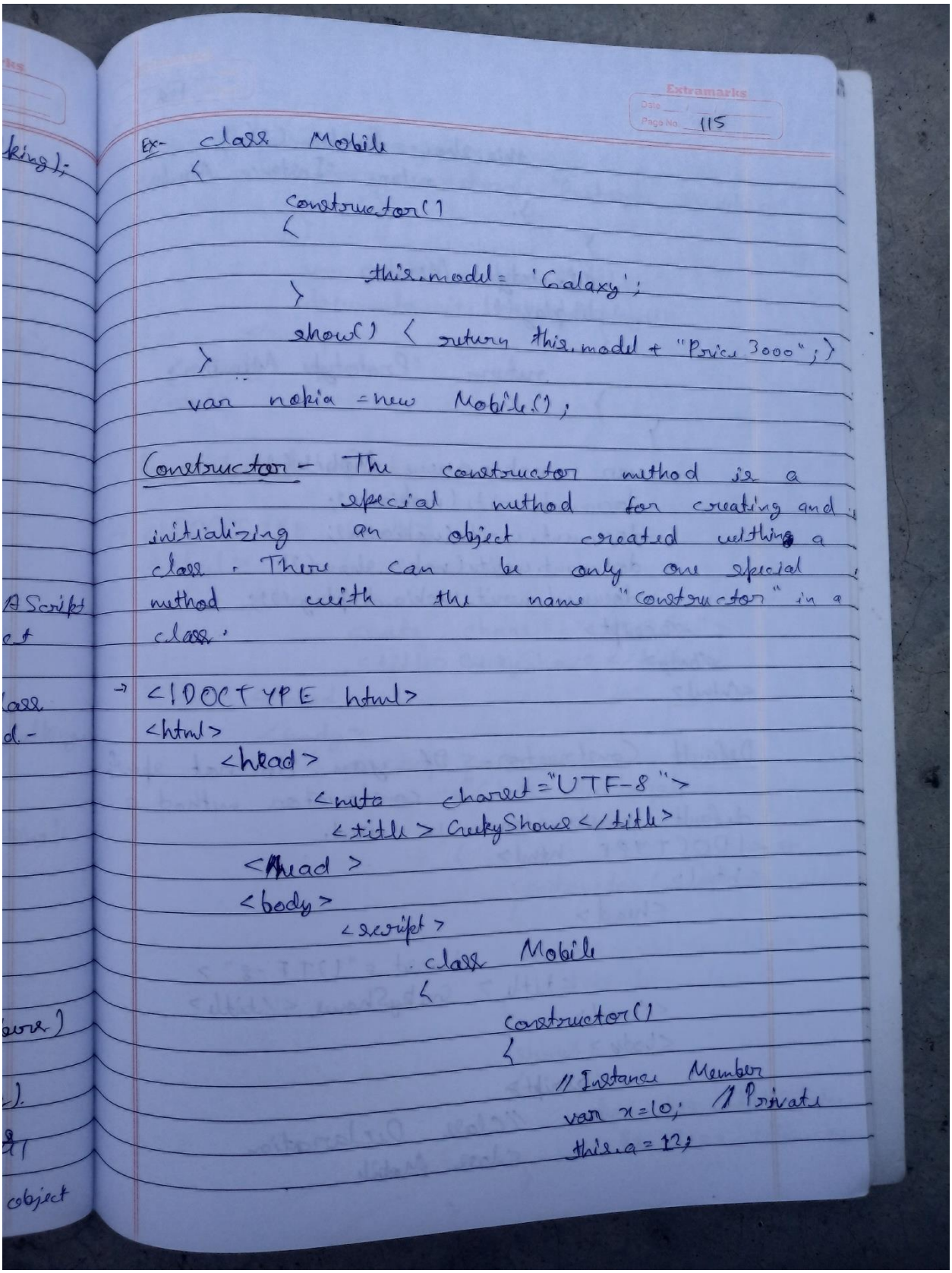
1. Class Declaration

Syntax-

```
class class_name
{
    constructor()
    {
        Properties (Instance Members)
    }
    Methods (Prototype Members).
}
```

Note:- Class_name का first letter Capital रखना चाहिये।

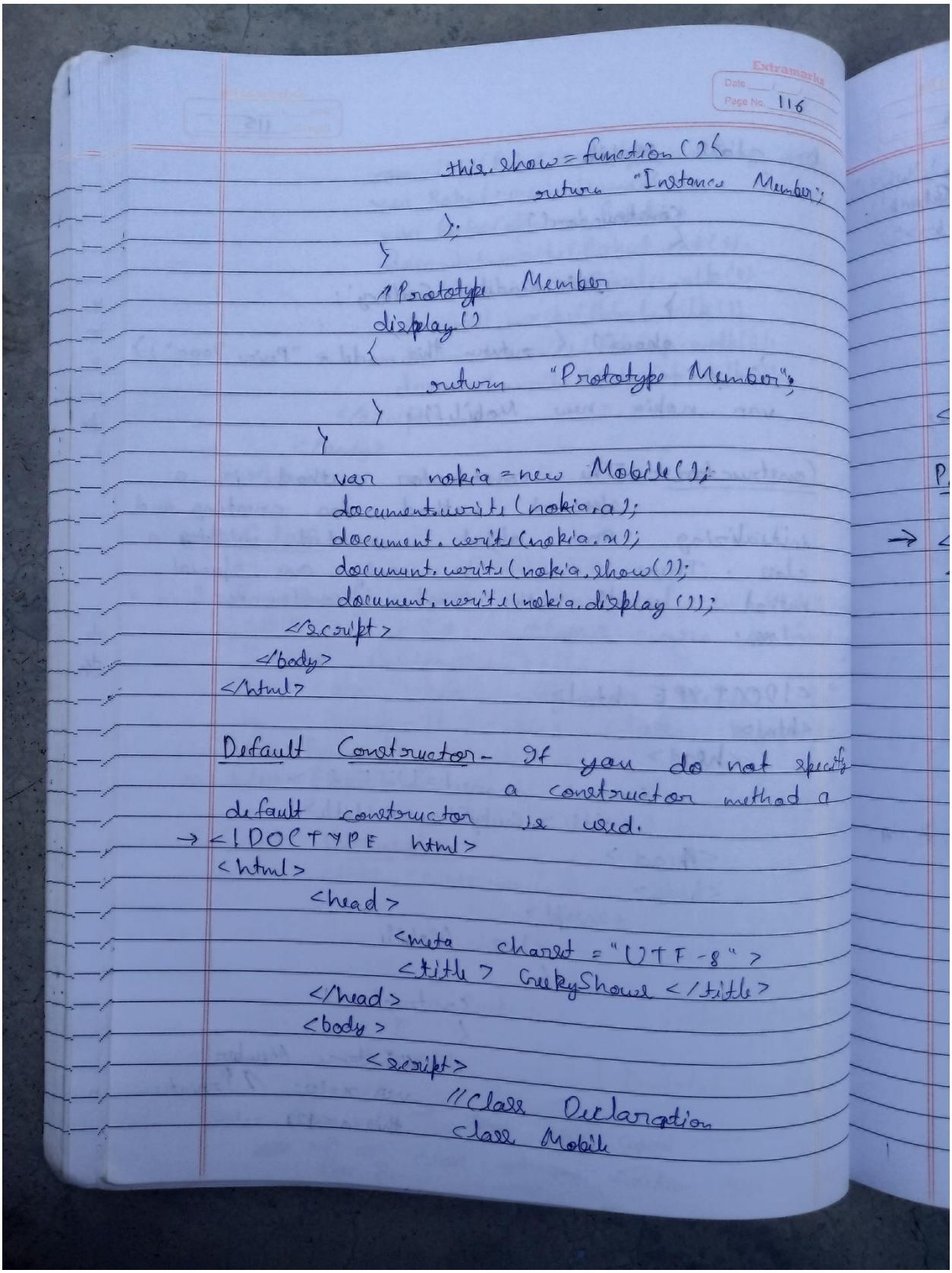
यहाँ पर class भी एक function type का object होता है।



Ex- class Mobile
{
constructor()
{
this.modul = 'Galaxy';
}
show() { return this.modul + "Price 3000"; }
}
var nokia = new Mobile();

Constructor - The constructor method is a special method for creating and initializing an object created within a class. There can be only one special method with the name "constructor" in a class.

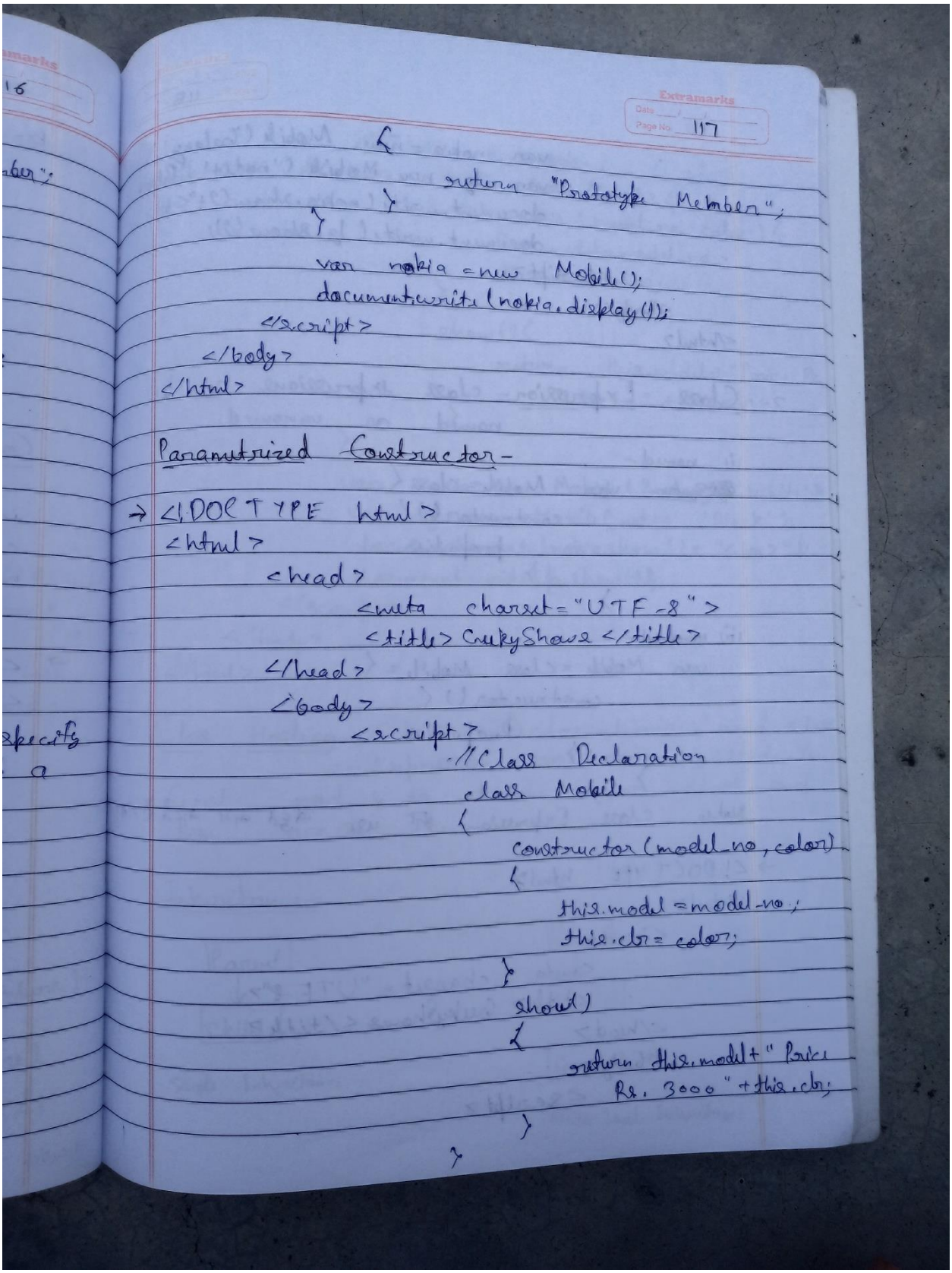
→ <!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title> GeekyShows </title>
</head>
<body>
<script>
class Mobile
{
constructor()
{
// Instance Member
var x=10; // Private
this.a=12;



```
        this.show = function(){
            return "Instance Member";
        };
    }
    //Prototype Member
    display()
    {
        return "Prototype Member";
    }
}
var nokia = new Mobile();
document.write(nokia.a);
document.write(nokia.x);
document.write(nokia.show());
document.write(nokia.display());
</script>
</body>
</html>
```

Default Constructor - If you do not specify a constructor method a default constructor is used.

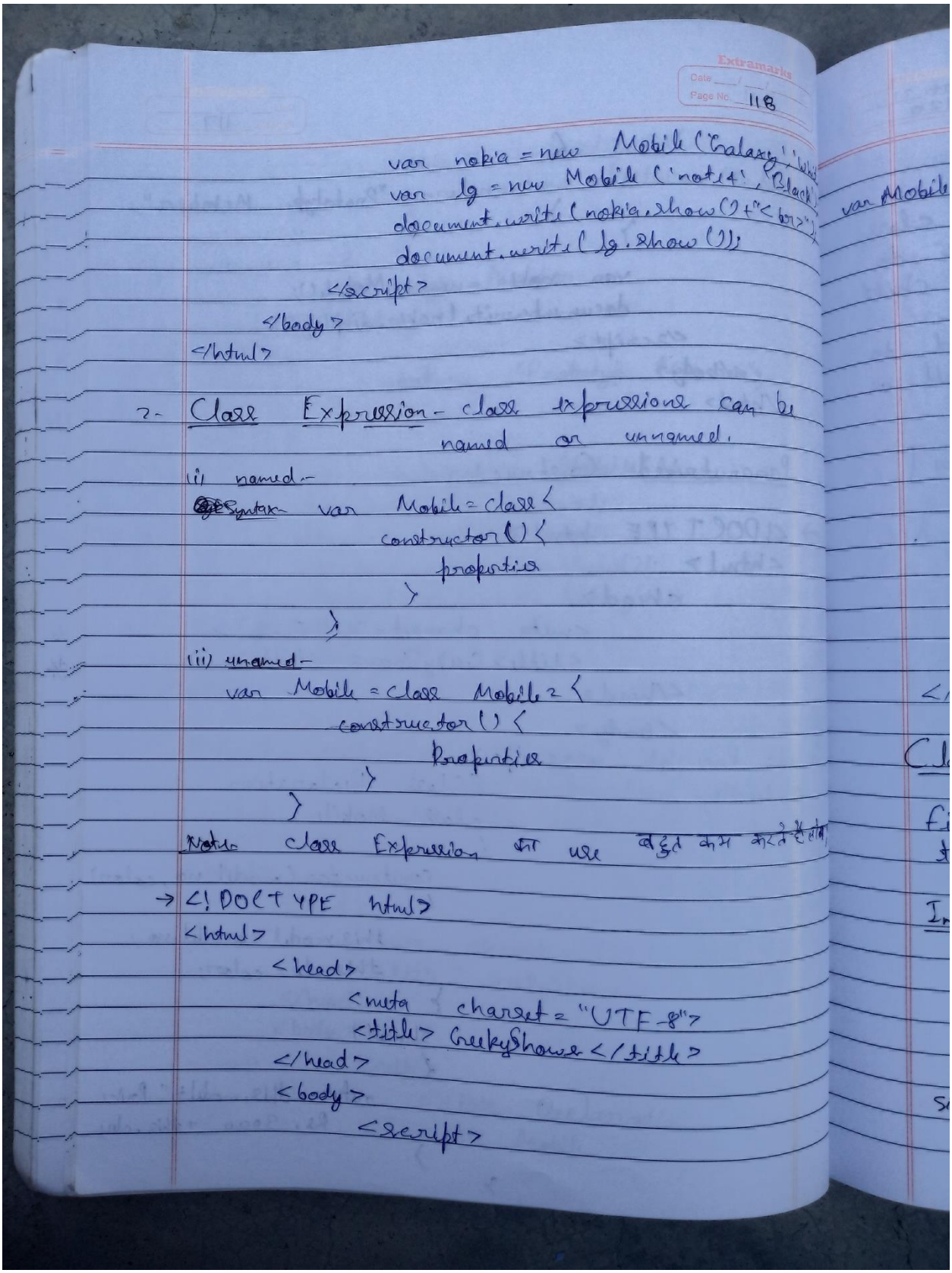
```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="UTF-8">
        <title> GeekyShows </title>
    </head>
    <body>
        <script>
            //Class Declaration
            class Mobile
```



```
        {
            return "Prototype Member";
        }
    }

    var nokia = new Mobile();
    document.write(nokia.display());
</script>
</body>
</html>
```

Paramatrized Constructor -

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="UTF-8">
        <title> CrukyShows </title>
    </head>
    <body>
        <script>
            // Class Declaration
            class Mobile
            {
                constructor(model_no, color)
                {
                    this.model = model_no;
                    this.clr = color;
                }
                show()
                {
                    return this.model + " Price
                        Rs. 3000 " + this.clr;
                }
            }
```



```
var nokia = new Mobile('Galaxy', 'White');
var lg = new Mobile('note4', 'Black');
document.write(nokia.show()+"<br>");
document.write(lg.show());
</script>
</body>
</html>
```

2. Class Expression - class expressions can be named or unnamed.

ii) named -

```
var Mobile = class {
    constructor() {
        properties
    }
}
```

iii) unnamed -

```
var Mobile = class Mobile2 {
    constructor() {
        Properties
    }
}
```

Note: class Expression का use बहुत कम करते हैं हम।

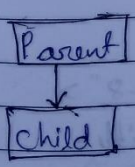
```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="UTF-8">
        <title> GeekyShows </title>
    </head>
    <body>
        <script>
```

// Class Expression

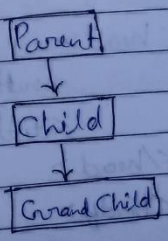
```
var Mobile = class Mobile{
    constructor(model_no, color){
        this.model = model_no;
        this.clr = color;
    }
    show(){
        return this.model + "Price Rs. 3000 " + this.clr;
    }
};
var nokia = new Mobile('Galaxy', 'White');
var lg = new Mobile('note4', 'Black');
document.write(nokia.show() + "<br>");
document.write(lg.show());
</script>
</body>
</html>
```

Class Hoisting - Class Declarations and class Expression are not hoisted. You first need to declare your class and then access it.

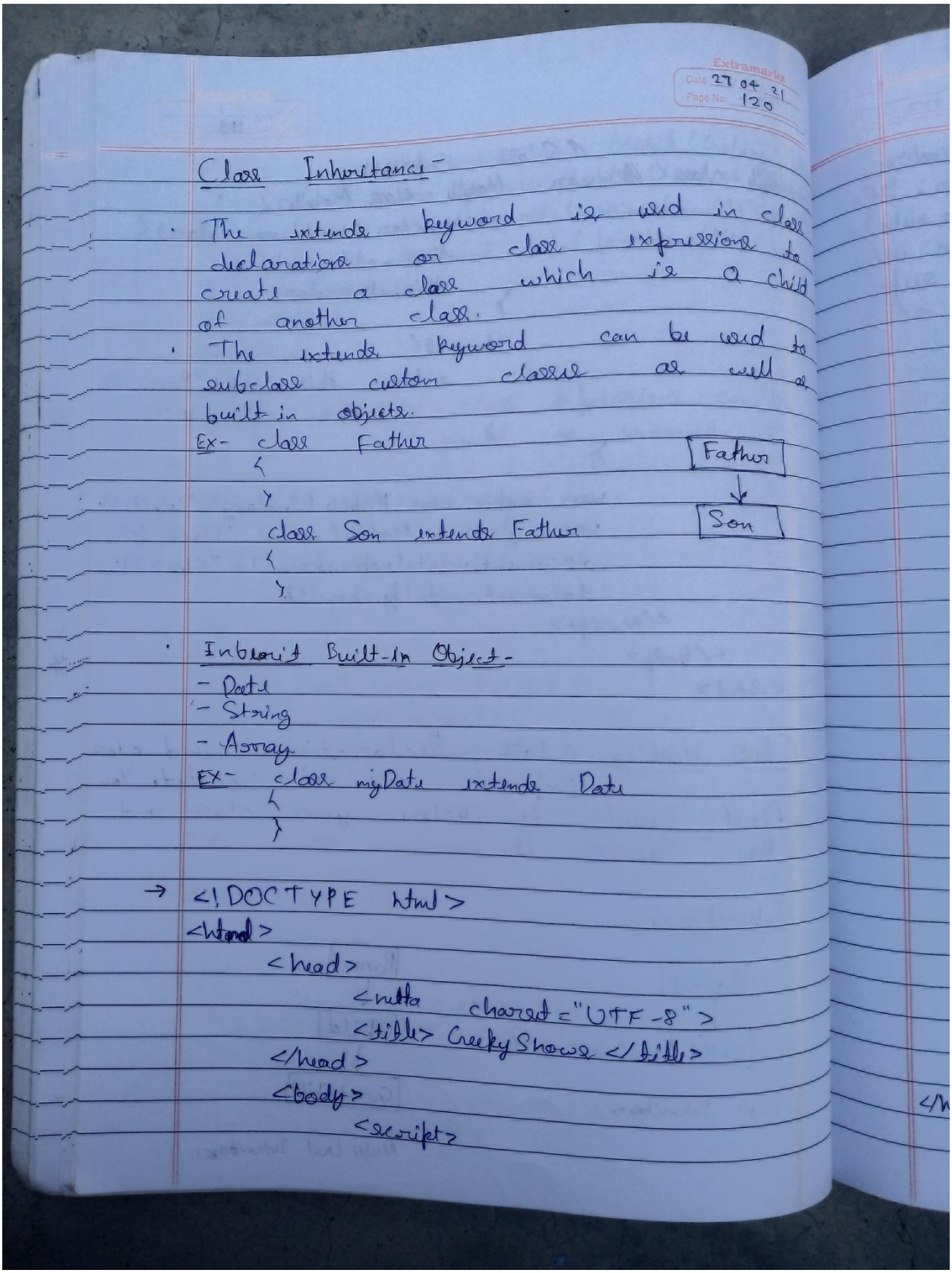
Inheritance -



Single Inheritance



Multi Level Inheritance



Extramarks
Date 27 04 21
Page No. 120

Class Inheritance -

- The extends keyword is used in class declarations or class expressions to create a class which is a child of another class.
- The extends keyword can be used to subclass custom classes as well as built-in objects.

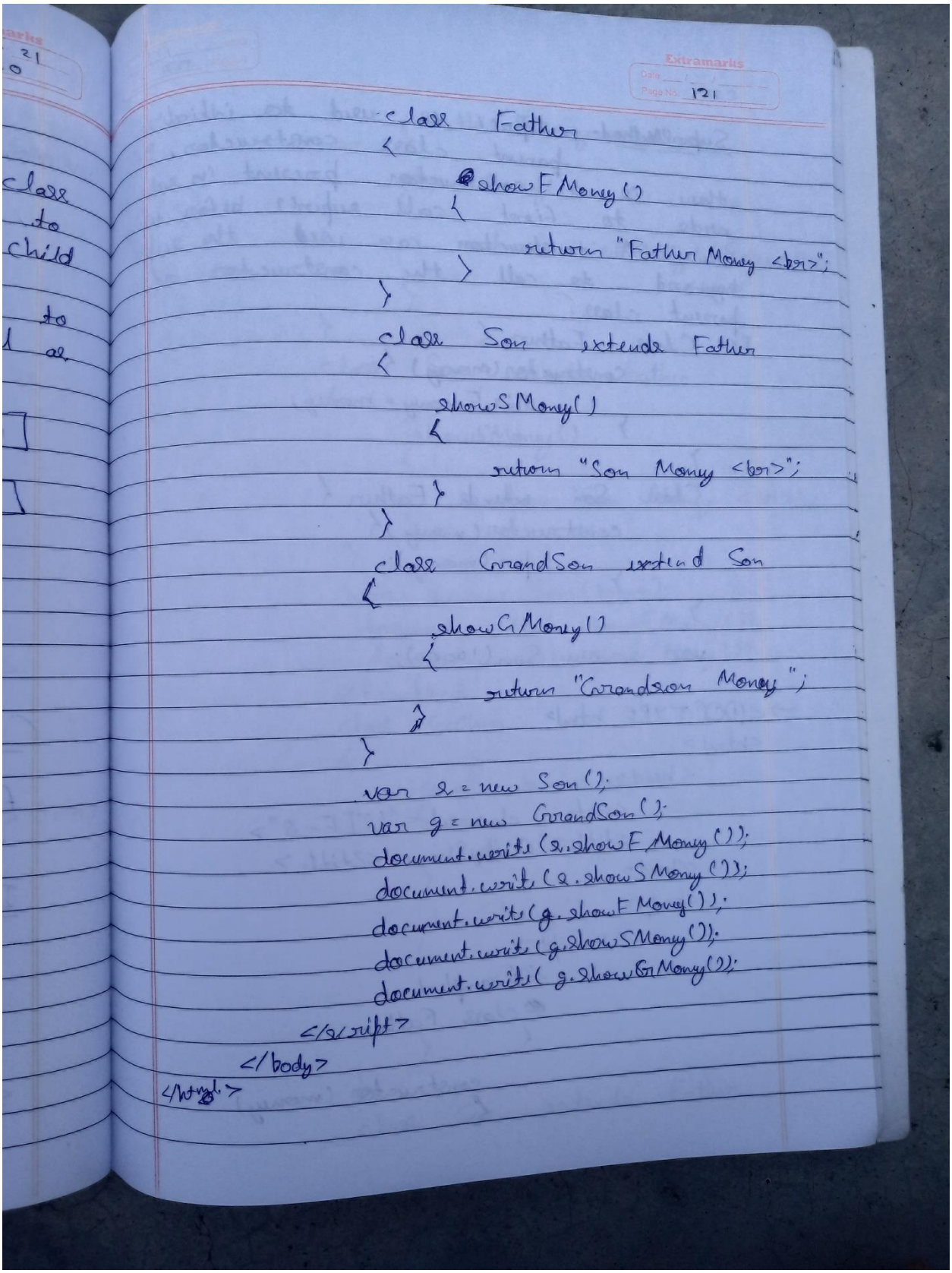
Ex- class Father
{
}
class Son extends Father
{
}

Father
↓
Son

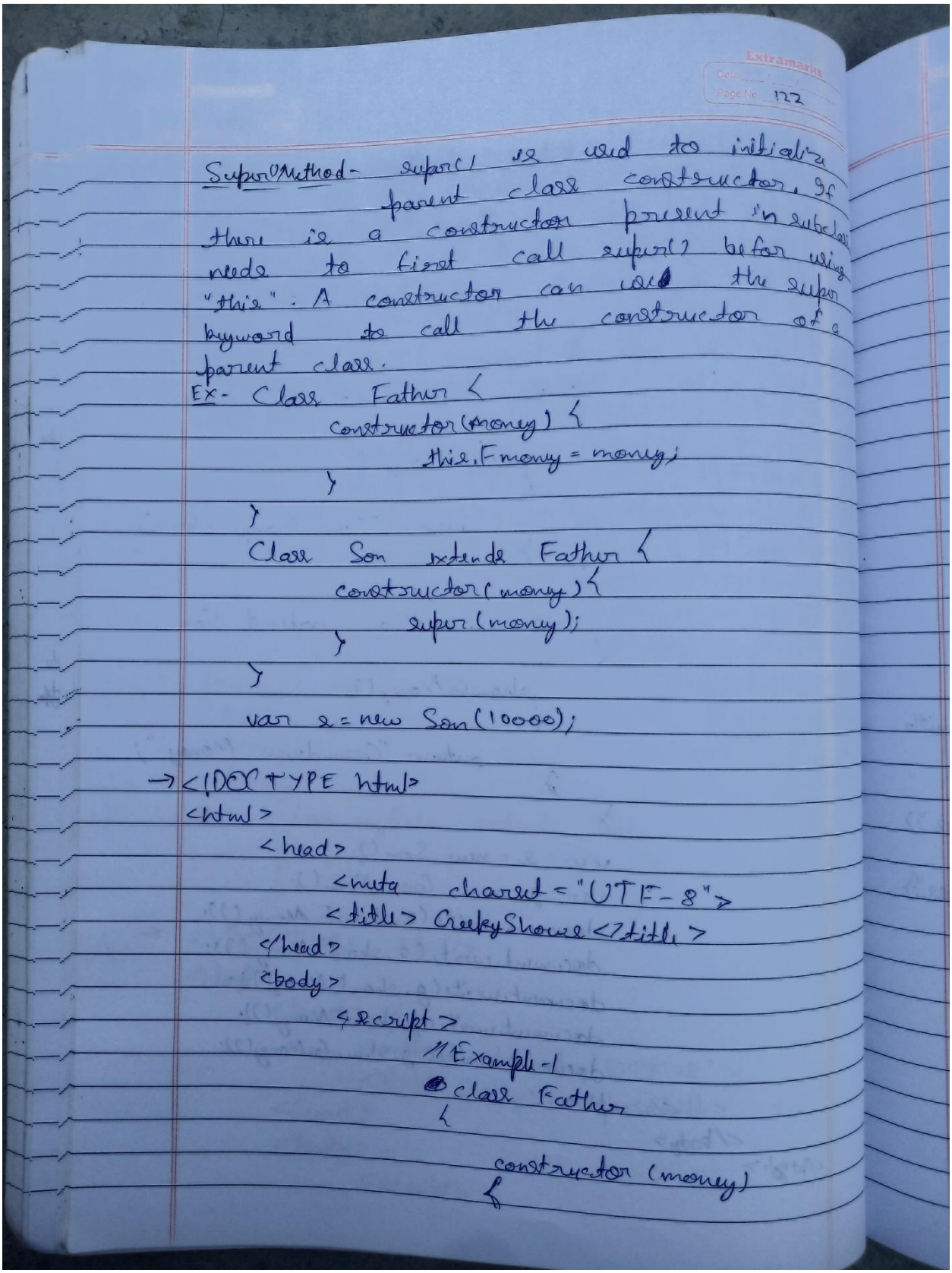
- Inbuilt Built-in Object -
  - Date
  - String
  - Array.

Ex- class myDate extends Date
{
}

→
```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="UTF-8">
        <title> Geeky Shows </title>
    </head>
    <body>
        <script>
```

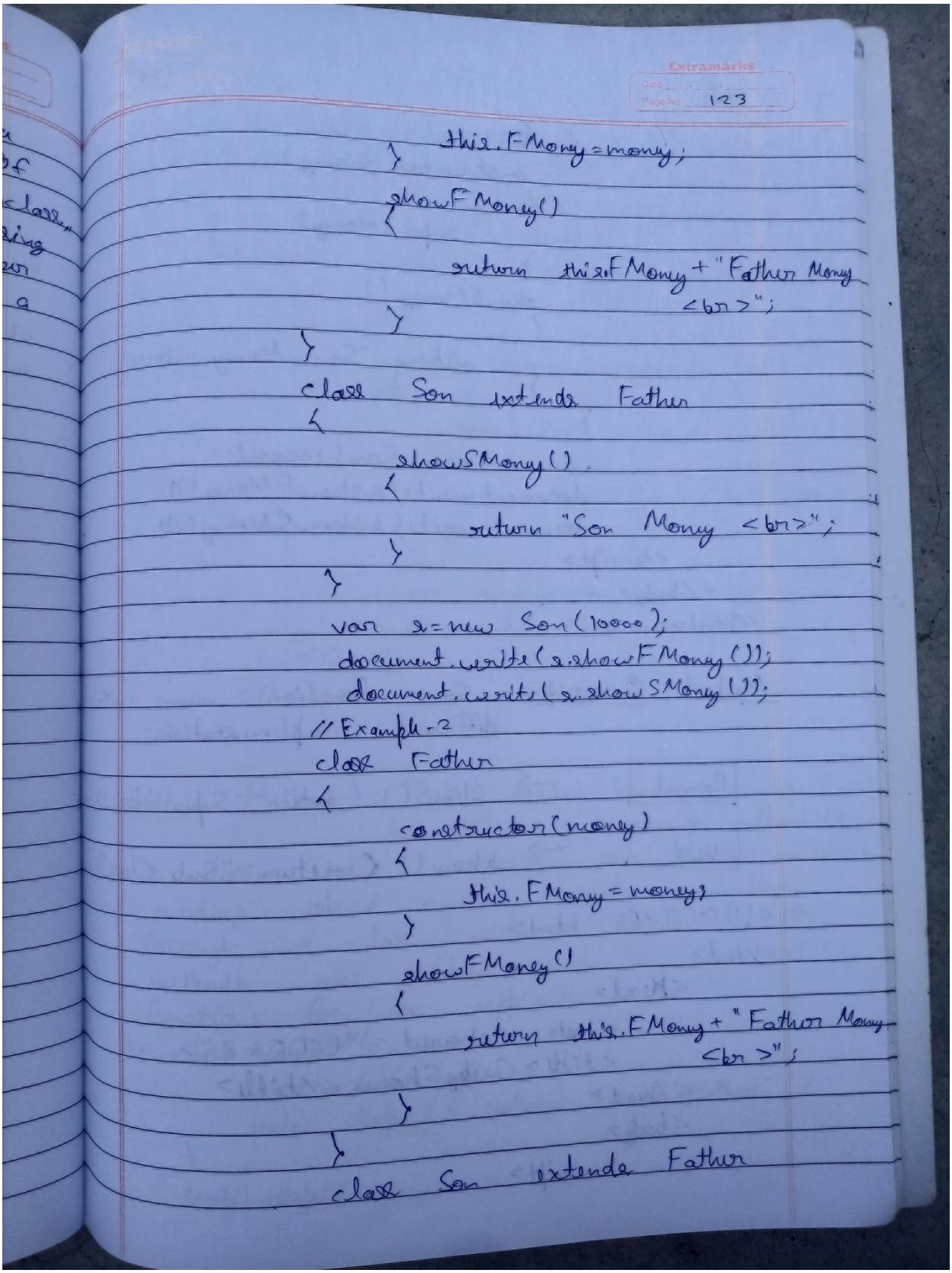


```
class Father
{
    showFMoney()
    {
        return "Father Money <br>";
    }
}
class Son extends Father
{
    showSMoney()
    {
        return "Son Money <br>";
    }
}
class GrandSon extend Son
{
    showGMoney()
    {
        return "Grandson Money";
    }
}
var s = new Son();
var g = new GrandSon();
document.write(s.showFMoney());
document.write(s.showSMoney());
document.write(g.showFMoney());
document.write(g.showSMoney());
document.write(g.showGrMoney());
</script>
</body>
</html>
```

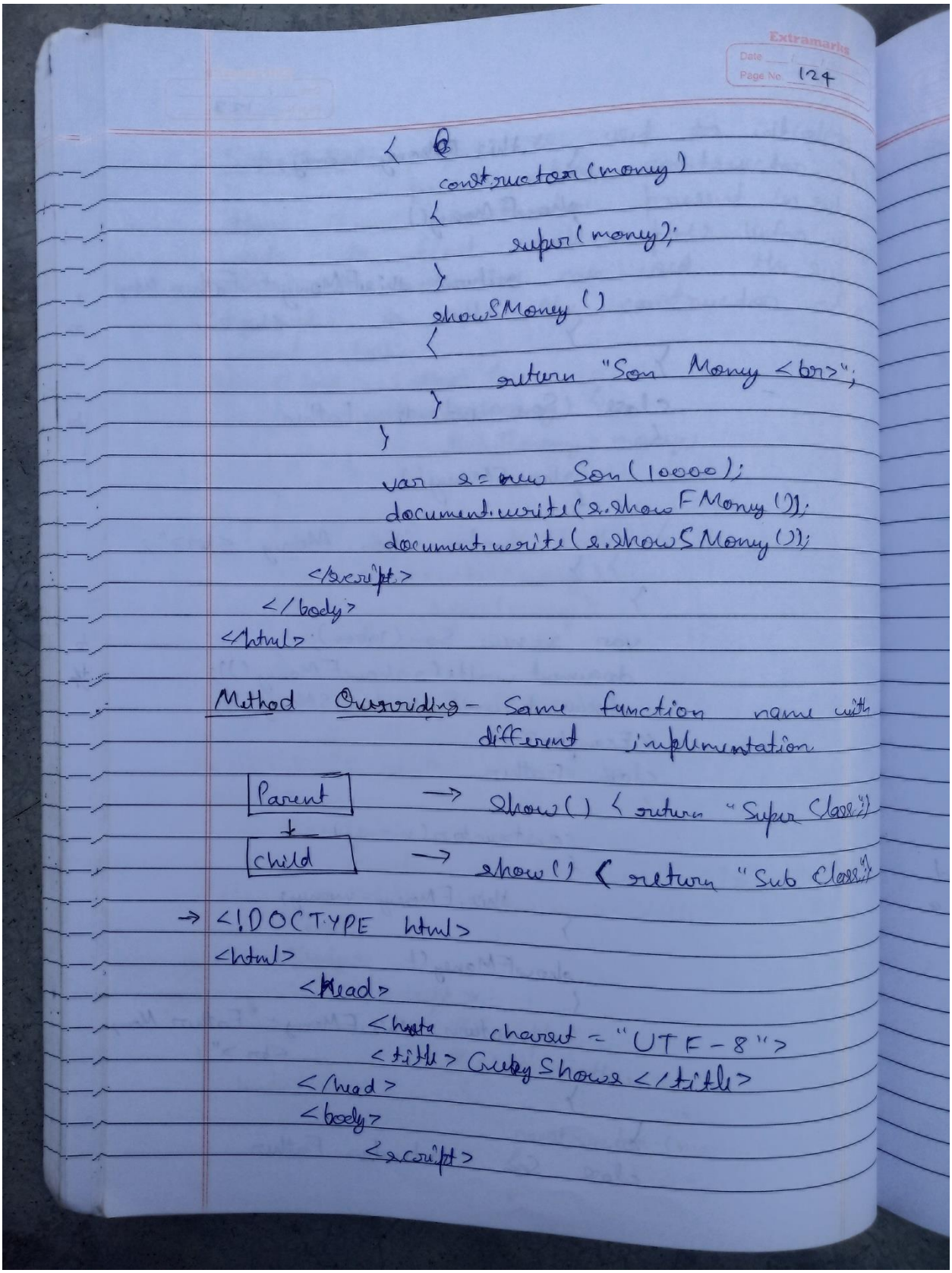


Super() Method - super() is used to initialize parent class constructor. If there is a constructor present in subclass needs to first call super() before using "this". A constructor can use the super keyword to call the constructor of a parent class.

```
Ex- Class Father {
        constructor(money) {
            this.Fmony = money;
        }
    }
    Class Son extends Father {
        constructor(mony) {
            super(mony);
        }
    }
    var s = new Son(10000);

→ <!DOCTYPE html>
  <html>
    <head>
      <meta charset="UTF-8">
      <title> GeekyShows </title>
    </head>
    <body>
      <script>
        //Example-1
        class Father
        {
          constructor(money)
          {
```



```
                this.FMony = money;
            }
            showFMoney()
            {
                return this.FMony + "Father Money
                                    <br>";
            }
        }
        class Son extends Father
        {
            showSMony()
            {
                return "Son Mony <br>";
            }
        }
        var s = new Son(10000);
        document.write(s.showFMony());
        document.write(s.showSMony());
        // Example-2
        class Father
        {
            constructor(money)
            {
                this.FMony = money;
            }
            showFMoney()
            {
                return this.FMony + "Father Money
                                    <br>";
            }
        }
        class Son extends Father
```



```
class A
    constructor(money)
    {
        super(money);
    }
    showSMoney()
    {
        return "Son Money <br>";
    }
}
var s = new Son(10000);
document.write(s.showFMony());
document.write(s.showSMony());
</script>
</body>
</html>
```

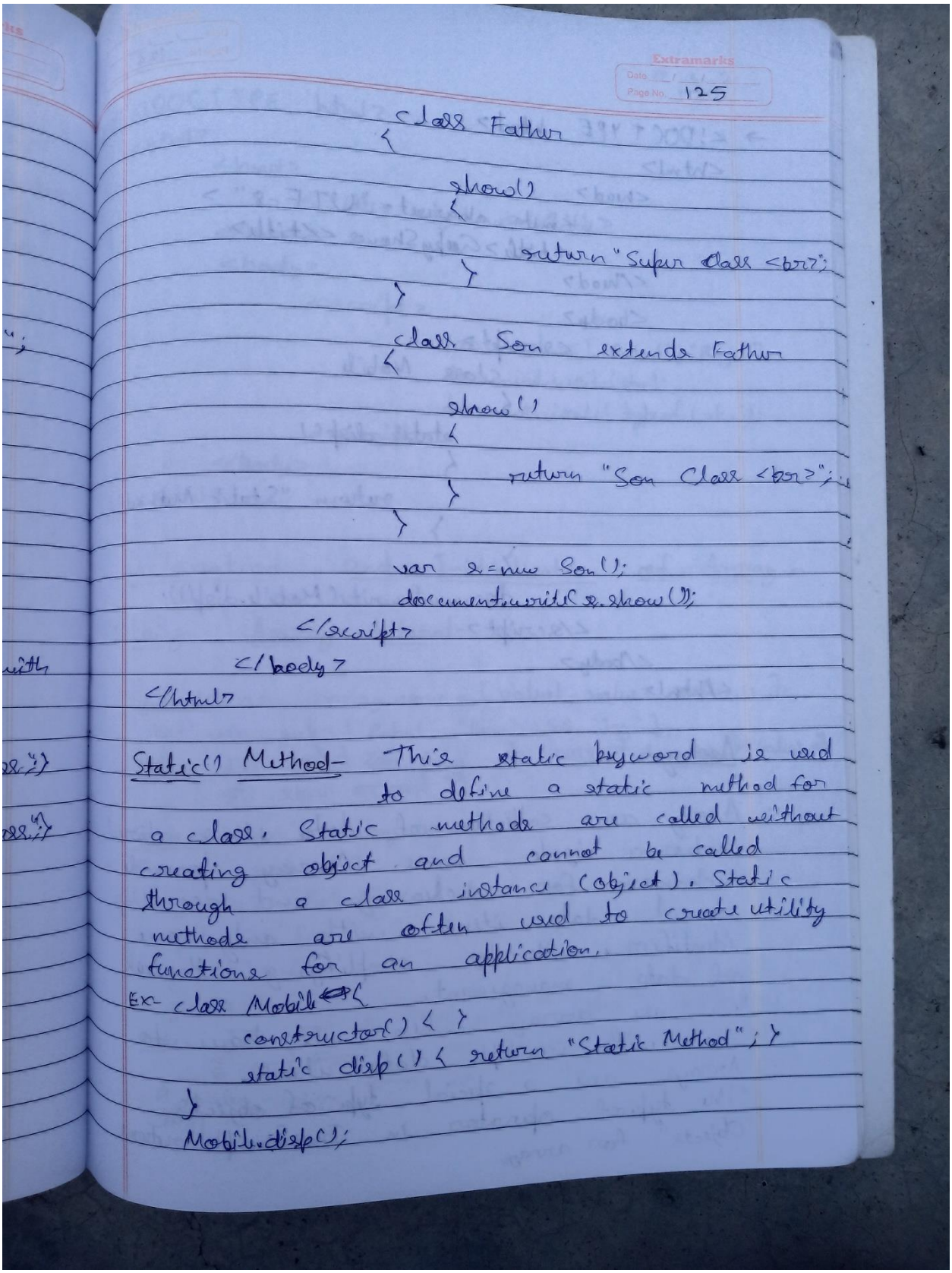
Method Overriding - Same function name with different implementation

Parent → show() { return "Super Class"; }

+

child → show() { return "Sub Class"; }

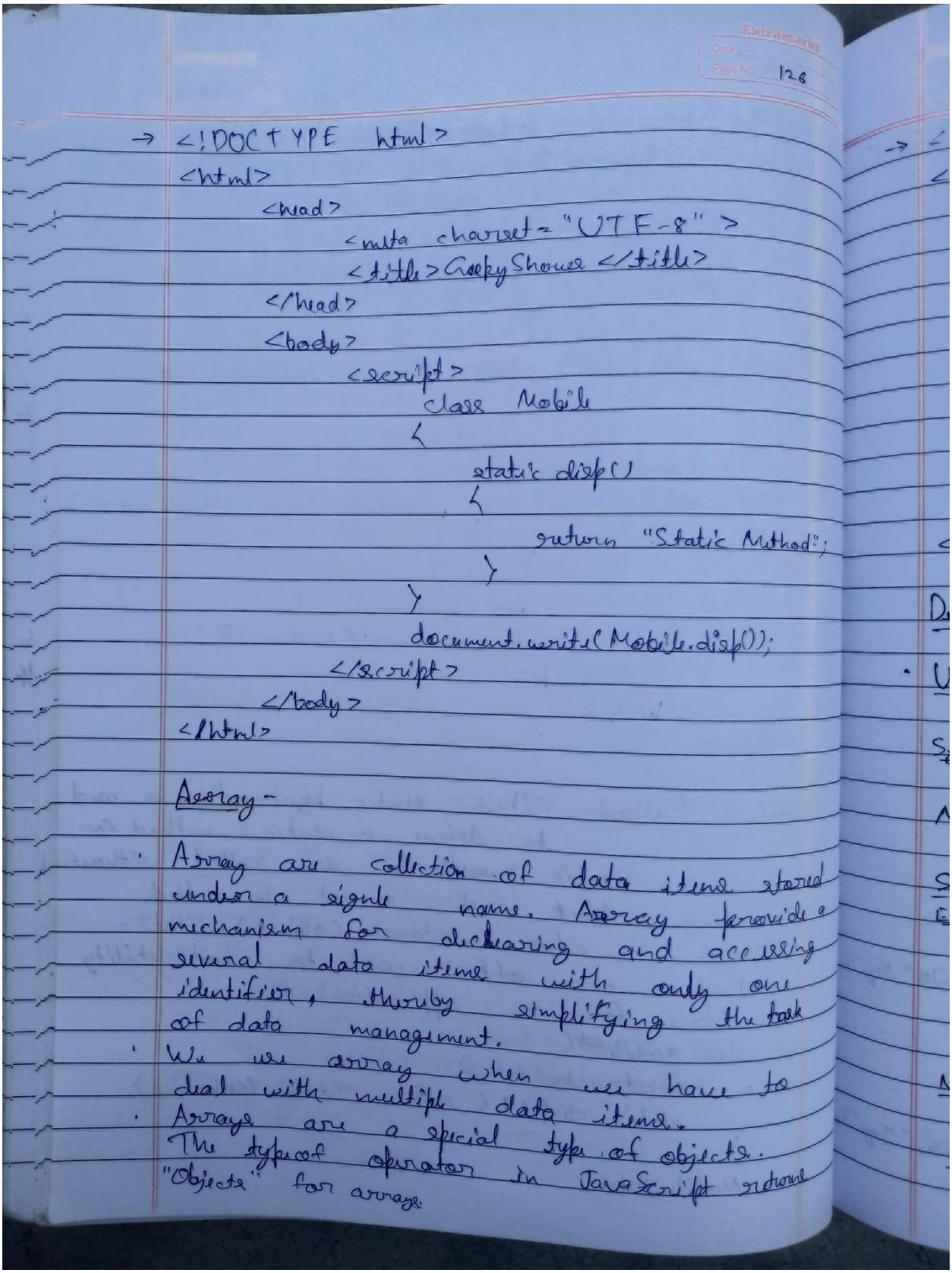
```
<!DOCTYPE html>
<html>
    <head>
        <meta charset = "UTF-8">
        <title> Geeky Shows </title>
    </head>
    <body>
        <script>
```



```
class Father
{
    show()
    {
        return "Super class <br>";
    }
}
class Son extends Father
{
    show()
    {
        return "Son Class <br>";
    }
}
var s=new Son();
document.write(s.show());
</script>
</body>
</html>
```

Static() Method- This static keyword is used to define a static method for a class. Static methods are called without creating object and cannot be called through a class instance (object). Static methods are often used to create utility functions for an application.

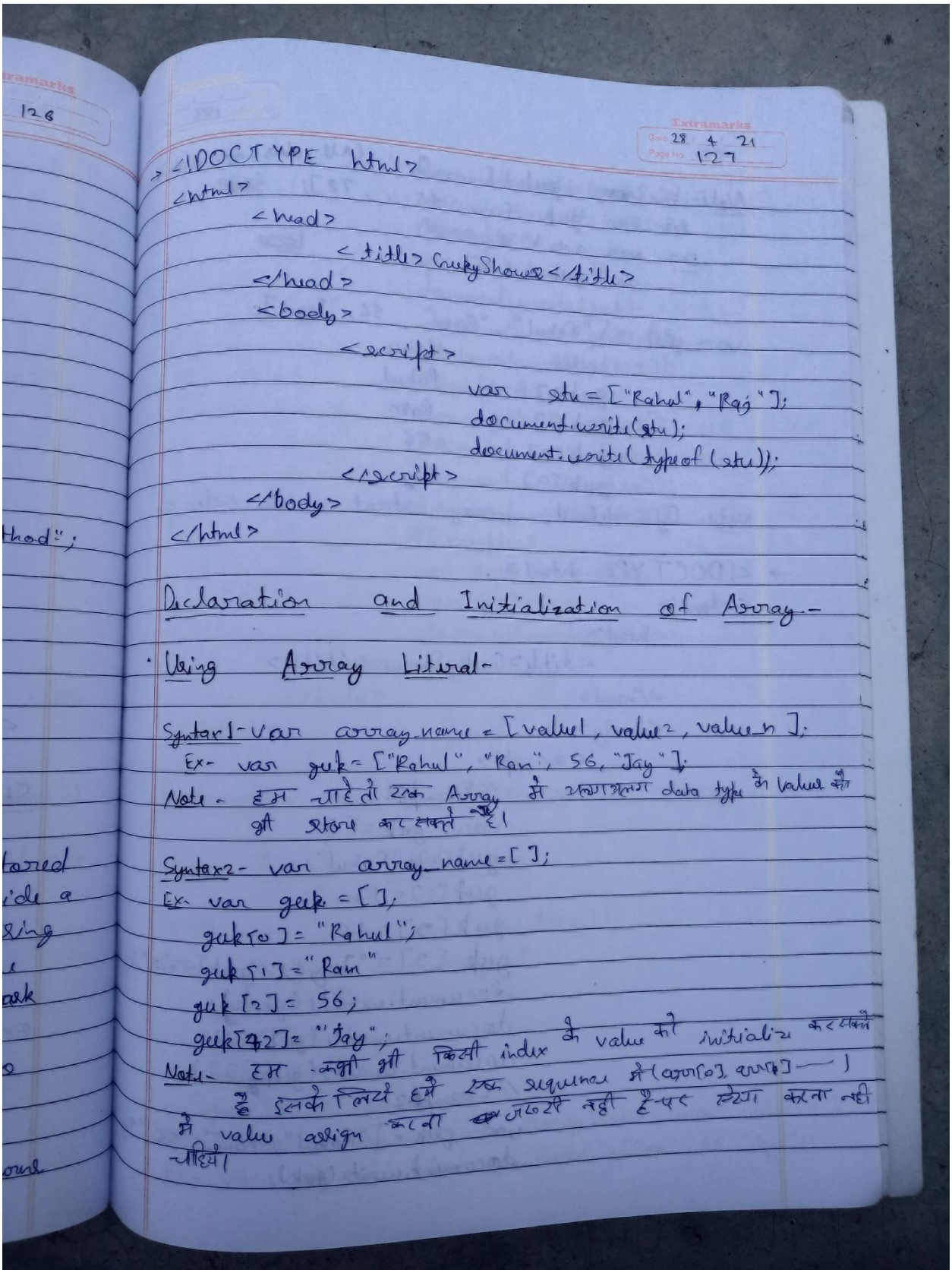
```
Ex class Mobile{
    constructor(){ }
    static disp(){ return "Static Method"; }
}
Mobile.disp();
```



```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="UTF-8">
        <title>Geeky Shows</title>
    </head>
    <body>
        <script>
            class Mobile
            {
                static disp()
                {
                    return "Static Method";
                }
            }
            document.write(Mobile.disp());
        </script>
    </body>
</html>
```

Array -

- Array are collection of data item stored under a single name. Array provide a mechanism for declaring and accessing several data item with only one identifier, thereby simplifying the task of data management.
- We use array when we have to deal with multiple data items.
- Arrays are a special type of objects. The typeof operator in JavaScript returns "Object" for arrays.



