Core Python Programming Language

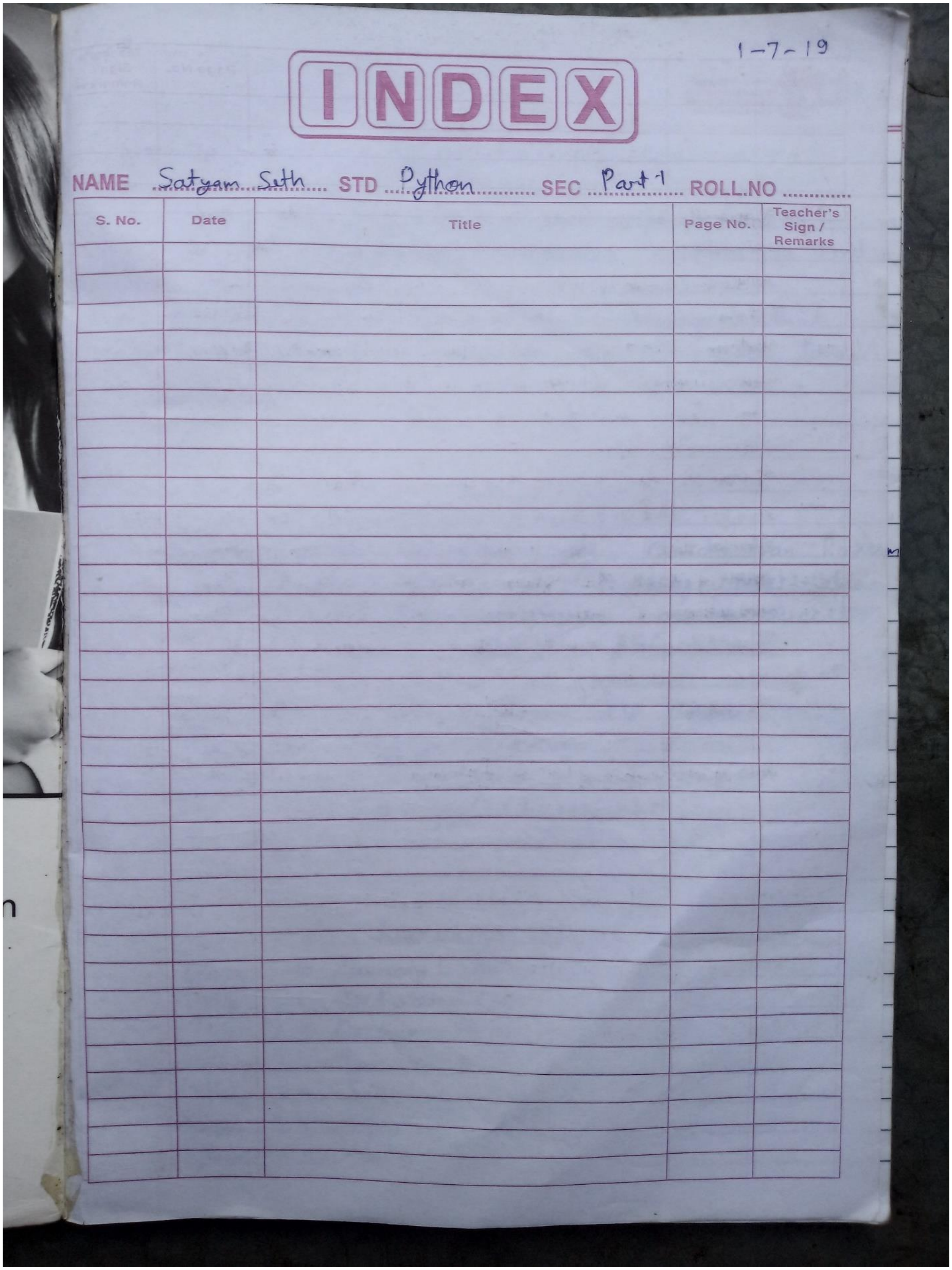
GEEKYSHOWS YOUTUBE CHANNEL LEARNING NOTES

Source Code- https://github.com/satyam-seth-learnings/python_learning/tree/main/Geeky%20Shows/Core%20Python

Playlist Link- https://youtube.com/playlist?list=PLbGui_ZYuhigZkqrHbI_ZkPBrIr5Rsd5L

SATYAM SETH

19-09-2021

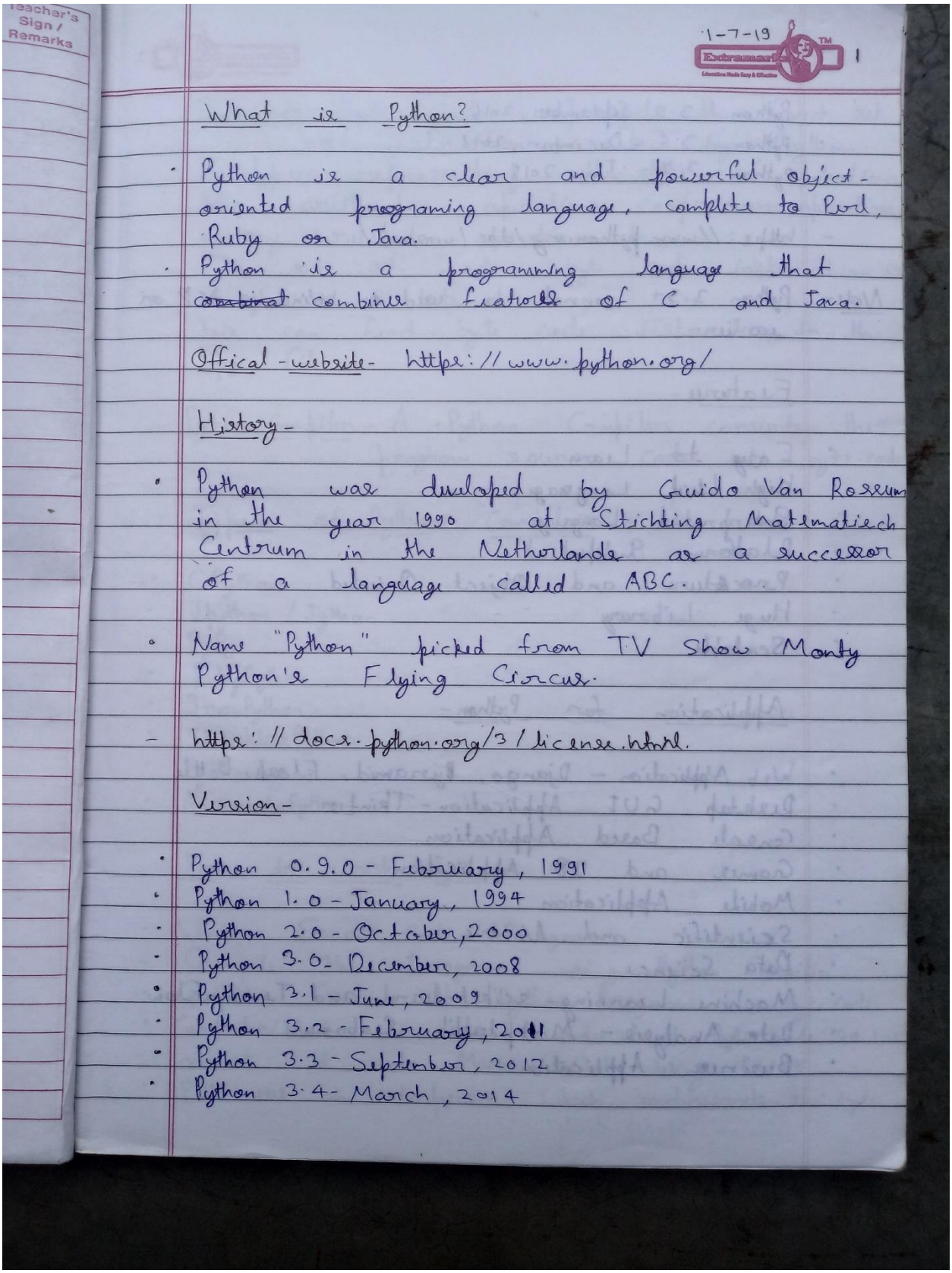


1-7-19

INDEX

NAME Satyam Seth STD Python SEC Part 1 ROLL.NO ....

S. No.	Date	Title	Page No.	Teacher's Sign / Remarks



Teacher's Sign / Remarks

1-7-19

What is Python?

- Python is a clear and powerful object-oriented programming language, complete to Perl, Ruby or Java.
- Python is a programming language that ~~combinat~~ combines features of C and Java.

Offical - website - https://www.python.org/

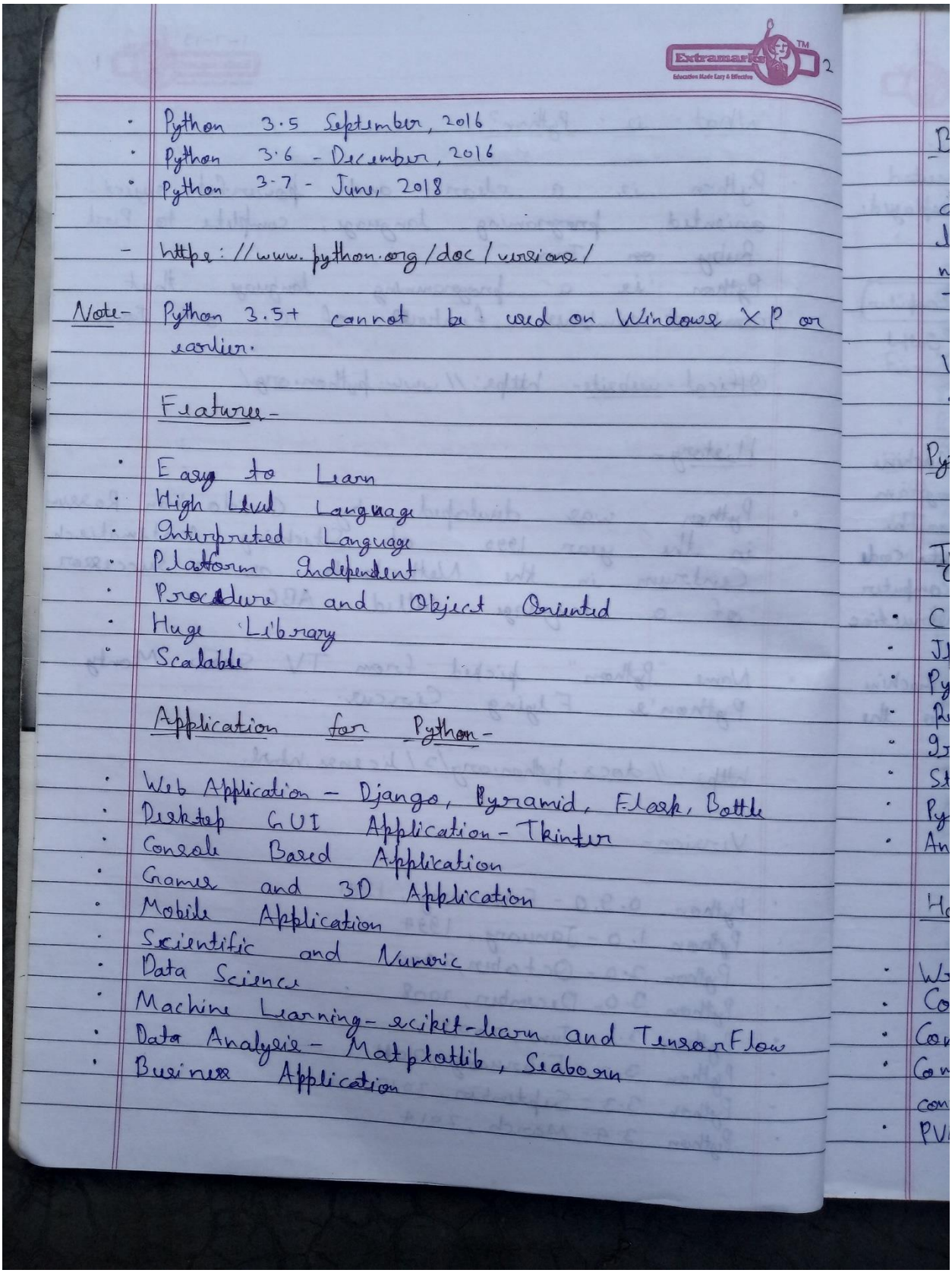
History -

- Python was developed by Guido Van Rossum in the year 1990 at Stichting Matematiech Centrum in the Netherlands as a successor of a language called ABC.
- Name "Python" picked from TV Show Monty Python's Flying Circus.

- https://docs.python.org/3/license.html.

Version -

- Python 0.9.0 - February, 1991
- Python 1.0 - January, 1994
- Python 2.0 - October, 2000
- Python 3.0 - December, 2008
- Python 3.1 - June, 2009
- Python 3.2 - February, 2011
- Python 3.3 - September, 2012
- Python 3.4 - March, 2014



- Python 3.5 September, 2016
- Python 3.6 - December, 2016
- Python 3.7 - June, 2018

- https://www.python.org/doc/versions/

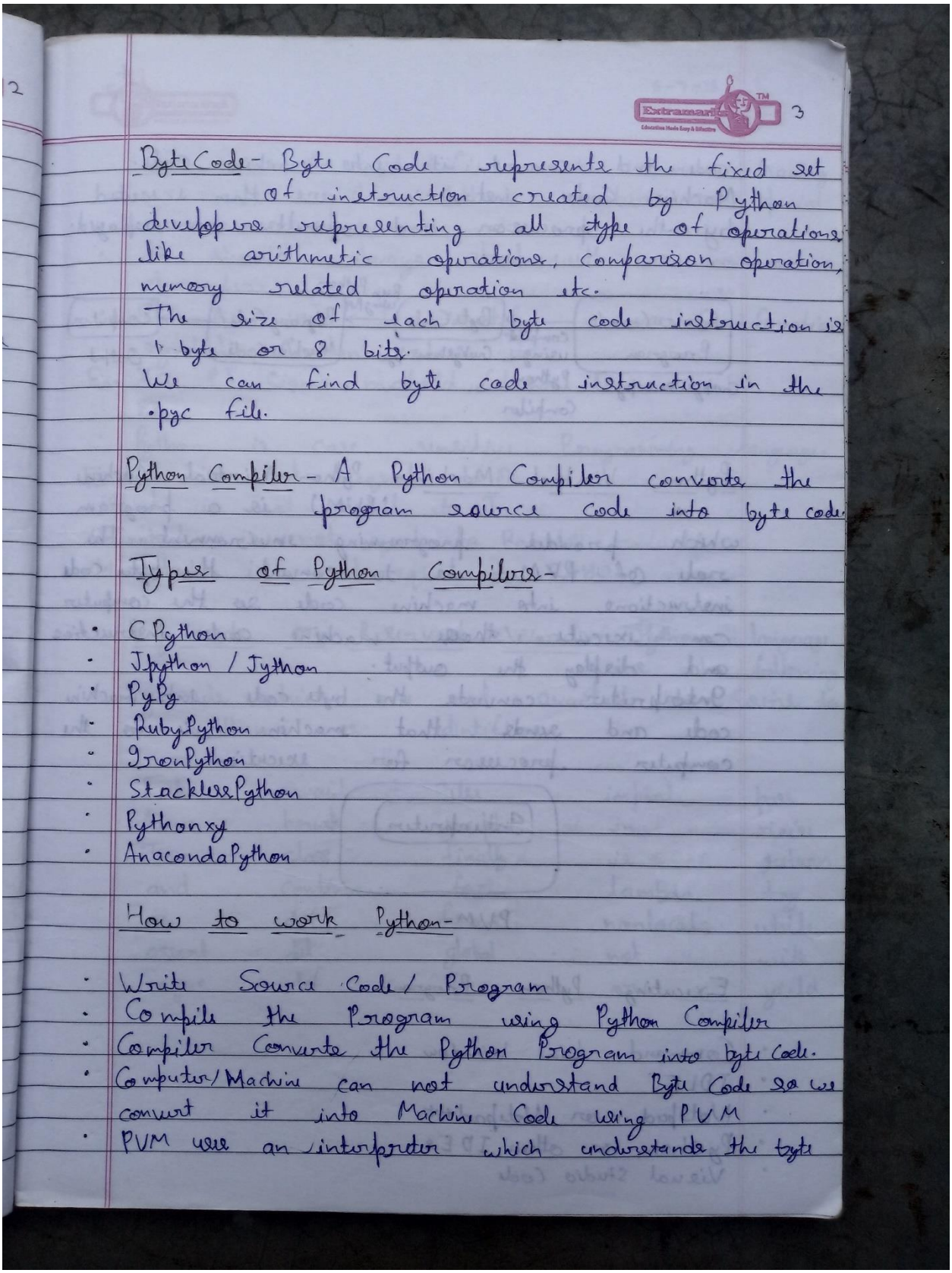
Note- Python 3.5+ cannot be used on Windows XP or earlier.

Features-

- Easy to Learn
- High Level Language
- Interpreted Language
- Platform Independent
- Procedure and Object Oriented
- Huge Library
- Scalable

Application for Python-

- Web Application - Django, Pyramid, Flask, Bottle
- Desktop GUI Application - Tkinter
- Console Based Application
- Games and 3D Application
- Mobile Application
- Scientific and Numeric
- Data Science
- Machine Learning - scikit-learn and TensorFlow
- Data Analysis - Matplotlib, Seaborn
- Business Application



Byte Code - Byte Code represents the fixed set of instruction created by Python developers representing all type of operations like arithmetic operations, comparison operation, memory related operation etc.
The size of each byte code instruction is 1 byte or 8 bits.
We can find byte code instruction in the .pyc file.

Python Compiler - A Python Compiler converts the program source code into byte code.

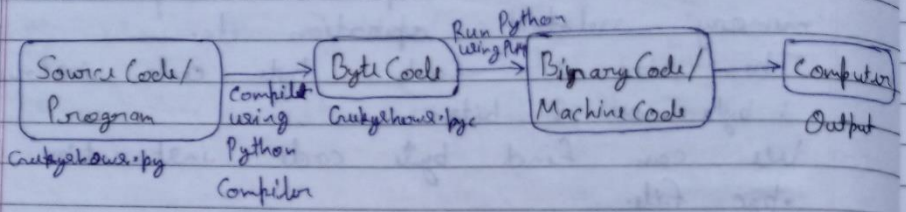
Types of Python Compilers -

- CPython
- Jpython / Jython
- PyPy
- RubyPython
- IronPython
- StacklessPython
- Pythonxy
- AnacondaPython

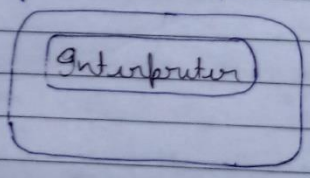
How to work Python -

- Write Source Code / Program
- Compile the Program using Python Compiler
- Compiler Converts the Python Program into byte code.
- Computer/Machine can not understand Byte Code so we convert it into Machine Code using PVM
- PVM use an interpreter which understands the byte

code and convert it into machine code. Machine Code instructions are then executed by the processor and results are displayed.



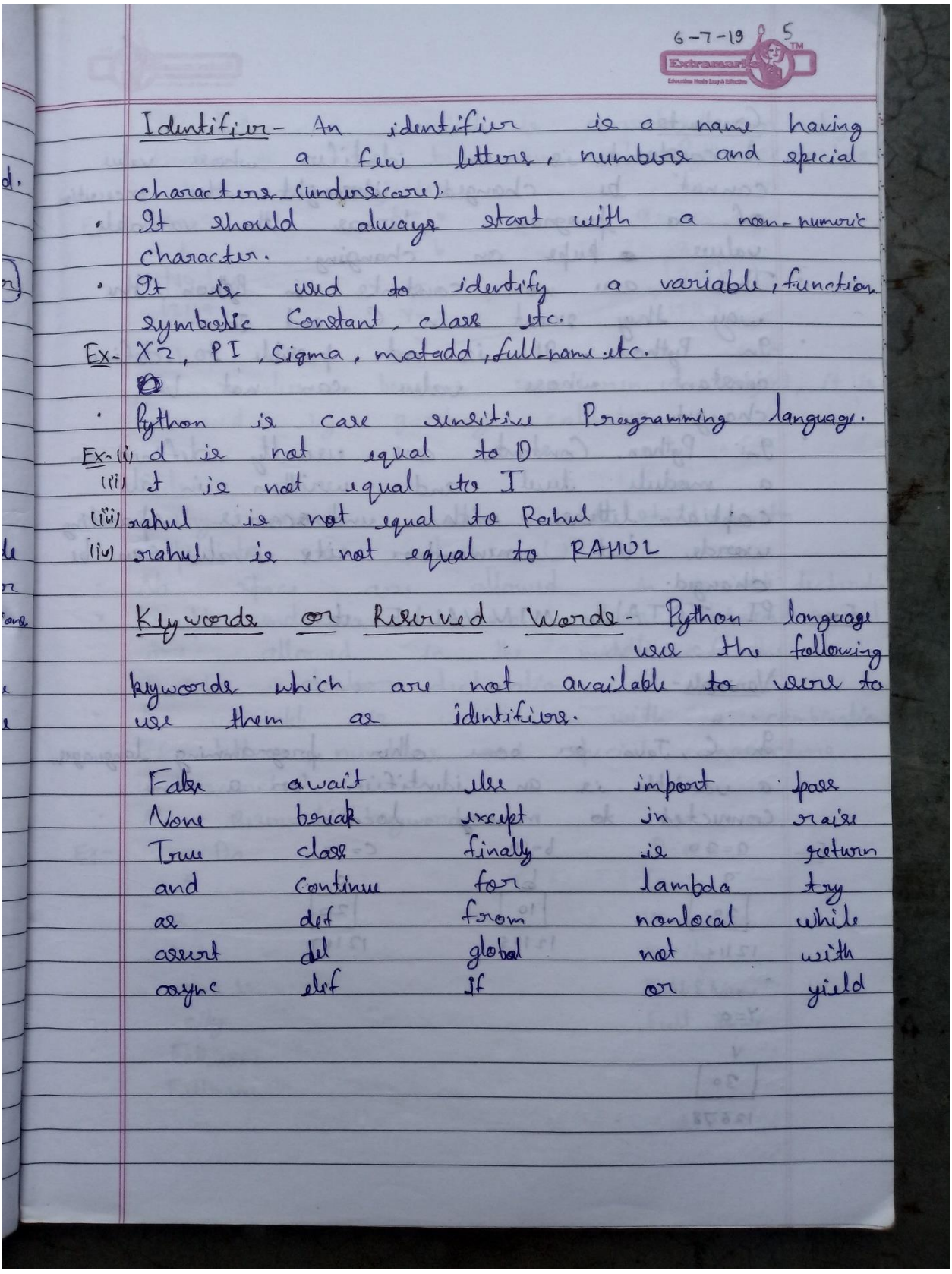
Python Virtual Machine - Python Virtual Machine (PVM) is a program which provides programming environment. The role of PVM is to convert the byte code instructions into machine code so the computer can execute those machine code instructions and display the output.
Interpreter converts the byte code into machine code and sends that machine code to the computer processor for execution.



PVM

Executing Python Program -

- Command Line Window
- IDLE
- Notepad or Notepad++
- PyCharm or other IDEs
- Visual Studio Code



6-7-19 5

Identifier - An identifier is a name having a few letters, numbers and special characters (underscore).

- It should always start with a non-numeric character.
- It is used to identify a variable, function, symbolic Constant, class etc.

Ex- X2, PI, Sigma, matadd, full_name etc.

- Python is case sensitive Programming language.

Ex- (i) d is not equal to D
(ii) t is not equal to T
(iii) rahul is not equal to Rahul
(iv) rahul is not equal to RAHUL

Keywords or Reserved Words - Python language uses the following keywords which are not available to users to use them as identifiers.

False	await	else	import	pass
None	break	except	in	raise
True	class	finally	is	return
and	continue	for	lambda	try
as	def	from	nonlocal	while
assert	del	global	not	with
async	elif	if	or	yield

Constants -

- A constant is an identifier whose value cannot be changed throughout the execution of a program whereas the variable value keeps on changing.
- There are no constants in Python, the way they exist in C and Java.
- In Python, it is not possible to define constant whose value can not be changed.
- In Python, Constants are usually defined on a module level and written in all capital letters with underscores separating words but remember its value can be changed.

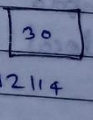
Ex- PI, TOTAL, MIN_VALUE etc.

Variable -

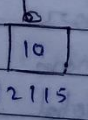
In C, Java or some other programming languages, a variable is an identifier or a name, connected to memory location.

Ex-

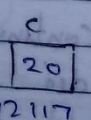
a = 30



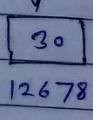
b = 10



c = 20



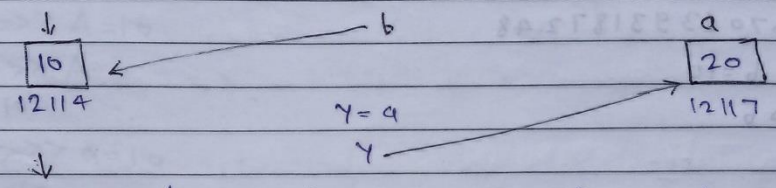
Y = a



9-7-19 7

In Python, a variable is considered as tag that is tied to some value. Python considers value as objects.

Ex- a=10 b=10 a=20



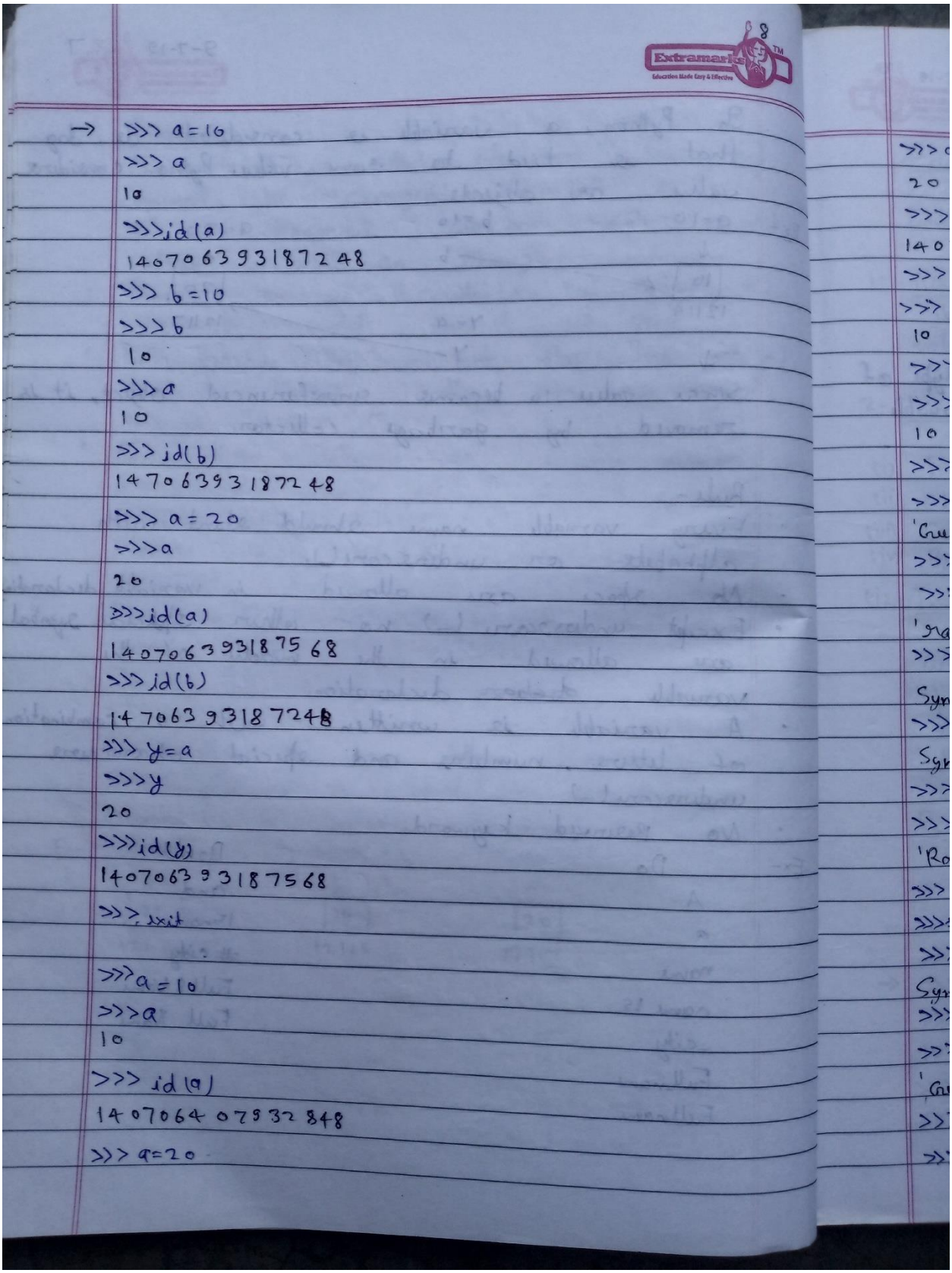
Since value 10 became unreferenced object, it is removed by garbage collector.

Rule -

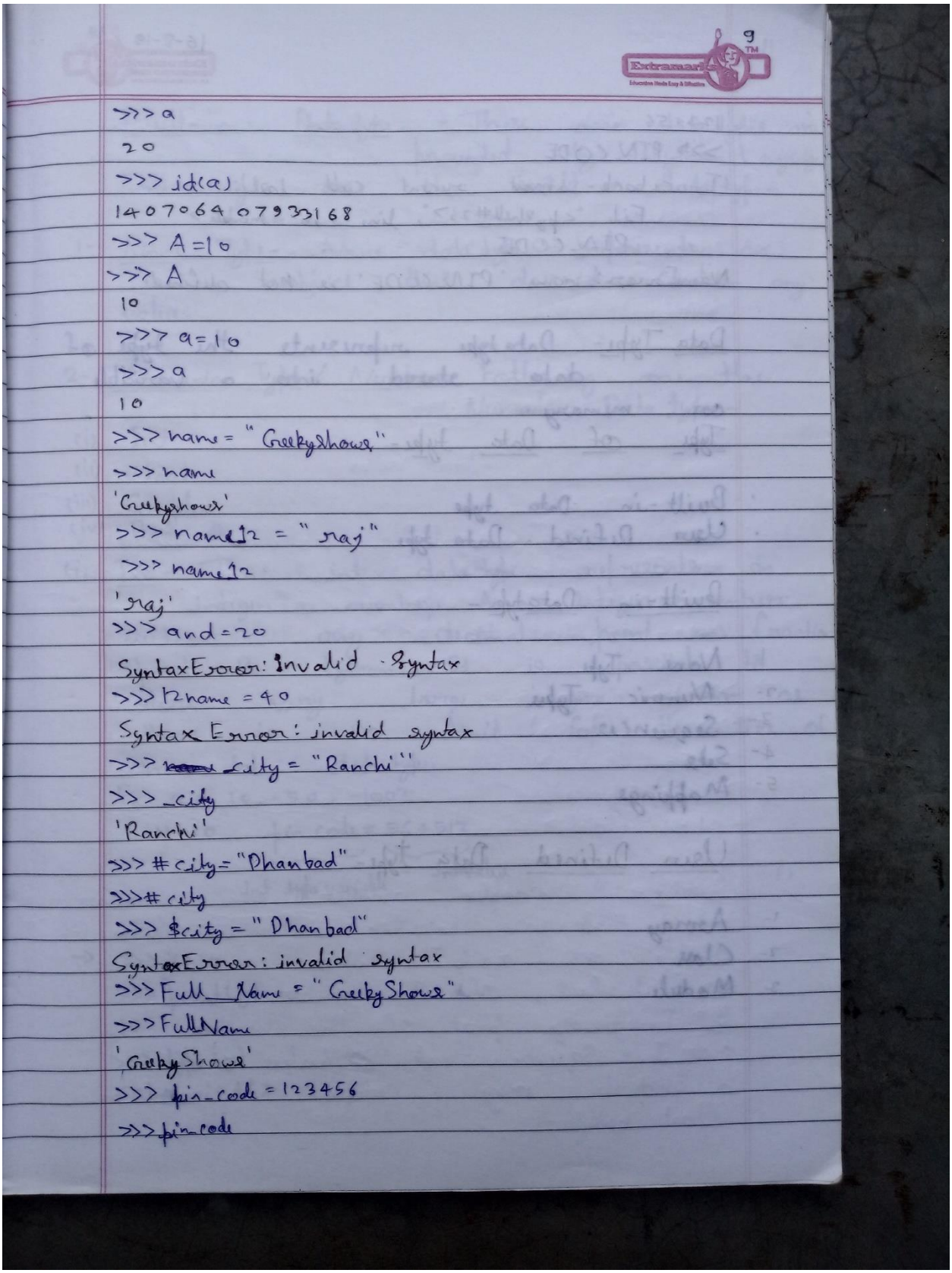
- Every variable name should start with alphabets or underscore(_).
- No space are allowed in variable declaration.
- Except underscore (_) no other special symbol are allowed in the middle of the variable declaration.
- A variable is written with a combination of letter, numbers and special characters underscore(_)
- No Reserved keyword.

Ex-

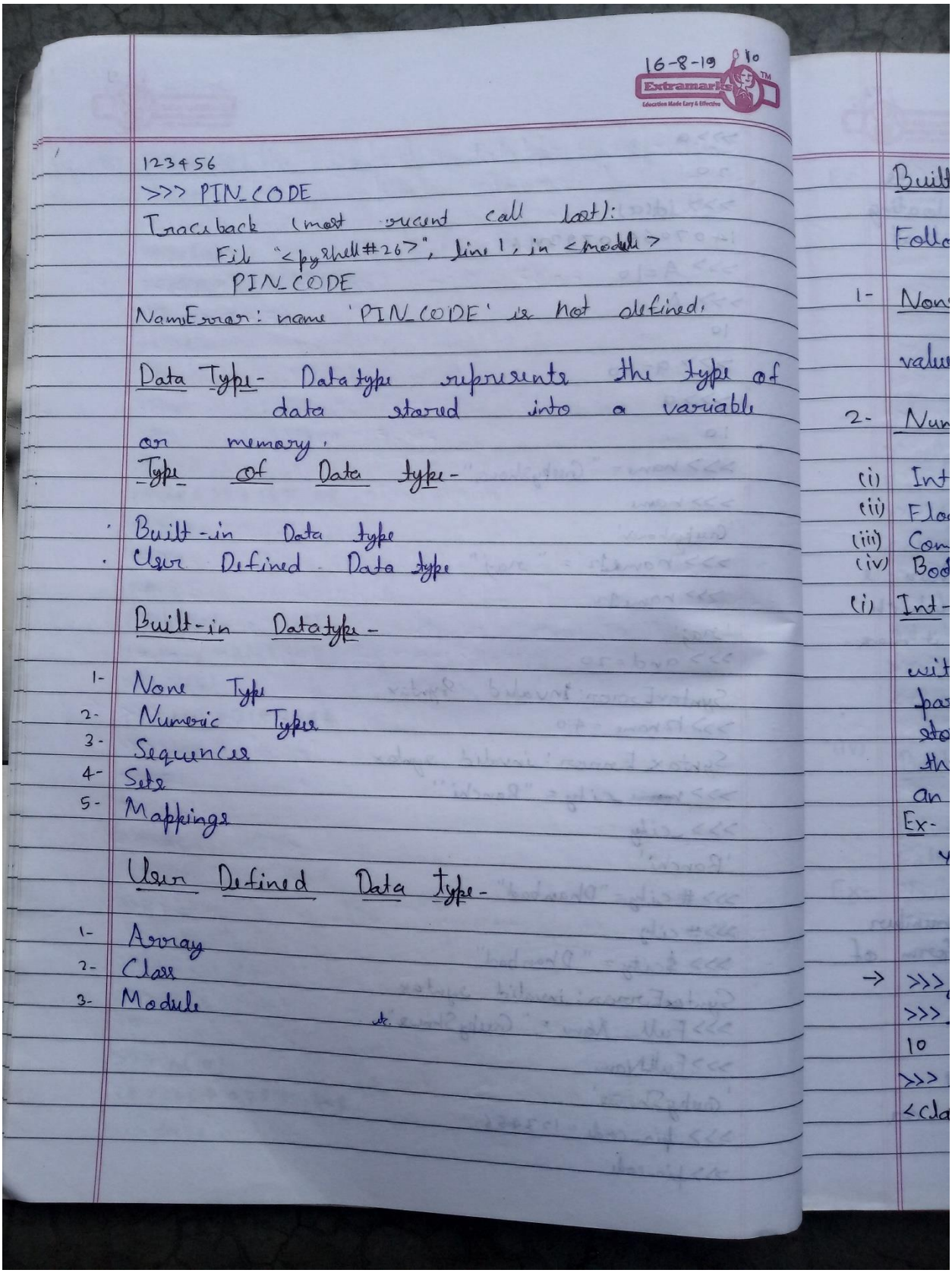
Do	Don't
A	and
a	15name
name	#city
name_15	Full$Name
_city	Full Name
Full_name	
Fullname	



```
>>> a=10
>>> a
10
>>>id(a)
140706393187248
>>> b=10
>>> b
10
>>> a
10
>>> id(b)
140706393187248
>>> a=20
>>>a
20
>>>id(a)
140706393187568
>>>id(b)
140706393187248
>>> y=a
>>>y
20
>>>id(y)
140706393187568
>>>exit

>>>a=10
>>>a
10
>>> id(a)
140706407932848
>>> a=20
```



```
>>> a
20
>>> id(a)
1407064 07933168
>>> A=10
>>> A
10
>>> a=10
>>> a
10
>>> name = "GeekyShows"
>>> name
'GeekyShows'
>>> name12 = "raj"
>>> name12
'raj'
>>> and=20
SyntaxError: invalid syntax
>>> 12name = 40
SyntaxError: invalid syntax
>>> _city = "Ranchi"
>>> _city
'Ranchi'
>>> #city="Dhanbad"
>>>#city
>>> $city = "Dhanbad"
SyntaxError: invalid syntax
>>> Full_Name = "GeekyShows"
>>> FullName
'GeekyShows'
>>> pin_code = 123456
>>> pin_code
```



16-8-19

123456
>>> PIN_CODE
Traceback (most recent call last):
File "<pyshell#26>", line 1, in <module>
PIN_CODE
NameError: name 'PIN_CODE' is not defined.

Data Type- Datatype represents the type of data stored into a variable or memory.

Type of Data type-

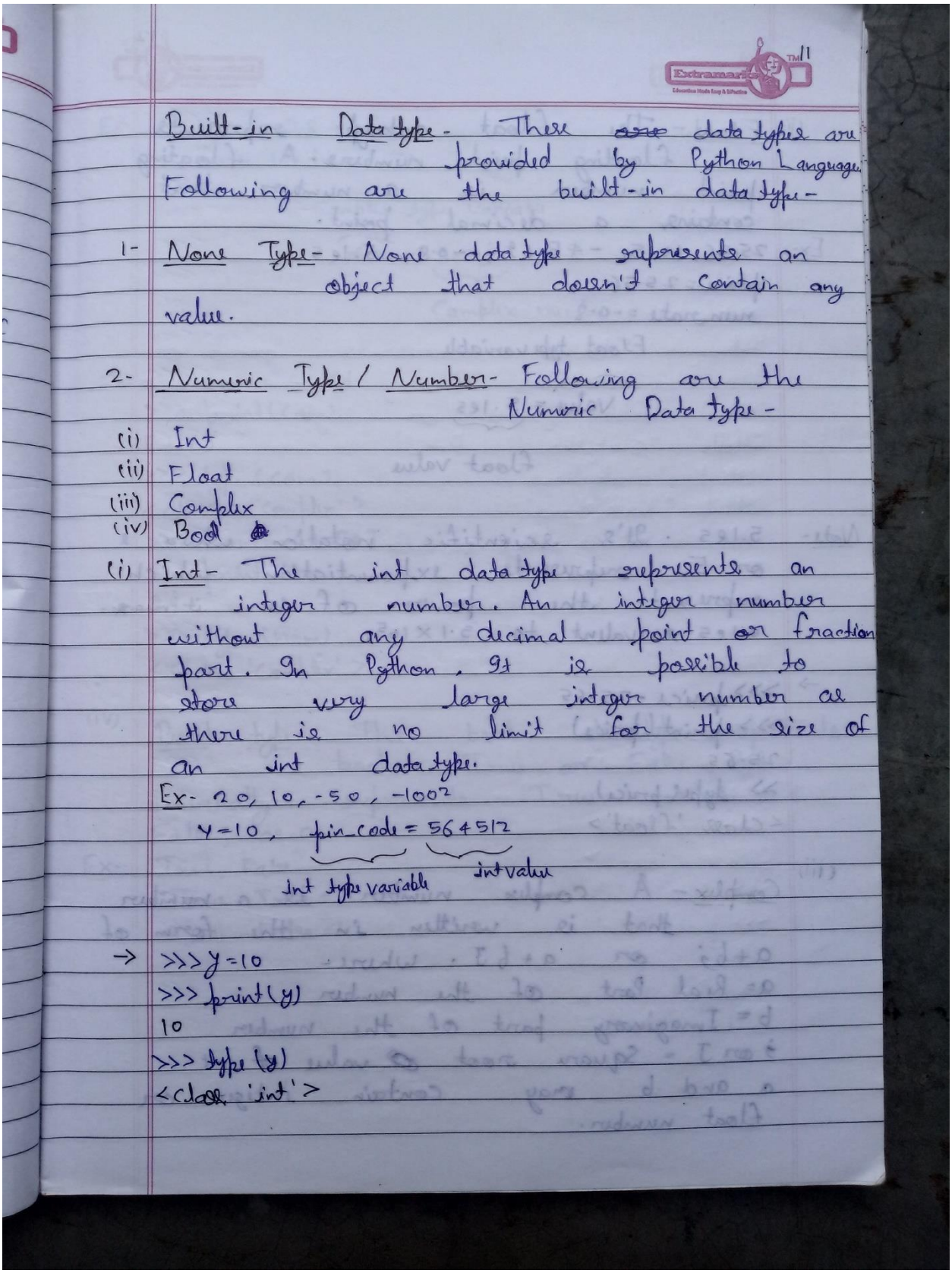
- Built-in Data type
- User Defined Data type

Built-in Datatype-

1- None Type
2- Numeric Types
3- Sequences
4- Sets
5- Mappings

User Defined Data type-

1- Array
2- Class
3- Module



Built-in Data type - These data types are provided by Python Language. Following are the built-in data type -

1- None Type - None data type represents an object that doesn't contain any value.

2- Numeric Type / Number - Following are the Numeric Data type -

(i) Int
(ii) Float
(iii) Complex
(iv) Bool

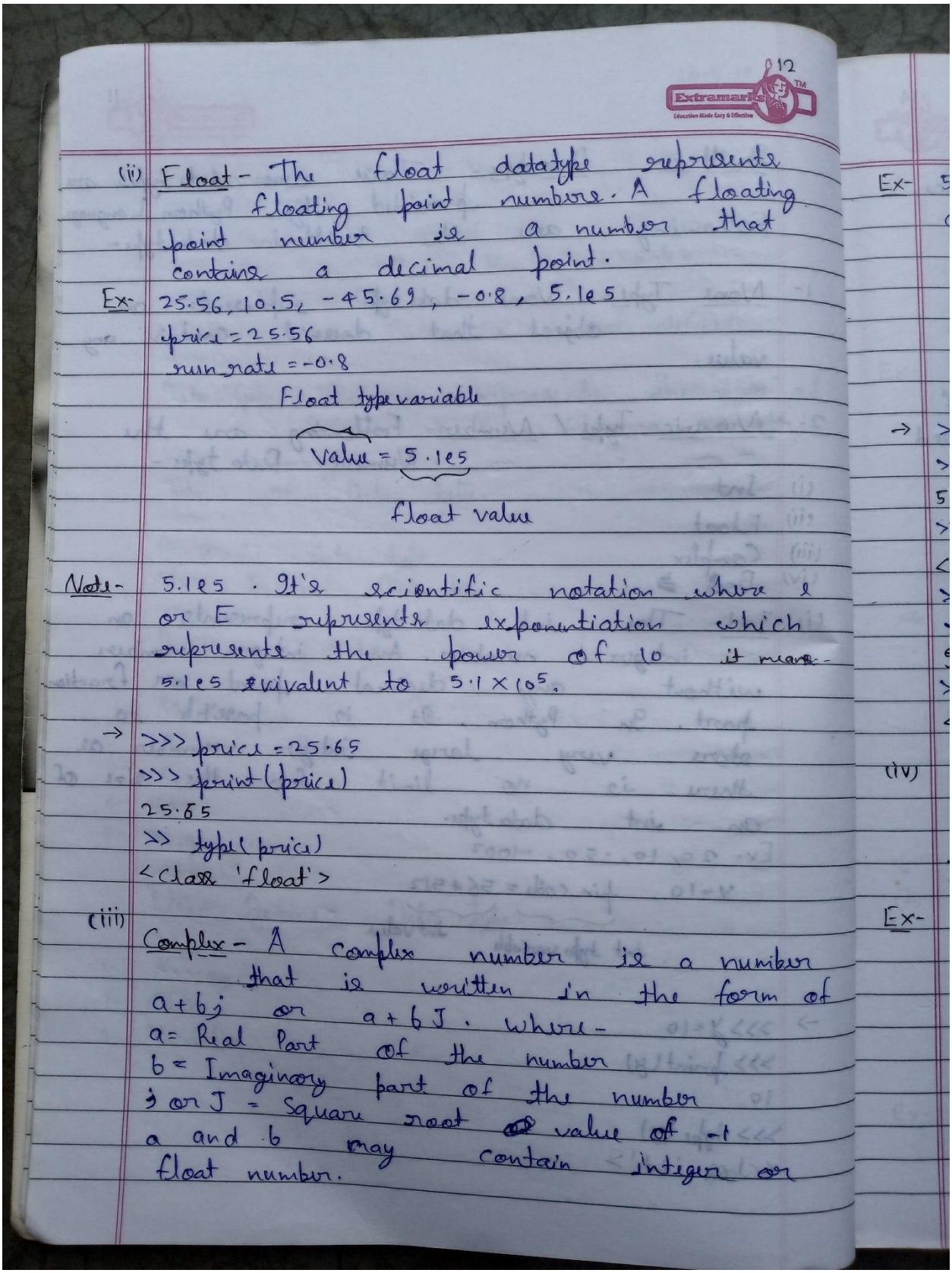
(i) Int - The int data type represents an integer number. An integer number without any decimal point or fraction part. In Python, it is possible to store very large integer number as there is no limit for the size of an int data type.

Ex - 20, 10, -50, -1002

y = 10, pin_code = 564512

int type variable, int value

→
```
>>> y = 10
>>> print(y)
10
>>> type(y)
<class 'int'>
```



(ii) Float - The float datatype represents floating point numbers. A floating point number is a number that contains a decimal point.

Ex- 25.56, 10.5, -45.69, -0.8, 5.1e5

price = 25.56

rain_rate = -0.8

Float type variable

value = 5.1e5

float value

Note- 5.1e5 . It's scientific notation where e or E represents exponentiation which represents the power of 10 it means 5.1e5 equivalent to 5.1×10^5.

→
```
>>> price = 25.65
>>> print(price)
25.65
>> type(price)
<class 'float'>
```

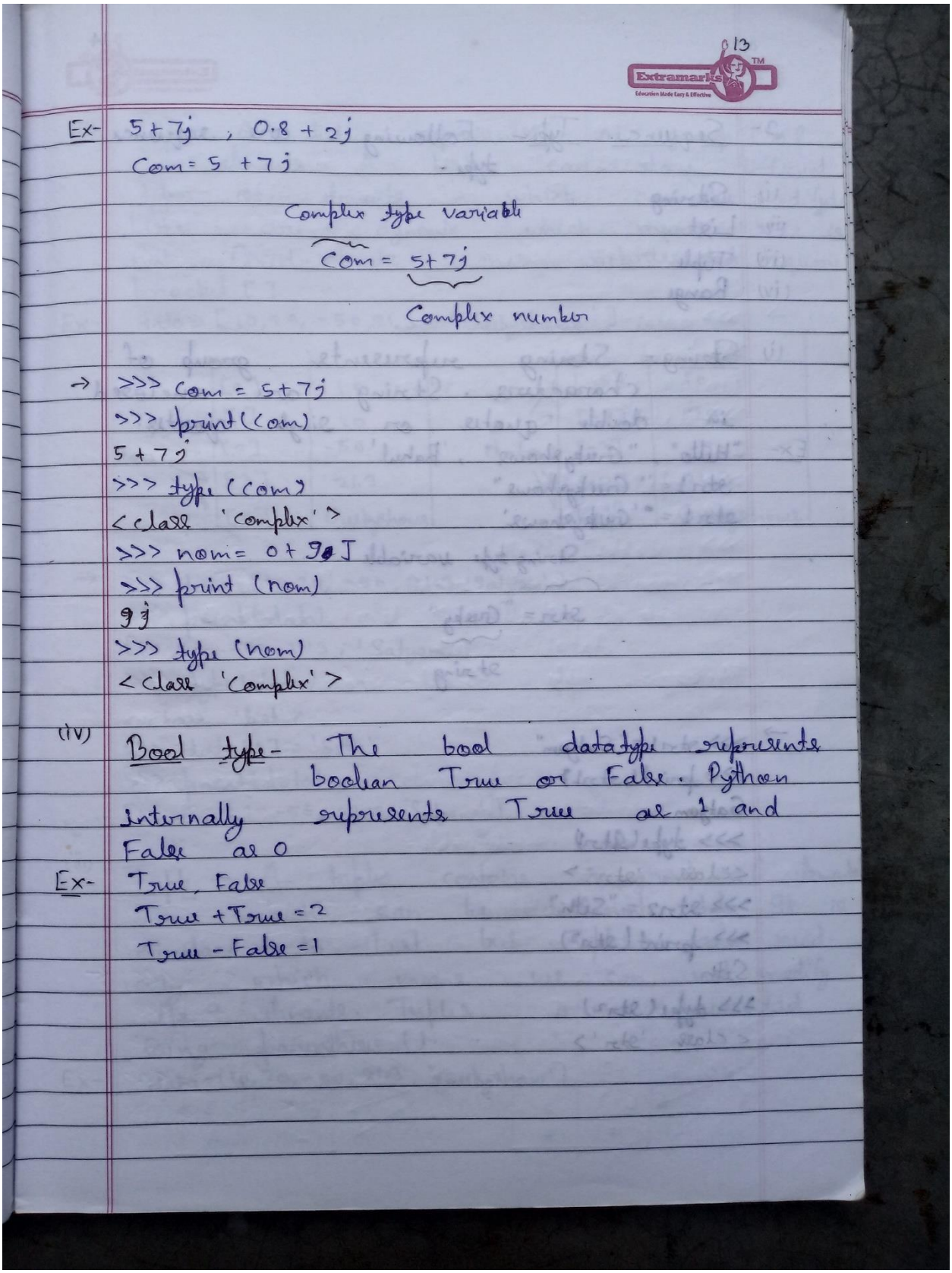
(iii) Complex - A complex number is a number that is written in the form of a+bj or a+bJ. where -

a = Real Part of the number

b = Imaginary part of the number

j or J = Square root of value of -1

a and b may contain integer or float number.



Ex- 5+7j , 0.8 + 2j

com = 5 +7j

Complex type variable

com = 5+7j

Complex number

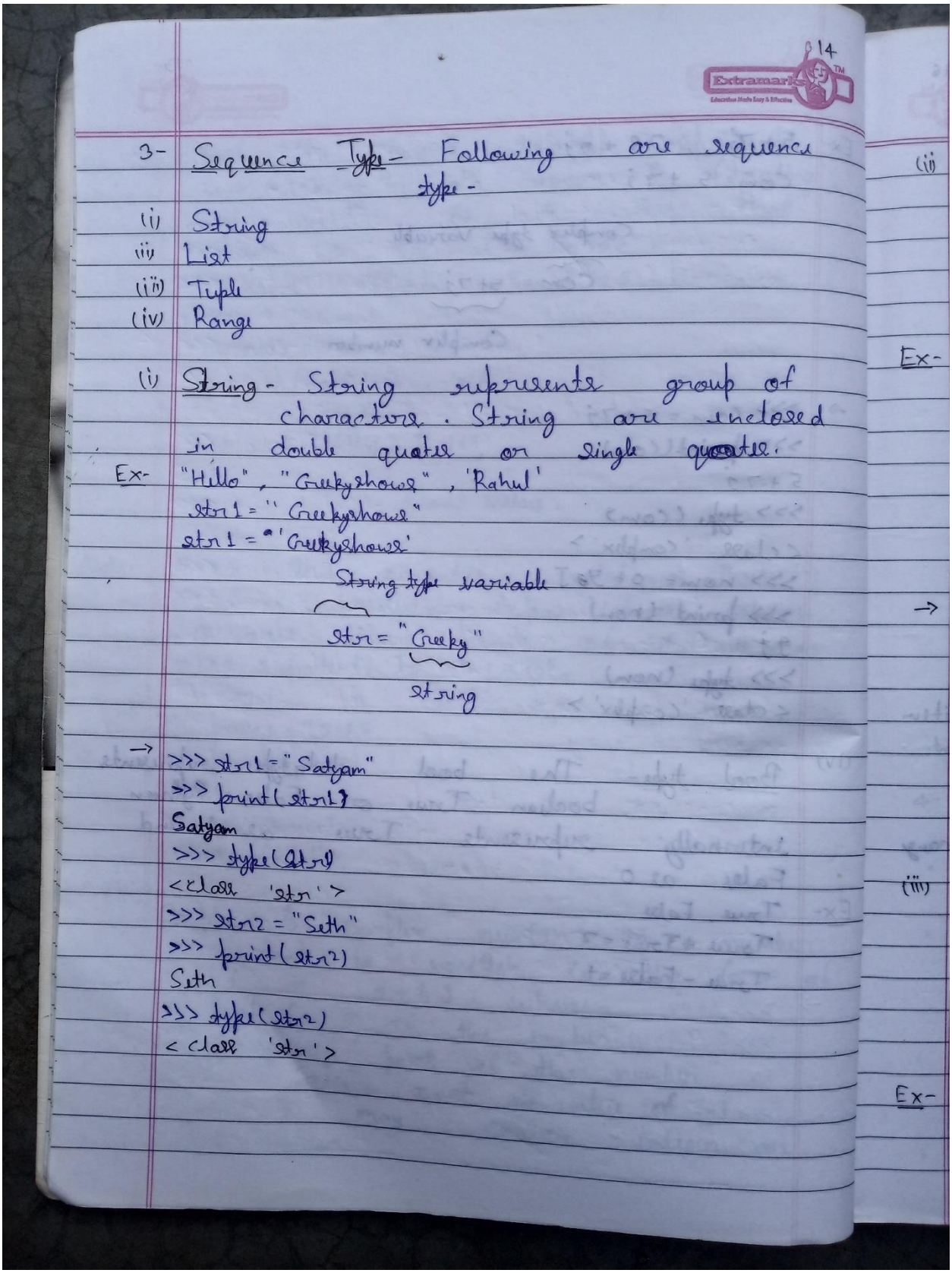
→
```
>>> com = 5+7j
>>> print(com)
5+7j
>>> type(com)
<class 'complex'>
>>> nom= 0+9J
>>> print (nom)
9j
>>> type (nom)
<class 'complex'>
```

(iv) Bool type- The bool datatype represents boolean True or False. Python internally represents True as 1 and False as 0

Ex- True, False

True + True = 2

True - False = 1



14

3- Sequence Type - Following are sequence type -

(i) String
(ii) List
(iii) Tuple
(iv) Range

(i) String - String represents group of characters. String are enclosed in double quotes or single quotes.

Ex- "Hello", "Geekyshows", 'Rahul'
str1 = "Geekyshows"
str1 = 'Geekyshows'

String type variable

str = "Geeky"

string

→
```
>>> str1 = "Satyam"
>>> print(str1)
Satyam
>>> type(str1)
<class 'str'>
>>> str2 = "Seth"
>>> print(str2)
Seth
>>> type(str2)
<class 'str'>
```

(ii)

Ex-

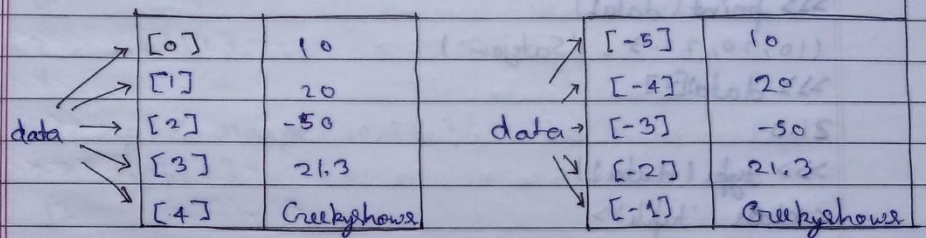
→

(iii)

Ex-

(ii) List - A list represents a group of elements. A list can store different types of elements which can be modified. Lists are dynamic which means size is not fixed. Lists are represented square bracked [ ].