```
<!DOCTYPE html>
<html>
    <head>
        <title> CrukyShows </title>
    </head>
    <body>
        <script>
            var stu = ["Rahul", "Raj"];
            document.write(stu);
            document.write(typeof(stu));
        </script>
    </body>
</html>
```

Declaration and Initialization of Array -

- Using Array Literal -

Syntax1 - var array_name = [value1, value2, value_n];

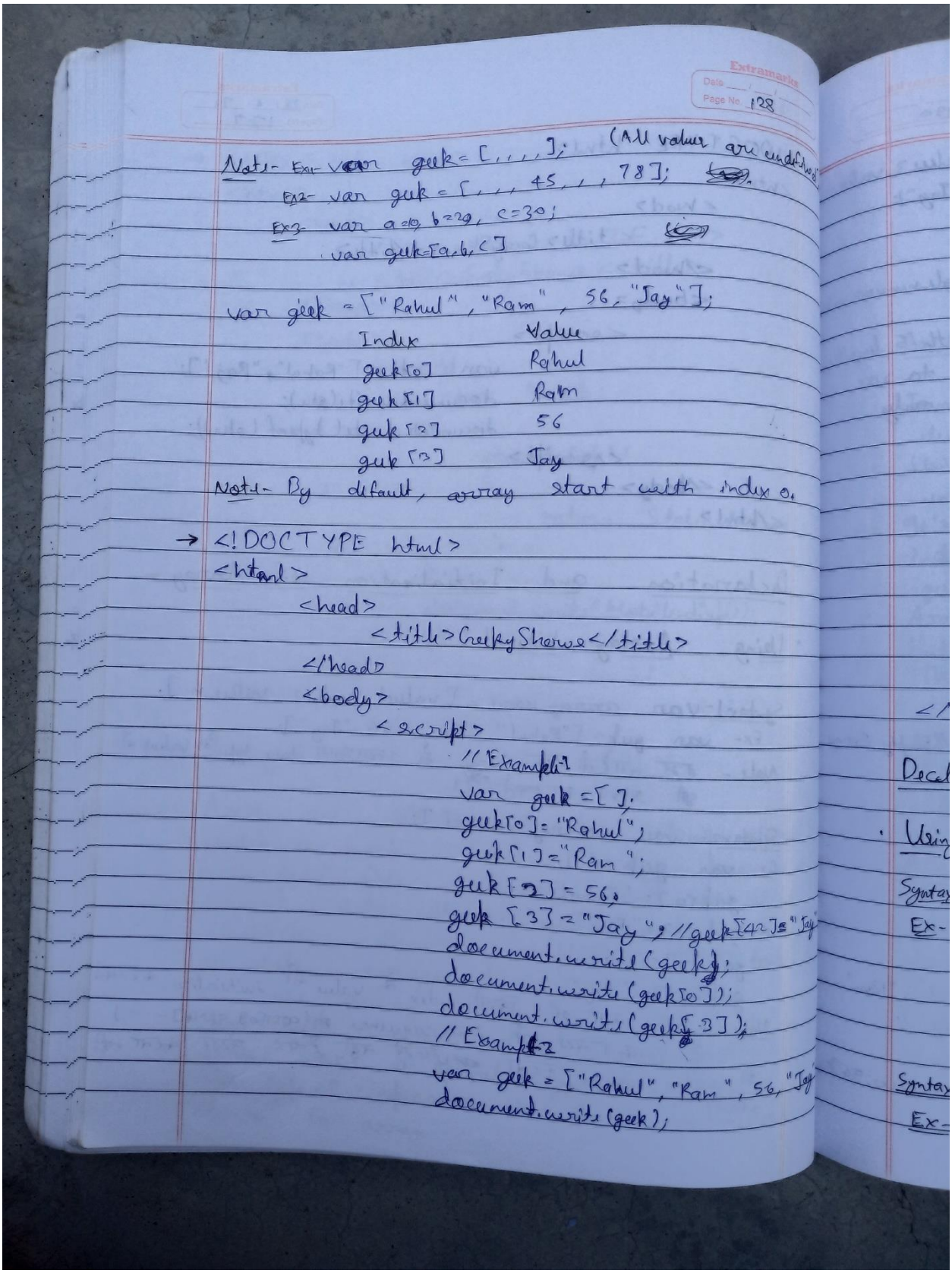
Ex- var geek = ["Rahul", "Ram", 56, "Jay"];

Note - हम चाहे तो एक Array में अलग-अलग data type के value को भी store कर सकते है।

Syntax2 - var array_name = [];

Ex. var geek = [];
geek[0] = "Rahul";
geek[1] = "Ram"
geek[2] = 56;
geek[42] = "Jay";

Note - हम कहीं भी किसी index के value को initialize कर सकते है इसके लिये हमें एक sequence में (arr[0], arr[1] ---) में value assign करना जरूरी नहीं है पर ऐसा करना नही चाहिये।



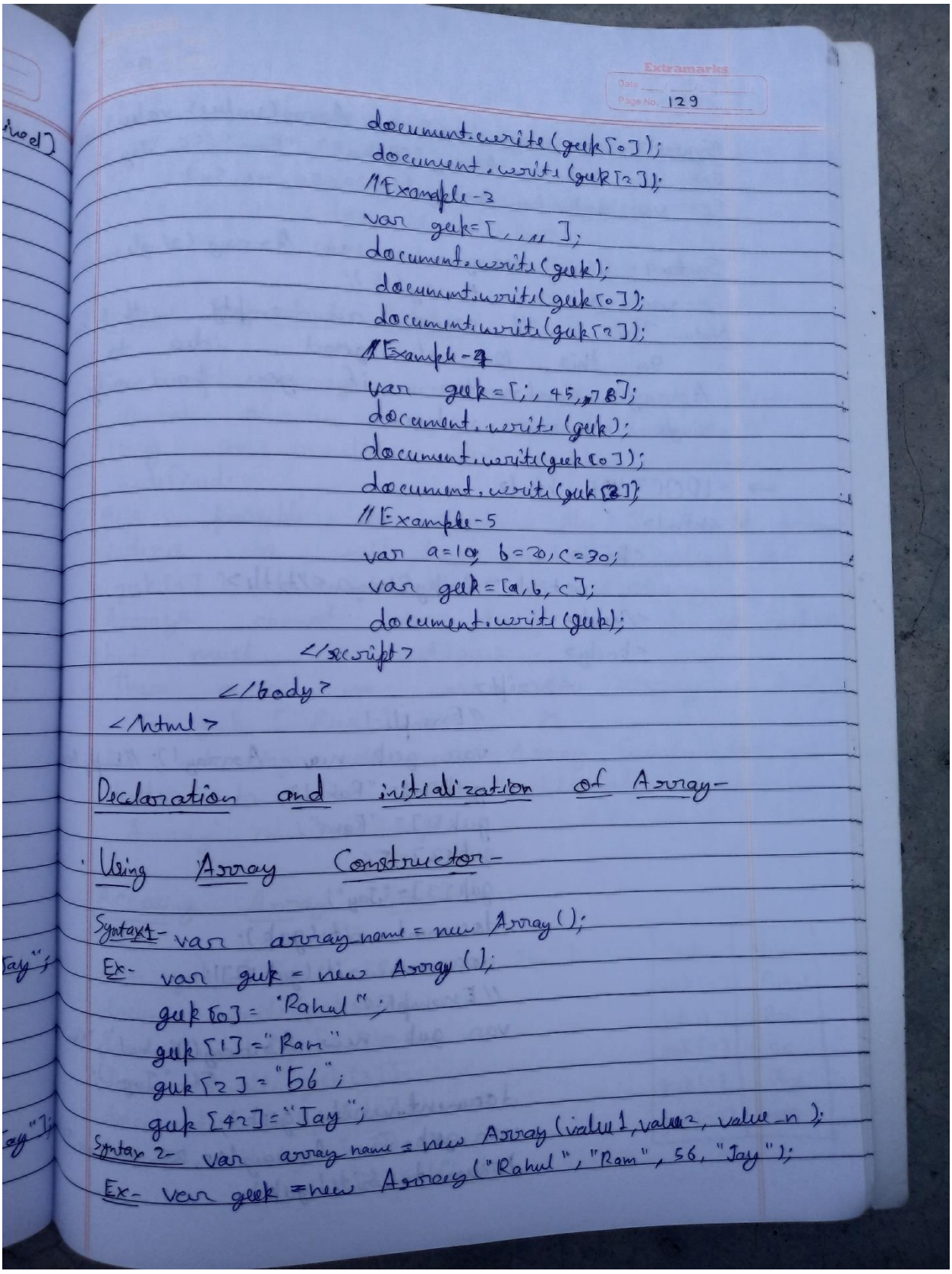
Note- Ex- var geek = [, , ,]; (All value are undefined)
Ex2- var guk = [, , , 45, , , 78];
Ex3- var a=10, b=20, c=30;
var guk=[a,b,c]

var geek = ["Rahul", "Ram", 56, "Jay"];

Index	Value
geek[0]	Rahul
geek[1]	Ram
guk[2]	56
guk[3]	Jay

Note- By default, array start with index 0.

→ <!DOCTYPE html>
<html>
<head>
<title>GeekyShows</title>
</head>
<body>
<script>
// Example-1
var geek = [];
geek[0] = "Rahul";
geek[1] = "Ram";
geek[2] = 56;
geek[3] = "Jay"; //geek[4] = "Jay"
document.write(geek);
document.write(geek[0]);
document.write(geek[3]);
// Example-2
var geek = ["Rahul", "Ram", 56, "Jay"]
document.write(geek);



```
document.write(geek[0]);
document.write(geek[2]);
//Example-3
var geek=[ , , , ];
document.write(geek);
document.write(geek[0]);
document.write(geek[2]);
//Example-4
var geek=[ , 45, 78];
document.write(geek);
document.write(geek[0]);
document.write(geek[2]);
//Example-5
var a=10, b=20, c=30;
var geek=[a,b,c];
document.write(geek);
</script>
</body>
</html>
```

Declaration and initialization of Array-

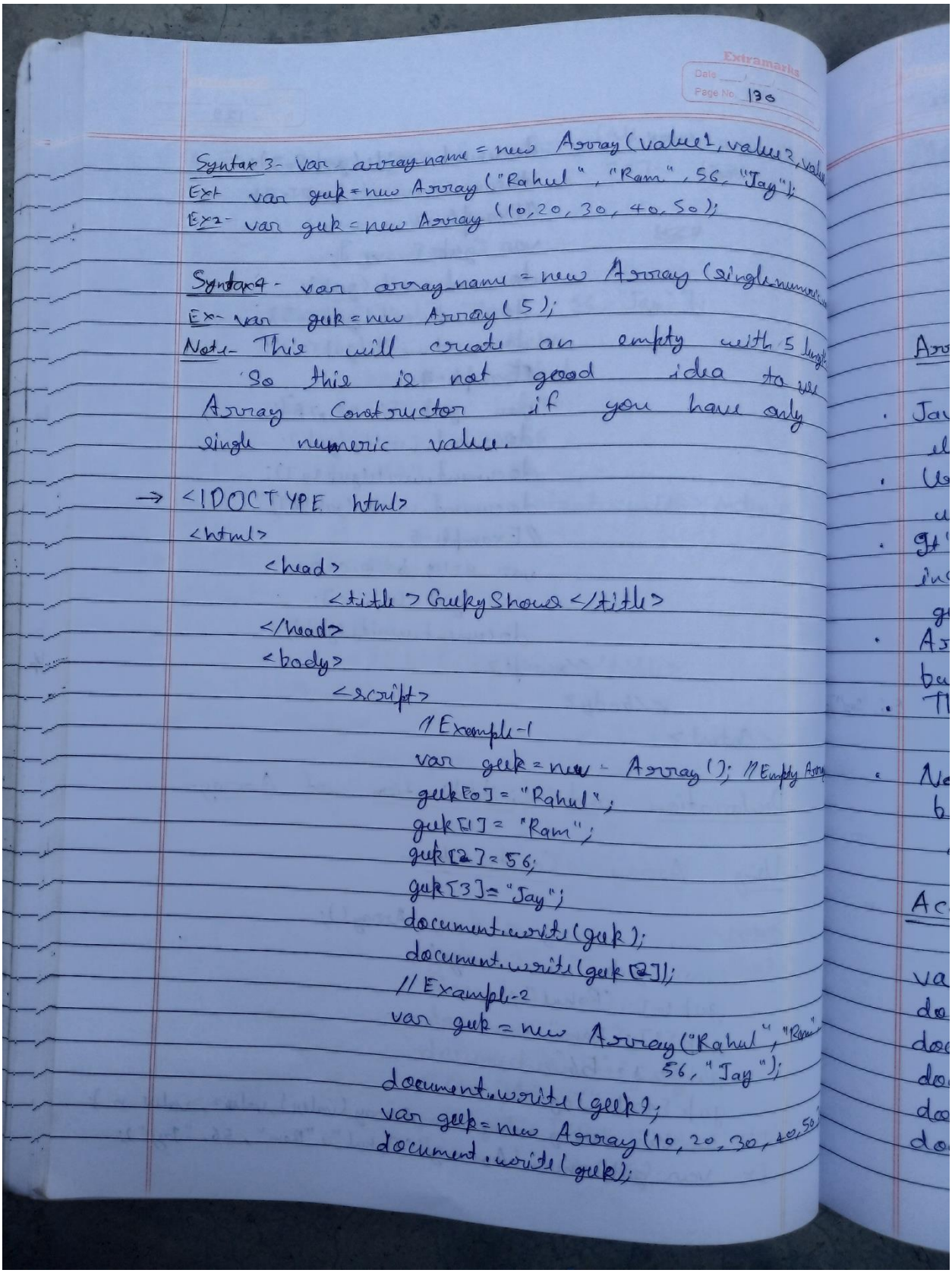
Using Array Constructor-

Syntax1- var arrayname = new Array();

Ex- var geek = new Array();
geek[0] = "Rahul";
geek[1] = "Ram"
geek[2] = "56";
geek[42] = "Jay";

Syntax 2- var arrayname = new Array(value1, value2, value-n);

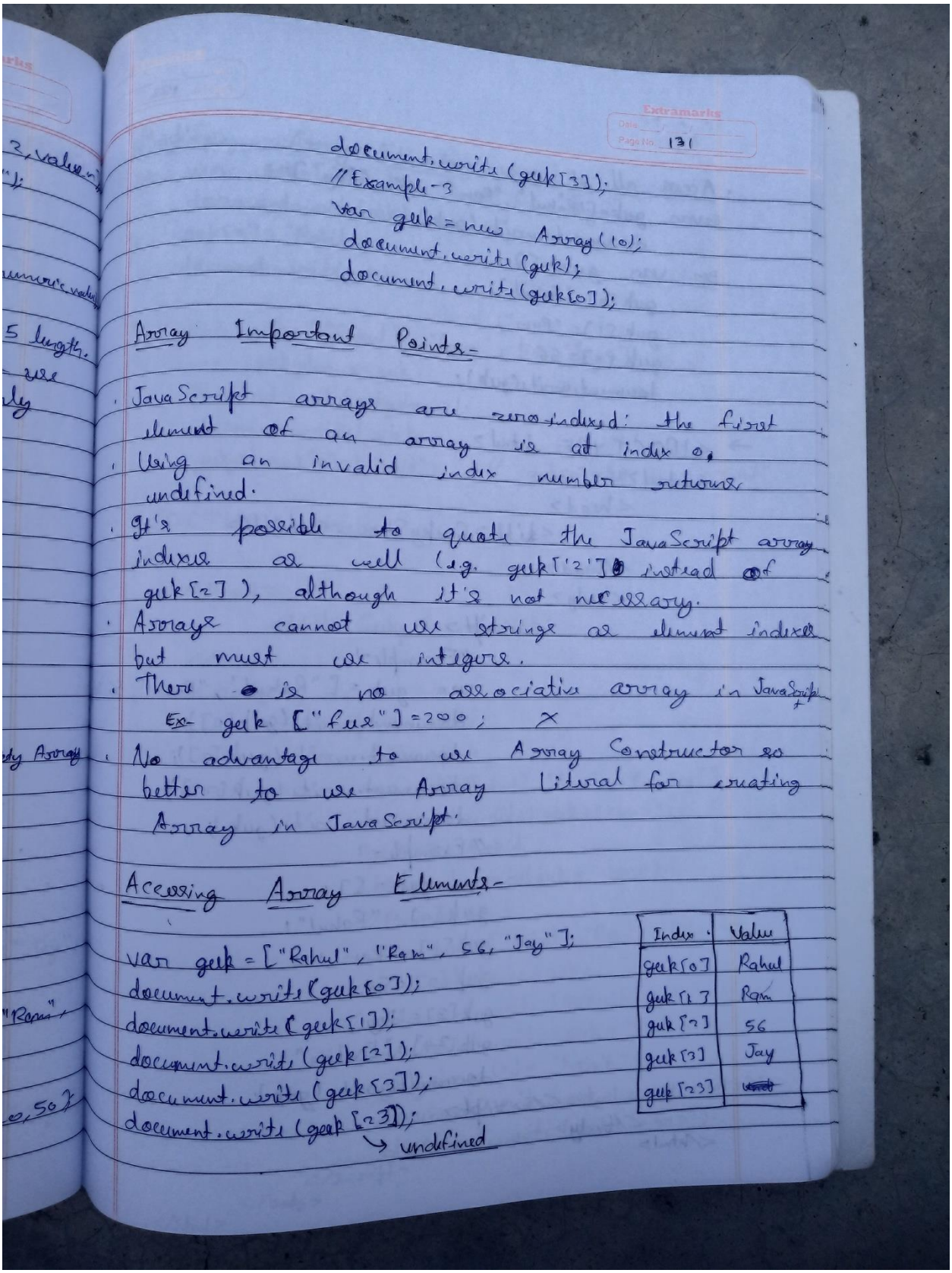
Ex- var geek = new Array("Rahul", "Ram", 56, "Jay");



Syntax 3- var array-name = new Array(value1, value2, value
Ex- var geek = new Array("Rahul", "Ram", 56, "Jay");
Ex2- var geek = new Array(10, 20, 30, 40, 50);

Syntax 4- var array-name = new Array(single numeric
Ex- var geek = new Array(5);
Note- This will create an empty with 5 length
So this is not good idea to use Array Constructor if you have only single numeric value.

→
```
<!DOCTYPE html>
<html>
    <head>
        <title> Geeky Shows </title>
    </head>
    <body>
        <script>
            // Example-1
            var geek = new Array(); // Empty Array
            geek[0] = "Rahul";
            geek[1] = "Ram";
            geek[2] = 56;
            geek[3] = "Jay";
            document.write(geek);
            document.write(geek[2]);
            // Example-2
            var geek = new Array("Rahul", "Ram",
                                 56, "Jay");
            document.write(geek);
            var geek = new Array(10, 20, 30, 40, 50);
            document.write(geek);
```



```
document.write(geek[3]);
//Example-3
var geek = new Array(10);
document.write(geek);
document.write(geek[0]);
```

Array Important Points-

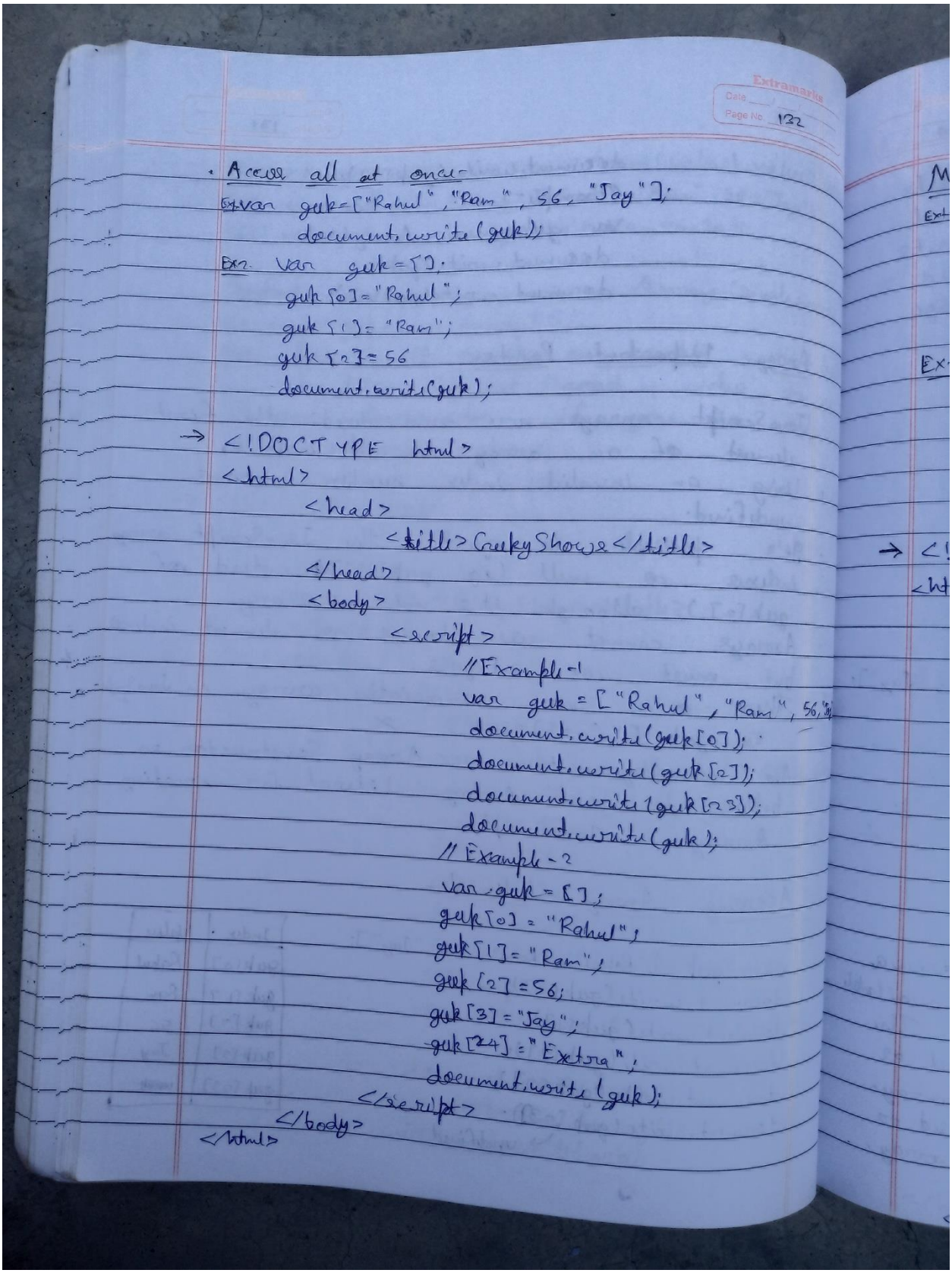
- JavaScript arrays are zero-indexed: the first element of an array is at index 0.
- Using an invalid index number returns undefined.
- It's possible to quote the JavaScript array indexes as well (e.g. geek['2'] instead of geek[2]), although it's not necessary.
- Arrays cannot use strings as element indexes but must use integers.
- There is no associative array in JavaScript
  Ex- geek["fun"] = 200; ✗
- No advantage to use Array Constructor so better to use Array Literal for creating Array in JavaScript.

Accessing Array Elements-

```
var geek = ["Rahul", "Ram", 56, "Jay"];
document.write(geek[0]);
document.write(geek[1]);
document.write(geek[2]);
document.write(geek[3]);
document.write(geek[23]);
```

→ undfined

Index	Value
geek[0]	Rahul
geek[1]	Ram
geek[2]	56
geek[3]	Jay
geek[23]	undefined

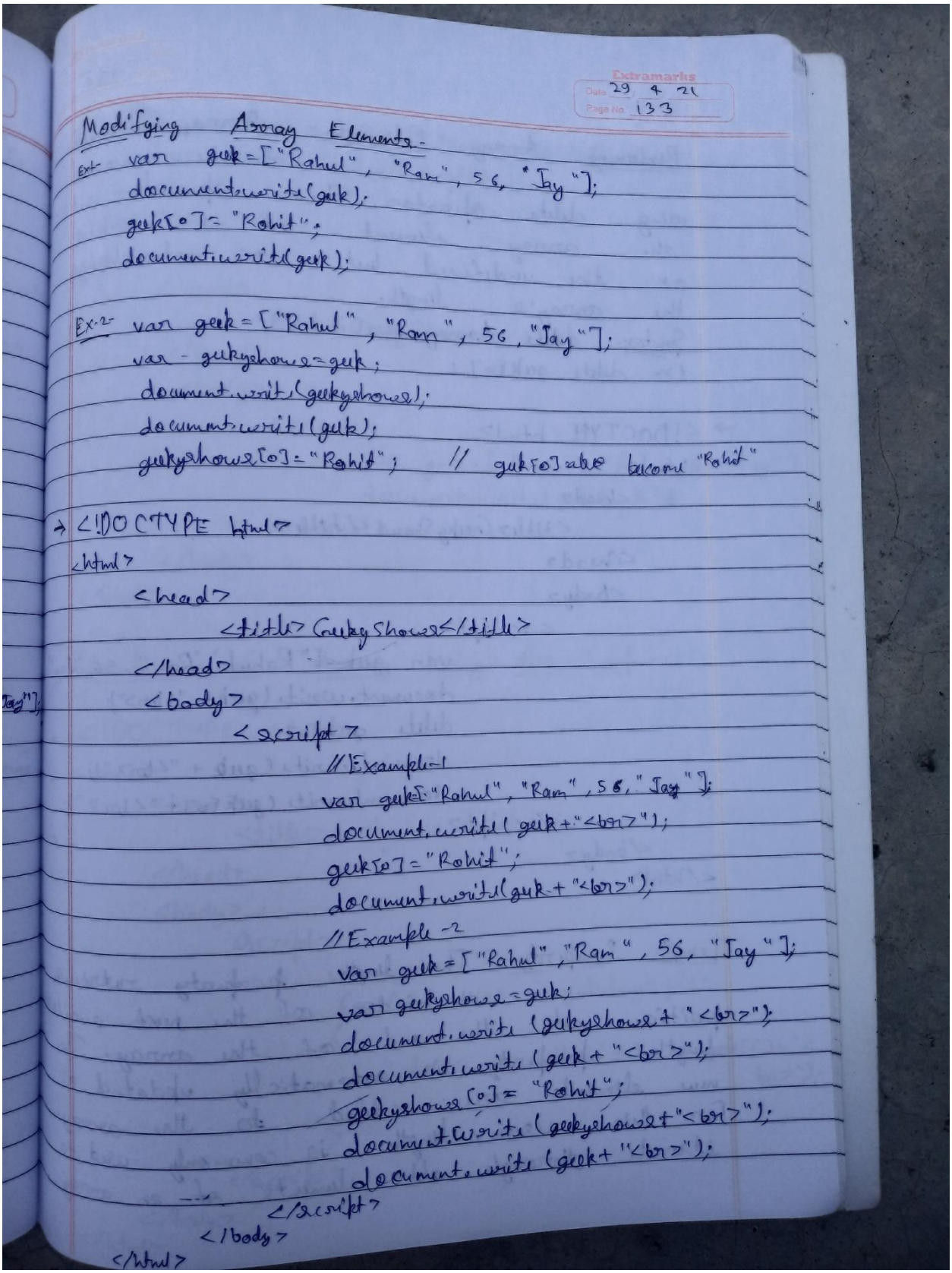


• Access all at once-

Ex var guk=["Rahul","Ram",56,"Jay"];
document.write(guk);

Ex. var guk=[];
guk[0]="Rahul";
guk[1]="Ram";
guk[2]=56
document.write(guk);

→ <!DOCTYPE html>
<html>
<head>
<title>GeekyShows</title>
</head>
<body>
<script>
//Example-1
var guk=["Rahul","Ram",56,"Jay"];
document.write(guk[0]);
document.write(guk[2]);
document.write(guk[23]);
document.write(guk);
//Example-2
var guk=[];
guk[0]="Rahul";
guk[1]="Ram";
guk[2]=56;
guk[3]="Jay";
guk[24]="Extra";
document.write(guk);
</script>
</body>
</html>



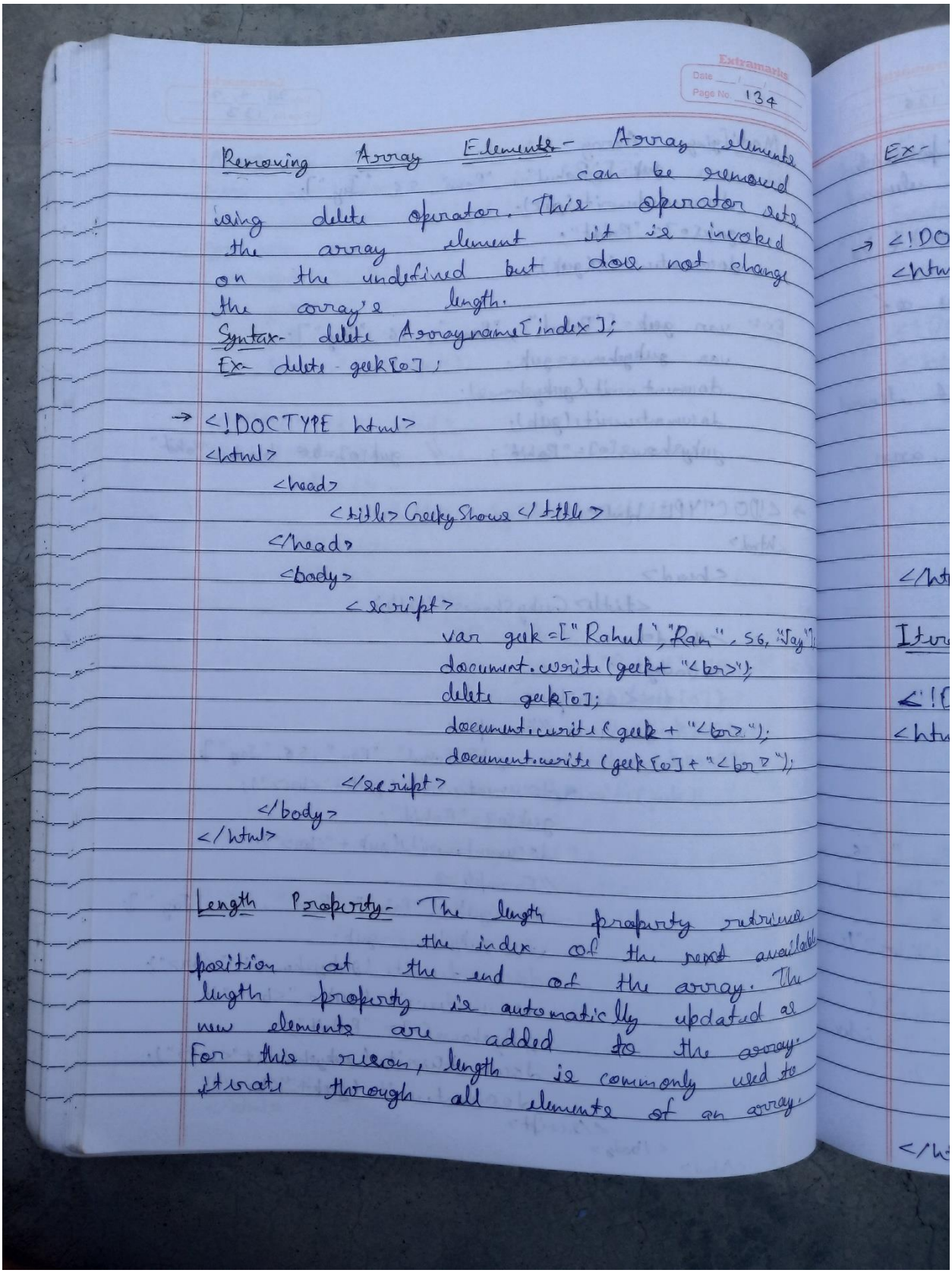
Extramarks
Date 29 4 21
Page No 133

Modifying Array Elements -

Ex-1
```
var geek = ["Rahul", "Ram", 56, "Jay"];
document.write(geek);
geek[0] = "Rohit";
document.write(geek);
```

Ex-2
```
var geek = ["Rahul", "Ram", 56, "Jay"];
var geekyshows = geek;
document.write(geekyshows);
document.write(geek);
geekyshows[0] = "Rohit";   // geek[0] also become "Rohit"
```

```
<!DOCTYPE html>
<html>
    <head>
        <title> Geeky Shows </title>
    </head>
    <body>
        <script>
            //Example-1
            var geek = ["Rahul", "Ram", 56, "Jay"];
            document.write(geek + "<br>");
            geek[0] = "Rohit";
            document.write(geek + "<br>");
            //Example-2
            var geek = ["Rahul", "Ram", 56, "Jay"];
            var geekyshows = geek;
            document.write(geekyshows + "<br>");
            document.write(geek + "<br>");
            geekyshows[0] = "Rohit";
            document.write(geekyshows + "<br>");
            document.write(geek + "<br>");
        </script>
    </body>
</html>
```



Extramarks
Date
Page No. 134

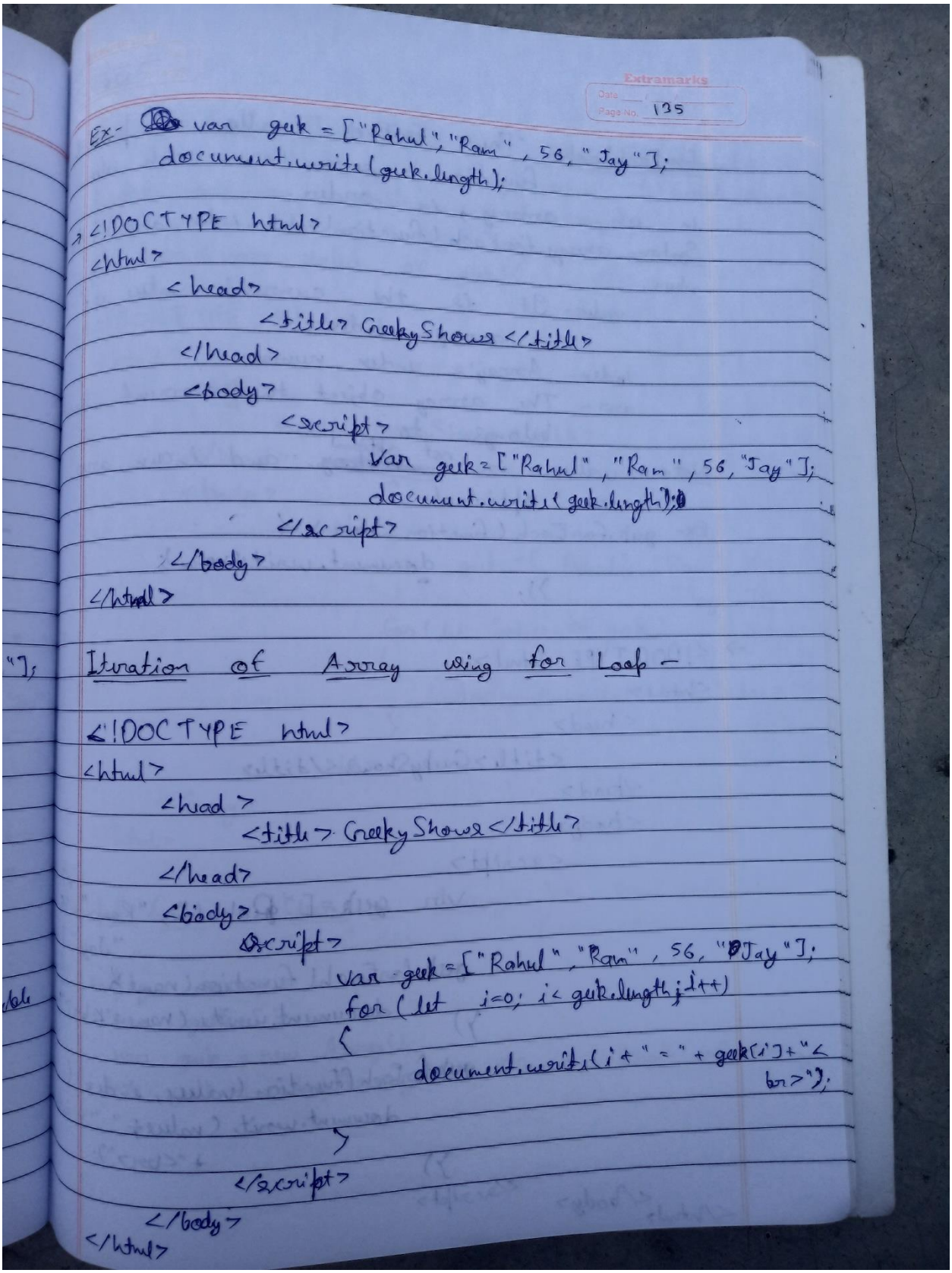
Removing Array Elements - Array elements can be removed using delete operator. This operator sets the array element it is invoked on the undefined but does not change the array's length.

Syntax - delete Arrayname[index];

Ex - delete geek[0];

```
<!DOCTYPE html>
<html>
    <head>
        <title> Geeky Shows </title>
    </head>
    <body>
        <script>
            var geek = ["Rahul", "Ram", 56, "Jay"];
            document.write(geek + "<br>");
            delete geek[0];
            document.write(geek + "<br>");
            document.write(geek[0] + "<br>");
        </script>
    </body>
</html>
```

Length Property - The length property retrieves the index of the next available position at the end of the array. The length property is automatically updated as new elements are added to the array. For this reason, length is commonly used to iterate through all elements of an array.

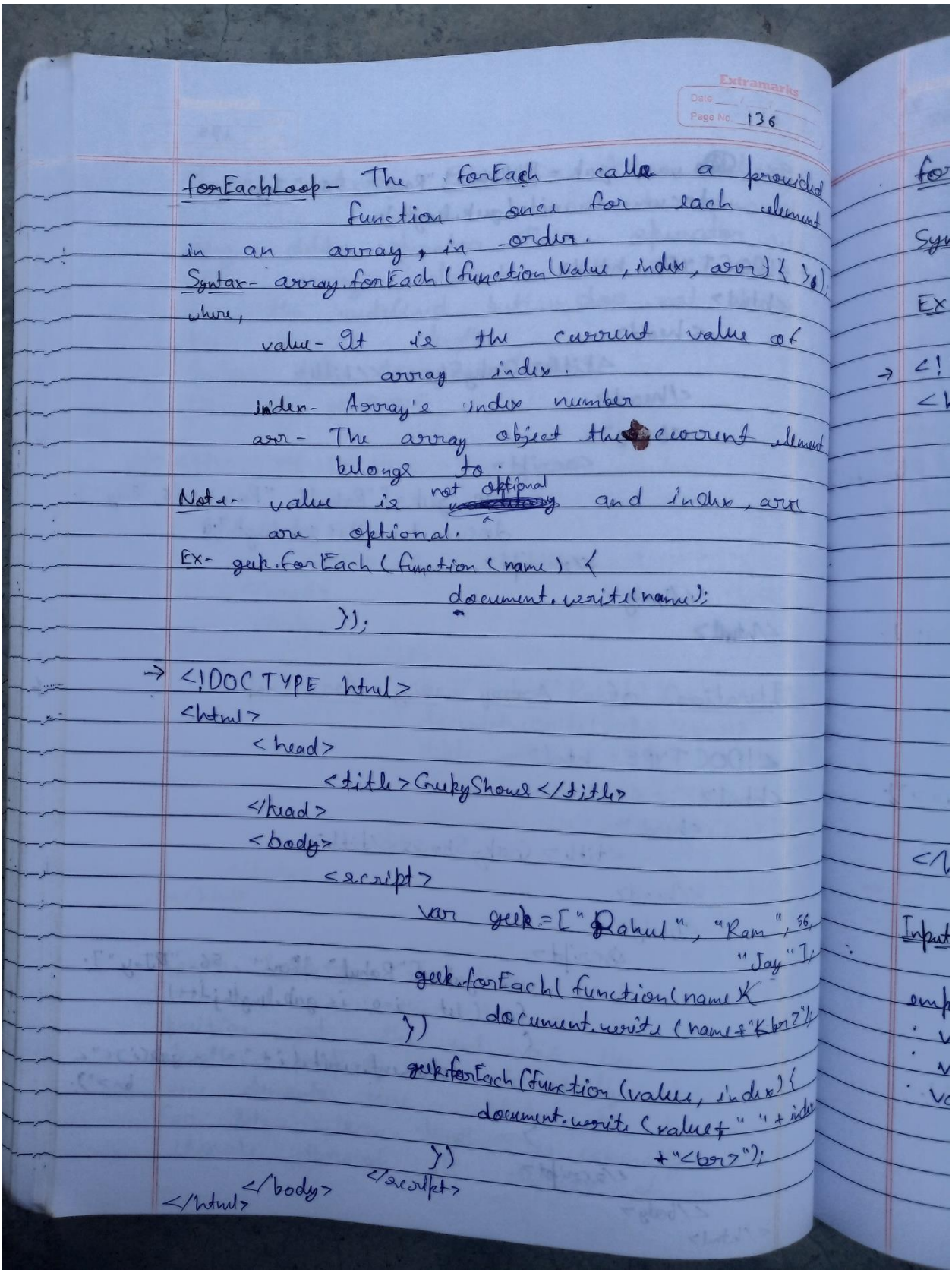


Ex- var geek = ["Rahul", "Ram", 56, "Jay"];
document.write(geek.length);

```
<!DOCTYPE html>
<html>
    <head>
        <title> GeekyShows </title>
    </head>
    <body>
        <script>
            var geek = ["Rahul", "Ram", 56, "Jay"];
            document.write(geek.length);
        </script>
    </body>
</html>
```

Iteration of Array using for Loop -

```
<!DOCTYPE html>
<html>
    <head>
        <title> GeekyShows </title>
    </head>
    <body>
        <script>
            var geek = ["Rahul", "Ram", 56, "Jay"];
            for (let i=0; i< geek.length; i++)
            {
                document.write(i + " = " + geek[i] + "<br>");
            }
        </script>
    </body>
</html>
```



forEachLoop- The forEach calls a provided function once for each element in an array, in order.

Syntax- array.forEach(function(value, index, arr){ });

where,

value- It is the current value of array index.

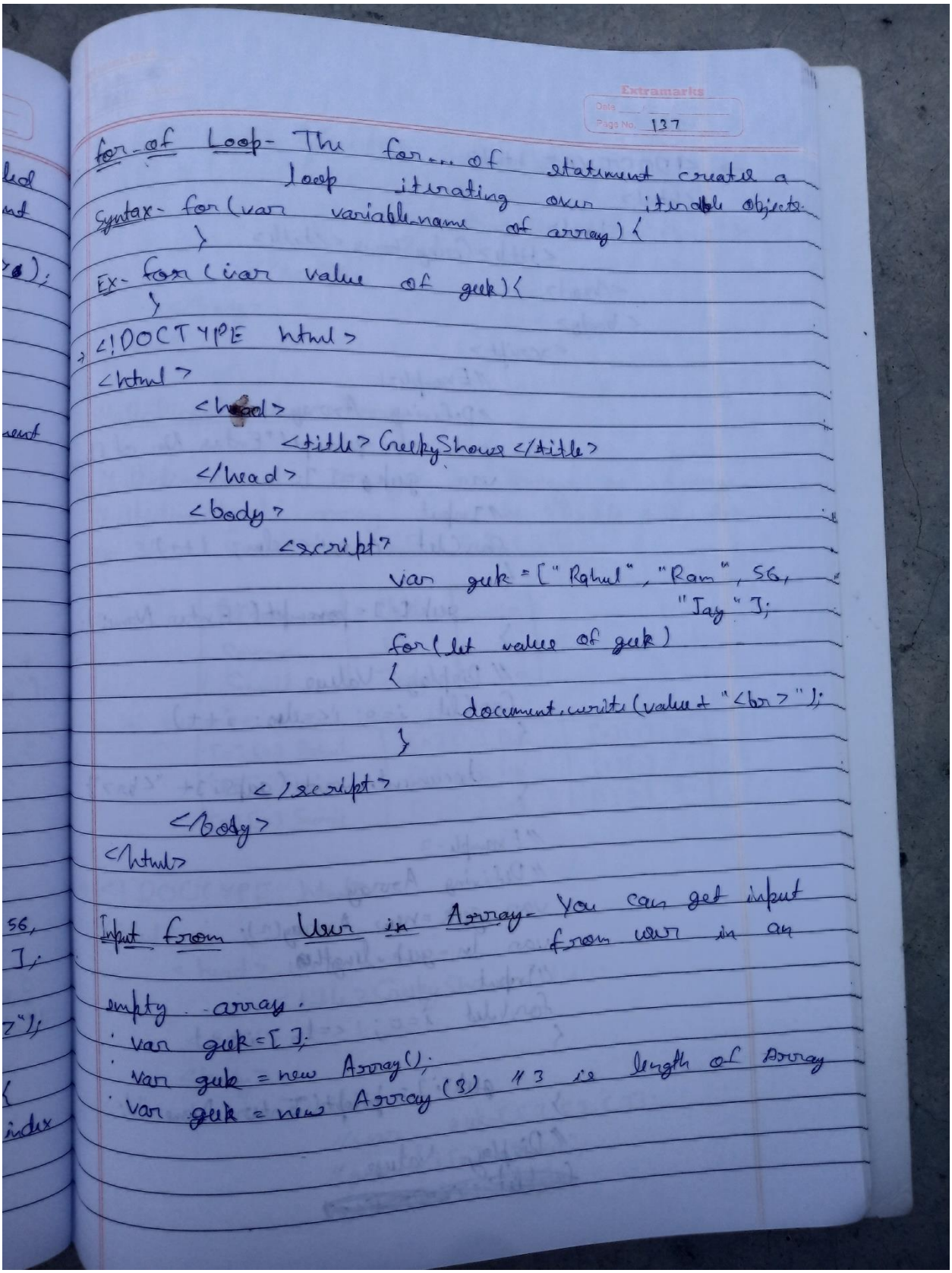
index- Array's index number

arr- The array object the current element belongs to.

Note- value is not optional and index, arr are optional.

Ex- geek.forEach(function(name){
document.write(name);
});

→
```
<!DOCTYPE html>
<html>
    <head>
        <title>GeekyShows</title>
    </head>
    <body>
        <script>
            var geek=["Rahul", "Ram", 56, "Jay"];
            geek.forEach(function(name){
                document.write(name+"<br>");
            })
            geek.forEach(function(value, index){
                document.write(value+" "+index
                +"<br>");
            })
        </script>
    </body>
</html>
```



for...of Loop- The for...of statement creates a loop iterating over iterable objects.

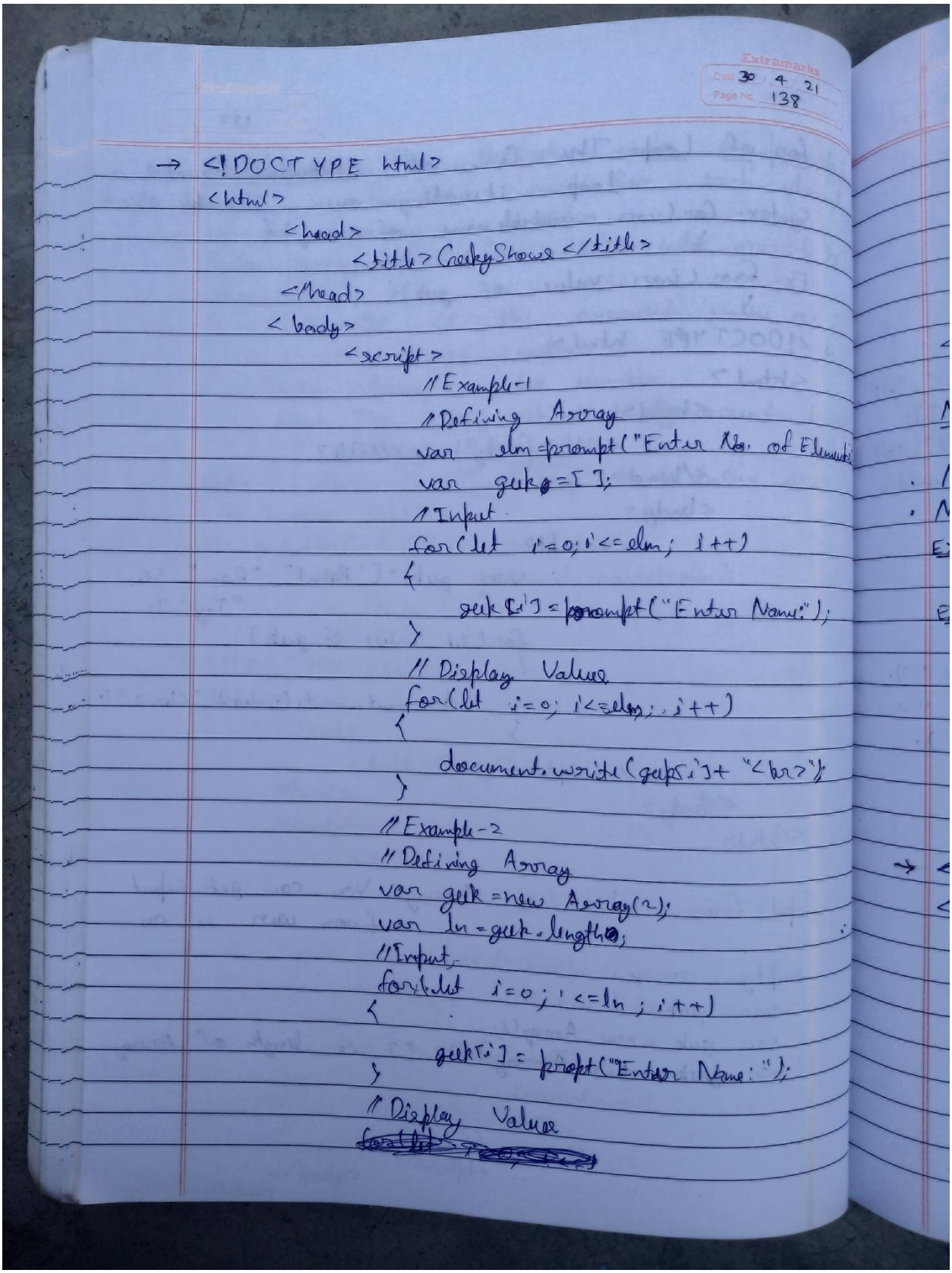
Syntax- for(var variable_name of array){
}

Ex- for(var value of geek){
}

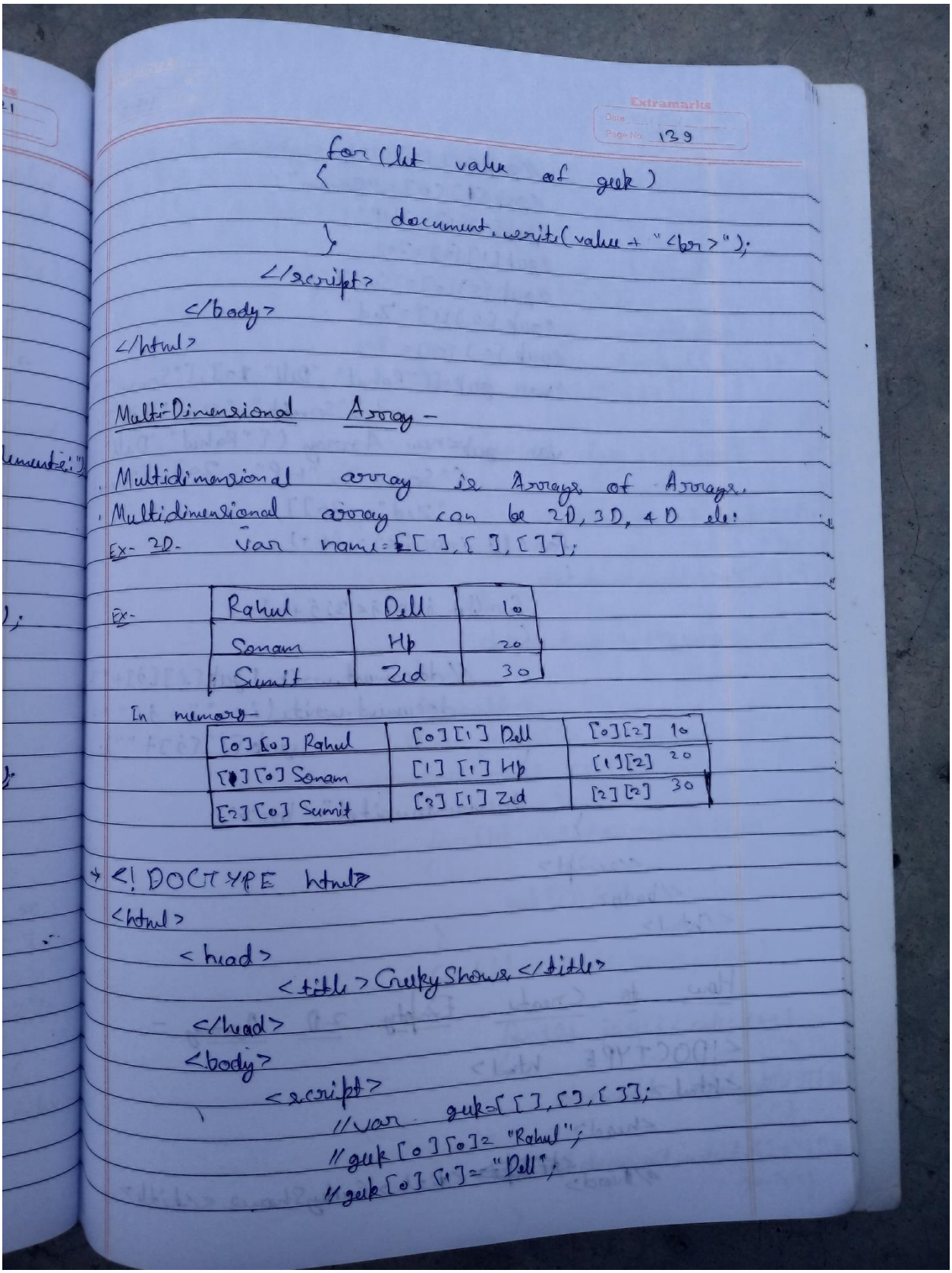
```
<!DOCTYPE html>
<html>
    <head>
        <title> GeekyShows </title>
    </head>
    <body>
        <script>
            var geek = ["Rahul", "Ram", 56,
                        "Jay"];
            for(let value of geek)
            {
                document.write(value + "<br>");
            }
        </script>
    </body>
</html>
```

Input from User in Array- You can get input from user in an empty array.

```
var geek = [];
var geek = new Array();
var geek = new Array(3) // 3 is length of Array
```



```
<!DOCTYPE html>
<html>
    <head>
        <title> GeekyShows </title>
    </head>
    <body>
        <script>
            // Example-1
            // Defining Array
            var elm = prompt("Enter No. of Elements");
            var geek = [ ];
            // Input
            for(let i=0; i<=elm; i++)
            {
                geek[i] = prompt("Enter Name:");
            }
            // Display Values
            for(let i=0; i<=elm; i++)
            {
                document.write(geek[i] + "<br>");
            }
            // Example-2
            // Defining Array
            var geek = new Array(2);
            var ln = geek.length;
            // Input
            for(let i=0; i<=ln; i++)
            {
                geek[i] = prompt("Enter Name:");
            }
            // Display Values
```



```
for(let value of geek)
{
    document.write(value + "<br>");
}
</script>
</body>
</html>
```

Multi-Dimensional Array -

- Multidimensional array is Arrays of Arrays.
- Multidimensional array can be 2D, 3D, 4D etc.

Ex- 2D- var name=[ ][ ][ ];

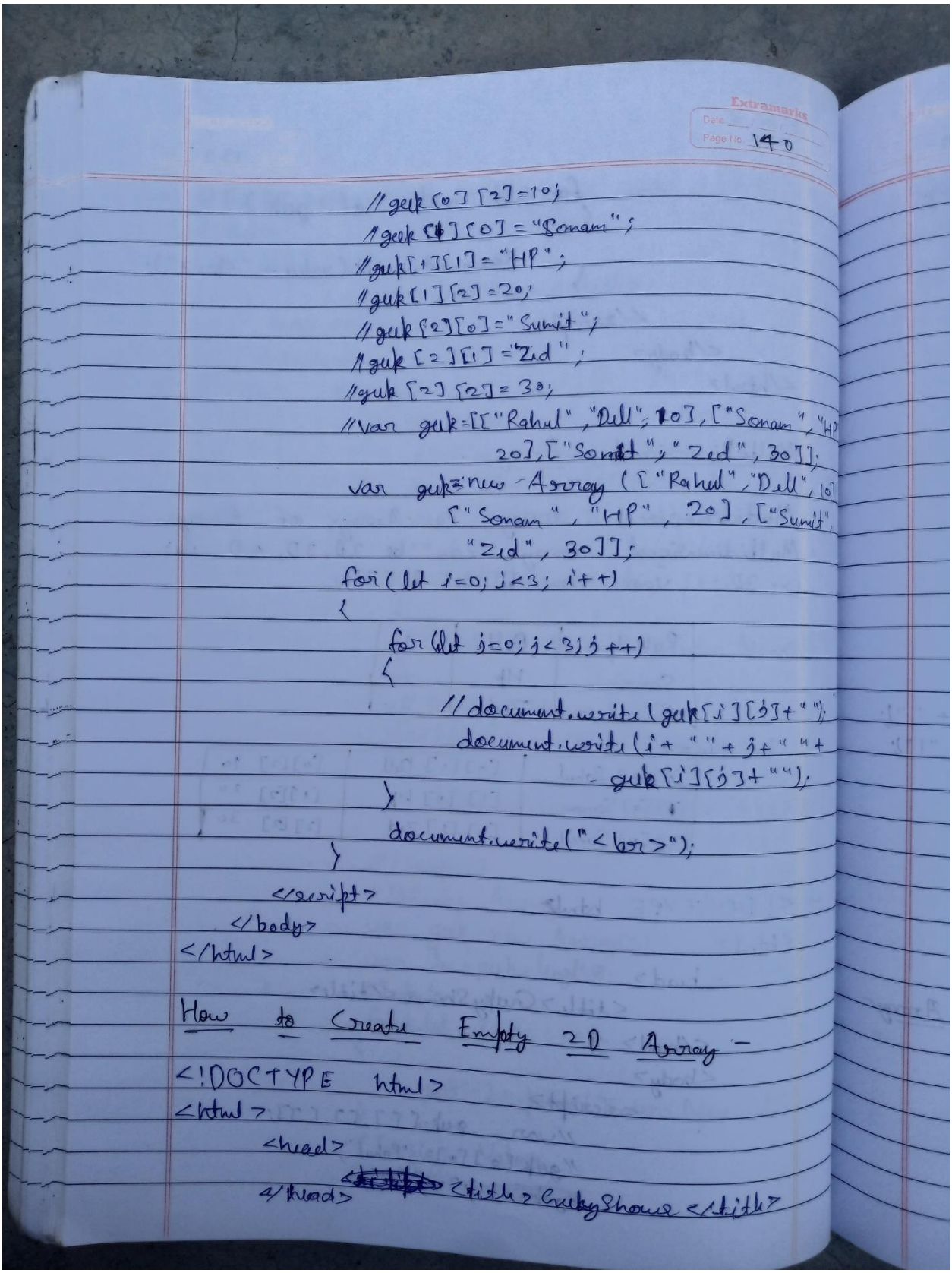
Ex-

Rahul	Dell	10
Sonam	Hp	20
Sumit	Zid	30

In memory-

[0][0] Rahul	[0][1] Dell	[0][2] 10
[1][0] Sonam	[1][1] Hp	[1][2] 20
[2][0] Sumit	[2][1] Zid	[2][2] 30

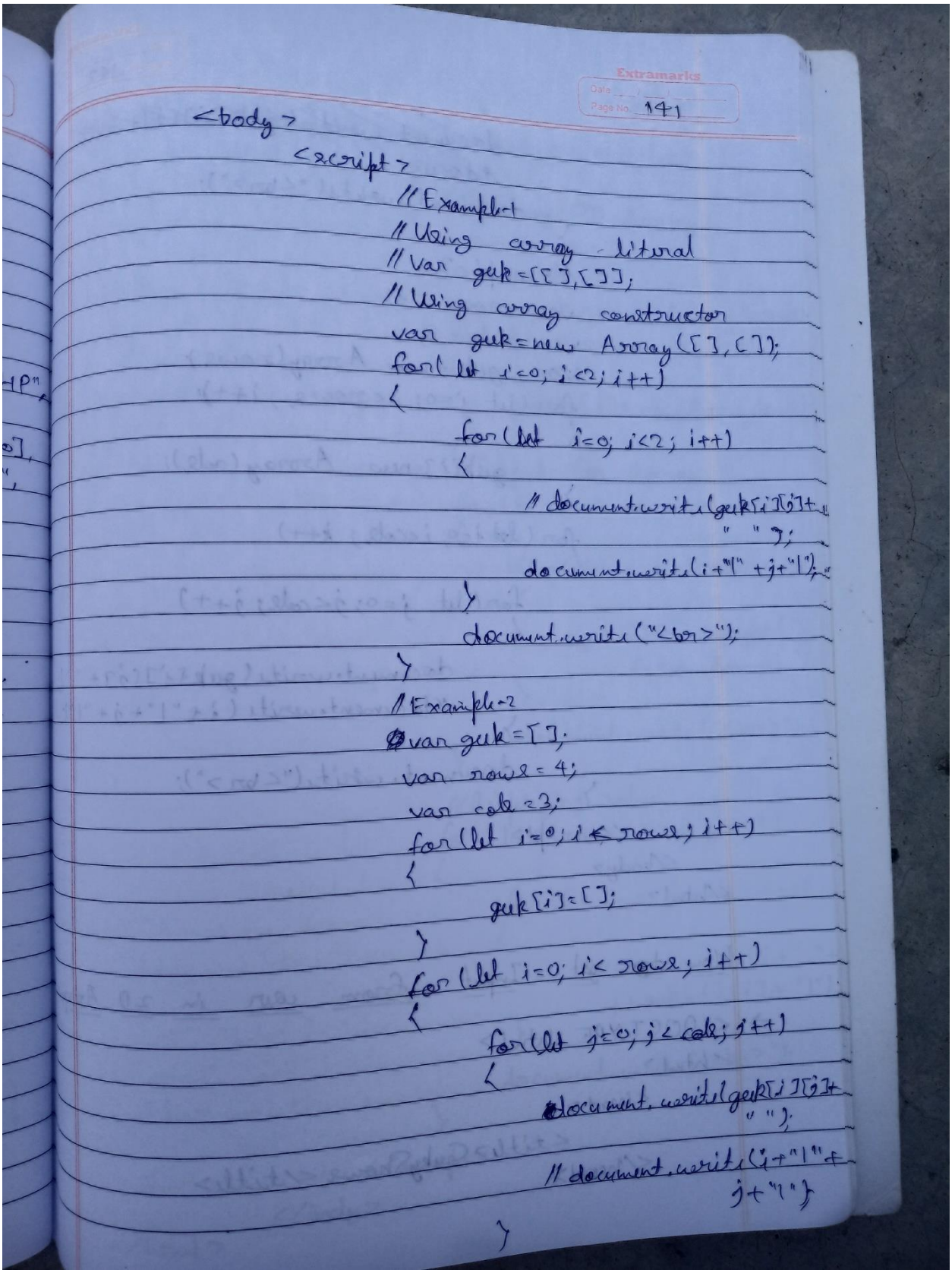
```
<!DOCTYPE html>
<html>
    <head>
        <title> Geeky Shows </title>
    </head>
    <body>
        <script>
            //var geek=[ ][ ][ ];
            //geek[0][0]= "Rahul";
            //geek[0][1]= "Dell";
```



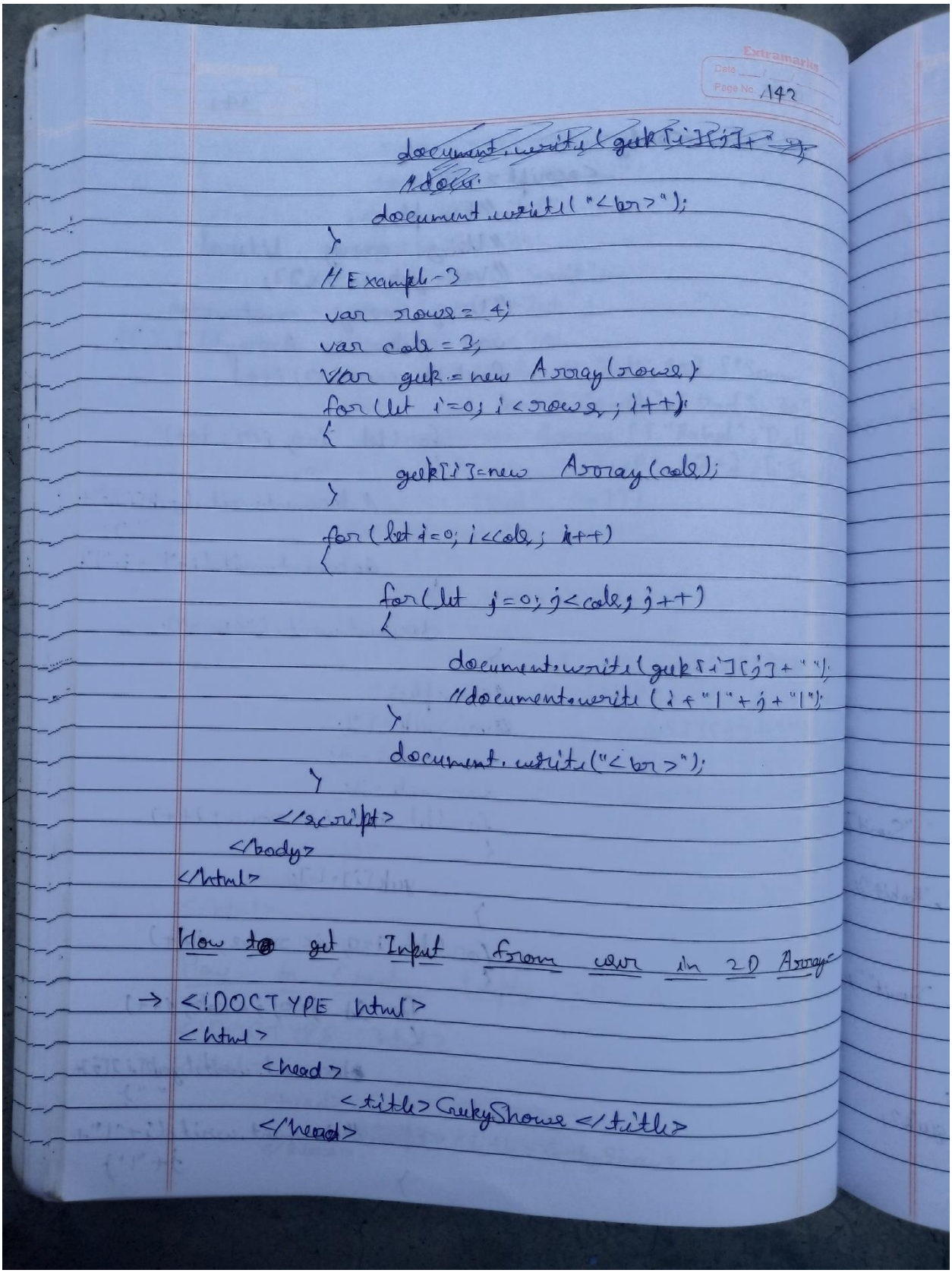
```
//geek[0][2]=10;
//geek[1][0]="Sonam";
//geek[1][1]="HP";
//geek[1][2]=20;
//geek[2][0]="Sumit";
//geek[2][1]="Zed";
//geek[2][2]=30;
//var geek=[["Rahul","Dell",10],["Sonam","HP",20],["Sumit","Zed",30]];
var geek=new Array(["Rahul","Dell",10],["Sonam","HP",20],["Sumit","Zed",30]];
for(let i=0;i<3;i++)
{
    for(let j=0;j<3;j++)
    {
        //document.write(geek[i][j]+" ");
        document.write(i+" "+j+" "+geek[i][j]+"");
    }
    document.write("<br>");
}
</script>
</body>
</html>
```

How to Create Empty 2D Array -

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



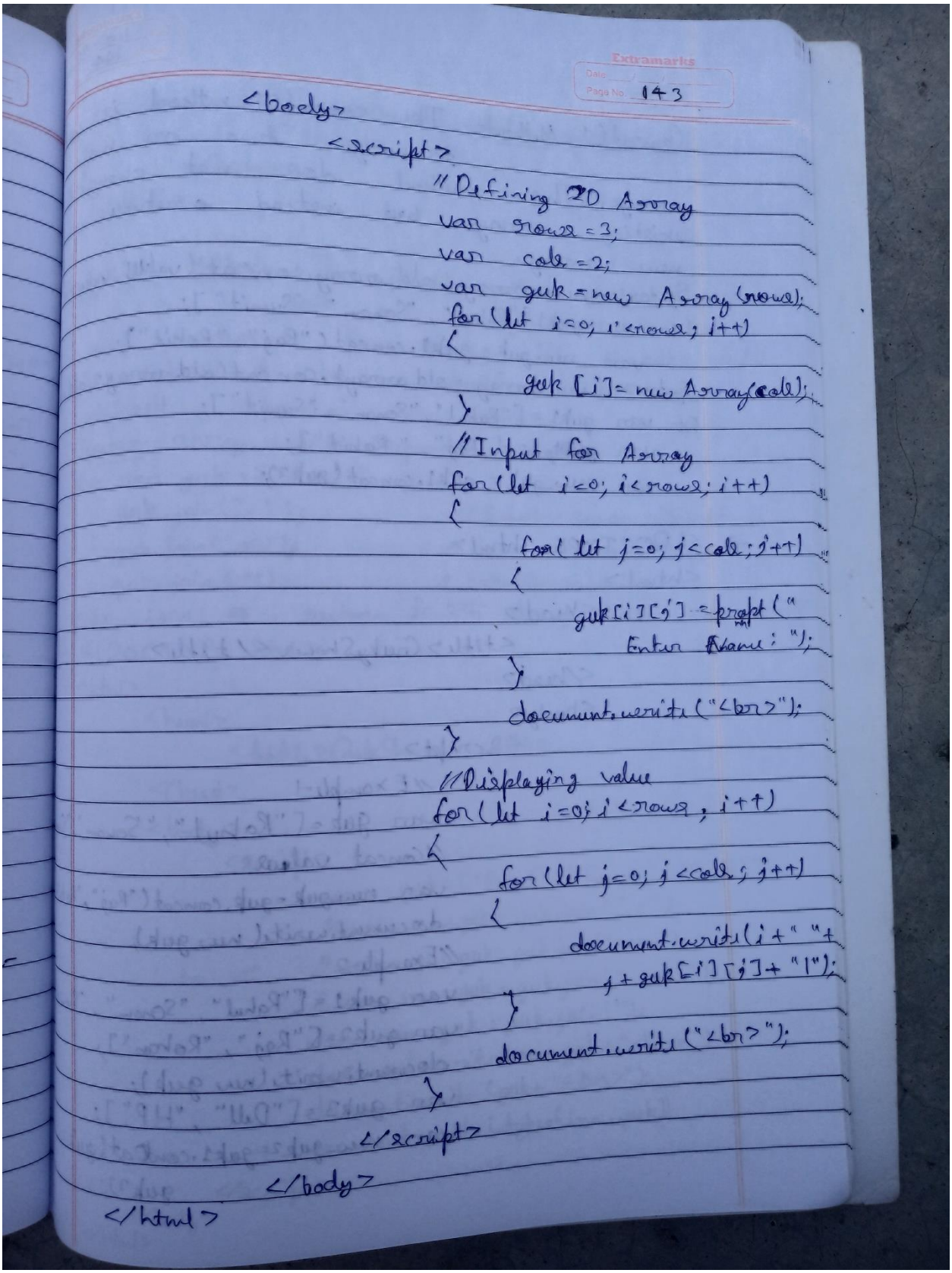
```
<body>
    <script>
        // Example-1
        // Using array literal
        // var geek=[[],[]];
        // Using array constructor
        var geek=new Array([],[]);
        for(let i=0;i<2;i++)
        {
            for(let i=0;i<2;i++)
            {
                // document.write(geek[i][j]+
                    " ");
                document.write(i+"|"+j+"|");
            }
            document.write("<br>");
        }
        // Example-2
        var geek=[];
        var rows=4;
        var cols=3;
        for(let i=0;i<rows;i++)
        {
            geek[i]=[];
        }
        for(let i=0;i<rows;i++)
        {
            for(let j=0;j<cols;j++)
            {
                document.write(geek[i][j]+
                    " ");
                // document.write(i+"|"+
                    j+"|");
            }
```



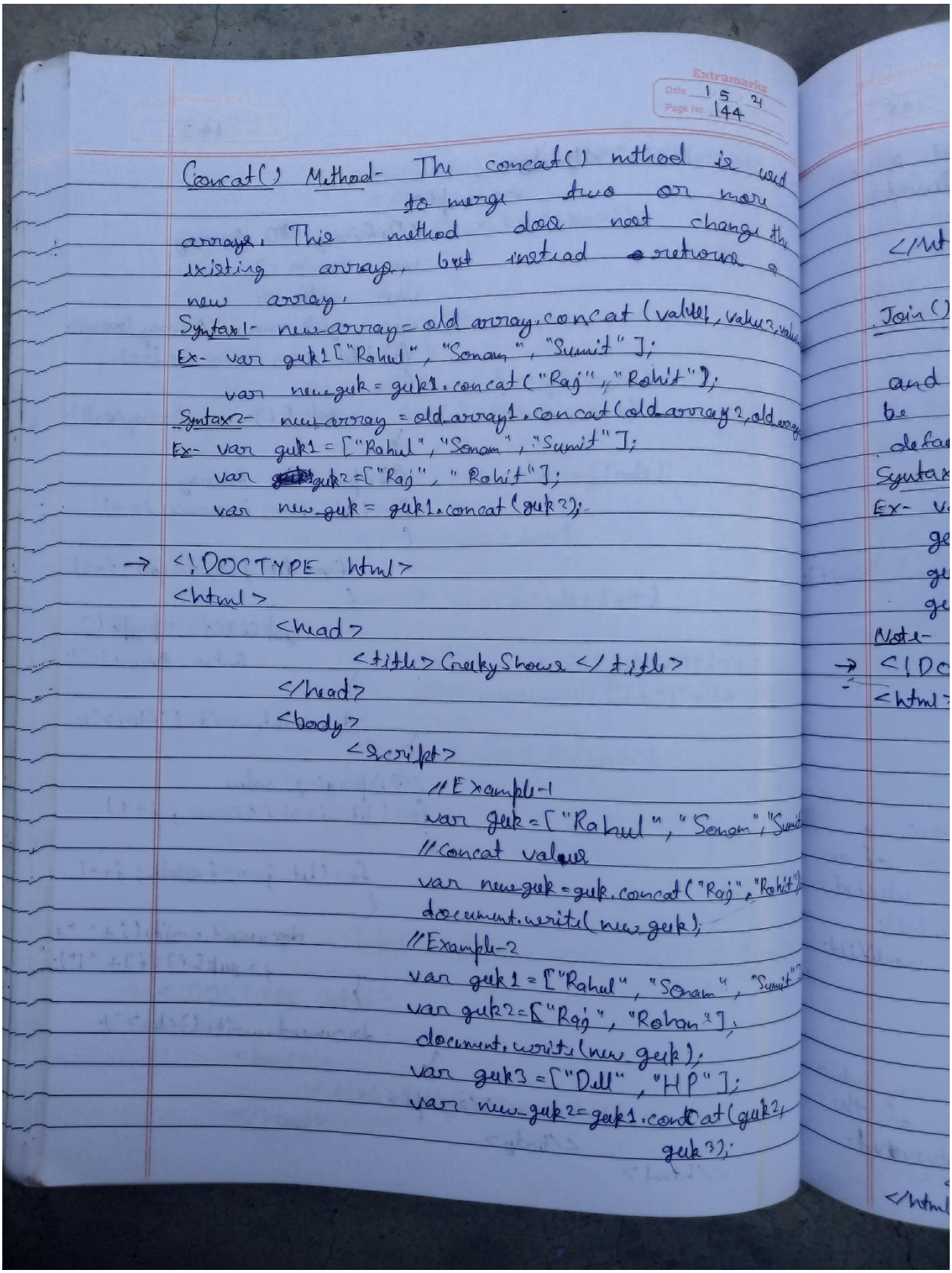
```
document.write(guk[i][j]+" ");
document.write("<br>");
}
//Example-3
var rows = 4;
var cols = 3;
var guk = new Array(rows);
for(let i=0; i<rows; i++)
{
    guk[i]=new Array(cols);
}
for(let i=0; i<cols; i++)
{
    for(let j=0; j<cols; j++)
    {
        document.write(guk[i][j]+" ");
        //document.write(i+"|"+j+"|");
    }
    document.write("<br>");
}
</script>
</body>
</html>
```

How to get Input from user in 2D Array?

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



```
<body>
<script>
// Defining 2D Array
var rows = 3;
var cols = 2;
var guk = new Array(rows);
for(let i=0; i<rows; i++)
{
    guk[i] = new Array(cols);
}
// Input for Array
for(let i=0; i<rows; i++)
{
    for(let j=0; j<cols; j++)
    {
        guk[i][j] = prompt("
            Enter Value: ");
    }
    document.write("<br>");
}
// Displaying value
for(let i=0; i<rows; i++)
{
    for(let j=0; j<cols; j++)
    {
        document.write(i+" "+
        j+guk[i][j]+"|");
    }
    document.write("<br>");
}
</script>
</body>
</html>
```



Concat() Method- The concat() method is used to merge two or more arrays. This method does not change the existing arrays, but instead returns a new array.

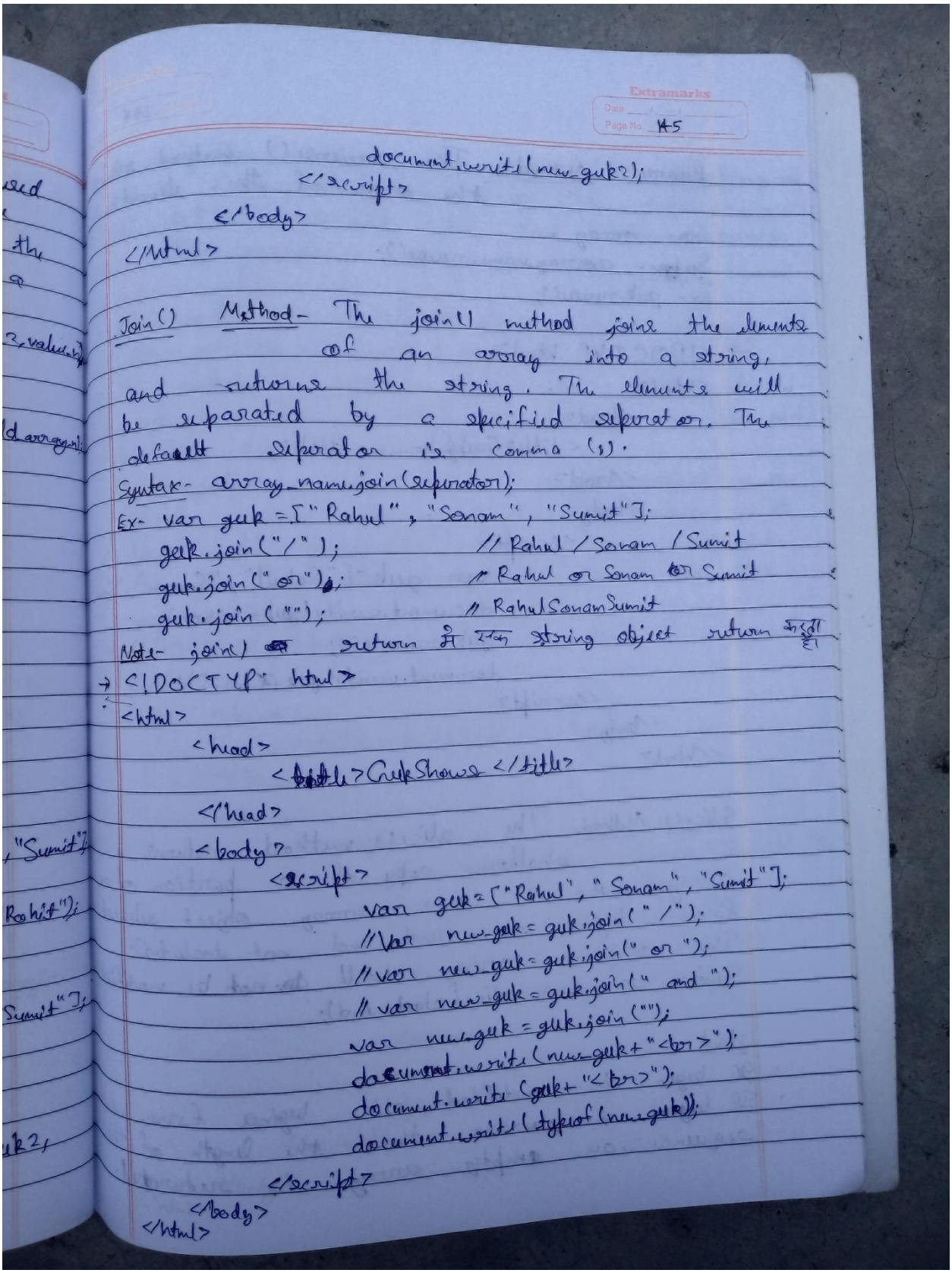
Syntax1- new_array = old_array.concat(value1, value2, value...

Ex- var geek1 = ["Rahul", "Sonam", "Sumit"];
var new_geek = geek1.concat("Raj", "Rohit");

Syntax2- new_array = old_array1.concat(old_array2, old_array...

Ex- var geek1 = ["Rahul", "Sonam", "Sumit"];
var geek2 = ["Raj", "Rohit"];
var new_geek = geek1.concat(geek2);

```
<!DOCTYPE html>
<html>
    <head>
        <title> GeekyShows </title>
    </head>
    <body>
        <script>
            //Example-1
            var geek = ["Rahul", "Sonam", "Sumit"
            //Concat value
            var new_geek = geek.concat("Raj", "Rohit");
            document.write(new_geek);
            //Example-2
            var geek1 = ["Rahul", "Sonam", "Sumit"];
            var geek2 = ["Raj", "Rohan"];
            document.write(new_geek);
            var geek3 = ["Dell", "HP"];
            var new_geek2 = geek1.concat(geek2,
                geek3);
```



```
        document.write(new_guk2);
    </script>
</body>
</html>
```

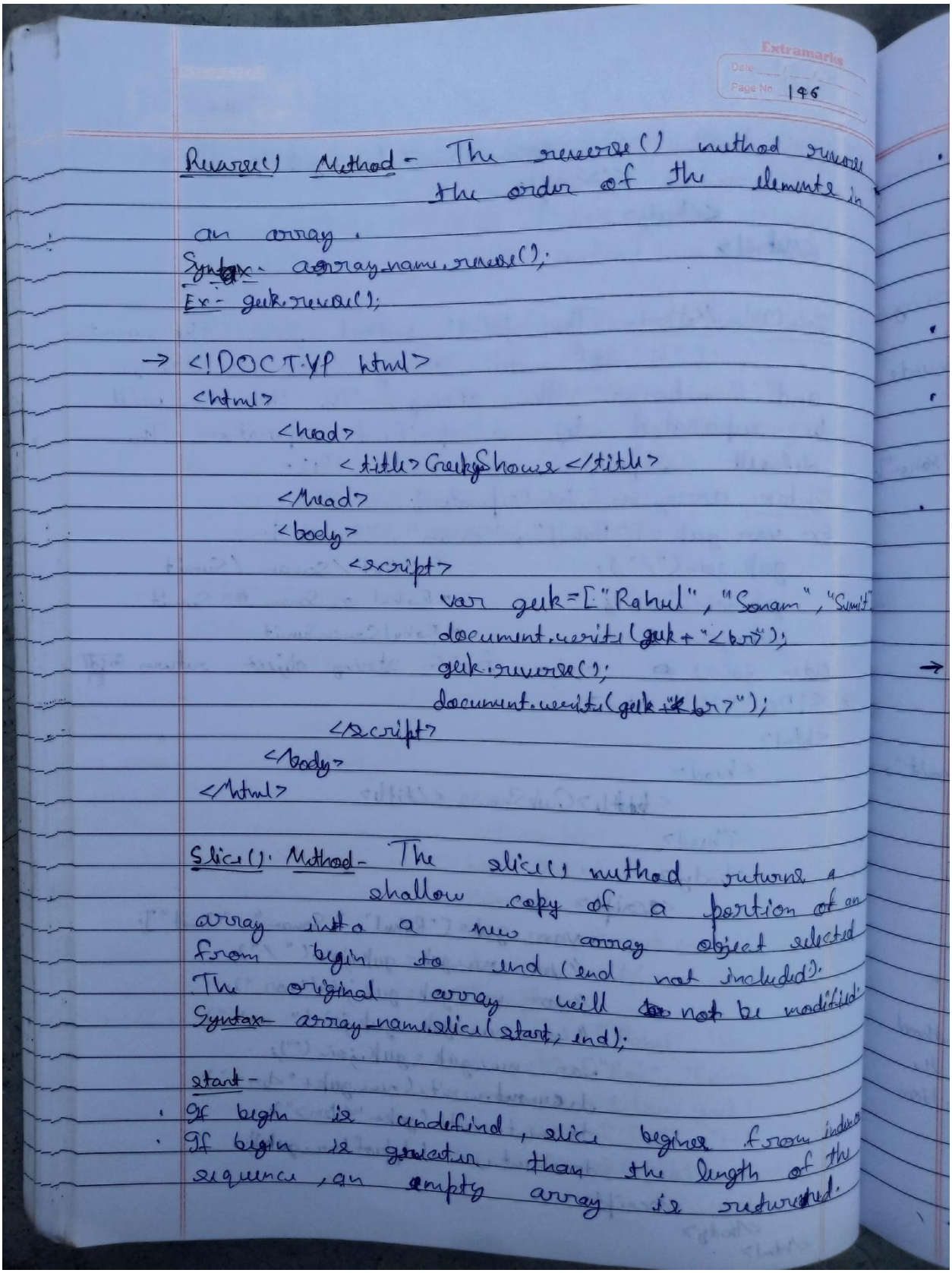
Join() Method - The join() method joins the elements of an array into a string, and returns the string. The elements will be separated by a specified separator. The default separator is comma (,).

Syntax - array_name.join(separator);

Ex - var guk = ["Rahul", "Sonam", "Sumit"];
guk.join("/"); // Rahul/Sonam/Sumit
guk.join(" or "); // Rahul or Sonam or Sumit
guk.join(""); // RahulSonamSumit

Note - join() return a string object return

```
<!DOCTYP html>
<html>
    <head>
        <title>GukShows</title>
    </head>
    <body>
        <script>
            var guk = ["Rahul", "Sonam", "Sumit"];
            //var new_guk = guk.join("/");
            //var new_guk = guk.join(" or ");
            //var new_guk = guk.join(" and ");
            var new_guk = guk.join("");
            document.write(new_guk+"<br>");
            document.write(guk+"<br>");
            document.write(typeof(new_guk));
        </script>
    </body>
</html>
```



Reverse() Method - The reverse() method reverses the order of the elements in an array.

Syntax - array_name.reverse();

Ex - geek.reverse();

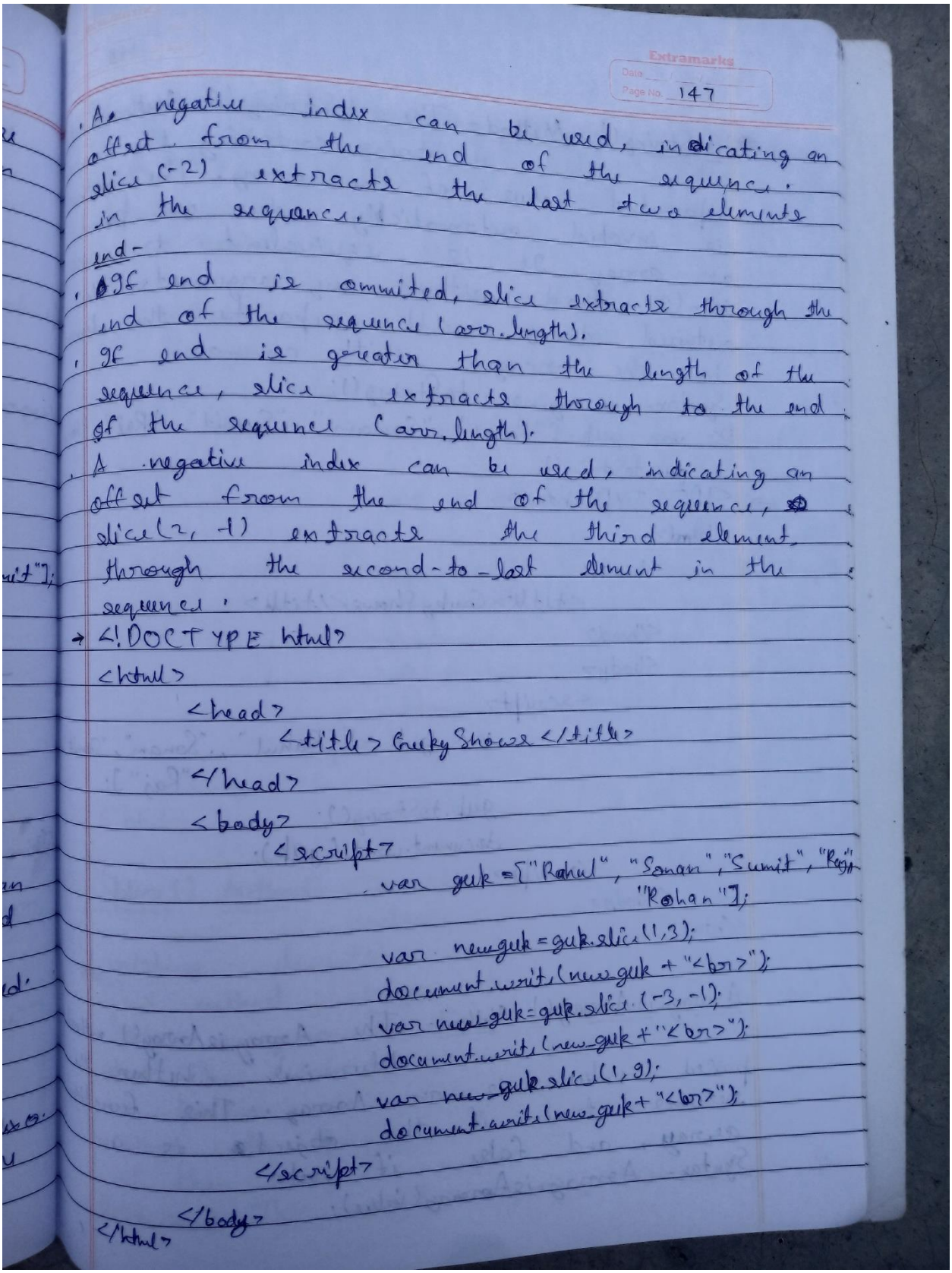
```
<!DOCTYP html>
<html>
    <head>
        <title> GeekyShows </title>
    </head>
    <body>
        <script>
            var geek=["Rahul", "Sonam", "Sumit"];
            document.write(geek + "<br>");
            geek.reverse();
            document.write(geek + "<br>");
        </script>
    </body>
</html>
```

Slice() Method - The slice() method returns a shallow copy of a portion of an array into a new array object selected from begin to end (end not included). The original array will not be modified.

Syntax - array_name.slice(start, end);

start -

- If begin is undefind, slice begins from index
- If begin is greater than the length of the sequence, an empty array is returned.



A negative index can be used, indicating an offset from the end of the sequence. slice(-2) extracts the last two elements in the sequence.

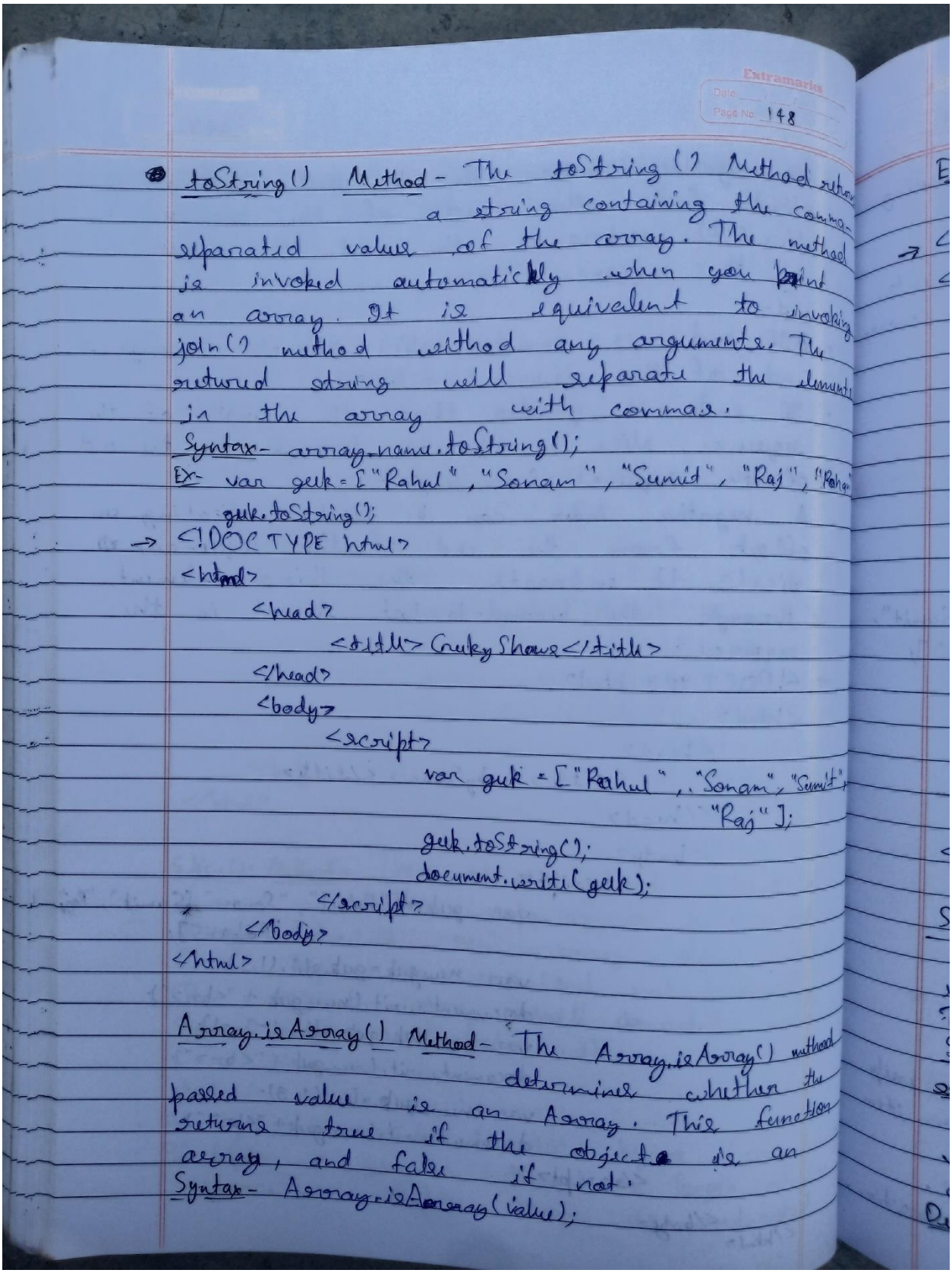
end-

If end is omitted, slice extracts through the end of the sequence (arr.length).

If end is greater than the length of the sequence, slice extracts through to the end of the sequence (arr.length).

A negative index can be used, indicating an offset from the end of the sequence, slice(2, -1) extracts the third element through the second-to-last element in the sequence.

```
<!DOCTYPE html>
<html>
    <head>
        <title> Geeky Shows </title>
    </head>
    <body>
        <script>
            var geek = ["Rahul", "Sonam", "Sumit", "Raj", "Rohan"];
            var newgeek = geek.slice(1,3);
            document.write(newgeek + "<br>");
            var newgeek = geek.slice(-3,-1);
            document.write(newgeek + "<br>");
            var newgeek.slice(1,9);
            document.write(newgeek + "<br>");
        </script>
    </body>
</html>
```



Extramarks
Date
Page No. 148

● toString() Method - The toString() Method returns a string containing the comma-separated value of the array. The method is invoked automatically when you print an array. It is equivalent to invoking join() method without any arguments. The returned string will separate the elements in the array with commas.

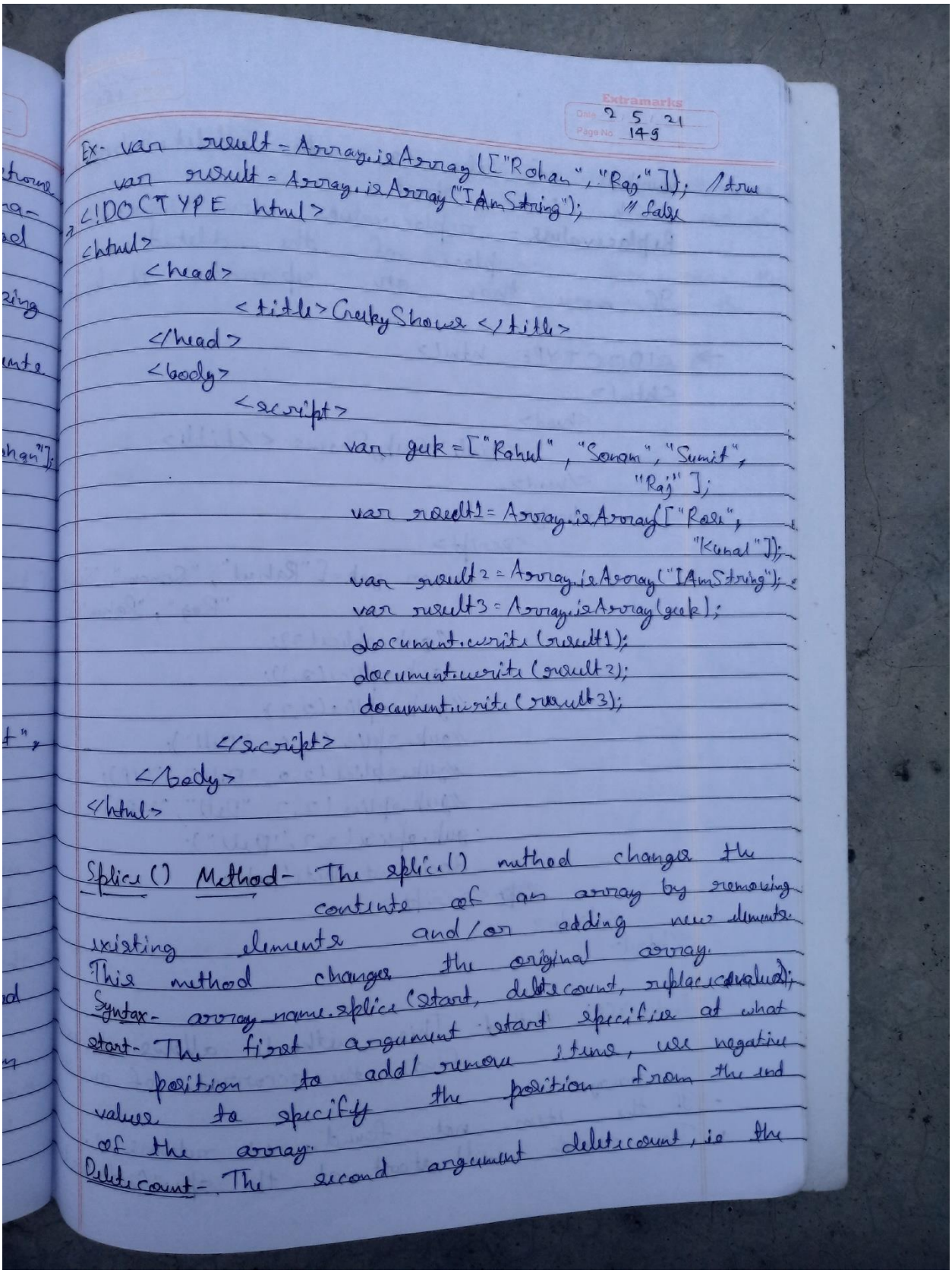
Syntax - array.name.toString();

Ex- var geek = ["Rahul", "Sonam", "Sumit", "Raj", "Rahul"]
geek.toString();

```
<!DOCTYPE html>
<html>
    <head>
        <title> Geeky Shows </title>
    </head>
    <body>
        <script>
            var geek = ["Rahul", "Sonam", "Sumit",
                        "Raj"];
            geek.toString();
            document.write(geek);
        </script>
    </body>
</html>
```

Array.isArray() Method - The Array.isArray() method determines whether the passed value is an Array. This function returns true if the object is an array, and false if not.

Syntax - Array.isArray(value);



Ex- var result = Array.isArray(["Rohan", "Raj"]); // true
var result = Array.isArray("IAmString"); // false

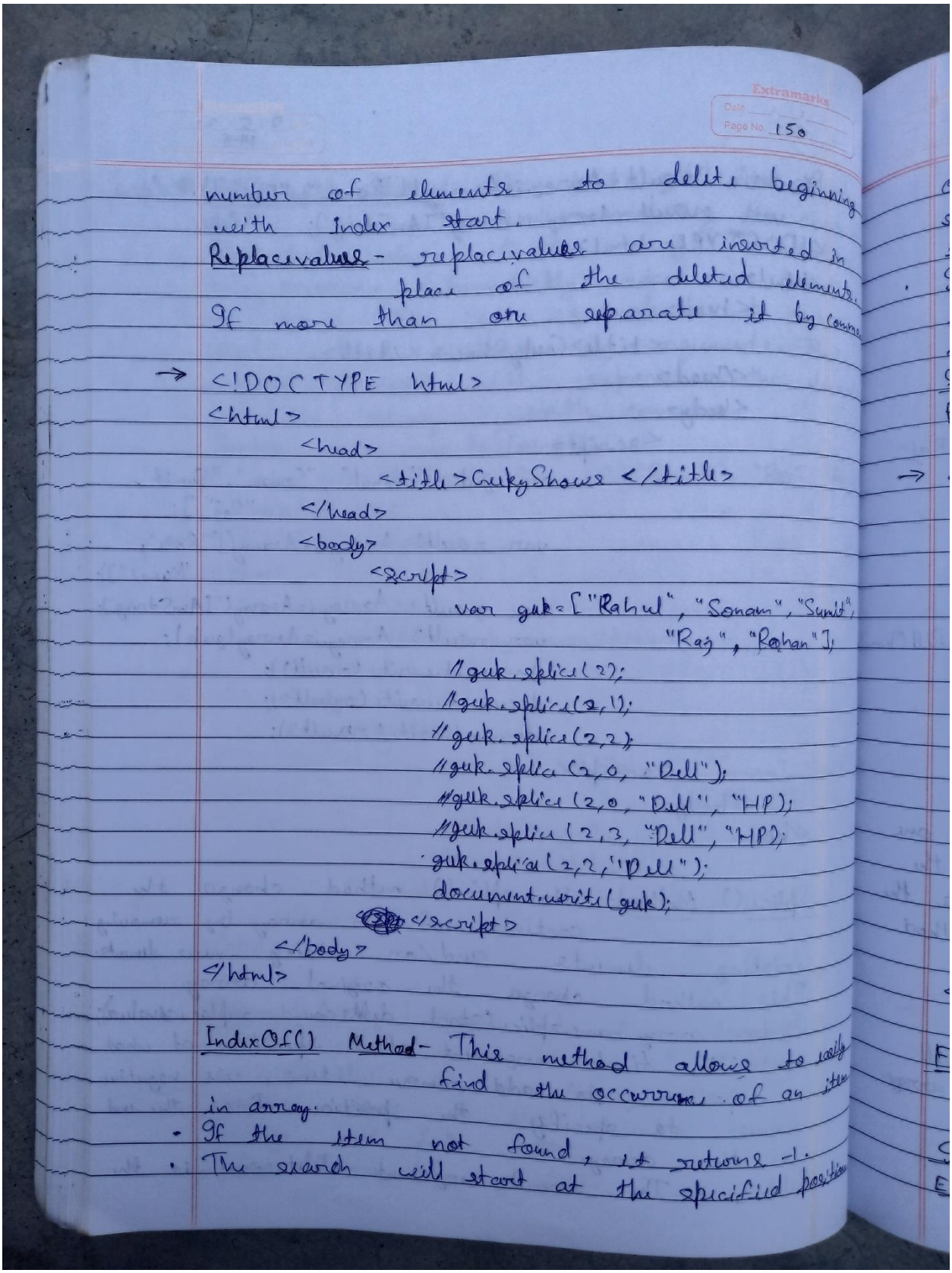
```
<!DOCTYPE html>
<html>
    <head>
        <title>Geeky Shows</title>
    </head>
    <body>
        <script>
            var geek = ["Rahul", "Sonam", "Sumit",
                        "Raj"];
            var result1 = Array.isArray(["Rosi",
                        "Kunal"]);
            var result2 = Array.isArray("IAmString");
            var result3 = Array.isArray(geek);
            document.write(result1);
            document.write(result2);
            document.write(result3);
        </script>
    </body>
</html>
```

Splice() Method- The splice() method changes the contents of an array by removing existing elements and/or adding new elements. This method changes the original array.

Syntax- array_name.splice(start, deletecount, replacevalue);

start- The first argument start specifies at what position to add/remove items, use negative values to specify the position from the end of the array.

Deletecount- The second argument deletecount, is the



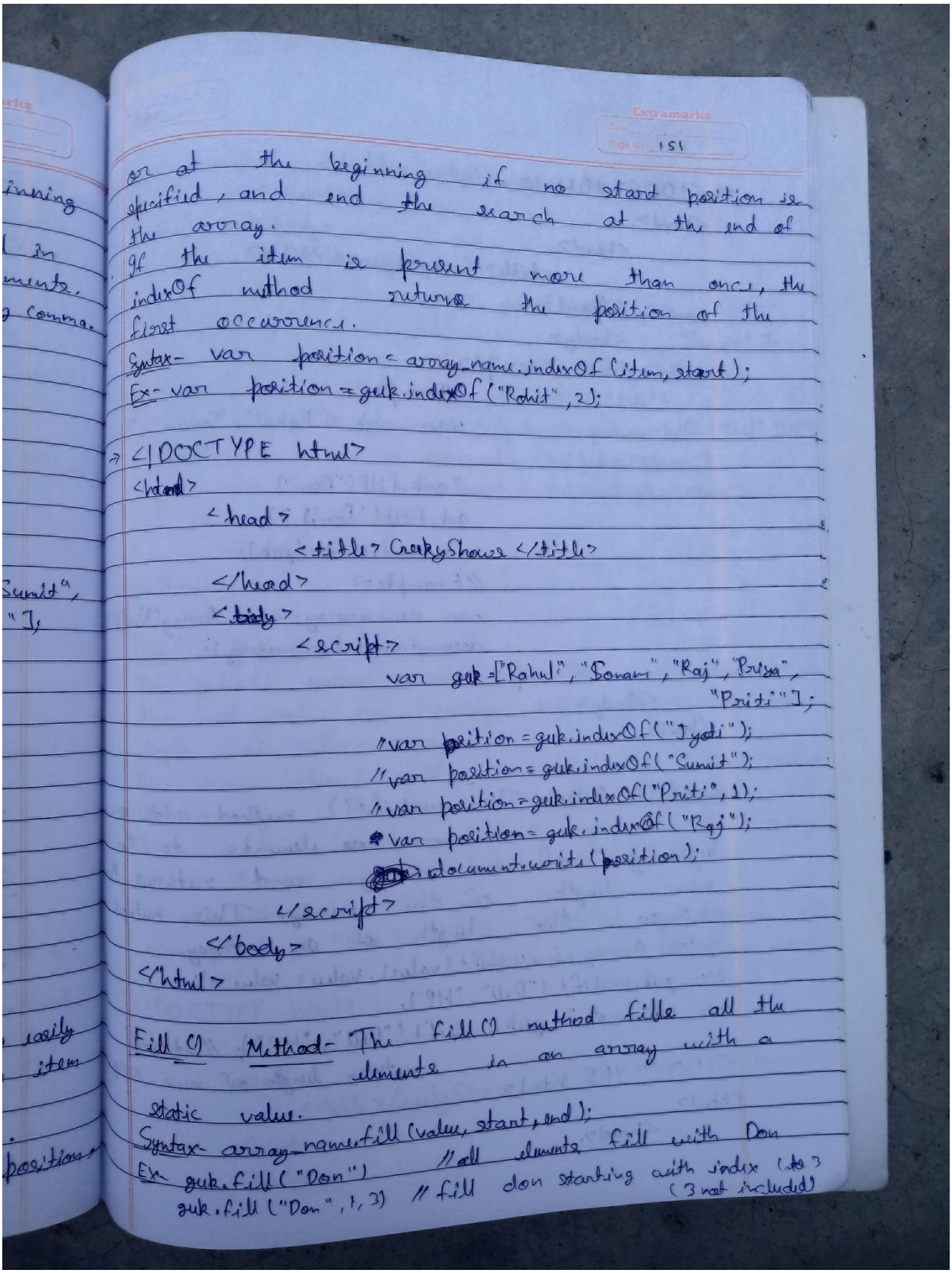
number of elements to delete beginning with index start.

Replacevalues - replacevalues are inserted in place of the deleted elements. If more than one separate it by comma.

→
```
<!DOCTYPE html>
<html>
    <head>
        <title> GeekyShows </title>
    </head>
    <body>
        <script>
            var geek = ["Rahul", "Sonam", "Sumit",
                        "Raj", "Rohan"];
            // geek.splice(2);
            // geek.splice(2,1);
            // geek.splice(2,2)
            // geek.splice(2,0, "Dell");
            // geek.splice(2,0, "Dell", "HP");
            // geek.splice(2,3, "Dell", "HP");
            geek.splice(2,2,'Dell");
            document.write(geek);
        </script>
    </body>
</html>
```

IndexOf() Method - This method allows to easily find the occurrence of an item in array.

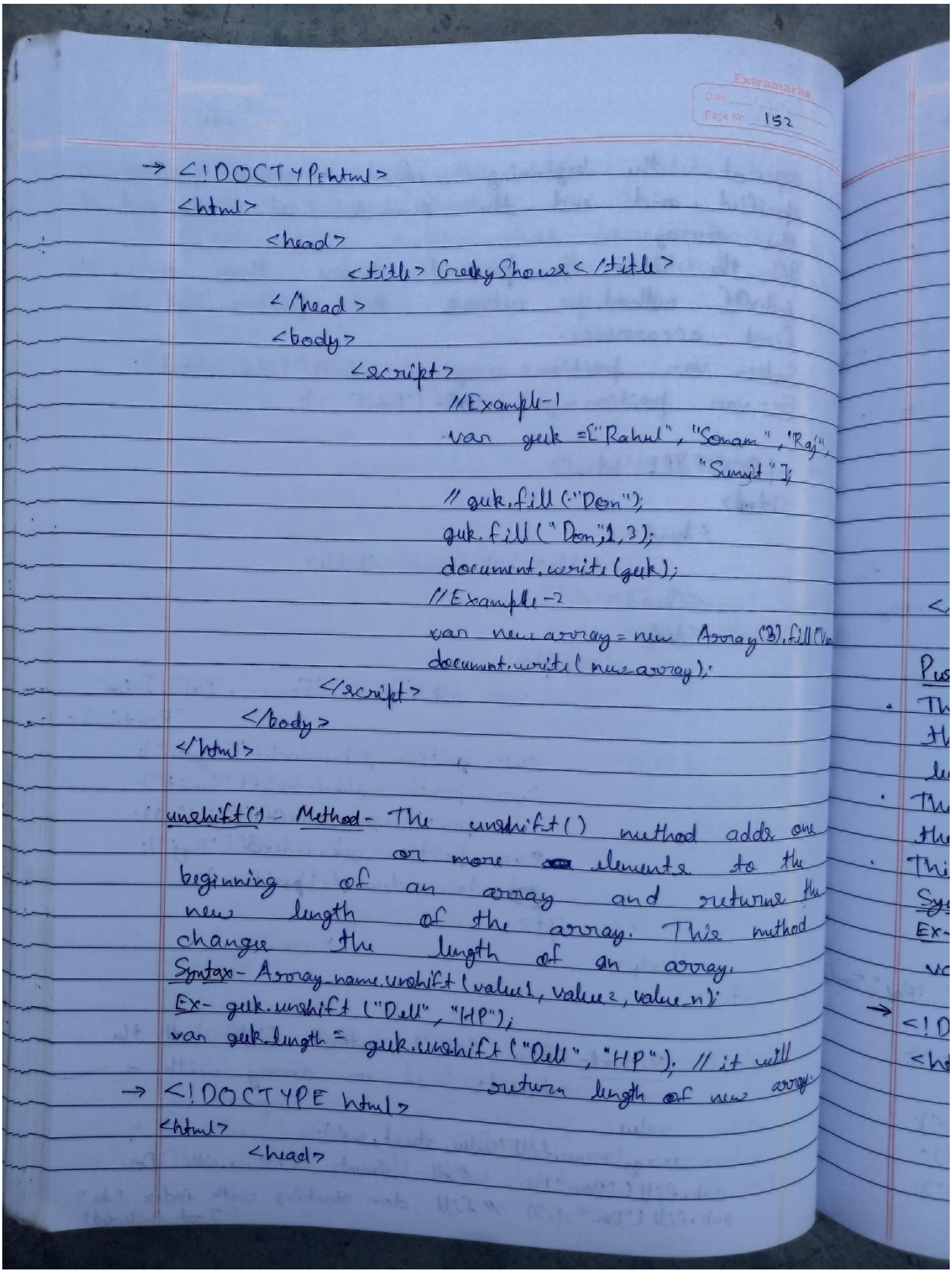
- If the item not found, it returns -1.
- The search will start at the specified position



or at the beginning if no start position is specified, and end the search at the end of the array.
If the item is present more than once, the indexOf method returns the position of the first occurrence.
Syntax- var position = array_name.indexOf(item, start);
Ex- var position = geek.indexOf("Rohit", 2);

```
<!DOCTYPE html>
<html>
    <head>
        <title> CreakyShows </title>
    </head>
    <body>
        <script>
            var geek = ["Rahul", "Sonam", "Raj", "Priya",
                        "Priti"];

            //var position = geek.indexOf("Jyoti");
            //var position = geek.indexOf("Sumit");
            //var position = geek.indexOf("Priti", 1);
            var position = geek.indexOf("Raj");
            document.write(position);
        </script>
    </body>
</html>
```

Fill() Method- The fill() method fills all the elements in an array with a static value.
Syntax- array_name.fill(value, start, end);
Ex- geek.fill("Don") // all elements fill with Don
geek.fill("Don", 1, 3) // fill don starting with index (1 to 3 (3 not included)