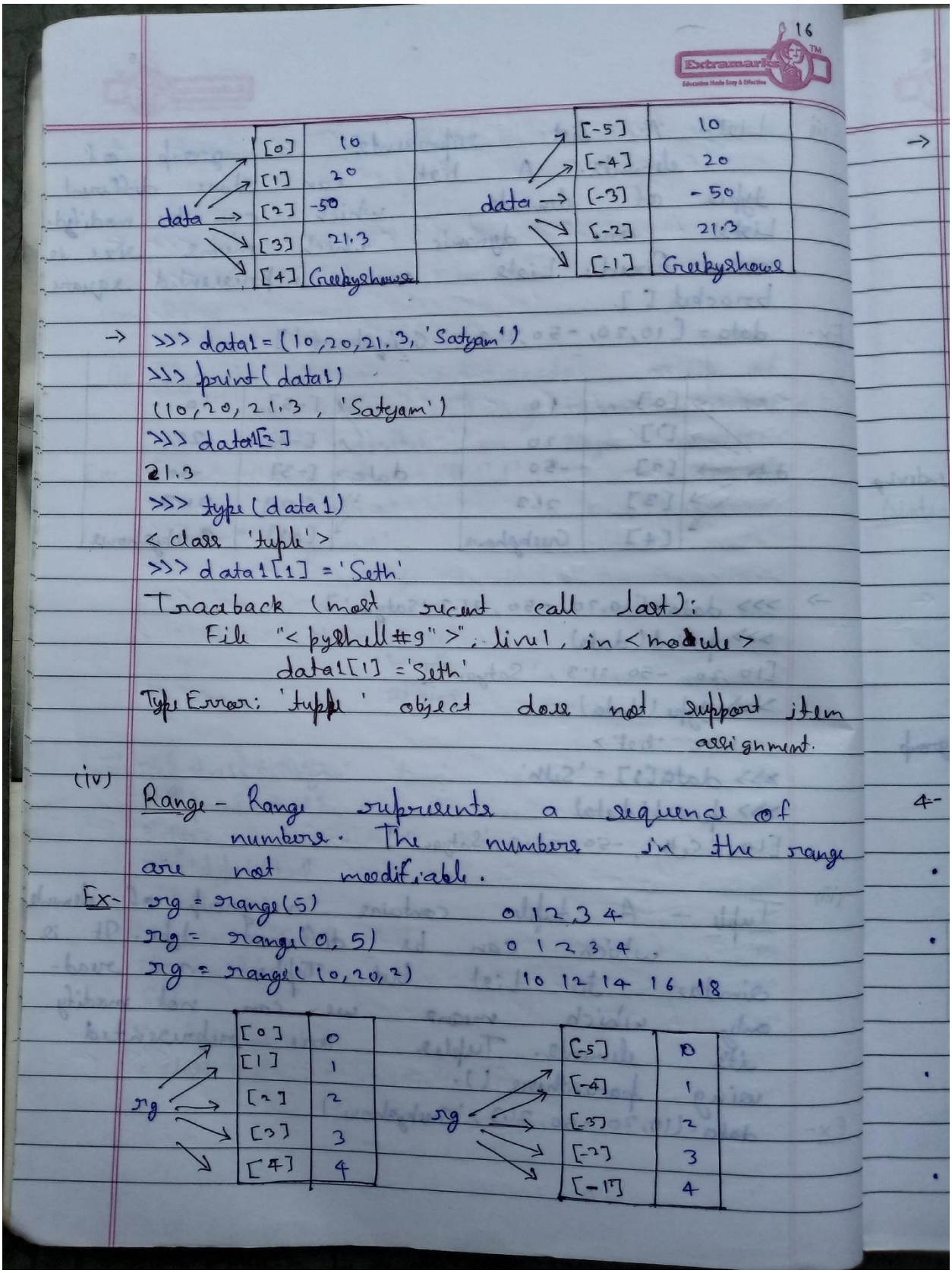
Ex- data = [10, 20, -50, 21.3, 'Geekyshows']

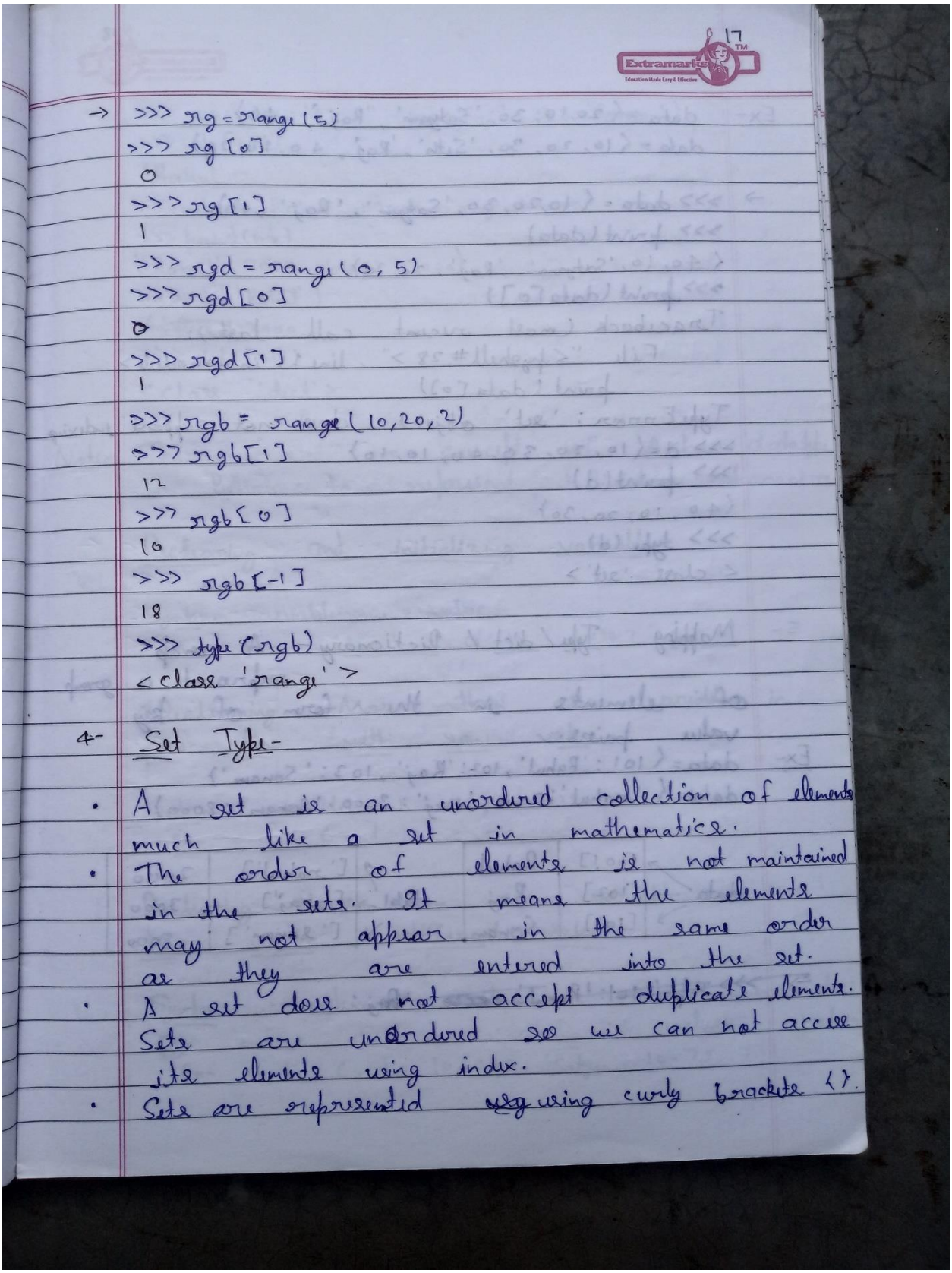


→
```
>>> data=[10,20, -50, 21.3 ,'Satyam']
>>> print(data)
[10 ,20, -50, 21.3 , 'Satyam']
>>> type(data)
<class 'list'>
>>> data[1] = 'Seth'
>>> print(data)
[10, 'Seth', -50, 21.3, 'Satyam']
```

(iii) Tuple - A tuple contains a group of elements which can be different type. It is similar to List but Tuples are read-only which means we can not modify it's elements. Tuples are represented using parenthesis ().

Ex- data = (10, 20, -50, 21.3, 'Geekyshow')


16
Extramarks
data
[0] 10
[1] 20
[2] -50
[3] 21.3
[4] Geekyshows
data
[-5] 10
[-4] 20
[-3] -50
[-2] 21.3
[-1] Geekyshows
→ >>> data1 = (10, 20, 21.3, 'Satyam')
>>> print(data1)
(10, 20, 21.3, 'Satyam')
>>> data1[2]
21.3
>>> type(data1)
<class 'tuple'>
>>> data1[1] = 'Seth'
Traceback (most recent call last):
File "<pyshell#9">, line 1, in <module>
data1[1] = 'Seth'
TypeError: 'tuple' object does not support item assignment.
(iv)
Range - Range represents a sequence of numbers. The numbers in the range are not modifiable.
Ex- rg = range(5) 0 1 2 3 4
rg = range(0, 5) 0 1 2 3 4
rg = range(10, 20, 2) 10 12 14 16 18
rg
[0] 0
[1] 1
[2] 2
[3] 3
[4] 4
rg
[-5] 10
[-4] 1
[-3] 2
[-2] 3
[-1] 4



```
>>> rg = range(5)
>>> rg[0]
0
>>> rg[1]
1
>>> rgd = range(0, 5)
>>> rgd[0]
0
>>> rgd[1]
1
>>> rgb = range(10,20,2)
>>> rgb[1]
12
>>> rgb[0]
10
>>> rgb[-1]
18
>>> type(rgb)
<class 'range'>
```

4- Set Type-

- A set is an unordered collection of elements much like a set in mathematics.
- The order of elements is not maintained in the sets. It means the elements may not appear in the same order as they are entered into the set.
- A set does not accept duplicate elements. Sets are unordered so we can not access its elements using index.
- Sets are represented using curly brackets {}.

Ex- data = {20, 10, 30, 'Satyam', 'Raj', 40}
data = {10, 20, 30, 'Seth', 'Raj', 40, 10, 20}

→
```
>>> data = {10,20,30,'Satyam','Raj',40}
>>> print(data)
{40, 10, 'Satyam', 'Raj', 20, 30}
>>> print(data[0])
Traceback (most recent call last):
  File "<pyshell#28>", line 1, in <module>
    print(data[0])
TypeError: 'set' object does not support indexing
>>> d={10,20,30,40,10,10}
>>> print(d)
{40, 10, 20, 30}
>>> type(d)
<class 'set'>
```

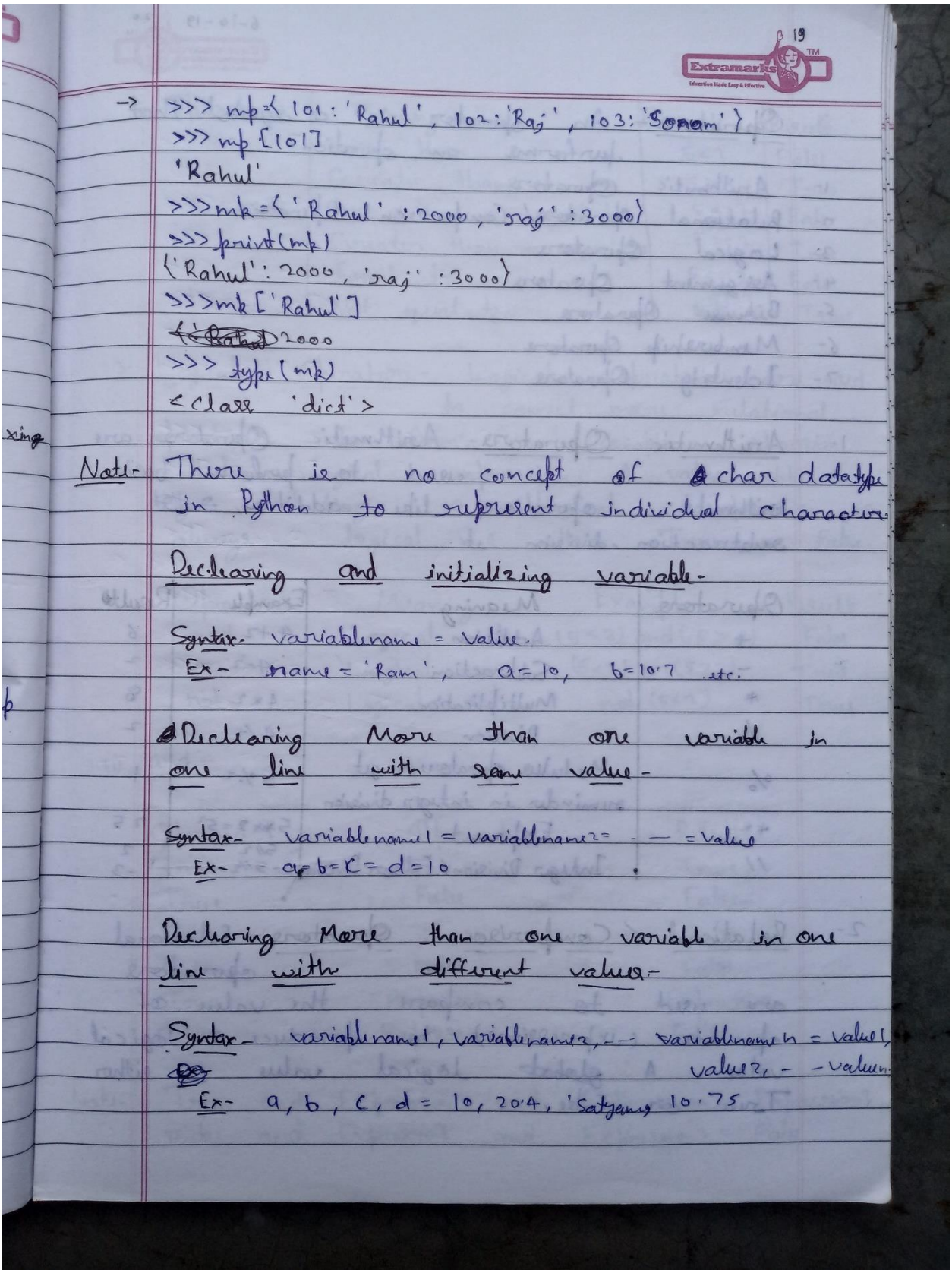
5- Mapping Type / dict / Dictionary - A map represent a group of elements in the form of key value pairs.

Ex- data = {101: 'Rahul', 102: 'Raj', 103: 'Sonam'}
data1 = {'rahul': 2000, 'raj': 3000, 'sonam': 8000}

data →	
[101]	Rahul
[102]	Raj
[103]	Sonam

data1 →	
['rahul']	2000
['raj']	3000
['sonam']	8000

⇒ >>> ~~info = {101: 'Rahul' ..... , 'Raj':~~



19

```
>>> mp={101:'Rahul', 102:'Raj', 103:'Sonam'}
>>> mp[101]
'Rahul'
>>>mk={'Rahul':2000, 'raj':3000}
>>>print(mk)
{'Rahul':2000, 'raj':3000}
>>>mk['Rahul']
2000
>>> type(mk)
<class 'dict'>
```

Note- There is no concept of char datatype in Python to represent individual character.

Declearing and initializing variable-

Syntax- variablename = value.
Ex- name = 'Ram', a=10, b=10.7 etc.

Declearing More than one variable in one line with same value-

Syntax- variablename1 = variablename2 = ... = value
Ex- a=b=c=d=10

Declearing More than one variable in one line with different value-

Syntax- variablename1, variablename2, ... variablenamen = value1, value2, ... valuen
Ex- a, b, c, d = 10, 20.4, 'Satyam', 10.75

6-10-19

Operators - An operator is a symbol that performs an operation.

1- Arithmetic Operators
2- Relational Operators / Comparison Operators
3- Logical Operators
4- Assignment Operators
5- Bitwise Operators
6- Membership Operators
7- Identity Operators.

1- **Arithmetic Operators** - Arithmetic Operators are used to perform basic arithmetic operations like addition, ~~subt~~ subtraction, division etc.

Operators	Meaning	Example	Results
+	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
**	Exponent	$5**2 = 5^2$	25
//	Integer Division / Floor Division	5//2	2
		-5//2	-3

2- **Relational / Comparison Operators** - Relational operators are used to compare the value of operands (expressions) to produce a logical value. A ~~global~~ logical value is either True or False.

Note-

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

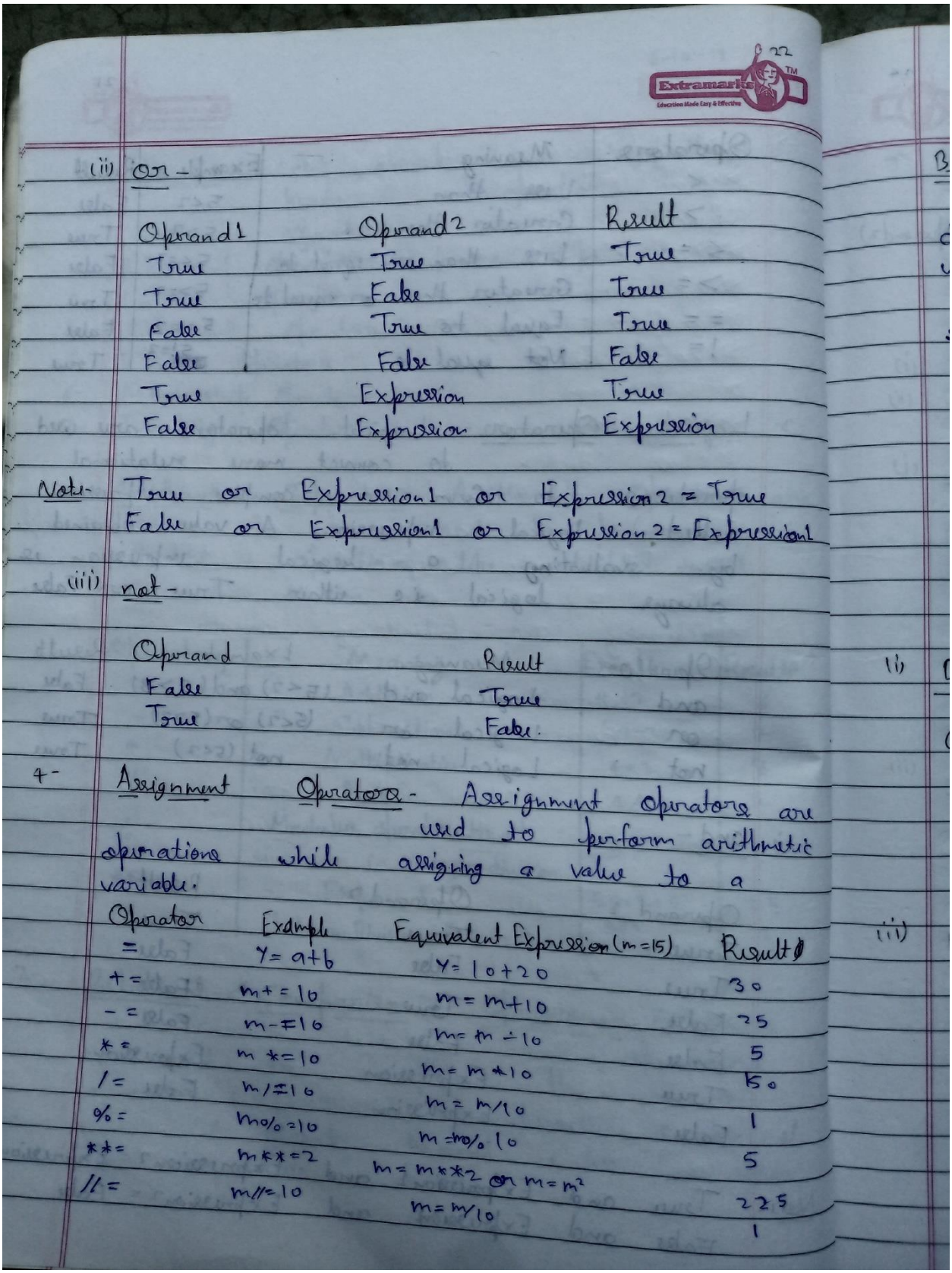
3- Logical Operators - Logical operators are used to connect more relational operations to form a complex expression called logical expression. A value obtained by evaluating a logical expression is always logical i.e either True or False.

Operator	Meaning	Example	Result
and	Logical and	(5<2) and (5>3)	False
or	Logical or	(5<2) or (5>3)	True
not	Logical not	not (5<2)	True

(i) and -

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

Note:- True and Expression1 and Expression2 = Expression2
False and Expression1 and Expression2 = False



(ii) or -

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

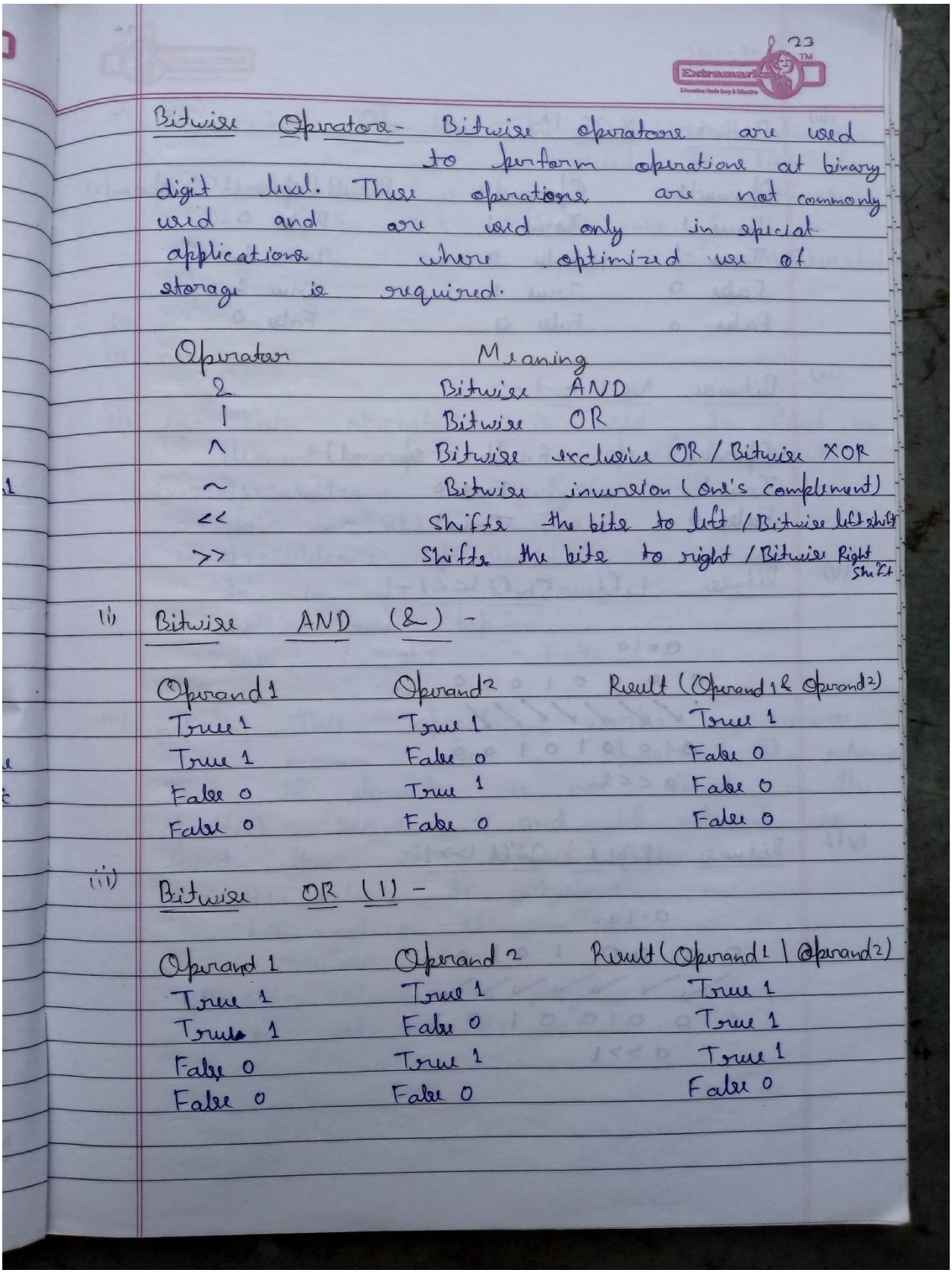
Note- True or Expression1 or Expression2 = True
False or Expression1 or Expression2 = Expression1

(iii) not -

Operand	Result
False	True
True	False

4- Assignment Operators - Assignment operators are used to perform arithmetic operations while assigning a value to a variable.

Operator	Example	Equivalent Expression (m=15)	Result
=	Y = a+b	Y = 10+20	30
+=	m+=10	m = m+10	25
-=	m-=10	m = m-10	5
*=	m*=10	m = m*10	150
/=	m/=10	m = m/10	1
%=	m%=10	m = m%10	5
**=	m**=2	m = m**2 or m = m²	225
//=	m//=10	m = m//10	1



Bitwise Operators - Bitwise operators are used to perform operations at binary digit level. These operations are not commonly used and are used only in special applications where optimized use of storage is required.

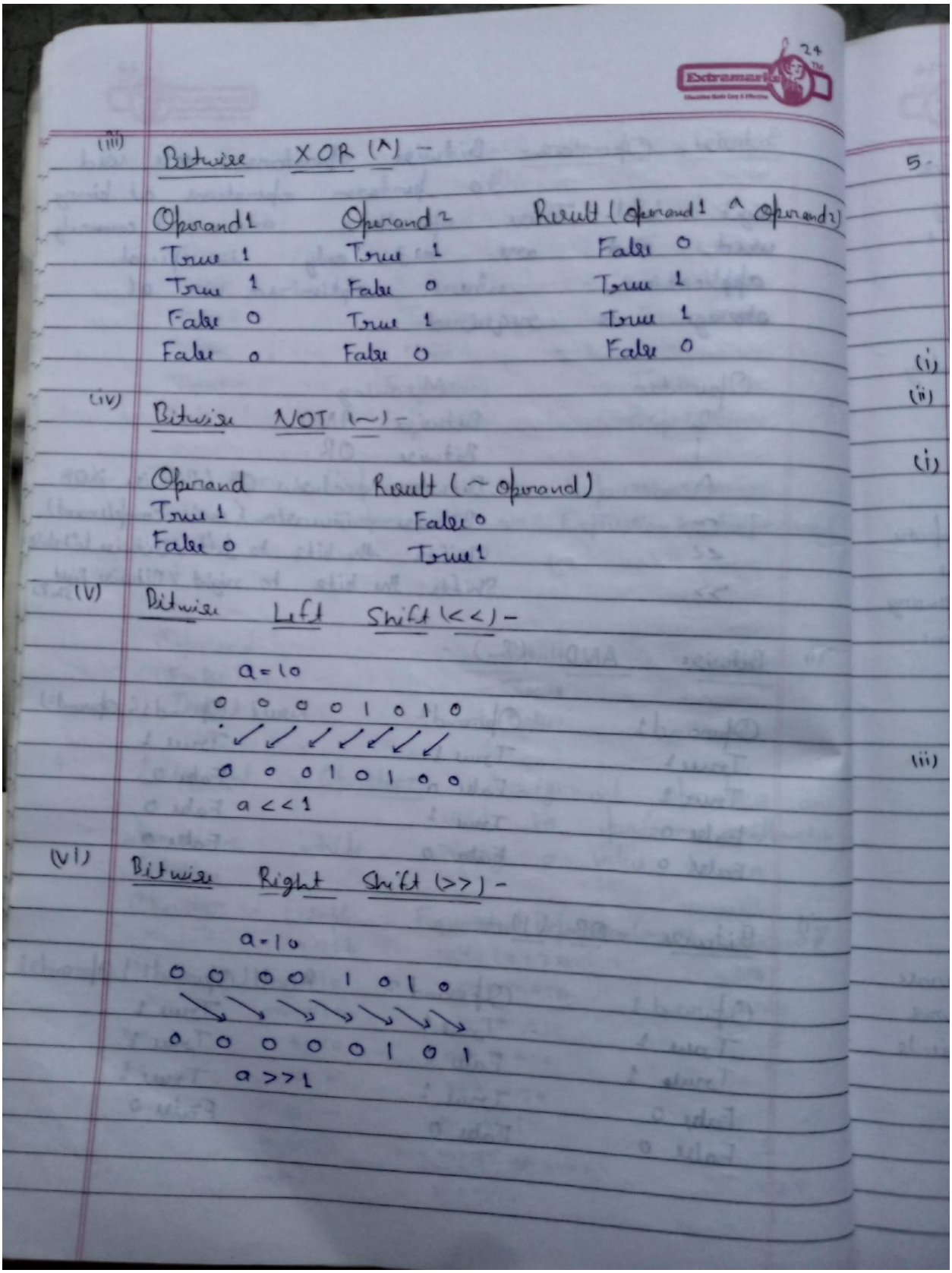
Operator	Meaning
&	Bitwise AND
|	Bitwise OR
^	Bitwise exclusive OR / Bitwise XOR
~	Bitwise inversion (One's complement)
<<	Shifts the bits to left / Bitwise left shift
>>	Shifts the bits to right / Bitwise Right Shift

ii) Bitwise AND (&) -

Operand 1	Operand 2	Result (Operand 1 & Operand 2)
True 1	True 1	True 1
True 1	False 0	False 0
False 0	True 1	False 0
False 0	False 0	False 0

iii) Bitwise OR (|) -

Operand 1	Operand 2	Result (Operand 1 | Operand 2)
True 1	True 1	True 1
True 1	False 0	True 1
False 0	True 1	True 1
False 0	False 0	False 0



(iii) Bitwise XOR (^) -

Operand1	Operand2	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

(iv) Bitwise NOT (~) -

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

(v) Bitwise Left Shift (<<) -

a = 10

0 0 0 0 1 0 1 0

0 0 0 1 0 1 0 0

a << 1

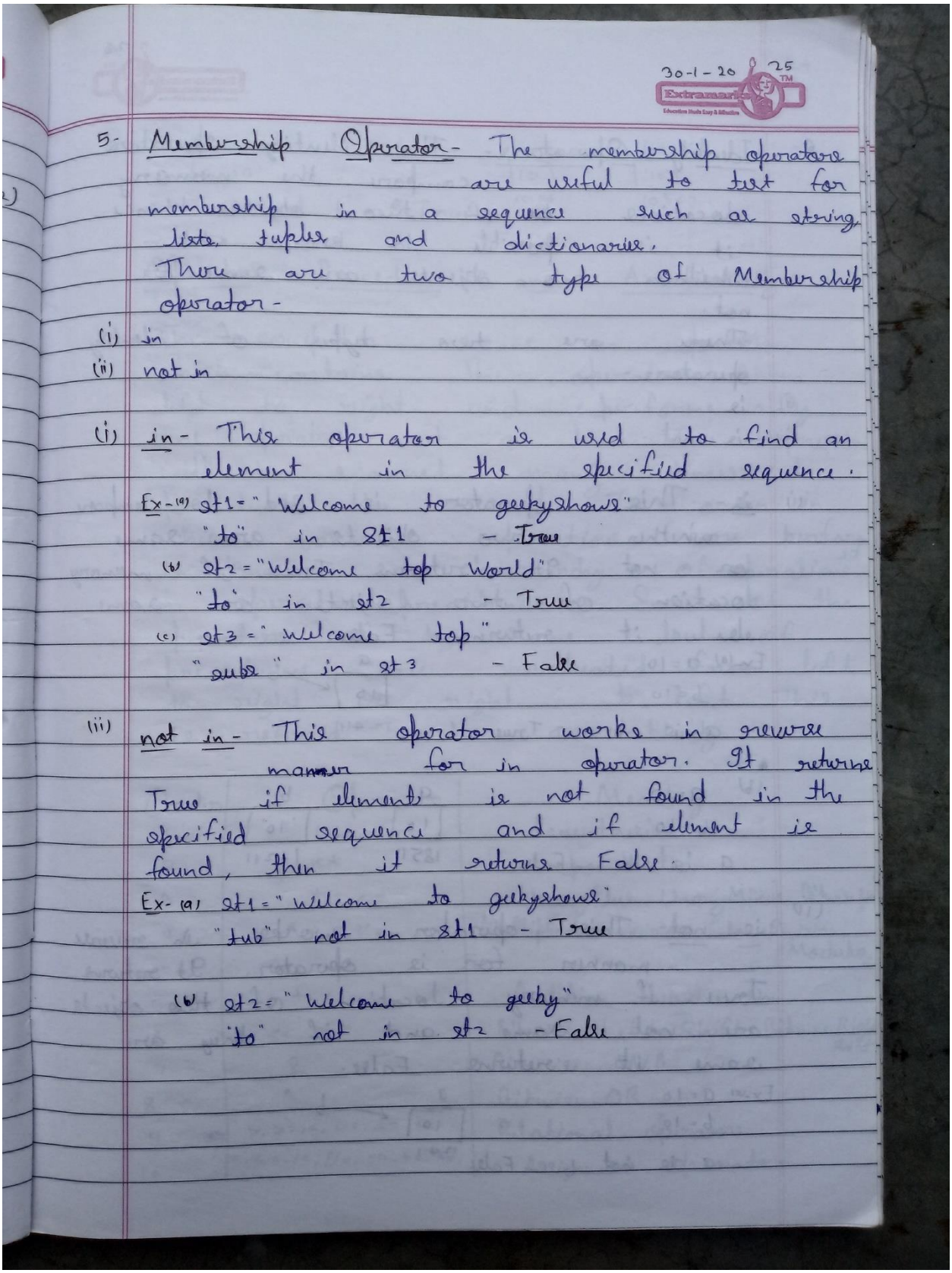
(vi) Bitwise Right Shift (>>) -

a = 10

0 0 0 0 1 0 1 0

0 0 0 0 0 1 0 1

a >> 1



30-1-2025

5- Membership Operator - The membership operators are useful to test for membership in a sequence such as string, lists, tuples and dictionaries.
There are two type of Membership operator -

(i) in
(ii) not in

(i) in - This operator is used to find an element in the specified sequence.

Ex - (a) st1 = "Welcome to geekyshows"
"to" in st1 - True

(b) st2 = "Welcome top World"
"to" in st2 - True

(c) st3 = "welcome top"
"sub" in st3 - False

(ii) not in - This operator works in reverse manner for in operator. It returns True if element is not found in the specified sequence and if element is found, then it returns False.

Ex - (a) st1 = "welcome to geekyshows"
"tub" not in st1 - True

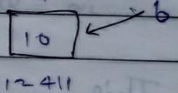
(b) st2 = "Welcome to geeky"
"to" not in st2 - False

6- Identity Operators- The identity operators compare the memory locations of two objects. Hence, it is possible to know whether two objects are same or not.
There are two types of identity operators -

(i) is
(ii) is not

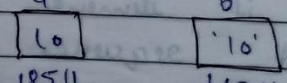
(i) is - This operator is used to compare whether two objects are same or not. It returns True if memory location of two objects are same else it returns False.

Ex-(a) a = 10
b = 10



a is b - True

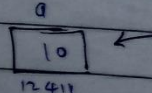
(b) a = 10
b = '10'



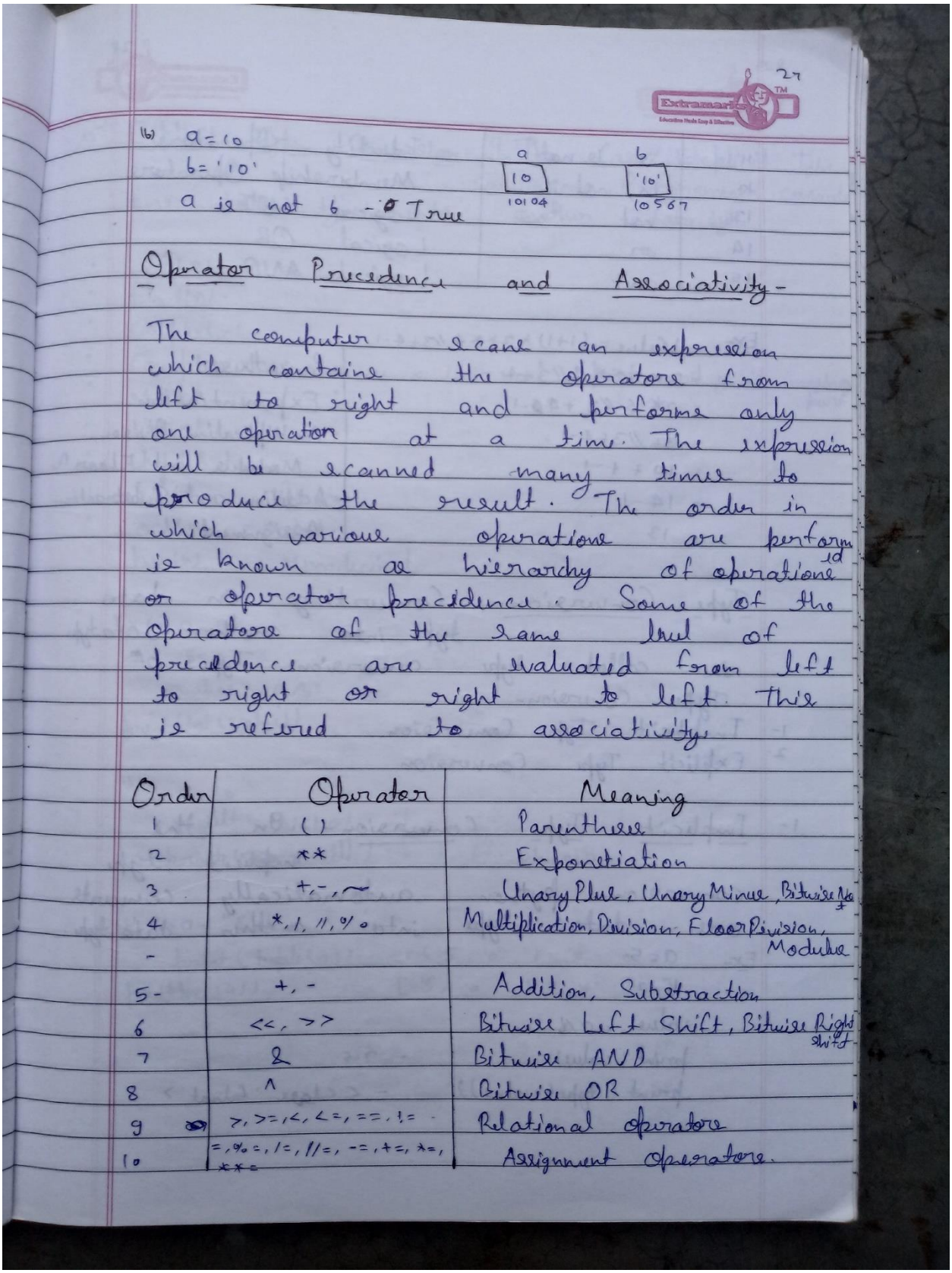
a is b - False

(ii) is not - This operator works in reverse manner for is operator. It returns True if memory location of two objects are not same and if they are same it returns False.

Ex-(a) a = 10
b = 10



a is not b - False

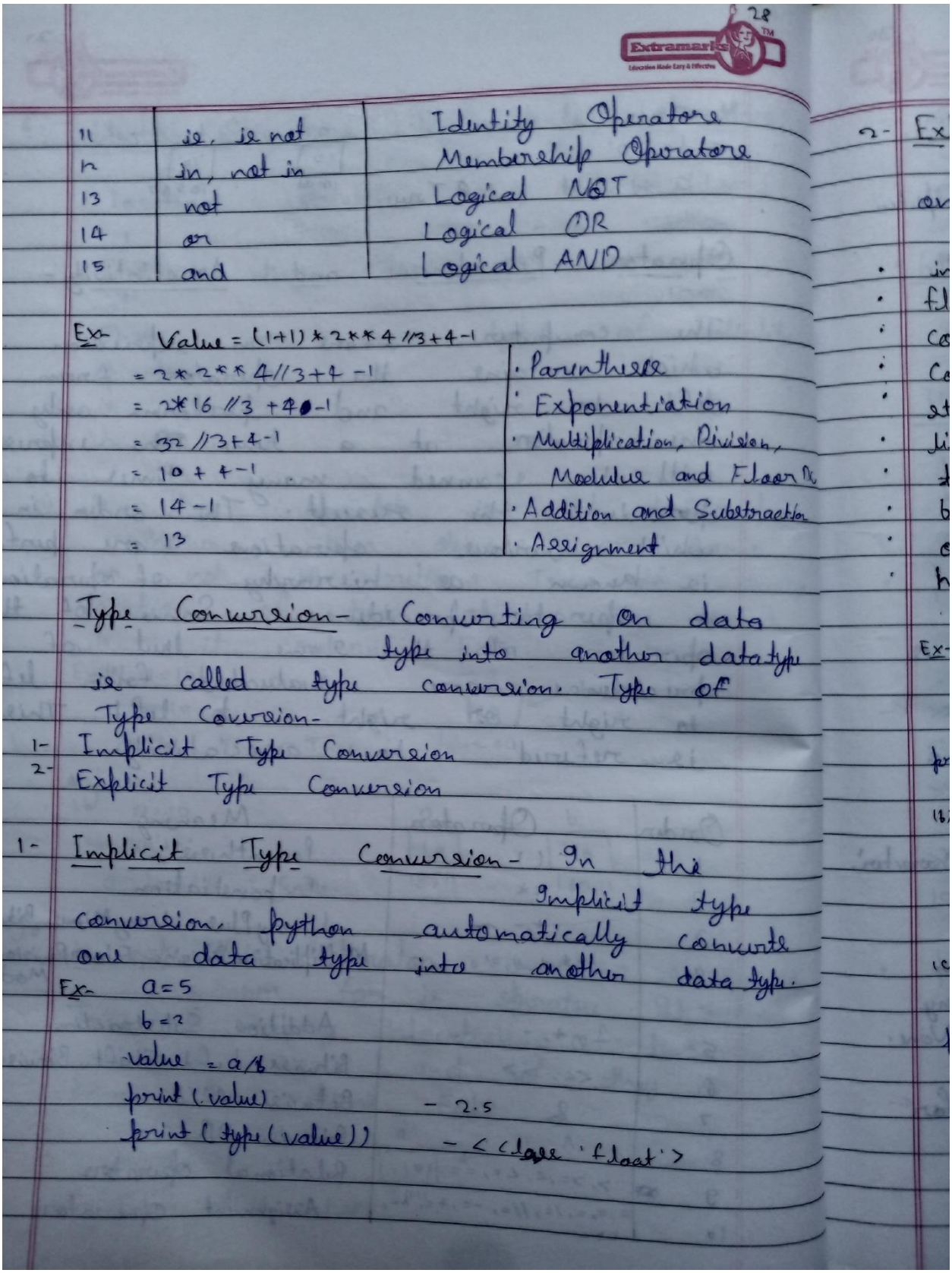


(b) a = 10
b = '10'
a is not b → True

Operator Precedence and Associativity -

The computer scans an expression which contains the operators from left to right and performs only one operation at a time. The expression will be scanned many times to produce the result. The order in which various operations are performed is known as hierarchy of operations or operator precedence. Some of the operators of the same level of precedence are evaluated from left to right or right to left. This is referred to associativity.

Order	Operator	Meaning
1	()	Parenthesis
2	**	Exponentiation
3	+, -, ~	Unary Plus, Unary Minus, Bitwise Not
4	*, /, //, %	Multiplication, Division, Floor Division, Modulus
5	+, -	Addition, Subtraction
6	<<, >>	Bitwise Left Shift, Bitwise Right Shift
7	&	Bitwise AND
8	^	Bitwise OR
9	>, >=, <, <=, ==, !=	Relational operators
10	=, %=, /=, //=, -=, +=, *=, **=	Assignment operators



11	is, is not	Identity Operators
12	in, not in	Membership Operators
13	not	Logical NOT
14	or	Logical OR
15	and	Logical AND

Ex- Value = (1+1) * 2 ** 4 // 3 + 4 - 1

= 2 * 2 ** 4 // 3 + 4 - 1
= 2 * 16 // 3 + 4 - 1
= 32 // 3 + 4 - 1
= 10 + 4 - 1
= 14 - 1
= 13

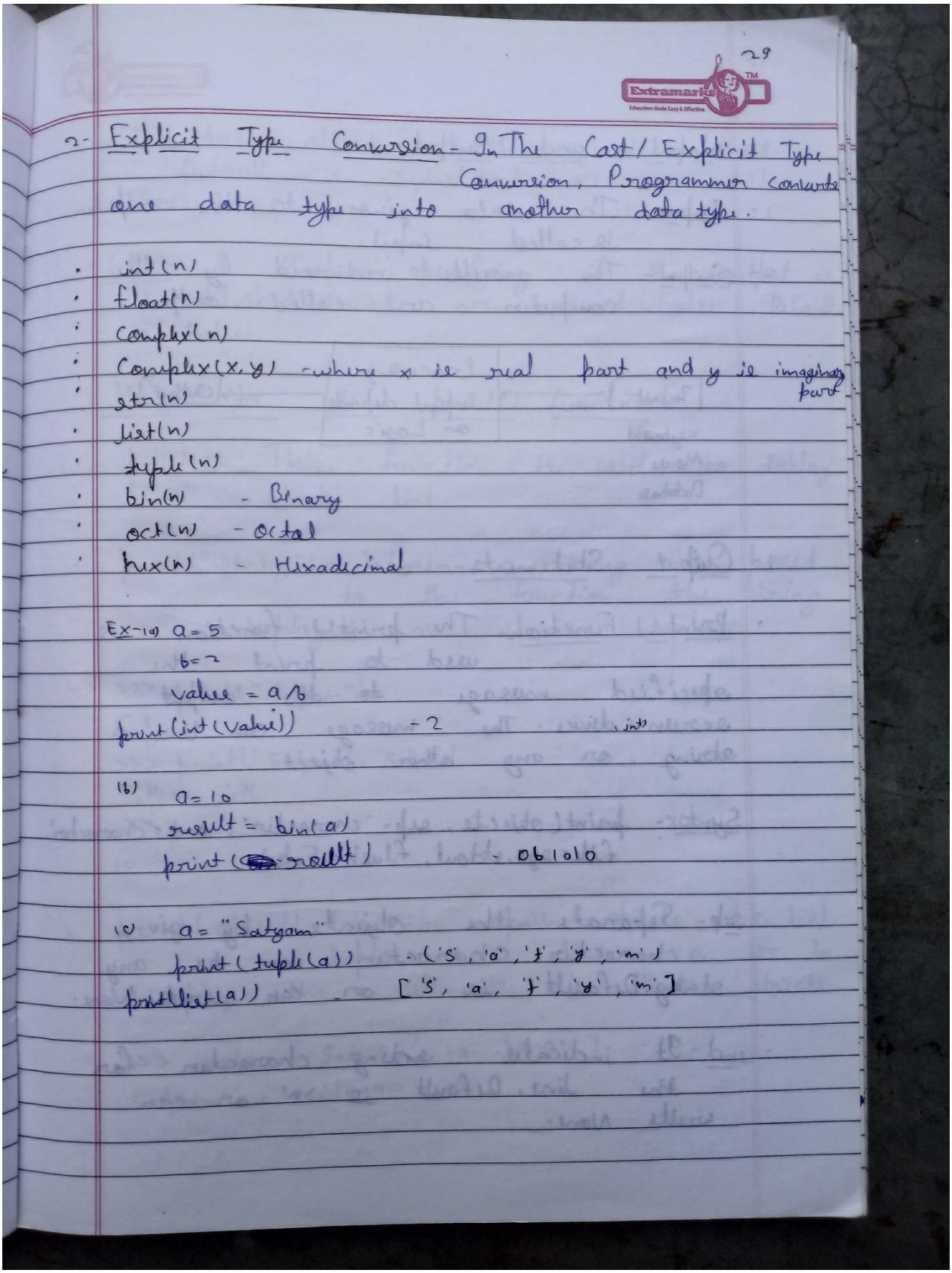
- Parenthesis
- Exponentiation
- Multiplication, Division, Modulus and Floor
- Addition and Substraction
- Assignment

Type Conversion- Converting on data type into another datatype is called type conversion. Type of Type Conversion-

1- Implicit Type Conversion
2- Explicit Type Conversion

1- Implicit Type Conversion- In this Implicit type conversion, python automatically converts one data type into another data type.

Ex- a = 5
b = 2
value = a / b
print (value) - 2.5
print (type (value)) - <class 'float'>



29

2- Explicit Type Conversion - In The Cast / Explicit Type Conversion, Programmer converts one data type into another data type.

- int(n)
- float(n)
- complex(n)
- complex(x,y) - where x is real part and y is imaginary part
- str(n)
- list(n)
- tuple(n)
- bin(n) - Binary
- oct(n) - Octal
- hex(n) - Hexadecimal

Ex-1(a) a = 5
b = 2
value = a/b
print(int(value)) - 2 int

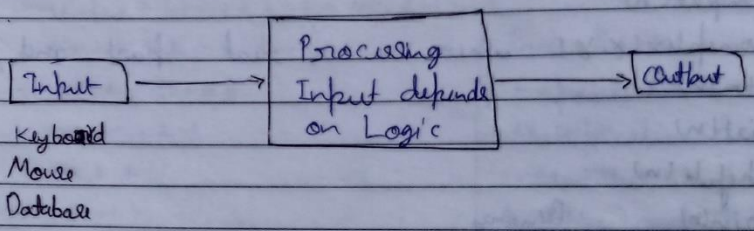
(b) a = 10
result = bin(a)
print(result) - 0b1010

(c) a = "Satyam"
print(tuple(a)) - ('S', 'a', 't', 'y', 'm')
print(list(a)) - ['S', 'a', 't', 'y', 'm']

Input and Output -

1- **Input** - The data given to the computer is called input.

2- **Output** - The results returned by the computer are called output.



1- Output Statements -

- **Print() Function** - The print() function is used to print the specified message to the output screen / device. The message can be a string, or any other object.

Syntax - print(object, sep='character', end='character', file=sys.stdout, flush=False)

- **sep** - Separate the objects by given character. character can be any string. Default is ' ' or can write None.

- **end** - It indicates ending character for the line. Default is '\n' or can write None.

- file - An object with a write method. Default is sys.stdout or can write None.

- flush - A Boolean, specifying if the output is flushed (True) or buffered (False). Default is False.

How to use print() function -

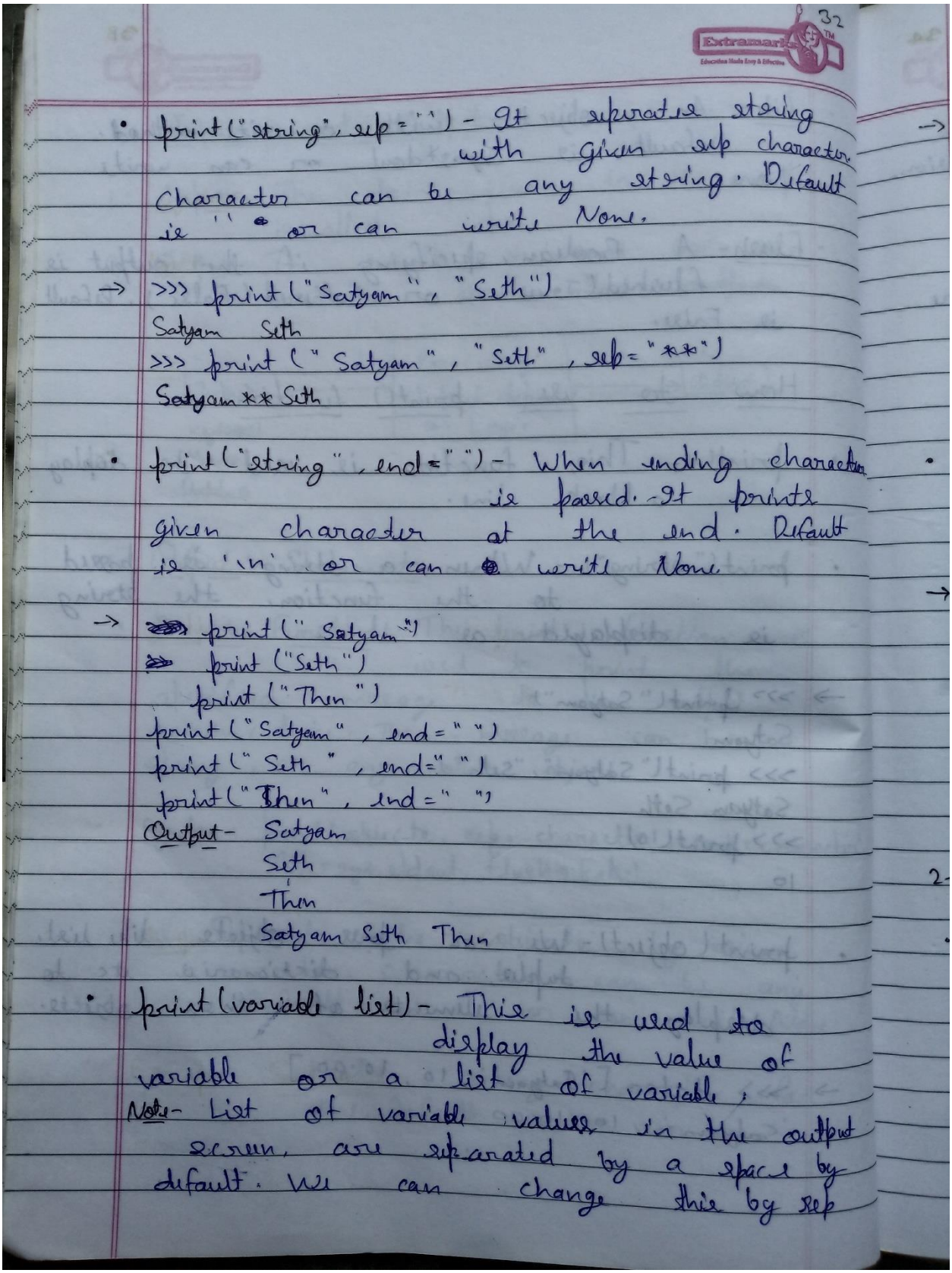
- print() - This function is used to display a blank line.

- print("string") - When a string is passed to the function, the string is displayed as it is.

```
>>> print("Satyam")
Satyam
>>> print("Satyam", "Seth")
Satyam Seth
>>> print(10)
10
```

- print(object) - We can pass objects like list, tuple, and dictionaries etc to display the elements of those objects.

```
>>> data = ['Satyam', 10, 10.00]
'Satyam', 10, 10.00
```



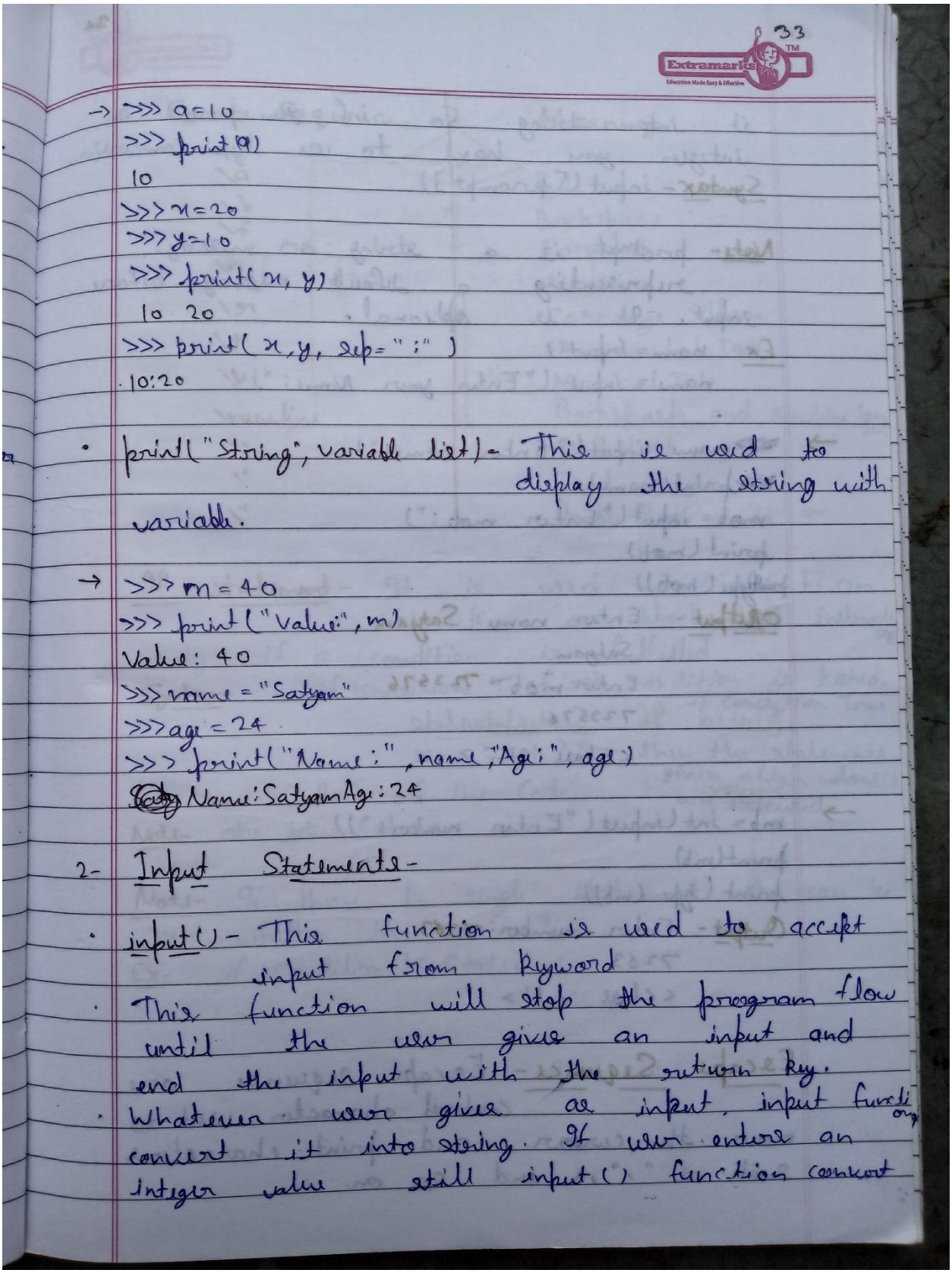
- print("string", sep="") - It separates string with given sep character. Character can be any string. Default is '' or can write None.

→ >>> print("Satyam", "Seth")
Satyam Seth
>>> print("Satyam", "Seth", sep="**")
Satyam**Seth

- print("string", end="") - When ending character is passed. It prints given character at the end. Default is '\n' or can write None.

→ print("Satyam")
print("Seth")
print("Then")
print("Satyam", end=" ")
print("Seth", end=" ")
print("Then", end=" ")
Output- Satyam
Seth
Then
Satyam Seth Then

- print(variable list) - This is used to display the value of variable or a list of variable.
Note- List of variable values in the output screen, are separated by a space by default. We can change this by sep



```
>>> a=10
>>> print(a)
10
>>> x=20
>>> y=10
>>> print(x, y)
 10 20
>>> print(x, y, sep=":")
10:20
```

- print("String", variable list) - This is used to display the string with variable.

```
>>> m=40
>>> print("Value:", m)
Value: 40
>>> name="Satyam"
>>> age=24
>>> print("Name:", name, "Age:", age)
Name: Satyam Age: 24
```

2- Input Statements -

- input() - This function is used to accept input from keyword
- This function will stop the program flow until the user gives an input and end the input with the return key.
- Whatever user gives as input, input function convert it into string. If user enter an integer value still input() function convert

it into string. So if you need an integer you have to use type conversion.

Syntax - input ([prompt])

Note - prompt is a string or message, representing a default message before input. It is optional.

Ex name = input()
name = input ("Enter your Name:")

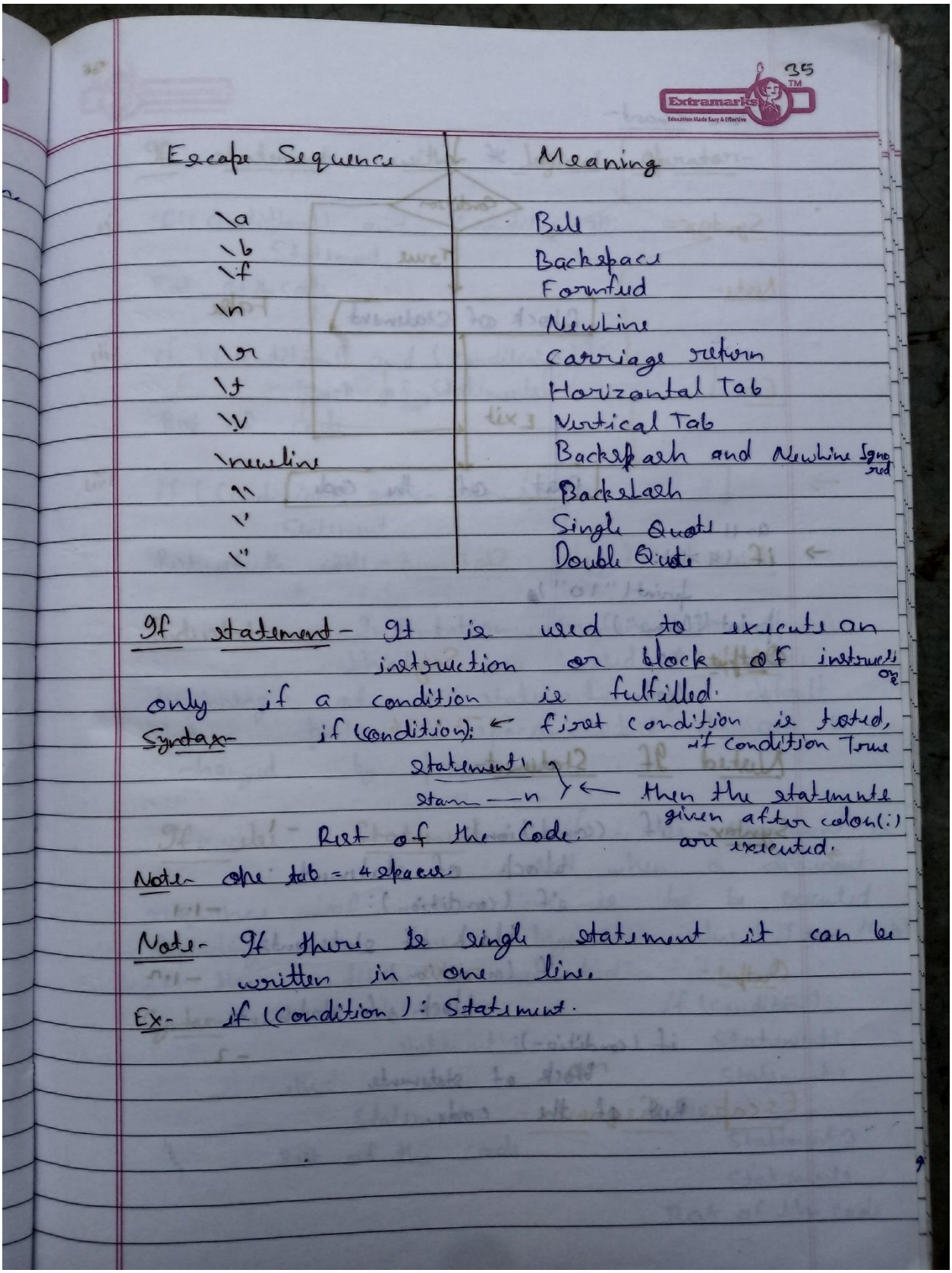
→ name = input ("Enter name:")
print(name)
mob = input ("Enter mob:")
print(mob)
print(type(mob))

Output - Enter name: Satyam
Satyam
Enter mob: 723576
723576
<class 'str'>

→ mb = int(input("Enter number:"))
print(mb)
print(type(mb))

Output - Enter number: 7263
7263
<class 'int'>

Escape Sequence - Escape sequence are control character used to move the cursor and print characters such as "\", \t and so on.



Escape Sequence	Meaning
\a	Bell
\b	Backspace
\f	Formfeed
\n	Newline
\r	Carriage return
\t	Horizontal Tab
\v	Vertical Tab
\newline	Backslash and Newline ignored
\\	Backslash
\'	Single Quote
\"	Double Quote

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

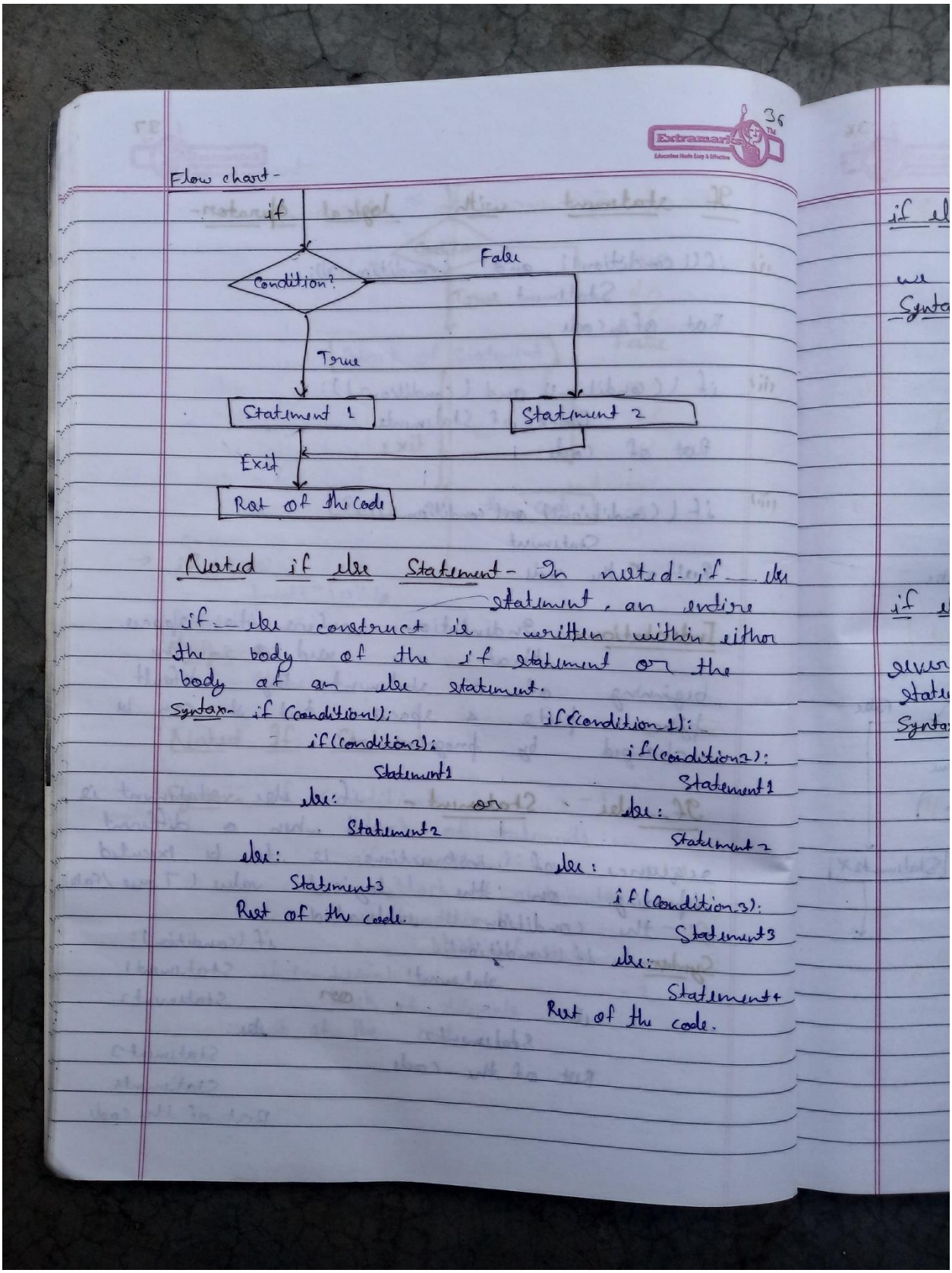
Syntax- if (condition): ← first condition is tested, if condition True
    statement
    statement
    statement ← Then the statements given after colon(:) are executed.
Rest of the Code.

Note- One tab = 4 spaces.

Note- If there is single statement it can be written in one line.

Ex- if (condition): Statement.

0



Flow chart -

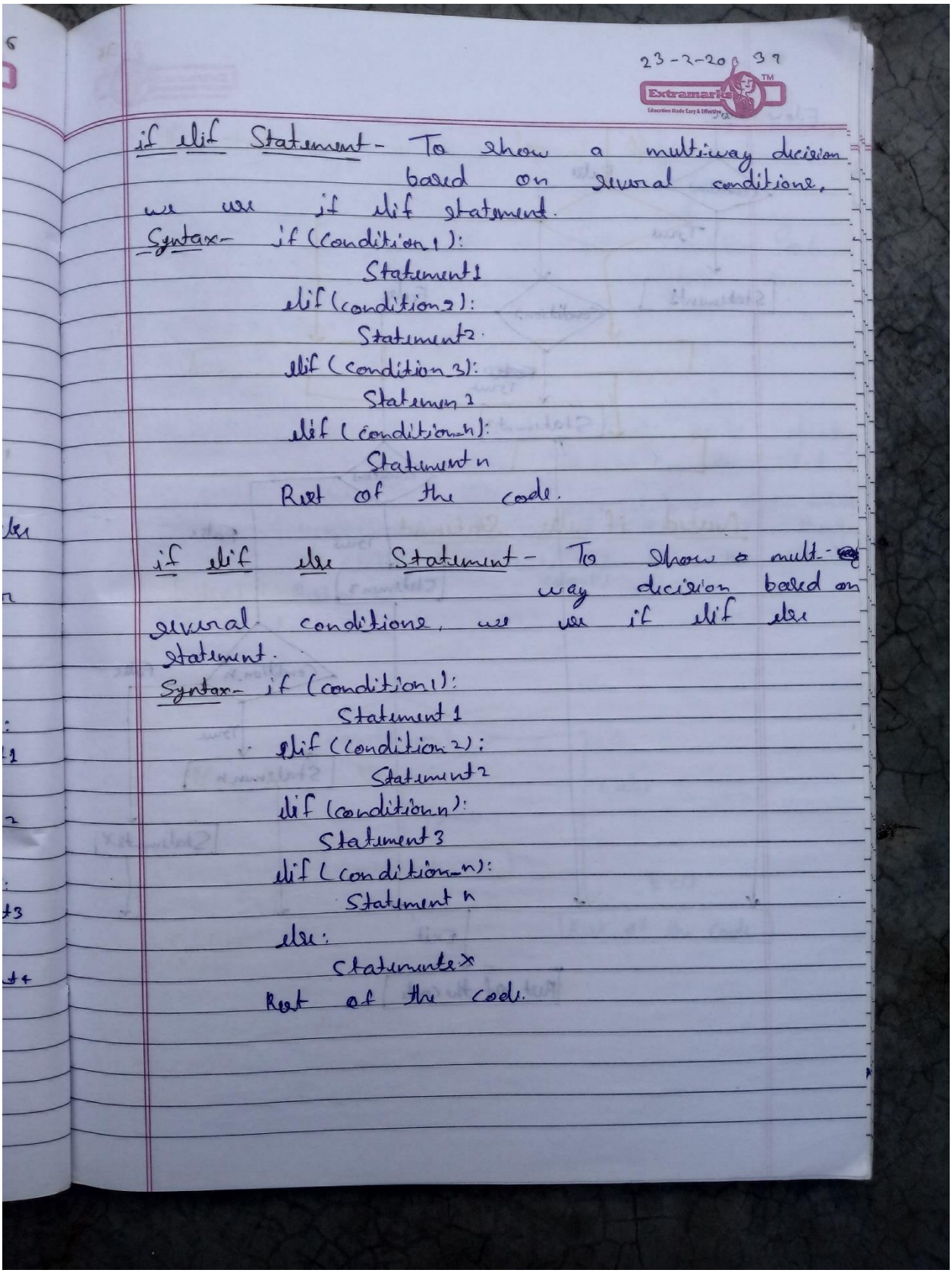
Nested if else Statement - In nested if else statement, an entire if else construct is written within either the body of the if statement or the body of an else statement.

Syntax -

```
if (condition1):
    if(condition2):
        Statement1
    else:
        Statement2
else:
    Statement3
Rest of the code.
```

or

```
if(condition1):
    if(condition2):
        Statement1
    else:
        Statement2
else:
    if(condition3):
        Statement3
    else:
        Statement4
Rest of the code.
```



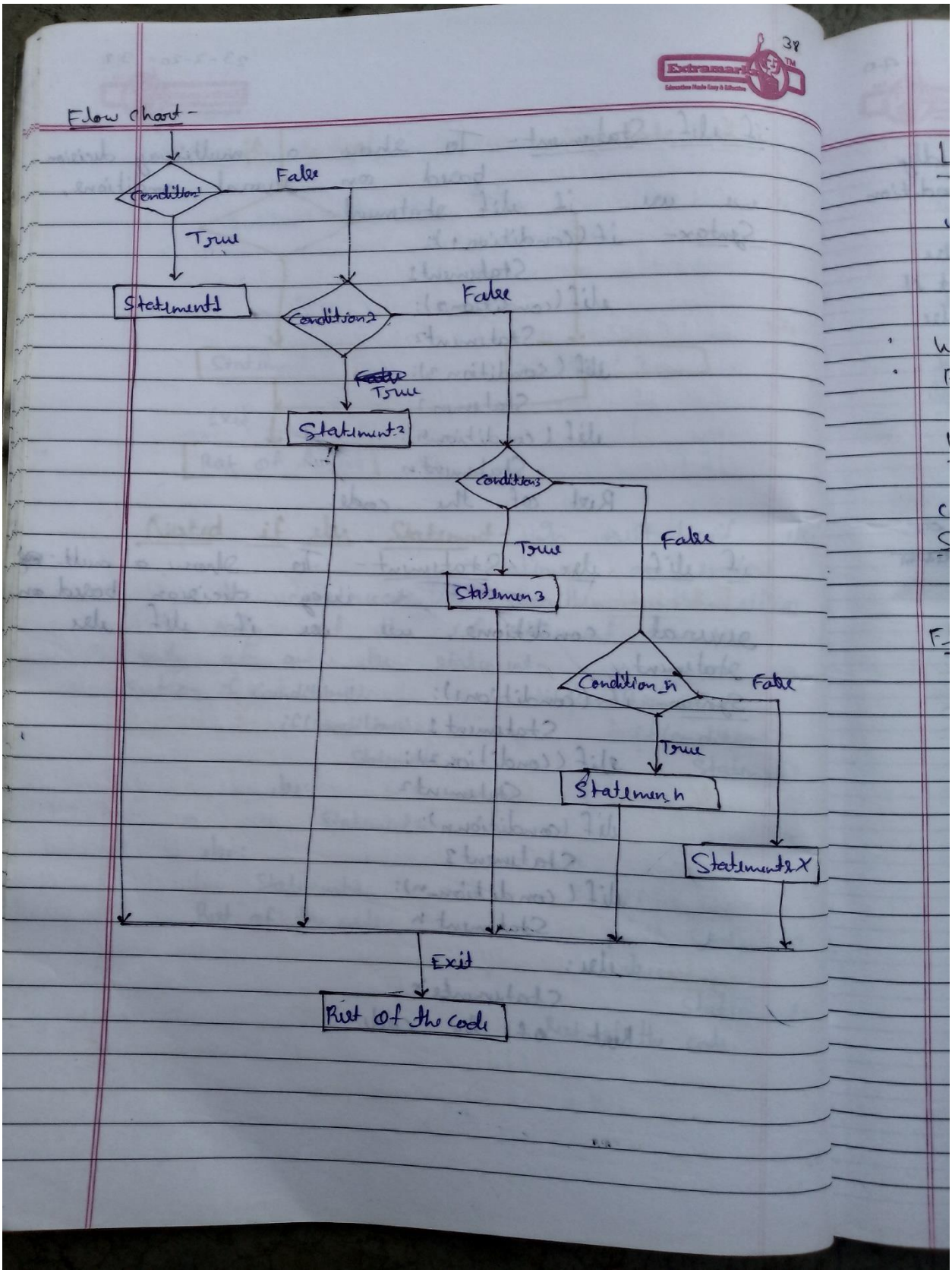
23-2-20 39

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

Syntax-
```
if (condition 1):
    Statement 1
elif (condition 2):
    Statement 2
elif (condition 3):
    Statement 3
elif (condition_n):
    Statement n
Rest of the code.
```

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

Syntax-
```
if (condition 1):
    Statement 1
elif (condition 2):
    Statement 2
elif (condition n):
    Statement 3
elif (condition_n):
    Statement n
else:
    Statements
Rest of the code.
```


Flow chart -
Condition1
False
True
Statement1
Condition2
False
True
Statement2
Condition3
True
Statement3
False
Condition_n
False
True
Statement_n
Statement_X
Exit
Rest of the code

39

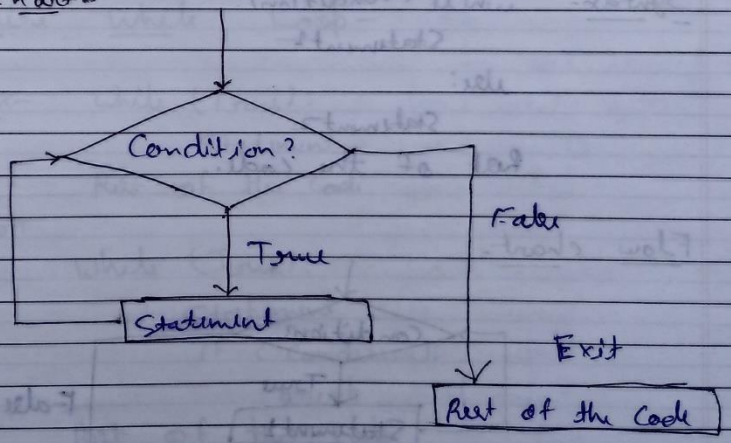
Loop Control Statements – Loop control statements are used when a section of code may either be executed a fixed number of time, or while some condition is true.

- While
- For

While Loop – The while loop keeps repeating an action until an associated condition returns false.

Syntax – while(condition): ← Python interpreter checks condition if condition is True, then
    Statement ← Execute statements written after colon(:)
Rest of code

Flow chart –



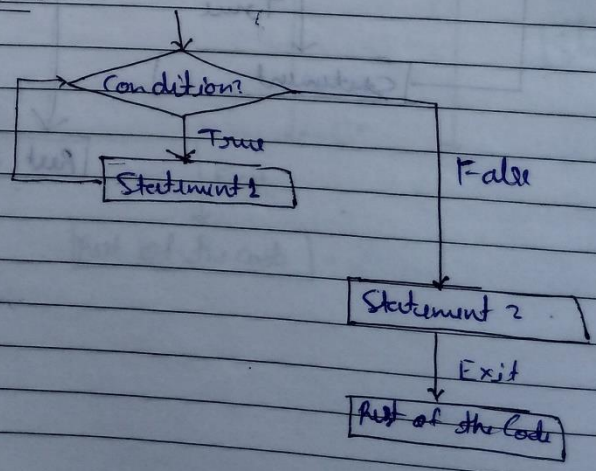
While Loop with else – This repeatedly tests the condition and, if it is True, executes the Statement 1; if the condition is False (which may be the first time it is tested) the statement 2 of the else clause is executed and the loop terminates. The else suite will be always executed irrespective of the statements in the loop are executed or not.

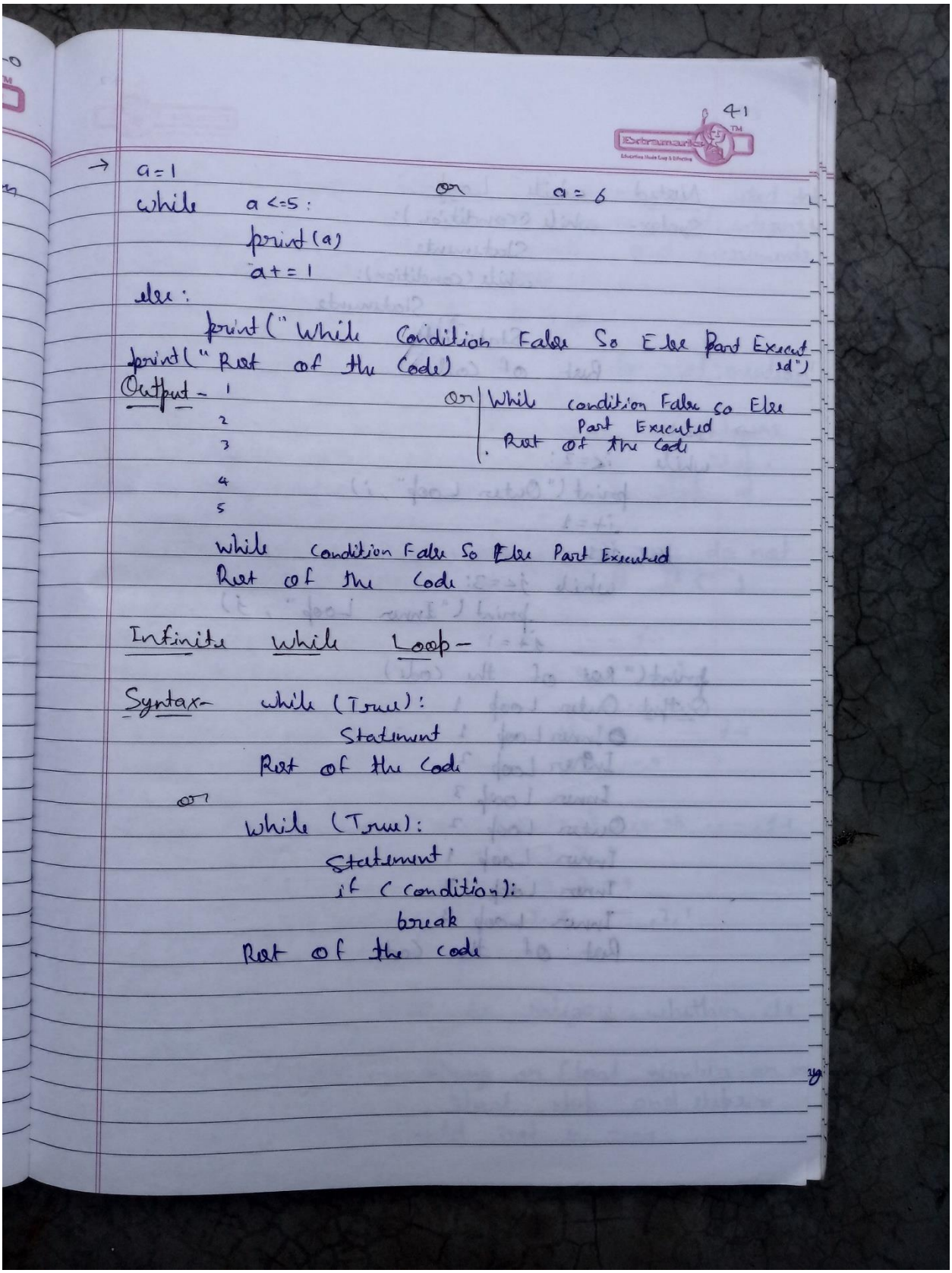
Note – जहाँ while loop कि condition true हो या वो हो while loop के बाद else part पक्का execute होगा।

Syntax –

```
while (Condition):
    Statement 1
else:
    Statement 2
Rest of the code.
```

Flow chart –





41

→ a = 1
while a <= 5:
    print(a)
    a += 1
else:
    print("While Condition False So Else Part Executed")
print("Rest of the Code")

Output -
1
2
3
4
5
While Condition False So Else Part Executed
Rest of the Code

or

a = 6
While condition False so Else Part Executed
Rest of the Code

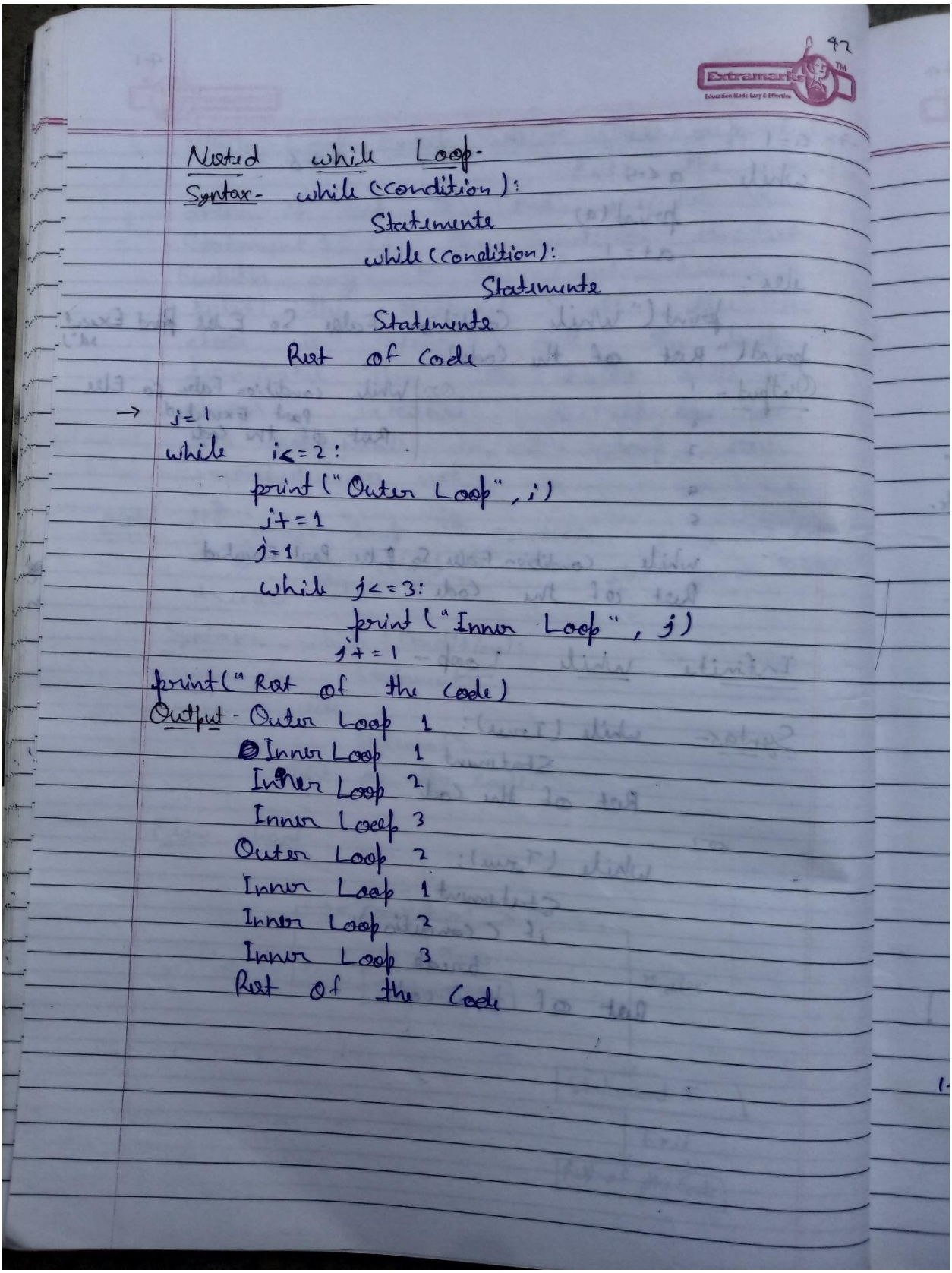
While Condition False So Else Part Executed
Rest of the Code

Infinite while Loop -

Syntax -
while (True):
    Statement
Rest of the Code

or

while (True):
    Statement
    if (condition):
        break
Rest of the code

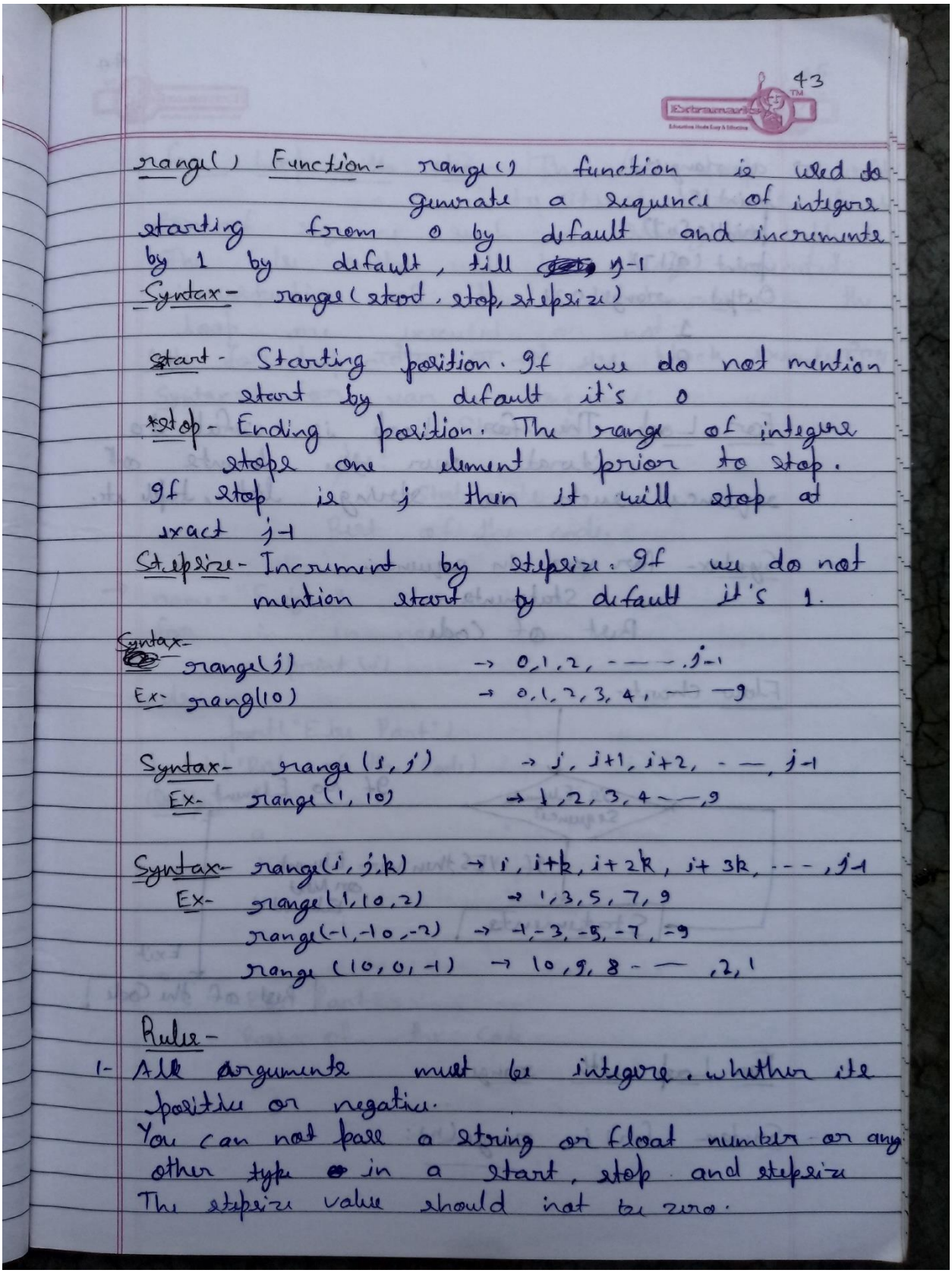


Nested while Loop-

Syntax- while (condition):
    Statements
    while (condition):
        Statements
    Statements
Rest of code

```
i=1
while i<=2:
    print("Outer Loop", i)
    i+=1
    j=1
    while j<=3:
        print("Inner Loop", j)
        j+=1
print("Rest of the code")
```

Output - Outer Loop 1
Inner Loop 1
Inner Loop 2
Inner Loop 3
Outer Loop 2
Inner Loop 1
Inner Loop 2
Inner Loop 3
Rest of the Code



43

range() Function- range() function is used to generate a sequence of integers starting from 0 by default and increments by 1 by default, till y-1

Syntax- range(start, stop, stepsize)

start - Starting position. If we do not mention start by default it's 0

stop - Ending position. The range of integers stops one element prior to stop. If stop is j then it will stop at exact j-1

Stepsize - Increment by stepsize. If we do not mention start by default it's 1.

Syntax- range(j) → 0, 1, 2, ---, j-1
Ex- range(10) → 0, 1, 2, 3, 4, --- 9

Syntax- range(i, j) → i, i+1, i+2, ---, j-1
Ex- range(1, 10) → 1, 2, 3, 4 --- 9

Syntax- range(i, j, k) → i, i+k, i+2k, i+3k, ---, j-1
Ex- range(1, 10, 2) → 1, 3, 5, 7, 9
range(-1, -10, -2) → -1, -3, -5, -7, -9
range(10, 0, -1) → 10, 9, 8 --- , 2, 1

Rules-

1- All arguments must be integers. whether its positive or negative.
You can not pass a string or float number or any other type in a start, stop and stepsize
The stepsize value should not be zero.

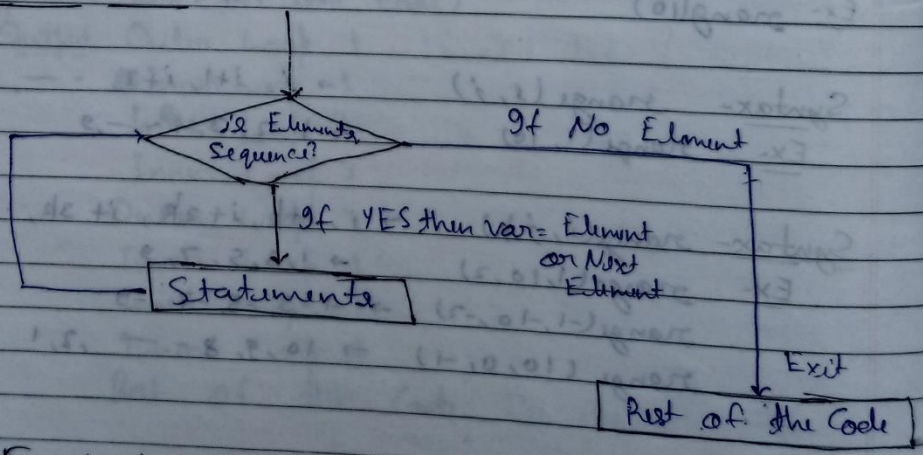
```
a = range(2)
print(a)
print(a[0])
print(a[1])
```

Output - range(0,2)
1
0

For Loop - The for Loop is useful to iterate over the elements of sequence such as string list, tuple etc.

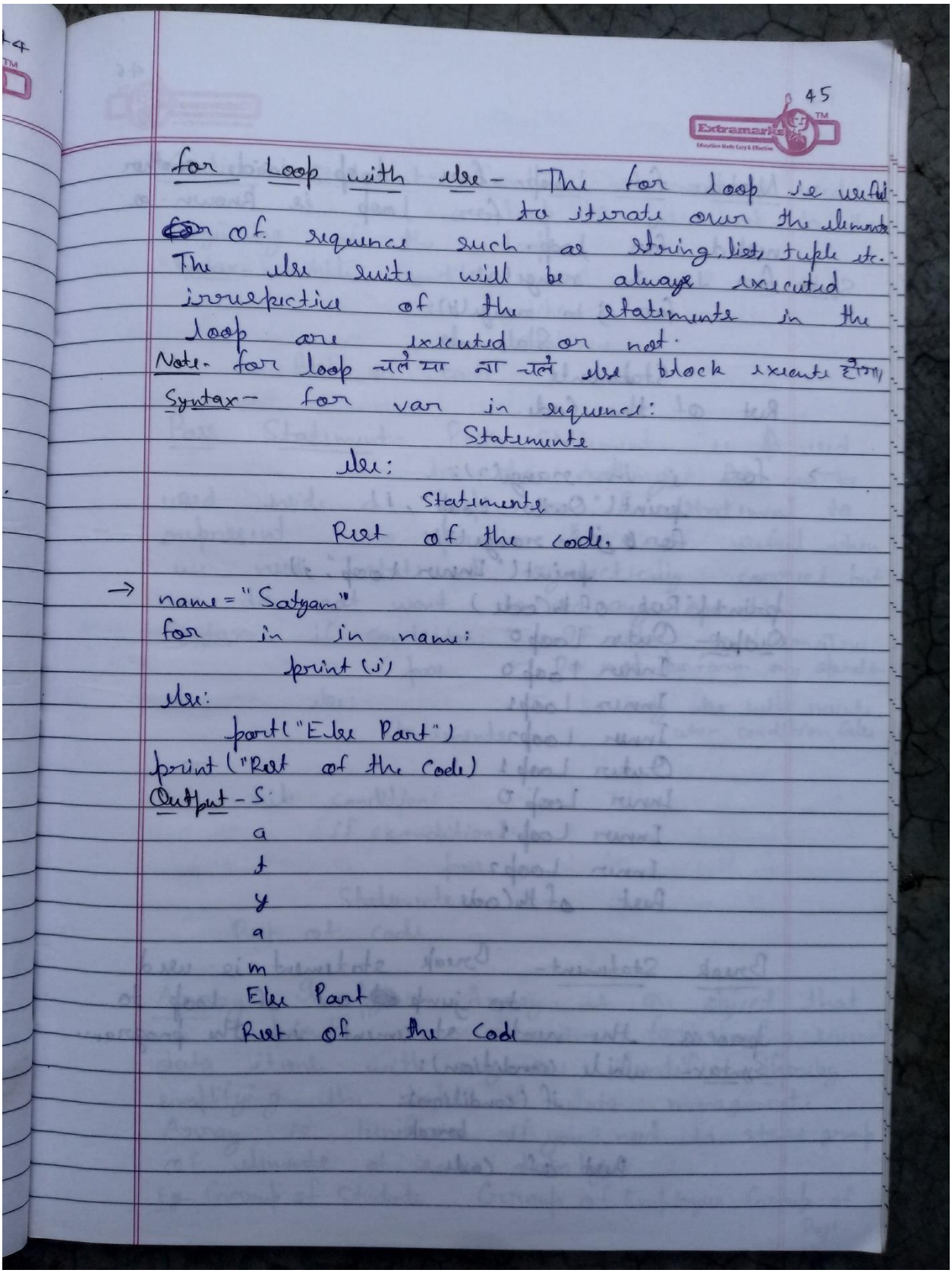
Syntax - for var in sequence:
    Statements
Rest of Code

Flow Chart -



For Loop with range -

Syntax - for i in range(n):



45

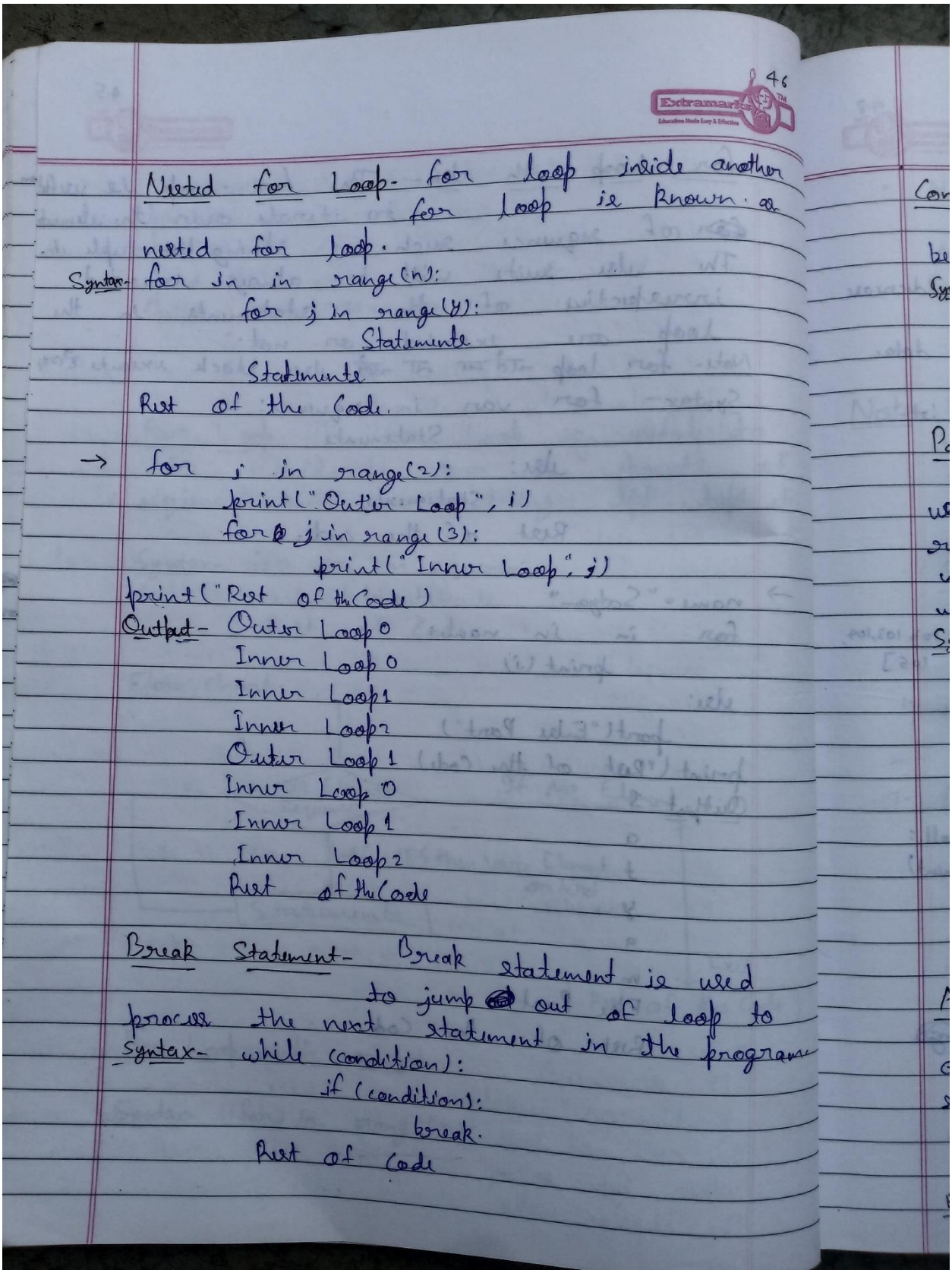
for Loop with else - The for loop is useful to iterate over the elements of sequence such as string, list, tuple etc. The else suite will be always executed irrespective of the statements in the loop are executed or not.

Note- for loop चले या ना चले else block execute होगा।

Syntax- for var in sequence:
    Statements
else:
    Statements
Rest of the code.

→
```
name = "Satyam"
for i in name:
    print(i)
else:
    print("Else Part")
print("Rest of the Code")
```

Output - S
a
t
y
a
m
Else Part
Rest of the Code



46

Nested for Loop- for loop inside another for loop is known as nested for loop.

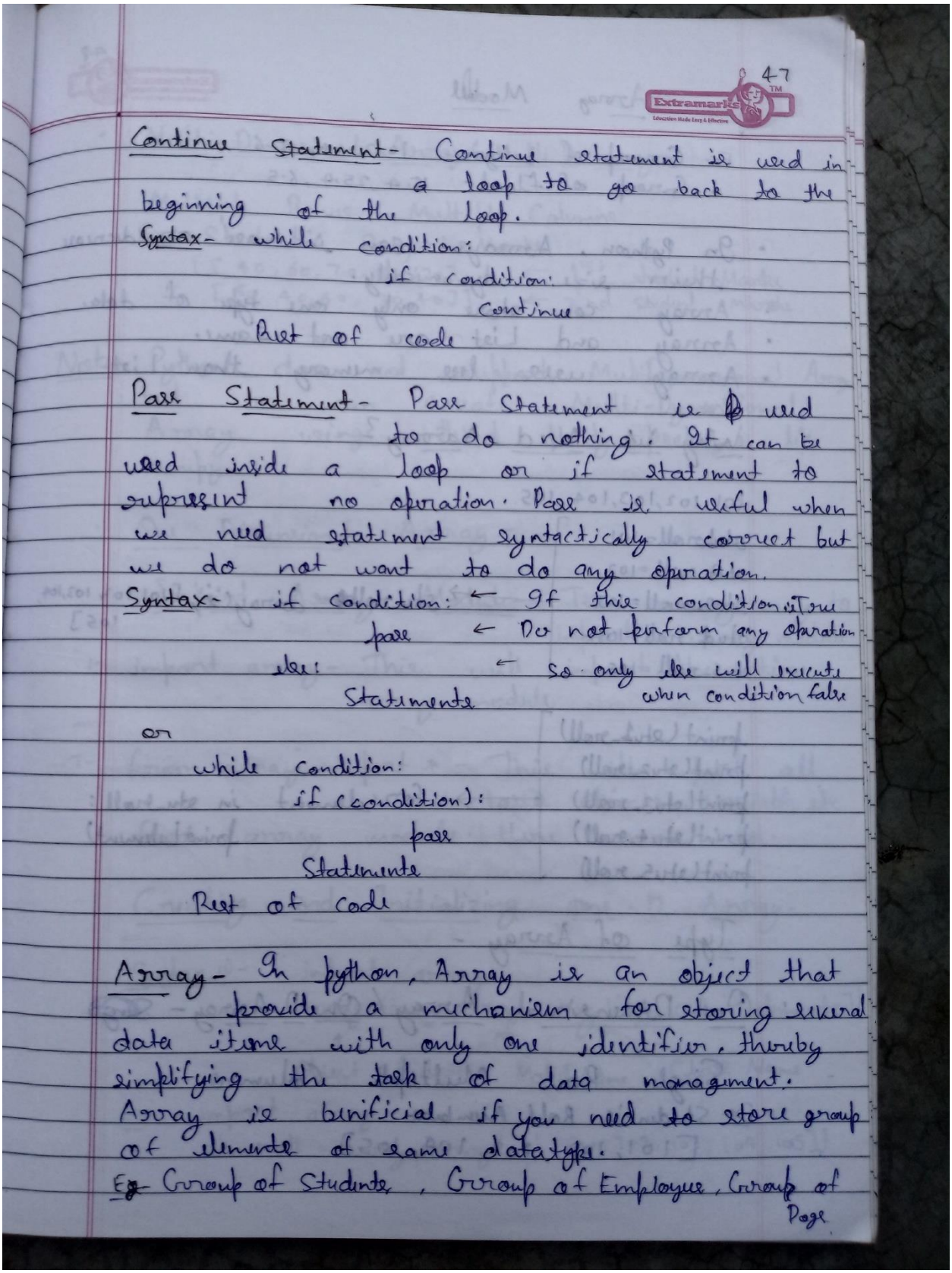
Syntax- for in in range(n):
    for j in range(y):
        Statements
    Statements
Rest of the Code.

→ for i in range(2):
    print("Outer Loop", i)
    for j in range(3):
        print("Inner Loop", j)
print("Rest of the Code")

Output- Outer Loop 0
Inner Loop 0
Inner Loop 1
Inner Loop 2
Outer Loop 1
Inner Loop 0
Inner Loop 1
Inner Loop 2
Rest of the Code

Break Statement- Break statement is used to jump out of loop to process the next statement in the program.

Syntax- while (condition):
    if (condition):
        break.
    Rest of code



47

Continue Statement- Continue statement is used in a loop to go back to the beginning of the loop.

Syntax- while condition:
    if condition:
        continue
    Rest of code

Pass Statement- Pass Statement is used to do nothing. It can be used inside a loop or if statement to represent no operation. Pass is useful when we need statement syntactically correct but we do not want to do any operation.

Syntax- if condition: ← If this condition is true
    pass ← Do not perform any operation
else: ← so only else will execute when condition false
    Statements

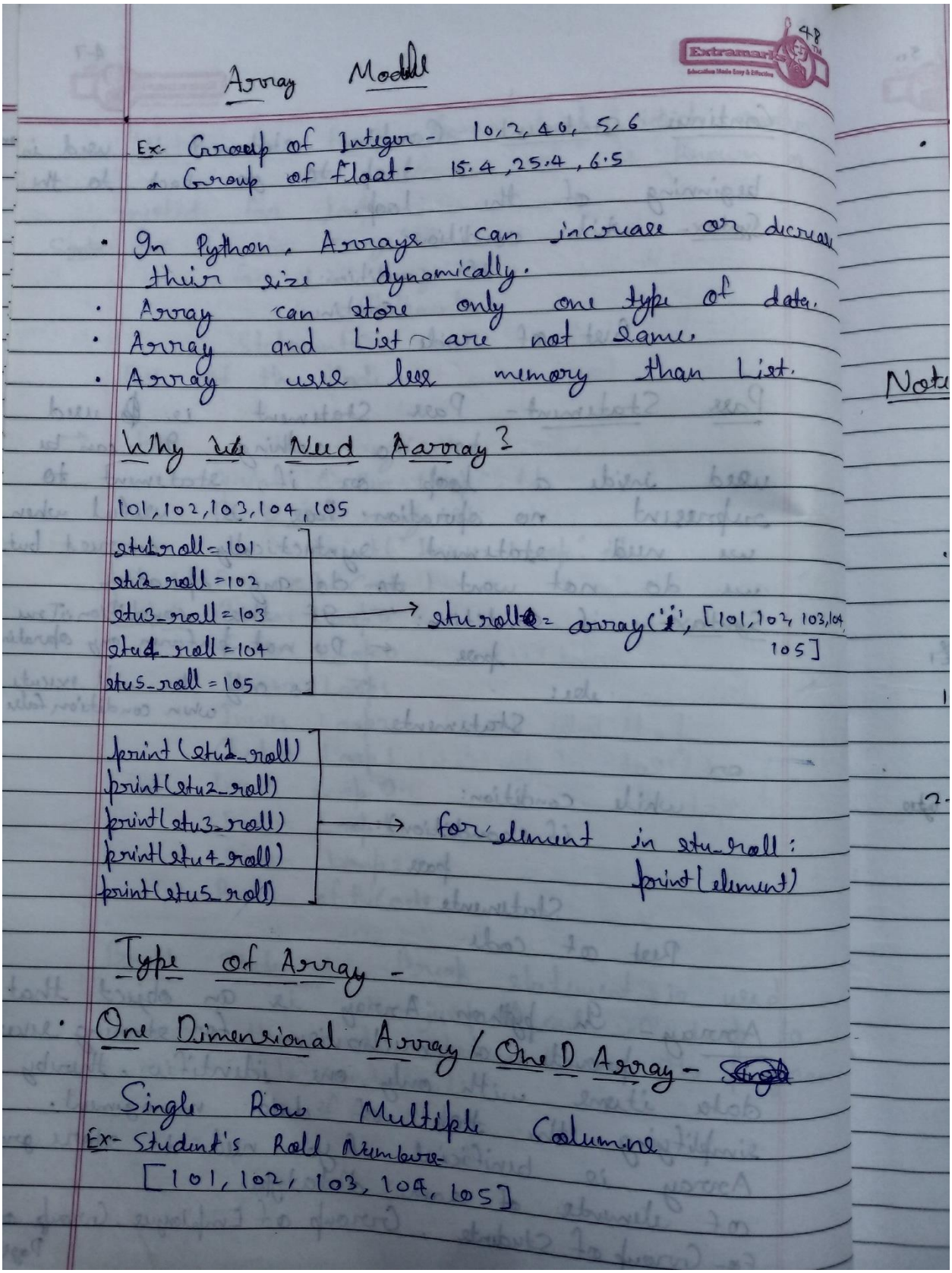
or

while condition:
    if (condition):
        pass
    Statements
Rest of code

Array- In python, Array is an object that provide a mechanism for storing several data items with only one identifier, thereby simplifying the task of data management. Array is benificial if you need to store group of elements of same datatype.

Eg- Group of Students, Group of Employee, Group of

Page



Array Module

Ex. Group of Integer - 10, 2, 40, 5, 6
or Group of float - 15.4, 25.4, 6.5

- In Python, Arrays can increase or decrease their size dynamically.
- Array can store only one type of data.
- Array and List are not same.
- Array use less memory than List.

Note

Why we Need Array?