```
<!DOCTYPE html>
<html>
    <head>
        <title> GeekyShows </title>
    </head>
    <body>
        <script>
            //Example-1
            var geek = ["Rahul", "Sonam", "Raj",
                        "Sumit"];
            // geek.fill("Don");
            geek.fill("Don",1,3);
            document.write(geek);
            //Example-2
            var new_array = new Array(3).fill("Ve
            document.write(new_array);
        </script>
    </body>
</html>
```

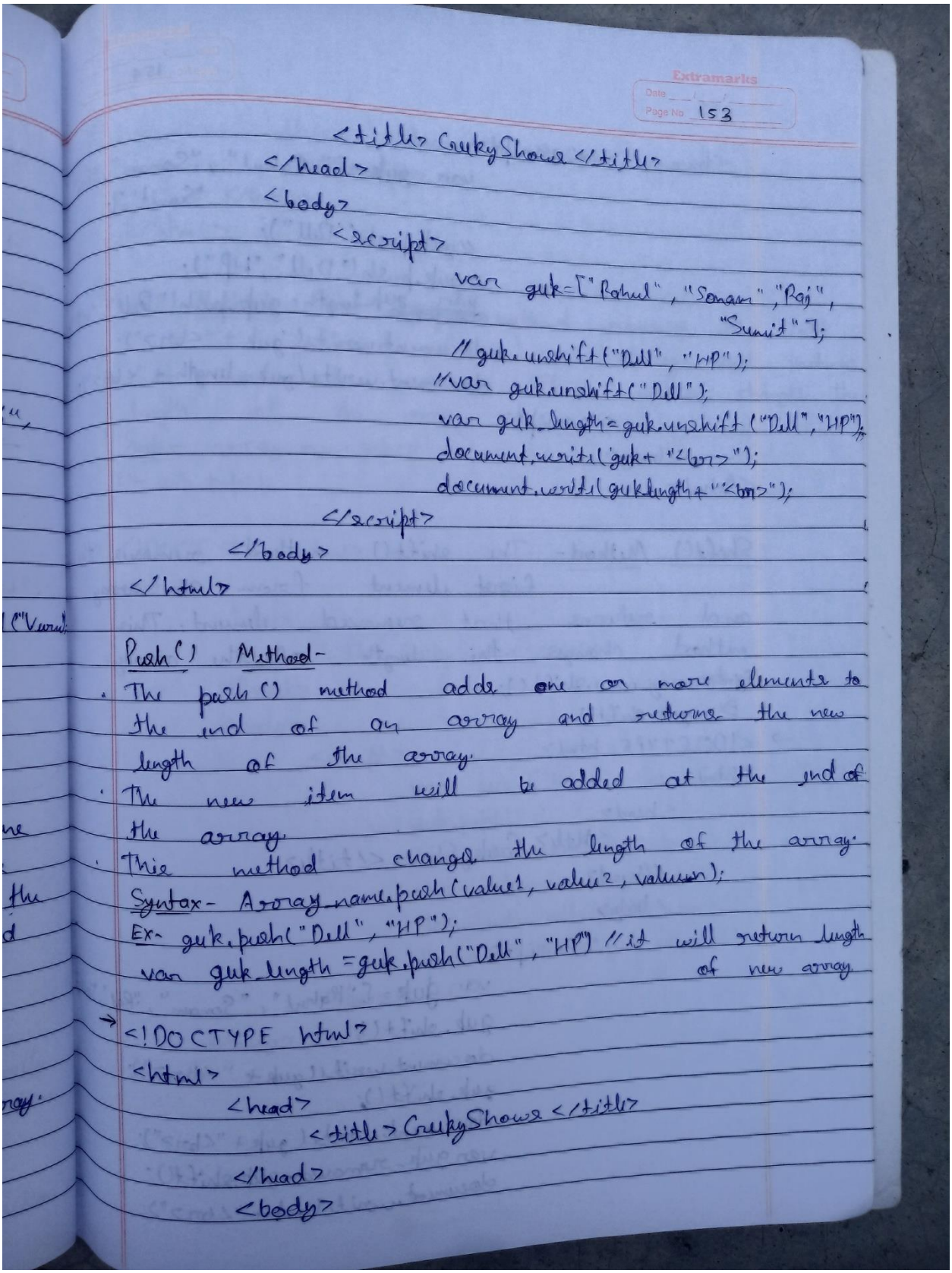
unshift() - Method - The unshift() method adds one or more elements to the beginning of an array and returns the new length of the array. This method changes the length of an array.

Syntax - Array_name.unshift(value1, value2, value_n);

Ex - geek.unshift("Dell", "HP");

var geek_length = geek.unshift("Dell", "HP"); // it will return length of new array.

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



```
<title> CrukyShows </title>
</head>
<body>
<script>
var guk=["Rahul","Sonam","Raj",
"Sumit"];
// guk.unshift("Dell","HP");
//var guk.unshift("Dell");
var guk_length=guk.unshift("Dell","HP");
document.write(guk+"<br>");
document.write(guk.length+"<br>");
</script>
</body>
</html>
```

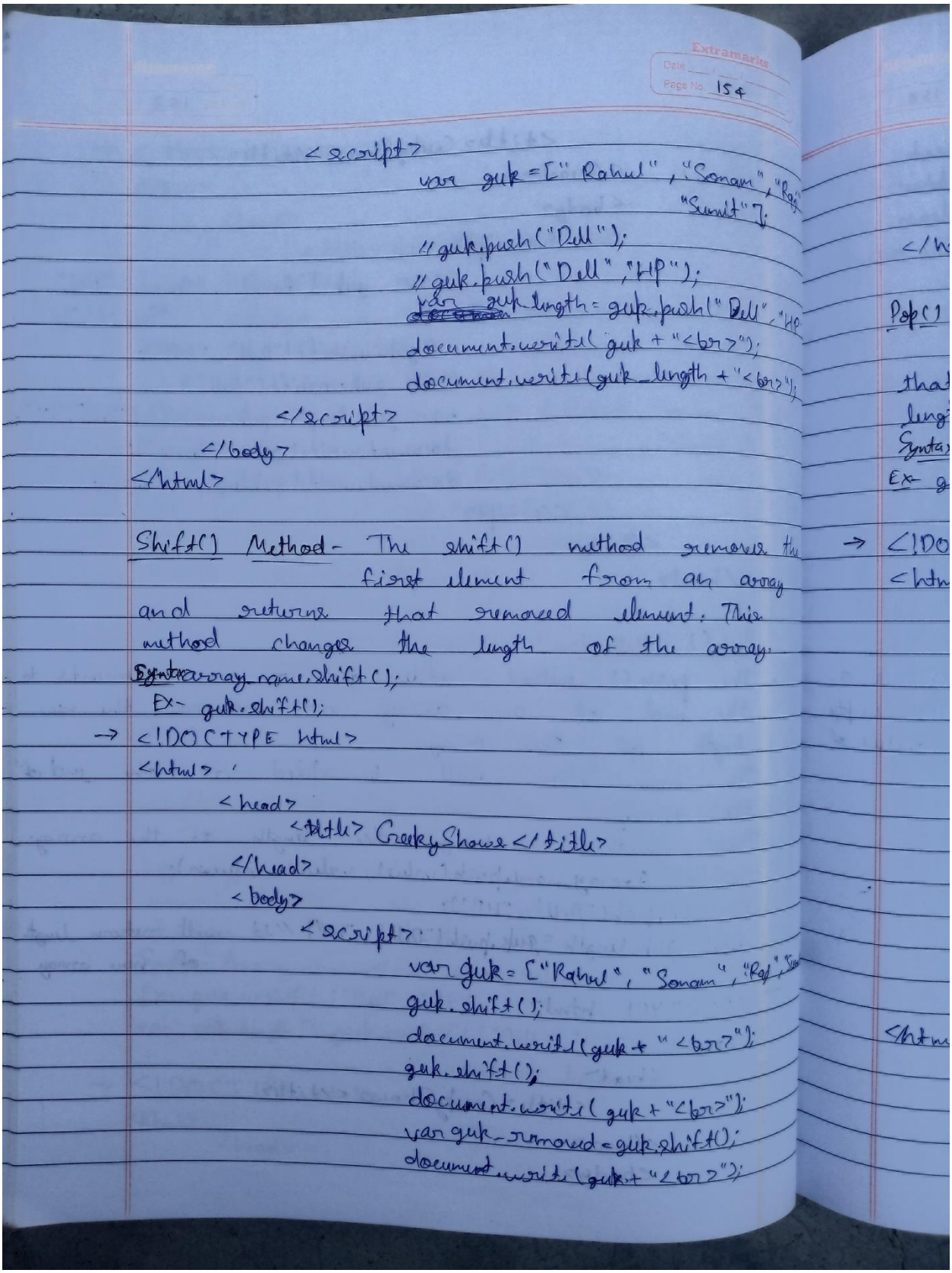
Push() Method-

- The push() method adds one or more elements to the end of an array and returns the new length of the array.
- The new item will be added at the end of the array.
- This method changes the length of the array.

Syntax- Array_name.push(value1, value2, valuen);

Ex- guk.push("Dell","HP");
var guk_length=guk.push("Dell","HP") // it will return length of new array.

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



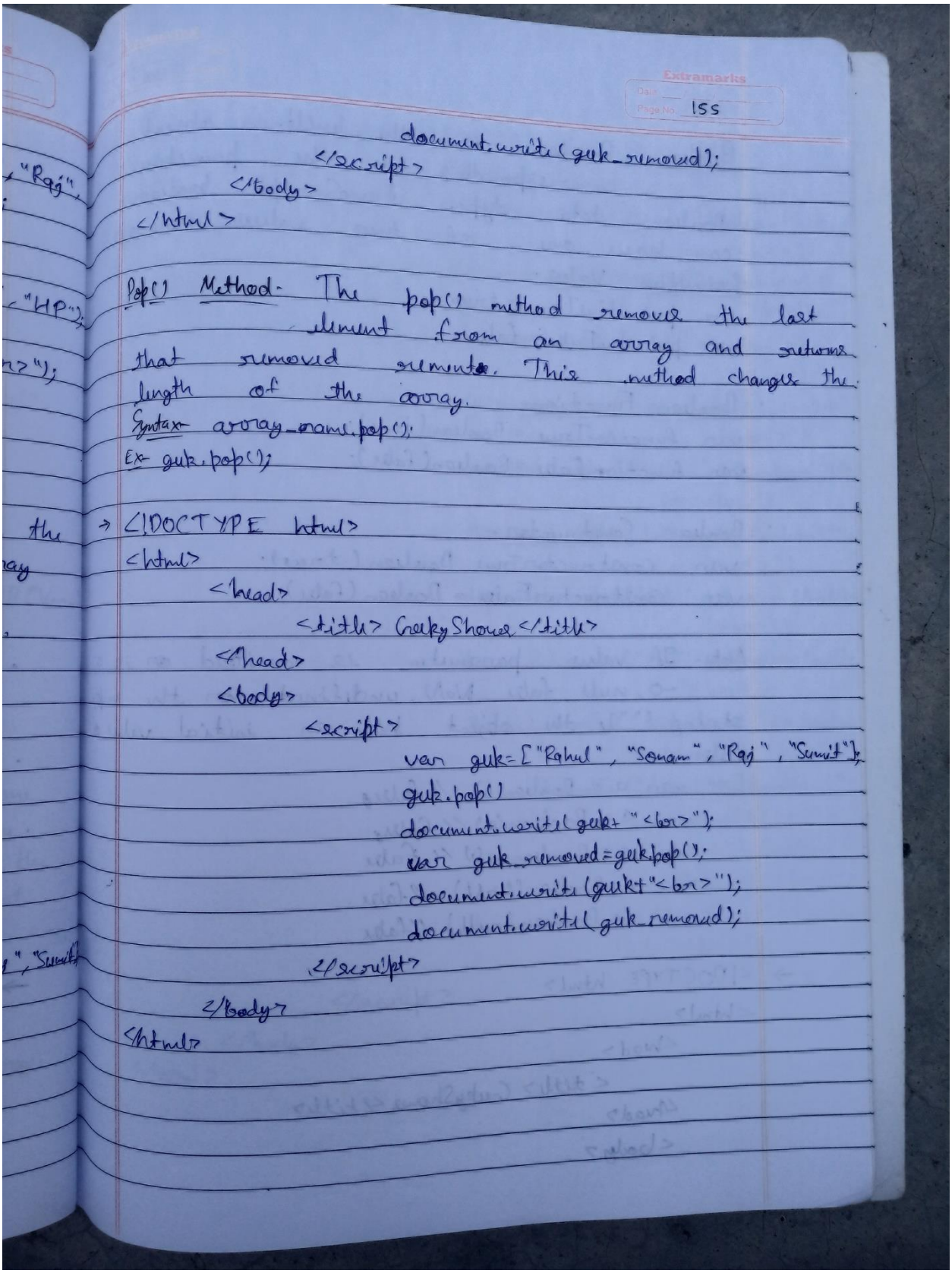
```
<script>
    var guk = [" Rahul", "Sonam", "Raj",
               "Sumit"];
    // guk.push("Dell");
    // guk.push("Dell","HP");
    var guk_length = guk.push("Dell","HP");
    document.write(guk + "<br>");
    document.write(guk_length + "<br>");
</script>
</body>
</html>
```

Shift() Method - The shift() method removes the first element from an array and returns that removed element. This method changes the length of the array.

Syntax: array_name.shift();

Ex- guk.shift();

```
<!DOCTYPE html>
<html>
    <head>
        <title> GeekyShows </title>
    </head>
    <body>
        <script>
            var guk = ["Rahul", "Sonam", "Raj", "Sumit"];
            guk.shift();
            document.write(guk + "<br>");
            guk.shift();
            document.write(guk + "<br>");
            var guk_removed = guk.shift();
            document.write(guk + "<br>");
```



Extramarks
Page No. 155

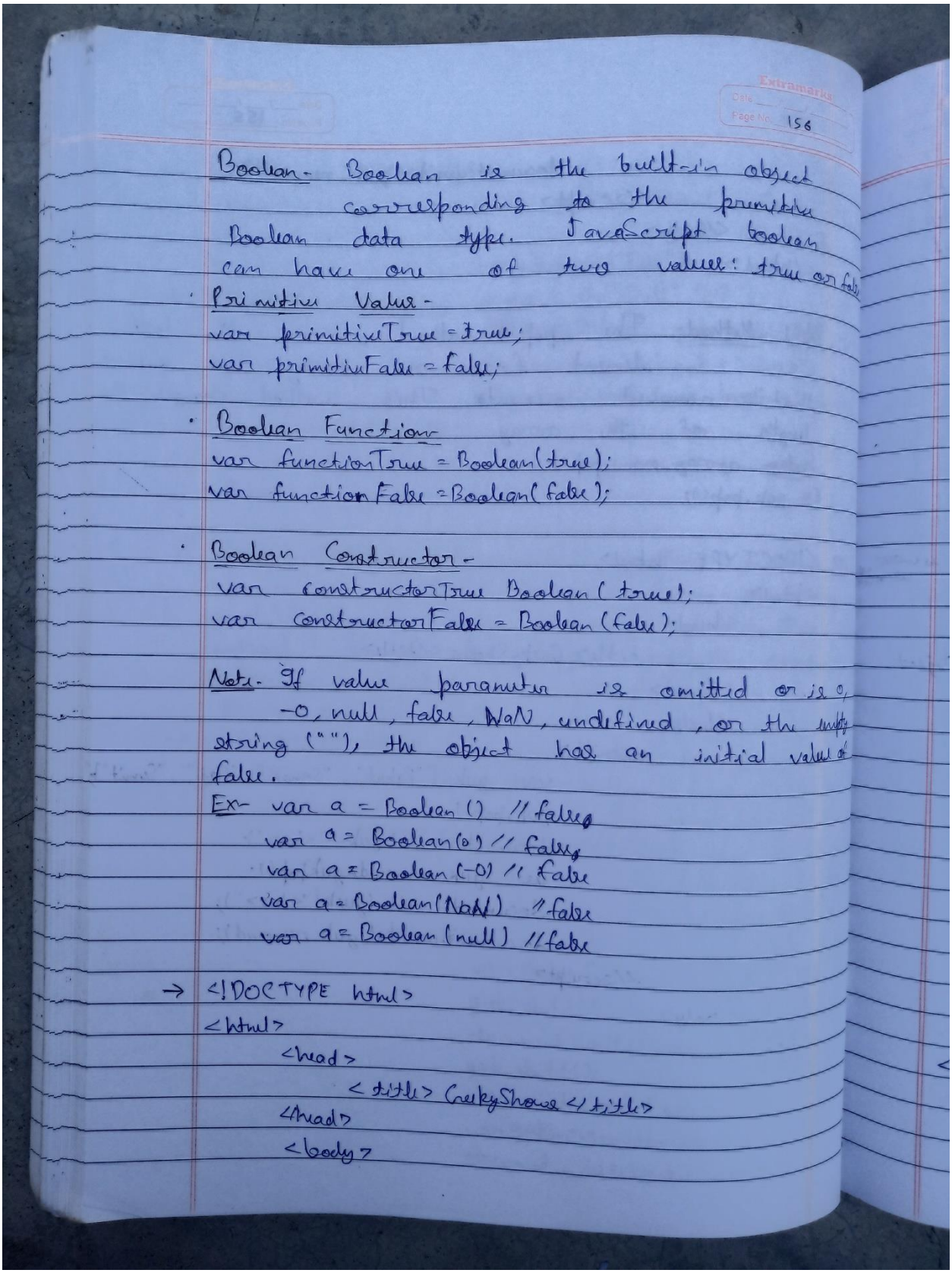
```
        document.write(guk_removed);
    </script>
</body>
</html>
```

Pop() Method- The pop() method removes the last element from an array and returns that removed element. This method changes the length of the array.

Syntax- array_name.pop();

Ex- guk.pop();

```
<!DOCTYPE html>
<html>
    <head>
        <title> Geeky Shows </title>
    </head>
    <body>
        <script>
            var guk=["Rahul", "Sonam", "Raj", "Sumit"];
            guk.pop()
            document.write(guk+ "<br>");
            var guk_removed=guk.pop();
            document.write(guk+"<br>");
            document.write(guk_removed);
        </script>
    </body>
</html>
```



Boolean - Boolean is the built-in object corresponding to the primitive Boolean data type. JavaScript boolean can have one of two values: true or false.

- Primitive Value -

```
var primitiveTrue = true;
var primitiveFalse = false;
```

- Boolean Function

```
var functionTrue = Boolean(true);
var functionFalse = Boolean(false);
```

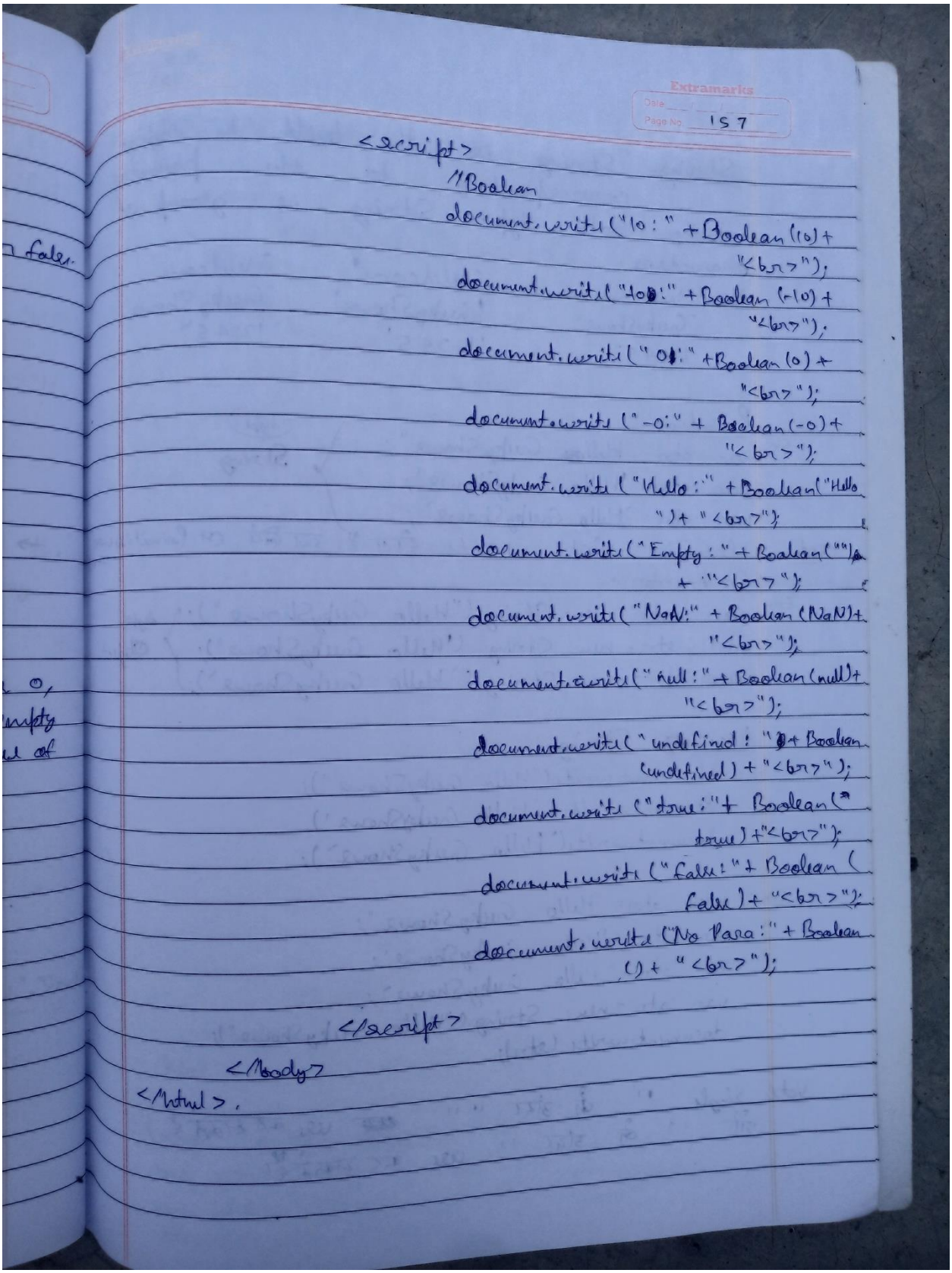
- Boolean Constructor -

```
var constructorTrue Boolean(true);
var constructorFalse = Boolean(false);
```

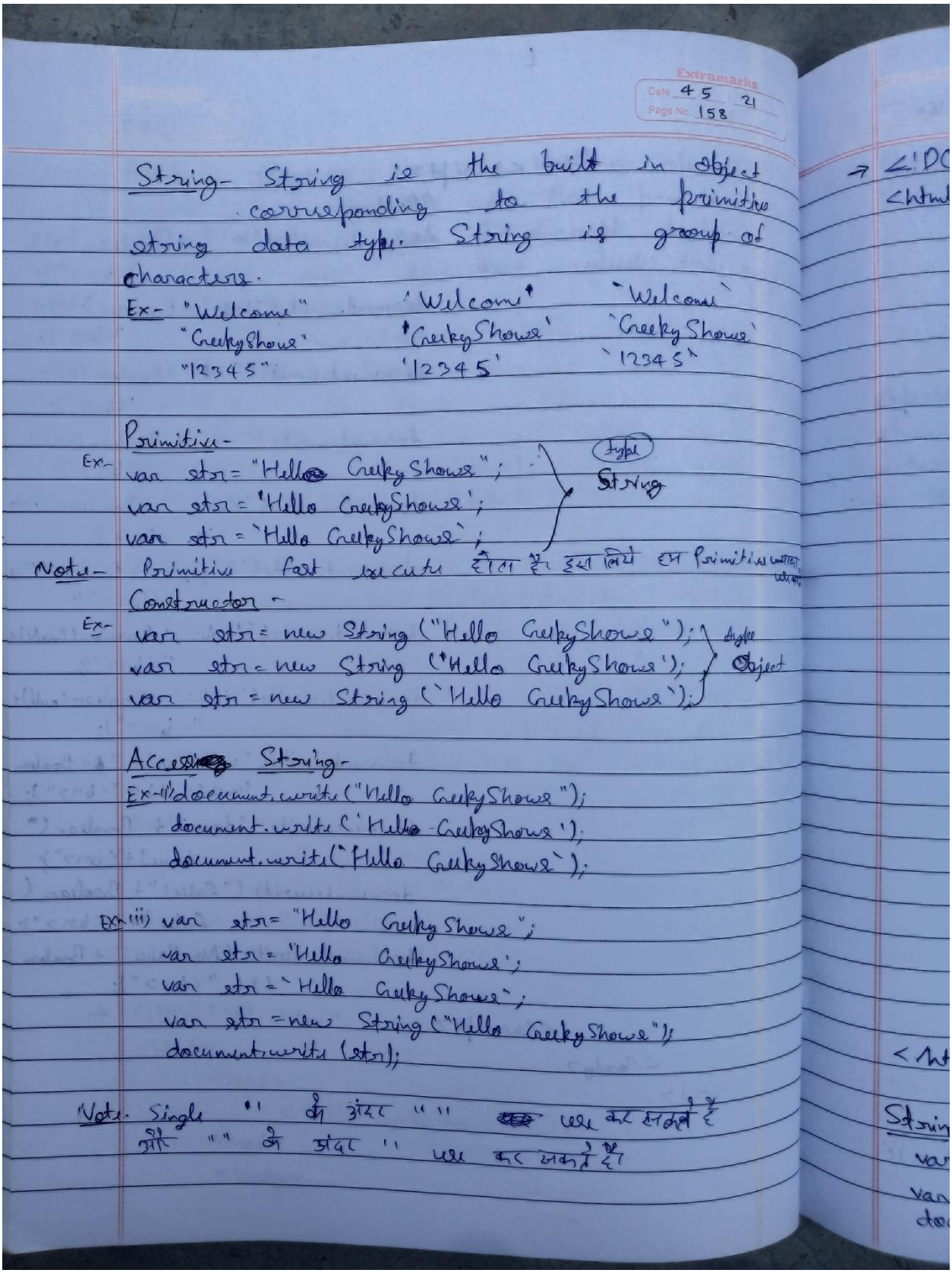
Note - If value parameter is omitted or is 0, -0, null, false, NaN, undefined, or the empty string (""), the object has an initial value of false.

Ex-
```
var a = Boolean() // false
var a = Boolean(0) // false
var a = Boolean(-0) // false
var a = Boolean(NaN) // false
var a = Boolean(null) // false
```

→
```
<!DOCTYPE html>
<html>
    <head>
        <title> GeekyShows </title>
    <head>
    <body>
```



```
<script>
//Boolean
document.write("10:" + Boolean(10) +
"<br>");
document.write("-10:" + Boolean(-10) +
"<br>");
document.write("0:" + Boolean(0) +
"<br>");
document.write("-0:" + Boolean(-0) +
"<br>");
document.write("Hello:" + Boolean("Hello
") + "<br>");
document.write("Empty:" + Boolean("")
+ "<br>");
document.write("NaN:" + Boolean(NaN) +
"<br>");
document.write("null:" + Boolean(null) +
"<br>");
document.write("undefined:" + Boolean
(undefined) + "<br>");
document.write("true:" + Boolean("
true") + "<br>");
document.write("false:" + Boolean(
false) + "<br>");
document.write("No Para:" + Boolean
() + "<br>");
</script>
</body>
</html>
```



String- String is the built in object corresponding to the primitive string data type. String is group of characters.

Ex- "Welcome" 'Welcome' `Welcome`
"GeekyShows" 'GeekyShows' `GeekyShows`
"12345" '12345' `12345`

Primitive -

Ex- var str = "Hello GeekyShows"; } type String
var str = 'Hello GeekyShows';
var str = `Hello GeekyShows`;

Note- Primitive fast execute होता है, इसलिये हम Primitive use करते हैं

Constructor -

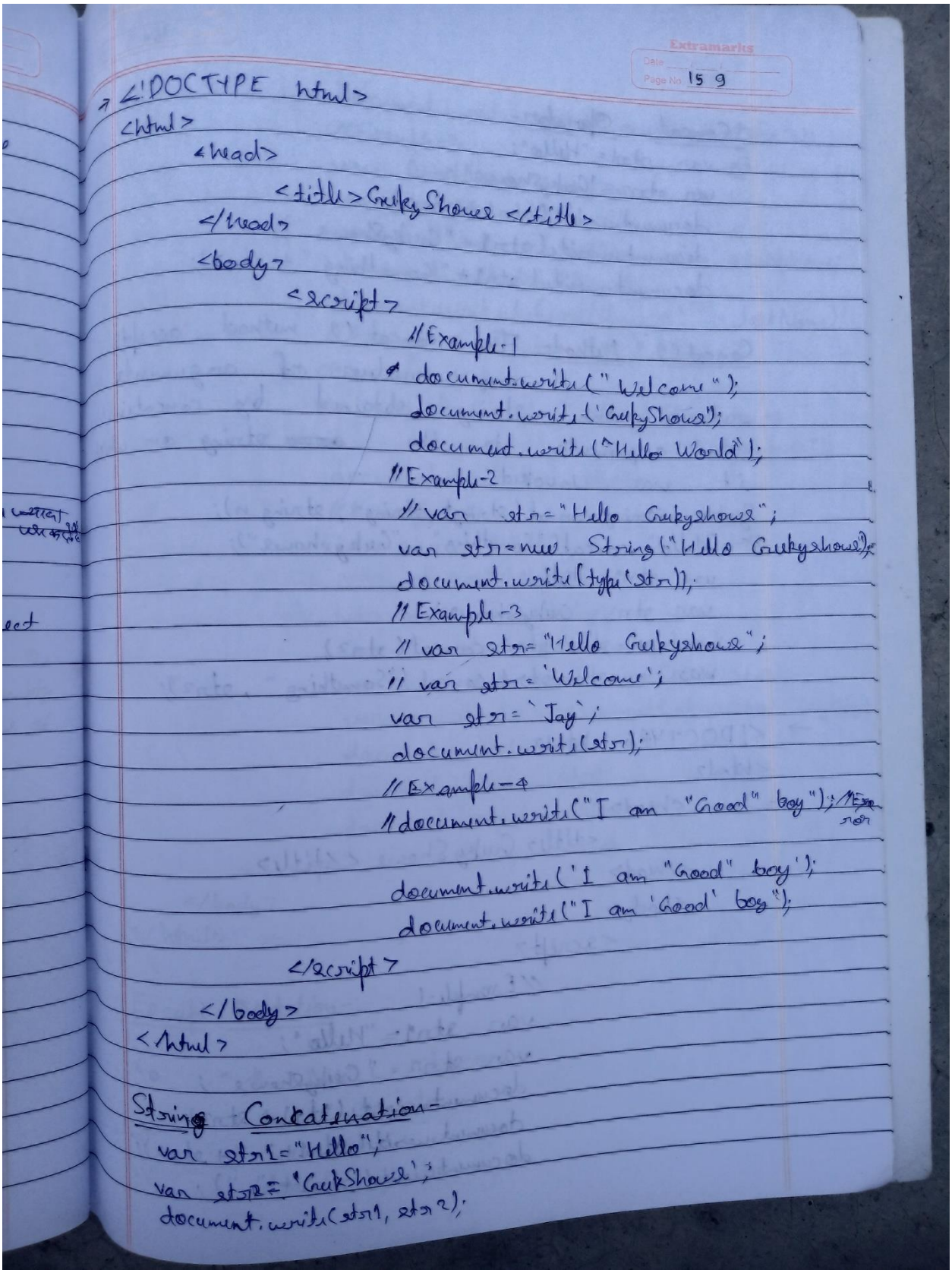
Ex- var str = new String ("Hello GeekyShows"); } type Object
var str = new String ('Hello GeekyShows');
var str = new String (`Hello GeekyShows`);

Access String -

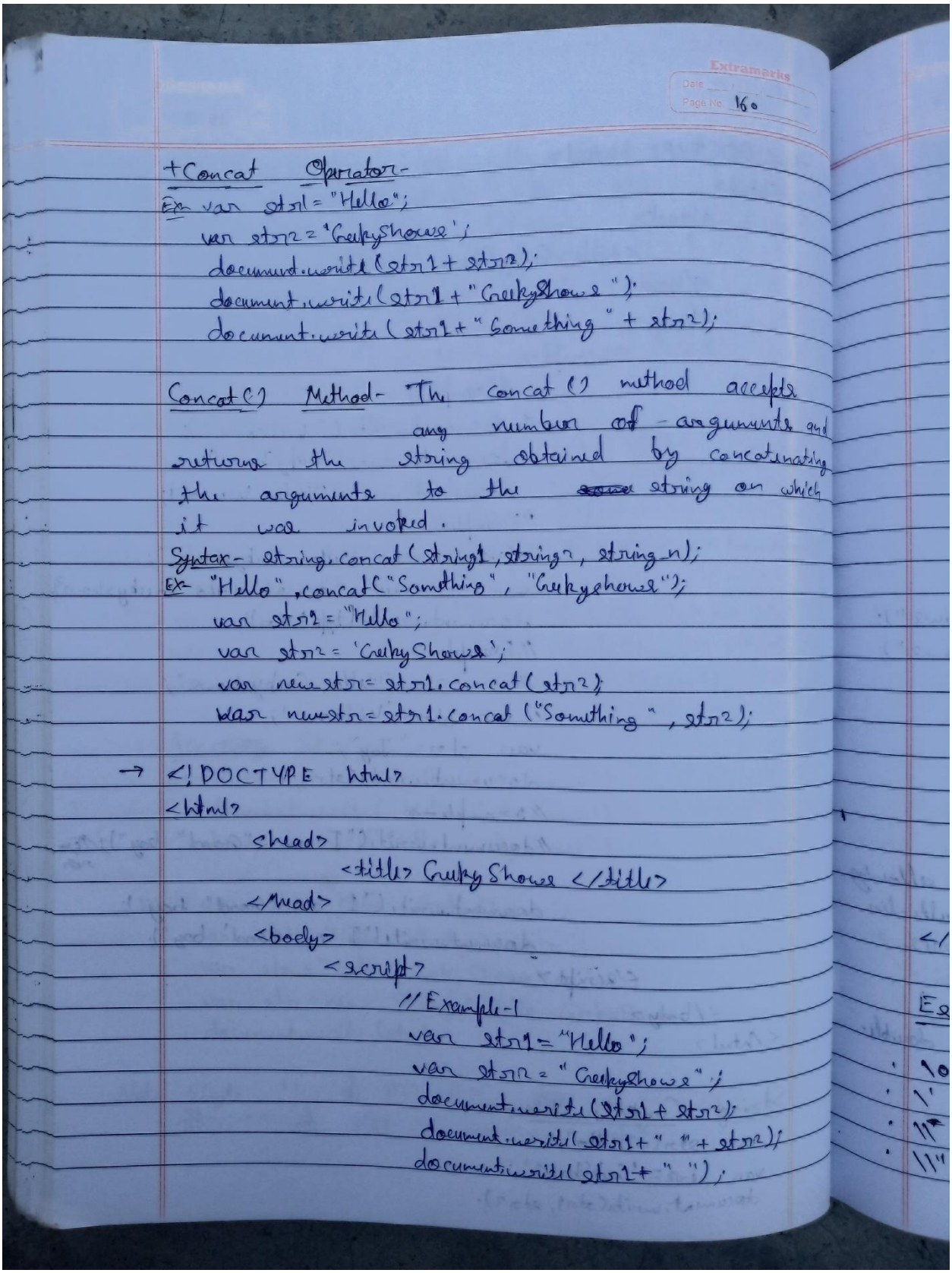
Ex- (i) document.write ("Hello GeekyShows");
document.write ('Hello GeekyShows');
document.write (`Hello GeekyShows`);

Ex (ii) var str = "Hello GeekyShows";
var str = 'Hello GeekyShows';
var str = `Hello GeekyShows`;
var str = new String ("Hello GeekyShows");
document.write (str);

Note- Single '' के अंदर "" use कर सकते हैं और "" के अंदर '' use कर सकते हैं



```
<!DOCTYPE html>
<html>
    <head>
        <title> GeekyShows </title>
    </head>
    <body>
        <script>
            // Example-1
            document.write("Welcome");
            document.write('GeekyShows');
            document.write(`Hello World`);
            // Example-2
            // var str = "Hello Geekyshows";
            var str = new String("Hello Geekyshows");
            document.write(typeof(str));
            // Example-3
            // var str = "Hello Geekyshows";
            // var str = 'Welcome';
            var str = `Jay`;
            document.write(str);
            // Example-4
            // document.write("I am "Good" boy");
            document.write('I am "Good" boy');
            document.write("I am 'Good' boy");
        </script>
    </body>
</html>

String Concatenation-
var str1 = "Hello";
var str2 = 'GeekShows';
document.write(str1, str2);
```



+Concat Operator-

Ex- var str1 = "Hello";
var str2 = 'GeekyShows';
document.write(str1 + str2);
document.write(str1 + "GeekyShows");
document.write(str1 + "Something" + str2);

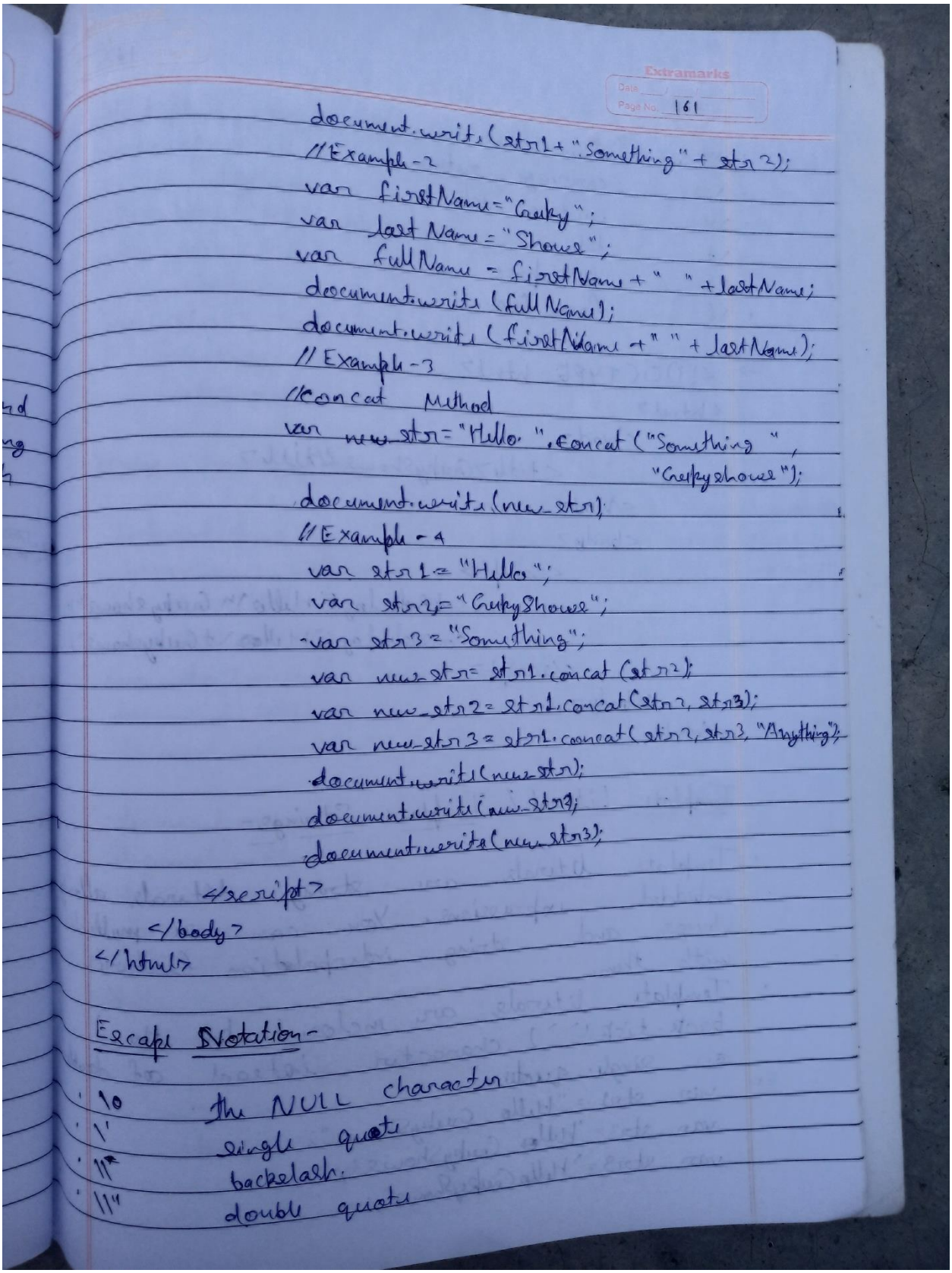
Concat() Method- The concat() method accepts any number of arguments and returns the string obtained by concatenating the arguments to the string on which it was invoked.

Syntax- string.concat(string1, string2, string_n);

Ex- "Hello".concat("Something", "Geekyshows");

var str2 = "Hello";
var str2 = 'GeekyShows';
var newstr = str1.concat(str2);
var newstr = str1.concat("Something", str2);

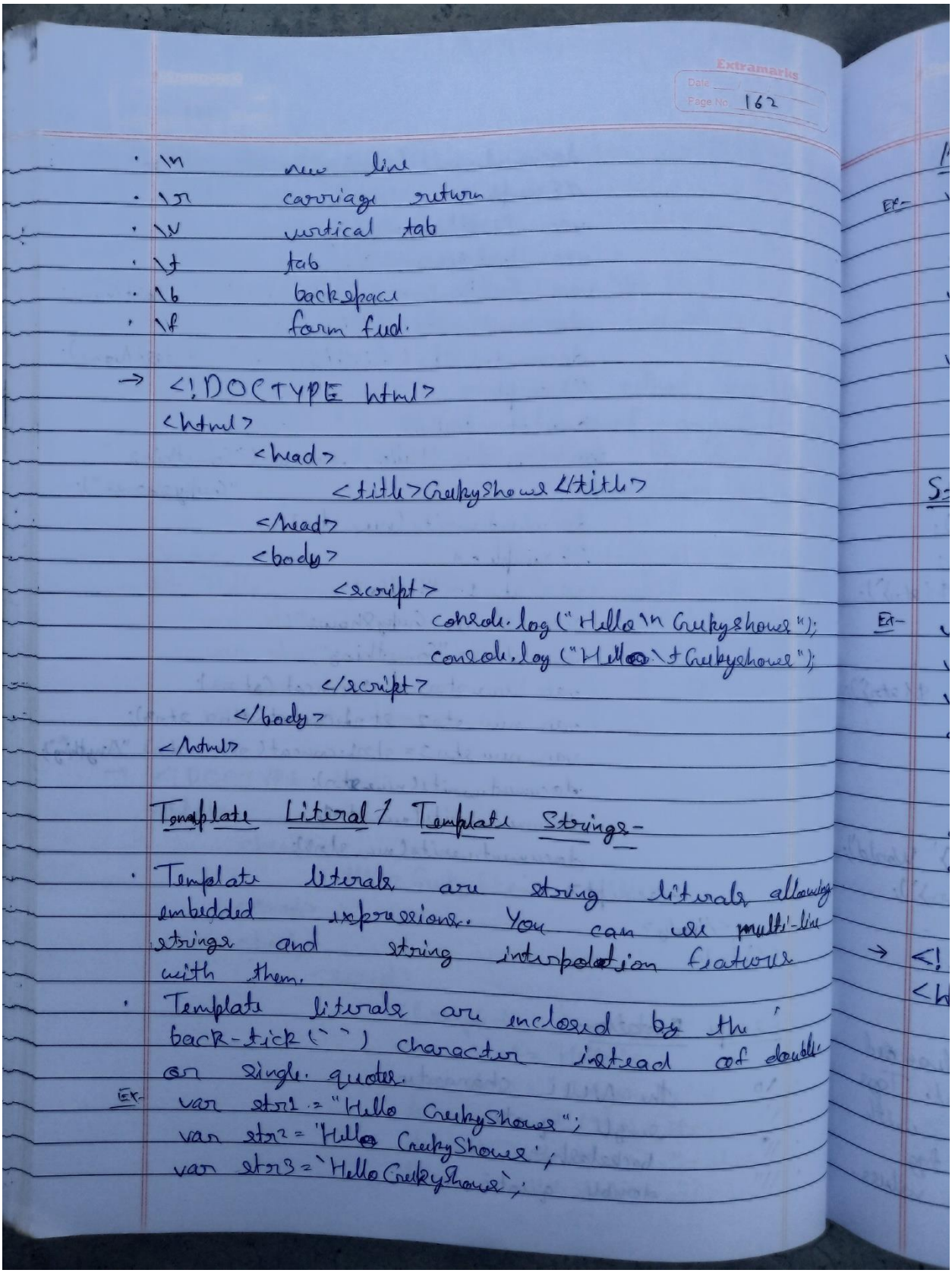
→ <!DOCTYPE html>
<html>
<head>
<title> GeekyShows </title>
</head>
<body>
<script>
// Example-1
var str1 = "Hello";
var str2 = "GeekyShows";
document.write(str1 + str2);
document.write(str1 + " " + str2);
document.write(str1 + " ");



```
document.write(str1+" Something "+str2);
//Example-2
var firstName="Cuky";
var lastName="Shows";
var fullName = firstName+" "+lastName;
document.write(fullName);
document.write(firstName+" "+lastName);
//Example-3
//concat Method
var new_str="Hello ".concat("Something ",
                            "CukyShows");
document.write(new_str);
//Example-4
var str1="Hello ";
var str2="CukyShows";
var str3="Something";
var new_str=str1.concat(str2);
var new_str2=str1.concat(str2,str3);
var new_str3=str1.concat(str2,str3,"Anything");
document.write(new_str);
document.write(new_str2);
document.write(new_str3);
</script>
</body>
</html>
```

Escape Notation-

\0	the NULL character
\'	single quote
\\	backslash
\"	double quote



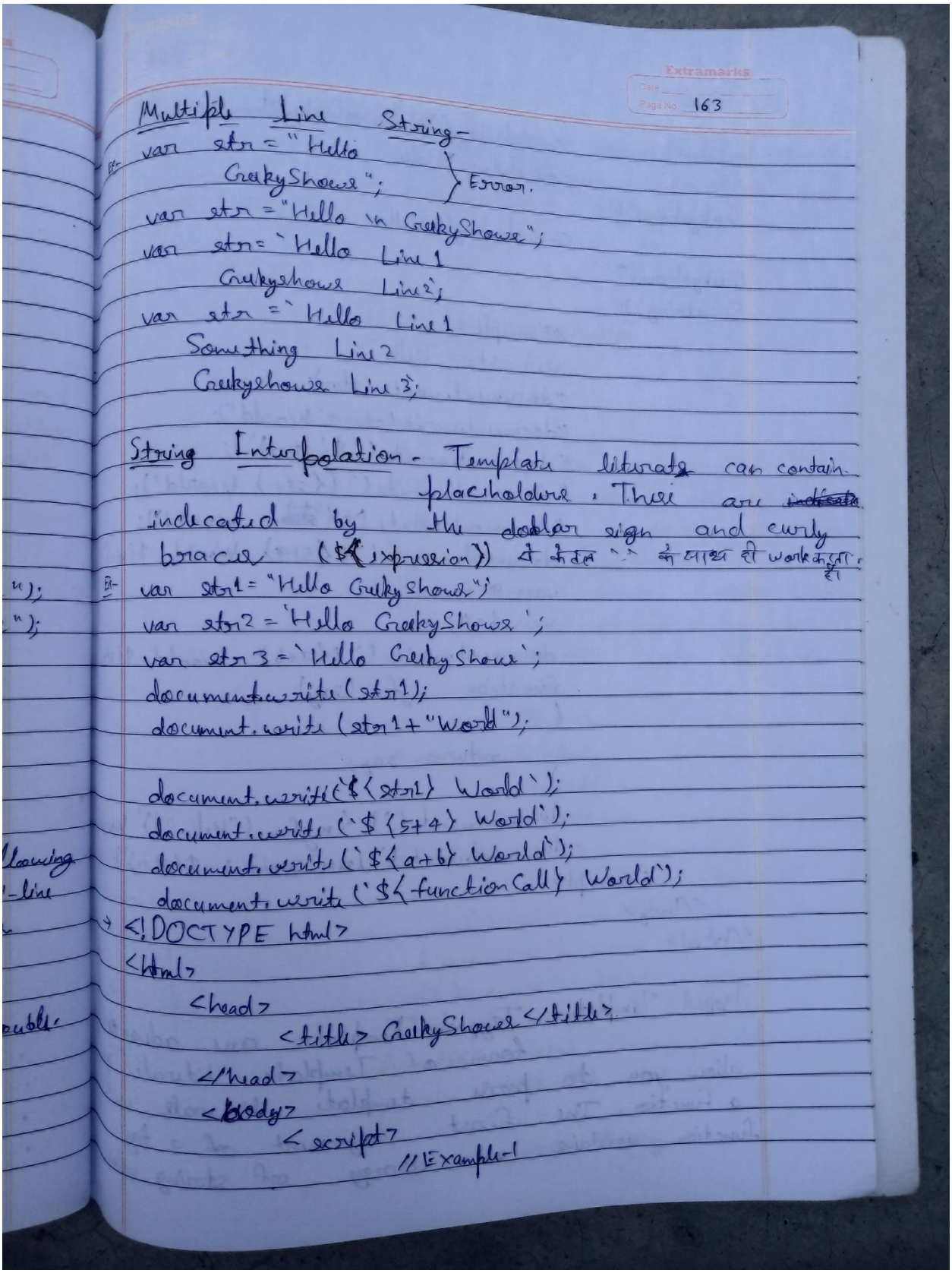
- \n new line
- \r carriage return
- \v vertical tab
- \t tab
- \b backspace
- \f form feed

→
```
<!DOCTYPE html>
<html>
    <head>
        <title>CreakyShows</title>
    </head>
    <body>
        <script>
            console.log("Hello\nCreakyShows");
            console.log("Hello\tCreakyShows");
        </script>
    </body>
</html>
```

Template Literal / Template Strings-

- Template literals are string literals allowing embedded expressions. You can use multi-line strings and string interpolation features with them.
- Template literals are enclosed by the back-tick (``) character instead of double or single quotes.

Ex-
```
var str1 = "Hello CreakyShows";
var str2 = 'Hello CreakyShows';
var str3 = `Hello CreakyShows`;
```



Extramarks
Page No. 163

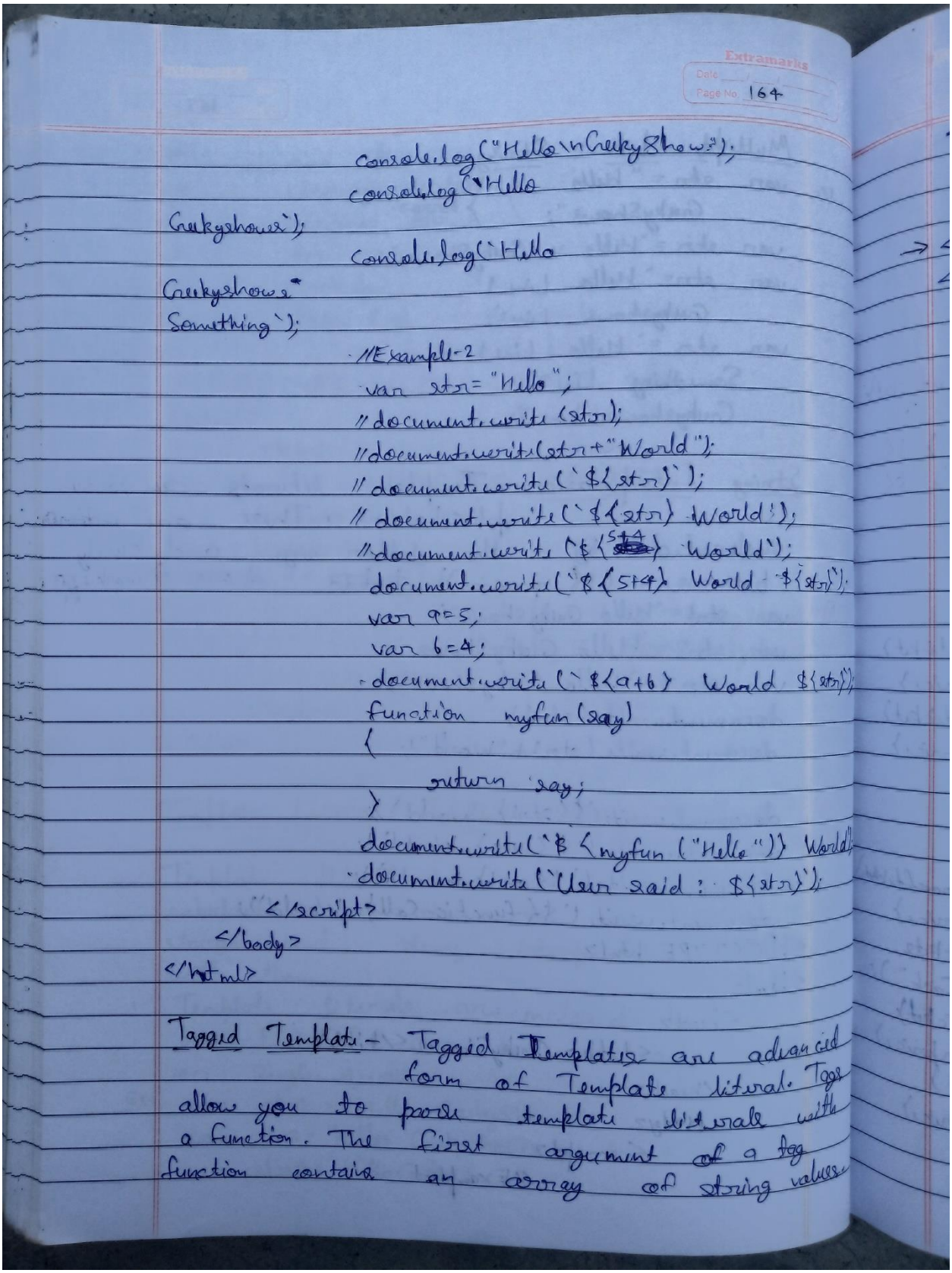
Multiple Line String -