101, 102, 103, 104, 105

```
stu1_roll = 101
stu2_roll = 102
stu3_roll = 103
stu4_roll = 104
stu5_roll = 105
```

→ stu_roll = array('i', [101, 102, 103, 104, 105]

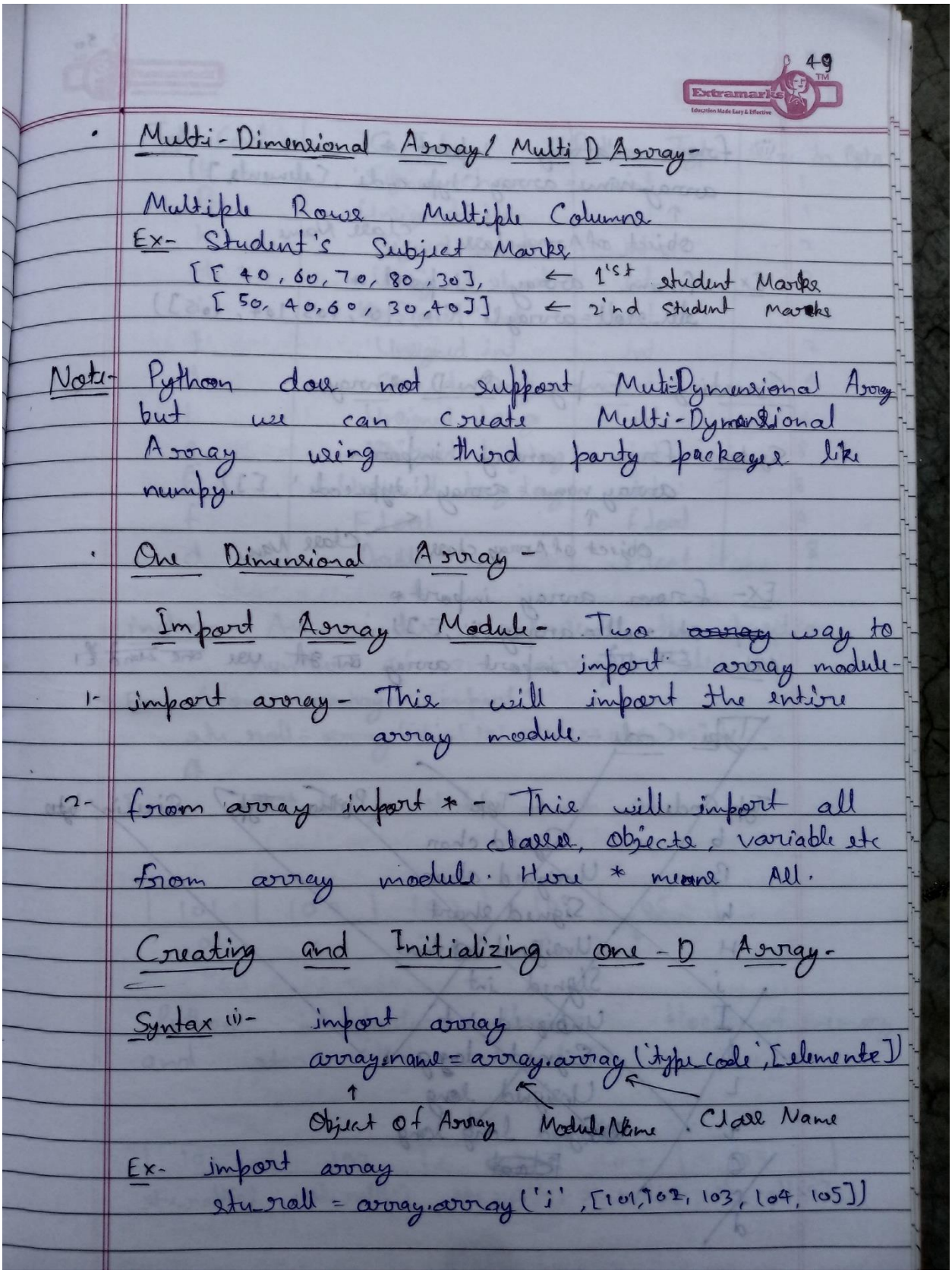
```
print(stu1_roll)
print(stu2_roll)
print(stu3_roll)
print(stu4_roll)
print(stu5_roll)
```

→ for element in stu_roll:
    print(element)

Type of Array -

- **One Dimensional Array / One D Array** - Single Row Multiple Columns

Ex- Student's Roll Numbers-
[101, 102, 103, 104, 105]



- Multi-Dimensional Array / Multi D Array-

Multiple Rows Multiple Columns
Ex- Student's Subject Marks
[[40, 60, 70, 80, 30], ← 1st student Marks
[50, 40, 60, 30, 40]] ← 2nd Student Marks

Note- Python does not support Multi-Dymensional Array but we can create Multi-Dymensional Array using third party packages like numpy.

- One Dimensional Array -

Import Array Module- Two ~~array~~ way to import array module-

1- import array- This will import the entire array module.

2- from array import * - This will import all classes, objects, variable etc from array module. Here * means All.

Creating and Initializing one-D Array-

Syntax (i)- import array
arrayname = array.array('type_code', [elements])
↑ Object of Array, Module Name . Class Name

Ex- import array
stu_roll = array.array('i', [101, 102, 103, 104, 105])

(ii) from array import *
array_name = array('type_code', [elements])
↑ Object of Array class ← class Name

Ex- from array import *
stu_roll = array('i', [101, 102, 103, 104, 105])

Creating Empty One-D Array-

Syntax- from array import *
array_name = array('type_code', [])
↑ Object of Array class ← class Name

Ex- from array import *
stu_roll = array('i', [])

Note- इस तरह से import array का ही use कर सकते हैं।

Type Code -

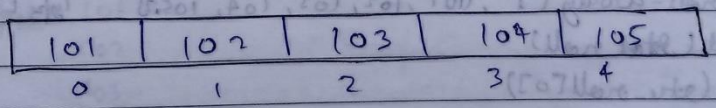
Type Code	C Type	Python Type	Size in byte
b	Signed char		
B	Unsigned char		
h	Signed short		
H	Unsigned short		
i	Signed int		
I	Unsigned int		
l	Signed long		
L	Unsigned long		
q	Signed long long		
Q			
f			
d			

Type Code	C Type	Python Type	Size in Bytes
b	Signed char	int	1
B	Unsigned char	int	1
h	Signed short	int	2
H	Unsigned short	int	2
i	Signed int	int	2
I	Unsigned int	int	2
l	Signed long	int	4
L	Unsigned long	int	4
q	Signed long long	int	8
Q	Unsigned long long	int	8
f	Float	float	4
d	Double	float	8

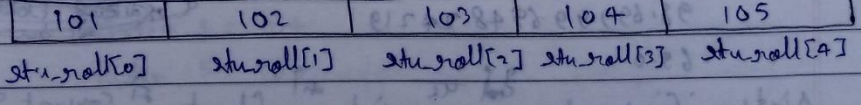
index- An index represents the position number of an array's element.

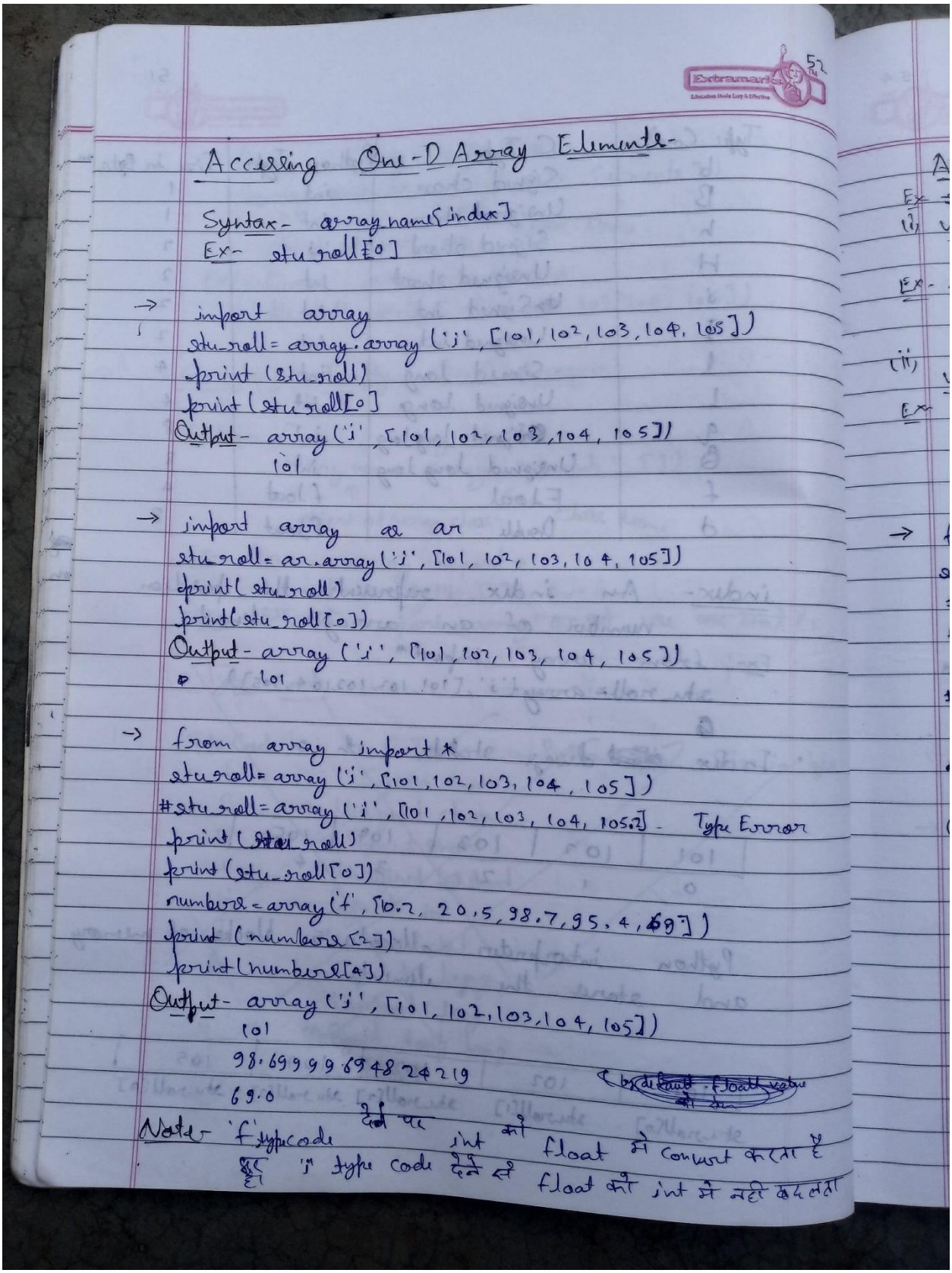
Ex- from array import *
stu_roll = array('i', [101, 102, 103, 104, 105])

- Index always starts with 0



Python interpreter allocates 5 blocks of memory and stores the elements





Accessing One-D Array Elements-

Syntax- array_name[index]
Ex- stu_roll[0]

→
```
import array
stu_roll = array.array('i', [101, 102, 103, 104, 105])
print(stu_roll)
print(stu_roll[0])
```
Output- array('i', [101, 102, 103, 104, 105])
101

→
```
import array as ar
stu_roll = ar.array('i', [101, 102, 103, 104, 105])
print(stu_roll)
print(stu_roll[0])
```
Output- array('i', [101, 102, 103, 104, 105])
101

→
```
from array import *
stu_roll = array('i', [101, 102, 103, 104, 105])
#stu_roll = array('i', [101, 102, 103, 104, 105.7]) - Type Error
print(stu_roll)
print(stu_roll[0])
number = array('f', [10.2, 20.5, 98.7, 95.4, 69])
print(number[2])
print(number[4])
```
Output- array('i', [101, 102, 103, 104, 105])
101
98.69999694824219
69.0

Note- 'f' typecode देने पर int को float में convert करता है
but 'i' type code देने से float को int में नहीं बदलता

25-2-20 53

Accessing Array using for Loop-

Ex stu_roll = array('i', [101, 102, 103, 104, 105])

(i) Without index-

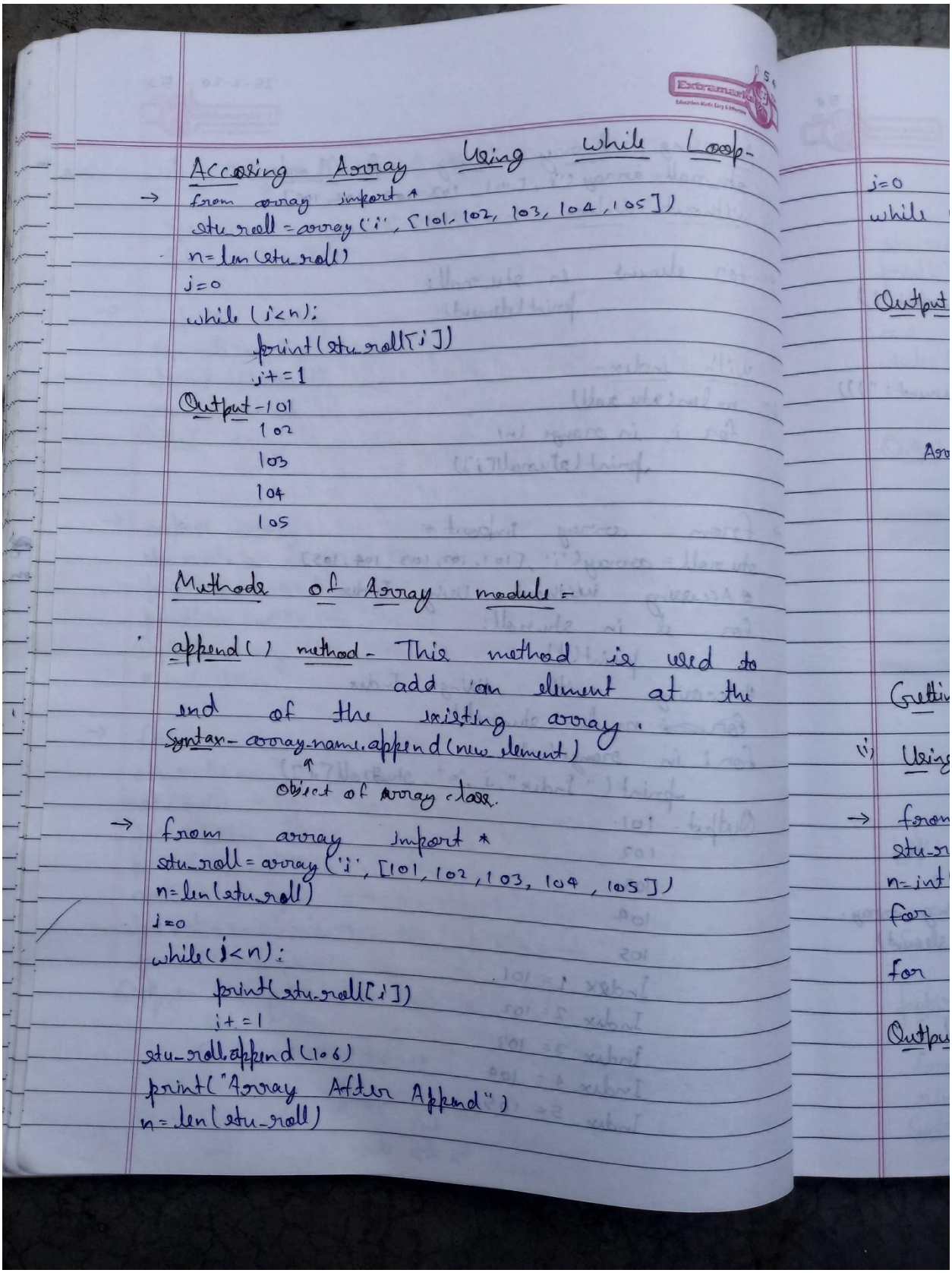
Ex- for element in stu_roll:
    print(element)

(ii) With index-

Ex n = len(stu_roll)
for i in range(n):
    print(stu_roll[i])

→ from array import *
stu_roll = array('i', [101, 102, 103, 104, 105])
# Accessing Without Using Index
for el in stu_roll:
    print(el)
# Accessing with Using Index
n = len(stu_roll)
for i in range(n):
    print("Index", i, "=", stu_roll[i])

Output - 101
102
103
104
105
Index 1 = 101
Index 2 = 102
Index 3 = 103
Index 4 = 104
Index 5 = 105



Accessing Array Using while Loop-

→
```
from array import *
stu_roll = array('i', [101, 102, 103, 104, 105])
n = len(stu_roll)
i = 0
while (i<n):
    print(stu_roll[i])
    i = i+1
```

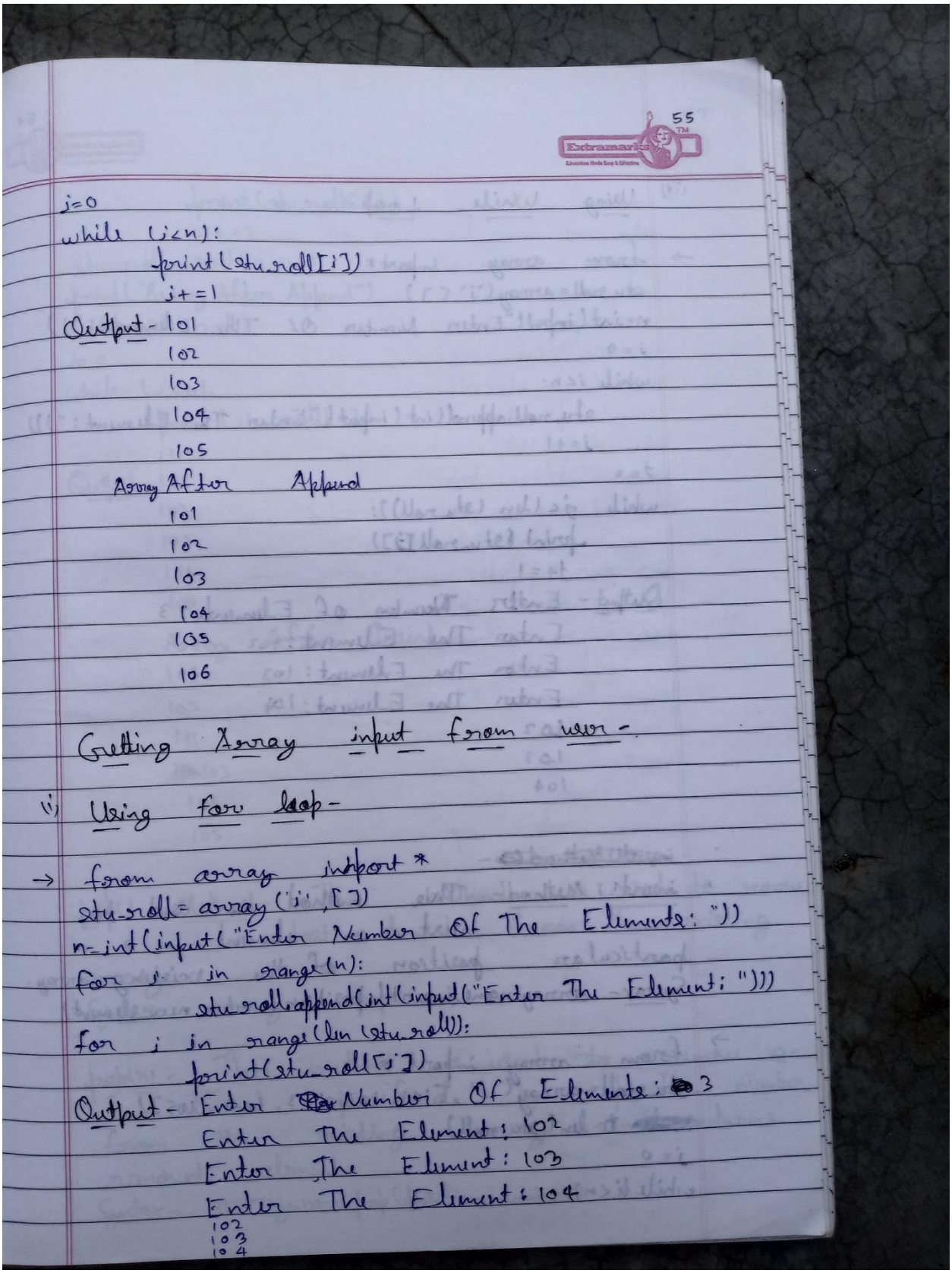
Output - 101
102
103
104
105

Methods of Array module -

append() method - This method is used to add an element at the end of the existing array.

Syntax - array_name.append(new_element)
↑
Object of array class.

→
```
from array import *
stu_roll = array('i', [101, 102, 103, 104, 105])
n = len(stu_roll)
i = 0
while(i<n):
    print(stu_roll[i])
    i = i+1
stu_roll.append(106)
print("Array After Append")
n = len(stu_roll)
```



```
i=0
while (i<n):
    print(stu_roll[i])
    i+=1
```

Output - 101
102
103
104
105

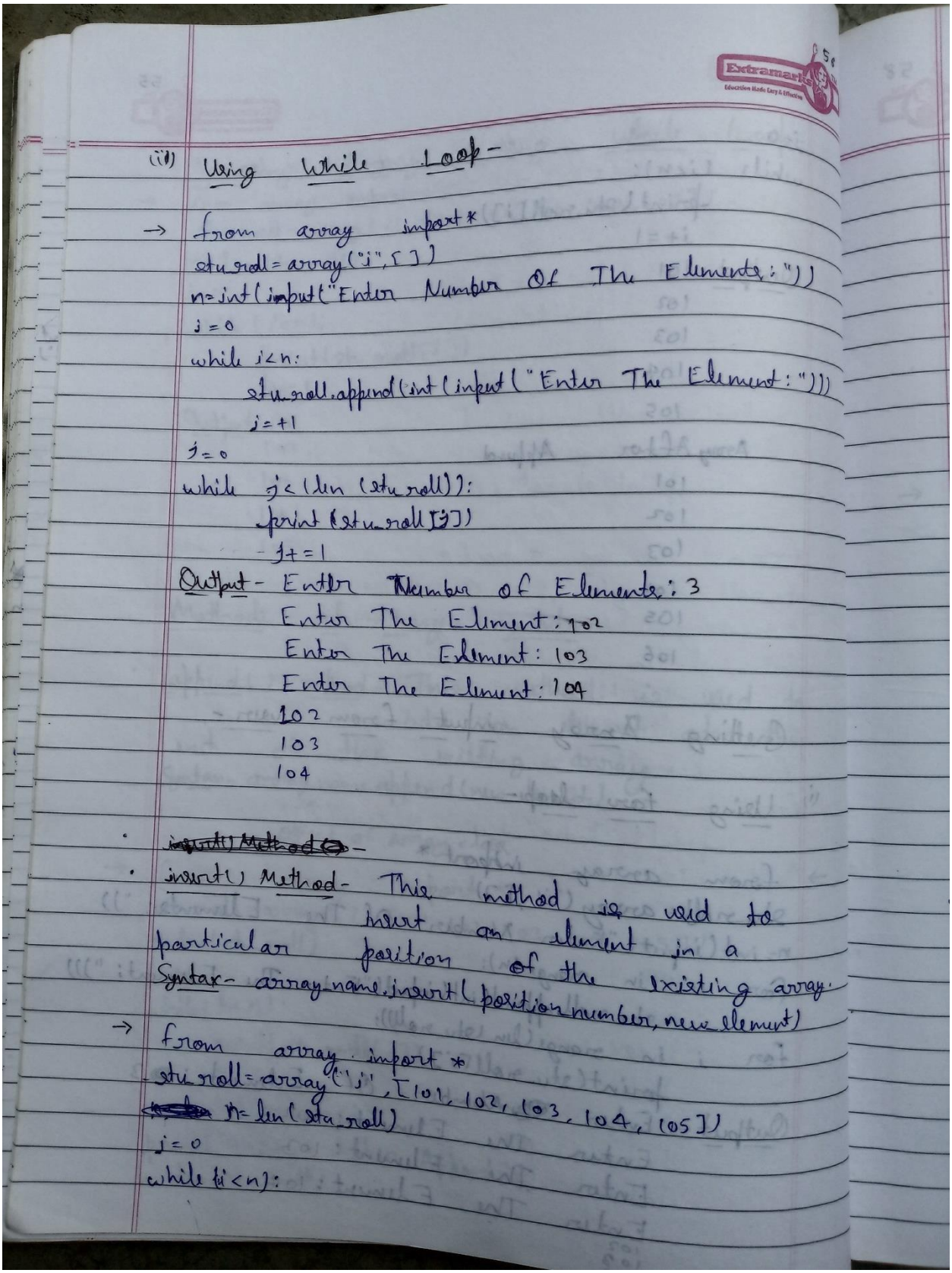
Array After Append
101
102
103
104
105
106

Getting Array input from user -

i) Using for loop -

→
```
from array import *
stu_roll=array('i',[])
n=int(input("Enter Number Of The Elements: "))
for i in range(n):
    stu_roll.append(int(input("Enter The Element: ")))
for i in range(len(stu_roll)):
    print(stu_roll[i])
```

Output - Enter Number Of Elements: 3
Enter The Element: 102
Enter The Element: 103
Enter The Element: 104
102
103
104



(ii) Using While Loop -

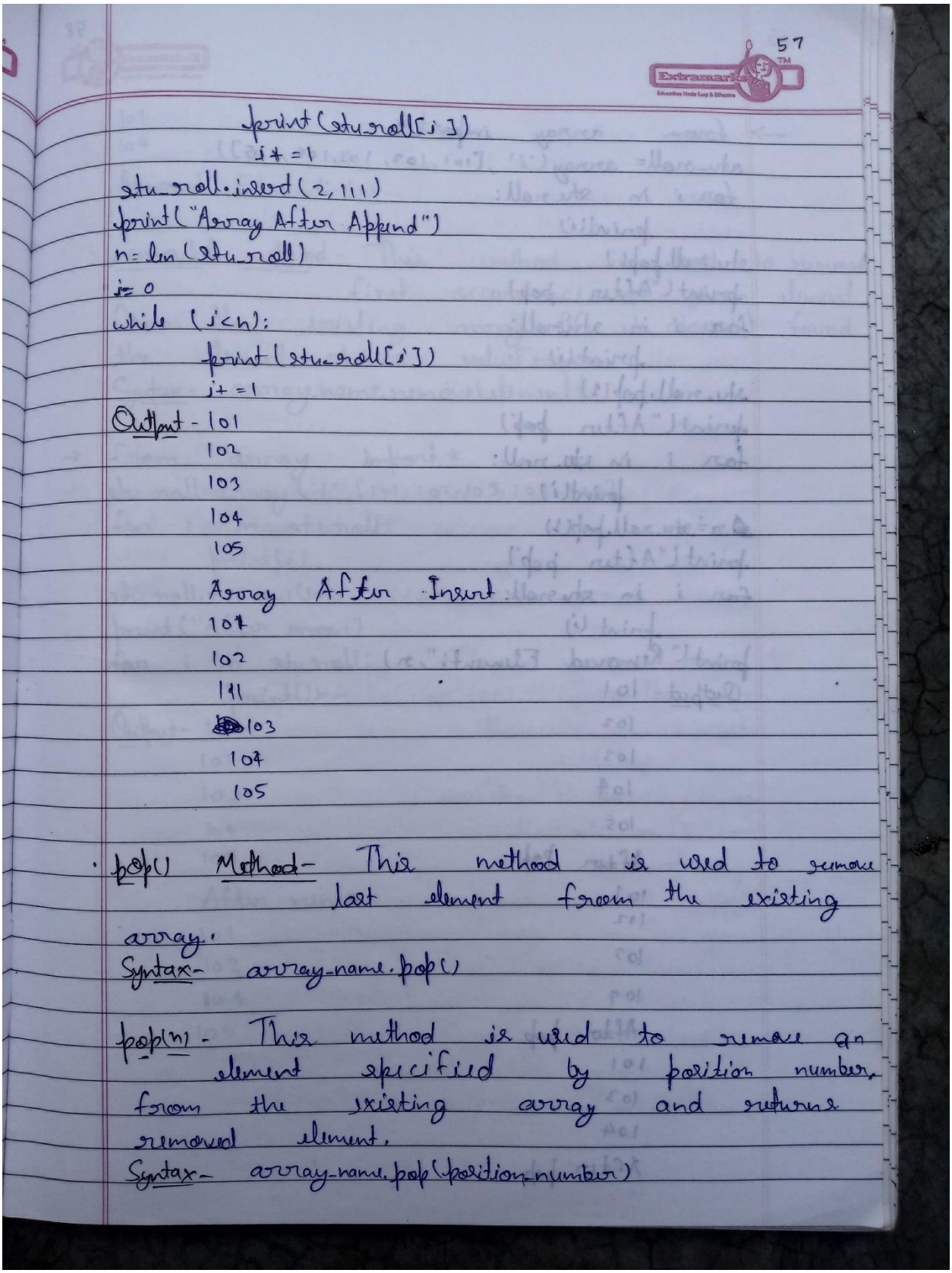
→
```
from array import *
stu_roll = array("i",[])
n=int(input("Enter Number Of The Elements:"))
i=0
while i<n:
    stu_roll.append(int(input("Enter The Element:")))
    i=+1
j=0
while j<(len(stu_roll)):
    print(stu_roll[j])
    j+=1
```

Output - Enter Number of Elements: 3
Enter The Element: 102
Enter The Element: 103
Enter The Element: 104
102
103
104

- ~~insert() Method~~ -
- insert() Method - This method is used to insert an element in a particular position of the existing array.

Syntax - array_name.insert(position number, new element)

→
```
from array import *
stu_roll=array('i',[101,102,103,104,105])
n=len(stu_roll)
i=0
while i<n):
```



```
        print(stu_roll[i])
        i+=1
stu_roll.insert(2,111)
print("Array After Append")
n=len(stu_roll)
i=0
while (i<n):
        print(stu_roll[i])
        i+=1
```

Output - 101
102
103
104
105

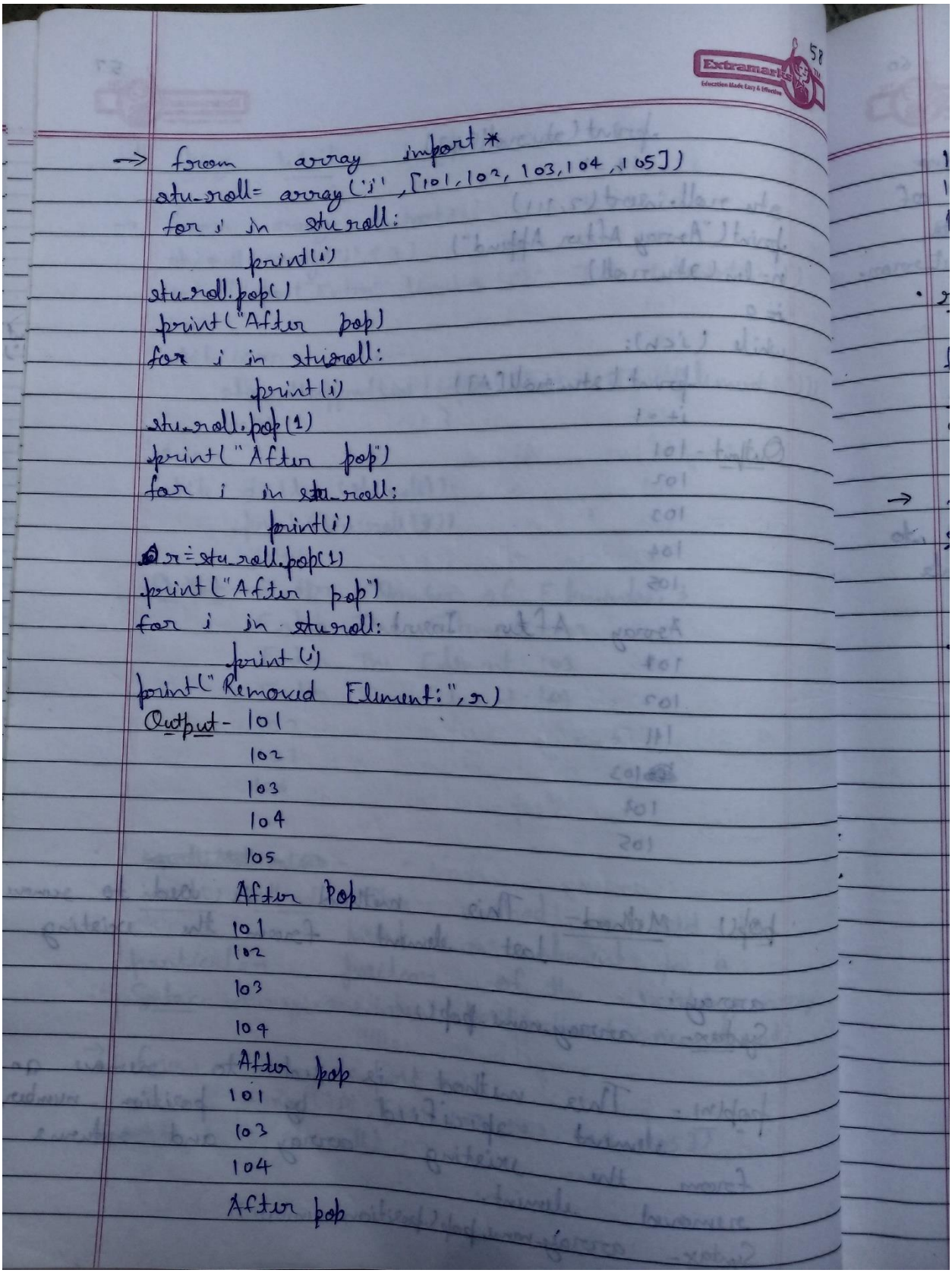
Array After Insert
101
102
111
103
104
105

- pop() Method - This method is used to remove last element from the existing array.

Syntax - array_name.pop()

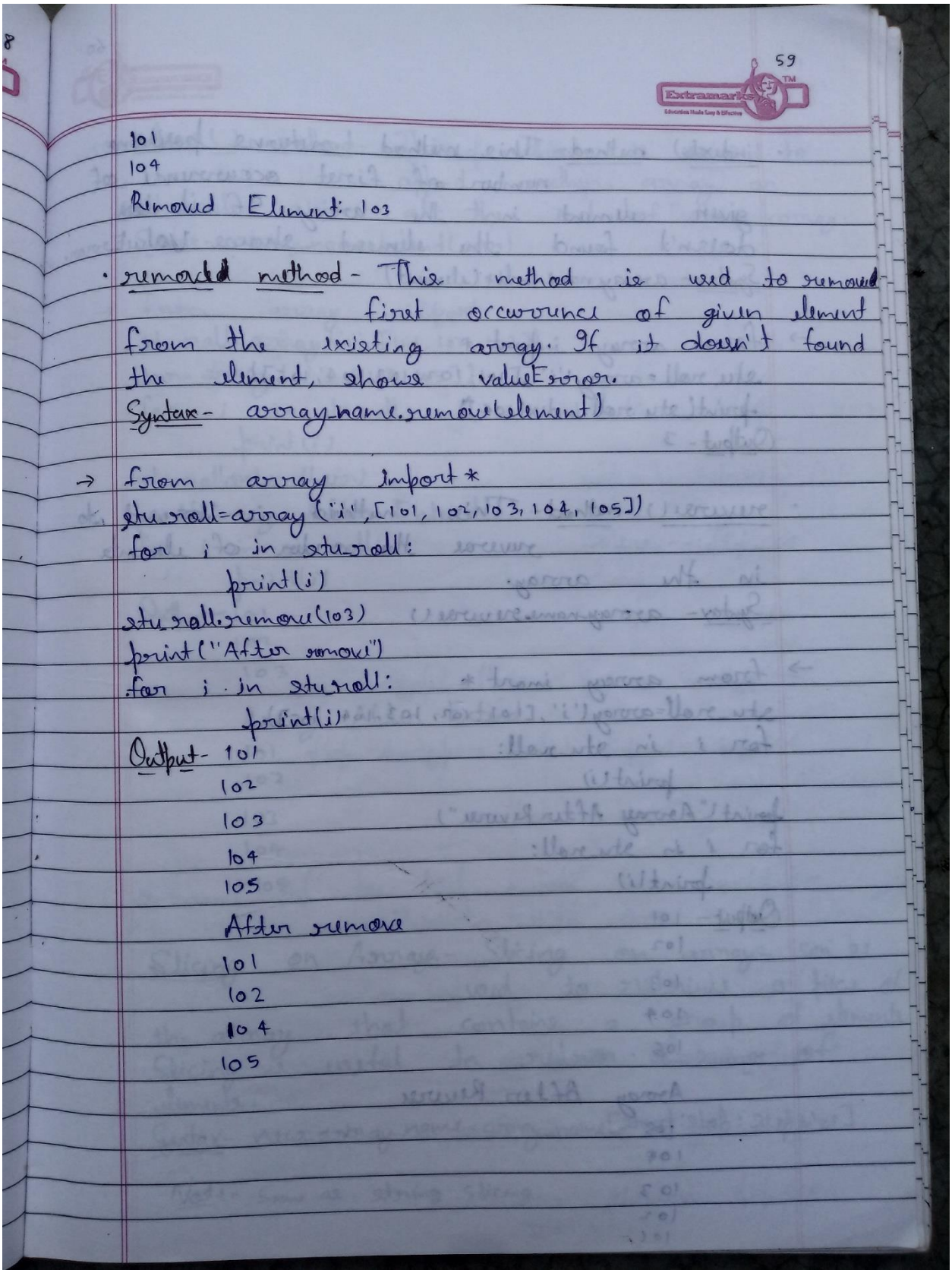
pop(n) - This method is used to remove an element specified by position number, from the existing array and returns removed element.

Syntax - array_name.pop(position_number)



```
→ from array import *
stu_roll= array('i',[101,102,103,104,105])
for i in stu_roll:
        print(i)
stu_roll.pop()
print("After pop")
for i in stu_roll:
        print(i)
stu_roll.pop(1)
print("After pop")
for i in stu_roll:
        print(i)
r=stu_roll.pop(1)
print("After pop")
for i in stu_roll:
        print(i)
print("Removed Element:",r)
```

Output-
```
101
102
103
104
105
After Pop
101
102
103
104
After pop
101
103
104
After pop
```



59

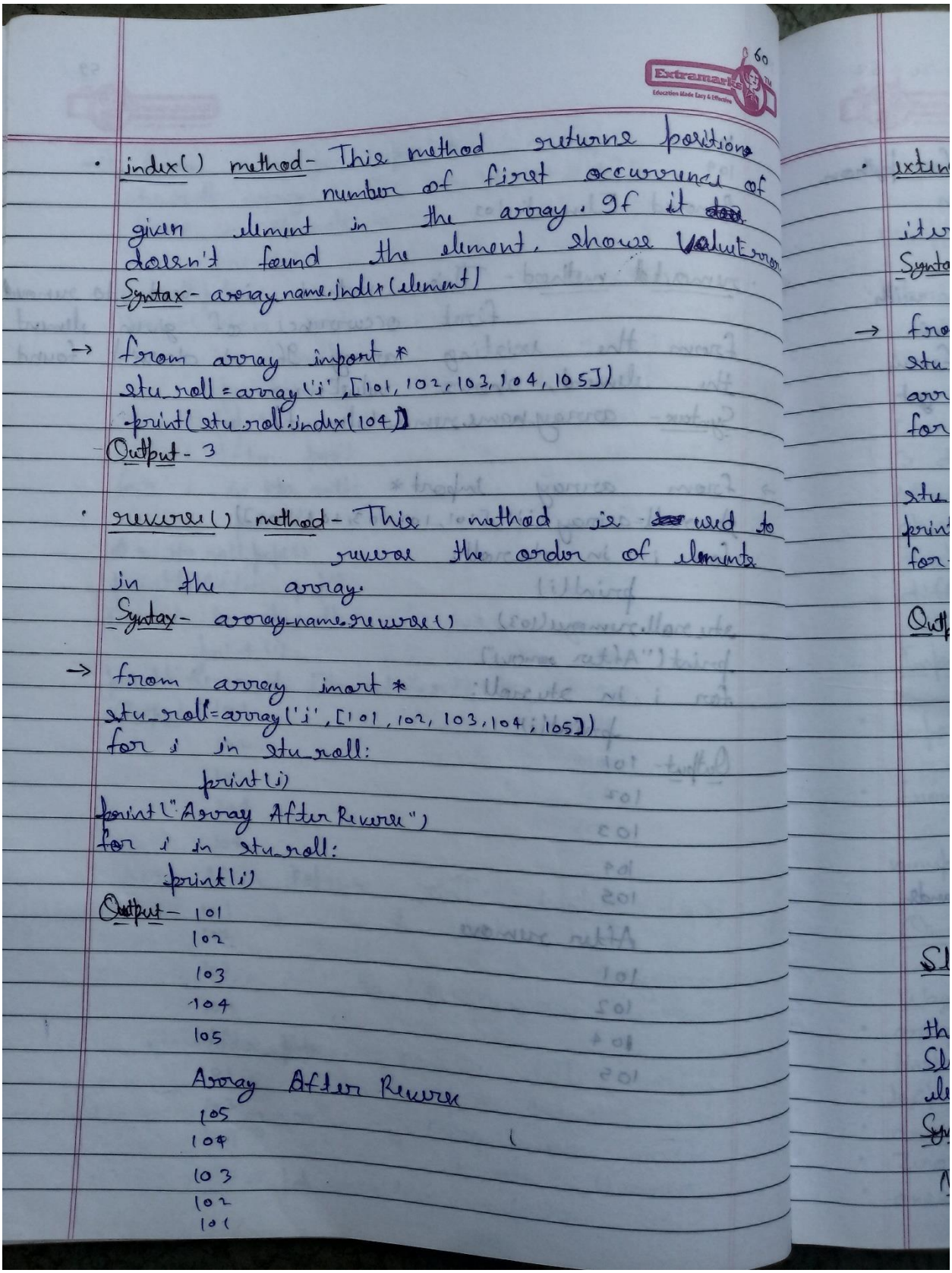
101
104
Removed Element: 103

- remove() method - This method is used to remove first occurrence of given element from the existing array. If it doesn't found the element, shows valueError.

Syntax - array_name.remove(element)

→
```
from array import *
stu_roll=array('i',[101,102,103,104,105])
for i in stu_roll:
    print(i)
stu_roll.remove(103)
print("After remove")
for i in stu_roll:
    print(i)
```

Output - 101
102
103
104
105
After remove
101
102
104
105

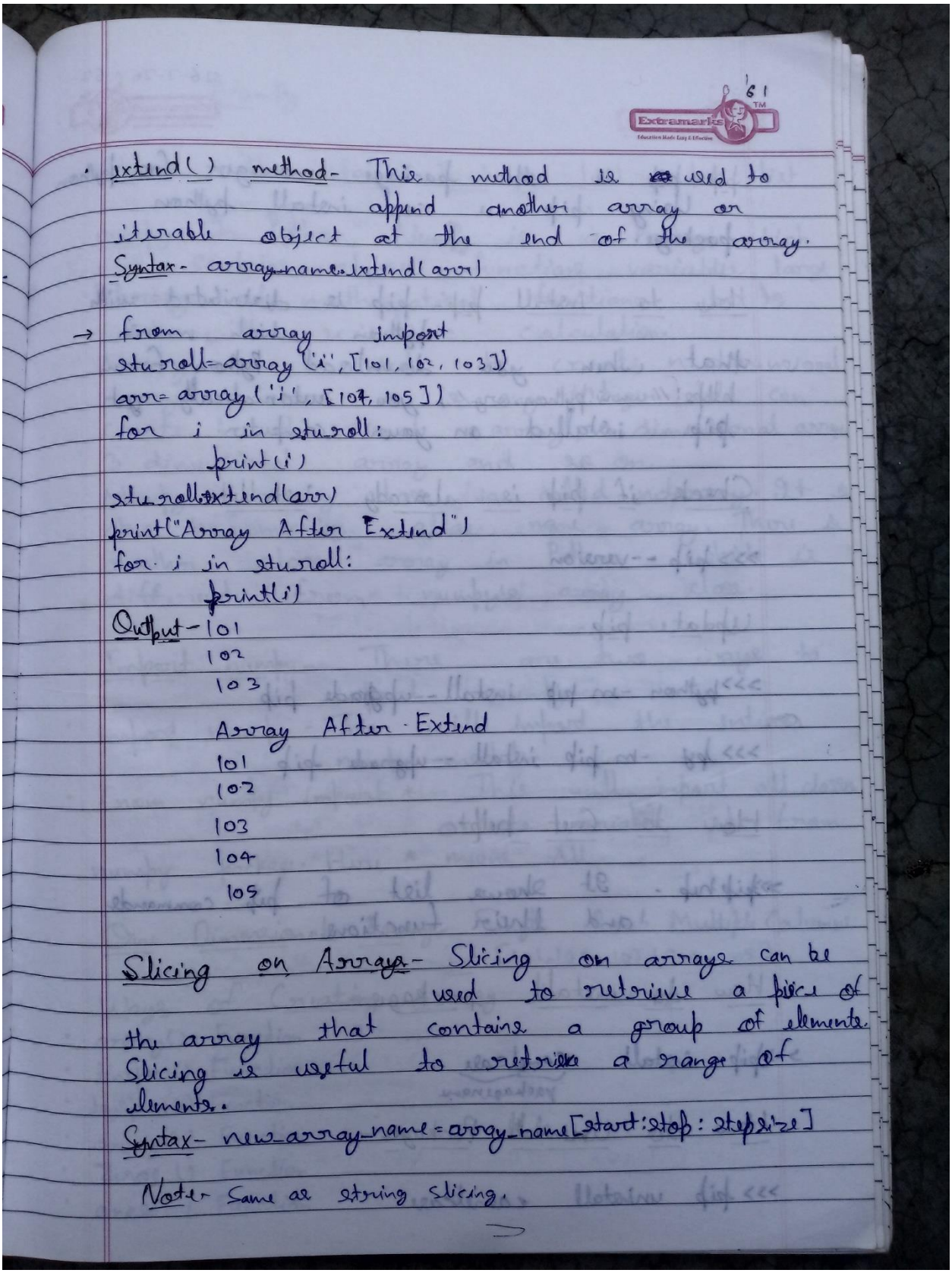


- index() method - This method returns position number of first occurrence of given element in the array. If it doesn't found the element, shows ValueError
Syntax - array_name.index(element)

→ from array import *
stu_roll = array('i', [101, 102, 103, 104, 105])
print(stu_roll.index(104))
Output - 3

- reverse() method - This method is used to reverse the order of elements in the array.
Syntax - array_name.reverse()

→ from array import *
stu_roll = array('i', [101, 102, 103, 104, 105])
for i in stu_roll:
    print(i)
print("Array After Reverse")
for i in stu_roll:
    print(i)
Output - 101
102
103
104
105
Array After Reverse
105
104
103
102
101



- extend() method- This method is used to append another array or iterable object at the end of the array.

Syntax- array_name.extend(arr)

→
```
from array import
stu_roll=array('i',[101,102,103])
arr=array('i',[104,105])
for i in stu_roll:
    print(i)
stu_roll.extend(arr)
print("Array After Extend")
for i in stu_roll:
    print(i)
```

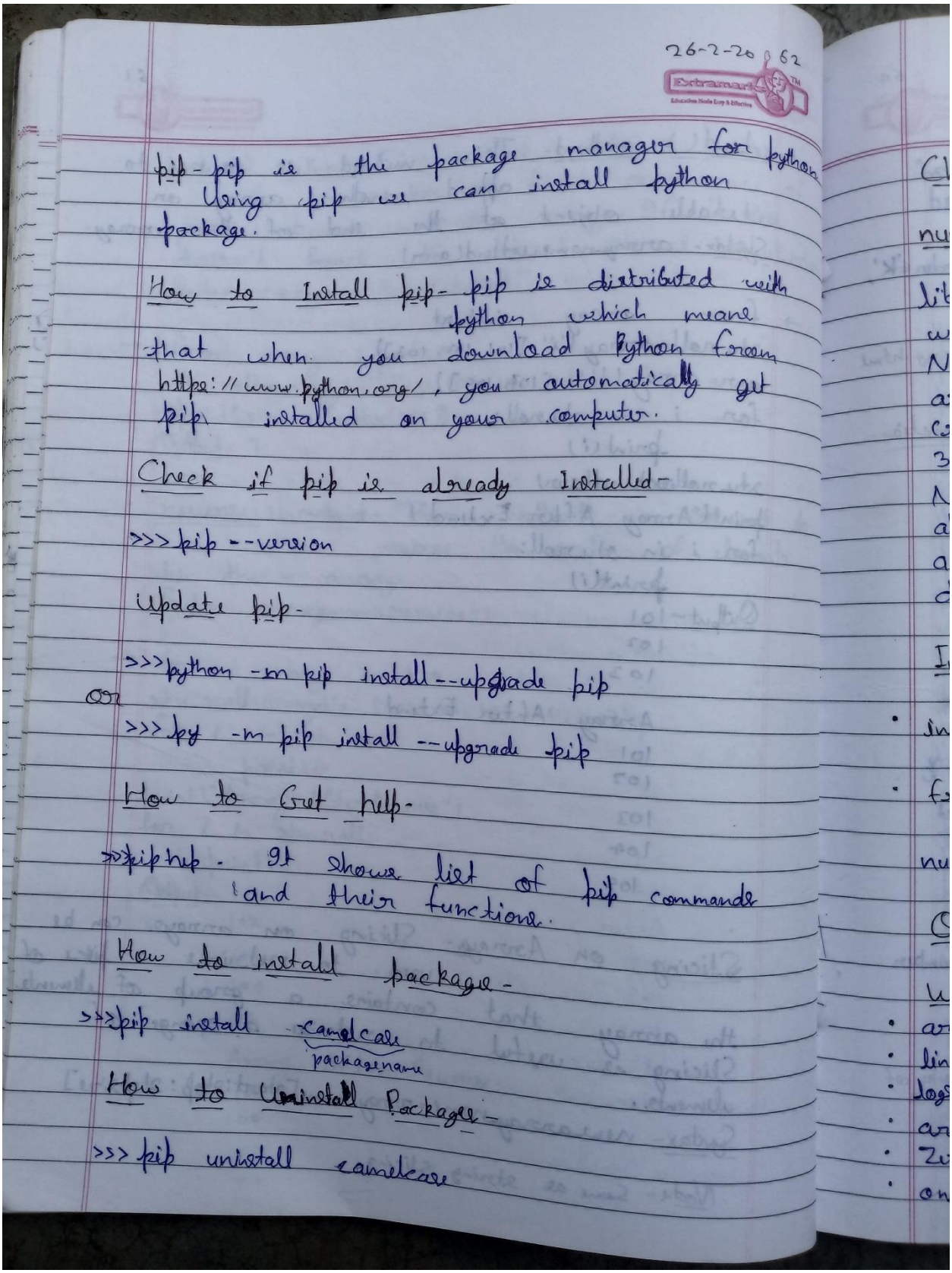
Output- 101
102
103

Array After Extend
101
102
103
104
105

Slicing on Arrays- Slicing on arrays can be used to retrieve a piece of the array that contains a group of elements. Slicing is useful to retrieve a range of elements.

Syntax- new_array_name=array_name[start:stop:stepsize]

Note- Same as string slicing.



26-2-20 62

pip - pip is the package manager for python. Using pip we can install python package.

How to Install pip - pip is distributed with python which means that when you download Python from https://www.python.org/, you automatically get pip installed on your computer.

Check if pip is already Installed -

>>> pip --version

Update pip -

>>> python -m pip install --upgrade pip

or

>>> py -m pip install --upgrade pip

How to Get help -

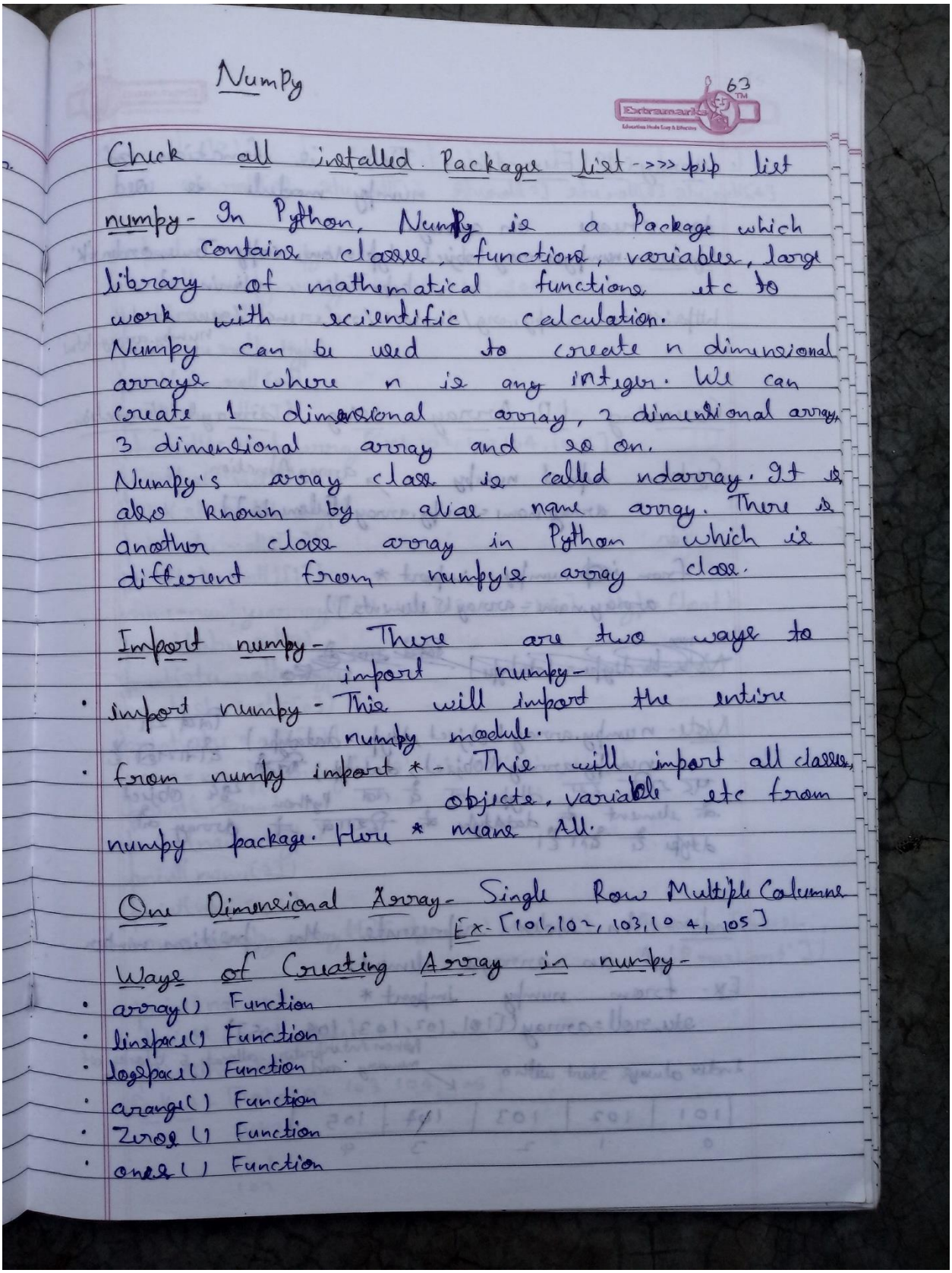
>>> pip help - It shows list of pip commands and their functions.

How to install package -

>>> pip install camelcase
(camelcase = package name)

How to Uninstall Packages -

>>> pip uninstall camelcase



Numpy

Check all installed Package list - >>> pip list

numpy - In Python, Numpy is a Package which contains classes, functions, variables, large library of mathematical functions etc to work with scientific calculation.
Numpy can be used to create n dimensional arrays where n is any integer. We can create 1 dimensional array, 2 dimensional array, 3 dimensional array and so on.
Numpy's array class is called ndarray. It is also known by alias name array. There is another class array in Python which is different from numpy's array class.

Import numpy - There are two ways to import numpy -

- import numpy - This will import the entire numpy module.
- from numpy import * - This will import all classes, objects, variable etc from numpy package. Here * means All.

One Dimensional Array - Single Row Multiple Columns
Ex - [101, 102, 103, 104, 105]

Ways of Creating Array in numpy -

- array() Function
- linspace() Function
- logspace() Function
- arange() Function
- Zeros() Function
- ones() Function

64

array() Function- This is function of numpy module is used to create an array.

Syntax- numpy.array(object, dtype=None, copy=True, order='K', subok=False, ndmin=0)

https://www.numpy.org/devdocs/reference/generated/numpy.array.html

Creating 1D Array using array() Function-

Syntax- import numpy
array_name = numpy.array([Elements])
↗ array function

or

from numpy import *
array_name = array([Elements])

~~Note- numpy.array(object, dtype=datatype)~~

Note- numpy.array(object, dtype=datatype) लिखें या numpy.array(object, datatype) लिखें दोनों सही है पर यदि हम dtype ना दें तब Python खुद object के element के datatype के हिसाब से array को dtype दे देता है।

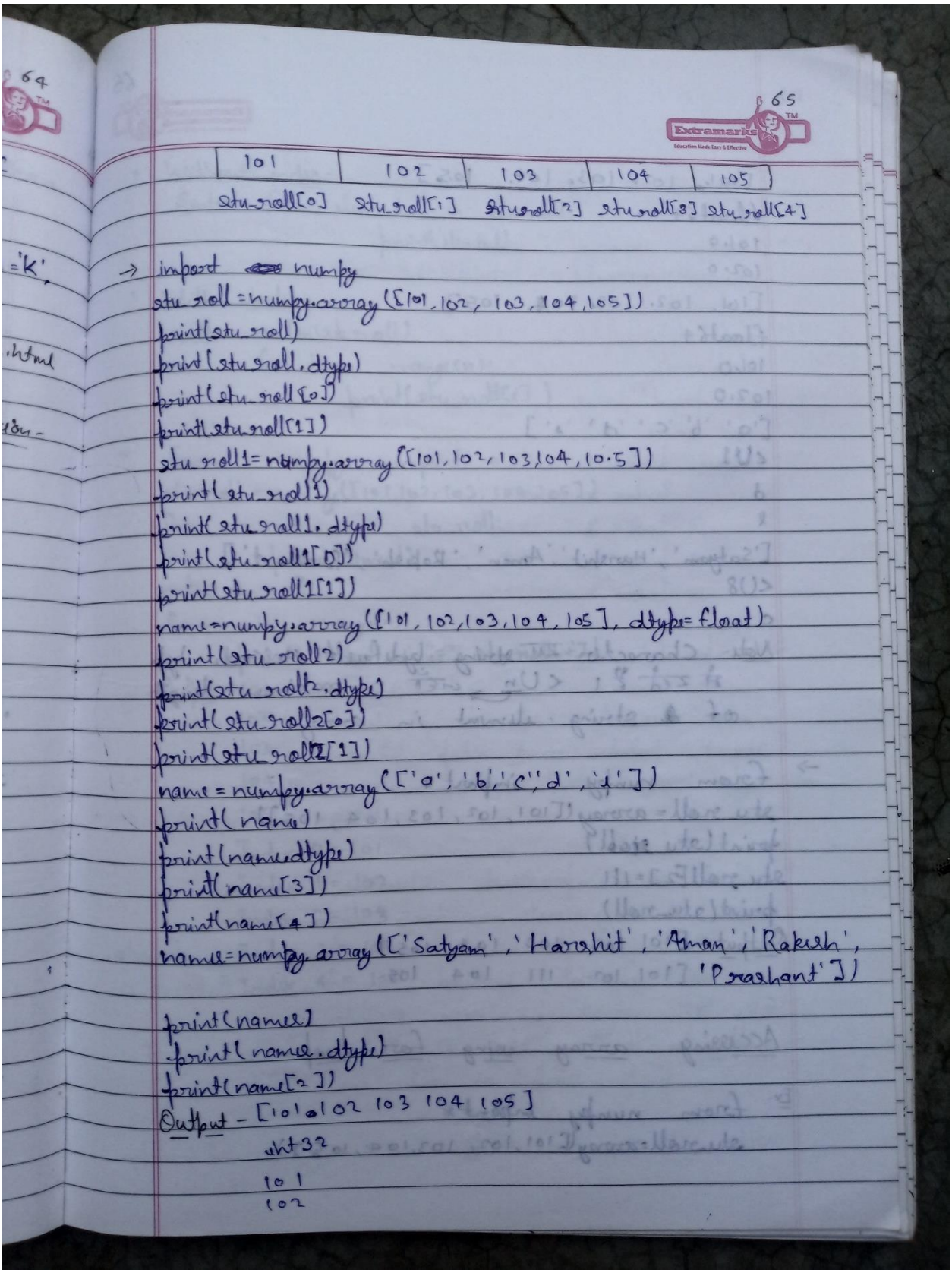
index- An index represents the position number of an array's element.

Ex- from numpy import *
stu_roll=array([101,102,103,104,105])

Index always start with 0

Python interpreter allocate 5 blocks of memory and store the elements

101	102	103	104	105
0	1	2	3	4



101	102	103	104	105
stu_roll[0]	stu_roll[1]	stu_roll[2]	stu_roll[3]	stu_roll[4]

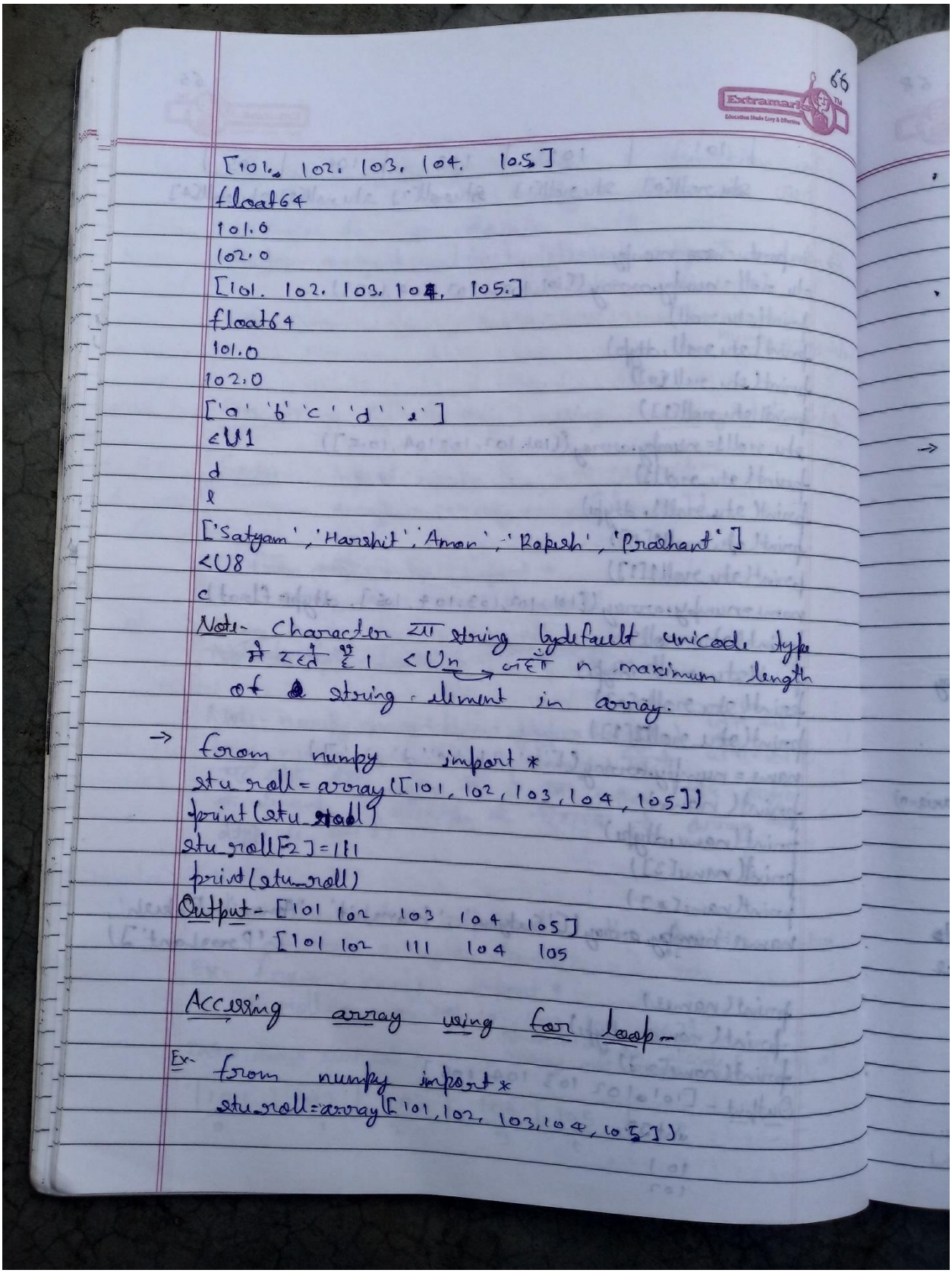
```
→ import numpy
stu_roll=numpy.array([101,102,103,104,105])
print(stu_roll)
print(stu_roll.dtype)
print(stu_roll[0])
print(stu_roll[1])
stu_roll1=numpy.array([101,102,103,104,105.5])
print(stu_roll1)
print(stu_roll1.dtype)
print(stu_roll1[0])
print(stu_roll1[1])
name=numpy.array([101,102,103,104,105], dtype=float)
print(stu_roll2)
print(stu_roll2.dtype)
print(stu_roll2[0])
print(stu_roll2[1])
name=numpy.array(['a','b','c','d','1'])
print(name)
print(name.dtype)
print(name[3])
print(name[4])
names=numpy.array(['Satyam','Harshit','Aman','Rakesh',
                   'Prashant'])

print(names)
print(names.dtype)
print(name[2])
```

Output - [101 102 103 104 105]

int32

101

102



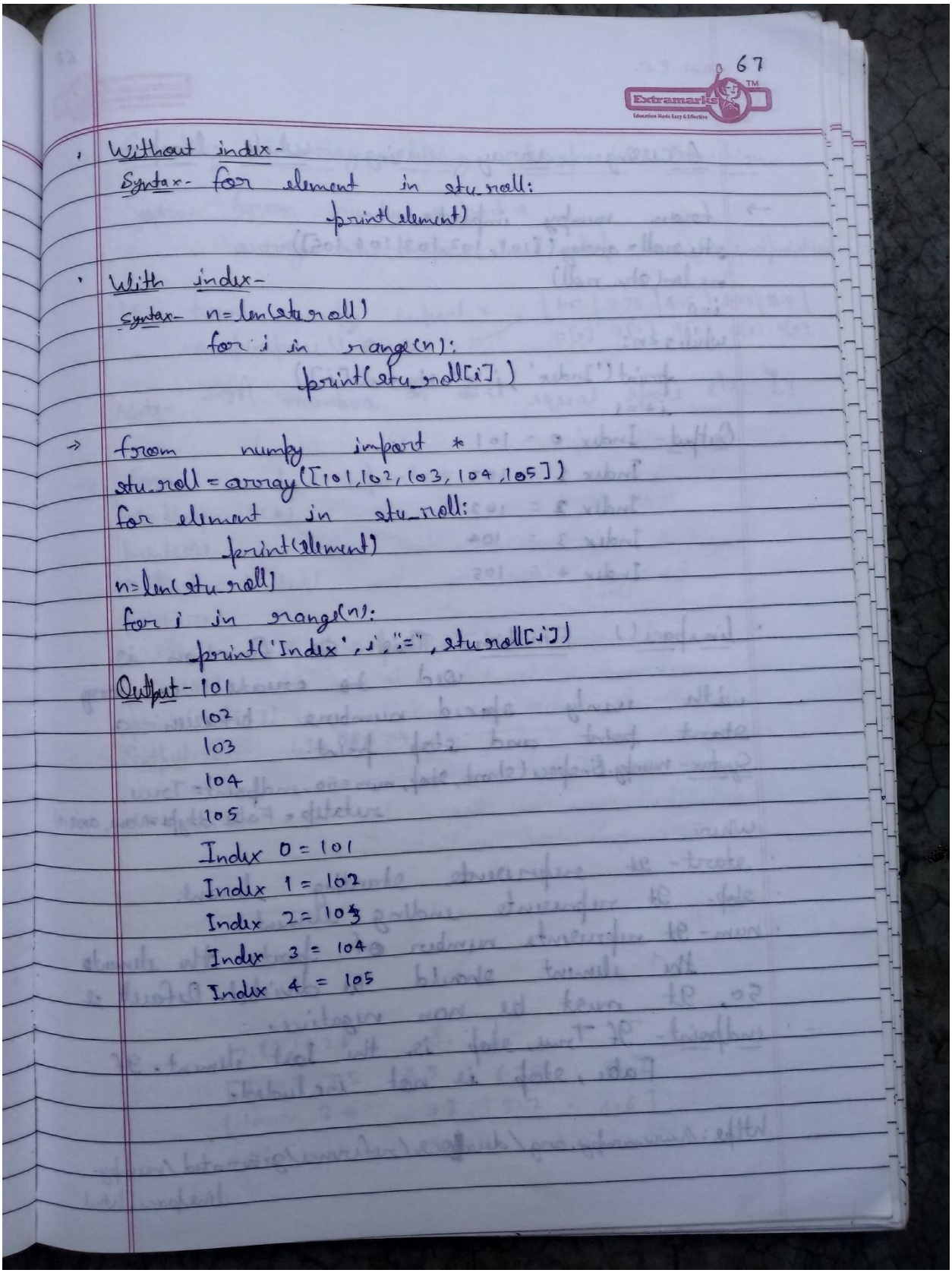
[101. 102. 103. 104. 105.]
float64
101.0
102.0
[101. 102. 103. 104. 105.]
float64
101.0
102.0
['a' 'b' 'c' 'd' 'e']
<U1
d
e
['Satyam' 'Harshit' 'Aman' 'Rakesh' 'Prashant']
<U8
c

Note- Character या string by default unicode type की होती है <Un → यहाँ n maximum length of a string element in array.

→ from numpy import *
stu_roll = array([101, 102, 103, 104, 105])
print(stu_roll)
stu_roll[2] = 111
print(stu_roll)
Output- [101 102 103 104 105]
[101 102 111 104 105]

Accessing array using for loop-

Ex- from numpy import *
stu_roll = array([101, 102, 103, 104, 105])



67

Extramarks

- Without index -

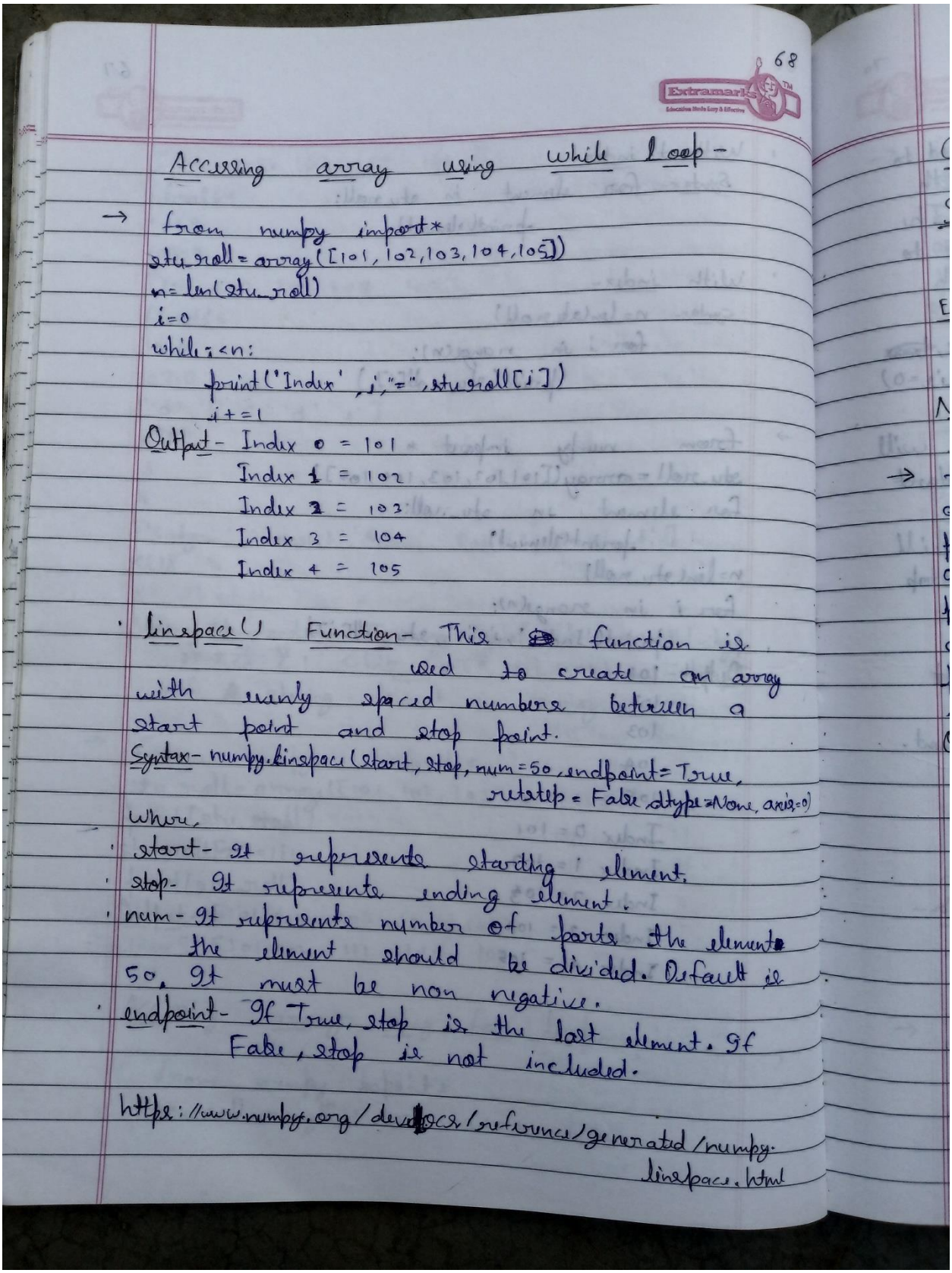
Syntax - for element in stu_roll:
print(element)

- With index -

Syntax - n=len(stu_roll)
for i in range(n):
print(stu_roll[i])

→ from numpy import *
stu_roll = array([101,102,103,104,105])
for element in stu_roll:
print(element)
n=len(stu_roll)
for i in range(n):
print('Index',i,":=",stu_roll[i])

Output - 101
102
103
104
105
Index 0 = 101
Index 1 = 102
Index 2 = 103
Index 3 = 104
Index 4 = 105



68

Accessing array using while loop -

→
```
from numpy import*
stu_roll = array([101,102,103,104,105])
n = len(stu_roll)
i=0
while i<n:
    print('Index',i,"=",stu_roll[i])
    i+=1
```

Output - Index 0 = 101
Index 1 = 102
Index 2 = 103
Index 3 = 104
Index 4 = 105

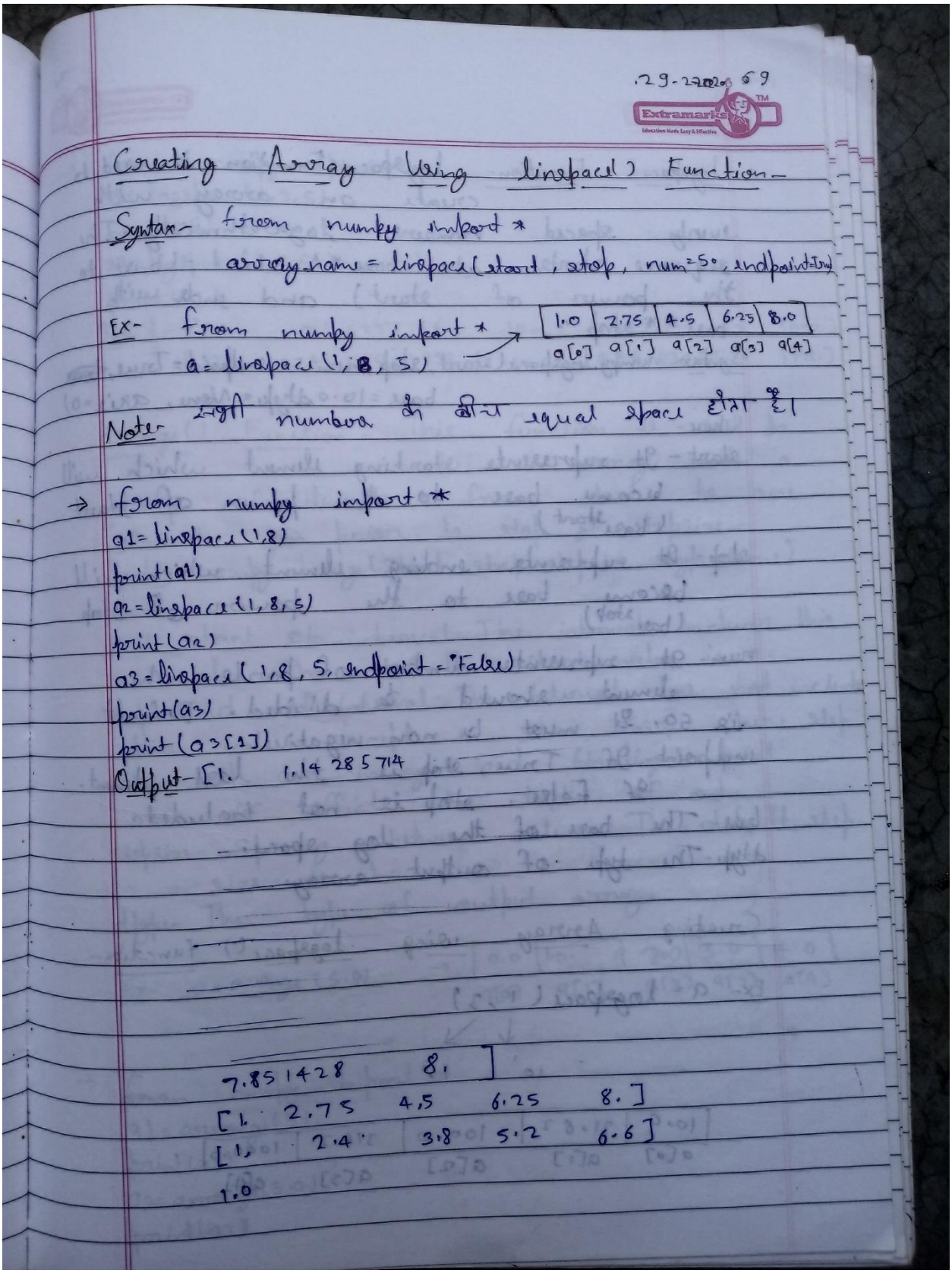
- linspace() Function - This function is used to create an array with evenly spaced numbers between a start point and stop point.

Syntax - numpy.linspace(start, stop, num=50, endpoint=True, retstep=False, dtype=None, axis=0)

Where,

- start - It represents starting element.
- stop - It represents ending element.
- num - It represents number of parts the elements the element should be divided. Default is 50. It must be non negative.
- endpoint - If True, stop is the last element. If False, stop is not included.

https://www.numpy.org/devdocs/reference/generated/numpy.linspace.html



Creating Array Using linspace() Function-

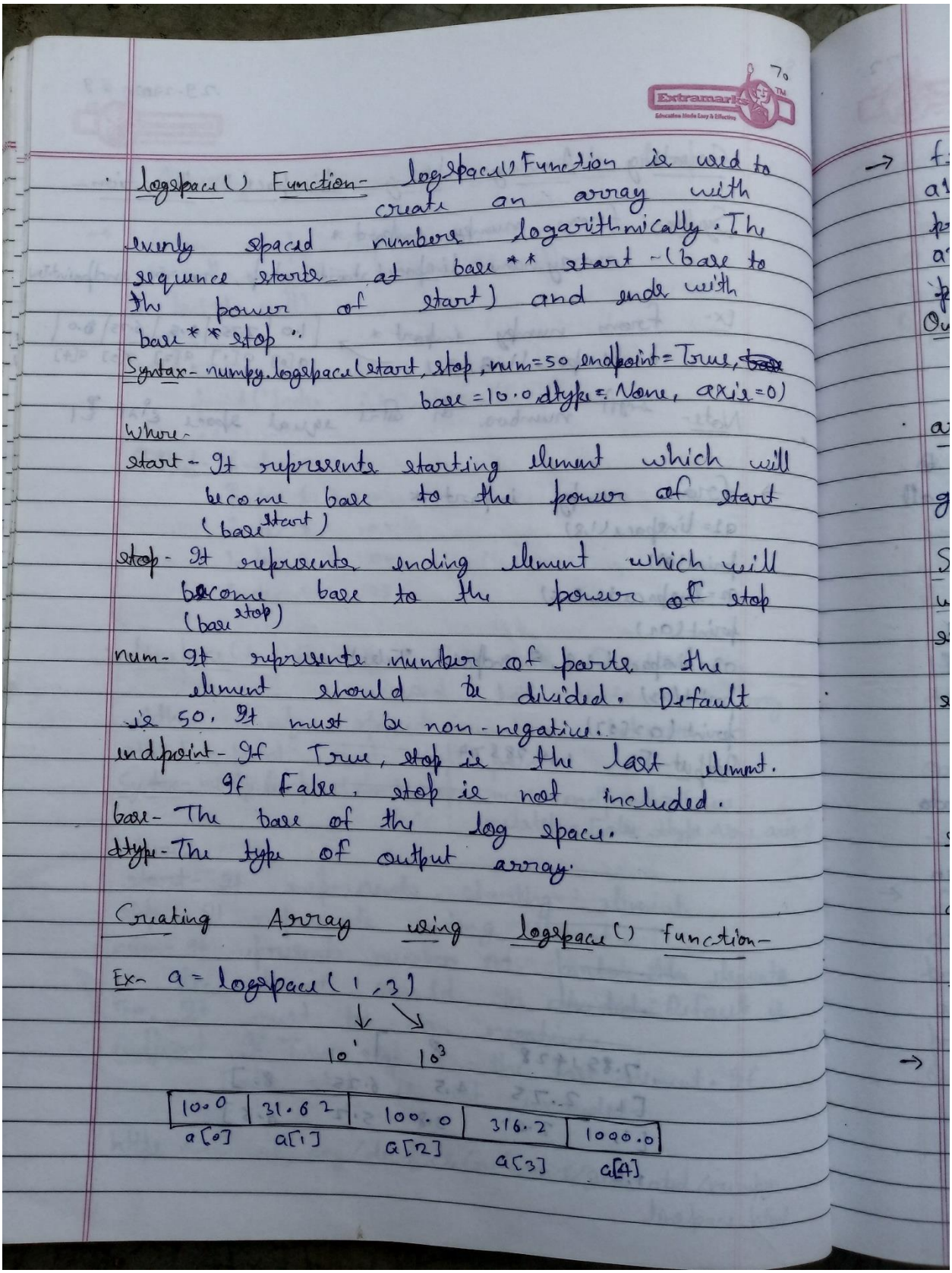
Syntax- from numpy import *
array_name = linspace(start, stop, num=50, endpoint=True)

Ex- from numpy import *
a = linspace(1, 8, 5)

1.0	2.75	4.5	6.25	8.0
a[0]	a[1]	a[2]	a[3]	a[4]

Note- सभी numbers के बीच equal space होता है।

→ from numpy import *
a1 = linspace(1,8)
print(a1)
a2 = linspace(1, 8, 5)
print(a2)
a3 = linspace(1,8, 5, endpoint = "False")
print(a3)
print(a3[1])
Output- [1. 1.14285714
7.85 1428 8.]
[1. 2.75 4.5 6.25 8.]
[1. 2.4 3.8 5.2 6.6]
1.0



- **logspace() Function-** logspace() Function is used to create an array with evenly spaced numbers logarithmically. The sequence starts at base ** start - (base to the power of start) and ends with base ** stop.

Syntax- numpy.logspace(start, stop, num=50, endpoint=True, base=10.0, dtype=None, axis=0)

Where-

start - It represents starting element which will become base to the power of start ($base^{start}$)

stop - It represents ending element which will become base to the power of stop ($base^{stop}$)

num - It represents number of parts the element should be divided. Default is 50. It must be non-negative.

endpoint - If True, stop is the last element. If False, stop is not included.

base - The base of the log space.

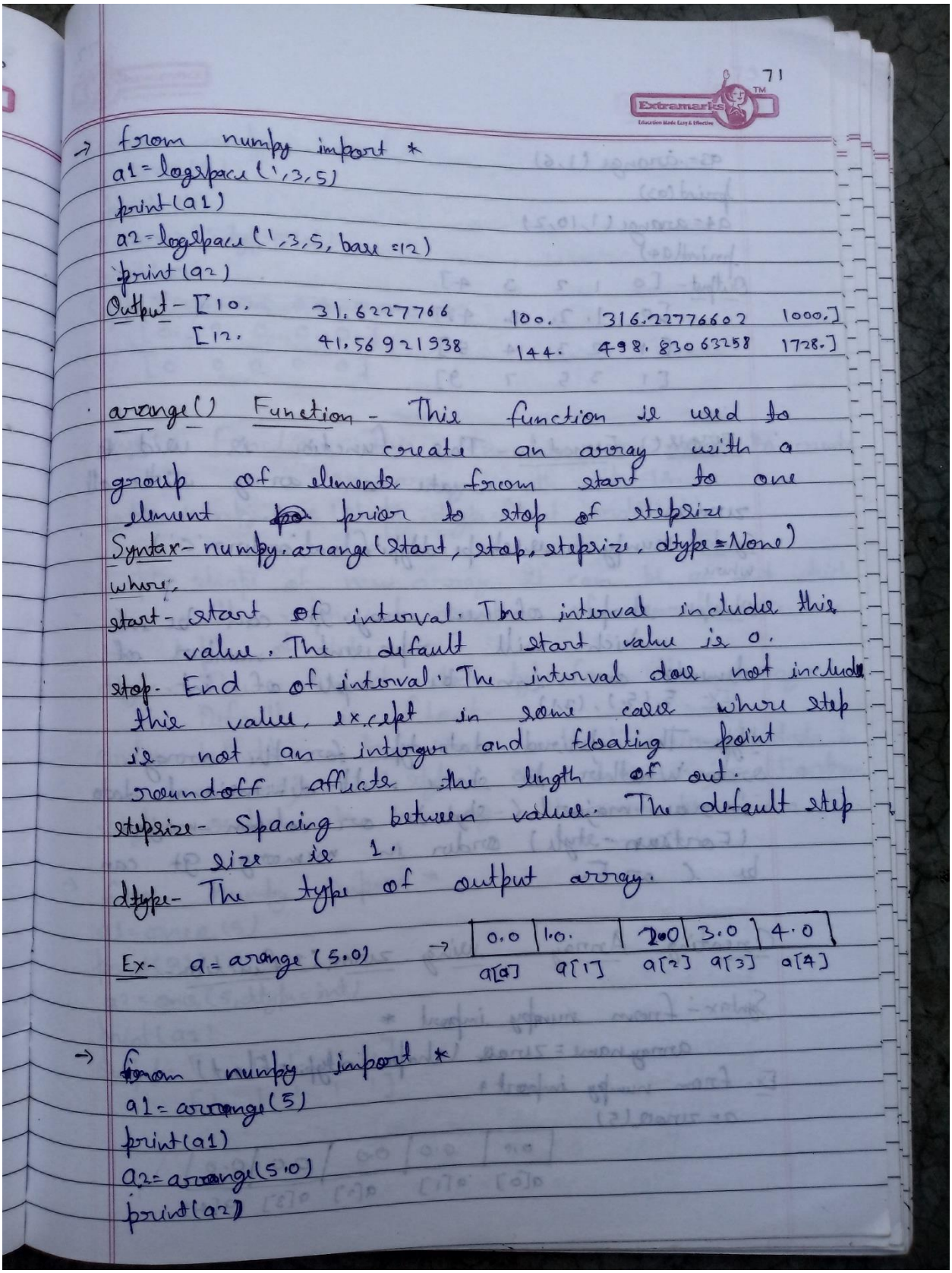
dtype - The type of output array.

Creating Array using logspace() Function-

Ex- a = logspace(1, 3)

10^1 10^3

10.0	31.62	100.0	316.2	1000.0
a[0]	a[1]	a[2]	a[3]	a[4]



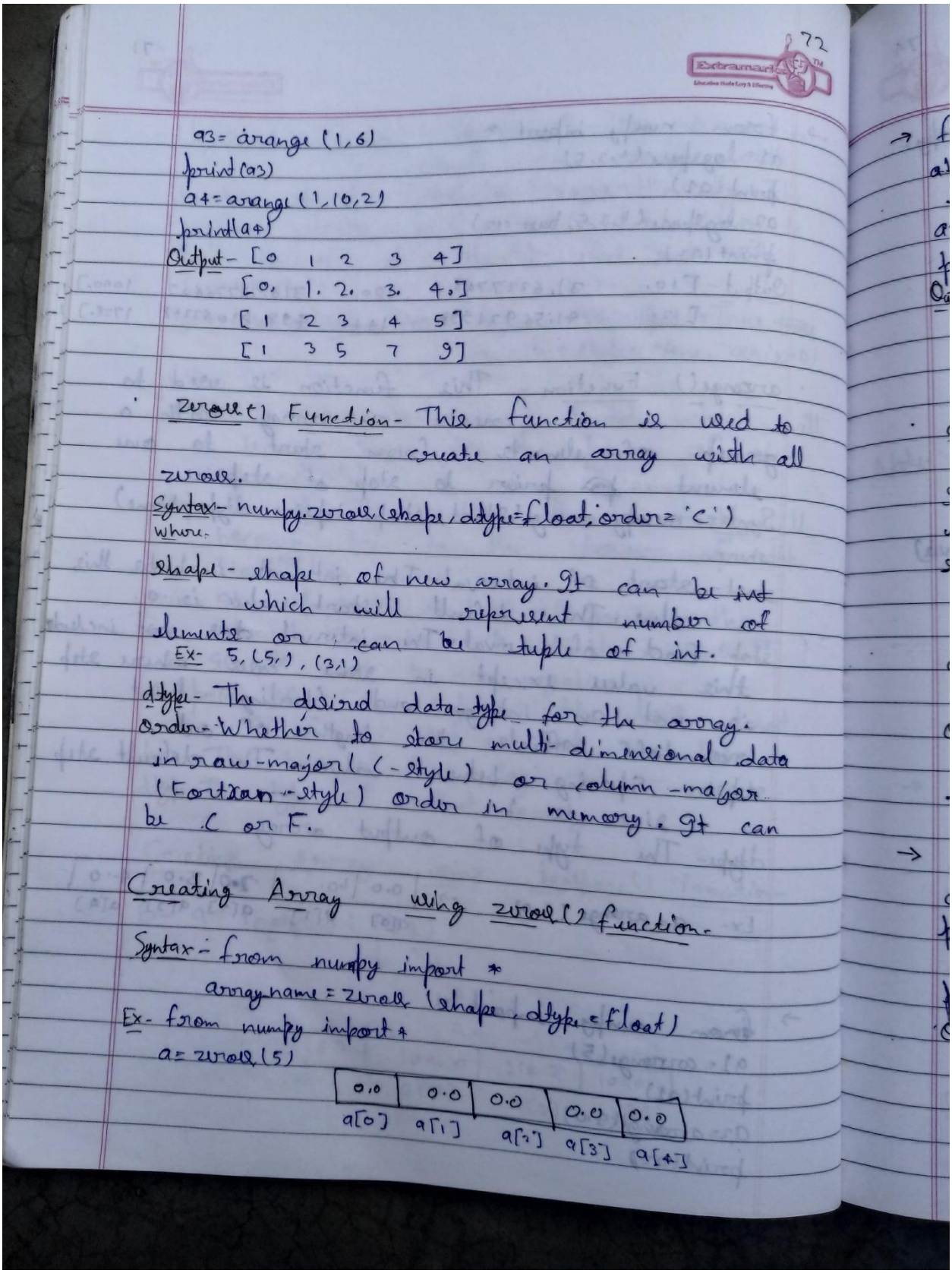
71

→ from numpy import *
a1 = logspace(1,3,5)
print(a1)
a2 = logspace(1,3,5, base=12)
print(a2)
Output - [10. 31.6227766 100. 316.22776602 1000.]
[12. 41.56921938 144. 498.83063258 1728.]

arange() Function - This function is used to create an array with a group of elements from start to one element prior to stop of stepsize
Syntax - numpy.arange(start, stop, stepsize, dtype=None)
where,
start - start of interval. The interval includes this value. The default start value is 0.
stop - End of interval. The interval does not include this value, except in some cases where step is not an integer and floating point roundoff affects the length of out.
stepsize - Spacing between values. The default step size is 1.
dtype - The type of output array.

Ex - a = arange(5.0) → | 0.0 | 1.0 | 2.0 | 3.0 | 4.0 |
a[0] a[1] a[2] a[3] a[4]

→ from numpy import *
a1 = arange(5)
print(a1)
a2 = arange(5.0)
print(a2)



```
a3 = arange(1,6)
print(a3)
a4 = arange(1,10,2)
print(a4)
Output - [0 1 2 3 4]
         [0. 1. 2. 3. 4.]
         [1 2 3 4 5]
         [1 3 5 7 9]
```

zeros() Function - This function is used to create an array with all zeros.

Syntax - numpy.zeros(shape, dtype=float, order='C')

Where.

shape - shape of new array. It can be int which will represent number of elements or can be tuple of int.
Ex- 5, (5,), (3,1)

dtype - The desired data-type for the array.

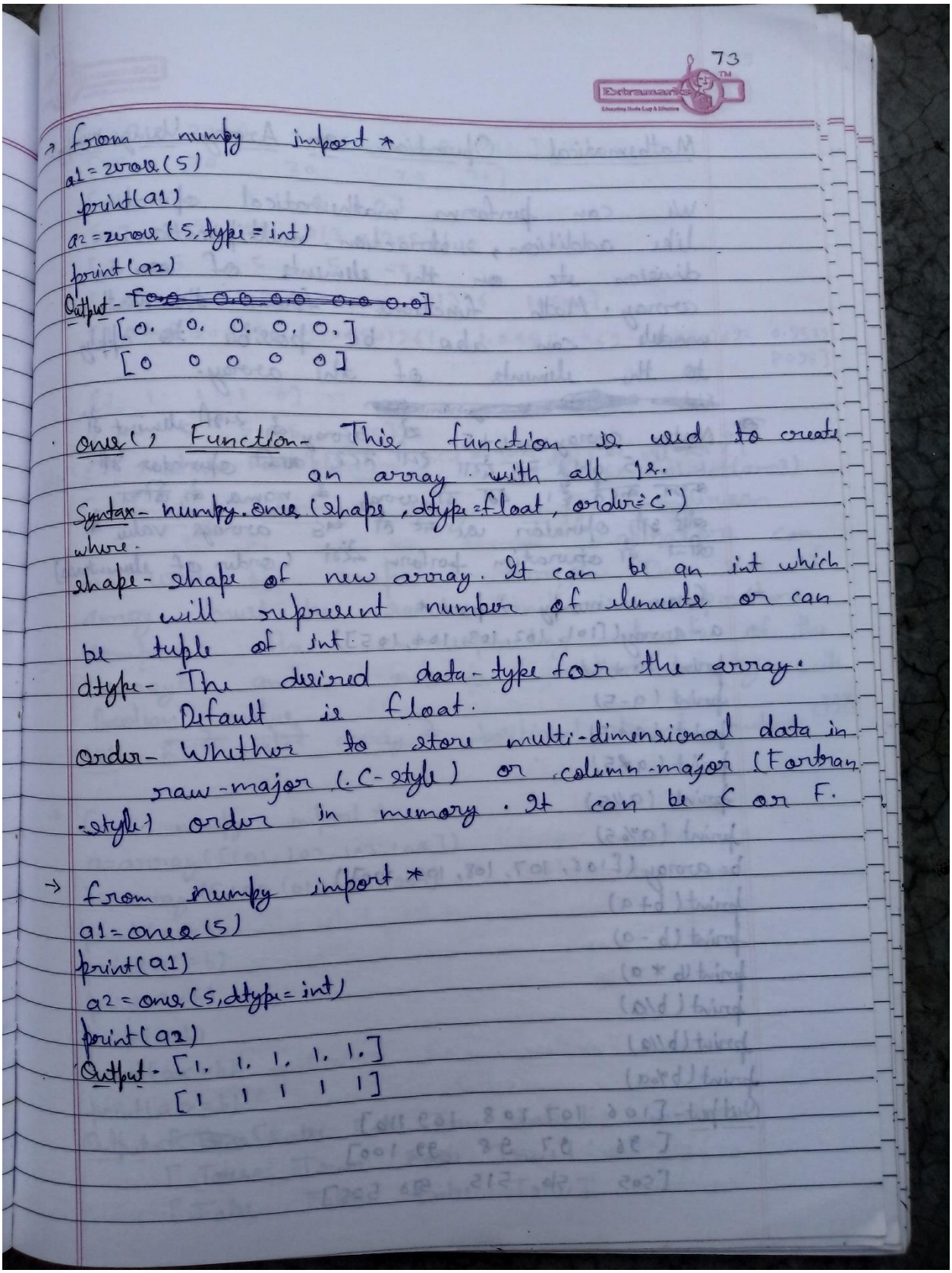
order - Whether to store multi-dimensional data in row-major (C-style) or column-major (Fortran-style) order in memory. It can be C or F.

Creating Array using zeros() function.

Syntax - from numpy import *
arrayname = zeros(shape, dtype=float)

Ex - from numpy import *
a = zeros(5)

0.0	0.0	0.0	0.0	0.0
a[0]	a[1]	a[2]	a[3]	a[4]

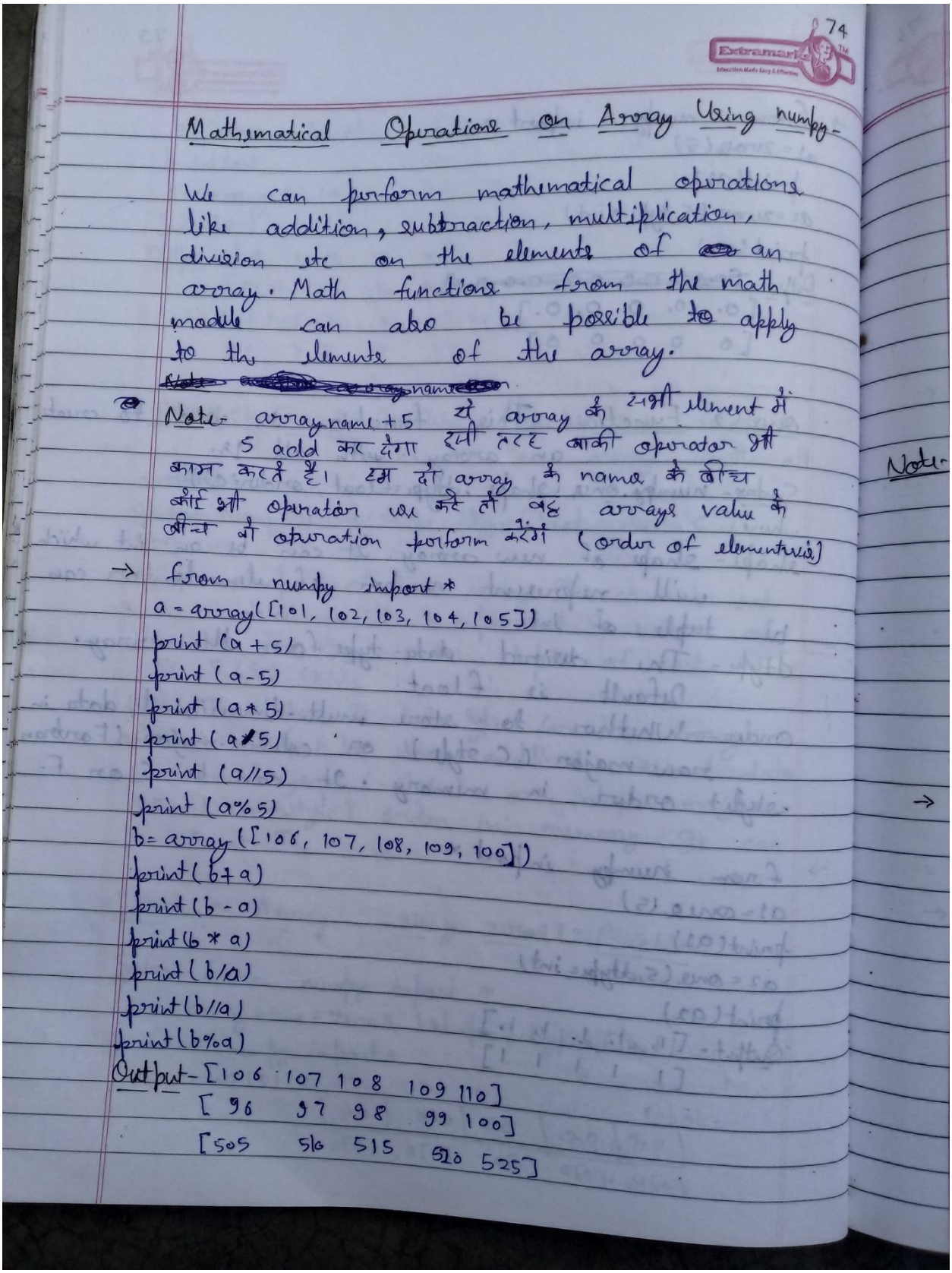


73

→ from numpy import *
a1 = zeros(5)
print(a1)
a2 = zeros(5, dtype = int)
print(a2)
Output - [0. 0. 0. 0. 0.]
[0 0 0 0 0]

· ones() Function - This function is used to create an array with all 1s.
Syntax - numpy.ones(shape, dtype=float, order='C')
where.
shape - shape of new array. It can be an int which will represent number of elements or can be tuple of int.
dtype - The desired data-type for the array. Default is float.
order - Whether to store multi-dimensional data in raw-major (C-style) or column-major (Fortran-style) order in memory. It can be C or F.

→ from numpy import *
a1 = ones(5)
print(a1)
a2 = ones(5, dtype = int)
print(a2)
Output - [1. 1. 1. 1. 1.]
[1 1 1 1 1]



74

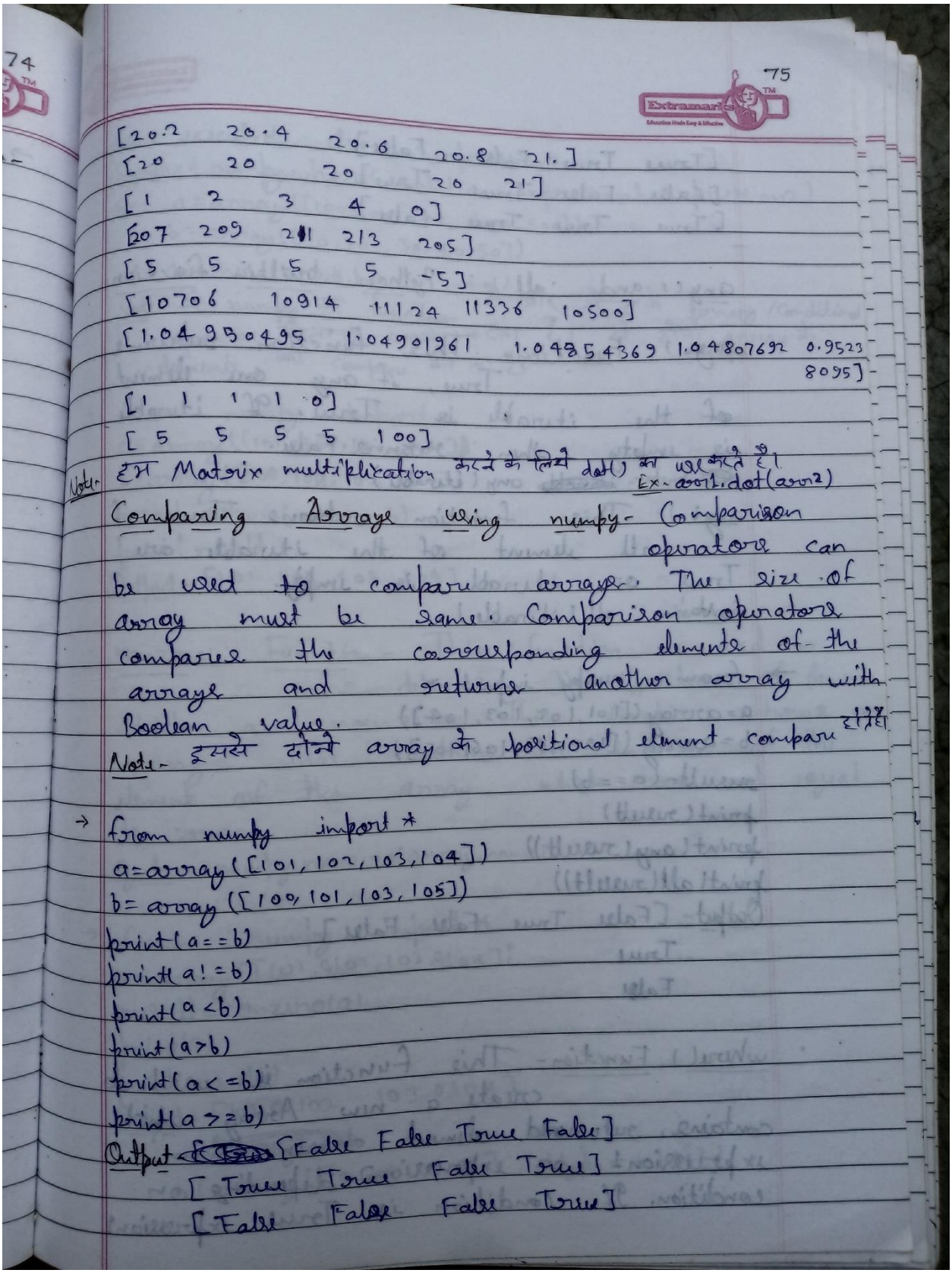
Mathematical Operations on Array Using numpy-

We can perform mathematical operations like addition, subtraction, multiplication, division etc on the elements of an array. Math functions from the math module can also be possible to apply to the elements of the array.

Note- arrayname + 5 से array के हर element में 5 add कर देगा इसी तरह बाकी operator भी काम करते है। हम दो array के name के बीच कोई भी operation भी करे तो वह arrays value के बीच में operation perform करेंगे (order of elements)

→
```
from numpy import *
a = array([101, 102, 103, 104, 105])
print (a+5)
print (a-5)
print (a*5)
print (a**5)
print (a//5)
print (a%5)
b = array([106, 107, 108, 109, 100])
print (b+a)
print (b-a)
print (b*a)
print (b/a)
print (b//a)
print (b%a)
```

Output- [106 107 108 109 110]
[96 97 98 99 100]
[505 516 515 520 525]



[20.2 20.4 20.6 20.8 21.]
[20 20 20 20 21]
[1 2 3 4 0]
[207 209 211 213 205]
[5 5 5 5 -5]
[10706 10914 11124 11336 10500]
[1.04950495 1.04901961 1.04854369 1.04807692 0.95238095]
[1 1 1 1 0]
[5 5 5 5 100]

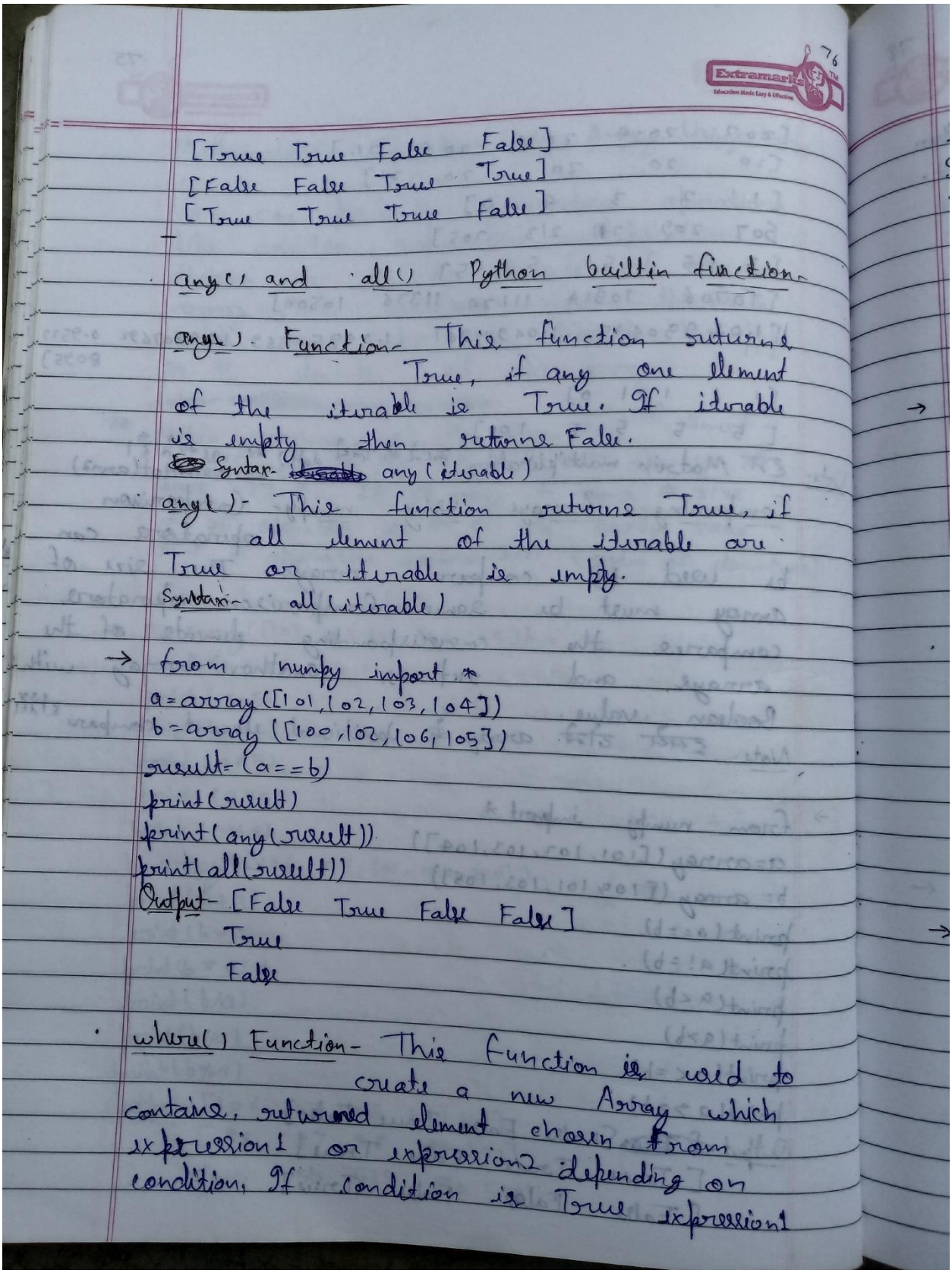
Note- ⓔⓧ Matrix multiplication करने के लिये dot() का use करते है।
Ex- arr1.dot(arr2)

Comparing Arrays using numpy- Comparison operators can be used to compare arrays. The size of array must be same. Comparison operators compares the corresponding elements of the arrays and returns another array with Boolean value.