```
var str = "Hello
    GeekyShows";        } Error.
var str = "Hello \n GeekyShows";
var str = `Hello Line 1
    Geekyshows Line 2`;
var str = `Hello Line 1
  Something Line 2
  Geekyshows Line 3`;
```

String Interpolation - Template literals can contain placeholders. These are indicated by the dollar sign and curly braces (${expression}) ~~के साथ~~ के साथ ही work करता है।

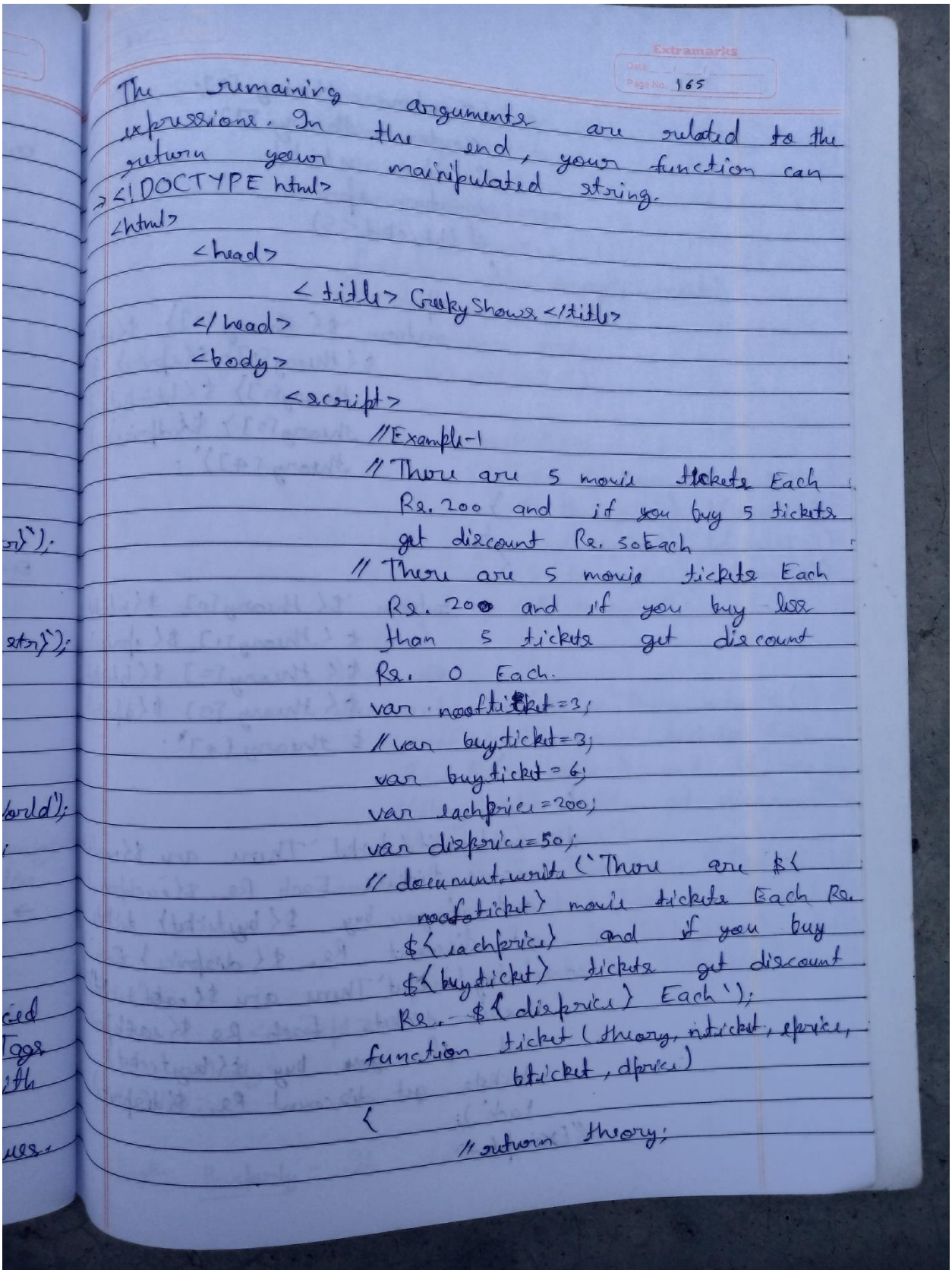
```
var str1 = "Hello GeekyShows";
var str2 = 'Hello GeekyShows';
var str3 = `Hello GeekyShows`;
document.write(str1);
document.write(str1+"World");

document.write(`${str1} World`);
document.write(`${5+4} World`);
document.write(`${a+b} World`);
document.write(`${functionCall} World`);
```

```
<!DOCTYPE html>
<Html>
  <head>
    <title> GeekyShows </title>
  </head>
  <body>
    <script>
      //Example-1
```



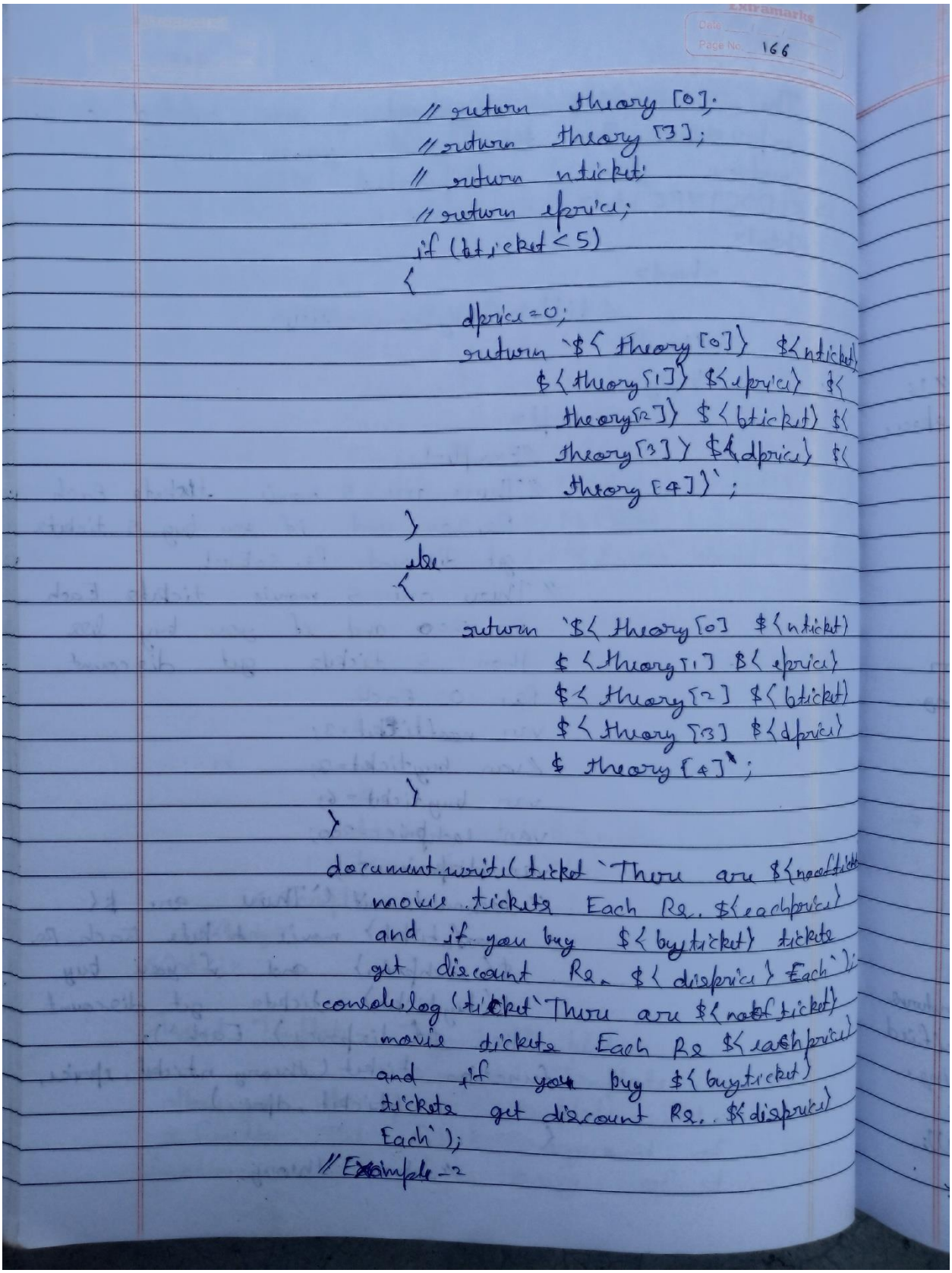
```
console.log("Hello\nGeekyShows");
console.log(`Hello
Geekyshows`);
console.log(`Hello
Geekyshows
Something`);
//Example-2
var str = "Hello";
//document.write(str);
//document.write(str+"World");
//document.write(`${str}`);
//document.write(`${str} World!`);
//document.write(`${5+4} World`);
document.write(`${5+4} World ${str}`);
var a=5;
var b=4;
document.write(`${a+b} World ${str}`);
function myfun(say)
{
    return say;
}
document.write(`${myfun("Hello")} World`);
document.write(`User said : ${str}`);
</script>
</body>
</html>
```

Tagged Template - Tagged Templates are advanced form of Template literal. Tags allow you to parse template literals with a function. The first argument of a tag function contains an array of string values

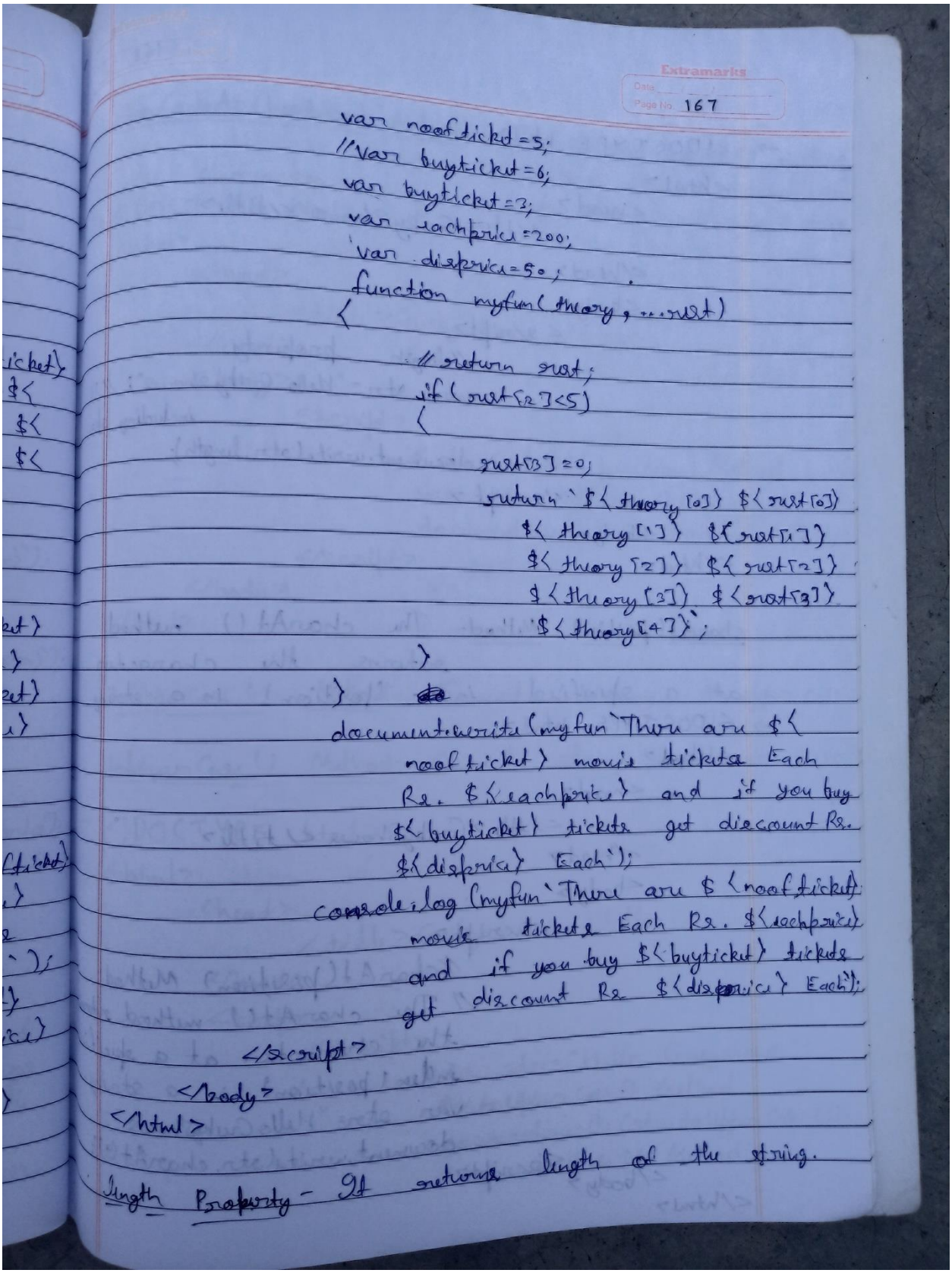


The remaining arguments are related to the expressions. In the end, your function can return your manipulated string.

```
<!DOCTYPE html>
<html>
    <head>
        <title> Geeky Shows. </title>
    </head>
    <body>
        <script>
            //Example-1
            // There are 5 movie tickets Each
               Rs. 200 and if you buy 5 tickets
               get discount Rs. 50 Each
            // There are 5 movie tickets Each
               Rs. 200 and if you buy less
               than 5 tickets get discount
               Rs. 0 Each.
            var noftiket=3;
            // var buyticket=3;
            var buyticket=6;
            var eachprice=200;
            var disprice=50;
            // document.write(`There are ${
               noftiket} movie tickets Each Rs.
               ${eachprice} and if you buy
               ${buyticket} tickets get discount
               Rs. ${disprice} Each`);
            function ticket(theory, ntticket, eprice,
                    btticket, dprice)
            {
                // return theory;
```

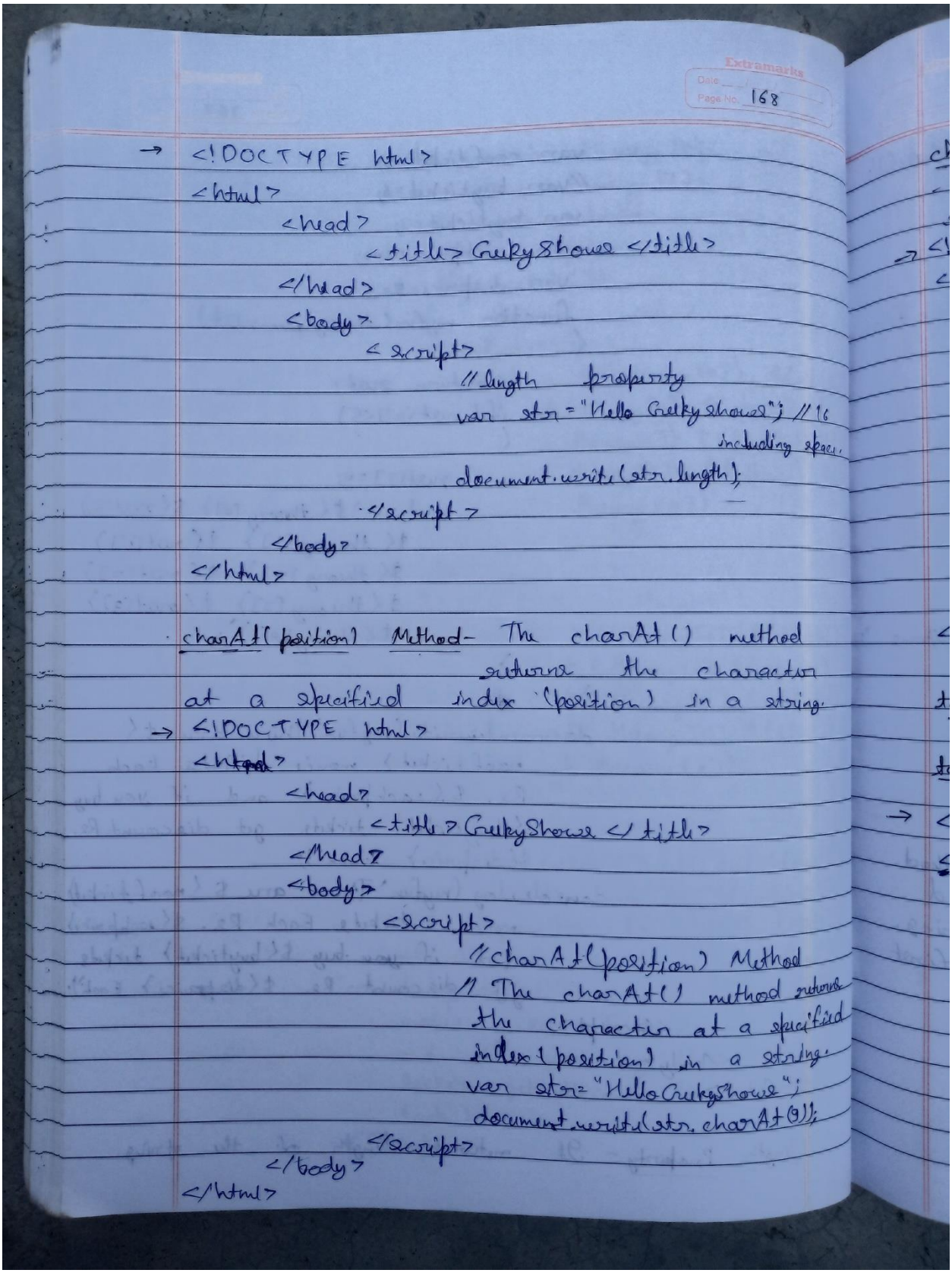


```
// return theory[0];
// return theory[3];
// return nticket;
// return eprice;
if (bticket < 5)
{
    dprice = 0;
    return `${theory[0]} ${nticket}
        ${theory[1]} ${eprice} ${
        theory[2]} ${bticket} ${
        theory[3]} ${dprice} ${
        theory[4]}`;
}
else
{
    return `${theory[0] ${nticket}
        ${theory[1] ${eprice}
        ${theory[2] ${bticket}
        ${theory[3] ${dprice}
        ${theory[4]`;
}
}
document.write(ticket `There are ${noofticket}
movie tickets Each Rs. ${eachprice}
and if you buy ${buyticket} tickets
get discount Rs. ${disprice} Each`);
console.log(ticket`There are ${noofticket}
movie tickets Each Rs ${eachprice}
and if you buy ${buyticket}
tickets get discount Rs. ${disprice}
Each`);
// Example-2
```



```
var noofticket=5;
//var buyticket=6;
var buyticket=3;
var eachprice=200;
var disprice=50;
function myfun(theory, ...rest)
{
    //return rest;
    if(rest[2]<5)
    {
        rest[3]=0;
        return `${theory[0]} ${rest[0]}
        ${theory[1]} ${rest[1]}
        ${theory[2]} ${rest[2]}
        ${theory[3]} ${rest[3]}
        ${theory[4]}`;
    }
}
document.write(myfun`There are ${
    noofticket} movie tickets Each
    Rs. ${eachprice} and if you buy
    ${buyticket} tickets get discount Rs.
    ${disprice} Each`);
console.log(myfun`There are ${noofticket}
    movie tickets Each Rs. ${eachprice}
    and if you buy ${buyticket} tickets
    get discount Rs ${disprice} Each`);
</script>
</body>
</html>
```

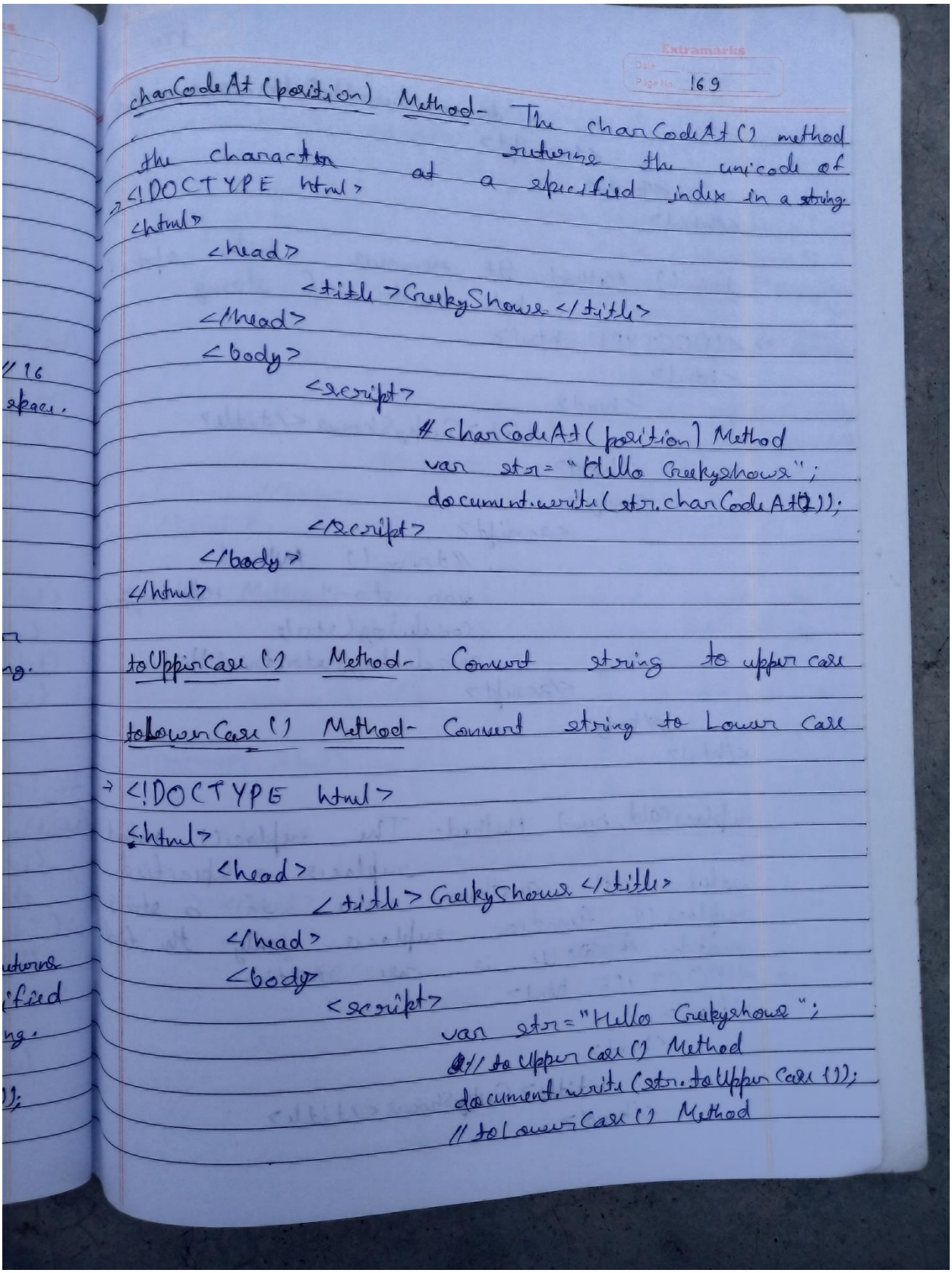
Length Property - It returns length of the string.



```
<!DOCTYPE html>
<html>
    <head>
        <title> GeekyShows </title>
    </head>
    <body>
        <script>
            // length property
            var str = "Hello GeekyShows"; // 16 including space.
            document.write(str.length);
        </script>
    </body>
</html>
```

charAt(position) Method- The charAt() method returns the character at a specified index (position) in a string.

```
<!DOCTYPE html>
<html>
    <head>
        <title> GeekyShows </title>
    </head>
    <body>
        <script>
            // charAt(position) Method
            // The charAt() method returns the character at a specified index (position) in a string.
            var str = "Hello GeekyShows";
            document.write(str.charAt(9));
        </script>
    </body>
</html>
```



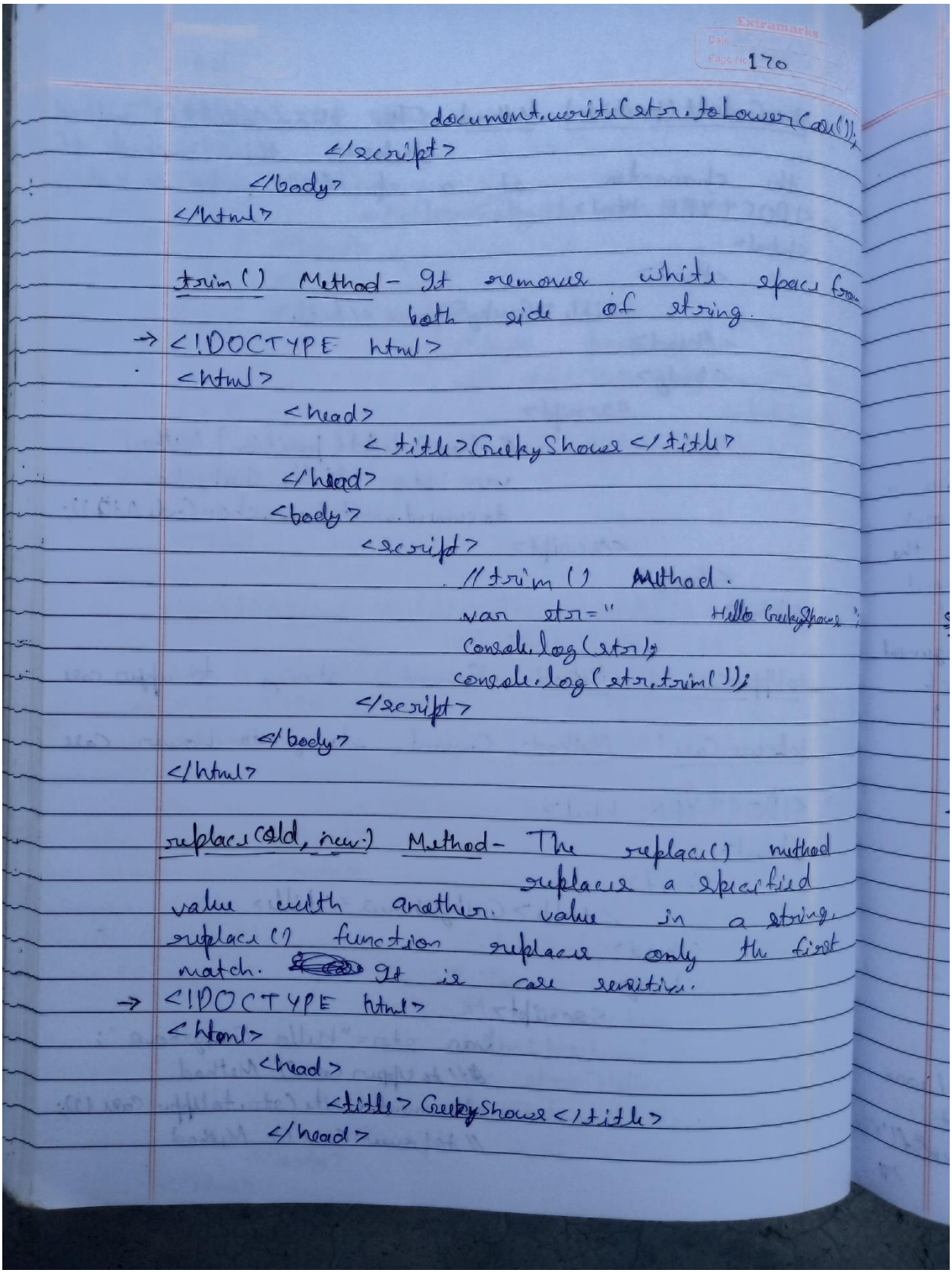
charCodeAt(position) Method- The charCodeAt() method returns the unicode of the character at a specified index in a string.

```
<!DOCTYPE html>
<html>
    <head>
        <title>GeekyShows</title>
    </head>
    <body>
        <script>
            // charCodeAt(position) Method
            var str = "Hello Geekyshows";
            document.write(str.charCodeAt(2));
        </script>
    </body>
</html>
```

toUpperCase() Method- Convert string to upper case

toLowerCase() Method- Convert string to Lower case

```
<!DOCTYPE html>
<html>
    <head>
        <title>GeekyShows</title>
    </head>
    <body>
        <script>
            var str = "Hello Geekyshows";
            // toUpperCase() Method
            document.write(str.toUpperCase());
            // toLowerCase() Method
```



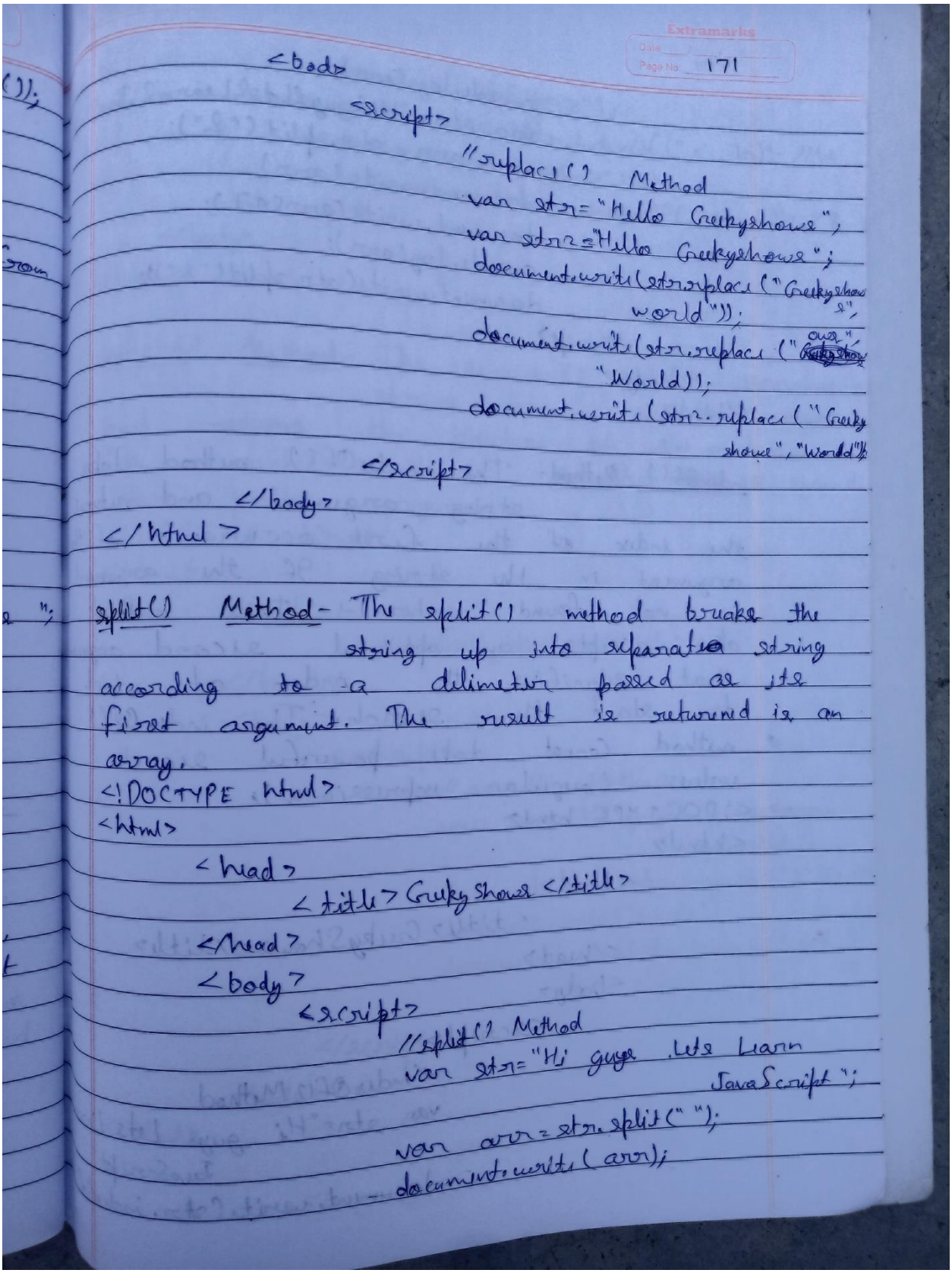
```
                document.write(str.toLowerCase());
            </script>
        </body>
    </html>
```

trim() Method - It removes white space from both side of string.

```
<!DOCTYPE html>
<html>
    <head>
        <title>GeekyShows</title>
    </head>
    <body>
        <script>
            // trim() Method.
            var str="    Hello GeekyShows    ";
            console.log(str);
            console.log(str.trim());
        </script>
    </body>
</html>
```

replace(old, new) Method - The replace() method replaces a specified value with another value in a string. replace() function replaces only the first match. It is case sensitive.

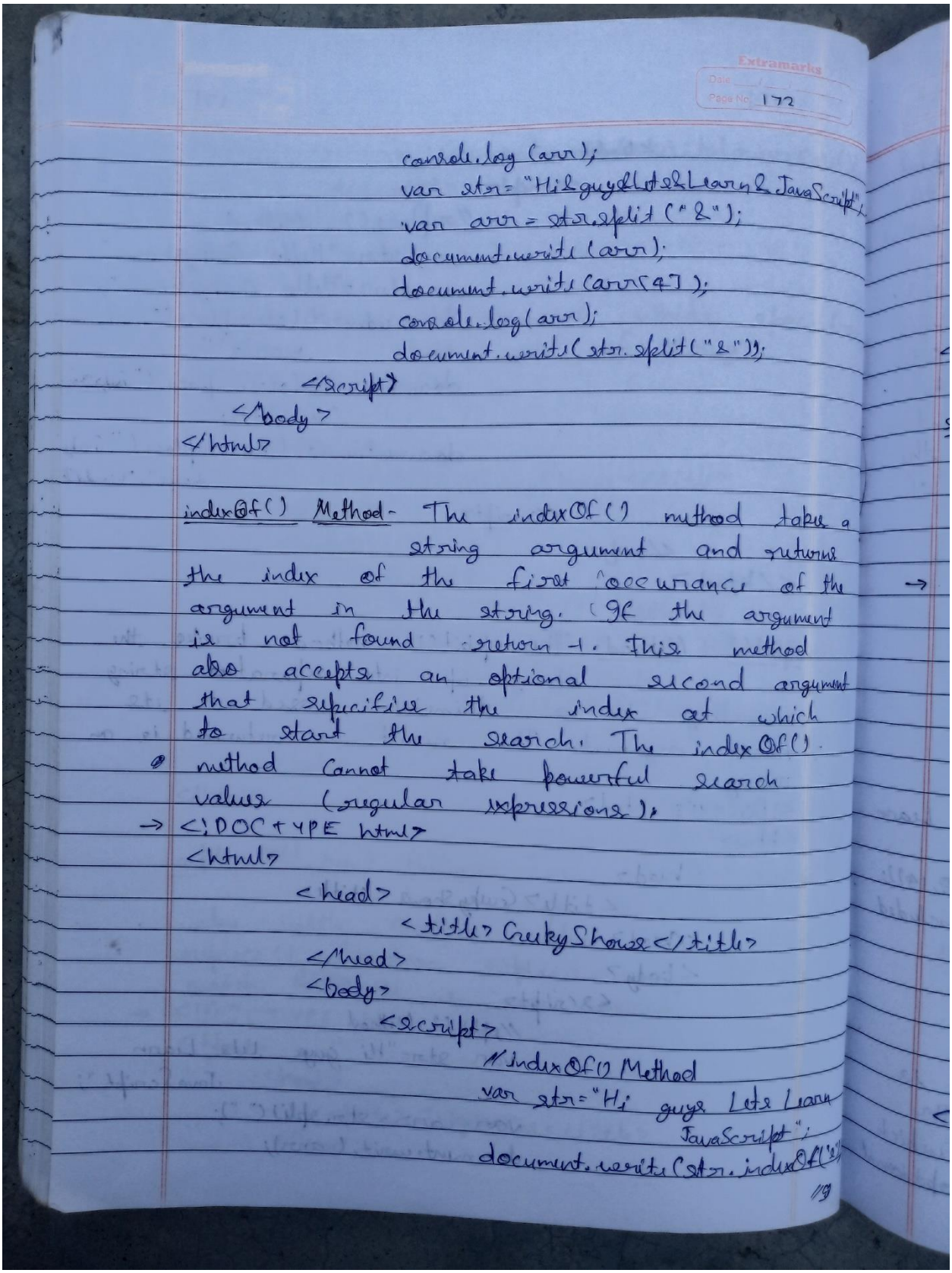
```
<!DOCTYPE html>
<html>
    <head>
        <title>GeekyShows</title>
    </head>
```



```
<body>
    <script>
        //replace() Method
        var str="Hello Geekyshows";
        var str2="Hello Geekyshows";
        document.write(str.replace("Geekyshows","world"));
        document.write(str.replace("Geekyshows","World"));
        document.write(str2.replace("Geekyshows","World"));
    </script>
</body>
</html>
```

split() Method - The split() method breaks the string up into separate string according to a delimeter passed as its first argument. The result is returned in an array.

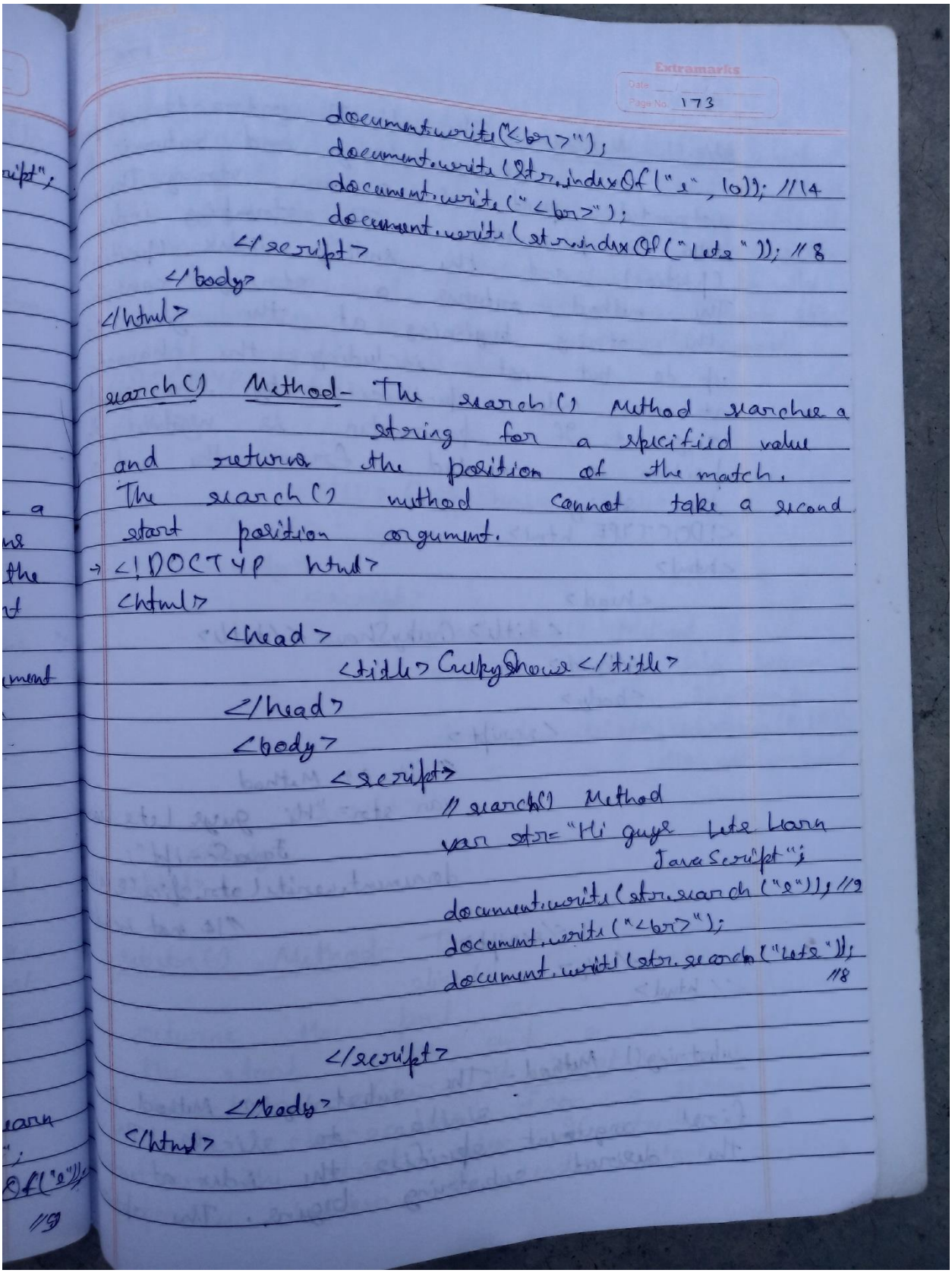
```
<!DOCTYPE html>
<html>
    <head>
        <title> Geeky Shows </title>
    </head>
    <body>
        <script>
            //split() Method
            var str="Hi guys Lets Learn JavaScript";
            var arr=str.split(" ");
            document.write(arr);
```



```
        console.log(arr);
        var str = "Hi guys Lets Learn & JavaScript";
        var arr = str.split("&");
        document.write(arr);
        document.write(arr[4]);
        console.log(arr);
        document.write(str.split("&"));
    </script>
</body>
</html>
```

indexOf() Method - The indexOf() method take a string argument and returns the index of the first occurance of the argument in the string. If the argument is not found return -1. This method also accepts an optional second argument that specifies the index at which to start the search. The indexOf() method cannot take powerful search values (regular expressions).

```
<!DOCTYPE html>
<html>
    <head>
        <title> CrukyShows </title>
    </head>
    <body>
        <script>
            // indexOf() Method
            var str = "Hi guys Lets Learn
                JavaScript";
            document.write(str.indexOf("&"));
```



```
document.write("<br>");
document.write(str.indexOf("s", 10)); //14
document.write("<br>");
document.write(str.indexOf("Lets")); //8
</script>
</body>
</html>
```

search() Method- The search() Method searches a string for a specified value and returns the position of the match. The search() method cannot take a second start position argument.

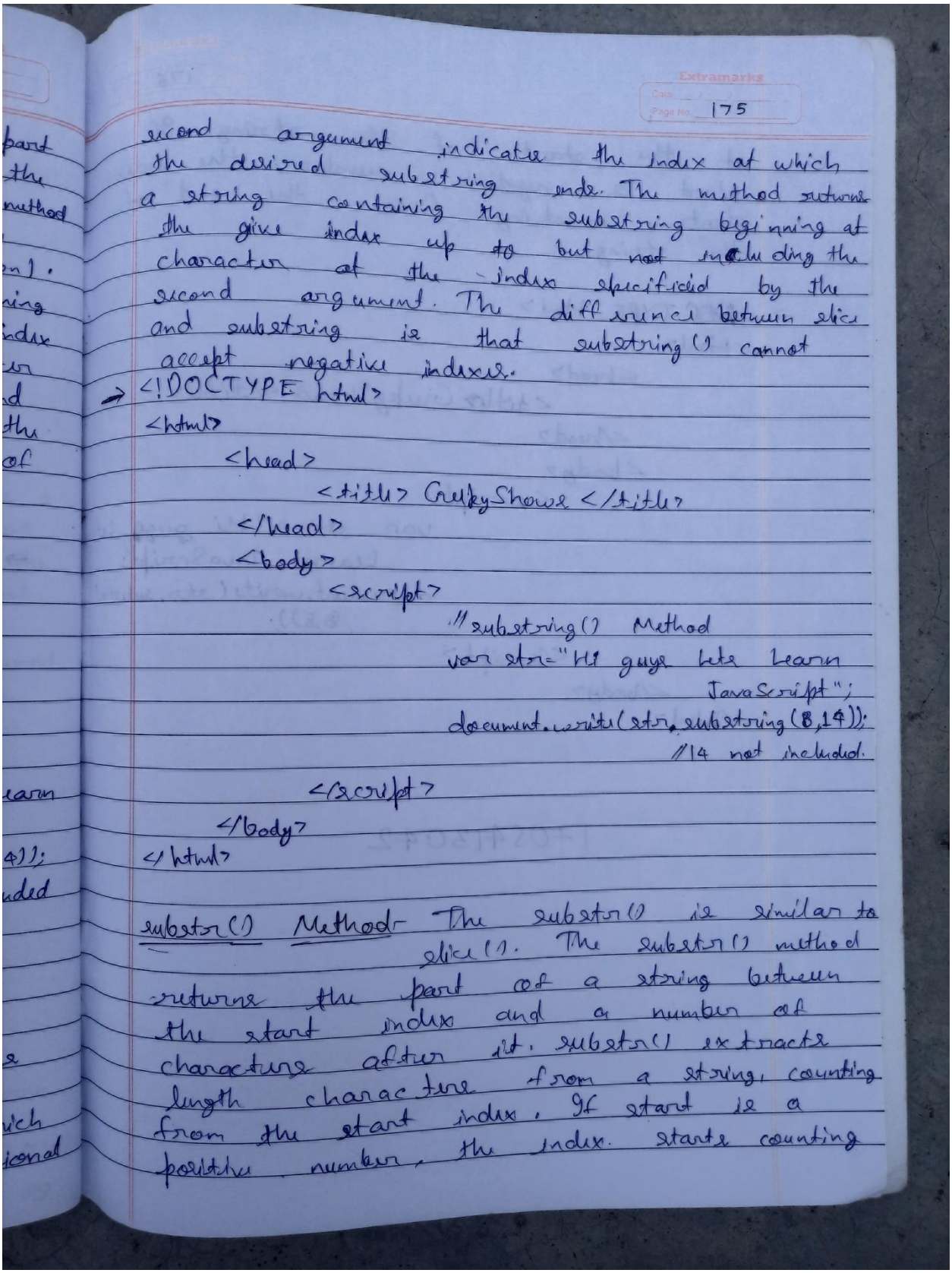
```
<!DOCTYP html>
<html>
    <head>
        <title>CodeyShows</title>
    </head>
    <body>
        <script>
            // search() Method
            var str="Hi guys Lets Learn JavaScript";
            document.write(str.search("s")); //3
            document.write("<br>");
            document.write(str.search("Lets")); //8

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

Extramarks
Date
Page No. 174

slice() Method- The slice() extracts a part of a string and returns the extracted part in a new string. The method takes 2 parameters:- The starting index (position), and the ending index (position). The method returns a string containing the string beginning at the given index up to but not including the character at the index specified by the second argument. If a parameter is negative, the position is counted from the end of the string.

```
<!DOCTYPE html>
<html>
    <head>
        <title> GeekyShows </title>
    </head>
    <body>
        <script>
            // slice() Method
            var str= "Hi guys Lets Learn
                      JavaScript";
            document.write(str.slice(8,14));
                                    //14 not included
        </script>
    </body>
</ html>
```

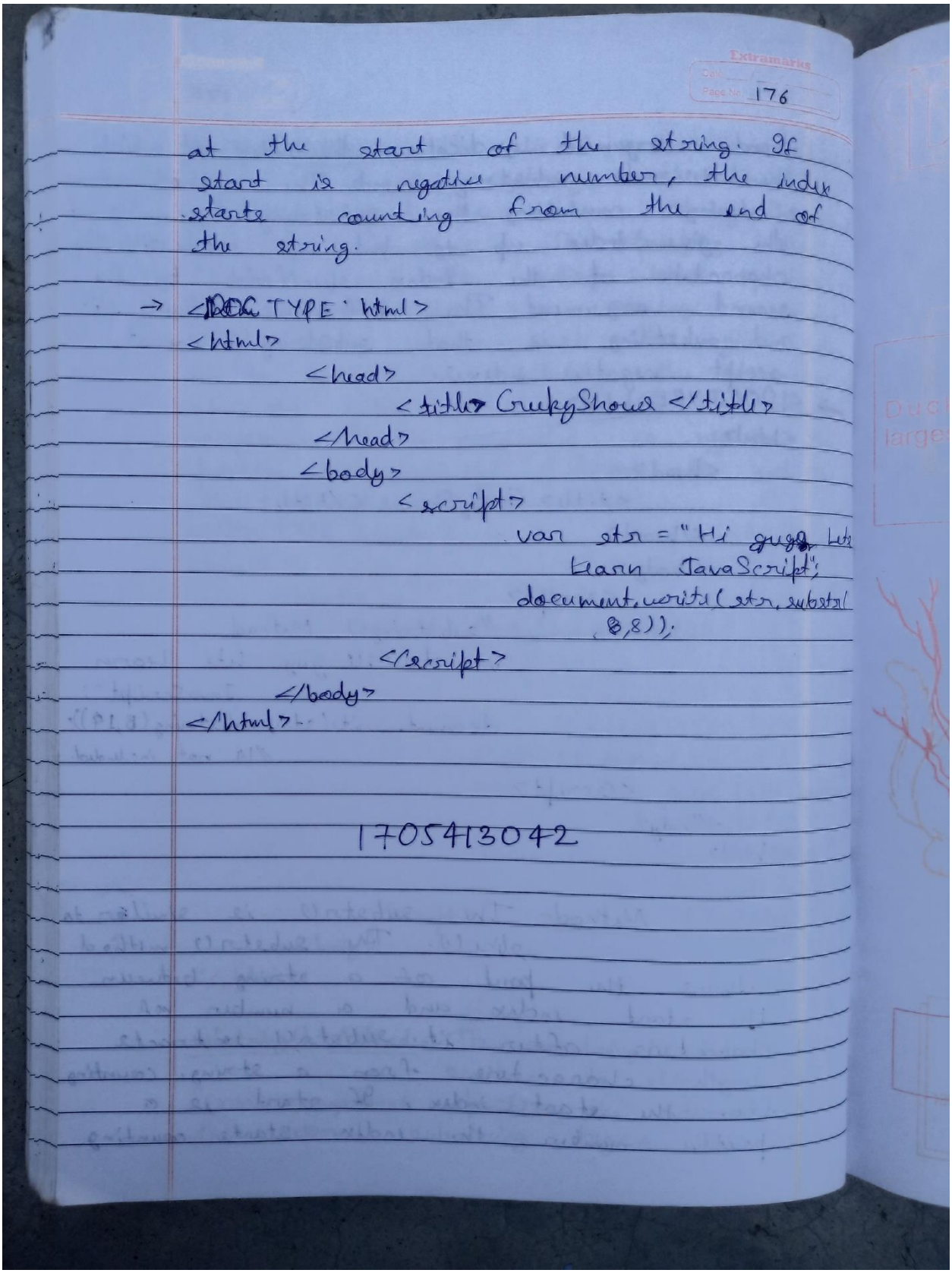
substring() Method - The substring() Method is similar to slice(). The first argument specifies the index at which the desired substring begins. The optional



second argument indicates the index at which the desired substring ends. The method returns a string containing the substring beginning at the give index up to but not including the character at the index specified by the second argument. The difference between slice and substring is that substring() cannot accept negative indexes.

```
<!DOCTYPE html>
<html>
    <head>
        <title> GeekyShows </title>
    </head>
    <body>
        <script>
            // substring() Method
            var str="Hi guys lets Learn
                    JavaScript";
            document.write(str.substring(8,14));
                    //14 not included.
        </script>
    </body>
</html>
```

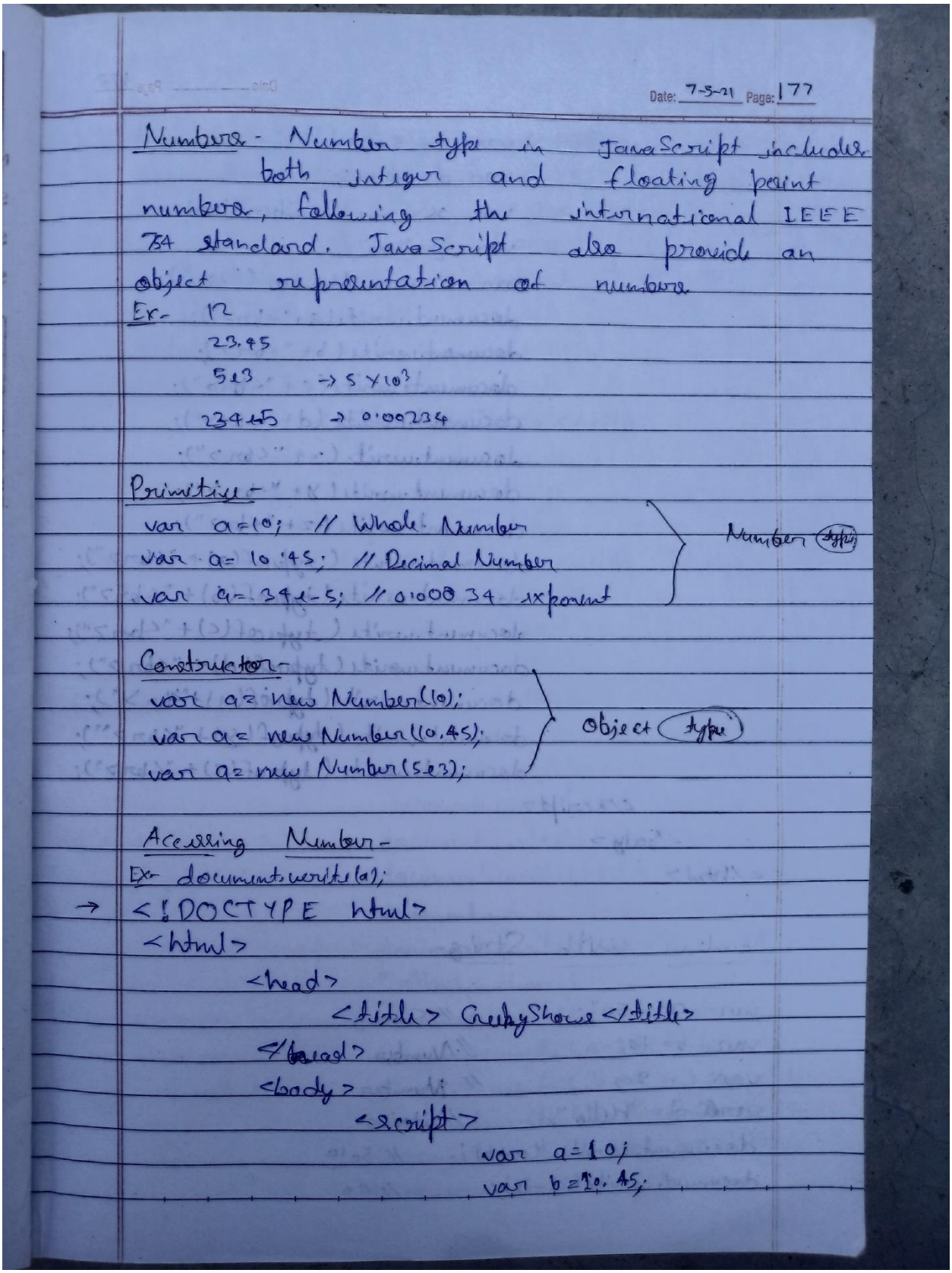
substr() Method- The substr() is similar to slice(). The substr() method returns the part of a string between the start index and a number of characters after it. substr() extracts length characters from a string, counting from the start index. If start is a positive number, the index starts counting



at the start of the string. If start is negative number, the index starts counting from the end of the string.

→ <!DOCTYPE html>
<html>
<head>
<title> GeekyShows </title>
</head>
<body>
<script>
var str = "Hi guys Lets Learn JavaScript";
document.write(str.substr(8,8));
</script>
</body>
</html>

1705413042



Date: 7-5-21 Page: 177

Numbers - Number type in JavaScript includes both integer and floating point numbers, following the international IEEE 754 standard. JavaScript also provide an object representation of numbers.

Ex- 12
23.45
5e3 → 5 X 10^3
23445 → 0.00234

Primitive -

```
var a=10; // Whole Number
var a= 10.45; // Decimal Number
var a= 34e-5; // 0.00034 exponent
```

} Number type

Constructor -

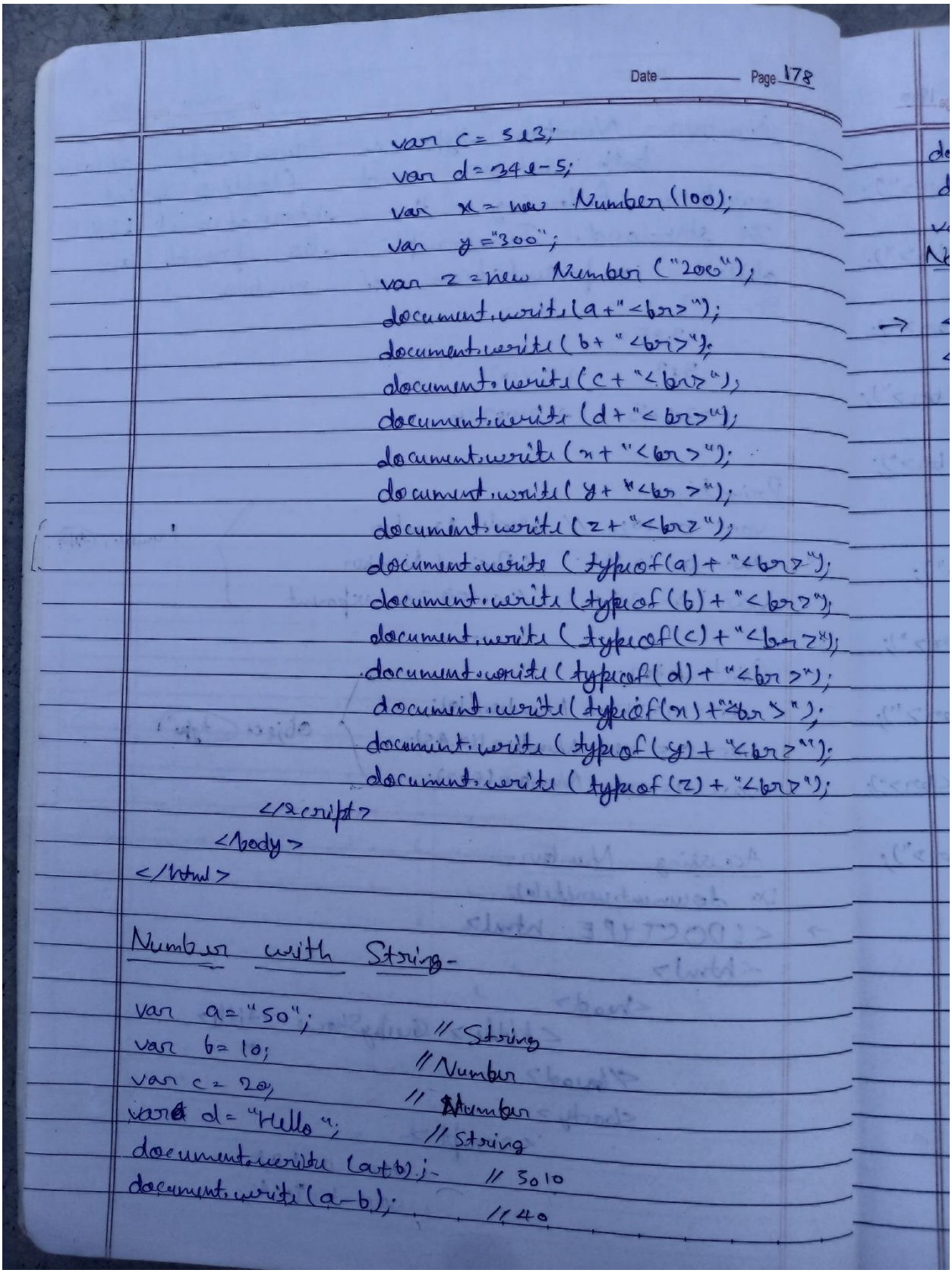
```
var a= new Number(10);
var a= new Number(10.45);
var a= new Number(5e3);
```

} Object type

Accessing Number -

Ex- document.write(a);

→
```
<!DOCTYPE html>
<html>
    <head>
        <title> GeekyShows </title>
    </head>
    <body>
        <script>
            var a=10;
            var b=10.45;
```

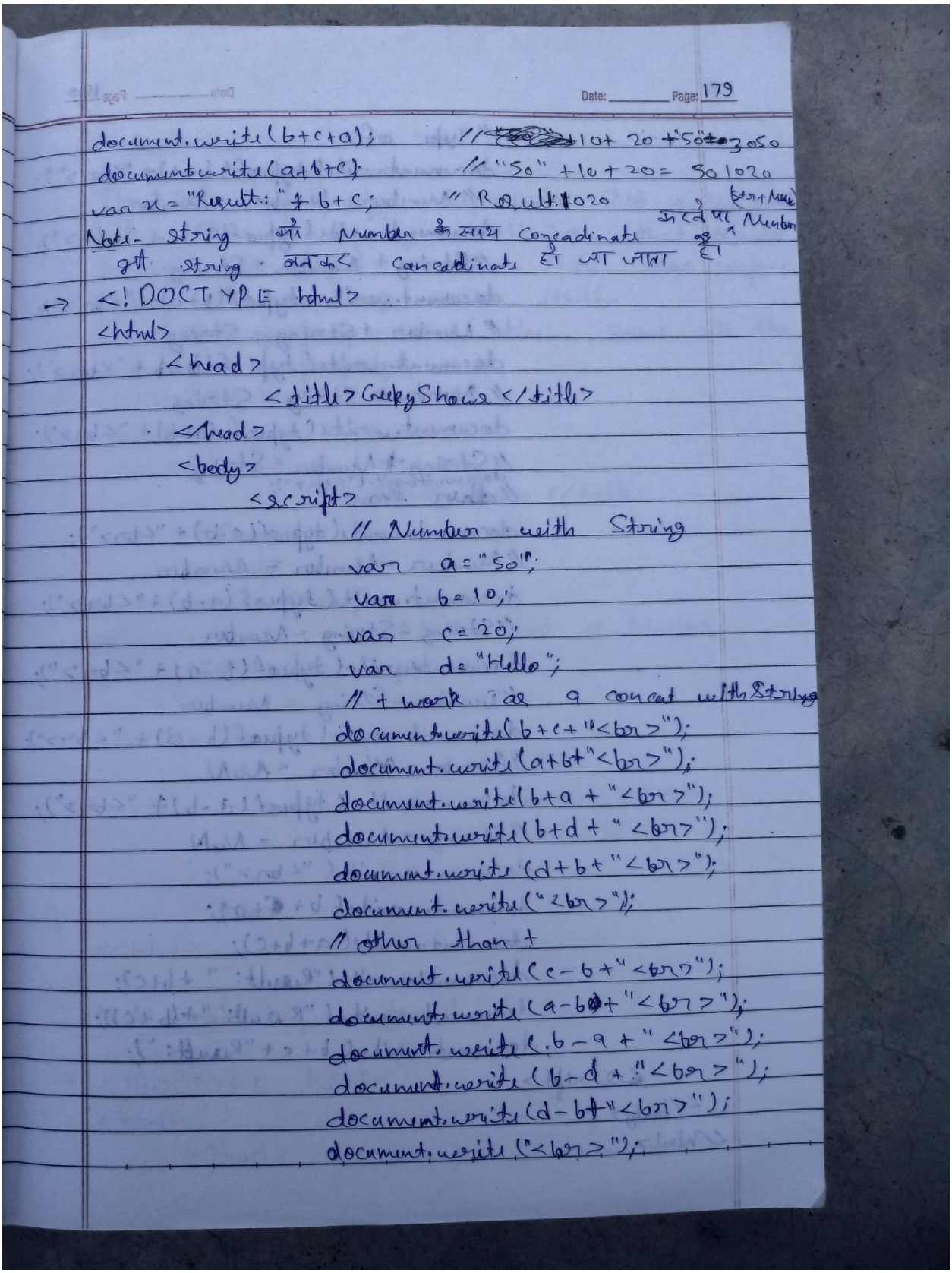


Date _____ Page 178

```
var c = 513;
var d = 34e-5;
var x = new Number(100);
var y = "300";
var z = new Number("200");
document.write(a+"<br>");
document.write(b+"<br>");
document.write(c+"<br>");
document.write(d+"<br>");
document.write(x+"<br>");
document.write(y+"<br>");
document.write(z+"<br>");
document.write(typeof(a)+"<br>");
document.write(typeof(b)+"<br>");
document.write(typeof(c)+"<br>");
document.write(typeof(d)+"<br>");
document.write(typeof(x)+"<br>");
document.write(typeof(y)+"<br>");
document.write(typeof(z)+"<br>");
</script>
</body>
</html>
```

Number with String-

```
var a="50";          // String
var b=10;            // Number
var c=20;            // Number
var d="Hello";       // String
document.write(a+b);- // 5010
document.write(a-b);  // 40
```



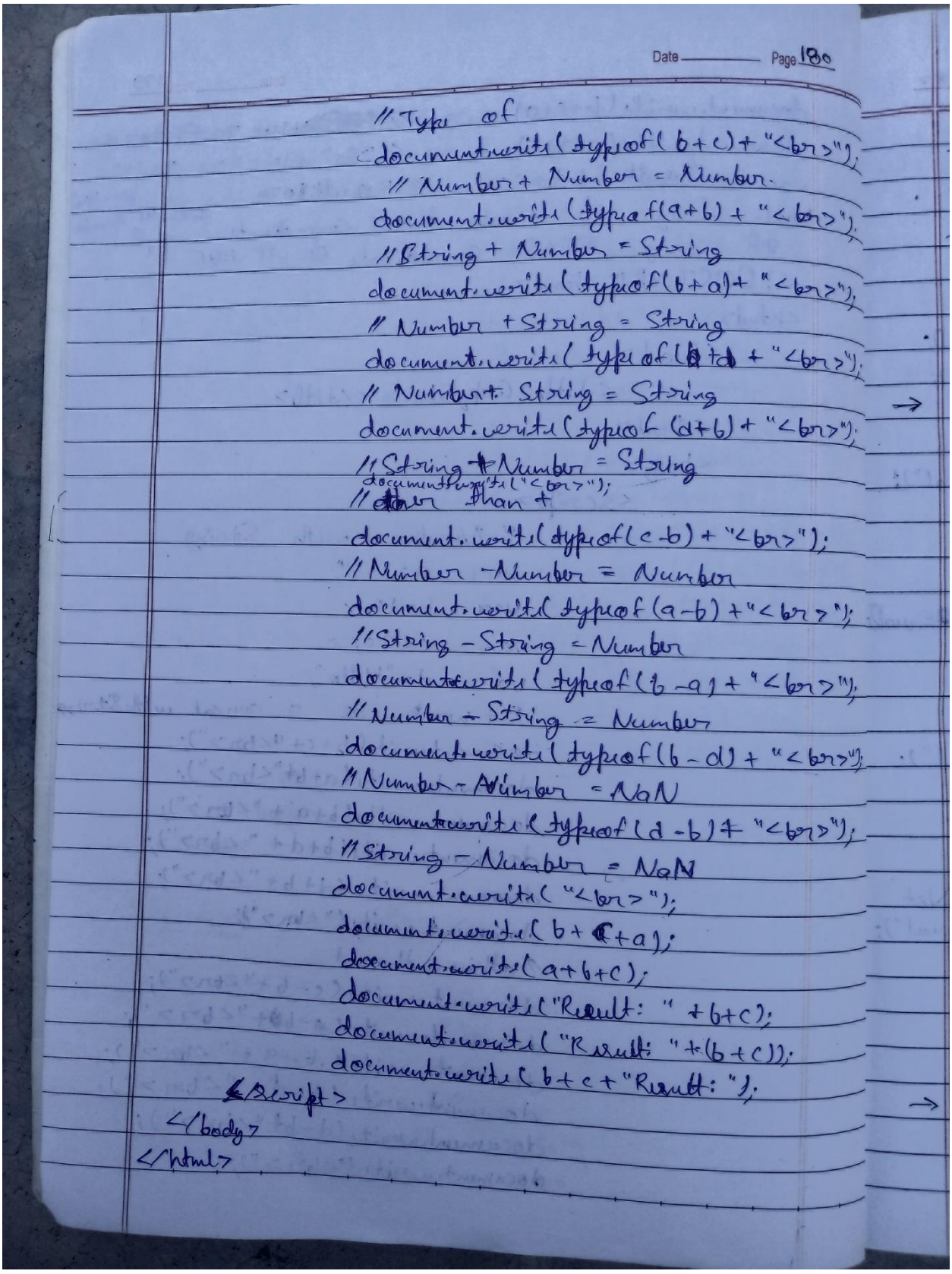
Date: Page: 179

```
document.write(b+c+a);          // 10+20+"50" = 3050
document.write(a+b+c);          // "50"+10+20 = 501020
var x = "Result: "+b+c;         // Result: 1020
```

Note- String और Number के साथ Concadinate करने पर Number भी String बन कर Concadinate हो जाता है

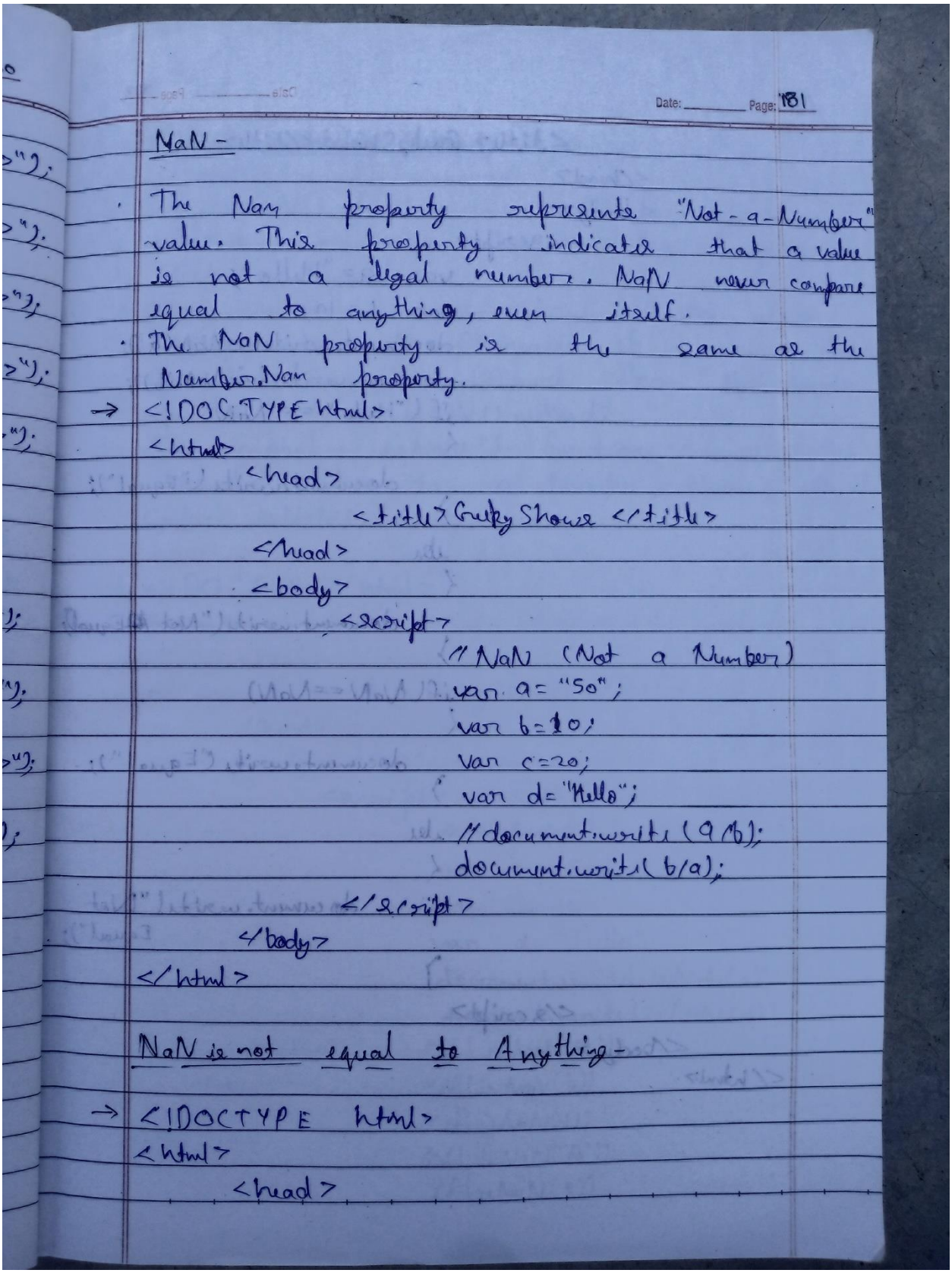
→

```
<!DOCTYPE html>
<html>
    <head>
        <title>GeekyShows</title>
    </head>
    <body>
        <script>
            // Number with String
            var a="50";
            var b=10;
            var c=20;
            var d="Hello";
            // + work as a concat with String
            document.write(b+c+"<br>");
            document.write(a+b+"<br>");
            document.write(b+a+"<br>");
            document.write(b+d+"<br>");
            document.write(d+b+"<br>");
            document.write("<br>");
            // other than +
            document.write(c-b+"<br>");
            document.write(a-b+"<br>");
            document.write(b-a+"<br>");
            document.write(b-d+"<br>");
            document.write(d-b+"<br>");
            document.write("<br>");
```



Date _____ Page 180

```
// Type of
document.write(typeof(b+c)+"<br>");
// Number + Number = Number.
document.write(typeof(a+b)+"<br>");
// String + Number = String
document.write(typeof(b+a)+"<br>");
// Number + String = String
document.write(typeof(a+d)+"<br>");
// Number + String = String
document.write(typeof(d+b)+"<br>");
// String + Number = String
document.write("<br>");
// other than +
document.write(typeof(c-b)+"<br>");
// Number - Number = Number
document.write(typeof(a-b)+"<br>");
// String - String = Number
document.write(typeof(b-a)+"<br>");
// Number - String = Number
document.write(typeof(b-d)+"<br>");
// Number - Number = NaN
document.write(typeof(d-b)+"<br>");
// String - Number = NaN
document.write("<br>");
document.write(b+c+a);
document.write(a+b+c);
document.write("Result: "+b+c);
document.write("Result: "+(b+c));
document.write(b+c+"Result: ");
</Script>
</body>
</html>
```



Date: Page: 181

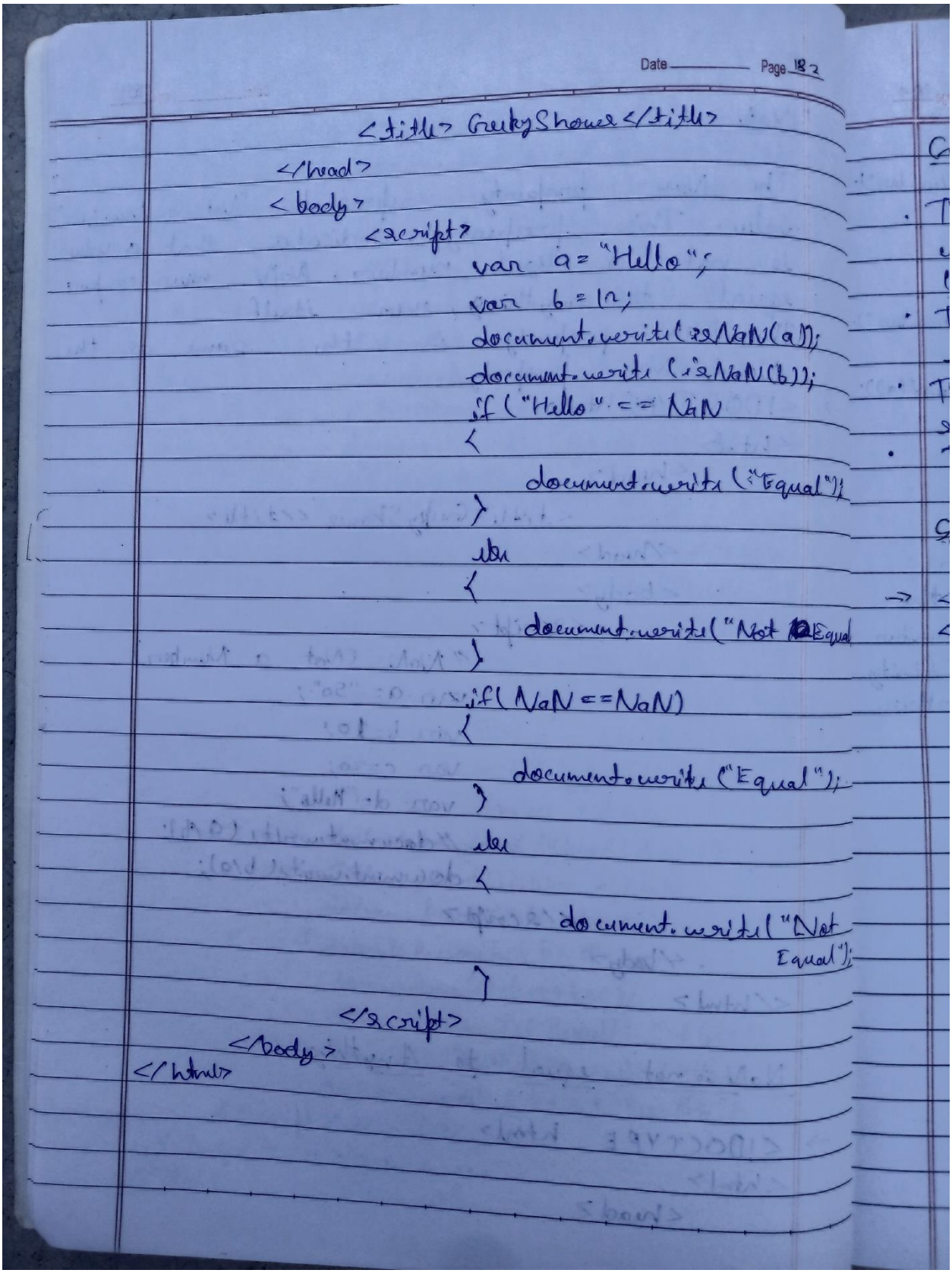
NaN -

- The Nan property represents "Not-a-Number" value. This property indicates that a value is not a legal number. NaN never compare equal to anything, even itself.
- The NaN property is the same as the Number.Nan property.

```
<!DOCTYPE html>
<html>
    <head>
        <title> Geeky Shows </title>
    </head>
    <body>
        <script>
            // NaN (Not a Number)
            var a = "50";
            var b = 10;
            var c = 20;
            var d = "Hello";
            // document.write(a*b);
            document.write(b/a);
        </script>
    </body>
</html>
```

NaN is not equal to Anything -

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



```
<title> GeekyShows </title>
</head>
<body>
<script>
var a = "Hello";
var b = 12;
document.write(isNaN(a));
document.write(isNaN(b));
if("Hello" == NaN
{
document.write("Equal");
}
else
{
document.write("Not Equal
}
if(NaN == NaN)
{
document.write("Equal");
}
else
{
document.write("Not
Equal");
}
</script>
</body>
</html>
```

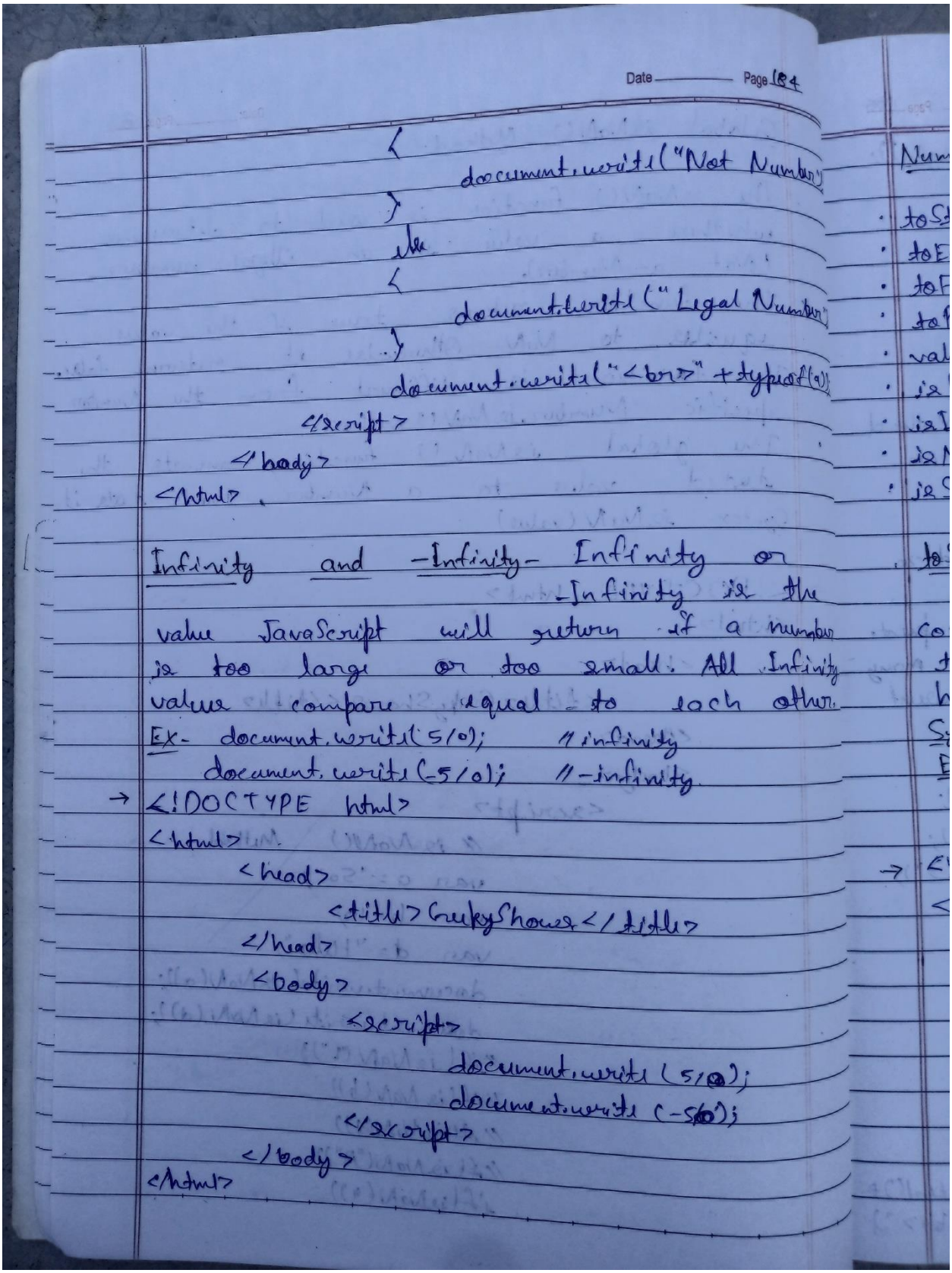
Date: Page: 183

Global isNaN() Method -

- The isNaN() function is used to determine whether a value is an illegal number (Not - a - Number).
- The function returns true if the value equates to NaN. Otherwise it returns false.
- The function is different from the Number specific Number.isNaN() method.
- The global isNaN() function, converts the tested value to a Number, then tests it.

Syntax - isNaN(value)

```
<!DOCTYPE html>
<html>
    <head>
        <title> Geeky Shows </title>
    </head>
    <body>
        <script>
            // isNaN() Method
            var a = "50";
            var b = 10;
            var d = "Hello";
            document.write( isNaN(a));
            document.write (isNaN (d));
            // if (isNaN ("1"))
            // if (isNaN(b))
            // if (isNaN(d))
            // if (isNaN("A"))
            if (isNaN(a))
```



```
        {
            document.write("Not Number");
        }
        else
        {
            document.write("Legal Number");
        }
        document.write("<br>" + typeof(a));
    </script>
</body>
</html>
```

Infinity and -Infinity- Infinity or -Infinity is the value JavaScript will return if a number is too large or too small. All Infinity value compare equal to each other.

Ex- document.write(5/0); // infinity
document.write(-5/0); // -infinity.

```
<!DOCTYPE html>
<html>
    <head>
        <title>GeekyShows</title>
    </head>
    <body>
        <script>
            document.write(5/0);
            document.write(-5/0);
        </script>
    </body>
</html>
```

Date: Page: 185

Number Method -

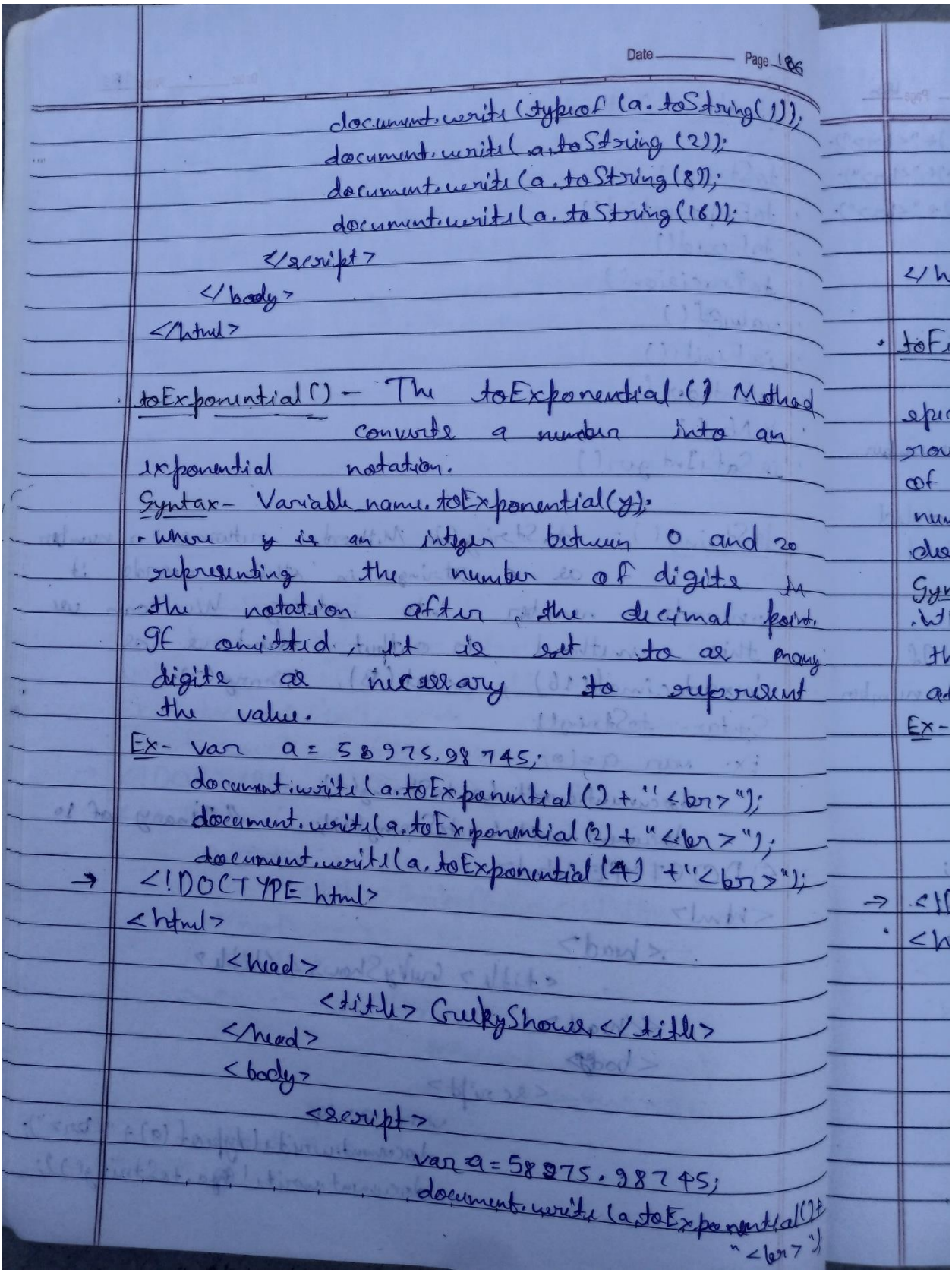
- toString()
- toExponential()
- toFixed()
- toPrecision()
- valueOf()
- isFinite()
- isInteger()
- isNan()
- isSafeInteger()

toString() - toString() Method returns a number as a string in other words it converts number into string. We can use this method to output numbers as hexadecimal(16), octal(8), binary(2);

Syntax - toString();

Ex- var a=10;
document.write(a.toString());
document.write(a.toString(2)); // Binary of 10

```
<!DOCTYPE html>
<html>
    <head>
        <title> GeekyShows </title>
    </head>
    <body>
        <script>
            var a=10;
            document.write(typeof(a)+"<br>");
            document.write(a.toString());
```



```
document.write(typeof(a.toString()));
document.write(a.toString(2));
document.write(a.toString(8));
document.write(a.toString(16));
</script>
</body>
</html>
```

toExponential() – The toExponential() Method converts a number into an exponential notation.

Syntax- Variable_name.toExponential(y);

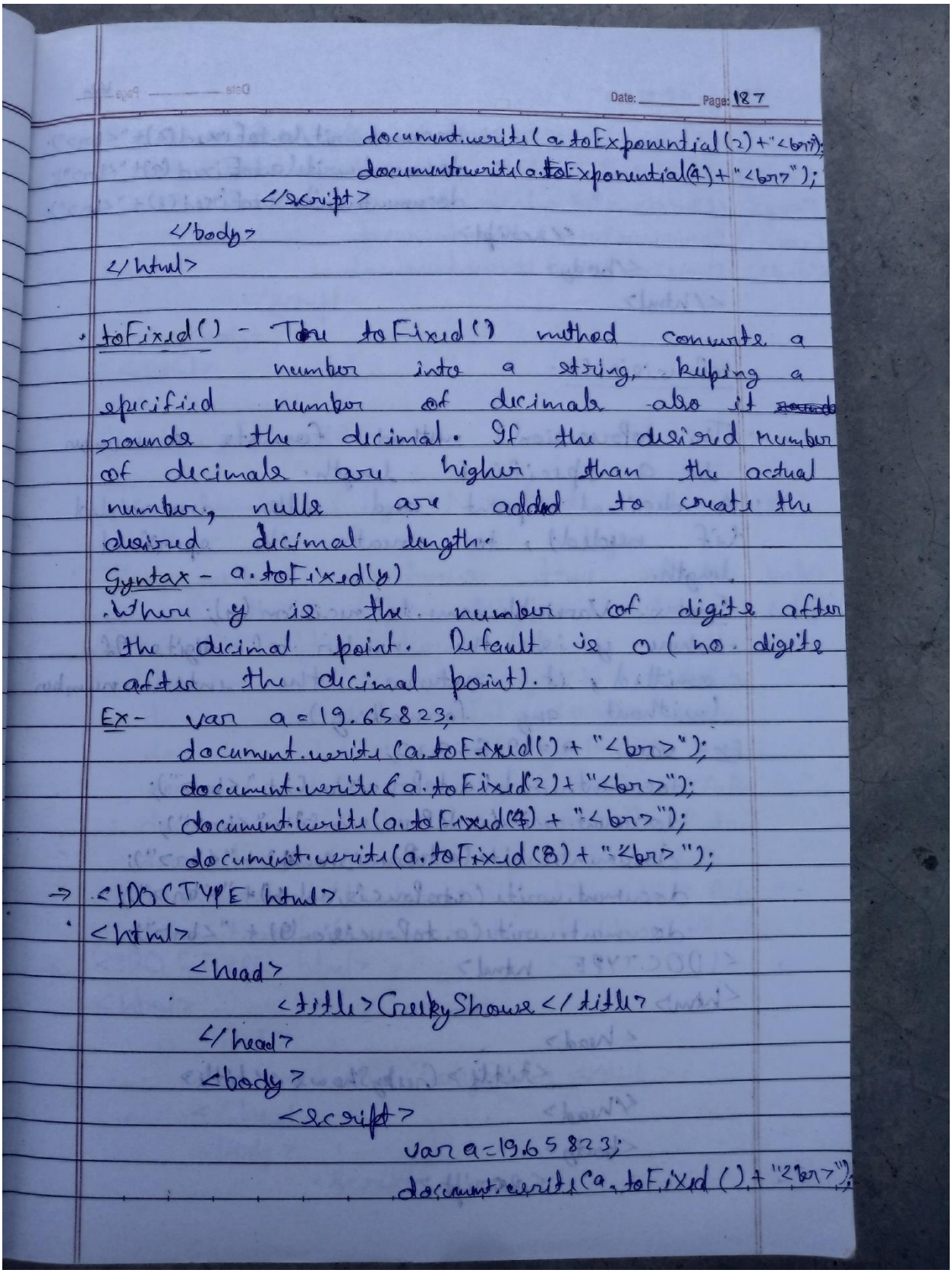
- Where y is an integer between 0 and 20 representing the number of digits in the notation after the decimal point. If omitted, it is set to as many digits as necessary to represent the value.

Ex- var a = 58975.98745;

```
document.write(a.toExponential() + "<br>");
document.write(a.toExponential(2) + "<br>");
document.write(a.toExponential(4) + "<br>");
```

→

```
<!DOCTYPE html>
<html>
<head>
<title> GeekyShows </title>
</head>
<body>
<script>
var a = 58975.98745;
document.write(a.toExponential() +
"<br>");
```



```
        document.write(a.toExponential(2)+"<br>");
        document.write(a.toExponential(4)+"<br>");
    </script>
  </body>
</html>
```

toFixed() - The toFixed() method converts a number into a string, keeping a specified number of decimals. Also it rounds the decimal. If the desired number of decimals are higher than the actual number, nulls are added to create the desired decimal length.

Syntax - a.toFixed(y)

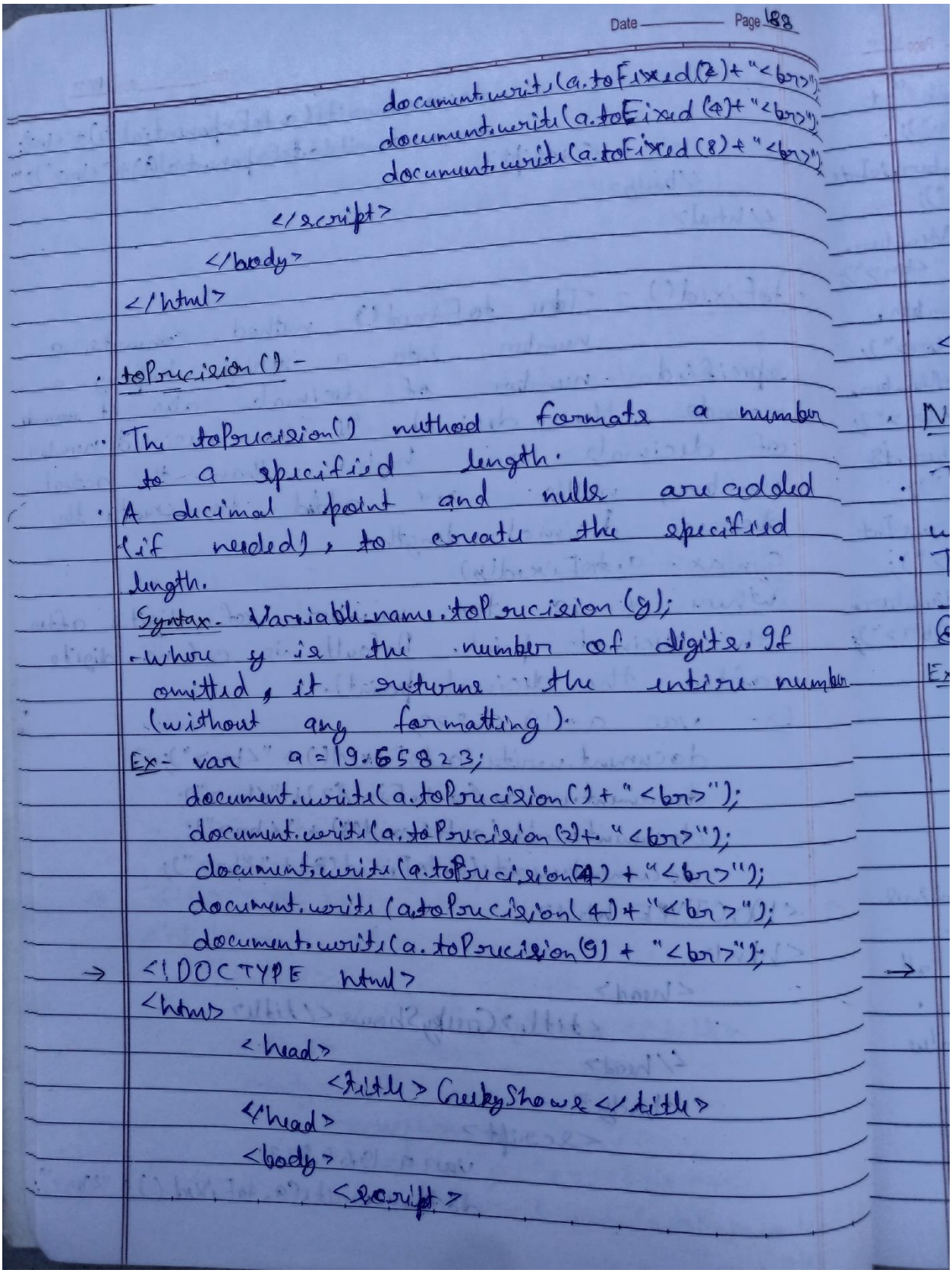
Where y is the number of digits after the decimal point. Default is 0 (no digits after the decimal point).

Ex-

```
var a = 19.65823;
document.write(a.toFixed() + "<br>");
document.write(a.toFixed(2) + "<br>");
document.write(a.toFixed(4) + "<br>");
document.write(a.toFixed(8) + "<br>");
```

→

```
<!DOCTYPE html>
<html>
  <head>
    <title>GeekyShows</title>
  </head>
  <body>
    <script>
      var a=19.65823;
      document.write(a.toFixed() + "<br>");
```



```
document.write(a.toFixed(2)+"<br>");
document.write(a.toFixed(4)+"<br>");
document.write(a.toFixed(8)+"<br>");
</script>
</body>
</html>
```

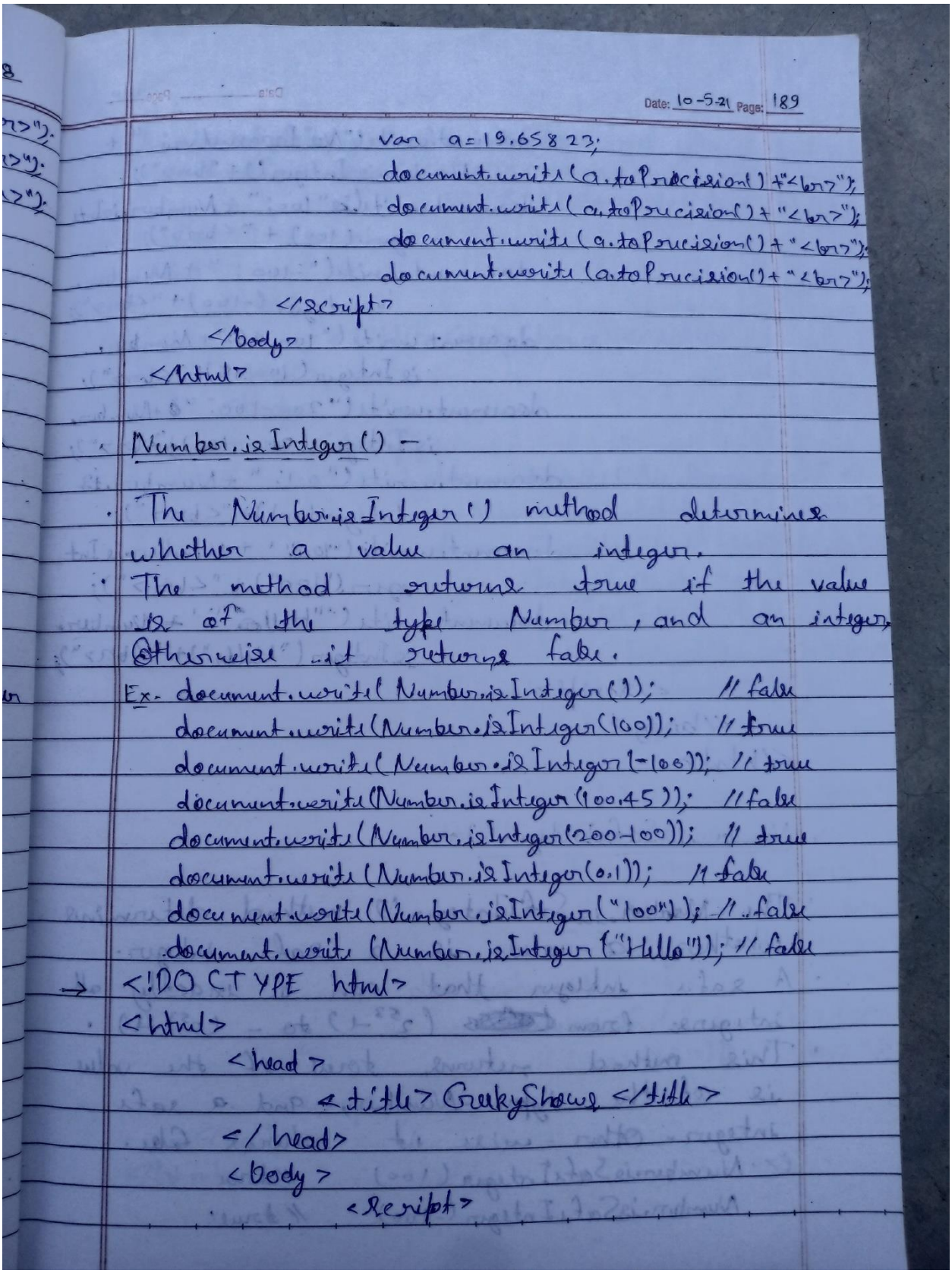
toPrecision() -

The toPrecision() method formats a number to a specified length.
A decimal point and nulls are added (if needed), to create the specified length.

Syntax - Variable_name.toPrecision(g);
where g is the number of digits. If omitted, it returns the entire number (without any formatting).

Ex -
```
var a=19.65823;
document.write(a.toPrecision()+"<br>");
document.write(a.toPrecision(2)+"<br>");
document.write(a.toPrecision(4)+"<br>");
document.write(a.toPrecision(4)+"<br>");
document.write(a.toPrecision(9)+"<br>");
```

```
<!DOCTYPE html>
<html>
    <head>
        <title> GeekyShows </title>
    </head>
    <body>
        <script>
```



Date: 10-5-21 Page: 189

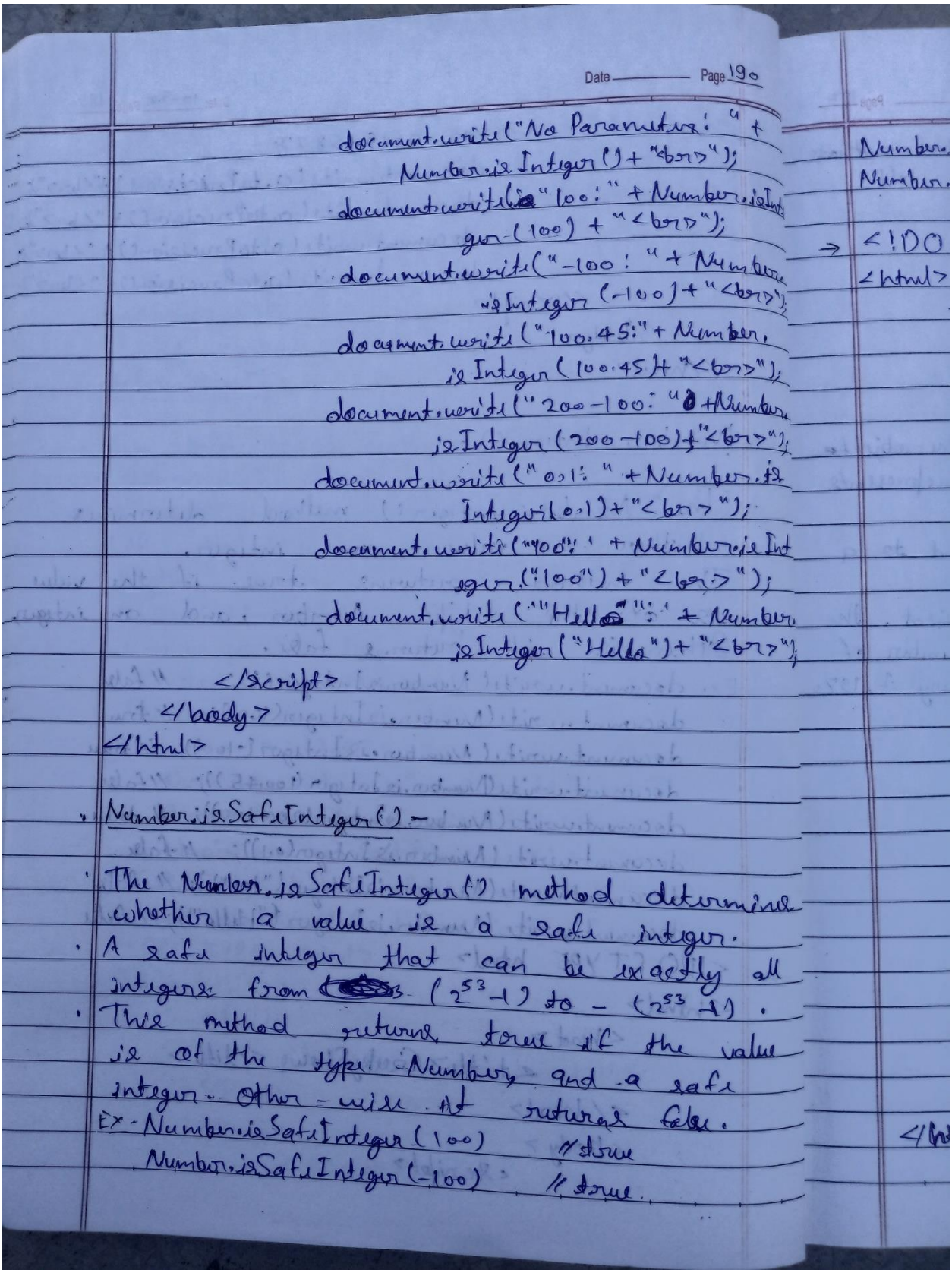
```
var a=19.658723;
document.write(a.toPrecision()+"<br>");
document.write(a.toPrecision()+"<br>");
document.write(a.toPrecision()+"<br>");
document.write(a.toPrecision()+"<br>");
</script>
</body>
</html>
```

Number.isInteger() –

The Number.isInteger() method determines whether a value an integer.

The method returns true if the value is of the type Number, and an integer. Otherwise it returns false.

```
Ex- document.write(Number.isInteger());          // false
    document.write(Number.isInteger(100));       // true
    document.write(Number.isInteger(-100));      // true
    document.write(Number.isInteger(100.45));    // false
    document.write(Number.isInteger(200+100));   // true
    document.write(Number.isInteger(0.1));       // false
    document.write(Number.isInteger("100"));     // false
    document.write(Number.isInteger("Hello"));   // false
→ <!DOCTYPE html>
<html>
    <head>
        <title>GeekyShows</title>
    </head>
    <body>
        <script>
```

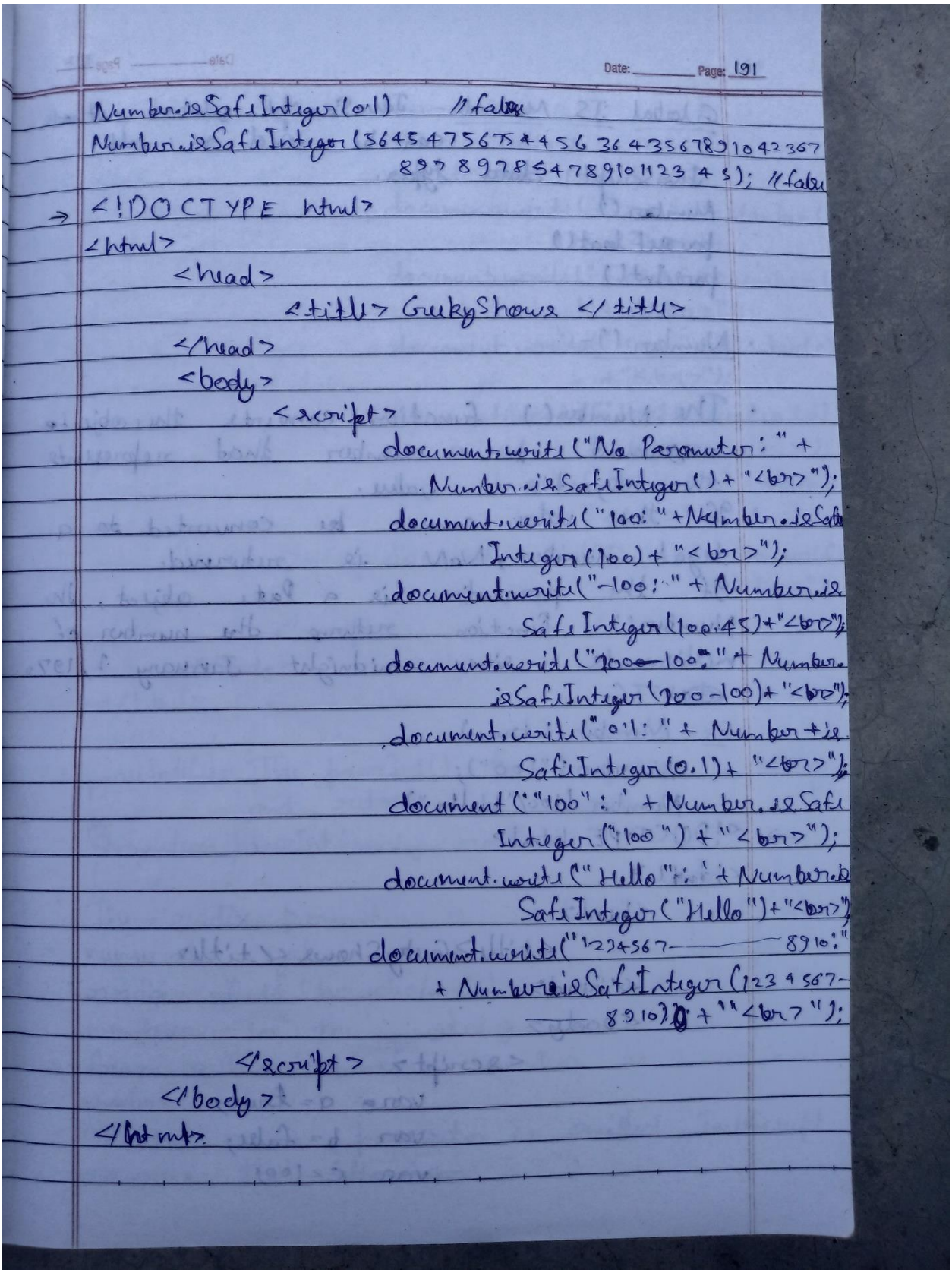


```
Date _____ Page 190

document.write("No Parameter: " +
    Number.isInteger() + "<br>");
document.write("100: " + Number.isInteger(100) + "<br>");
document.write("-100: " + Number.isInteger(-100) + "<br>");
document.write("100.45:" + Number.isInteger(100.45) + "<br>");
document.write("200-100: " + Number.isInteger(200-100) + "<br>");
document.write("0.1: " + Number.isInteger(0.1) + "<br>");
document.write("'100': " + Number.isInteger("100") + "<br>");
document.write("'"Hello"': " + Number.isInteger("Hello") + "<br>");
</script>
</body>
</html>
```

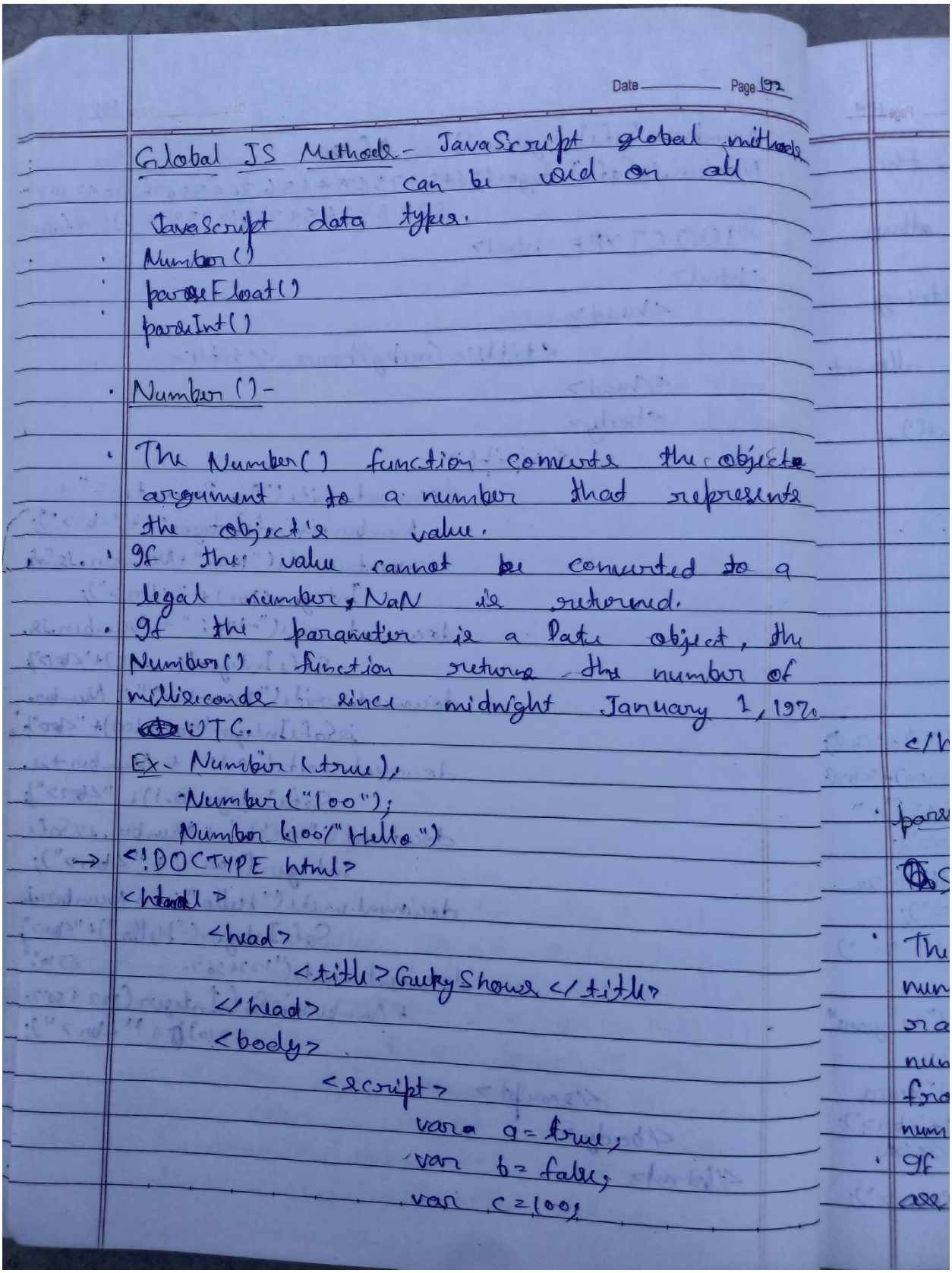
- **Number.isSafeInteger() –**

- The Number.isSafeInteger() method determines whether a value is a safe integer.
- A safe integer that can be exactly all integers from $-(2^{53}-1)$ to $(2^{53}-1)$.
- This method returns true if the value is of the type Number, and a safe integer. Other-wise it returns false.

```
Ex- Number.isSafeInteger(100)      // true
    Number.isSafeInteger(-100)     // true
```



```
Number.isSafeInteger(0.1)    // false
Number.isSafeInteger(3645475675445636435678910423678978978547891011234 5);  // false

<!DOCTYPE html>
<html>
    <head>
        <title> GeekyShows </title>
    </head>
    <body>
        <script>
            document.write("No Parameter: " +
                Number.isSafeInteger() + "<br>");
            document.write("100: " + Number.isSafe
                Integer(100) + "<br>");
            document.write("-100: " + Number.is
                SafeInteger(100.45) + "<br>");
            document.write("200-100: " + Number.
                isSafeInteger(200-100) + "<br>");
            document.write("0.1: " + Number.is
                SafeInteger(0.1) + "<br>");
            document("'100': ' + Number.isSafe
                Integer("100") + "<br>");
            document.write("Hello": ' + Number.is
                SafeInteger("Hello") + "<br>");
            document.write("1234567-------8910: "
                + Number.isSafeInteger(1234567-
                ------8910) + "<br>");
        </script>
    </body>
</html>
```



Date _____ Page 192

Global JS Methods - JavaScript global methods can be used on all JavaScript data types.