Note- इससे दोनों array के positional element compare होते है।

→
```
from numpy import *
a=array([101,102,103,104])
b=array([100,101,103,105])
print(a==b)
print(a!=b)
print(a<b)
print(a>b)
print(a<=b)
print(a>=b)
```

Output- [False False True False]
[True True False True]
[False False False True]



[True True False False]
[False False True True]
[True True True False]

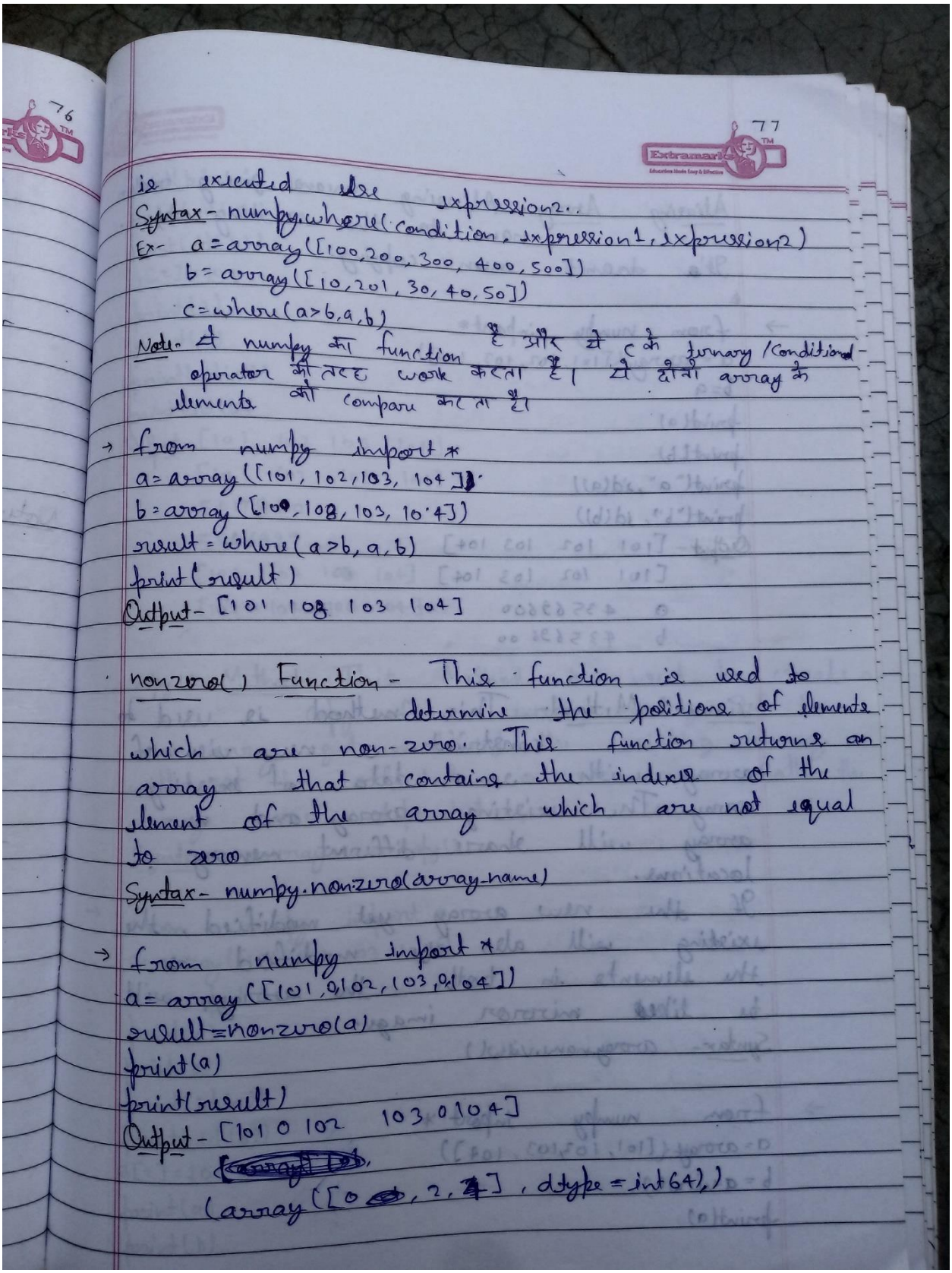
any() and all() Python builtin function-

any() Function- This function returns True, if any one element of the iterable is True. If iterable is empty then returns False.
Syntax- any(iterable)

any()- This function returns True, if all element of the iterable are True or iterable is empty.
Syntax- all(iterable)

→ from numpy import *
a=array([101,102,103,104])
b=array([100,102,106,105])
result=(a==b)
print(result)
print(any(result))
print(all(result))
Output- [False True False False]
True
False

where() Function- This function is used to create a new Array which contains returned element chosen from expression1 or expression2 depending on condition. If condition is True expression1

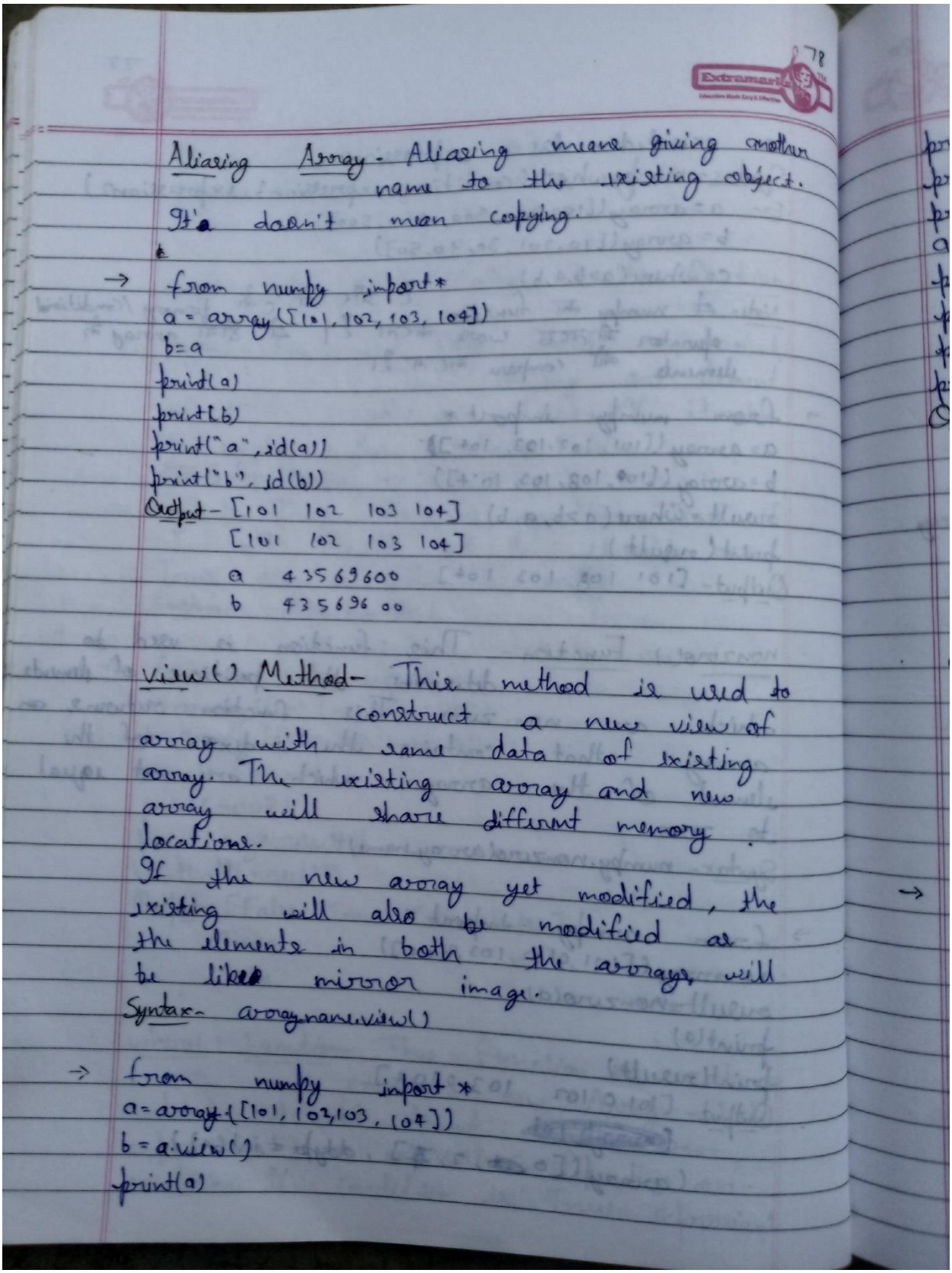


is executed else expression2.
Syntax- numpy.where(condition, expression1, expression2)
Ex- a=array([100,200,300,400,500])
b=array([10,201,30,40,50])
c=where(a>b,a,b)
Note- यह numpy का function है और यह C के ternary/conditional operator की तरह work करता है। यह दोनों array के elements को compare करता है।

→ from numpy import *
a=array([101,102,103,104])
b=array([100,108,103,104])
result=where(a>b,a,b)
print(result)
Output- [101 108 103 104]

nonzero() Function- This function is used to determine the positions of elements which are non-zero. This function returns an array that contains the indexes of the element of the array which are not equal to zero
Syntax- numpy.nonzero(array_name)

→ from numpy import *
a=array([101,0,102,103,0,104])
result=nonzero(a)
print(a)
print(result)
Output- [101 0 102 103 0 104]
(array([0, 2, 3], dtype=int64),)

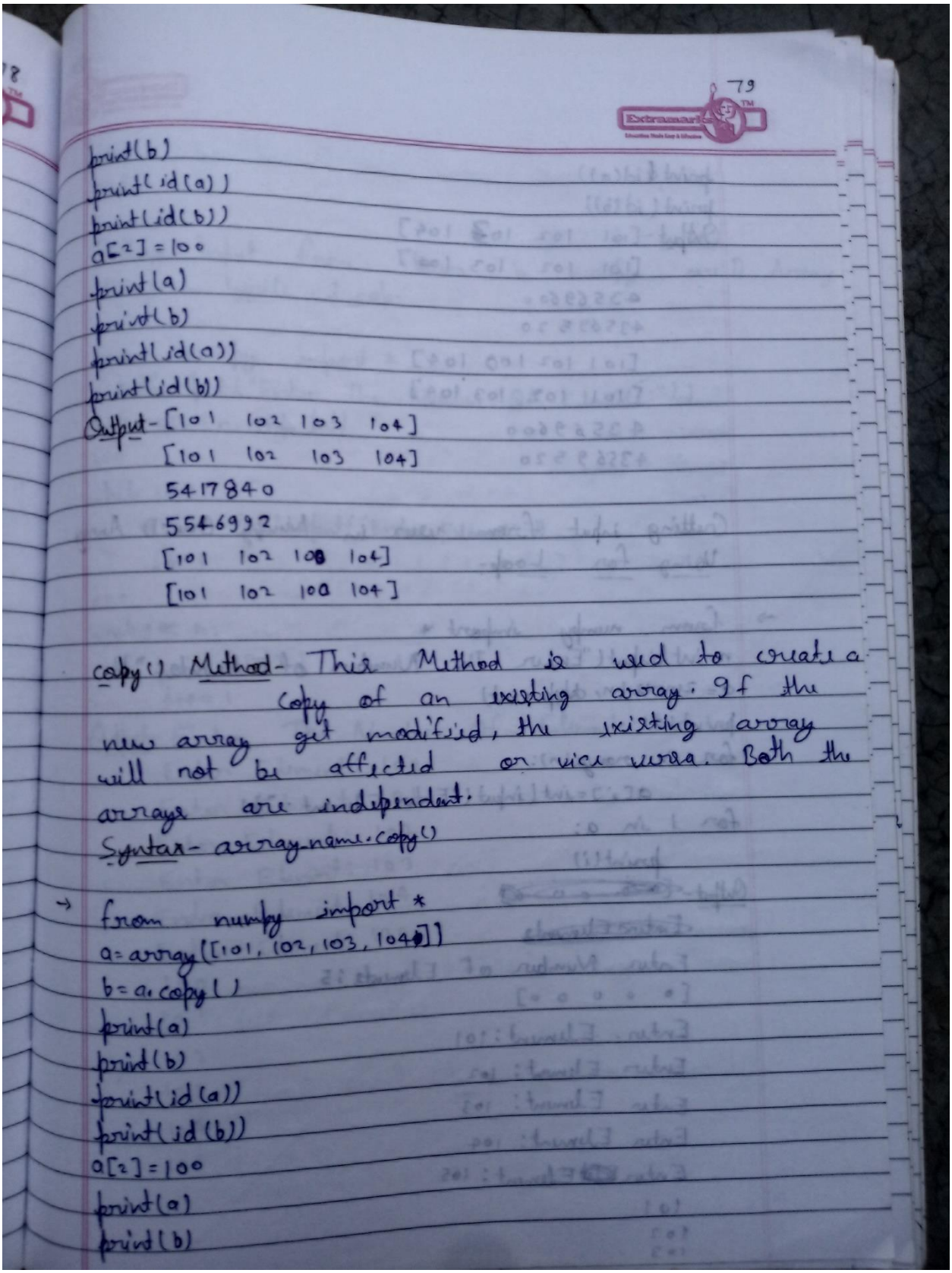


Aliasing Array - Aliasing means giving another name to the existing object. It doesn't mean copying.

→
```
from numpy import *
a = array([101, 102, 103, 104])
b=a
print(a)
print(b)
print("a", id(a))
print("b", id(b))
```
Output - [101 102 103 104]
[101 102 103 104]
a 43569600
b 43569600

view() Method - This method is used to construct a new view of array with same data of existing array. The existing array and new array will share diffrent memory locations.
If the new array yet modified, the existing will also be modified as the elements in both the arrays will be like mirror image.
Syntax - arrayname.view()

→
```
from numpy import *
a=array([101, 102, 103, 104])
b=a.view()
print(a)
```



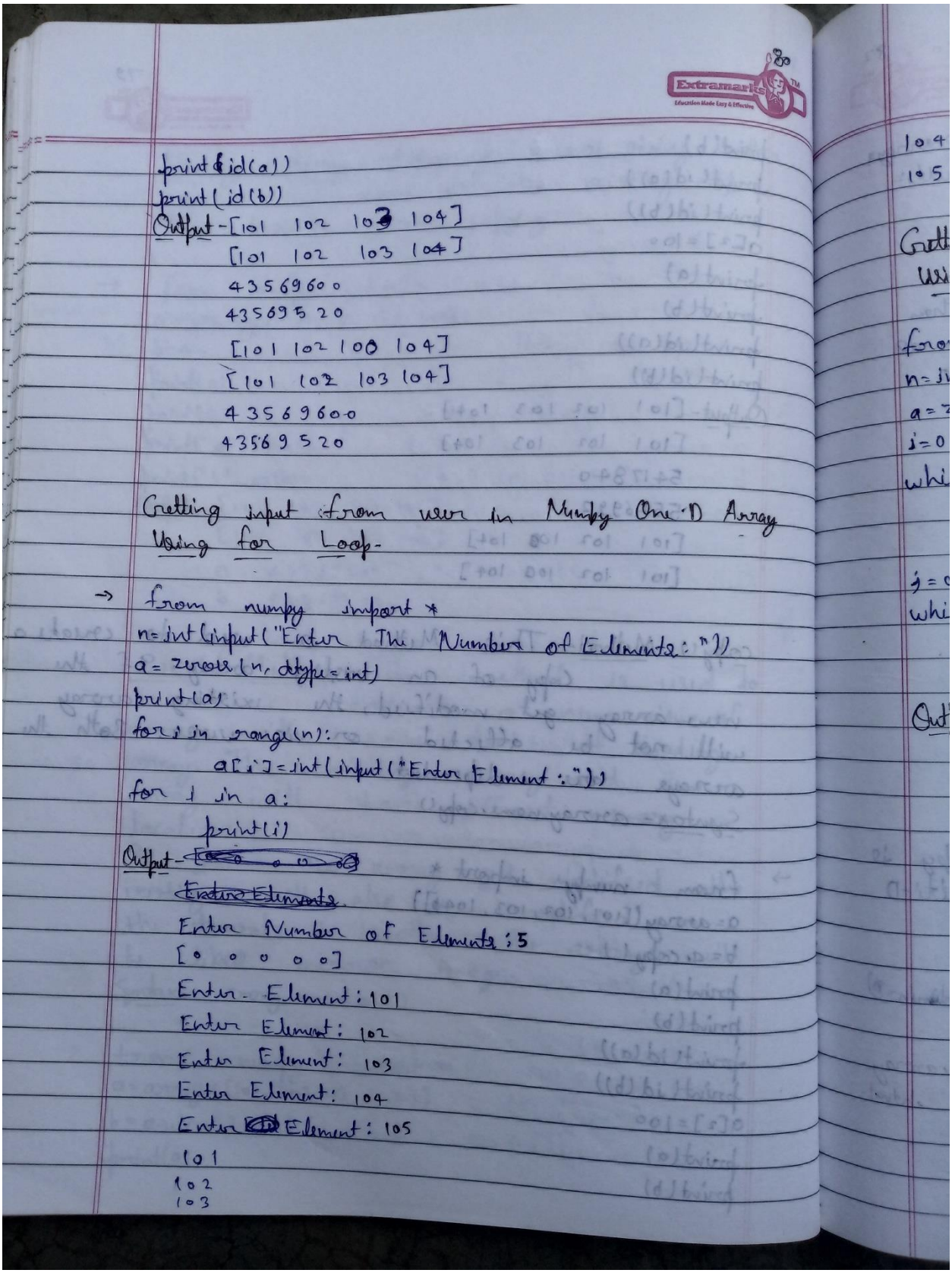
```
print(b)
print(id(a))
print(id(b))
a[2] = 100
print(a)
print(b)
print(id(a))
print(id(b))
```

Output- [101 102 103 104]
[101 102 103 104]
5417840
5546992
[101 102 100 104]
[101 102 100 104]

copy() Method- This Method is used to create a copy of an existing array. If the new array get modified, the existing array will not be affected or vice versa. Both the arrays are independent.

Syntax- array_name.copy()

→
```
from numpy import *
a=array([101, 102, 103, 104])
b=a.copy()
print(a)
print(b)
print(id(a))
print(id(b))
a[2]=100
print(a)
print(b)
```



```
print(id(a))
print(id(b))
Output - [101  102  103  104]
         [101  102  103  104]
         43569600
         43569520
         [101 102 100 104]
         [101  102  103 104]
         43569600
         43569520
```

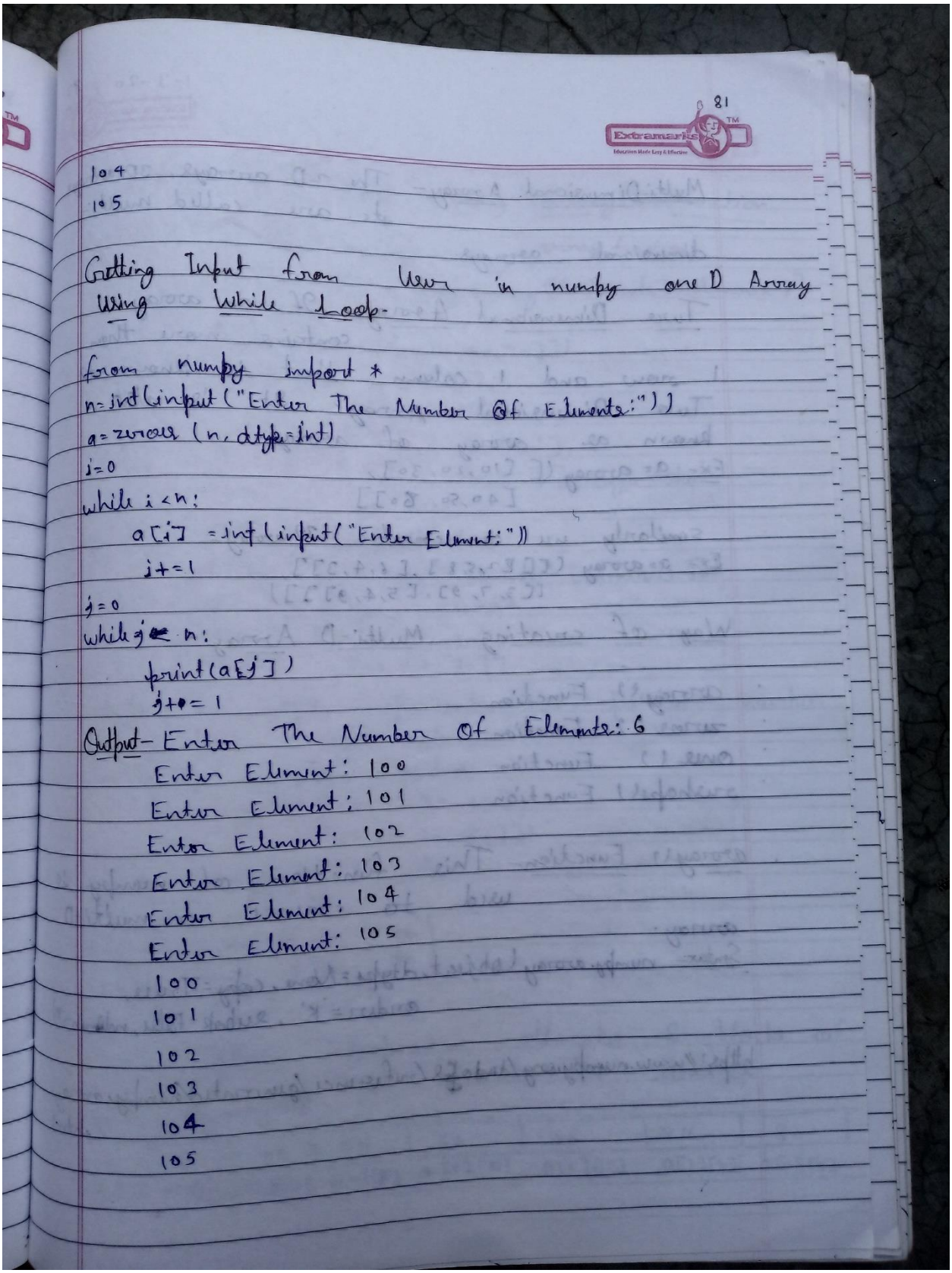
Getting input from user in Numpy One-D Array Using for Loop-

→

```
from numpy import *
n=int(input("Enter The Number of Elements: "))
a=zeros(n, dtype=int)
print(a)
for i in range(n):
    a[i]=int(input("Enter Element: "))
for i in a:
    print(i)
```

Output -

```
Enter Number of Elements: 5
[0 0 0 0 0]
Enter Element: 101
Enter Element: 102
Enter Element: 103
Enter Element: 104
Enter Element: 105
101
102
103
```

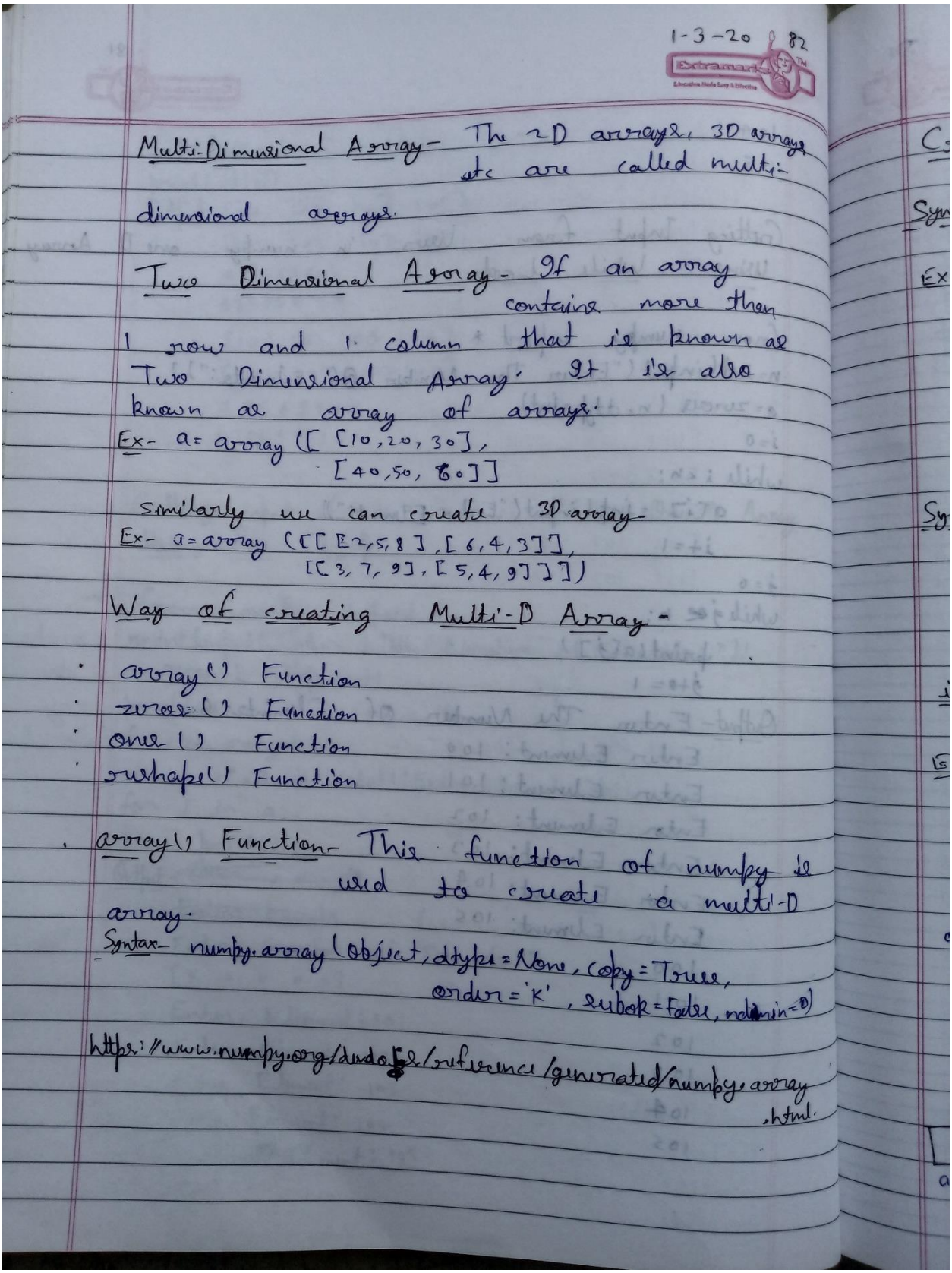


104
105

Getting Input from User in numpy one D Array Using While Loop.

```
from numpy import *
n=int(input("Enter The Number Of Elements:"))
a=zeros(n,dtype=int)
i=0
while i<n:
    a[i]=int(input("Enter Element:"))
    i+=1
j=0
while j<n:
    print(a[j])
    j+=1
```

Output- Enter The Number Of Elements: 6
Enter Element: 100
Enter Element: 101
Enter Element: 102
Enter Element: 103
Enter Element: 104
Enter Element: 105
100
101
102
103
104
105



1-3-20 82

Multi-Dimensional Array- The 2D arrays, 3D arrays etc are called multi-dimensional arrays.

Two Dimensional Array- If an array contains more than 1 row and 1 column that is known as Two Dimensional Array. It is also known as array of arrays.

Ex- a= array ([[10,20,30],
[40,50,60]]

Similarly we can create 3D array-

Ex- a=array ([[[2,5,8],[6,4,3]],
[[3,7,9],[5,4,9]]])

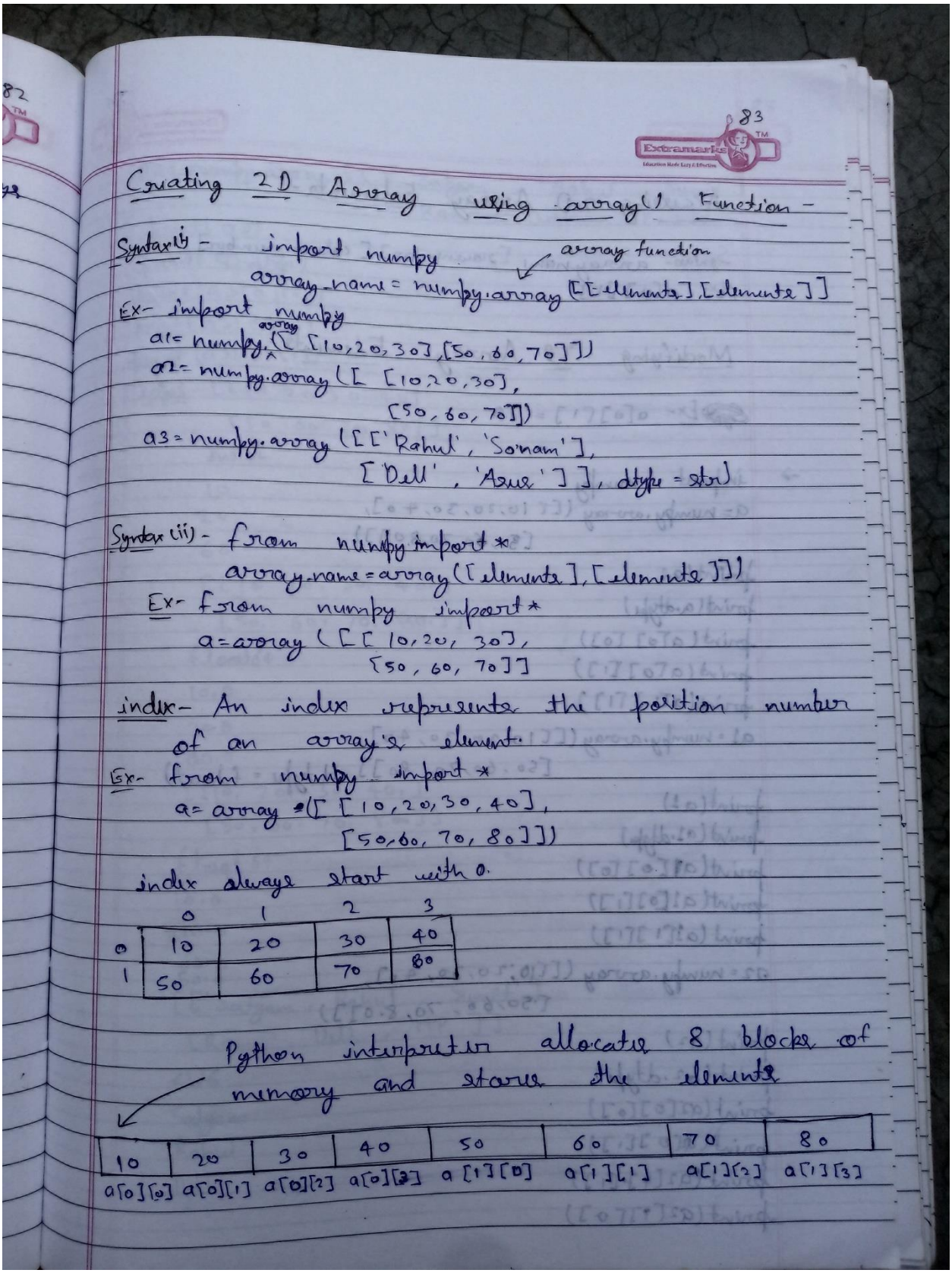
Way of creating Multi-D Array -

- array() Function
- zeros() Function
- ones() Function
- reshape() Function

. array() Function- This function of numpy is used to create a multi-D array.

Syntax- numpy.array (object, dtype=None, copy=True, order='K', subok=False, ndmin=0)

https://www.numpy.org/devdocs/reference/generated/numpy.array.html.



Creating 2D Array using array() Function -

Syntax (i) - import numpy
array_name = numpy.array([[elements],[elements]])
(array function)

Ex- import numpy
a1 = numpy.array([[10,20,30],[50,60,70]])
a2 = numpy.array([[10,20,30],
[50,60,70]])
a3 = numpy.array([['Rahul','Sonam'],
['Dell','Asus']], dtype = str)

Syntax (ii) - from numpy import *
array_name = array([elements],[elements]])

Ex- from numpy import *
a = array([[10,20,30],
[50,60,70]]

index- An index represents the position number of an array's element.

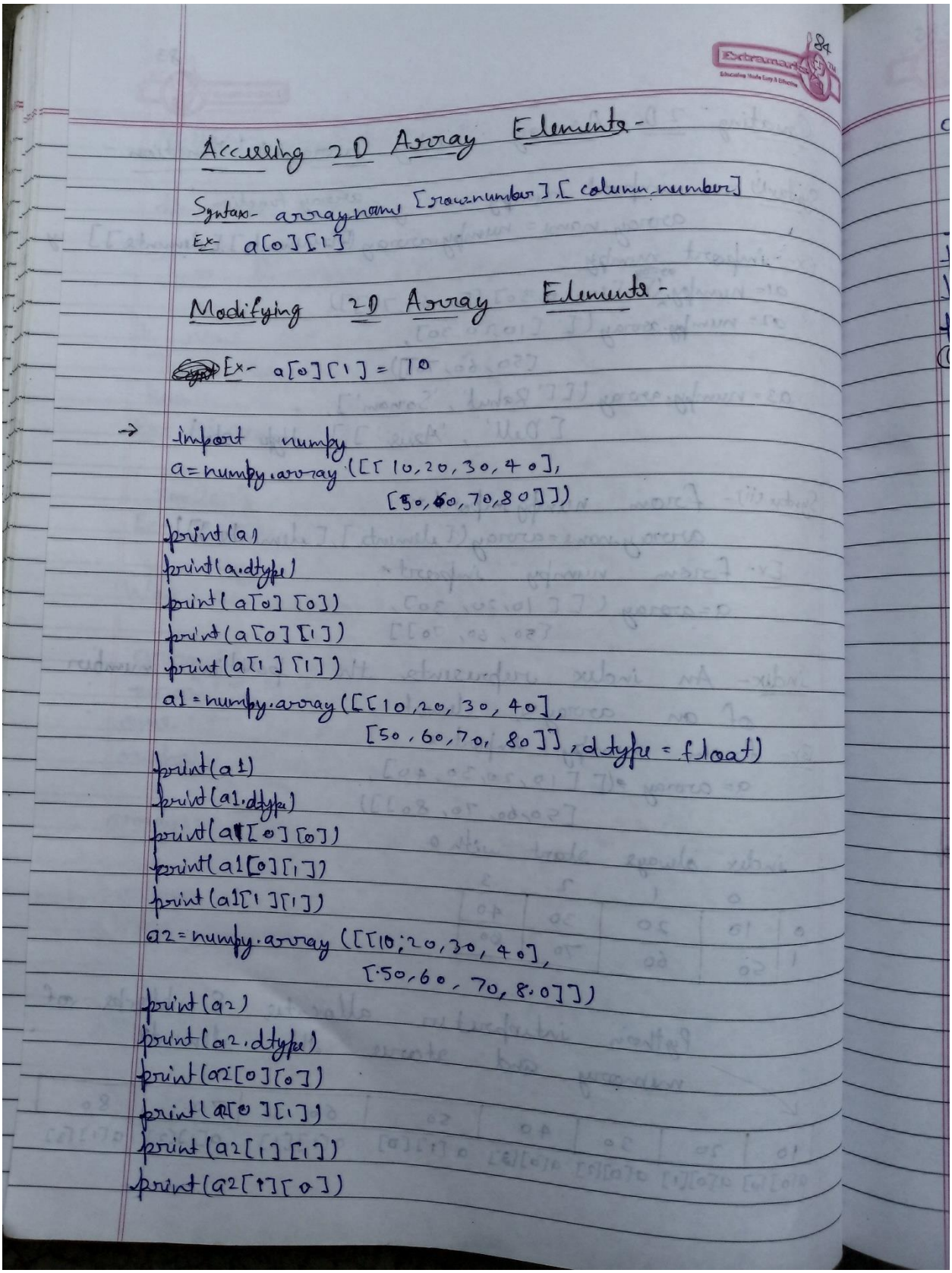
Ex- from numpy import *
a = array([[10,20,30,40],
[50,60,70,80]])

index always start with 0.

	0	1	2	3
0	10	20	30	40
1	50	60	70	80

Python interpreter allocates 8 blocks of memory and stores the elements

10	20	30	40	50	60	70	80
a[0][0]	a[0][1]	a[0][2]	a[0][3]	a[1][0]	a[1][1]	a[1][2]	a[1][3]



Accessing 2D Array Elements -

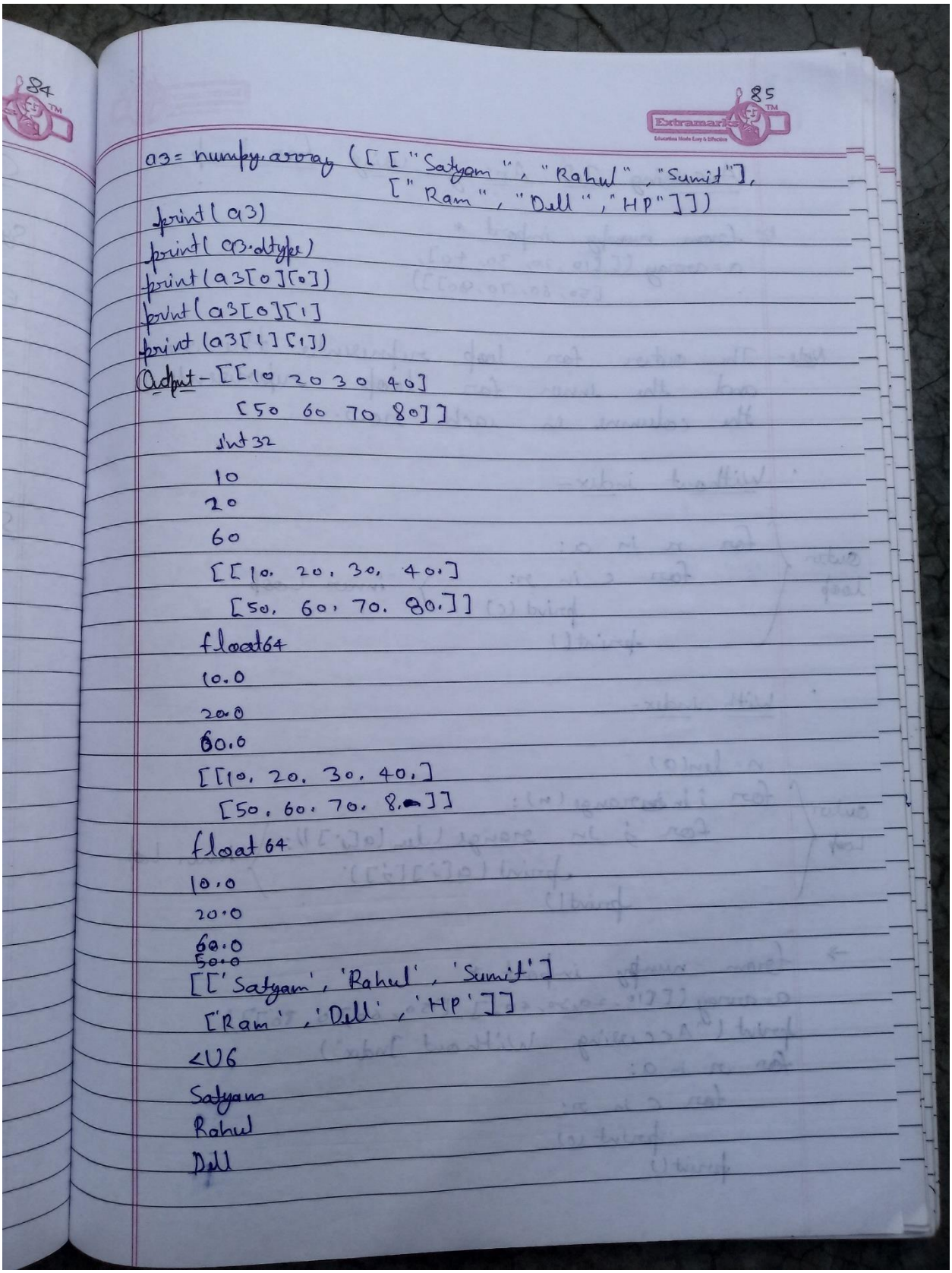
Syntax- arrayname [row_number] [column_number]

Ex- a[0][1]

Modifying 2D Array Elements -

Ex- a[0][1] = 10

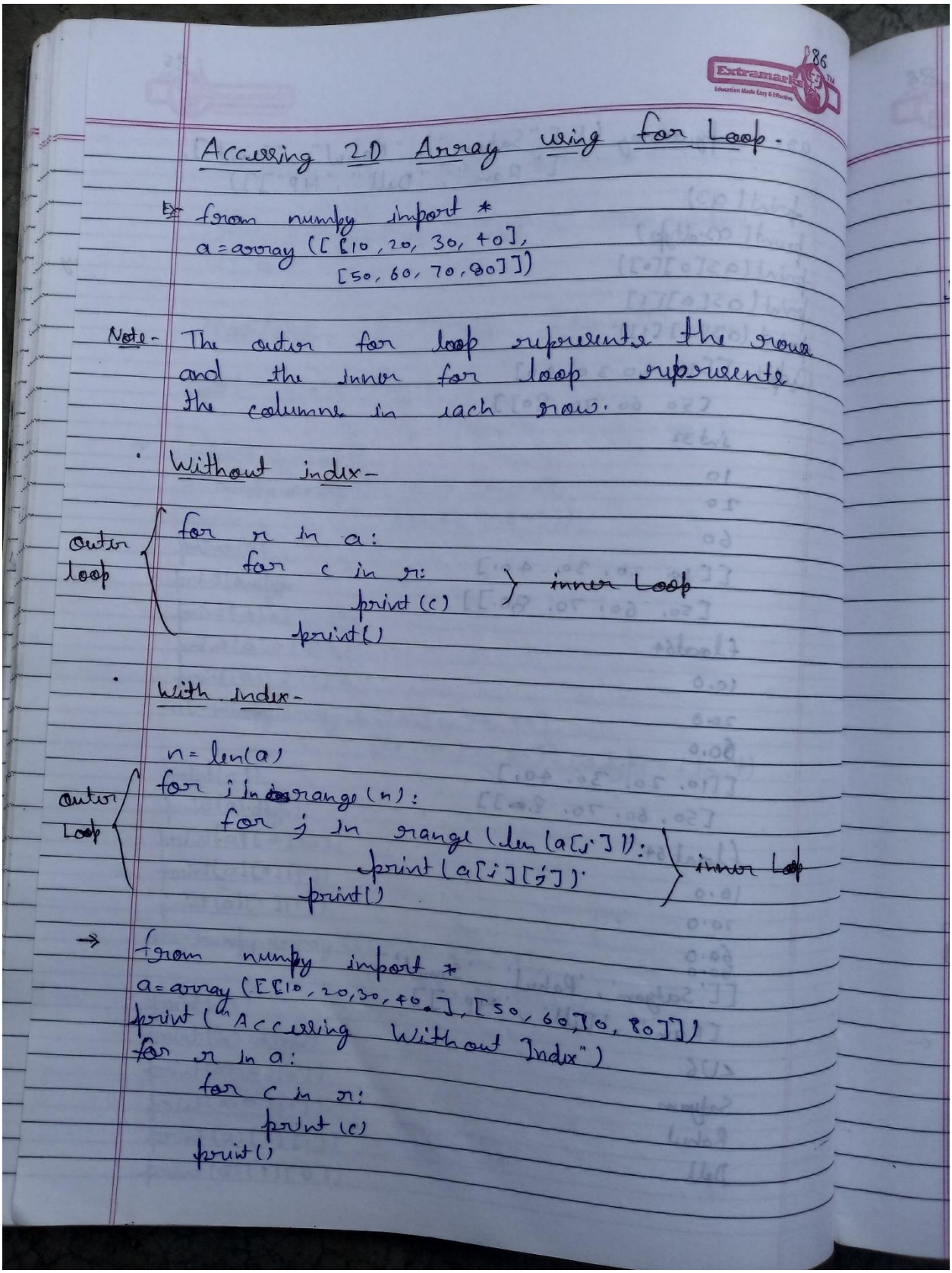
```
import numpy
a=numpy.array([[10,20,30,40],
               [50,60,70,80]])
print(a)
print(a.dtype)
print(a[0][0])
print(a[0][1])
print(a[1][1])
a1=numpy.array([[10,20,30,40],
                [50,60,70,80]],dtype=float)
print(a1)
print(a1.dtype)
print(a1[0][0])
print(a1[0][1])
print(a1[1][1])
a2=numpy.array([[10,20,30,40],
                [50,60,70,80]])
print(a2)
print(a2.dtype)
print(a2[0][0])
print(a2[0][1])
print(a2[1][1])
print(a2[1][0])
```



```
a3 = numpy.array([["Satyam", "Rahul", "Sumit"],
                  ["Ram", "Dell", "HP"]])
print(a3)
print(a3.dtype)
print(a3[0][0])
print(a3[0][1]
print(a3[1][:])
```

Output -

```
[[10 20 30 40]
 [50 60 70 80]]
int32
10
20
60
[[10. 20. 30. 40.]
 [50. 60. 70. 80.]]
float64
10.0
20.0
60.0
[[10. 20. 30. 40.]
 [50. 60. 70. 8.]]
float64
10.0
20.0
60.0
50.0
[['Satyam' 'Rahul' 'Sumit']
 ['Ram' 'Dell' 'HP']]
<U6
Satyam
Rahul
Dell
```



Accessing 2D Array using for Loop.

Ex
```
from numpy import *
a=array([[10,20,30,40],
         [50,60,70,80]])
```

Note- The outer for loop represents the rows and the inner for loop represents the columns in each row.

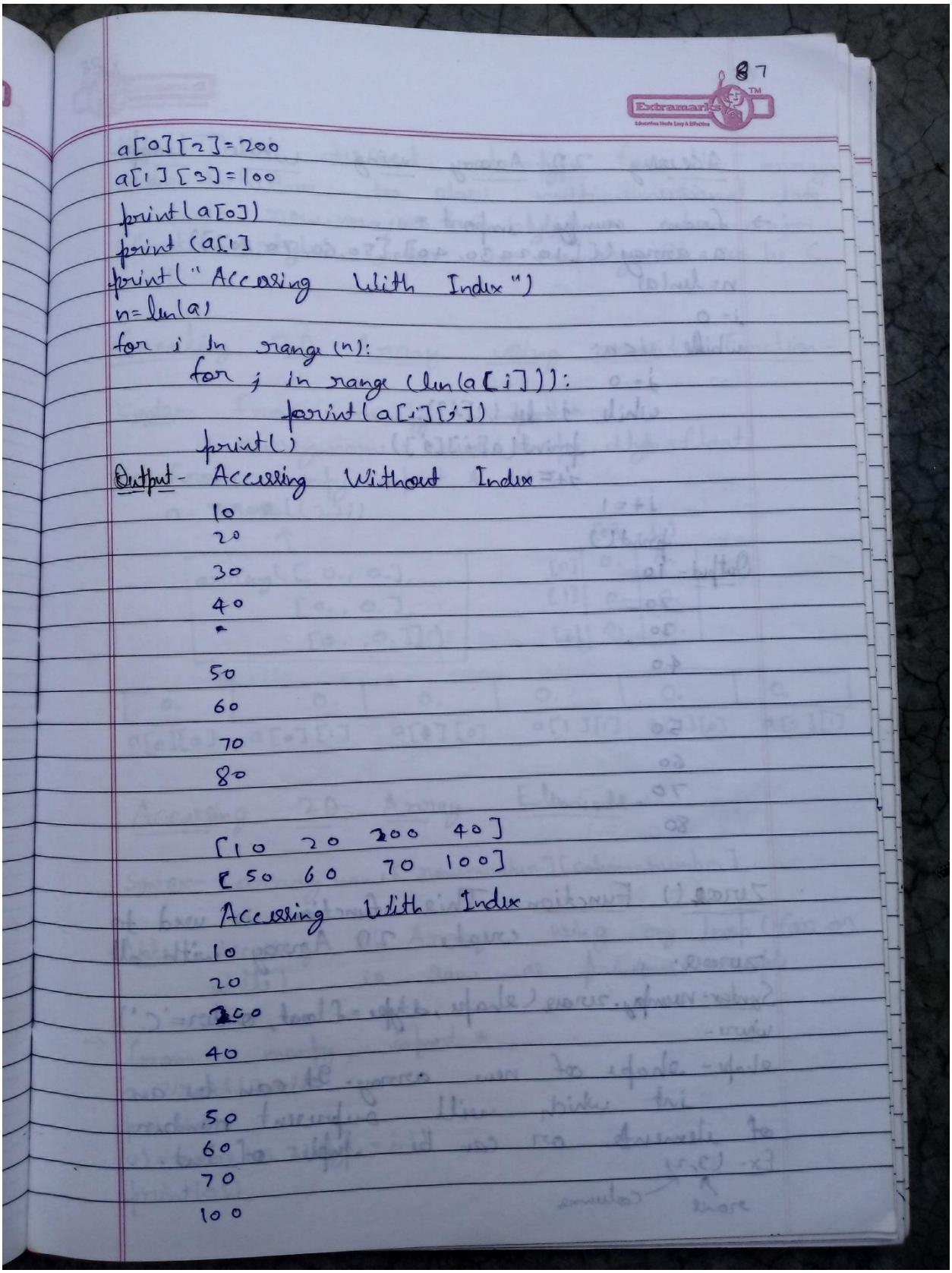
- Without index-

```
for r in a:
    for c in r:
        print(c)
    print()
```
(outer loop: `for r in a:` ; inner loop: `for c in r: print(c)`)

- With index-

```
n=len(a)
for i in range(n):
    for j in range(len(a[i])):
        print(a[i][j])
    print()
```
(outer loop: `for i in range(n):` ; inner loop: `for j in range(len(a[i])): print(a[i][j])`)

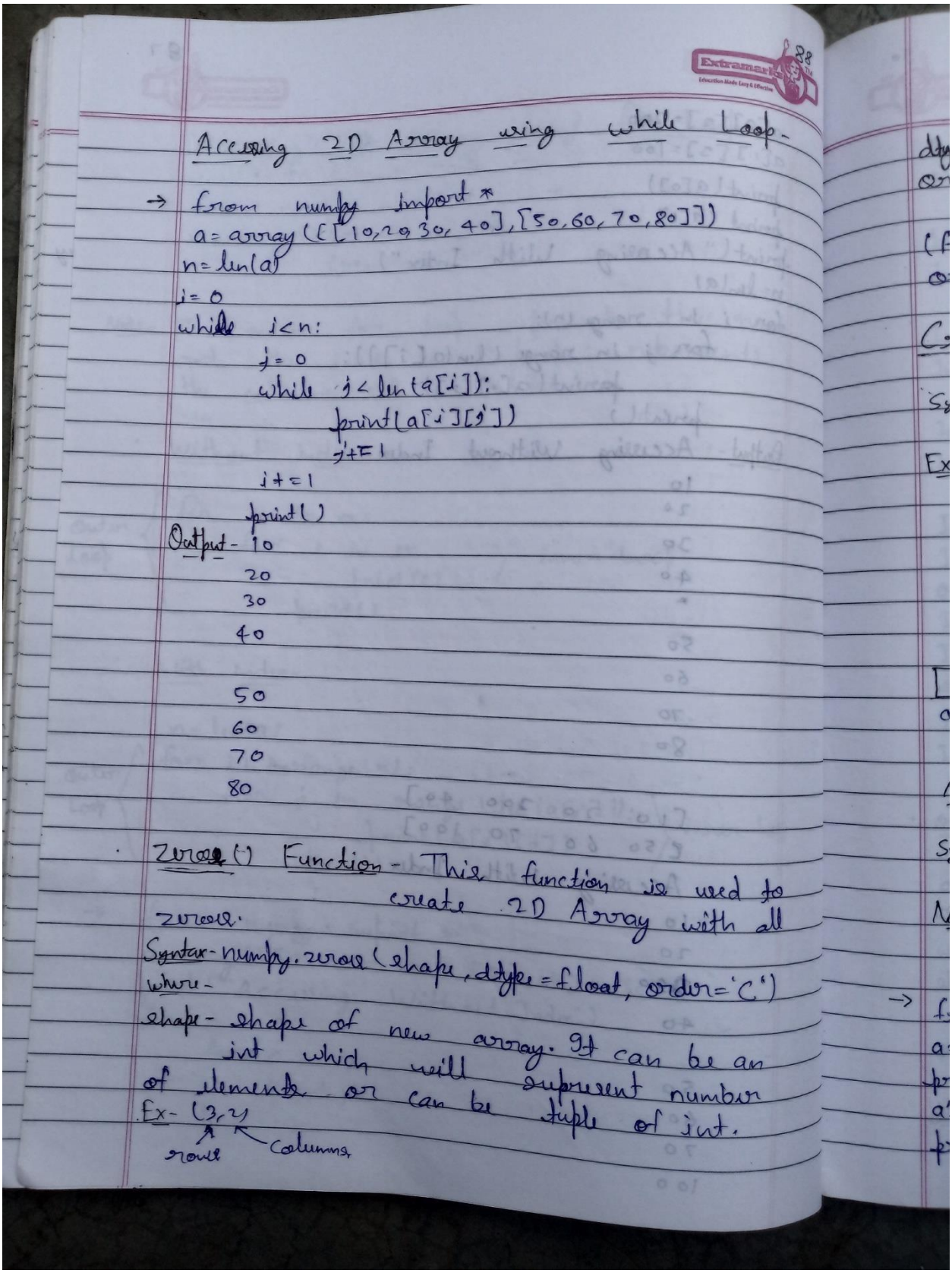
→
```
from numpy import *
a=array([[10,20,30,40],[50,60,70,80]])
print("Accessing Without Index")
for r in a:
    for c in r:
        print(c)
    print()
```



```
a[0][2]=200
a[1][3]=100
print(a[0])
print(a[1])
print("Accessing With Index")
n=len(a)
for i in range(n):
    for j in range(len(a[i])):
        print(a[i][j])
    print()
```

Output - Accessing Without Index

```
10
20
200
40

50
60
70
100
[10, 20, 200, 40]
[50, 60, 70, 100]
Accessing With Index
10
20
200
40

50
60
70
100
```



Accessing 2D Array using while Loop-

→
```
from numpy import *
a=array([[10,20,30,40],[50,60,70,80]])
n=len(a)
i=0
while i<n:
    j=0
    while j<len(a[i]):
        print(a[i][j])
        j+=1
    i+=1
    print()
```

Output- 10
20
30
40

50
60
70
80

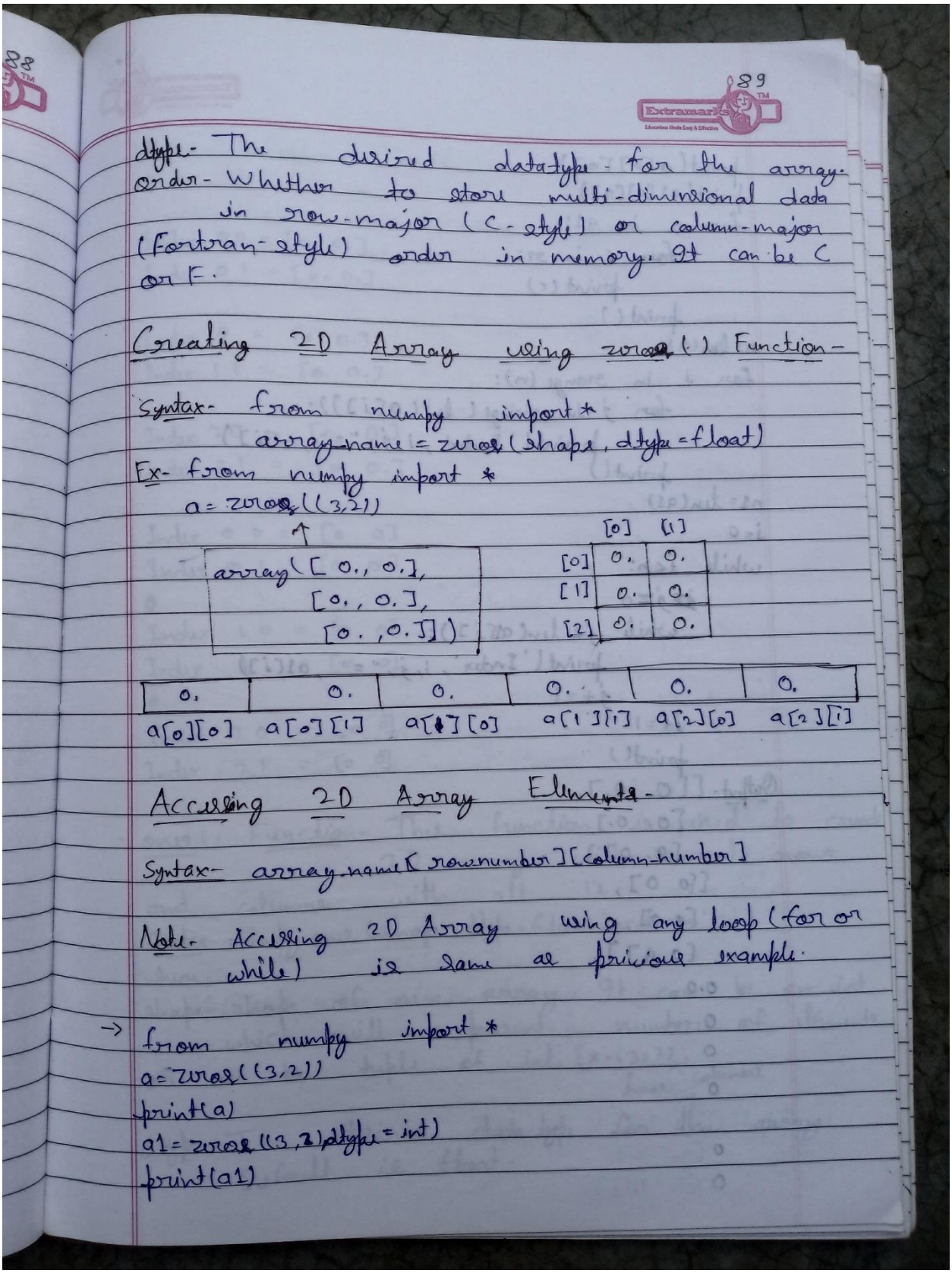
zeros() Function - This function is used to create 2D Array with all zeros.

Syntax- numpy.zeros(shape, dtype=float, order='C')

where-

shape- shape of new array. It can be an int which will represent number of elements or can be tuple of int.

Ex- (3,2)
↑ rows ↖ Columns



dtype- The desired datatype for the array.
order- Whether to store multi-dimensional data in row-major (C-style) or column-major (Fortran-style) order in memory. It can be C or F.

Creating 2D Array using zeros() Function-

Syntax- from numpy import *
array_name = zeros(shape, dtype=float)

Ex- from numpy import *
a = zeros((3,2))

```
array([[0., 0.],
       [0., 0.],
       [0., 0.]])
```

	[0]	[1]
[0]	0.	0.
[1]	0.	0.
[2]	0.	0.

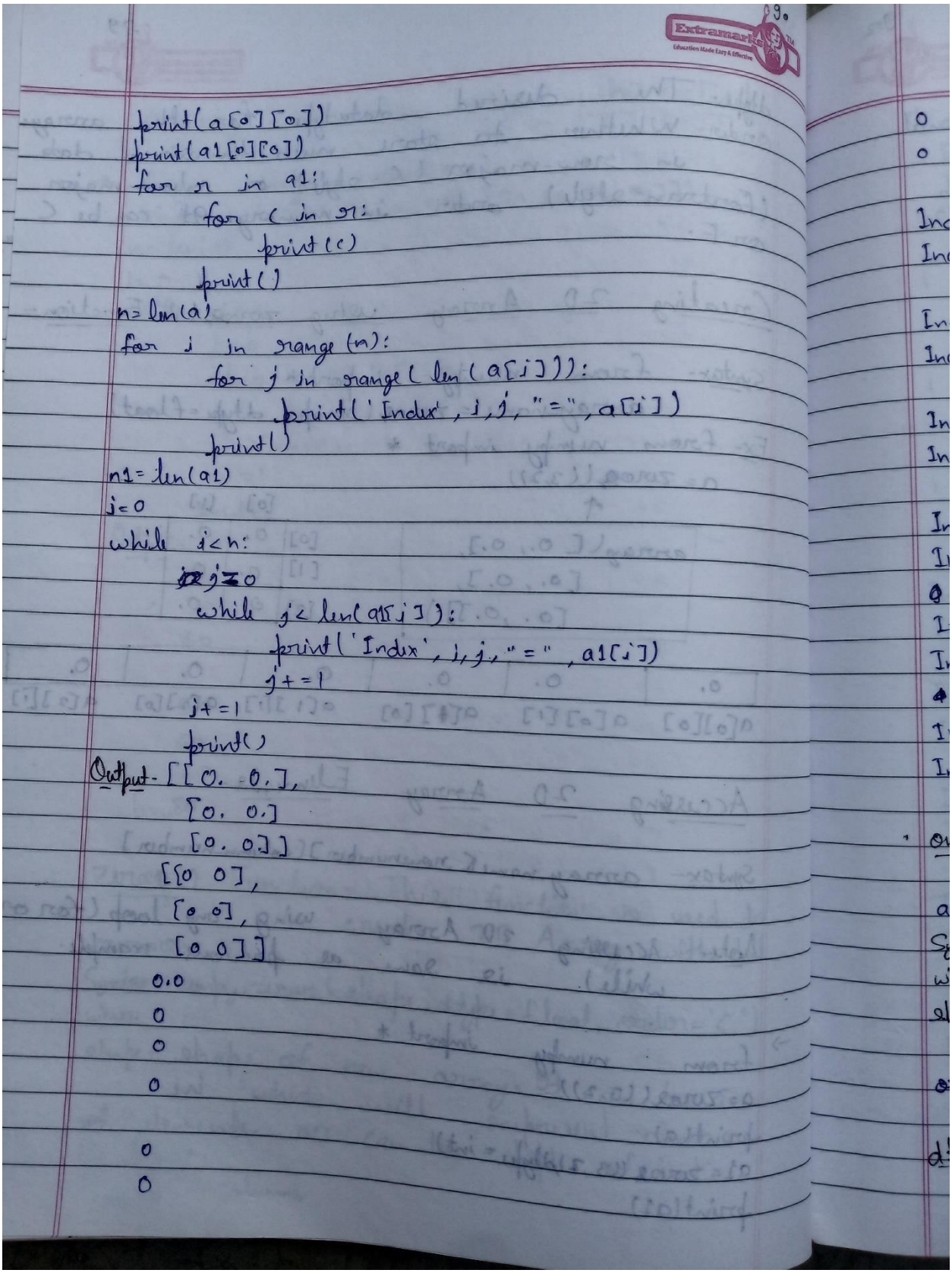
0.	0.	0.	0.	0.	0.
a[0][0]	a[0][1]	a[1][0]	a[1][1]	a[2][0]	a[2][1]

Accessing 2D Array Elements-

Syntax- array_name[rownumber][column-number]

Note- Accessing 2D Array using any loop (for or while) is same as previous example.

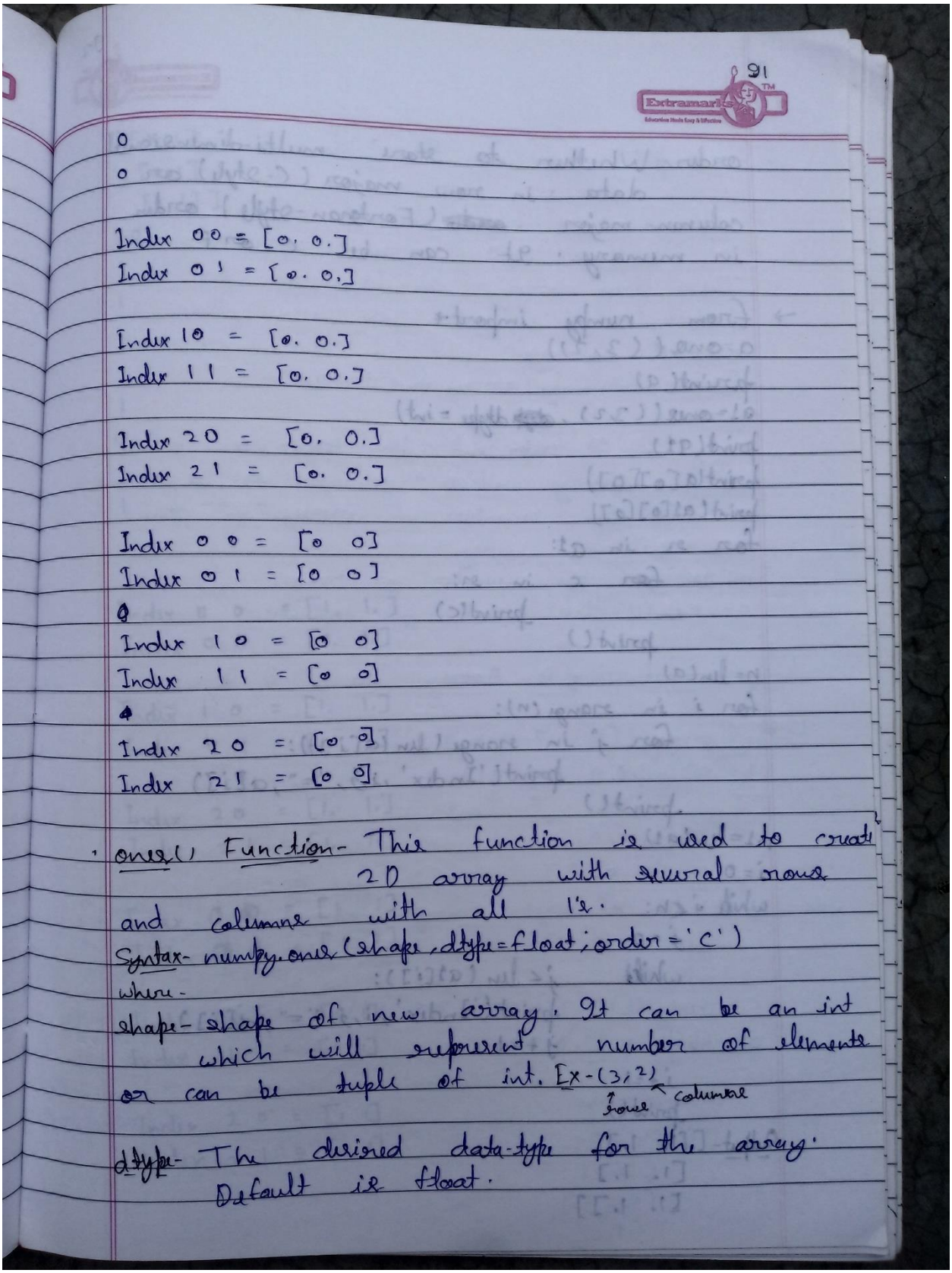
→
```
from numpy import *
a=zeros((3,2))
print(a)
a1=zeros((3,2),dtype=int)
print(a1)
```



```
print(a[0][0])
print(a1[0][0])
for r in a1:
    for c in r:
        print(c)
    print()
n=len(a)
for i in range(n):
    for j in range(len(a[i])):
        print('Index',i,j,"=",a[i])
    print()
n1=len(a1)
i=0
while i<n:
    j=0
    while j<len(a1[i]):
        print('Index',i,j,"=",a1[i])
        j+=1
    i+=1
    print()
```

Output-

```
[[0. 0.],
 [0. 0.]
 [0. 0.]]
[[0 0],
 [0 0],
 [0 0]]
0.0
0
0
0

0
0
```



0
0

Index 00 = [0. 0.]
Index 01 = [0. 0.]

Index 10 = [0. 0.]
Index 11 = [0. 0.]

Index 20 = [0. 0.]
Index 21 = [0. 0.]

Index 00 = [0 0]
Index 01 = [0 0]
0
Index 10 = [0 0]
Index 11 = [0 0]
4
Index 20 = [0 0]
Index 21 = [0 0]

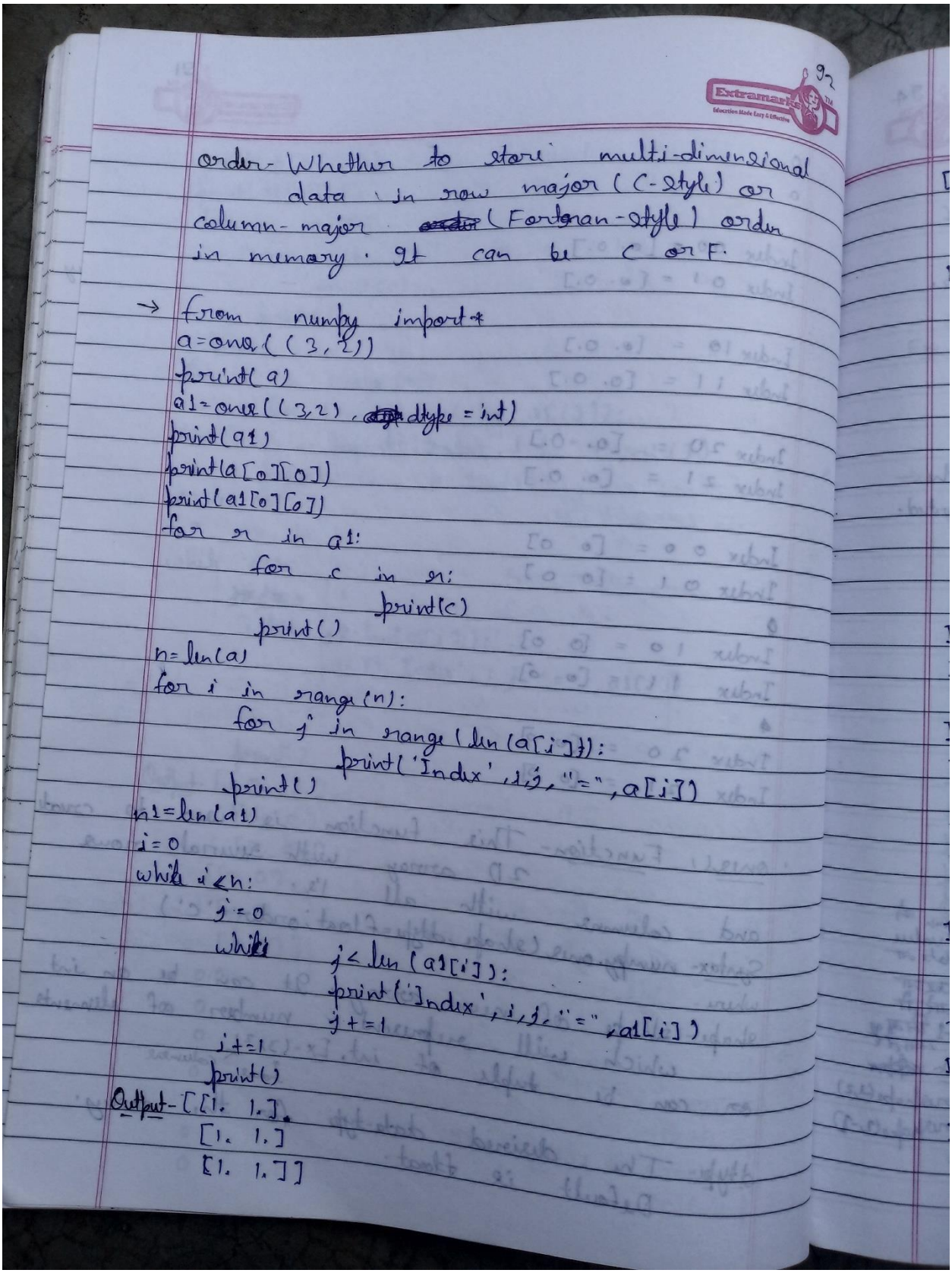
- ones() Function - This function is used to create 2D array with several rows and columns with all 1's.

Syntax - numpy.ones(shape, dtype=float, order='C')

where -

shape - shape of new array. It can be an int which will represent number of elements or can be tuple of int. Ex - (3, 2) (3 rows, 2 columns)

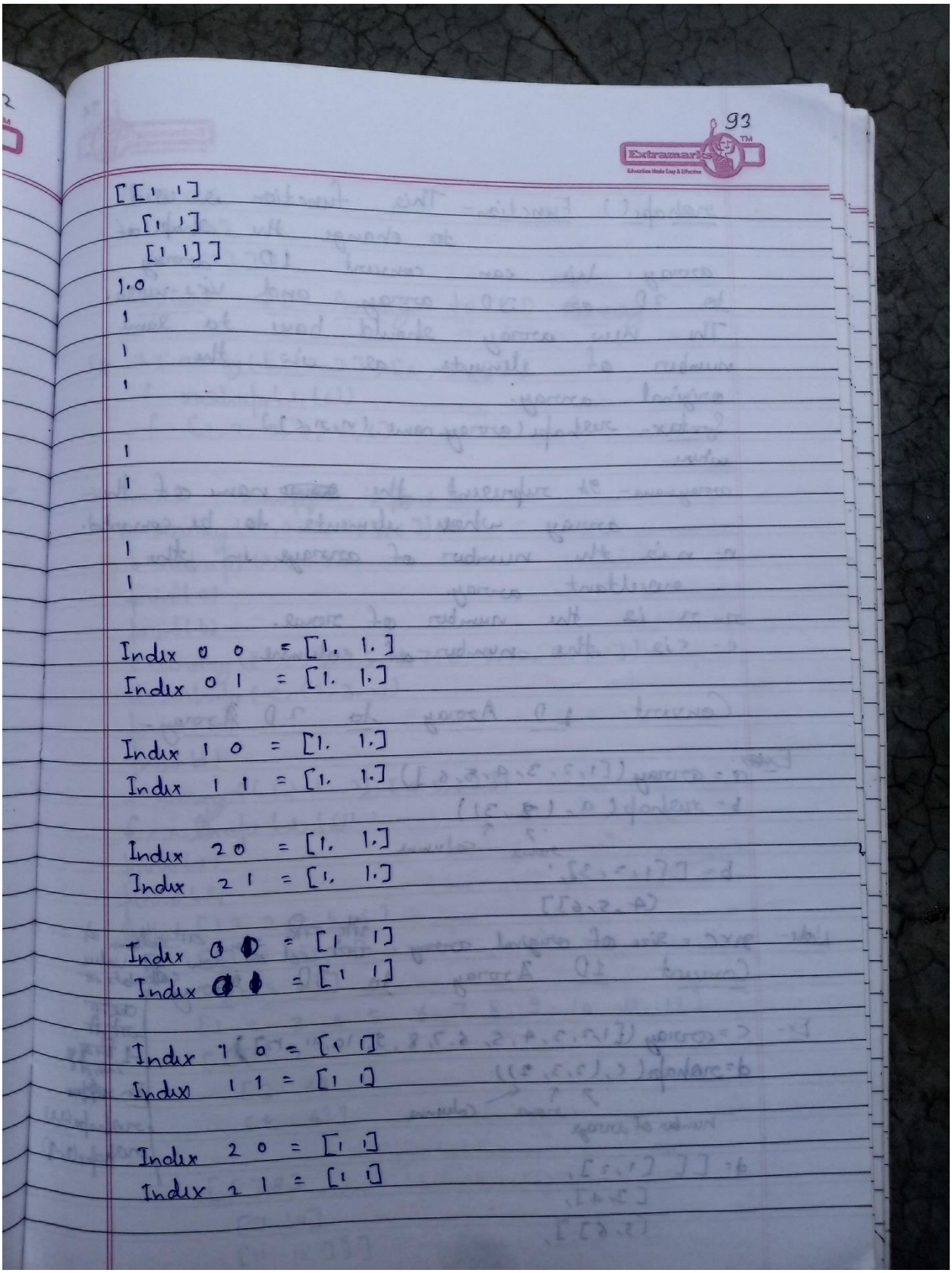
dtype - The desired data-type for the array. Default is float.



order - Whether to store multi-dimensional data in row major (C-style) or column-major (Fortran-style) order in memory. It can be C or F.

→
```
from numpy import *
a=ones((3,2))
print(a)
a1=ones((3,2), dtype=int)
print(a1)
print(a[0][0])
print(a1[0][0])
for r in a1:
    for c in r:
        print(c)
    print()
n=len(a)
for i in range(n):
    for j in range(len(a[i])):
        print('Index',i,j,"=",a[i])
    print()
n1=len(a1)
i=0
while i<n:
    j=0
    while j<len(a1[i]):
        print('Index',i,j,"=",a1[i])
        j+=1
    i+=1
    print()
```

Output-
```
[[1. 1.]
 [1. 1.]
 [1. 1.]]
```


93
Extramarks
[[1 1]
[1 1]
[1 1]]
1.0
1
1
1
1
1
1
1
Indx 0 0 = [1. 1.]
Indx 0 1 = [1. 1.]
Indx 1 0 = [1. 1.]
Indx 1 1 = [1. 1.]
Indx 2 0 = [1. 1.]
Indx 2 1 = [1. 1.]
Indx 0 0 = [1 1]
Indx 0 1 = [1 1]
Indx 1 0 = [1 1]
Indx 1 1 = [1 1]
Indx 2 0 = [1 1]
Indx 2 1 = [1 1]

- **reshape() Function**- This function is used to change the shape of array. We can convert 1D array to 2D or 3D array and vice-versa. The new array should have to same number of elements as in the original array.

Syntax- reshape(array-name, (n,r,c))

where

array-name - It represent the name of the array whose elements to be converted.

n- n is the number of arrays in the resultant array.

r- r is the number of rows.

c- c is the number of columns.

Convert 1D Array to 2D Array-

Ex-
```
a = array([1, 2, 3, 4, 5, 6])
b = reshape(a, (2,3))
                ↑  ↑
             rows  columns
b = [[1, 2, 3],
     [4, 5, 6]]
```

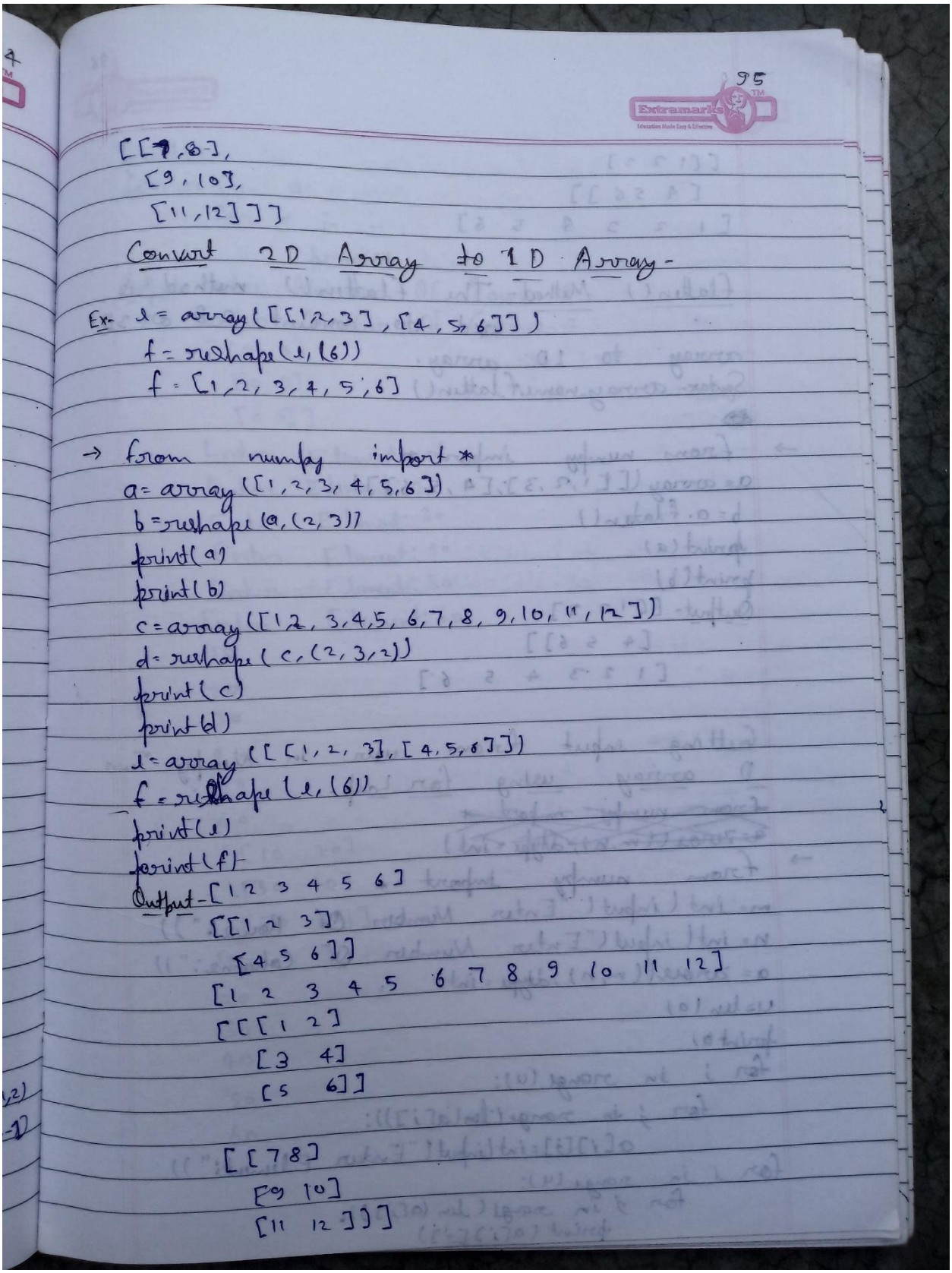
Note- r×c = size of original array, और यदि row या column को फिक्स col या row की value calculate करना चाहते तो -1 लिख सकते है

Ex- reshape(-1,2)

reshape(2,-1)

Convert 1D Array to 3D Array-

Ex-
```
c = array([1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12])
d = reshape(c, (2, 3, 2))
                ↑  ↑  ↑
  number of arrays rows columns
d = [[ [1, 2],
       [3, 4],
       [5, 6] ],
```



95

[[7,8],
[9,10],
[11,12]]]

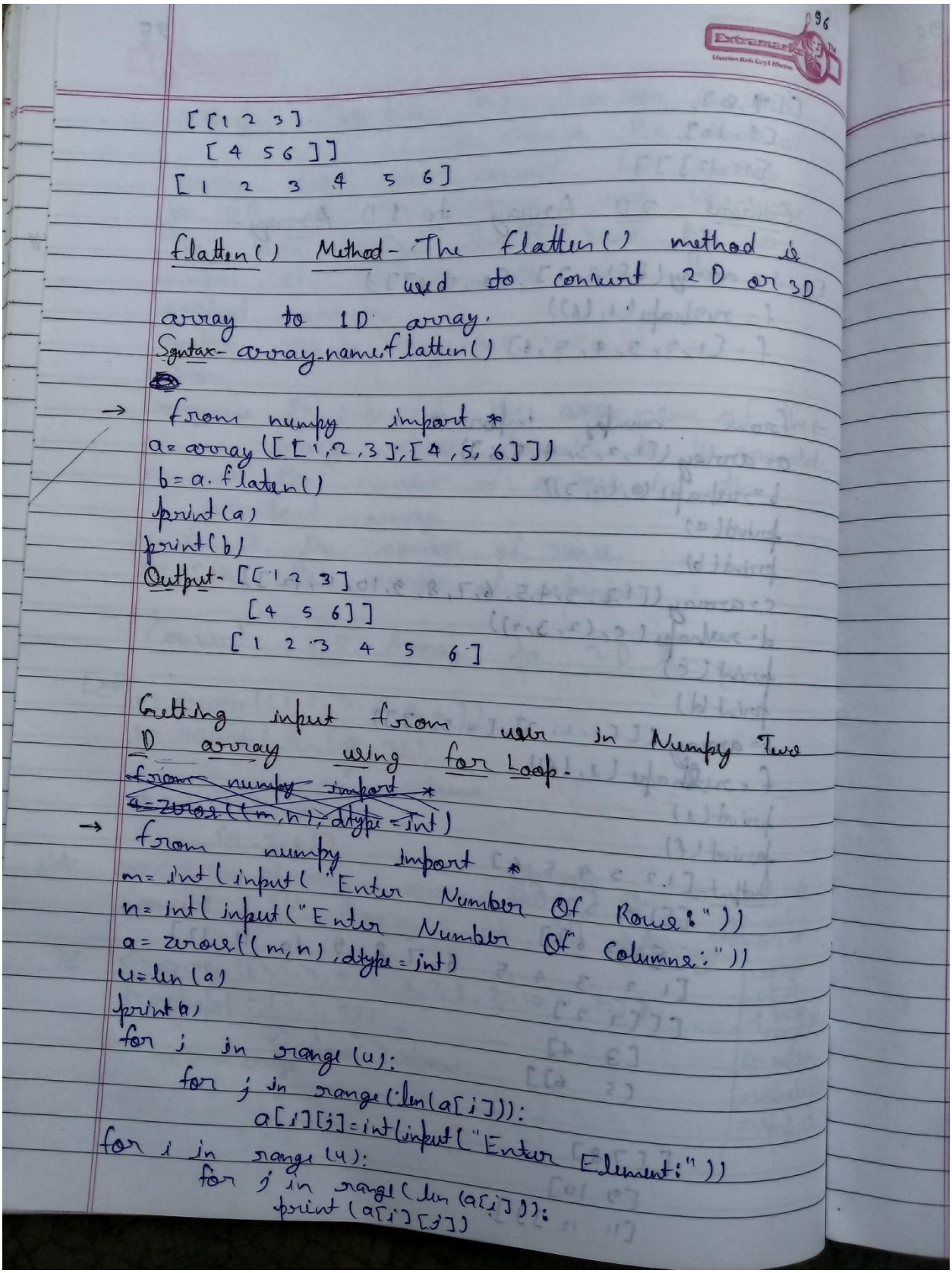
Convert 2D Array to 1D Array -

Ex- a = array([[1,2,3],[4,5,6]])
f = reshape(a,(6))
f = [1,2,3,4,5,6]

→ from numpy import *
a = array([1,2,3,4,5,6])
b = reshape(a,(2,3))
print(a)
print(b)
c = array([1,2,3,4,5,6,7,8,9,10,11,12])
d = reshape(c,(2,3,2))
print(c)
print(d)
a = array([[1,2,3],[4,5,6]])
f = reshape(a,(6))
print(a)
print(f)

Output - [1 2 3 4 5 6]
[[1 2 3]
[4 5 6]]
[1 2 3 4 5 6 7 8 9 10 11 12]
[[[1 2]
[3 4]
[5 6]]

[[7 8]
[9 10]
[11 12]]]



```
[[1 2 3]
 [4 5 6]]
[1 2 3 4 5 6]
```

flatten() Method - The flatten() method is used to convert 2D or 3D array to 1D array.

Syntax - array_name.flatten()

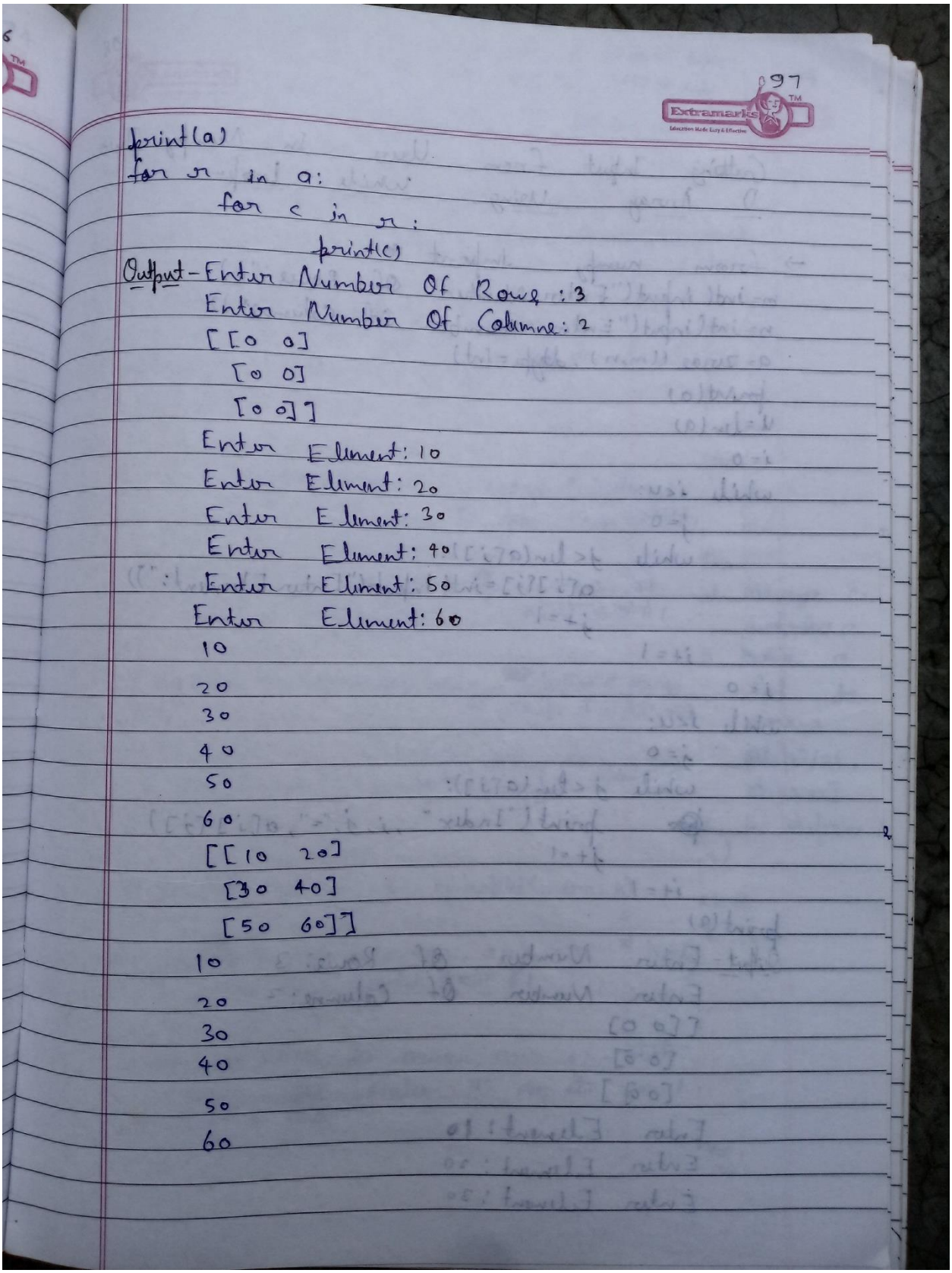
→
```
from numpy import *
a=array([[1,2,3],[4,5,6]])
b=a.flatten()
print(a)
print(b)
```

Output -
```
[[1 2 3]
 [4 5 6]]
[1 2 3 4 5 6]
```

Getting input from user in Numpy Two D array using for Loop.

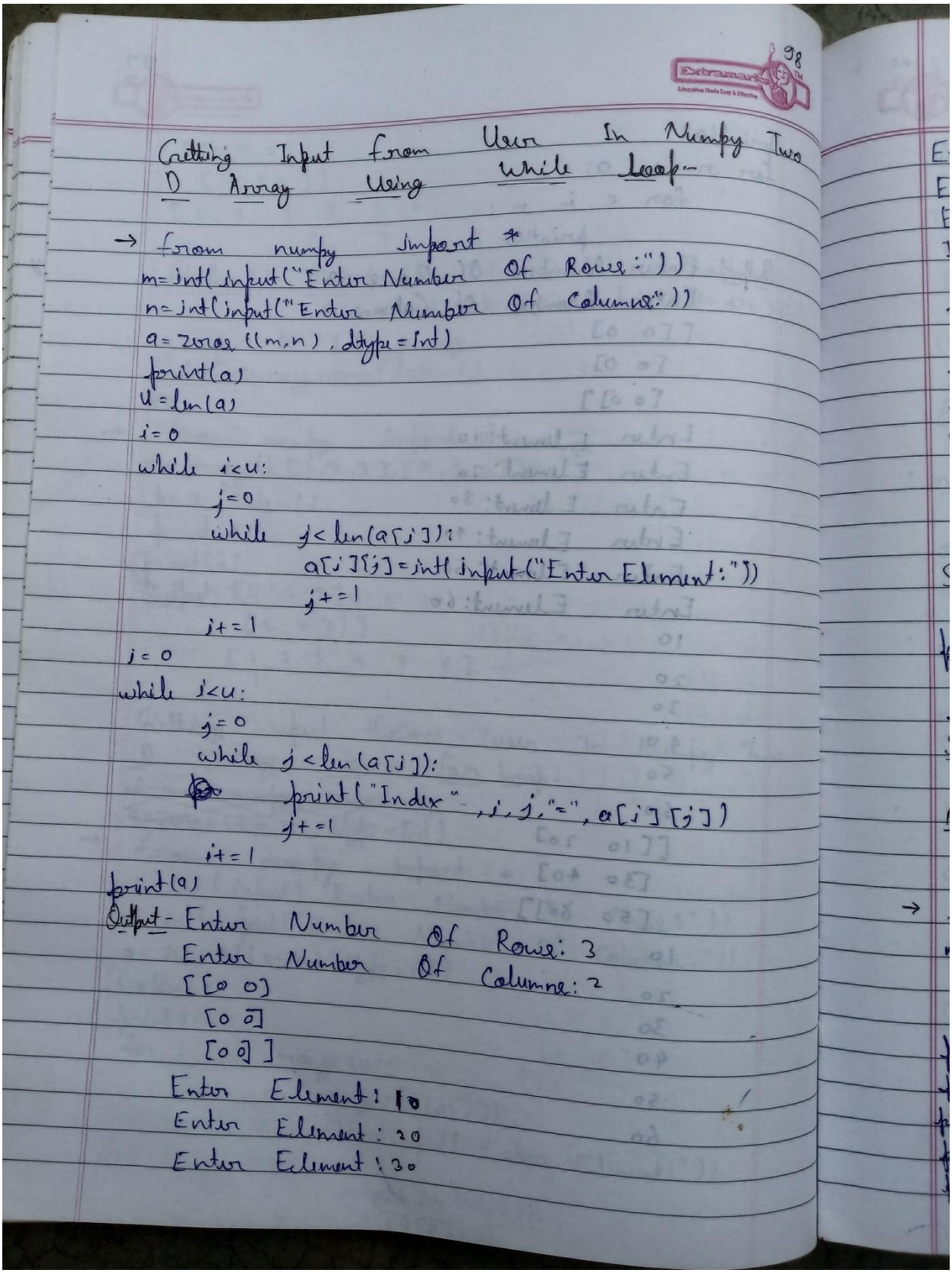
~~from numpy import *~~
~~a=zeros((m,n),dtype=int)~~

→
```
from numpy import *
m=int(input("Enter Number Of Rows:"))
n=int(input("Enter Number Of Columns:"))
a=zeros((m,n),dtype=int)
u=len(a)
print(a)
for i in range(u):
    for j in range(len(a[i])):
        a[i][j]=int(input("Enter Element:"))
for i in range(u):
    for j in range(len(a[i])):
        print(a[i][j])
```



```
print(a)
for r in a:
    for c in r:
        print(c)
```

Output -
```
Enter Number Of Rows : 3
Enter Number Of Columns: 2
[[0 0]
 [0 0]
 [0 0]]
Enter Element: 10
Enter Element: 20
Enter Element: 30
Enter Element: 40
Enter Element: 50
Enter Element: 60
10
20
30
40
50
60
[[10 20]
 [30 40]
 [50 60]]
10
20
30
40
50
60
```

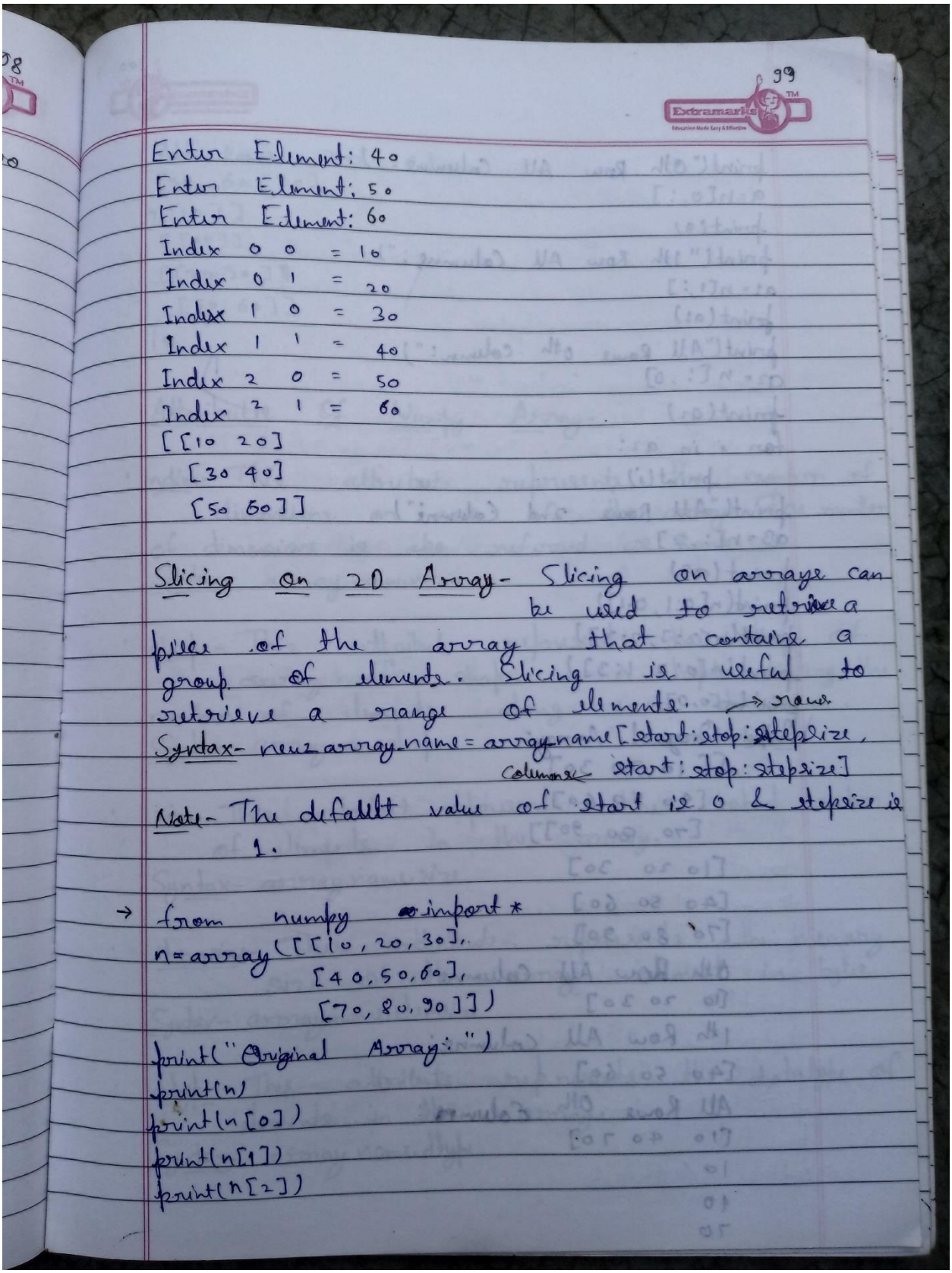


98

Getting Input from User In Numpy Two D Array Using while Loop:-

→
```
from numpy import *
m=int(input("Enter Number Of Rows:"))
n=int(input("Enter Number Of Columns:"))
a=zeros((m,n),dtype=int)
print(a)
u=len(a)
i=0
while i<u:
    j=0
    while j<len(a[i]):
        a[i][j]=int(input("Enter Element:"))
        j+=1
    i+=1
i=0
while i<u:
    j=0
    while j<len(a[i]):
        print("Index",i,j,"=",a[i][j])
        j+=1
    i+=1
print(a)
```

Output- Enter Number Of Rows: 3
Enter Number Of Columns: 2
[[0 0]
[0 0]
[0 0]]
Enter Element: 10
Enter Element: 20
Enter Element: 30



99

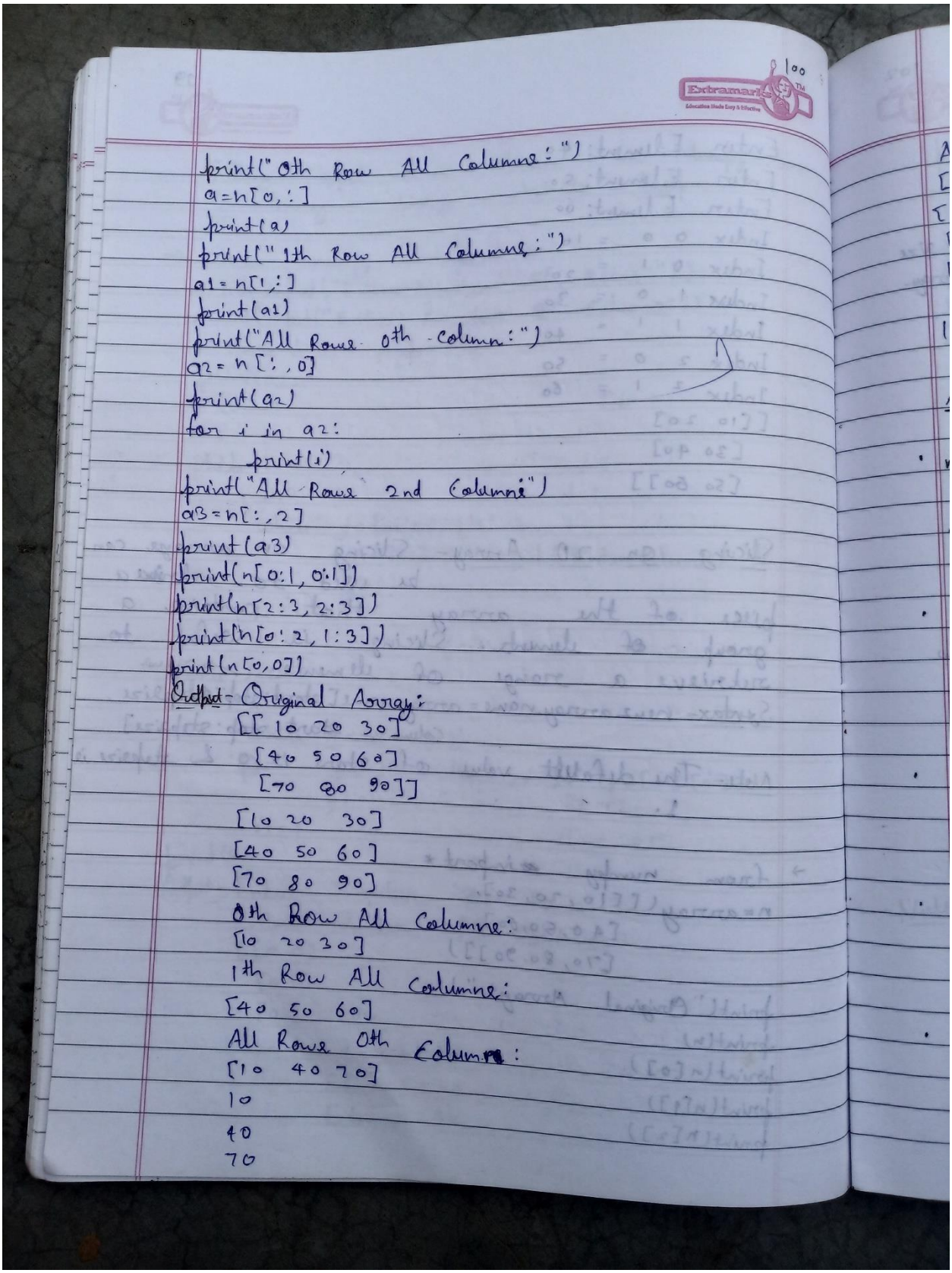
```
Enter Element: 40
Enter Element: 50
Enter Element: 60
Index 0 0 = 10
Index 0 1 = 20
Index 1 0 = 30
Index 1 1 = 40
Index 2 0 = 50
Index 2 1 = 60
[[10 20]
 [30 40]
 [50 60]]
```

Slicing on 2D Array- Slicing on arrays can be used to retrieve a piece of the array that contains a group of elements. Slicing is useful to retrieve a range of elements.

Syntax- new_array_name = array_name[start:stop:stepsize, start:stop:stepsize]
(row) (column)

Note- The default value of start is 0 & stepsize is 1.

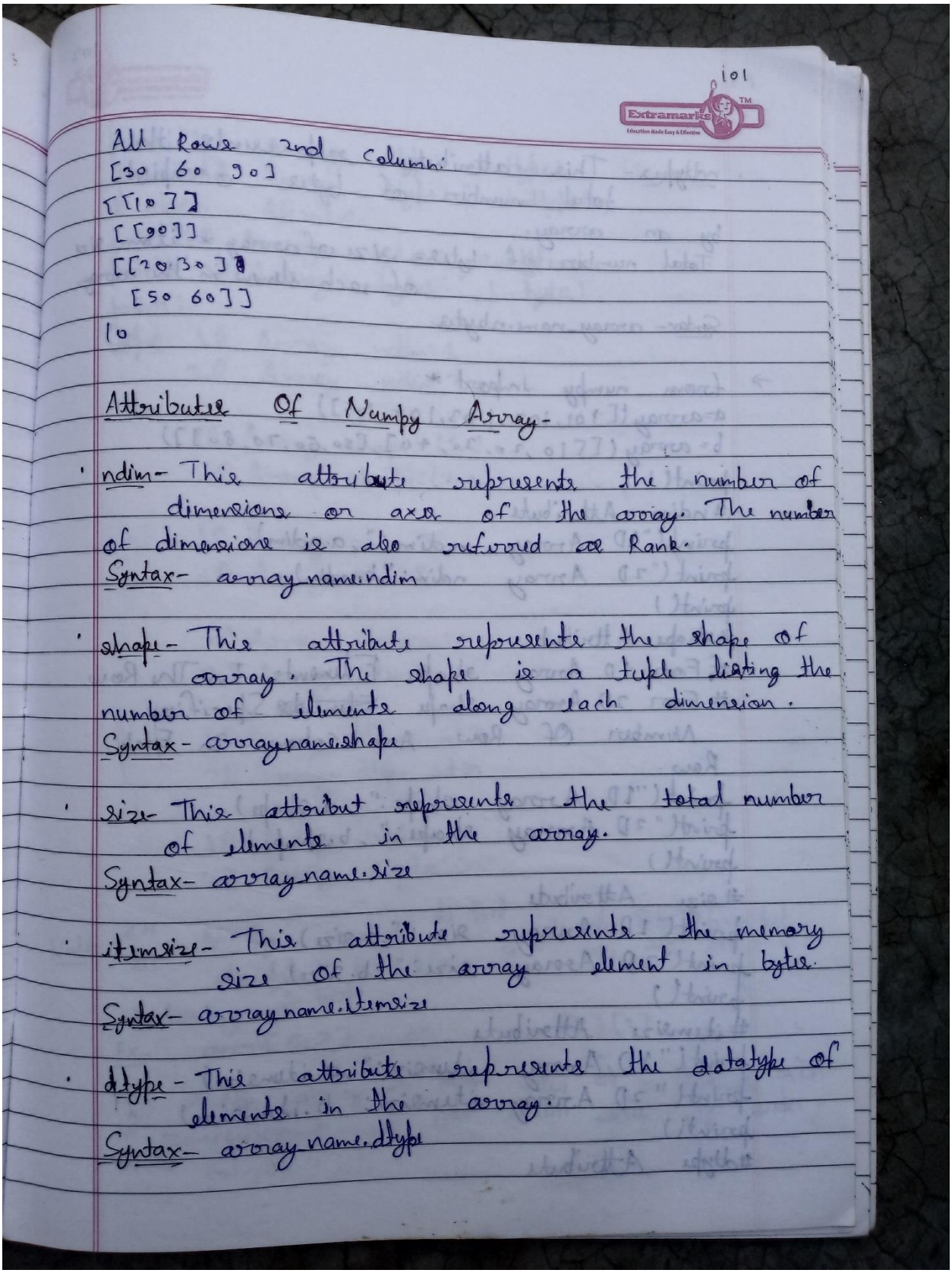
→
```
from numpy import *
n=array([[10,20,30],
         [40,50,60],
         [70,80,90]])
print("Original Array:")
print(n)
print(n[0])
print(n[1])
print(n[2])
```



```
print("0th Row All Columns:")
a=n[0,:]
print(a)
print("1th Row All Columns:")
a1=n[1,:]
print(a1)
print("All Rows 0th Column:")
a2=n[:,0]
print(a2)
for i in a2:
    print(i)
print("All Rows 2nd Columns")
a3=n[:,2]
print(a3)
print(n[0:1,0:1])
print(n[2:3,2:3])
print(n[0:2,1:3])
print(n[0,0])
```

Output - Original Array:

```
[[10 20 30]
 [40 50 60]
 [70 80 90]]
[10 20 30]
[40 50 60]
[70 80 90]
0th Row All Columns:
[10 20 30]
1th Row All Columns:
[40 50 60]
All Rows 0th Columns:
[10 40 70]
10
40
70
```



101

All Rows 2nd Column:
[30 60 90]
[[10]]
[[90]]
[[20 30]
 [50 60]]
10

Attributes Of Numpy Array-

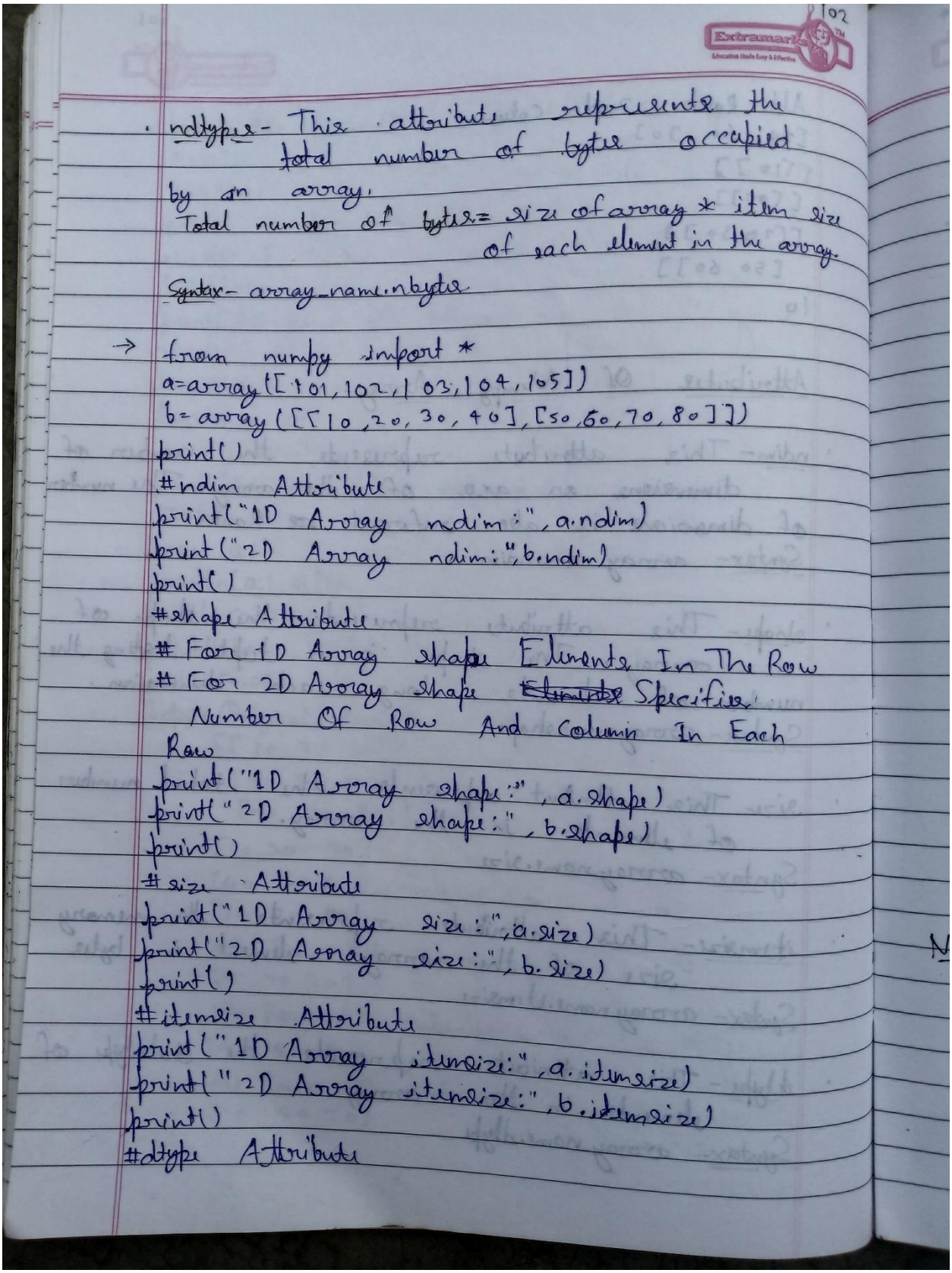
- **ndim**- This attribute represents the number of dimensions or axes of the array. The number of dimensions is also referred as Rank.
  Syntax- arrayname.ndim

- **shape**- This attribute represents the shape of array. The shape is a tuple listing the number of elements along each dimension.
  Syntax- arrayname.shape

- **size**- This attribut represents the total number of elements in the array.
  Syntax- arrayname.size

- **itemsize**- This attribute represents the memory size of the array element in byte.
  Syntax- arrayname.itemsize

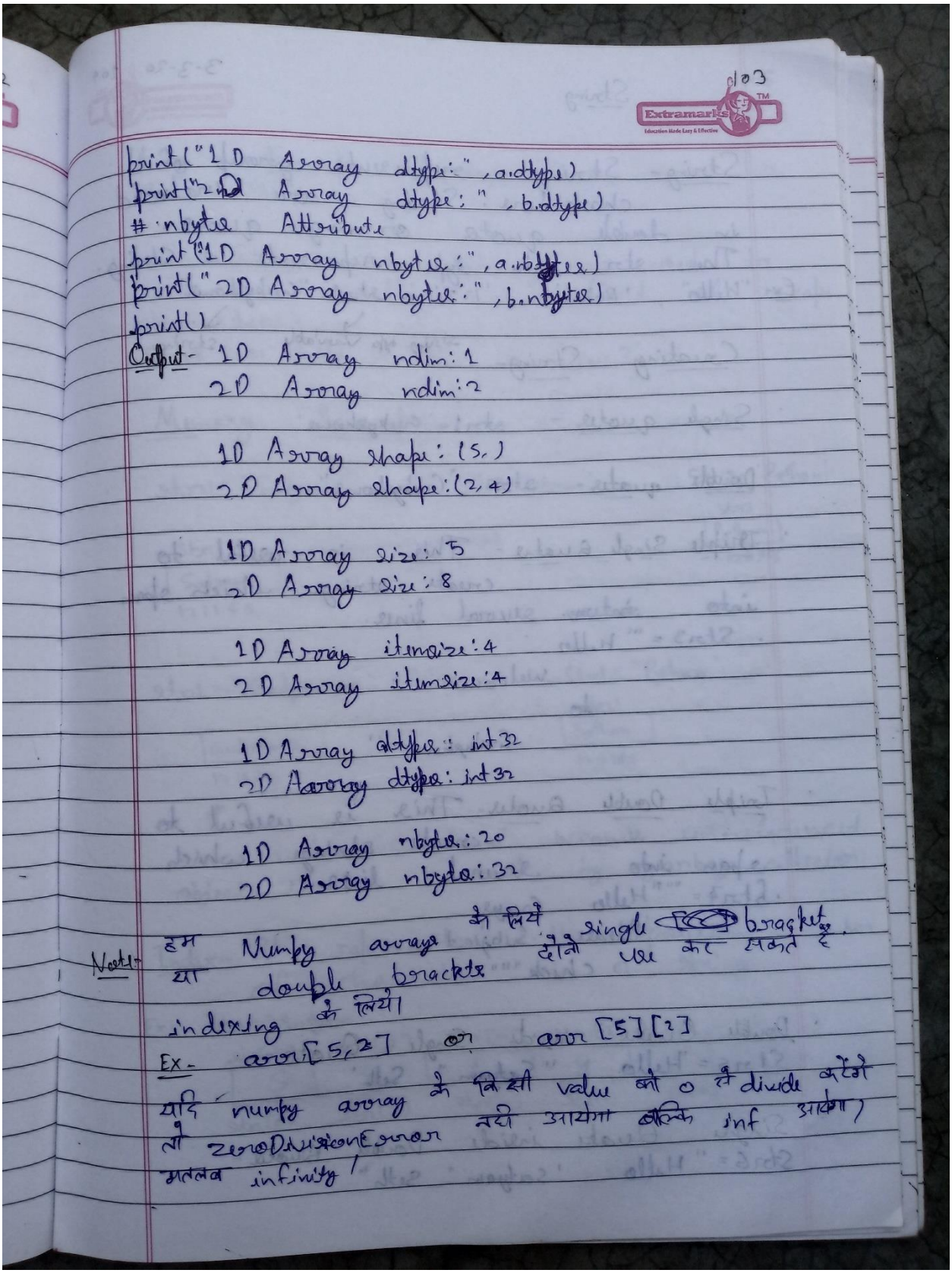
- **dtype**- This attribute represents the datatype of elements in the array.
  Syntax- arrayname.dtype



- nbytes - This attribute represents the total number of bytes occupied by an array.
  Total number of bytes = size of array * item size of each element in the array.