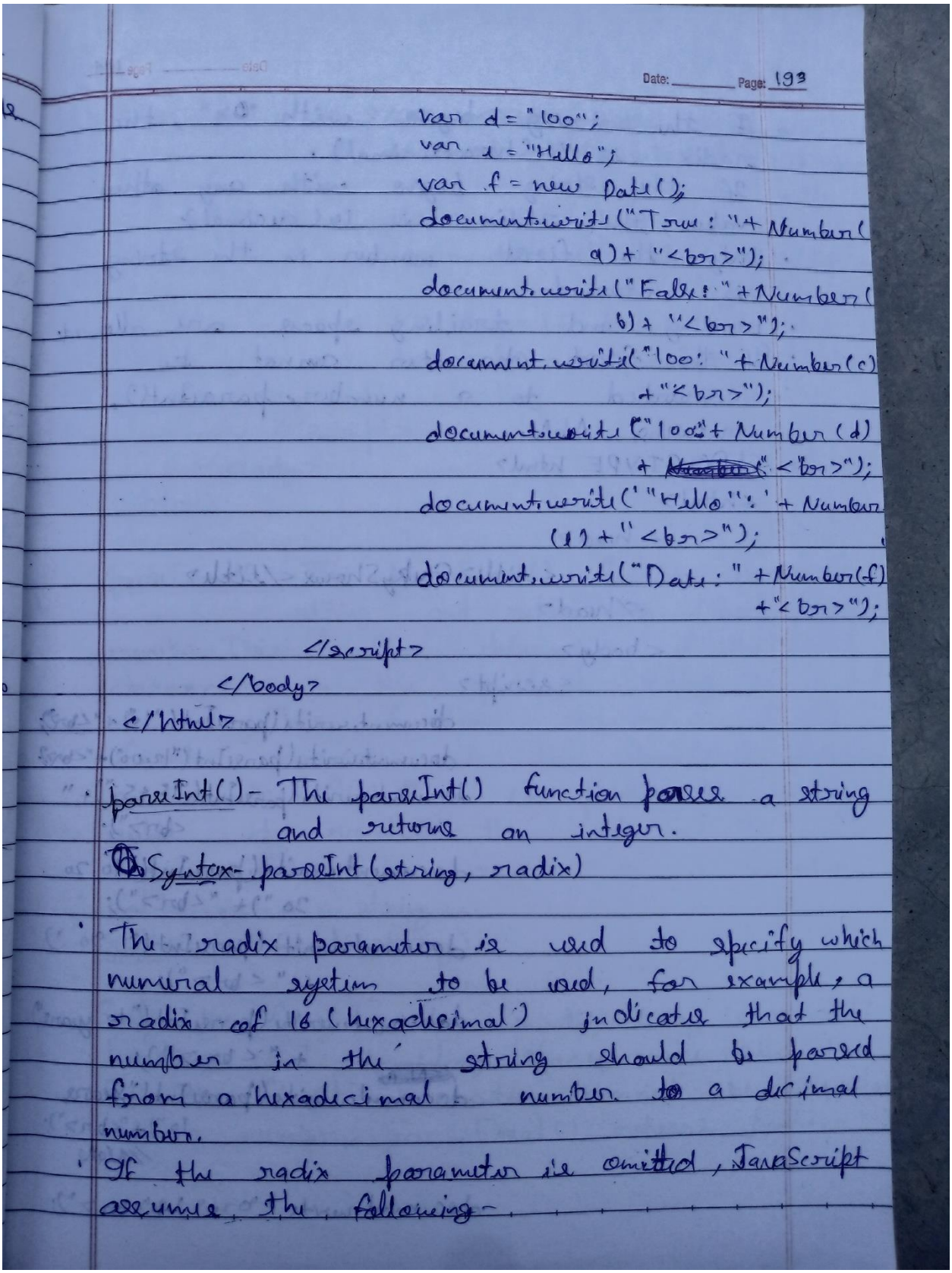
Number()
parseFloat()
parseInt()

Number() -

The Number() function converts the object's argument to a number that represents the object's value.
If the value cannot be converted to a legal number, NaN is returned.
If the parameter is a Date object, the Number() function returns the number of milliseconds since midnight January 1, 1970 UTC.

Ex - Number(true),
Number("100");
Number("100","Hello")

```
<!DOCTYPE html>
<html>
    <head>
        <title> Geeky Shows </title>
    </head>
    <body>
        <script>
            var a = true;
            var b = false;
            var c = 100;
```



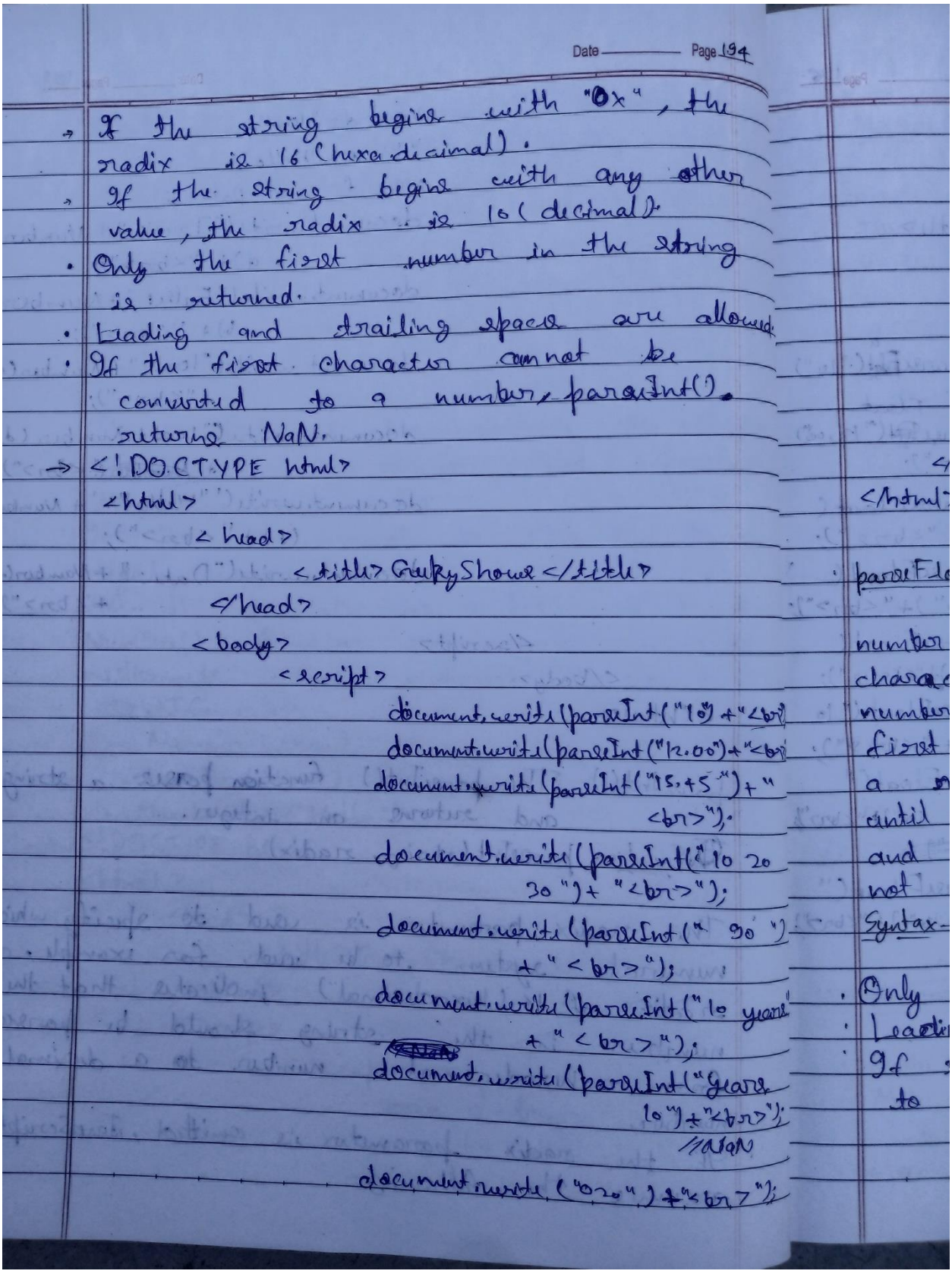
```
var d = "100";
var i = "Hello";
var f = new Date();
document.write("True : " + Number(
        a) + "<br>");
document.write("False : " + Number(
        b) + "<br>");
document.write("100 : " + Number(c)
        + "<br>");
document.write("100" + Number(d)
        + "<br>");
document.write("'Hello' : " + Number
        (i) + "<br>");
document.write("Date : " + Number(f)
        + "<br>");
```

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

- parseInt() - The parseInt() function parses a string and returns an integer.

Syntax - parseInt(string, radix)

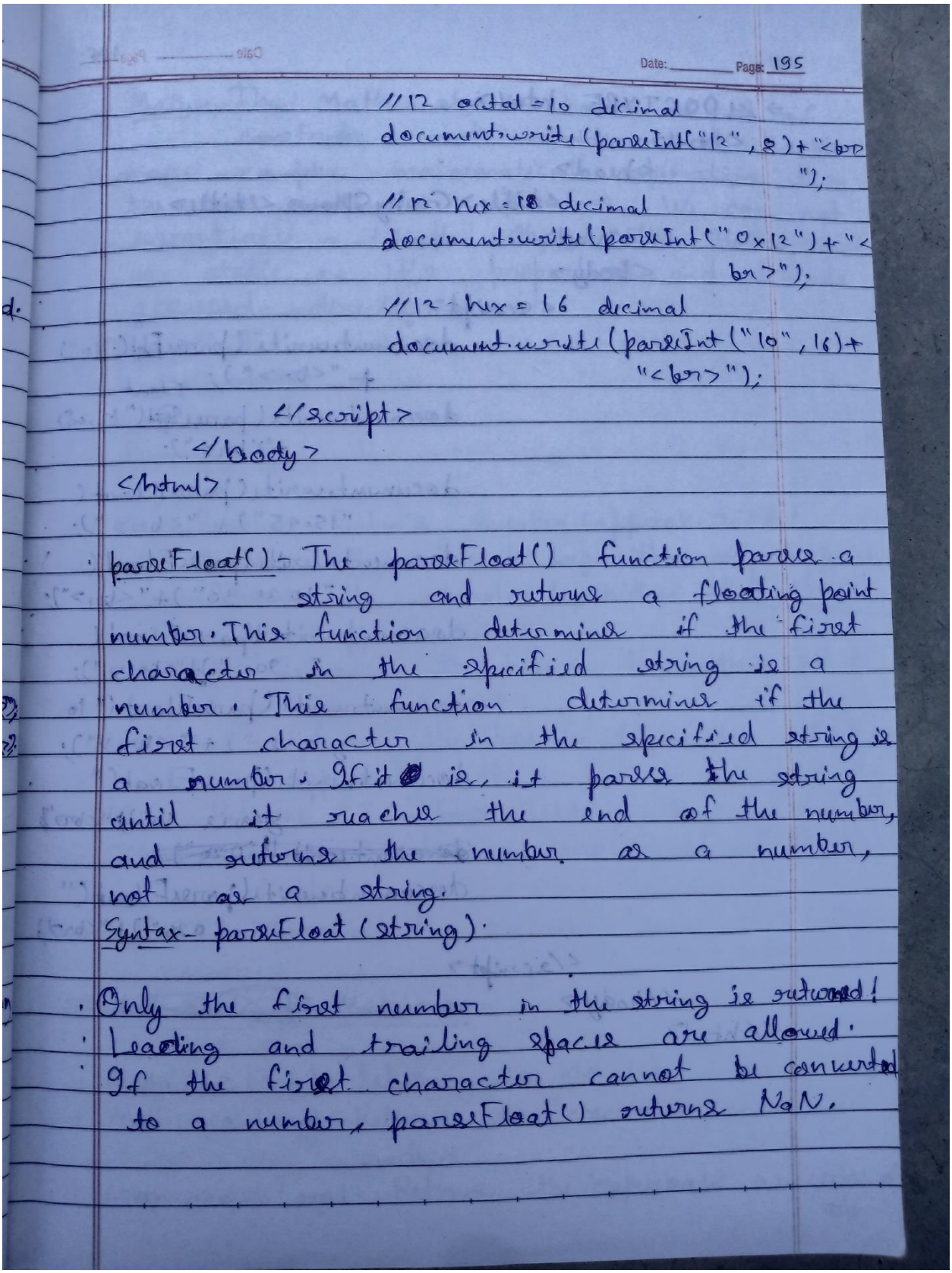
- The radix parameter is used to specify which numeral system to be used, for example, a radix of 16 (hexadecimal) indicates that the number in the string should be parsed from a hexadecimal number to a decimal number.
- If the radix parameter is omitted, JavaScript assumes the following -



Date _____ Page 194

→ If the string begins with "0x", the radix is 16 (hexadecimal).
→ If the string begins with any other value, the radix is 10 (decimal).
• Only the first number in the string is returned.
• Leading and trailing spaces are allowed.
• If the first character cannot be converted to a number, parseInt() returns NaN.

```
→ <!DOCTYPE html>
<html>
    <head>
        <title> GeekyShows </title>
    </head>
    <body>
        <script>
            document.write(parseInt("10") + "<br>");
            document.write(parseInt("12.00") + "<br>");
            document.write(parseInt("15.45") + "
                <br>");
            document.write(parseInt("10 20
                30") + "<br>");
            document.write(parseInt(" 90 ")
                + "<br>");
            document.write(parseInt("10 years"
                + "<br>");
            document.write(parseInt("years
                10") + "<br>");
            //NaN
            document.write("020" + "<br>");
```

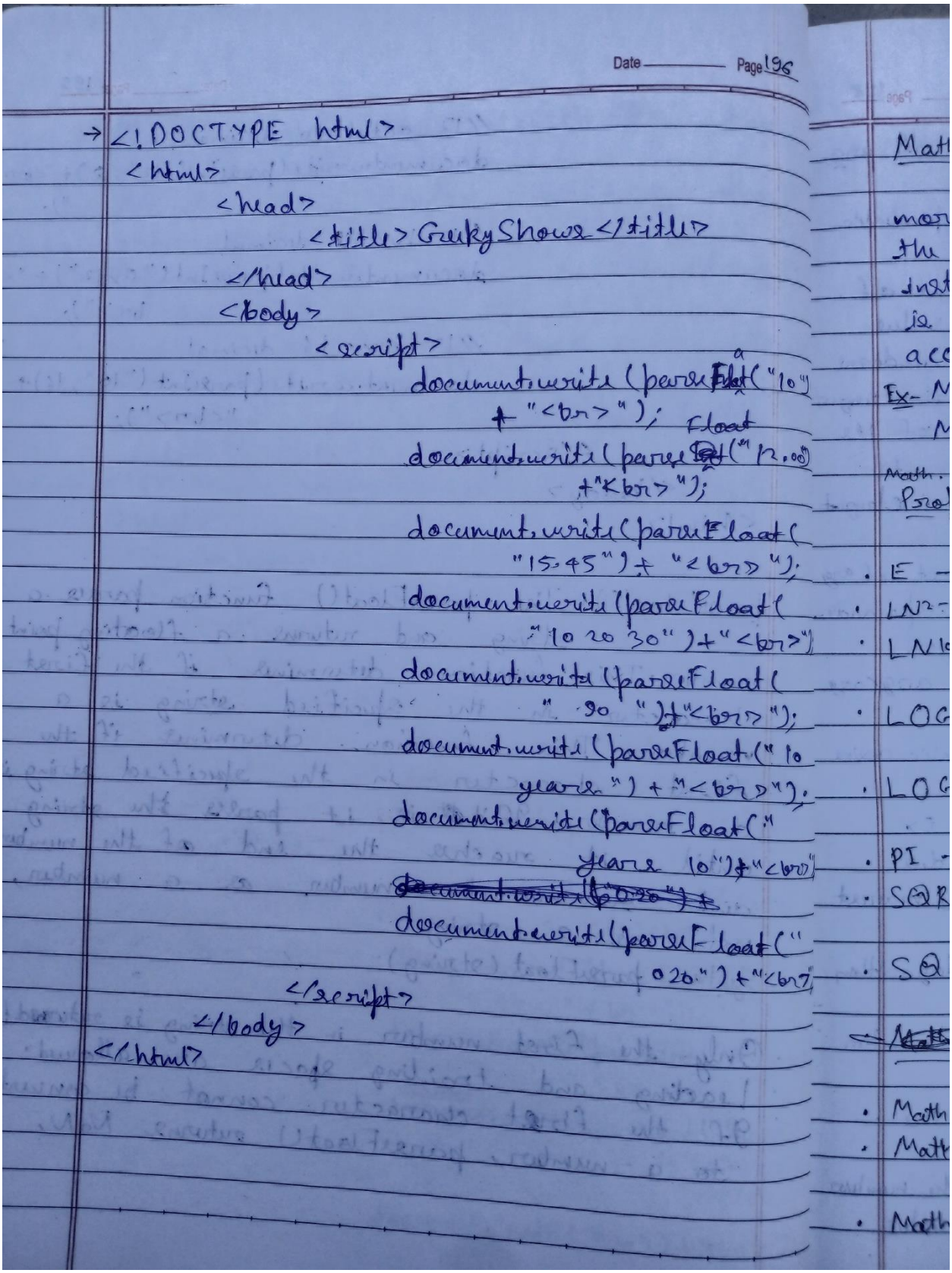


```
//12 octal = 10 decimal
document.write(parseInt("12", 8)+"<br>
");
//12 hex = 18 decimal
document.write(parseInt("0x12")+"<
br>");
//12 hex = 16 decimal
document.write(parseInt("10", 16)+
"<br>");
</script>
</body>
</html>
```

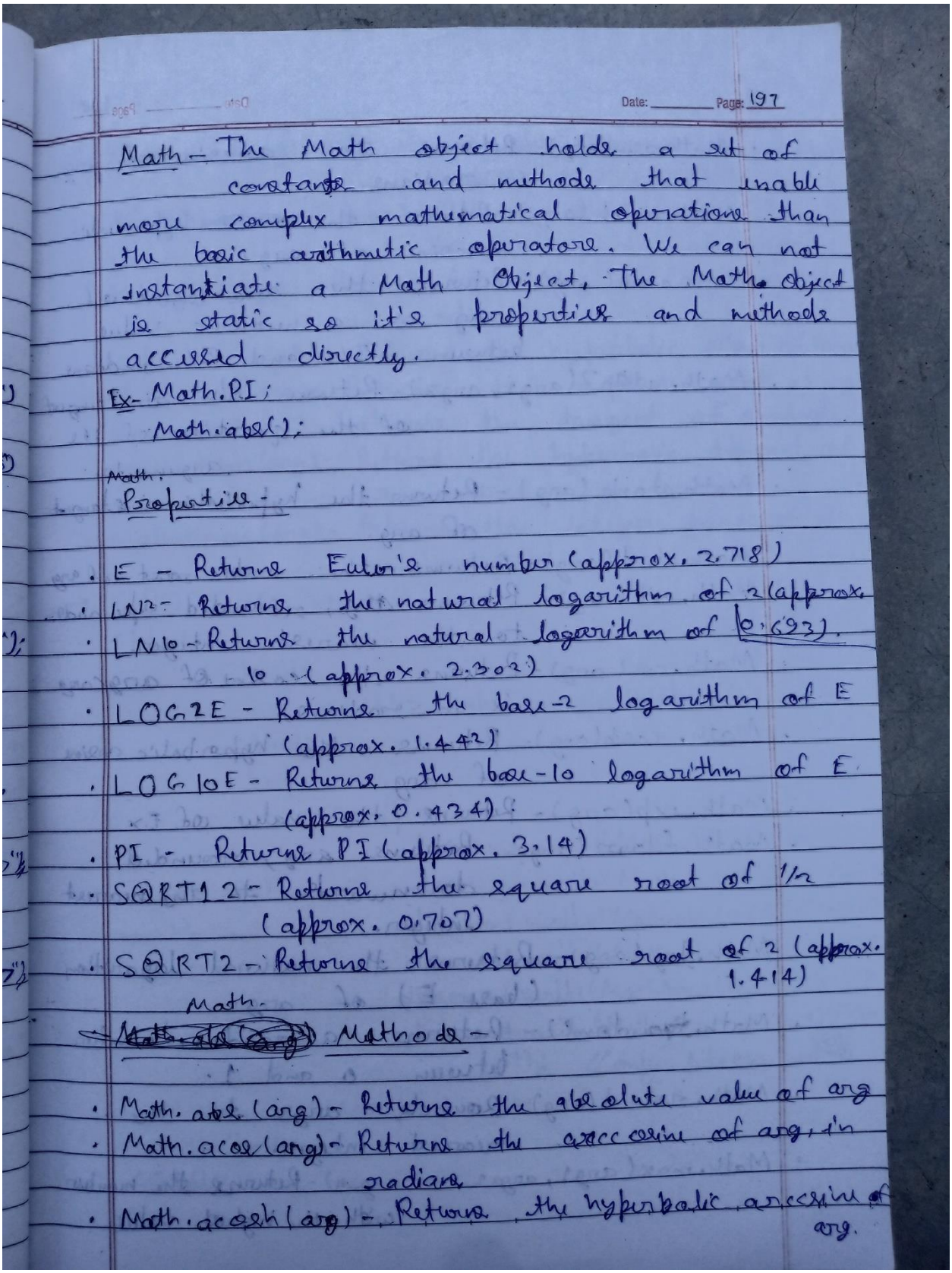
parseFloat() - The parseFloat() function parses a string and returns a floating point number. This function determines if the first character in the specified string is a number. This function determines if the first character in the specified string is a number. If it is, it parses the string until it reaches the end of the number, and returns the number as a number, not as a string.

Syntax - parseFloat(string).

- Only the first number in the string is returned!
- Leading and trailing spaces are allowed.
- If the first character cannot be converted to a number, parseFloat() returns NaN.



```
<!DOCTYPE html>
<html>
    <head>
        <title> GeekyShows </title>
    </head>
    <body>
        <script>
            document.write(parseFloat("10")
                + "<br>");
            document.write(parseFloat("12.00")
                + "<br>");
            document.write(parseFloat(
                "15.45") + "<br>");
            document.write(parseFloat(
                "10 20 30") + "<br>");
            document.write(parseFloat(
                "90") + "<br>");
            document.write(parseFloat("10
                years") + "<br>");
            document.write(parseFloat("
                years 10") + "<br>");
            document.write(parseFloat("
                020") + "<br>");
        </script>
    </body>
</html>
```



Date: Page: 197

Math - The Math object holds a set of constants and methods that enable more complex mathematical operations than the basic arithmetic operators. We can not instantiate a Math Object. The Math object is static so it's properties and methods accessed directly.

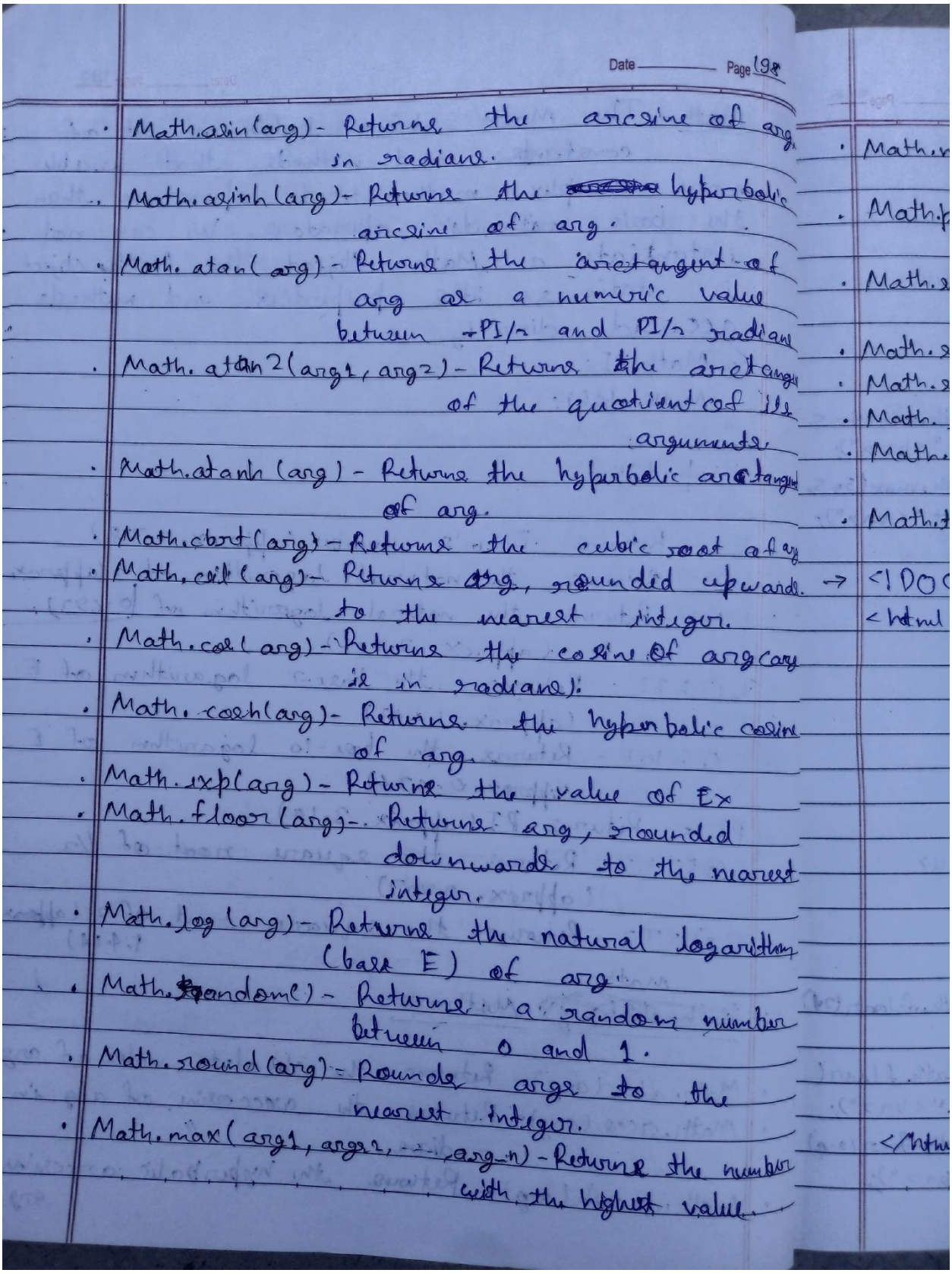
Ex- Math.PI;
Math.abs();

Math.
Properties -

- E - Returns Euler's number (approx. 2.718)
- LN2 - Returns the natural logarithm of 2 (approx. 0.693)
- LN10 - Returns the natural logarithm of 10 (approx. 2.302)
- LOG2E - Returns the base-2 logarithm of E (approx. 1.442)
- LOG10E - Returns the base-10 logarithm of E (approx. 0.434)
- PI - Returns PI (approx. 3.14)
- SQRT1_2 - Returns the square root of 1/2 (approx. 0.707)
- SQRT2 - Returns the square root of 2 (approx. 1.414)

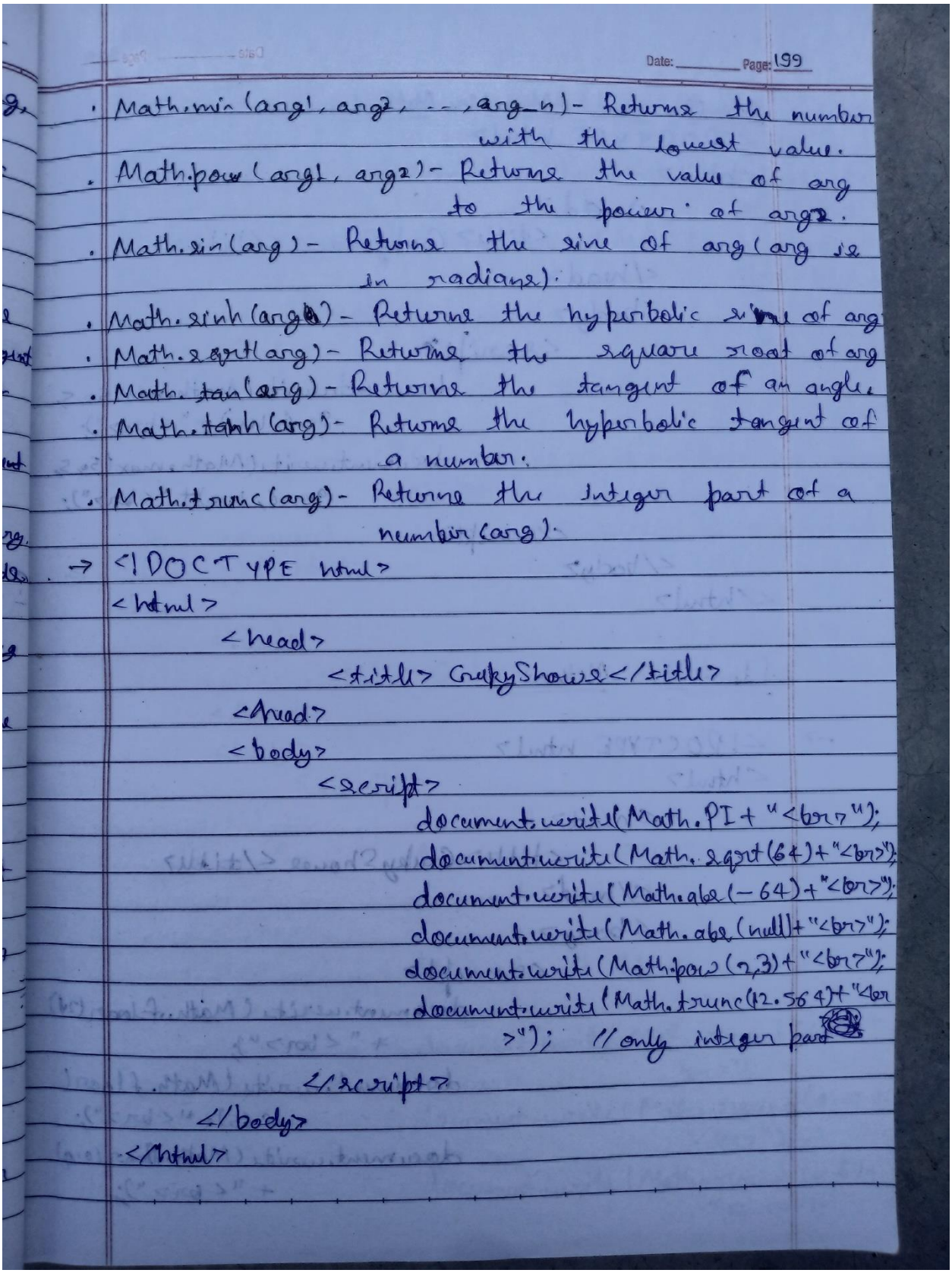
Math.
Methods -

- Math.abs(arg) - Returns the absolute value of arg
- Math.acos(arg) - Returns the arccosine of arg, in radians
- Math.acosh(arg) - Returns the hyperbolic arccosine of arg.



Date _____ Page 198

- Math.asin(arg) - Returns the arcsine of arg in radians.
- Math.asinh(arg) - Returns the hyperbolic arcsine of arg.
- Math.atan(arg) - Returns the arctangent of arg as a numeric value between -PI/2 and PI/2 radians
- Math.atan2(arg1, arg2) - Returns the arctangent of the quotient of its arguments
- Math.atanh(arg) - Returns the hyperbolic arctangent of arg.
- Math.cbrt(arg) - Returns the cubic root of arg
- Math.ceil(arg) - Returns arg, rounded upwards to the nearest integer.
- Math.cos(arg) - Returns the cosine of arg (arg is in radians).
- Math.cosh(arg) - Returns the hyperbolic cosine of arg.
- Math.exp(arg) - Returns the value of Ex
- Math.floor(arg) - Returns arg, rounded downwards to the nearest integer.
- Math.log(arg) - Returns the natural logarithm (base E) of arg.
- Math.random() - Returns a random number between 0 and 1.
- Math.round(arg) - Rounds arg to the nearest integer.
- Math.max(arg1, arg2, ... arg_n) - Returns the number with the highest value.

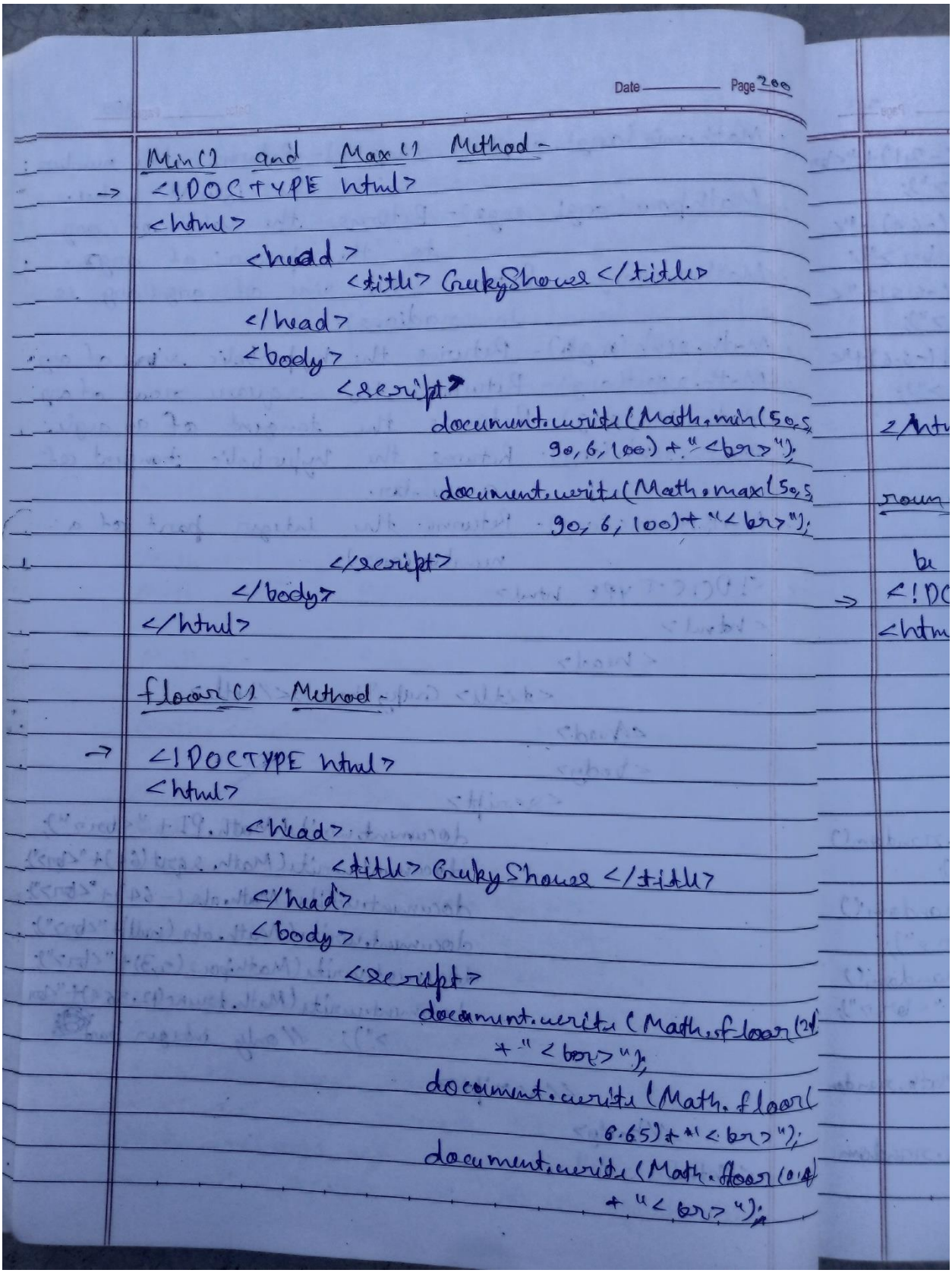


Date: Page: 199

- Math.min(arg1, arg2, ..., arg_n) - Returns the number with the lowest value.
- Math.pow(arg1, arg2) - Returns the value of arg to the power of arg2.
- Math.sin(arg) - Returns the sine of arg (arg is in radians).
- Math.sinh(arg) - Returns the hyperbolic sine of arg
- Math.sqrt(arg) - Returns the square root of arg
- Math.tan(arg) - Returns the tangent of an angle.
- Math.tanh(arg) - Returns the hyperbolic tangent of a number.
- Math.trunc(arg) - Returns the integer part of a number (arg).

→

```
<!DOCTYPE html>
<html>
    <head>
        <title> GeekyShows </title>
    </head>
    <body>
        <script>
            document.write(Math.PI + "<br>");
            document.write(Math.sqrt(64) + "<br>");
            document.write(Math.abs(-64) + "<br>");
            document.write(Math.abs(null) + "<br>");
            document.write(Math.pow(2,3) + "<br>");
            document.write(Math.trunc(12.564) + "<br
            >"); // only integer part
        </script>
    </body>
</html>
```



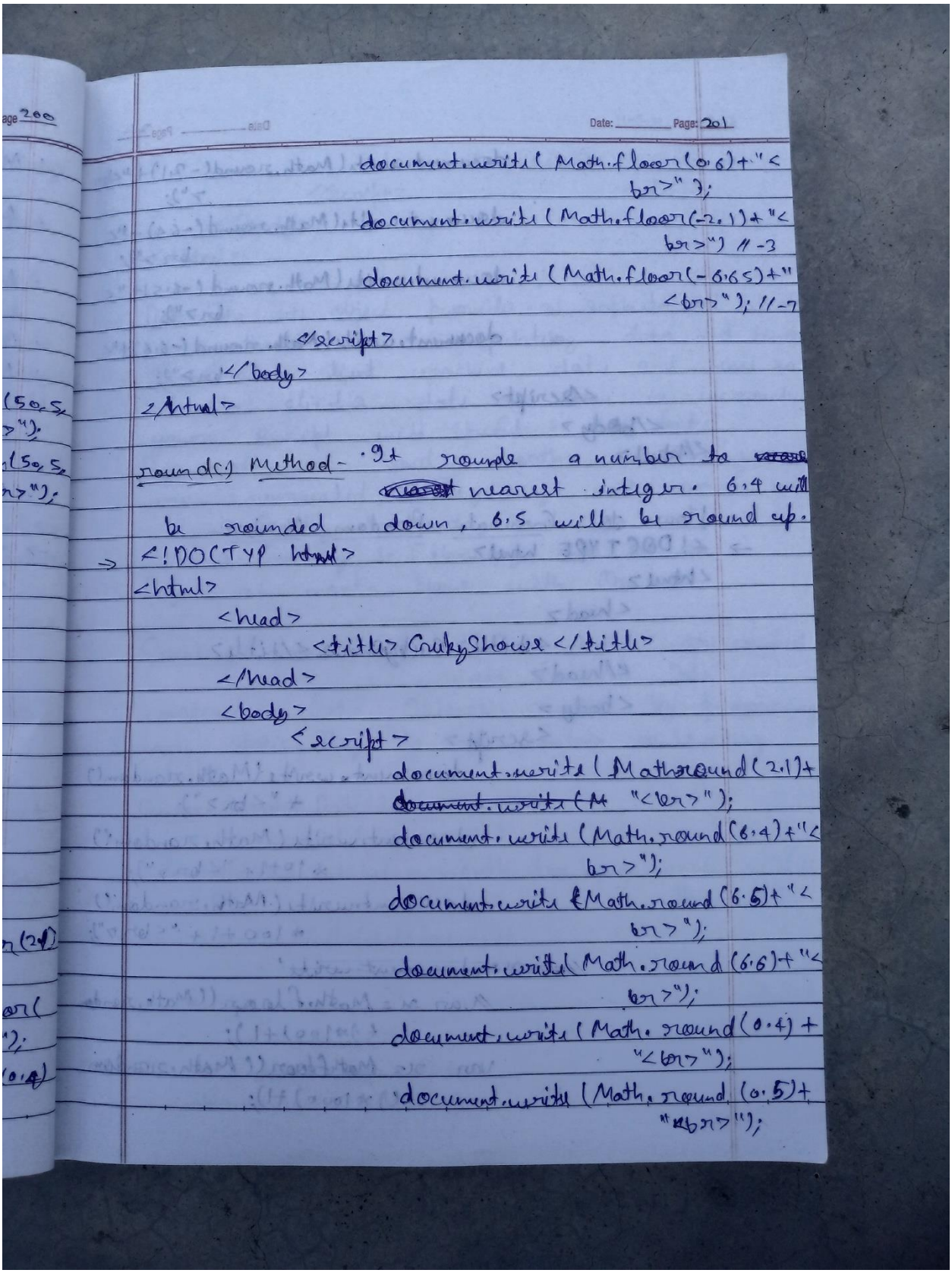
Date _____ Page 200

Min() and Max() Method -

```
<!DOCTYPE html>
<html>
    <head>
        <title> GeekyShows </title>
    </head>
    <body>
        <script>
            document.write(Math.min(50,90,6,100) + "<br>");
            document.write(Math.max(50,90,6,100) + "<br>");
        </script>
    </body>
</html>
```

floor() Method -

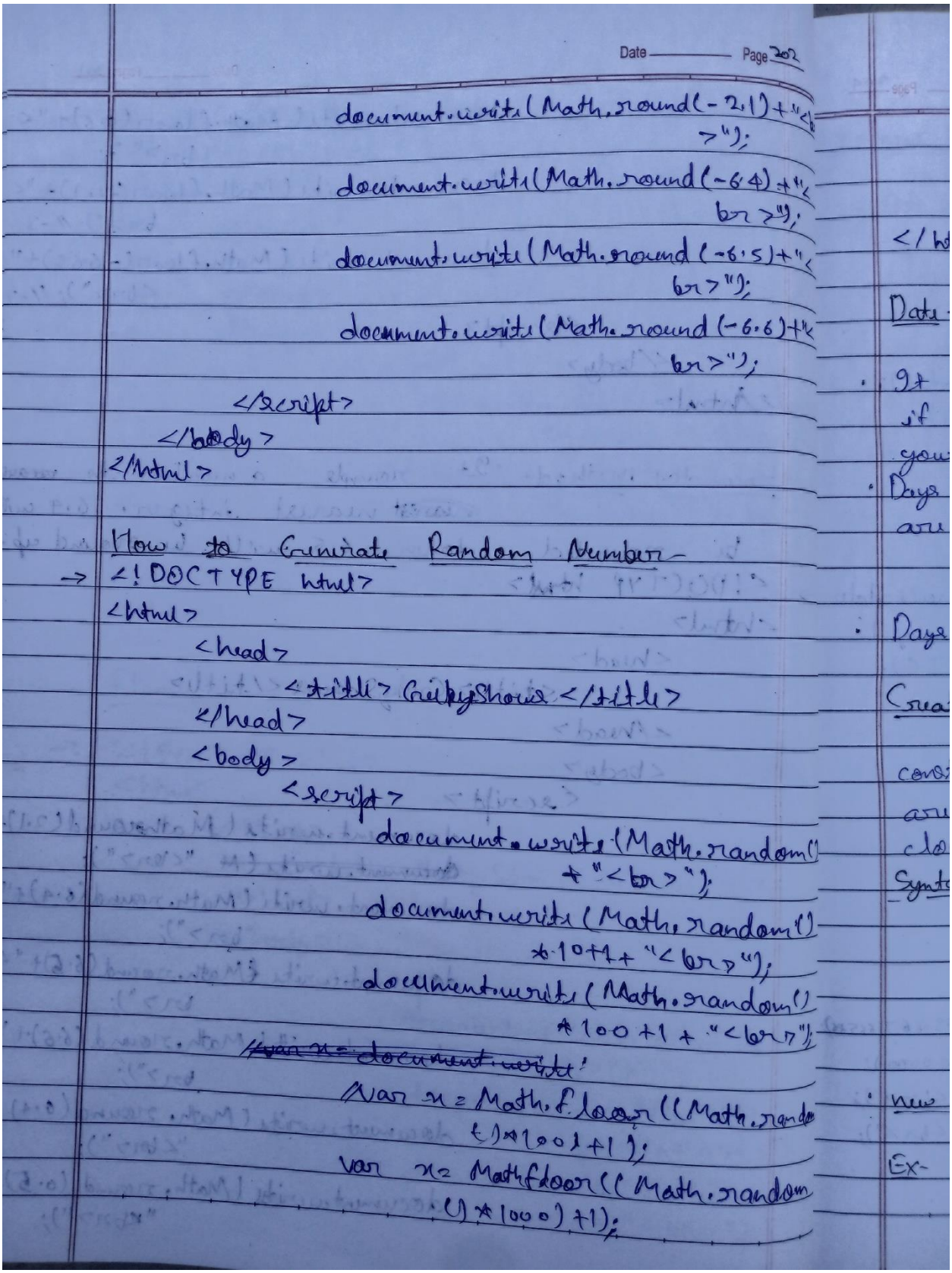
```
<!DOCTYPE html>
<html>
    <head>
        <title> GeekyShows </title>
    </head>
    <body>
        <script>
            document.write(Math.floor(2) + "<br>");
            document.write(Math.floor(6.65) + "<br>");
            document.write(Math.floor(0.4) + "<br>");
```



```
document.write( Math.floor(0.6)+"<
                br>" );
document.write( Math.floor(-2.1)+"<
                br>") // -3
document.write( Math.floor(-6.65)+"
                <br>" ); // -7
    </script>
  </body>
</html>
```

round() Method - It rounds a number to ~~nearest~~ nearest integer. 6.4 will be rounded down, 6.5 will be round up.

```
<!DOCTYP html>
<html>
    <head>
        <title> CrukyShows </title>
    </head>
    <body>
        <script>
            document.write( Math.round(2.1)+
            ~~document.write(M~~ "<br>" );
            document.write( Math.round(6.4)+"<
                br>" );
            document.write( Math.round(6.5)+"<
                br>" );
            document.write( Math.round(6.6)+"<
                br>" );
            document.write( Math.round(0.4) +
                "<br>" );
            document.write( Math.round(0.5)+
                "<br>" );
```



```
Date _____ Page 202

document.write(Math.round(-2.1)+"<br>");
document.write(Math.round(-6.4)+"<br>");
document.write(Math.round(-6.5)+"<br>");
document.write(Math.round(-6.6)+"<br>");
</script>
</body>
</html>
```

How to Generate Random Number -

```
<!DOCTYPE html>
<html>
    <head>
        <title> GeekyShows </title>
    </head>
    <body>
        <script>
            document.write(Math.random()
                + "<br>");
            document.write(Math.random()
                *10+1+ "<br>");
            document.write(Math.random()
                *100+1+ "<br>");
            //var x = document.write(
            var x = Math.floor((Math.rando
                t)*1001+1);
            var x = Math.floor((Math.random
                ()*1000)+1);
```

Date: 11-5-21 Page: 203

```
    document.write(x);
  </script>
</body>
</html>
```

Date - The Date object provides a sophisticated set of methods for manipulating date and times.

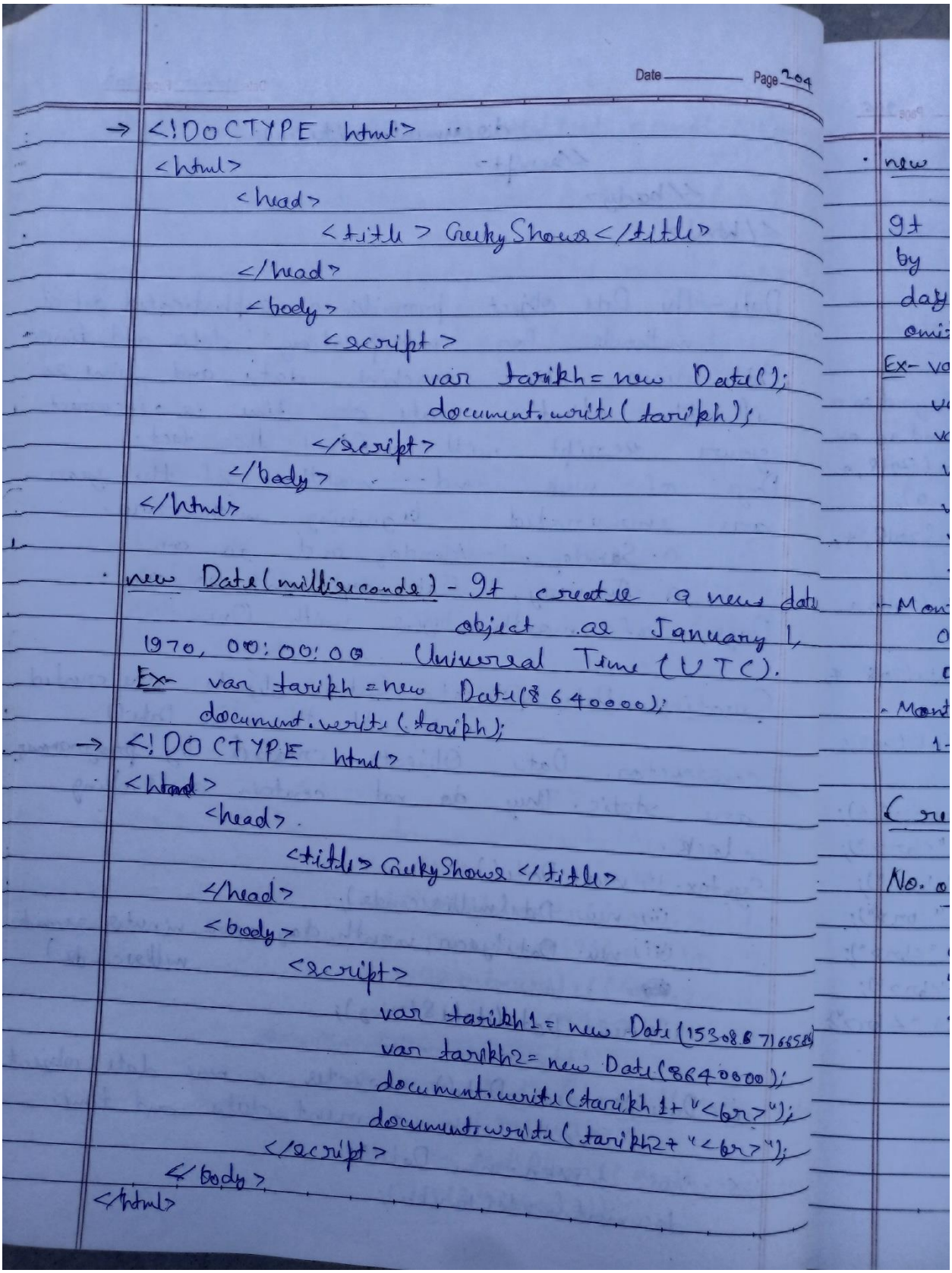
- It reads client machine date and time so if the client's date or time is incorrect, your script will reflect this fact.
- Days of week and months of the year are enumerated beginning with zero.
  0 - Sunday, 1 - Monday and so on.
  0 - January, 1 - February and so on.
- Days of month begins with One.

Creating Date Object - Date objects are created with the new Date() constructor. Date Objects created by programmers are static. They do not contain a ticking clock.

Syntax - (i) new Date();
(ii) new Date(milliseconds);
(iii) new Date(year, month, day, hours, minutes, seconds, milliseconds)
(iv) new Date(dateString);

- **new Date()** - new Date() creates a new date object with the current date and time.

Ex- var tarikh = new Date();
document.write(tarikh);

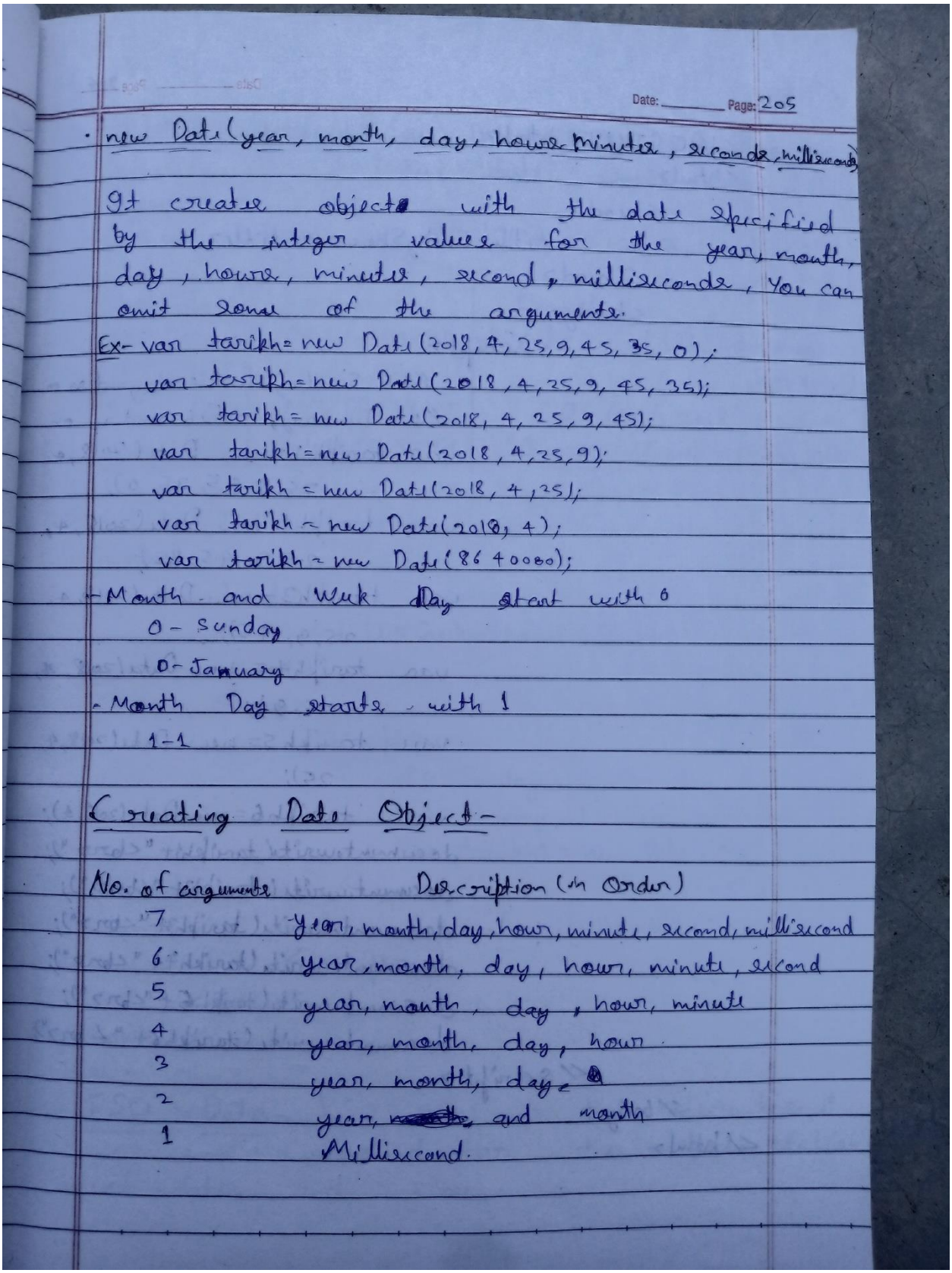


```
<!DOCTYPE html>
<html>
    <head>
        <title> GeekyShows </title>
    </head>
    <body>
        <script>
            var tarikh = new Date();
            document.write(tarikh);
        </script>
    </body>
</html>
```

new Date(milliseconds) - It create a new date object as January 1, 1970, 00:00:00 Universal Time (UTC).

Ex- var tarikh = new Date(8640000);
document.write(tarikh);

```
<!DOCTYPE html>
<html>
    <head>
        <title> GeekyShows </title>
    </head>
    <body>
        <script>
            var tarikh1 = new Date(1530871668520);
            var tarikh2 = new Date(8640000);
            document.write(tarikh1 + "<br>");
            document.write(tarikh2 + "<br>");
        </script>
    </body>
</html>
```



Date: Page: 205

- new Date(year, month, day, hours, minutes, seconds, milliseconds)

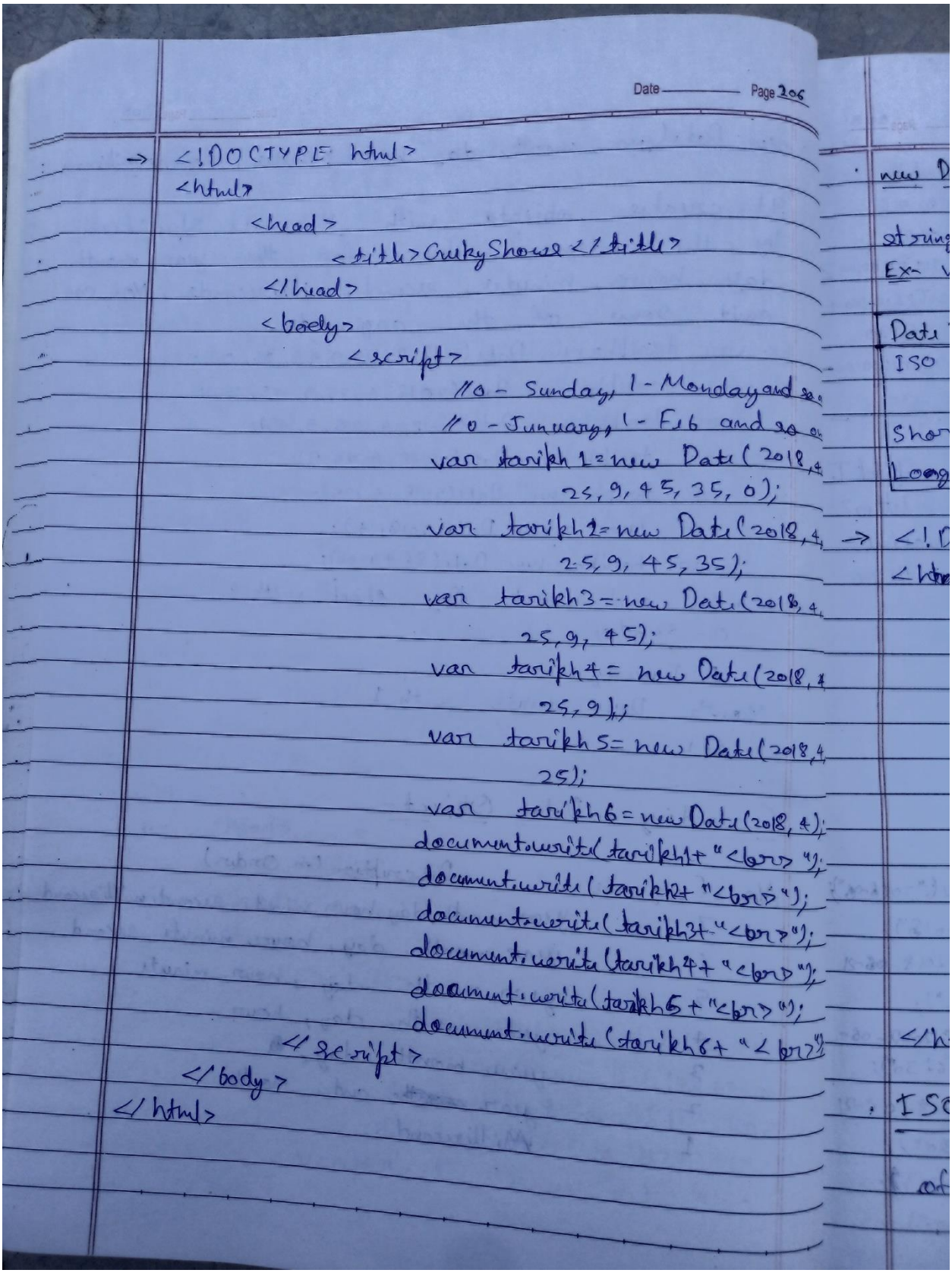
It creates objects with the date specified by the integer values for the year, month, day, hours, minutes, second, milliseconds. You can omit some of the arguments.

Ex- var tarikh = new Date(2018, 4, 25, 9, 45, 35, 0);
var tarikh = new Date(2018, 4, 25, 9, 45, 35);
var tarikh = new Date(2018, 4, 25, 9, 45);
var tarikh = new Date(2018, 4, 25, 9);
var tarikh = new Date(2018, 4, 25);
var tarikh = new Date(2018, 4);
var tarikh = new Date(86400000);

- Month and Week Day start with 0
  0 - Sunday
  0 - January
- Month Day starts with 1
  1 - 1

Creating Date Object -

No. of arguments	Description (in Order)
7	year, month, day, hour, minute, second, millisecond
6	year, month, day, hour, minute, second
5	year, month, day, hour, minute
4	year, month, day, hour
3	year, month, day
2	year, and month
1	Millisecond.



```
<!DOCTYPE html>
<html>
    <head>
        <title>GeekyShows</title>
    </head>
    <body>
        <script>
            //0 - Sunday, 1 - Monday and so.
            //0 - January, 1 - Feb and so on
            var tarikh1 = new Date(2018, 4,
                25, 9, 45, 35, 0);
            var tarikh2 = new Date(2018, 4,
                25, 9, 45, 35);
            var tarikh3 = new Date(2018, 4,
                25, 9, 45);
            var tarikh4 = new Date(2018, 4,
                25, 9);
            var tarikh5 = new Date(2018, 4,
                25);
            var tarikh6 = new Date(2018, 4);
            document.write(tarikh1 + "<br>");
            document.write(tarikh2 + "<br>");
            document.write(tarikh3 + "<br>");
            document.write(tarikh4 + "<br>");
            document.write(tarikh5 + "<br>");
            document.write(tarikh6 + "<br>");
        </script>
    </body>
</html>
```

Date: Page: 207

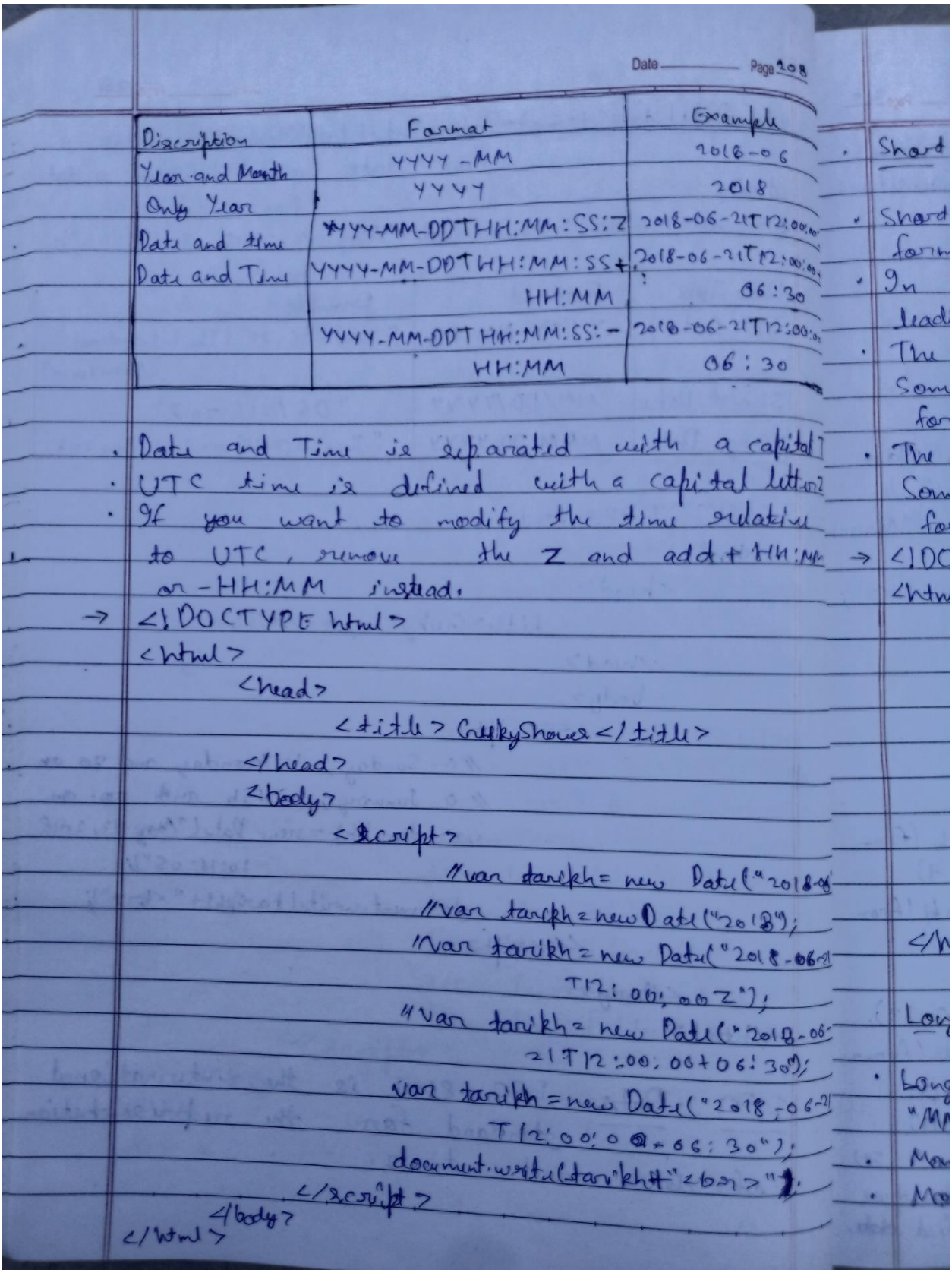
- new Date(dateString) - new Date(dateString) creates a new date object from a date string.

Ex- var tarikh = new Date("May 12, 2018 10:16:05");

Date Type	Format	Example
ISO Date	YYYY-MM-DD	"2018-06-21" (The International Standard)
Short Date	MM/DD/YYYY	"06/21/2018"
Long Date	MMM DD YYYY	"June 21 2018" or "21 June 2018"

→
```
<!DOCTYPE html>
<html>
    <head>
        <title> GeekyShows </title>
    </head>
    <body>
        <script>
            // 0 - Sunday, 1 - Monday and so on
            // 0 - January, 1 - Feb and so on
            var tarikh1 = new Date("May 12, 2018 10:16:05");
            document.write(tarikh1 + "<br>");
        </script>
    </body>
</html>
```

- ISO Date - ISO 8601 is the international standard for the representation of dates and times.

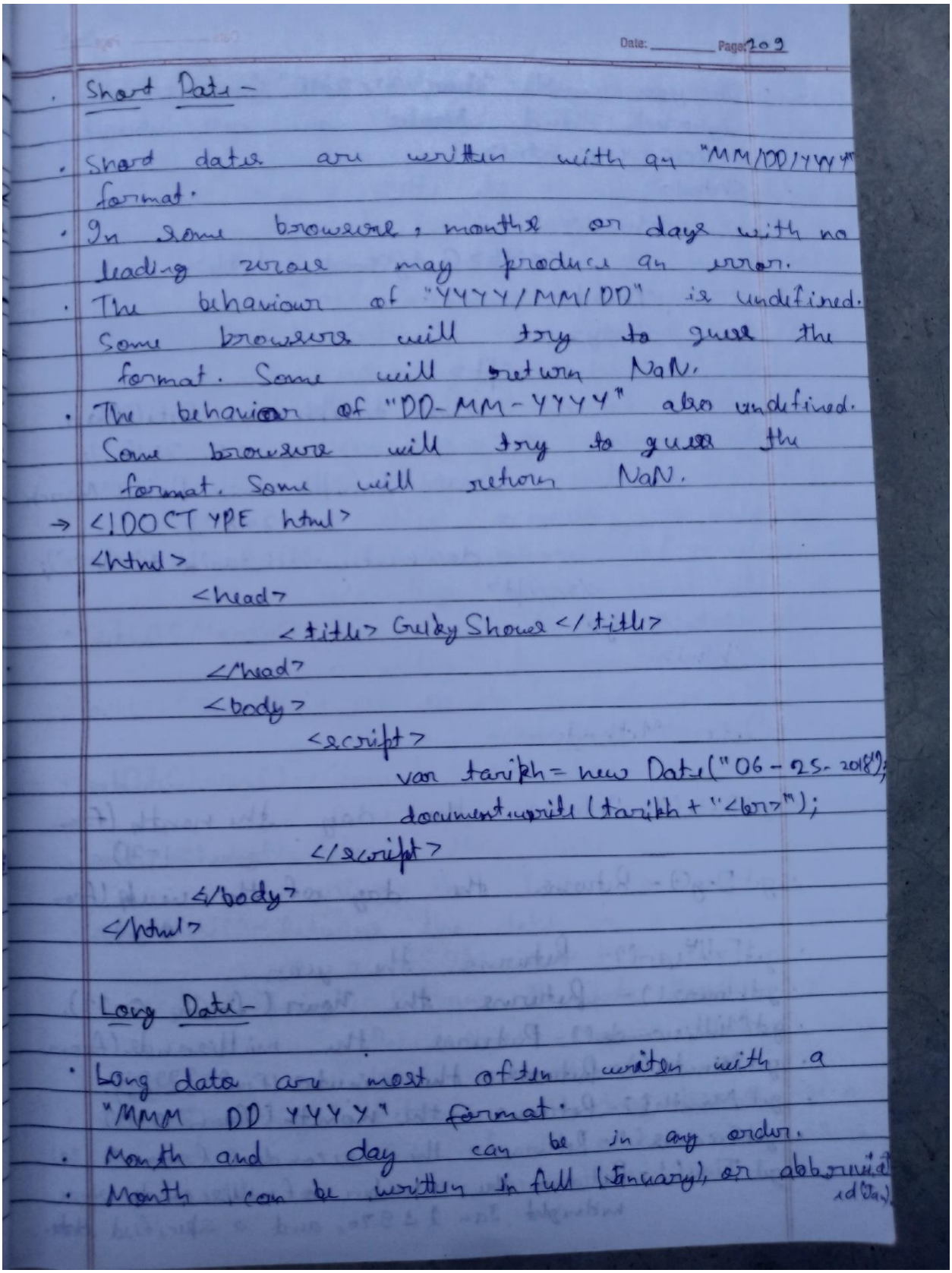


Date _____ Page 208

Discription	Format	Example
Year and Month	YYYY-MM	2018-06
Only Year	YYYY	2018
Date and time	YYYY-MM-DDTHH:MM:SS:Z	2018-06-21T12:00:00.00
Date and Time	YYYY-MM-DDTHH:MM:SS+ HH:MM	2018-06-21T12:00:00 06:30
	YYYY-MM-DDT HH:MM:SS:- HH:MM	2018-06-21T12:00:00 06:30

- Date and Time is separated with a capital T
- UTC time is defined with a capital letter Z
- If you want to modify the time relative to UTC, remove the Z and add +HH:MM or -HH:MM instead.

```
<!DOCTYPE html>
<html>
    <head>
        <title> GeekyShows </title>
    </head>
    <body>
        <script>
            //var tarikh = new Date("2018-06
            //var tarikh = new Date("2018");
            //var tarikh = new Date("2018-06-21
                T12:00:00.00Z");
            //var tarikh = new Date("2018-06-
                21T12:00:00+06:30");
            var tarikh = new Date("2018-06-21
                T12:00:00.00+06:30");
            document.write(tarikh+"<br>");
        </script>
    </body>
</html>
```



Date: Page: 209

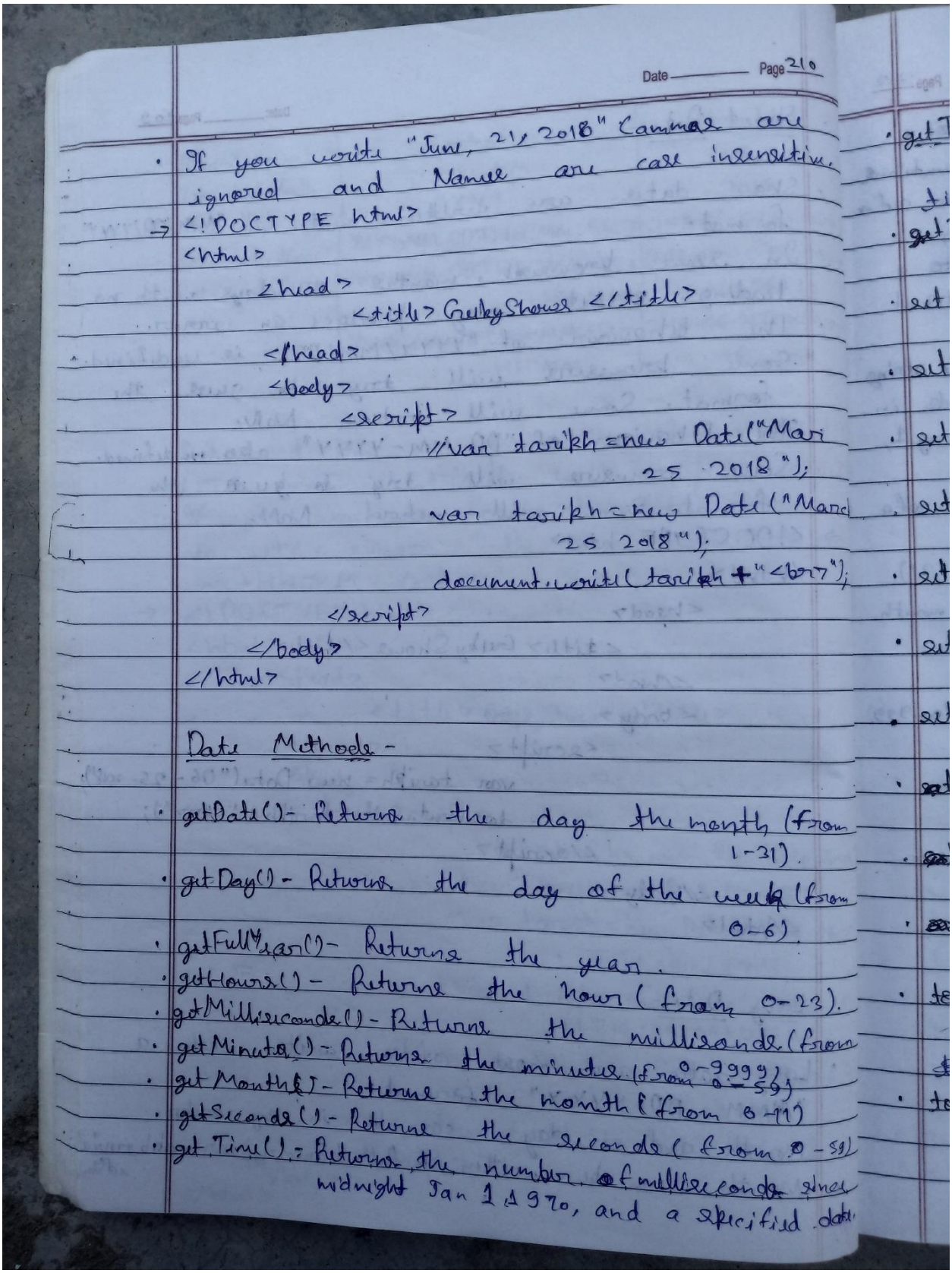
. Short Date -

. Short dates are written with an "MM/DD/YYYY" format.
. In some browsers, months or days with no leading zeros may produce an error.
. The behaviour of "YYYY/MM/DD" is undefined. Some browsers will try to guess the format. Some will return NaN.
. The behaviour of "DD-MM-YYYY" also undefined. Some browsers will try to guess the format. Some will return NaN.

→
```
<!DOCTYPE html>
<html>
    <head>
        <title> Geeky Shows </title>
    </head>
    <body>
        <script>
            var tarikh = new Date("06-25-2018");
            document.write(tarikh + "<br>");
        </script>
    </body>
</html>
```

Long Date -

. Long dates are most often written with a "MMM DD YYYY" format.
. Month and day can be in any order.
. Month can be written in full (January) or abbreviated (Jan).



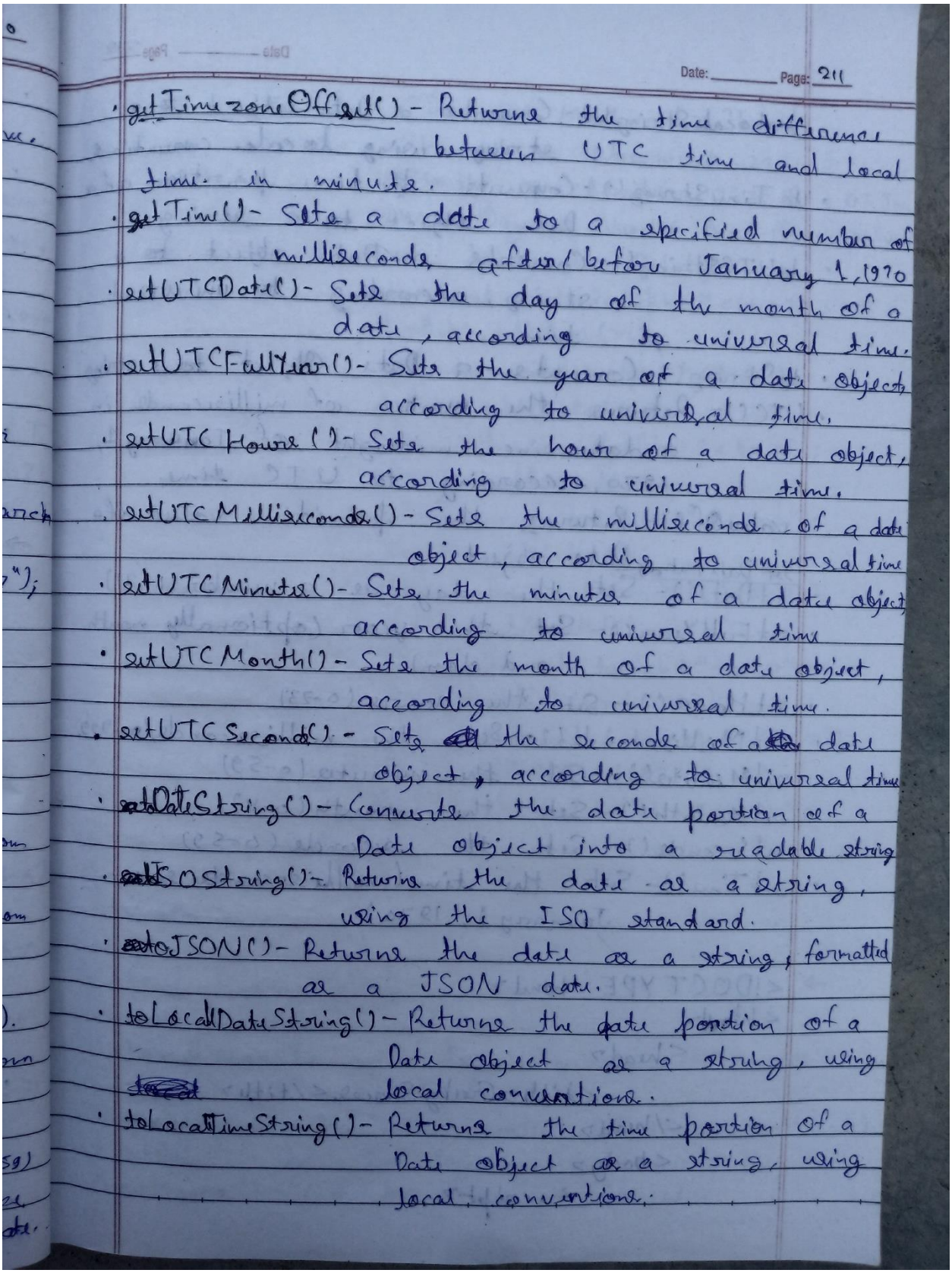
Date _____ Page 210

- If you write "June, 21, 2018" Cammas are ignored and Names are case insensitive.

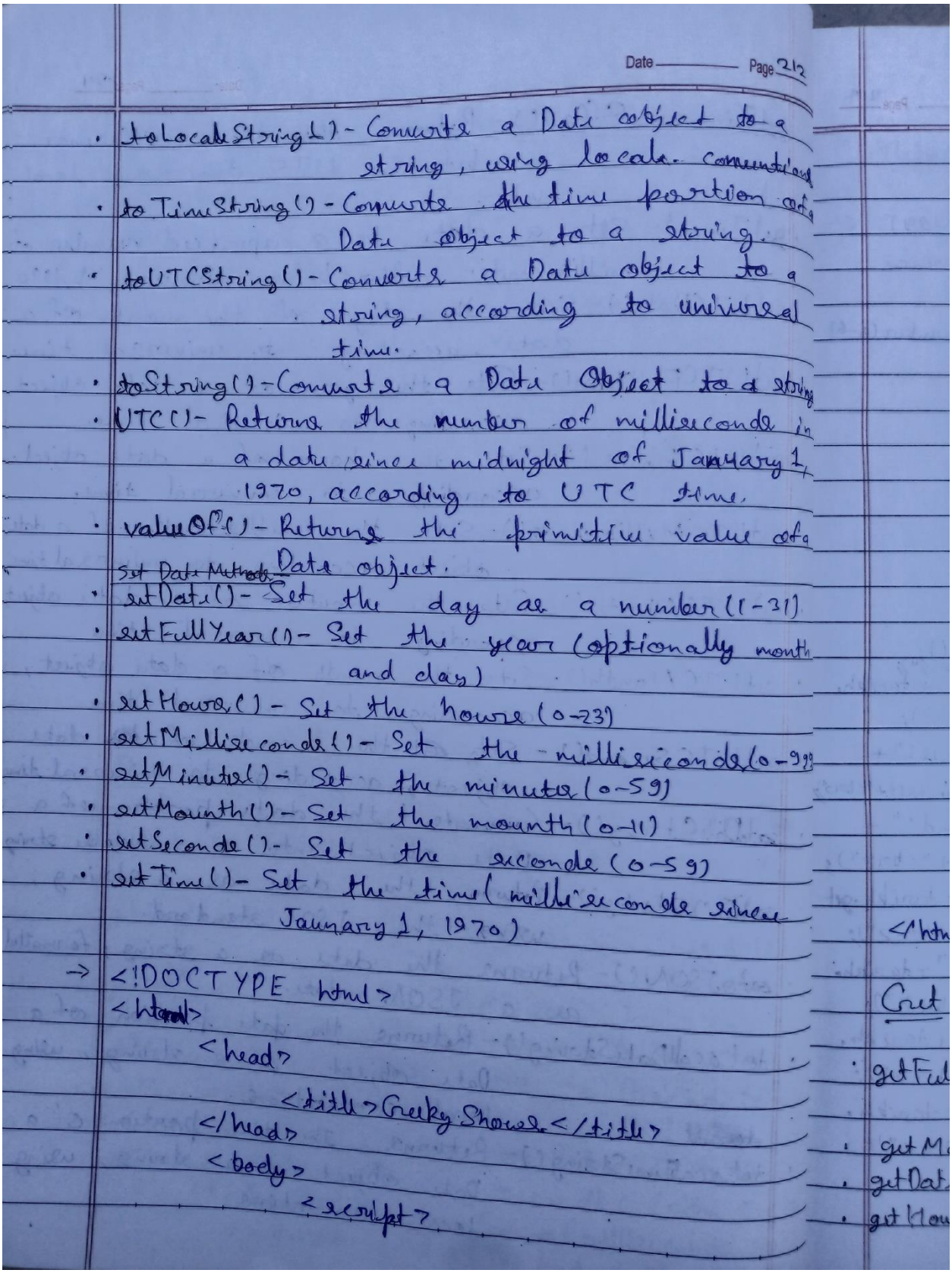
```
<!DOCTYPE html>
<html>
    <head>
        <title> GeekyShows </title>
    </head>
    <body>
        <script>
            //var tarikh = new Date("Mar
                                25 2018");
            var tarikh = new Date("March
                                25 2018");
            document.write(tarikh + "<br>");
        </script>
    </body>
</html>
```

Date Methods -

- getDate() - Returns the day the month (from 1-31).
- getDay() - Returns the day of the week (from 0-6).
- getFullYear() - Returns the year.
- getHours() - Returns the hours (from 0-23).
- getMilliseconds() - Returns the milliseconds (from 0-999).
- getMinutes() - Returns the minutes (from 0-59).
- getMonth() - Returns the month (from 0-11)
- getSeconds() - Returns the seconds (from 0-59)
- getTime() - Returns the number of milliseconds since midnight Jan 1, 1970, and a specified date



- getTimezoneOffset() - Returns the time difference between UTC time and local time in minute.
- getTime() - Sets a date to a specified number of milliseconds after/before January 1, 1970
- setUTCDate() - Sets the day of the month of a date, according to universal time.
- setUTCFullYear() - Sets the year of a date object according to universal time.
- setUTCHours() - Sets the hour of a date object, according to universal time.
- setUTCMilliseconds() - Sets the milliseconds of a date object, according to universal time
- setUTCMinutes() - Sets the minutes of a date object according to universal time
- setUTCMonth() - Sets the month of a date object, according to universal time.
- setUTCSeconds() - Sets the seconds of a date object, according to universal time
- toDateString() - Converts the date portion of a Date object into a readable string
- toISOString() - Returns the date as a string, using the ISO standard.
- toJSON() - Returns the date as a string, formatted as a JSON date.
- toLocaleDateString() - Returns the date portion of a Date object as a string, using local conventions.
- toLocaleTimeString() - Returns the time portion of a Date object as a string, using local conventions.



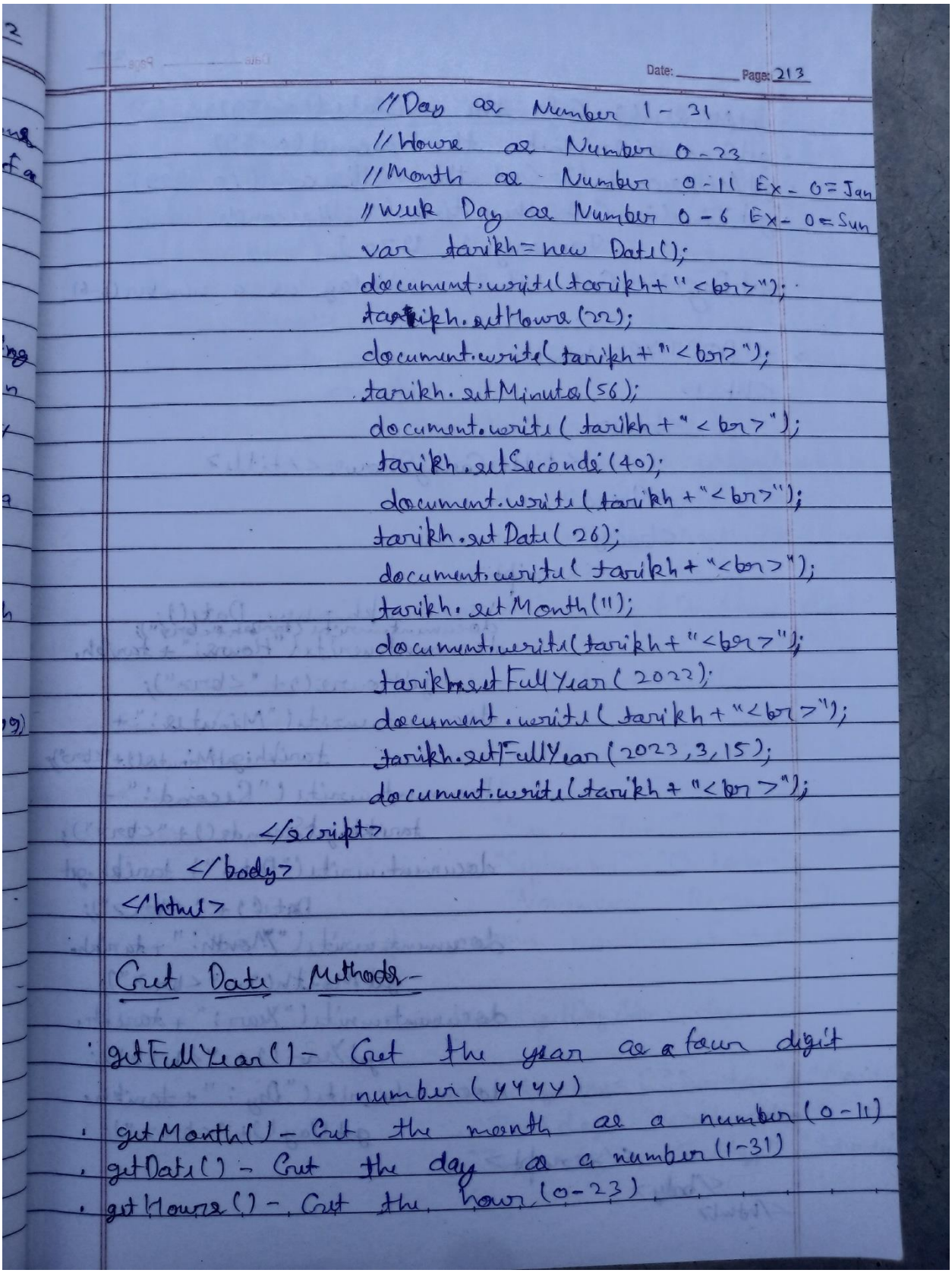
Date _____ Page 212

- toLocaleString() - Converts a Date object to a string, using locale conventions
- toTimeString() - Converts the time portion of a Date object to a string.
- toUTCString() - Converts a Date object to a string, according to universal time.
- toString() - Converts a Date Object to a string
- UTC() - Returns the number of milliseconds in a date since midnight of January 1, 1970, according to UTC time.
- valueOf() - Returns the primitive value of a Date object.

Set Date Methods

- setDate() - Set the day as a number (1-31)
- setFullYear() - Set the year (optionally month and day)
- setHours() - Set the hours (0-23)
- setMilliseconds() - Set the milliseconds (0-999)
- setMinutes() - Set the minutes (0-59)
- setMonth() - Set the month (0-11)
- setSeconds() - Set the seconds (0-59)
- setTime() - Set the time (milliseconds since January 1, 1970)

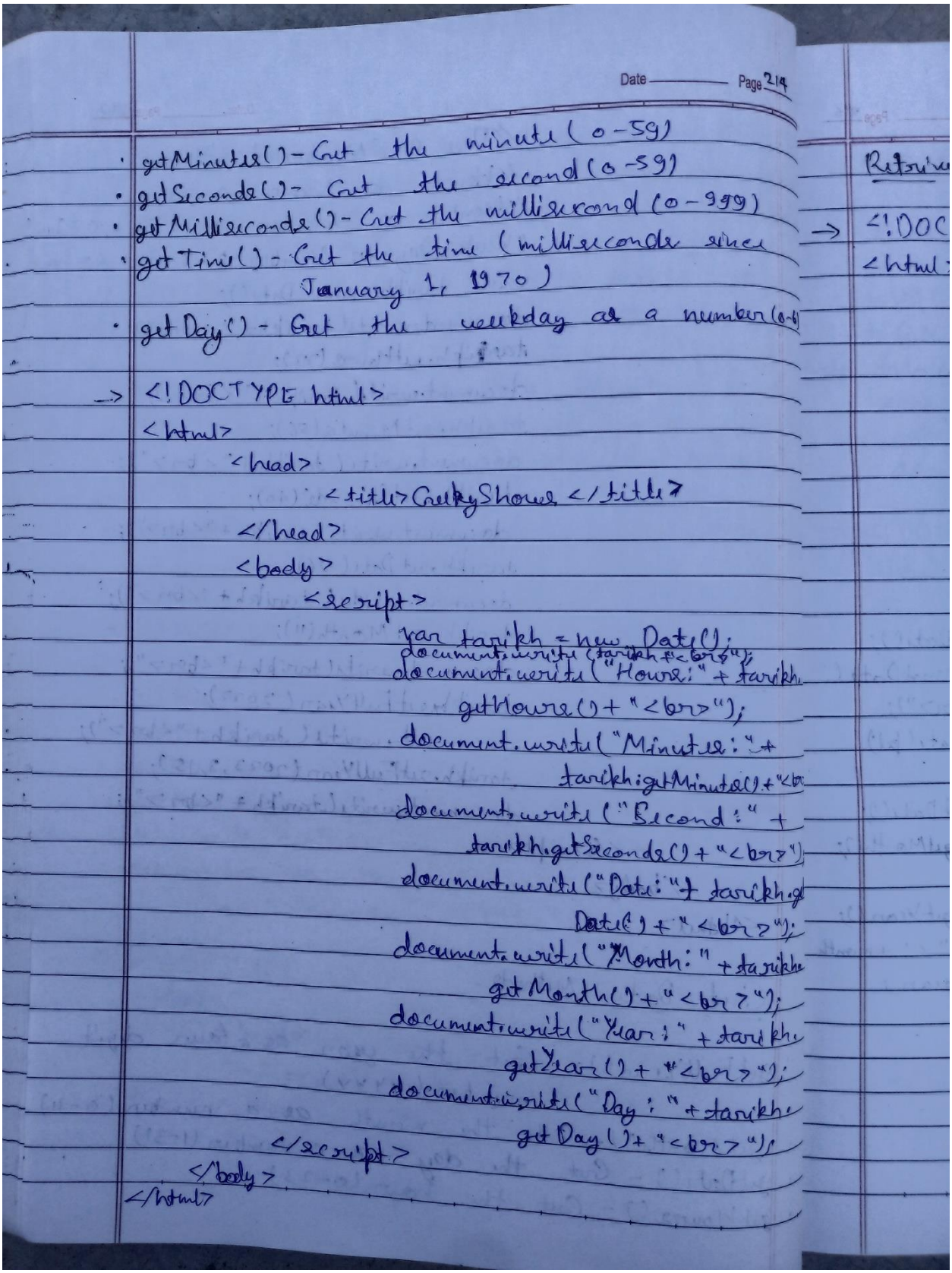
→
```
<!DOCTYPE html>
<html>
    <head>
        <title>Geeky Shows</title>
    </head>
    <body>
        <script>
```



```
//Day as Number 1 - 31
//Hours as Number 0 - 23
//Month as Number 0 - 11 Ex - 0 = Jan
//Week Day as Number 0 - 6 Ex - 0 = Sun
var tarikh = new Date();
document.write(tarikh + "<br>");
tarikh.setHours(22);
document.write(tarikh + "<br>");
tarikh.setMinutes(56);
document.write(tarikh + "<br>");
tarikh.setSeconds(40);
document.write(tarikh + "<br>");
tarikh.setDate(26);
document.write(tarikh + "<br>");
tarikh.setMonth(11);
document.write(tarikh + "<br>");
tarikh.setFullYear(2022);
document.write(tarikh + "<br>");
tarikh.setFullYear(2023,3,15);
document.write(tarikh + "<br>");
</script>
</body>
</html>
```

Get Date Methods-

- getFullYear() - Get the year as a four digit number (yyyy)
- getMonth() - Get the month as a number (0-11)
- getDate() - Get the day as a number (1-31)
- getHours() - Get the hour (0-23)



Date _____ Page 214

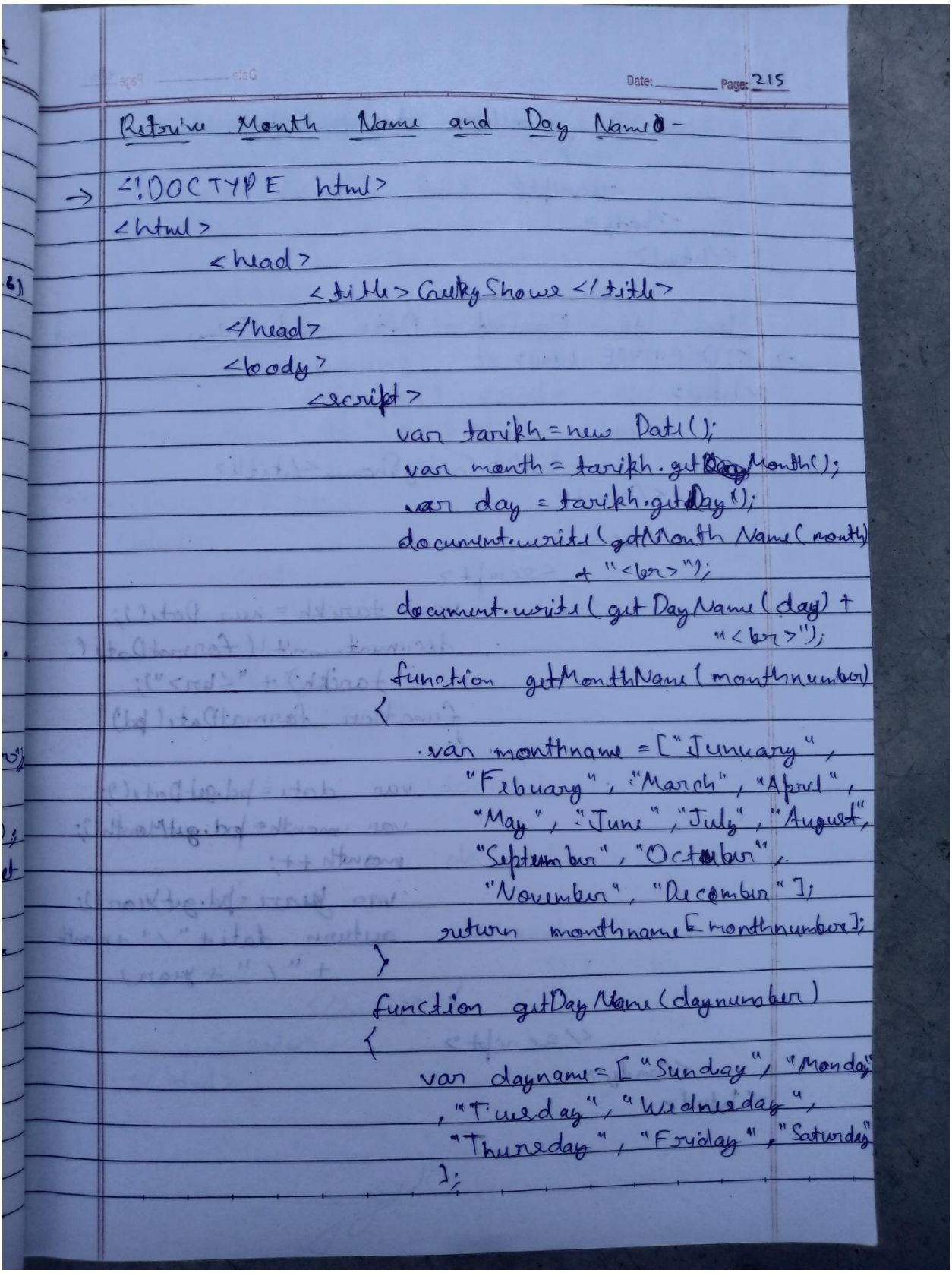
- getMinutes() - Get the minute (0-59)
- getSeconds() - Get the second (0-59)
- getMilliseconds() - Get the millisecond (0-999)
- getTime() - Get the time (milliseconds since January 1, 1970)
- getDay() - Get the weekday as a number (0-6)

```
<!DOCTYPE html>
<html>
    <head>
        <title>GeekyShows</title>
    </head>
    <body>
        <script>
            var tarikh = new Date();
            document.write(tarikh+"<br>");
            document.write("Hours:" + tarikh.
                getHours() + "<br>");
            document.write("Minutes:" +
                tarikh.getMinutes()+"<br
            document.write("Second:" +
                tarikh.getSeconds() + "<br>");
            document.write("Date:" + tarikh.g
                Date() + "<br>");
            document.write("Month:" + tarikh
                getMonth() + "<br>");
            document.write("Year:" + tarikh.
                getYear() + "<br>");
            document.write("Day:" + tarikh.
                getDay() + "<br>");
        </script>
    </body>
</html>
```

Retriv

→ <!DOC

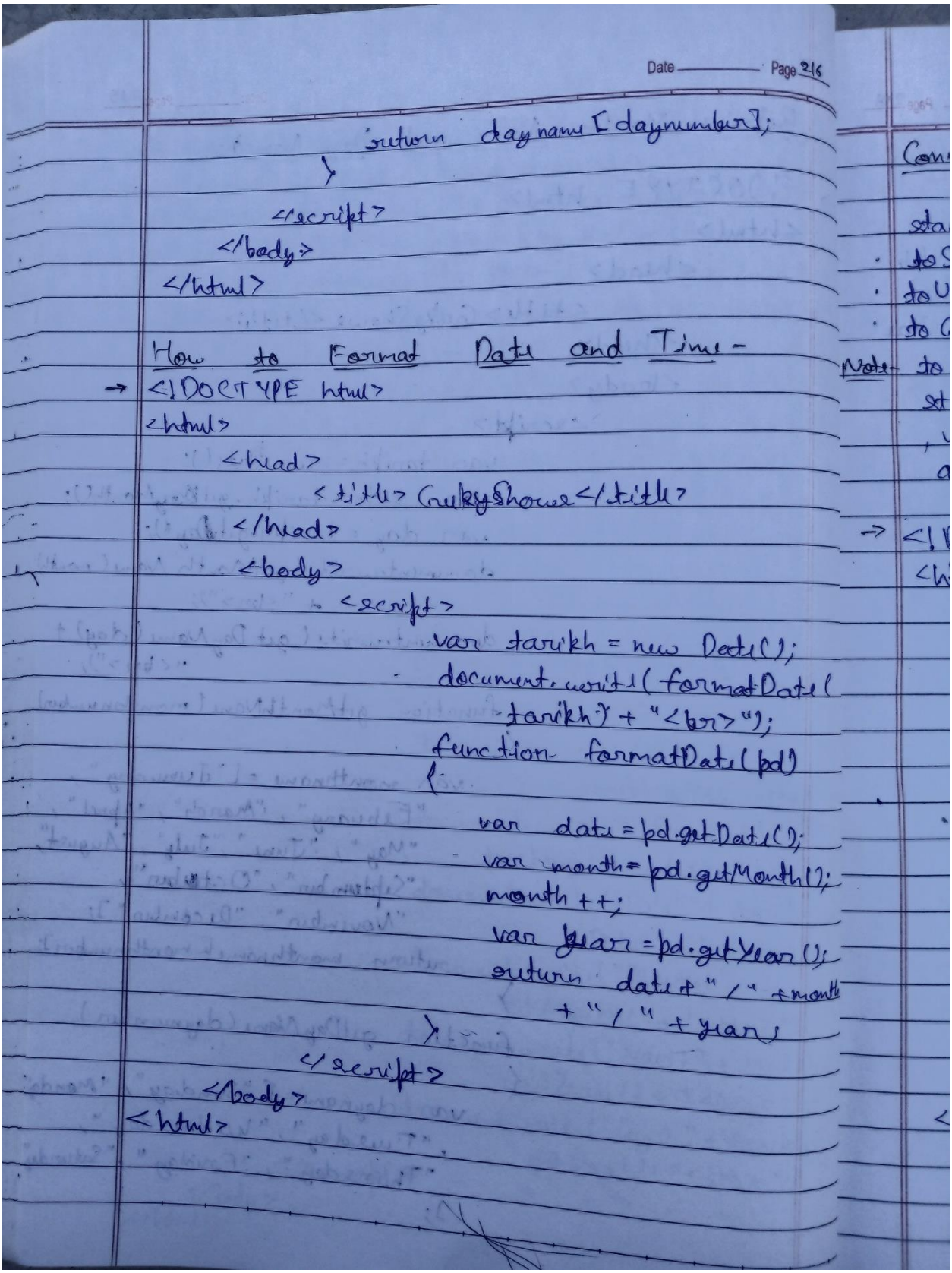
<html



Date: Page: 215

Retrive Month Name and Day Name -

```
<!DOCTYPE html>
<html>
    <head>
        <title> Geeky Shows </title>
    </head>
    <body>
        <script>
            var tarikh = new Date();
            var month = tarikh.getMonth();
            var day = tarikh.getDay();
            document.write( getMonthName(month)
                + "<br>");
            document.write( getDayName(day) +
                "<br>");
            function getMonthName(monthnumber)
            {
                var monthname = ["January",
                "February", "March", "April",
                "May", "June", "July", "August",
                "September", "October",
                "November", "December"];
                return monthname[monthnumber];
            }
            function getDayName(daynumber)
            {
                var dayname = ["Sunday", "Monday"
                , "Tuesday", "Wednesday",
                "Thursday", "Friday", "Saturday"
                ];
```



Date _____ Page 216

```
        return dayname[daynumber];
    }
</script>
</body>
</html>
```

How to Format Date and Time -

```
<!DOCTYPE html>
<html>
    <head>
        <title>GeekyShows</title>
    </head>
    <body>
        <script>
            var tarikh = new Date();
            document.write(formatDate(
                tarikh) + "<br>");
            function formatDate(pd)
            {
                var date = pd.getDate();
                var month = pd.getMonth();
                month++;
                var year = pd.getYear();
                return date + "/" + month
                    + "/" + year;
            }
        </script>
    </body>
</html>
```

Date: Page: 217

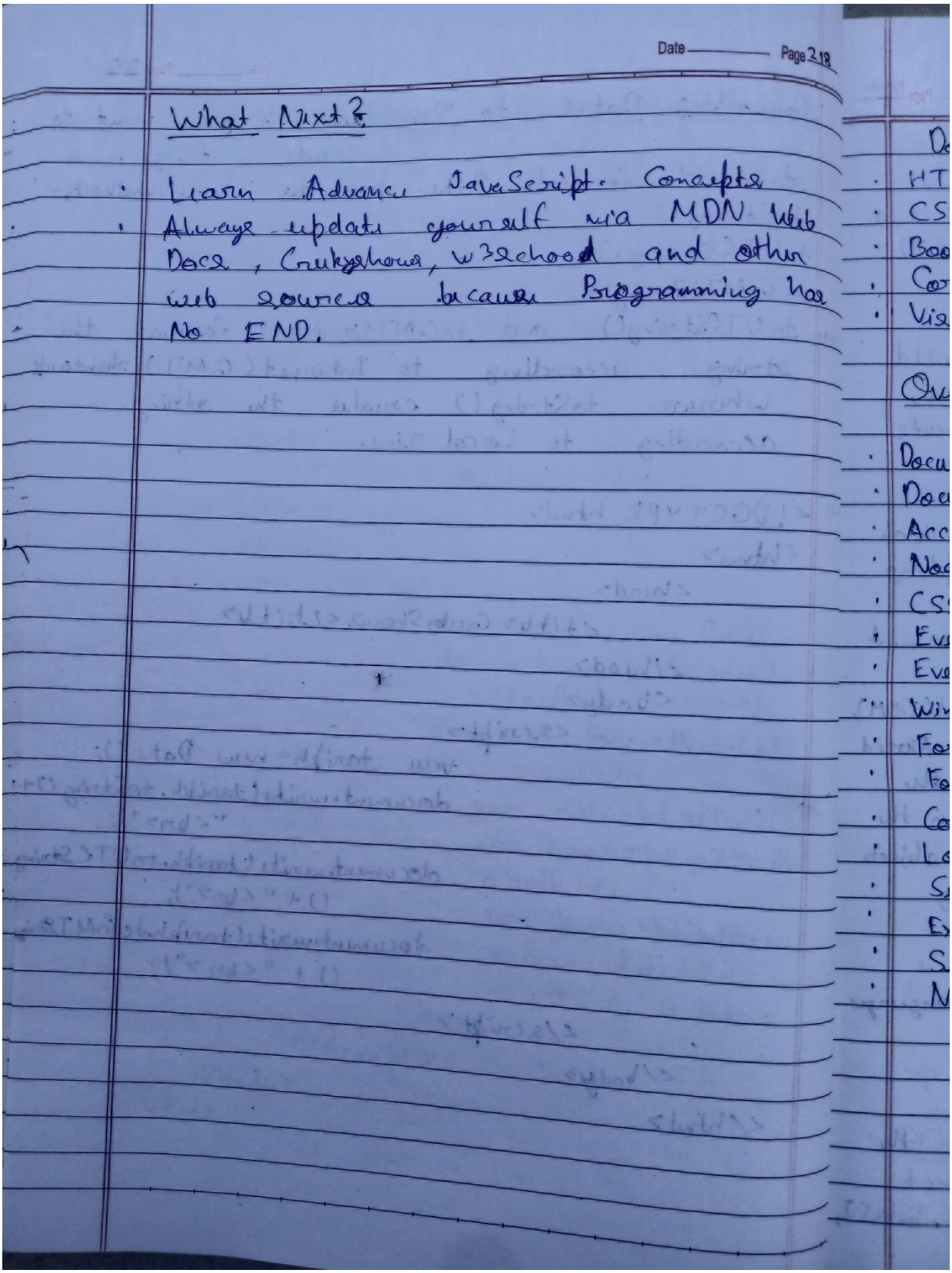
Converting Date to String - If you want to create a string in a standard format, Date provides three methods -

- toString()
- toUTCString()
- toGMTString()

Note - toUTCString() and toGMTString() format the string according to Internet (GMT) standards, whereas toString() creates the string according to Local Time.

→

```
<!DOCTYPE html>
<html>
    <head>
        <title> GeekyShows </title>
    </head>
    <body>
        <script>
            var tarikh = new Date();
            document.write(tarikh.toString()+
                "<br>");
            document.write(tarikh.toUTCString
                ()+"<br>");
            document.write(tarikh.toGMTString
                ()+"<br>");
        </script>
    </body>
</html>
```



Date _____ Page 218

What Next?

- Learn Advance JavaScript Concepts
- Always update yourself via MDN Web Docs, Geekyshows, w3schools and other web sources because Programming has No END.