  Syntax - array_name.nbytes

→
```
from numpy import *
a=array([101,102,103,104,105])
b=array([[10,20,30,40],[50,60,70,80]])
print()
#ndim Attribute
print("1D Array ndim:",a.ndim)
print("2D Array ndim:",b.ndim)
print()
#shape Attribute
# For 1D Array shape Elements In The Row
# For 2D Array shape Specifies
  Number Of Row And Column In Each
  Row
print("1D Array shape:",a.shape)
print("2D Array shape:",b.shape)
print()
#size Attribute
print("1D Array size:",a.size)
print("2D Array size:",b.size)
print()
#itemsize Attribute
print("1D Array itemsize:",a.itemsize)
print("2D Array itemsize:",b.itemsize)
print()
#dtype Attribute
```



```
print("1D Array dtype:", a.dtype)
print("2nd Array dtype:", b.dtype)
# nbytes Attribute
print("1D Array nbytes:", a.nbytes)
print("2D Array nbytes:", b.nbytes)
print()
```

Output- 1D Array ndim: 1
2D Array ndim: 2

1D Array shape: (5,)
2D Array shape: (2, 4)

1D Array size: 5
2D Array size: 8

1D Array itemsize: 4
2D Array itemsize: 4

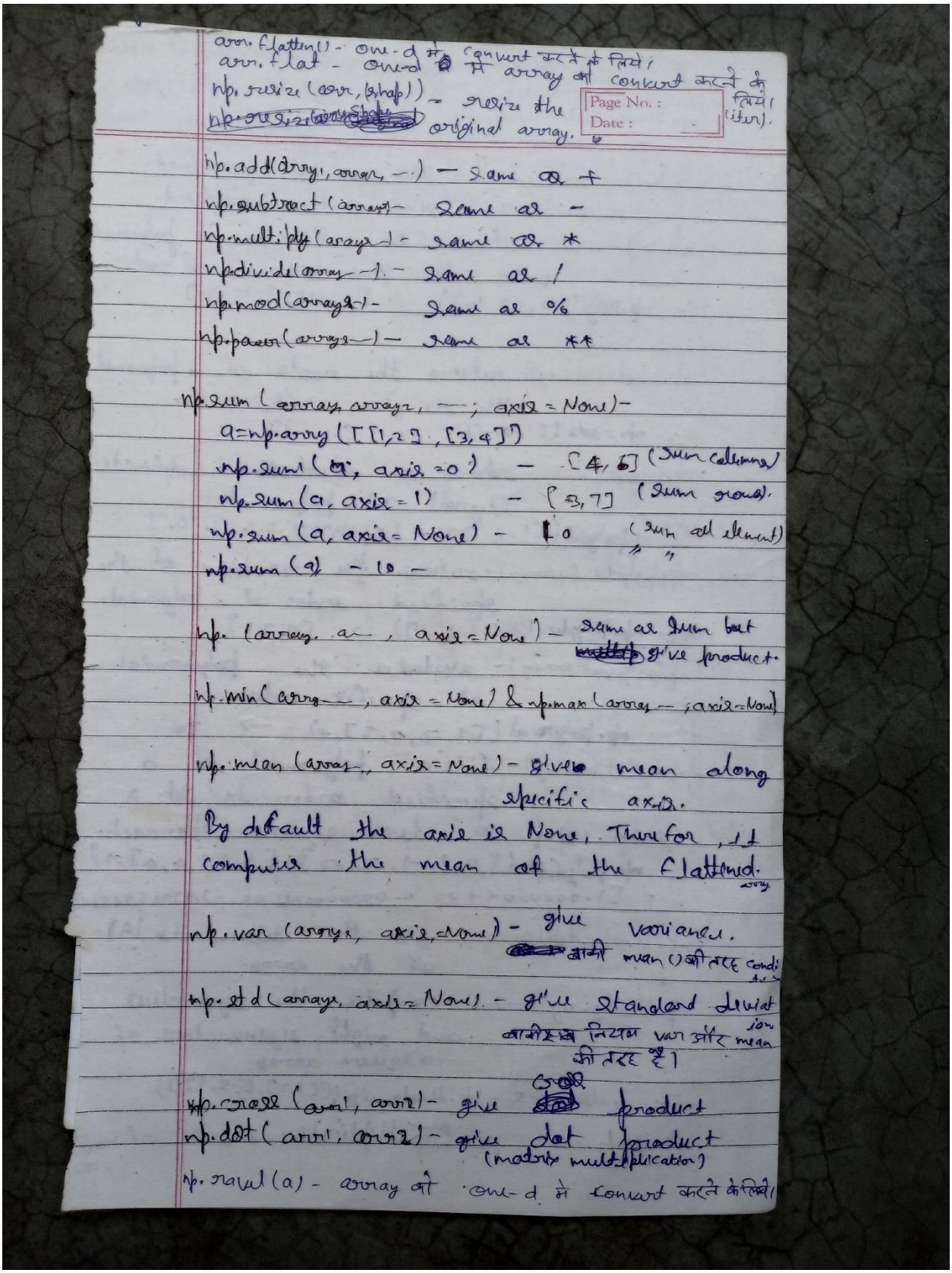
1D Array dtype: int32
2D Aarray dtype: int32

1D Array nbytes: 20
2D Array nbytes: 32

Note- हम Numpy array के लिये single bracket दोनों use कर सकते है. या double brackets indexing के लिये।

Ex- arr[5,2] or arr[5][2]

यदि numpy array के किसी value को 0 से divide करेंगे तो ZeroDivisionError नहीं आयेगा बल्कि inf आयेगा। मतलब infinity।



arr.flatten() - one-d में convert करने के लिये;
arr.flat - one-d में array को convert करने के लिये;
np.resize(arr, shape) - resize the original array.

np.add(array1, array2, ...) — same as +
np.subtract(array1, array2) — same as -
np.multiply(array1, array2) - same as *
np.divide(array1, array2) - same as /
np.mod(array1, array2) - same as %
np.power(array1, array2) — same as **

np.sum(array1, array2, ...; axis = None) -
a = np.array([[1,2], [3,4]])
np.sum(a, axis=0) — [4, 6] (sum columns)
np.sum(a, axis=1) — [3, 7] (sum rows)
np.sum(a, axis=None) — 10 (sum all elements)
np.sum(a) — 10 — " "

np.prod(array1, axis=None) - same as sum but give product.

np.min(array1, axis=None) & np.max(array1, axis=None)

np.mean(array1, axis=None) - gives mean along specific axis.
By default the axis is None, Therefore it computes the mean of the flattened array.

np.var(array1, axis=None) - give variance.
यानी mean() की तरह condition

np.std(array1, axis=None) - give standard deviation यानी इसका निराला var और mean की तरह है।

np.cross(arr1, arr2) - give cross product
np.dot(arr1, arr2) - give dot product (matrix multiplication)
np.ravel(a) - array को one-d में convert करने के लिये।

np.swapaxes(arr, indexofshap, indexofshap) :- ये shape को tuple return करता है इनके index को shape करता।

np.rollaxis(arr, indexofshp, indexofshap) - किसी shap के index को specific जगह पर shift करने के लिये।

np.inner(arr1, arr2) - give inner product.

np.outer(arr1, arr2) - give outer product.

np.poly(array) - returns the cofficients of polynomial with the given sequence of roots.

Ex- np.poly([-1, 1, 1, 10]) → [1. -11. 9. 11 -10]

np.roots(array) - returns the roots of a polynomial with the given cofficients.

Ex- np.roots([1, 0, -1]) → [-1. 1]

np.polyint(array) - returns an antiderivative (indefinite itegral) of a polynomial.

Ex- np.polyint([1, 1, 1]) → [0.33333333 0.5 1. 0.]

np.polyder(array) - returns the derivative of the specified order of a polynomial.

Ex- np.polyder([1, 1, 1, 1]) → [3 2 1]

np.polyval(array) - evaluates the polynomial at specific value.

Ex: np.polyval([1, 2, 0, 2], 4) → 34

np.polyfit(array) - fits a polynomial of a specified order to set a data using a least-squares approach.

Ex- np.polyfit([0, 1, -1, 2, -2], [0, 1, 1, 4, 4], 2)

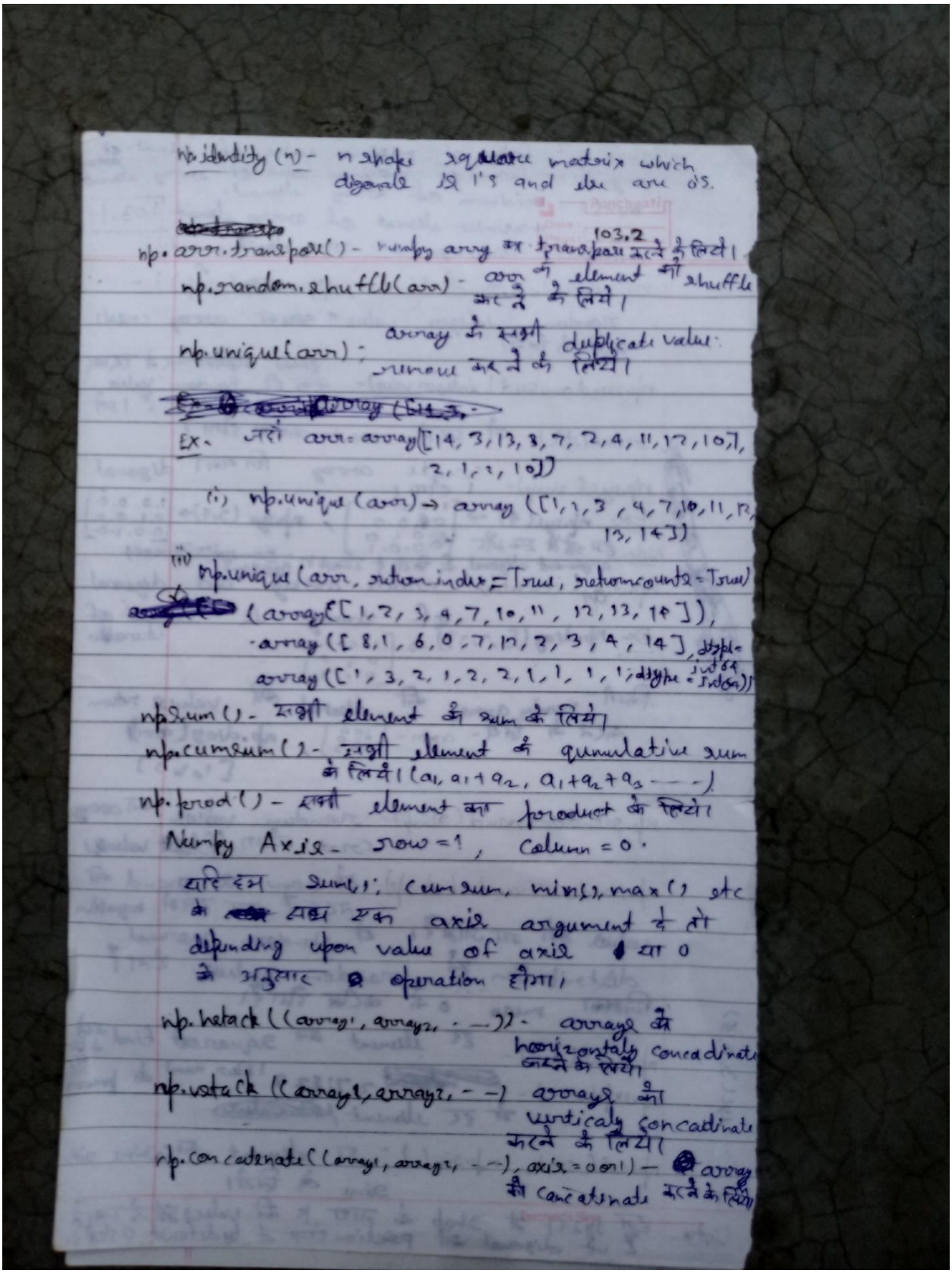
o/p- [1.0000000e+00 0.00000000e+00 -3.97205465e-16]

np.linalg.det(array) - give the determinate |A| of the array

np.linalg.eig(array) - computes the eigenvalues and right eigenvectors of a square array.

Ex- vals, vecs = np.linalg.eig([[1, 2], [2, 1]])

np.linalg.inv(array) - give multiplicative inverse of matrix



np.identity(n) - n shape square matrix which diagonals 1's and else are 0's.

np.arr.transpose() - numpy array का transpose करने के लिये। 103.2

np.random.shuffle(arr) - arr के element को shuffle करने के लिये।

np.unique(arr) - array के सभी duplicate values remove करने के लिये।

Ex- यदि arr = array([14, 3, 13, 8, 7, 2, 4, 11, 12, 10, 1, 2, 1, 2, 10])

(i) np.unique(arr) → array([1, 2, 3, 4, 7, 10, 11, 12, 13, 14])

(ii) np.unique(arr, return_index=True, return_counts=True)

(array([1, 2, 3, 4, 7, 10, 11, 12, 13, 14]), array([8, 1, 6, 0, 7, 12, 2, 3, 4, 14], dtype=int64), array([1, 3, 2, 1, 2, 2, 1, 1, 1, 1], dtype=int64))

np.sum() - सभी element के sum के लिये।

np.cumsum() - सभी element के cumulative sum के लिये। ($a_1, a_1+a_2, a_1+a_2+a_3$ ---)

np.prod() - सभी element का product के लिये।

Numpy Axis - row = 1, column = 0.

यदि हम sum(), cumsum, min(), max() etc के साथ एक axis argument दे तो depending upon value of axis 1 या 0 के अनुसार operation होगा।

np.hstack((array1, array2, ---)) - arrays को horizontally concatinate करने के लिये।

np.vstack((array1, array2, ---) arrays को vertically concatinate करने के लिये।

np.concatenate((array1, array2, ---), axis=0 or 1) - arrays को concatenate करने के लिये

argmin() - index of maximum value of element of array
argmax() - index of maximum value of array element
min() - minimum of array element.
max() - maximum element of array element. [103.1]
midean() - avarage of array element (row, col.)

np.random.randint(start, stop, (shap)) - random integer value का array create करने के लिये।

np.random.seed(integer value) - इसका उपयोग करने से हर बार एक ही Random value generate कर सकते है। इसे randint() से पहले execute करना होगा।

np.eye(shap) - create array जिसका digonal 1 होगा।

Ex- np.eye(4) → [[1. 0. 0. 0.], [0. 1. 0. 0.], [0. 0. 1. 0.], [0. 0. 0. 1.]], np.eye(3,4) → [[1. 0. 0. 0.], [0. 1. 0. 0.], [0. 0. 1. 0.]]

Note- हम चाहे तो एक और argument digonal के लिये दे सकते है। (position)

np.diag(iterable) - create array and digonal of array is element of iterable

Ex- np.diag([1,2,3]) → [[1 0 0], [0 2 0], [0 0 3]]

किसी numpy array की digonal की values return करने के लिये - arr = [[1 0 0], [0 2 0], [0 0 3]], np.diag(arr) → [1 2 3]

np.random.rand(shap) - random values का array create करेगा (float values)

np.random.randn(shap) - ये rand की तरह है बस इसमे negative values भी आ जाती है। ये standard normal distribution से random values देता है जिनका mean 0 के करीब होता है

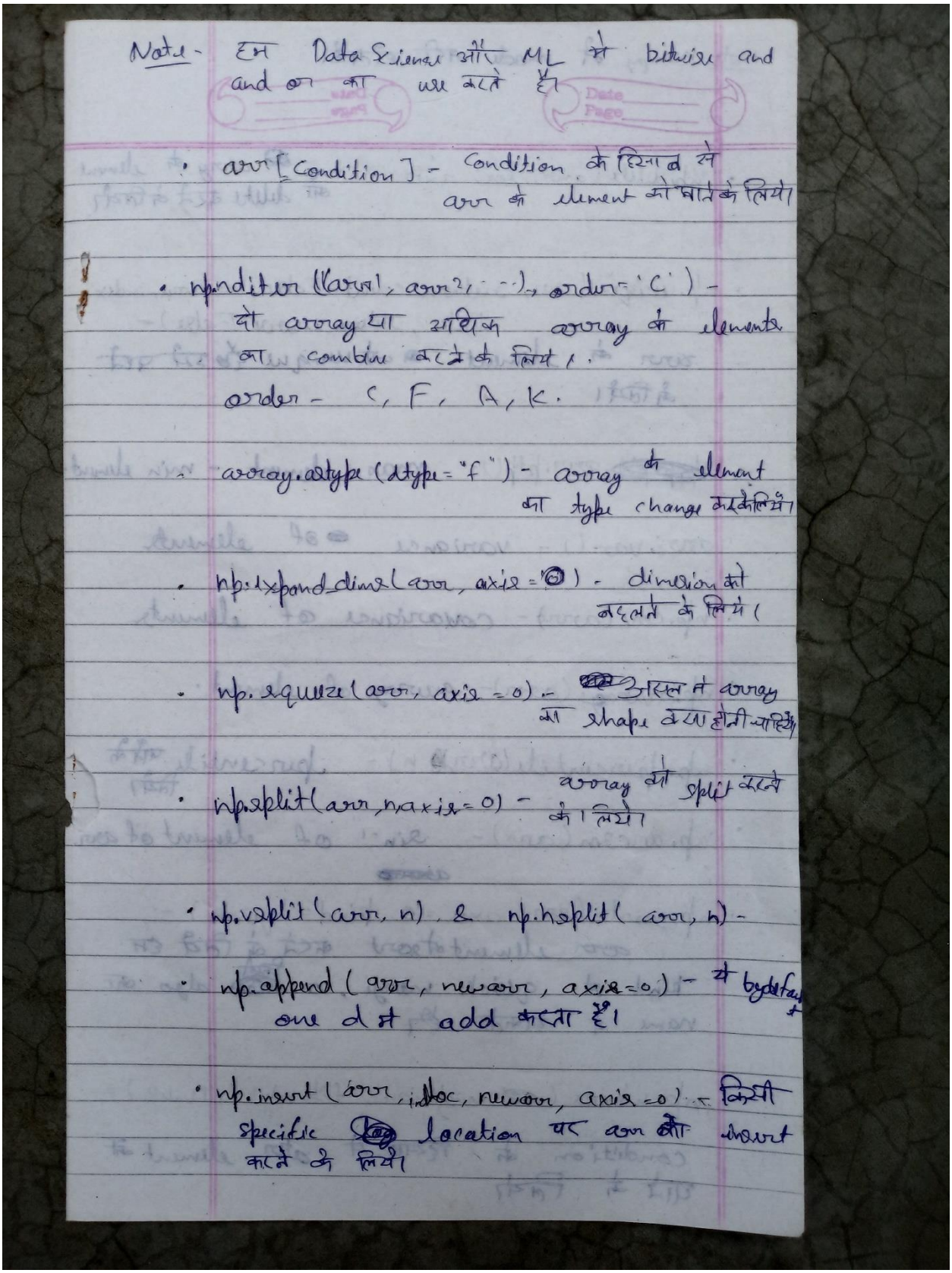
np.sqrt(arr) - हर element का square find करते है।

np.exp(arr) - 2.71828 (Euler num) के power मे हर element

np.cos(arr), np.sin() - हर element का cosine और sine के लिये।

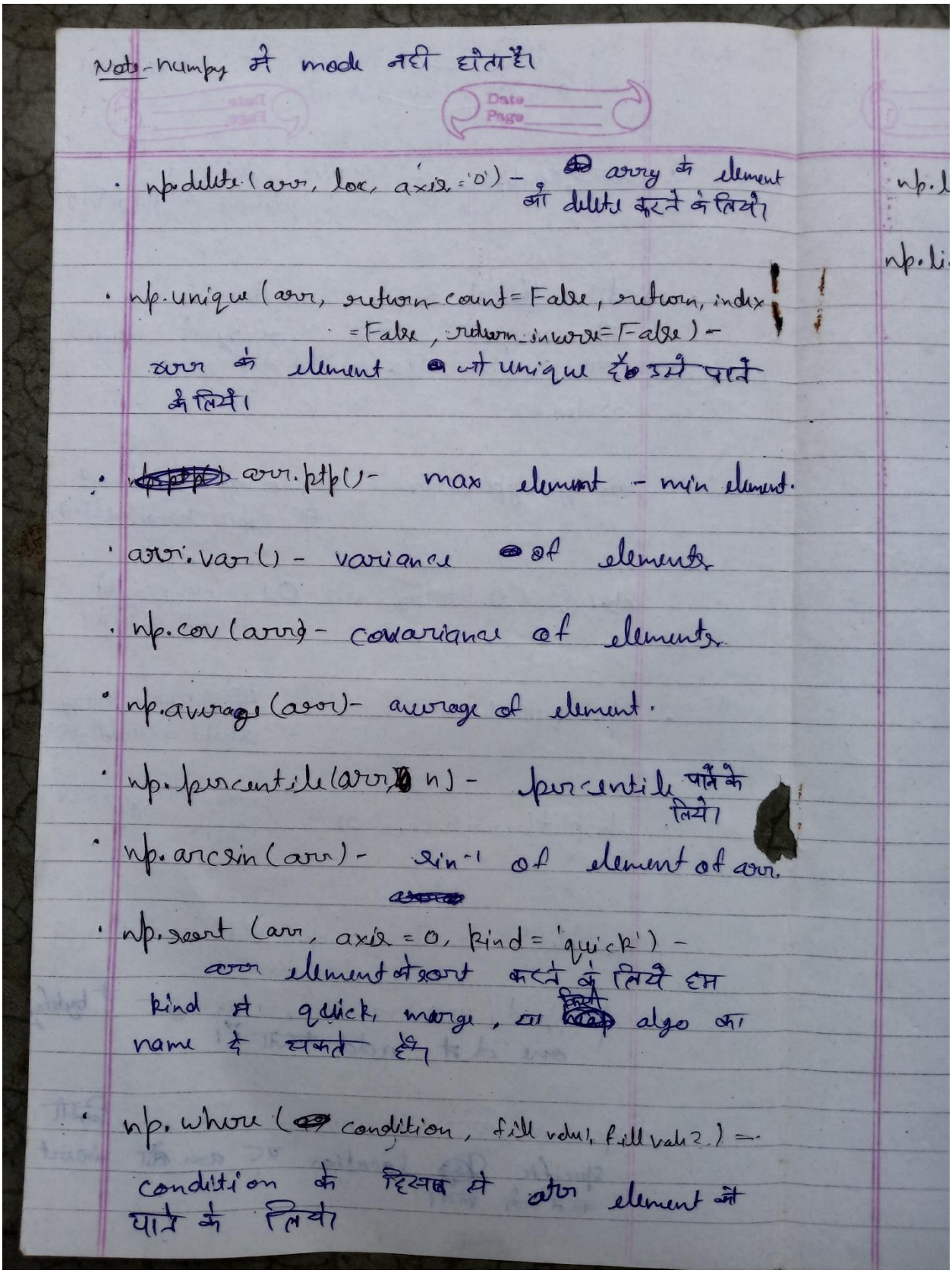
Note- हम eye() मे shap के साथ k की value भी दे सकते है जो digonal की position होगा ये by default 0 होती है।

Ex- eye(3,4,5)



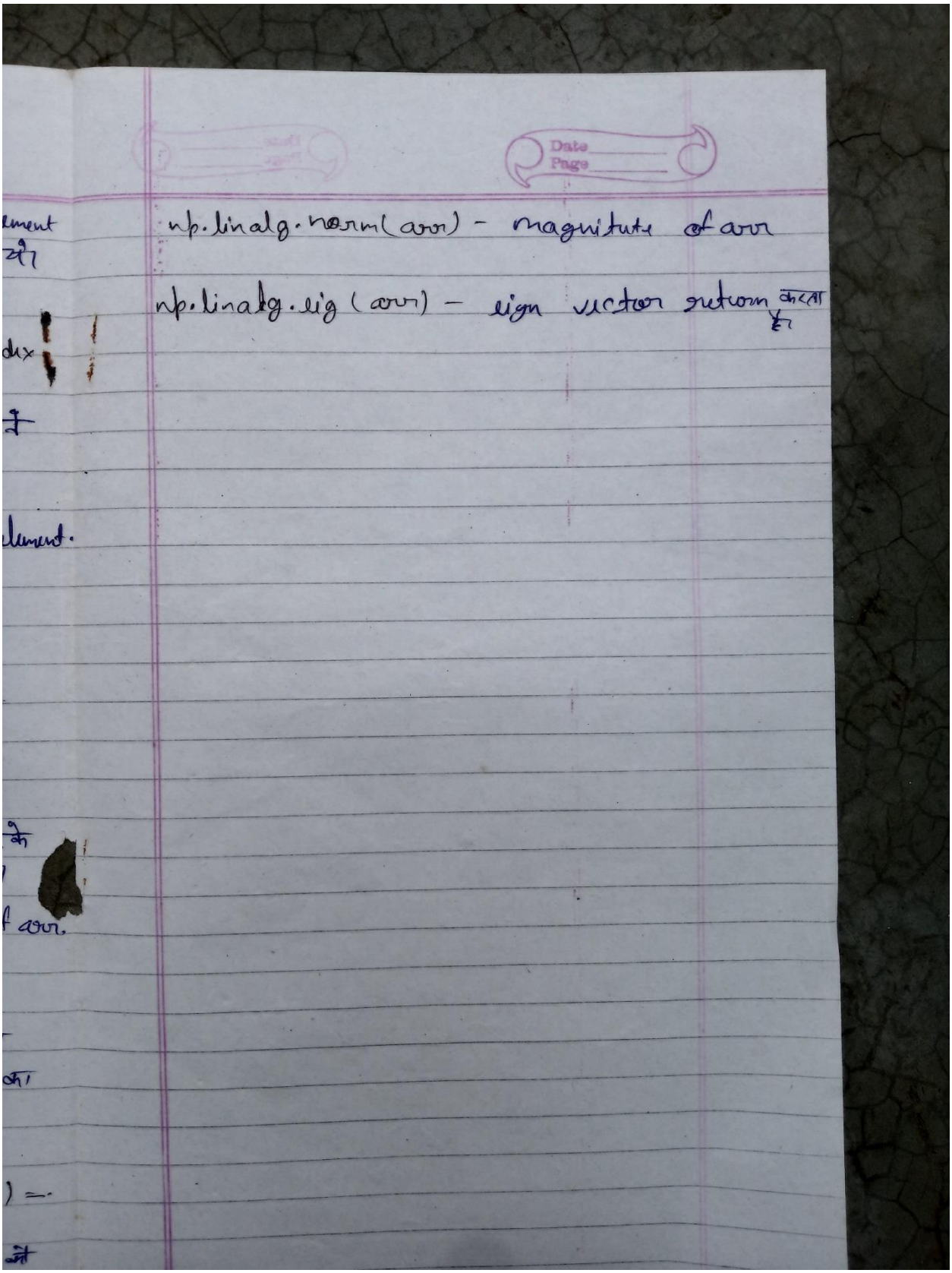
Note - हम Data Science और ML में bitwise and and का use करते हैं।

- arr[Condition] - Condition के हिसाब से arr के element को पाने के लिये।
- np.nditer([arr1, arr2, ...], order='C') - दो array या अधिक array के elements का combine करने के लिये।
  order - C, F, A, K.
- array.astype(dtype="f") - array के element का type change करने के लिये।
- np.expand_dims(arr, axis=0) - dimension को बढ़ाने के लिये।
- np.squeeze(arr, axis=0) - असल में array का shape कैसा होना चाहिये।
- np.split(arr, n, axis=0) - array को split करने के लिये।
- np.vsplit(arr, n) & np.hsplit(arr, n) -
- np.append(arr, newarr, axis=0) - ये by default one d में add करता है।
- np.insert(arr, idloc, newarr, axis=0) - किसी specific location पर arr को insert करने के लिये।



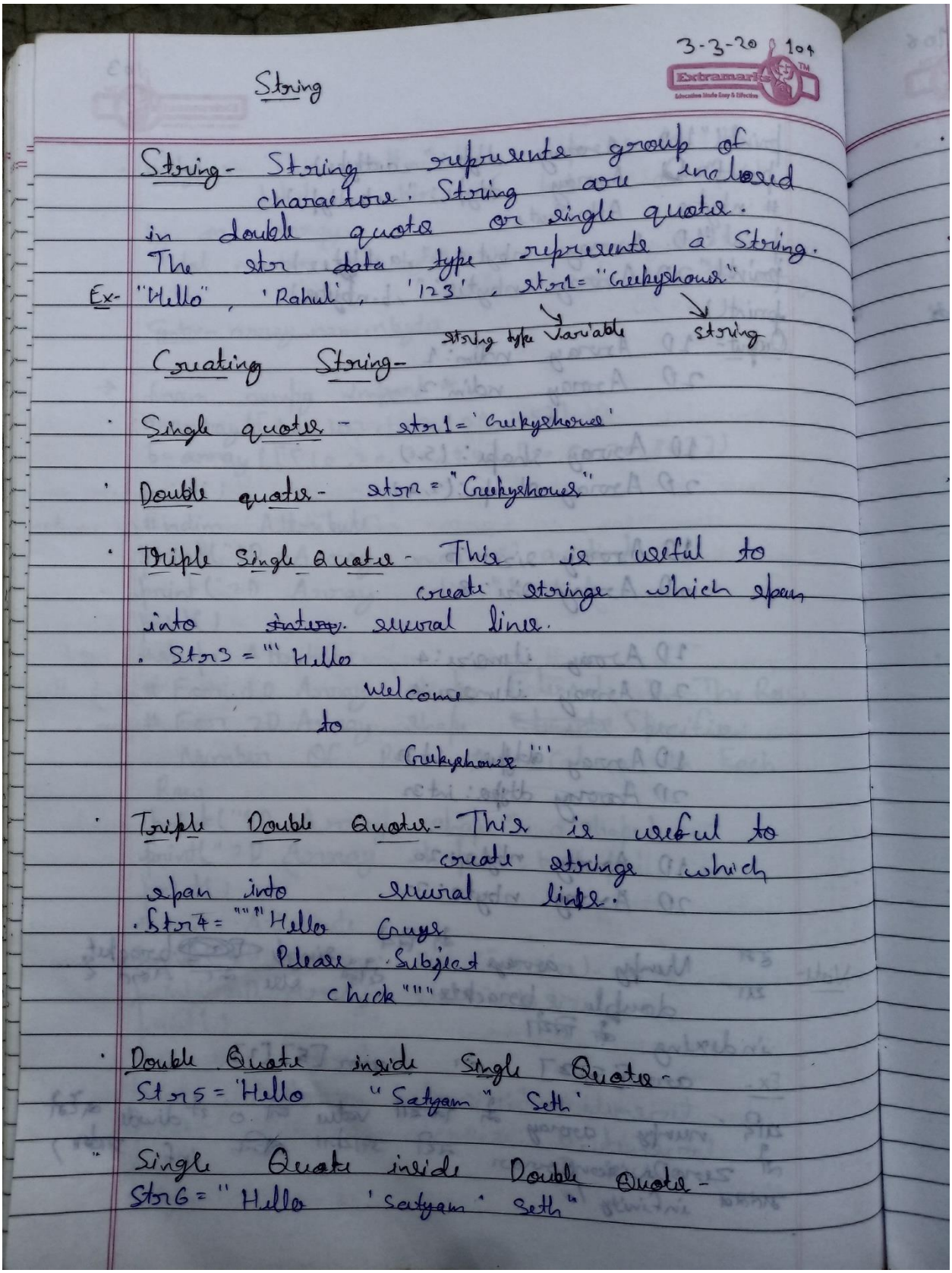
Note - numpy में mode नहीं होता है।

- np.delete(arr, loc, axis='0') - arrय के element को delete करने के लिये।
- np.unique(arr, return_count=False, return_index=False, return_inverse=False) - arr के element को unique देखने के लिये।
- arr.ptp() - max element - min element.
- arr.var() - variance of elements.
- np.cov(arr) - covariance of elements.
- np.average(arr) - average of element.
- np.percentile(arr, n) - percentile पाने के लिये।
- np.arcsin(arr) - sin-1 of element of arr.
- np.sort(arr, axis=0, kind='quick') - arr element को sort करने के लिये इस kind में quick, merge, algo का name दे सकते हैं।
- np.where(condition, fill val1, fill val2) - condition के हिसाब से arr element को पाने के लिये।



np.linalg.norm(arr) - magnitute of arr

np.linalg.sig(arr) - sign vector return करता है



3-3-20 104

String

String - String represents group of characters. String are inclosed in double quotes or single quotes. The str data type represents a String.

Ex- "Hello", 'Rahul', '123', str1 = "Geekyshows"
(str1 → string type variable; "Geekyshows" → string)

Creating String -

- **Single quotes** - str1 = 'Geekyshows'
- **Double quotes** - str2 = "Geekyshows"
- **Triple Single Quotes** - This is useful to create strings which span into several lines.

```
Str3 = ''' Hello
           Welcome
               to
           Geekyshows '''
```

- **Triple Double Quotes** - This is useful to create strings which span into several lines.

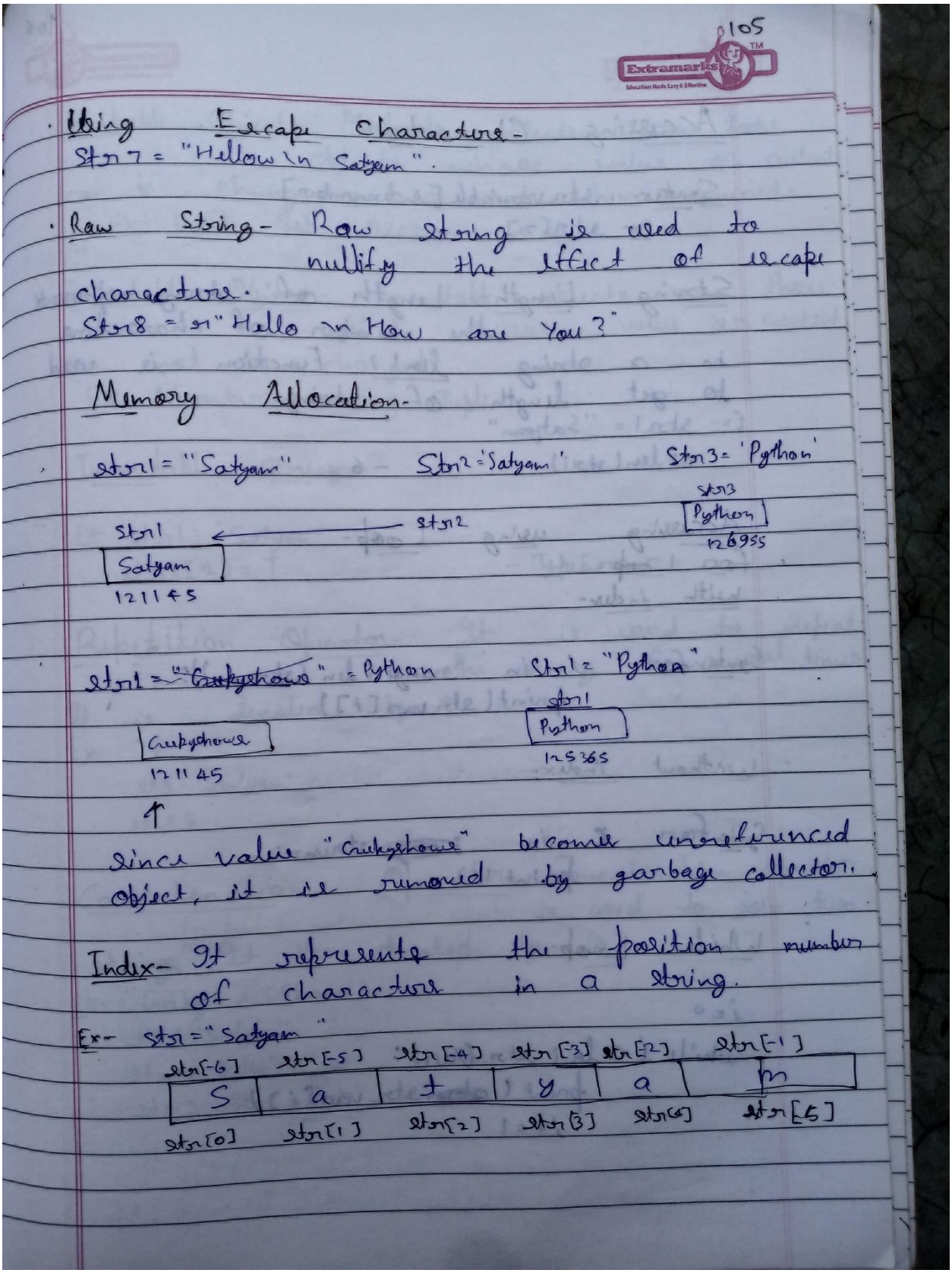
```
Str4 = """ Hello Guys
       Please Subject
           check """
```

- **Double Quote inside Single Quotes** -

```
Str5 = 'Hello "Satyam" Seth'
```

- **Single Quote inside Double Quotes** -

```
Str6 = " Hello 'Satyam' Seth"
```



105

Extramarks

- Using Escape Characters -

Str7 = "Hellow \n Satyam".

- Raw String - Raw string is used to nullify the effect of escape characters.

Str8 = r"Hello \n How are You?"

Memory Allocation -

str1 = "Satyam" Str2 = 'Satyam' Str3 = 'Python'

str3
Python
126955

str1 ← str2
Satyam
121145

str1 = "Geekyshows" = Python Str1 = "Python"

str1
Python
125365

Geekyshows
121145

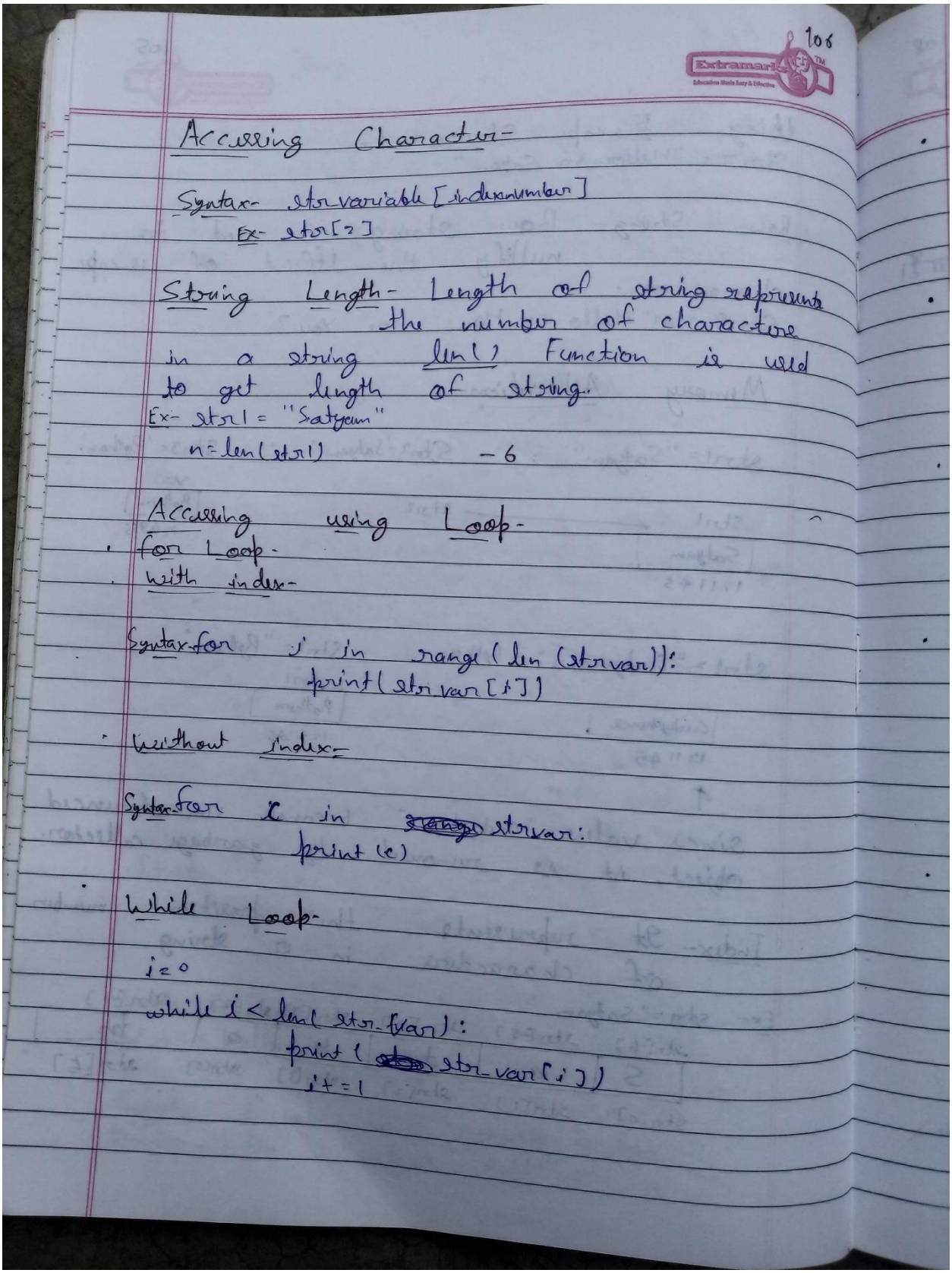
↑

Since value "Geekyshows" become unreferenced object, it is removed by garbage collector.

Index - It represents the position number of characters in a string.

Ex - str = "Satyam"

str[-6]	str[-5]	str[-4]	str[-3]	str[-2]	str[-1]
S	a	t	y	a	m
str[0]	str[1]	str[2]	str[3]	str[4]	str[5]



108

Accessing Character-

Syntax- str variable [index number]
Ex- str[2]

String Length- Length of string represents the number of characters in a string len() Function is used to get length of string.
Ex- str1 = "Satyam"
n = len(str1) -6

Accessing using Loop-
- for Loop-
- with index-

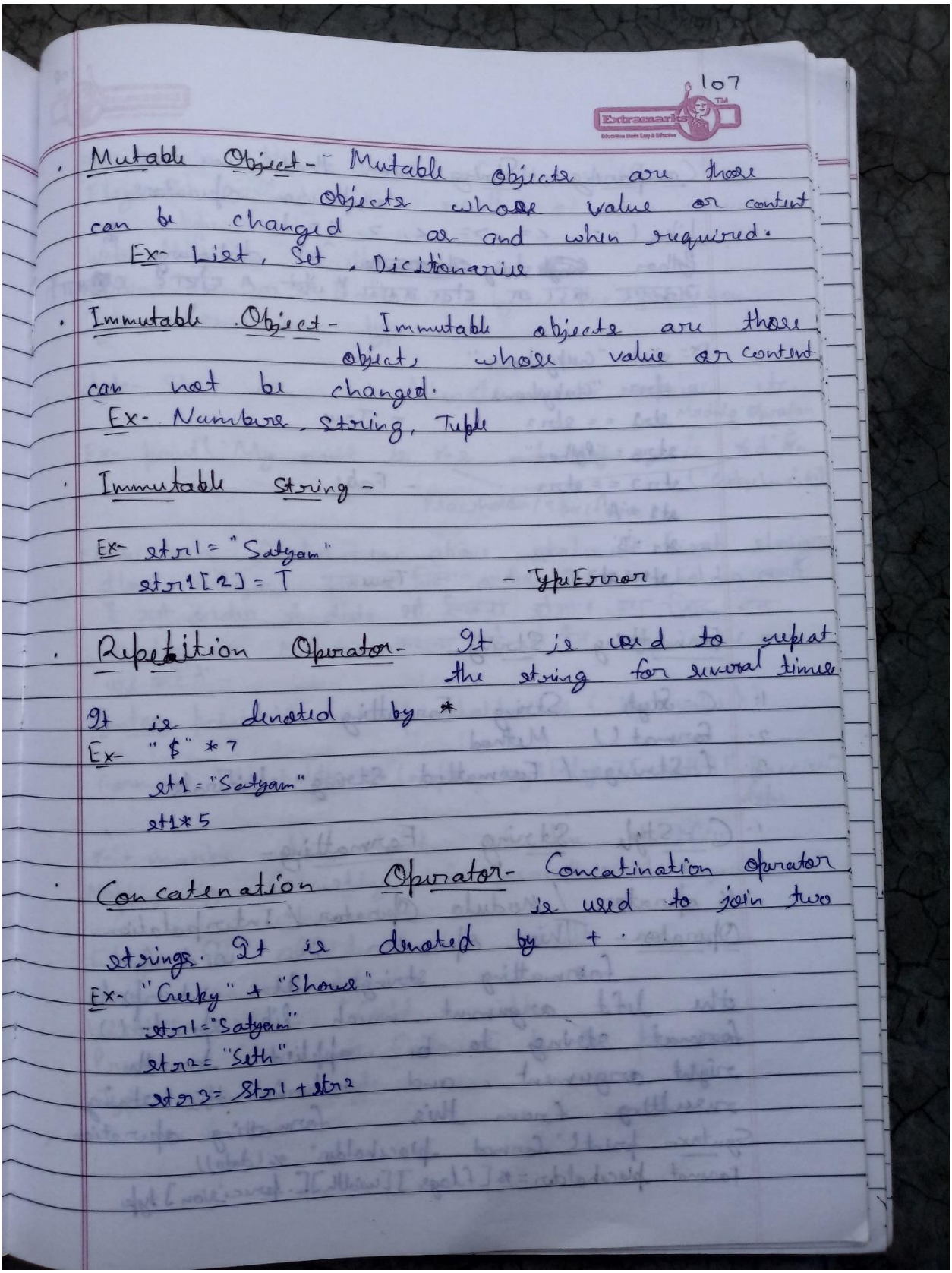
Syntax for i in range (len (str var)):
print (str var [i])

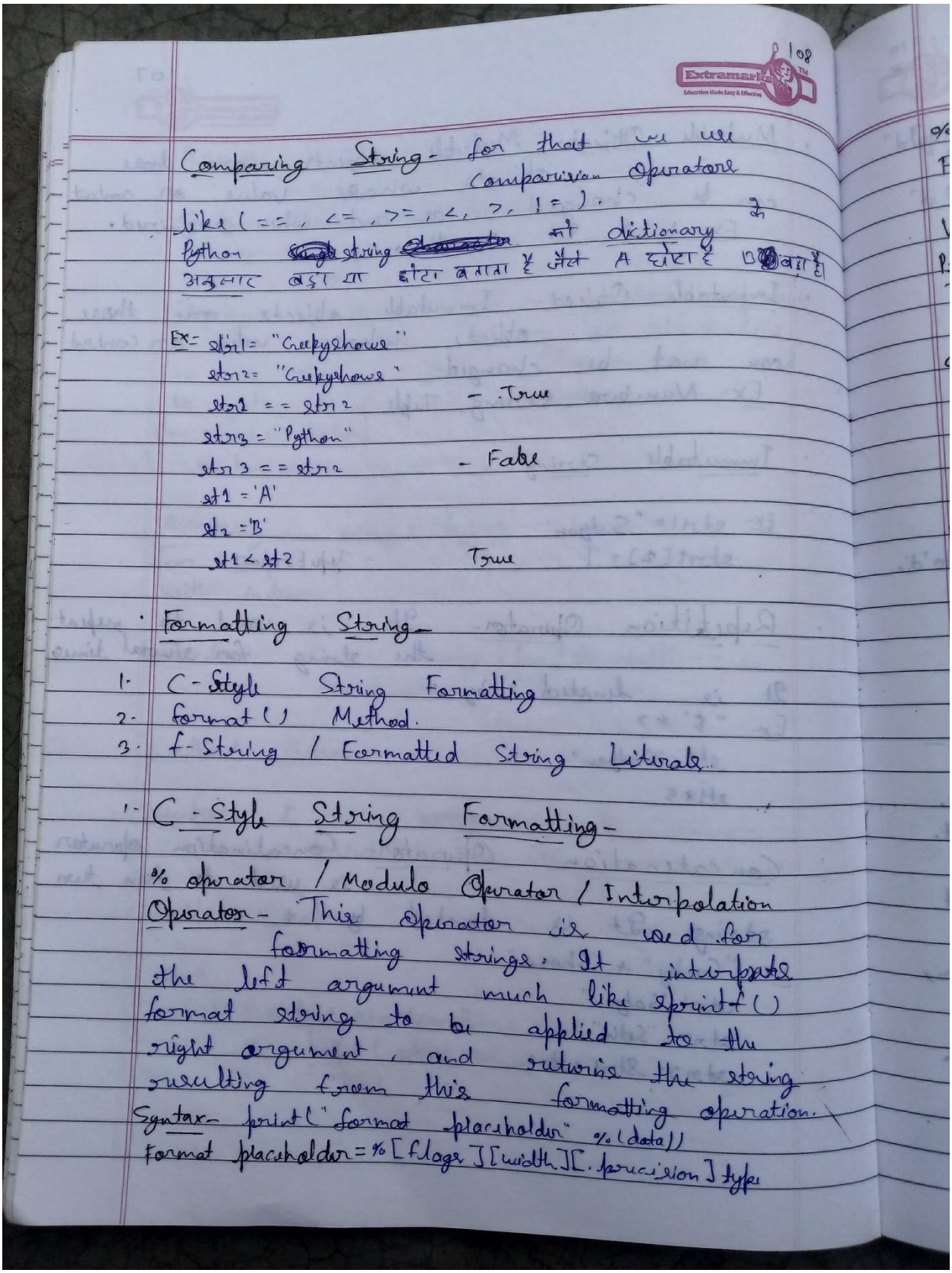
- without index-

Syntax for c in str var:
print (c)

While Loop-

i = 0
while i < len (str_var):
print (str_var [i])
i += 1


107
Extramarks
Mutable Object - Mutable objects are those objects whose value or content can be changed as and when required.
Ex- List, Set, Dictionaries
Immutable Object - Immutable objects are those object, whose value or content can not be changed.
Ex- Number, String, Tuple
Immutable String -
Ex- str1 = "Satyam"
str1[2] = T - TypeError
Repetition Operator - It is used to repeat the string for several times. It is denoted by *
Ex- "$" * 7
st1 = "Satyam"
st1 * 5
Concatenation Operator - Concatination operator is used to join two strings. It is denoted by +.
Ex- "Geeky" + "Shows"
str1 = "Satyam"
str2 = "Seth"
str3 = Str1 + str2



Comparing String - for that we use comparision operators like (==, <=, >=, <, >, !=).
Python string dictionary
अनुसार बड़ा या छोटा बताता है जैसे A छोटा है B बड़ा है

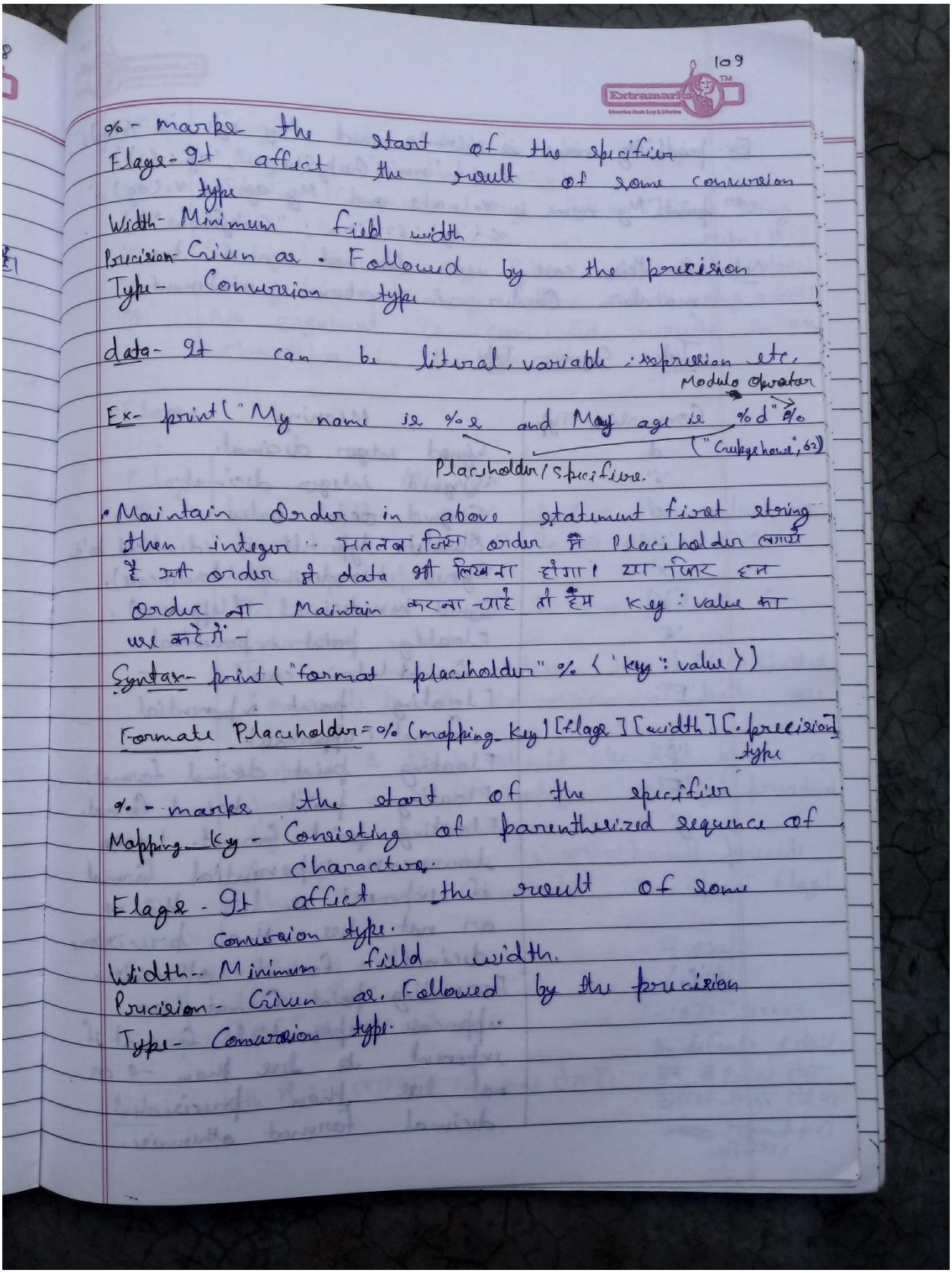
Ex- str1 = "Geekyshows"
str2 = "Geekyshows"
str1 == str2 - True
str3 = "Python"
str3 == str2 - False
st1 = 'A'
st2 = 'B'
st1 < st2 True

- Formatting String -

1. C-Style String Formatting
2. format() Method.
3. f-String / Formatted String Literals.

1. C-Style String Formatting -

% operator / Modulo Operator / Interpolation Operator - This operator is used for formatting strings. It interprets the left argument much like sprintf() format string to be applied to the right argument, and returns the string resulting from this formatting operation.
Syntax - print("format placeholder" %(data))
Format placeholder = %[flags][width][.precision]type



109

% - marks the start of the specifier
Flags - It affect the result of some conversion type
Width - Minimum field width
Precision - Given as . Followed by the precision
Type - Conversion type

data - It can be literal, variable, expression etc.

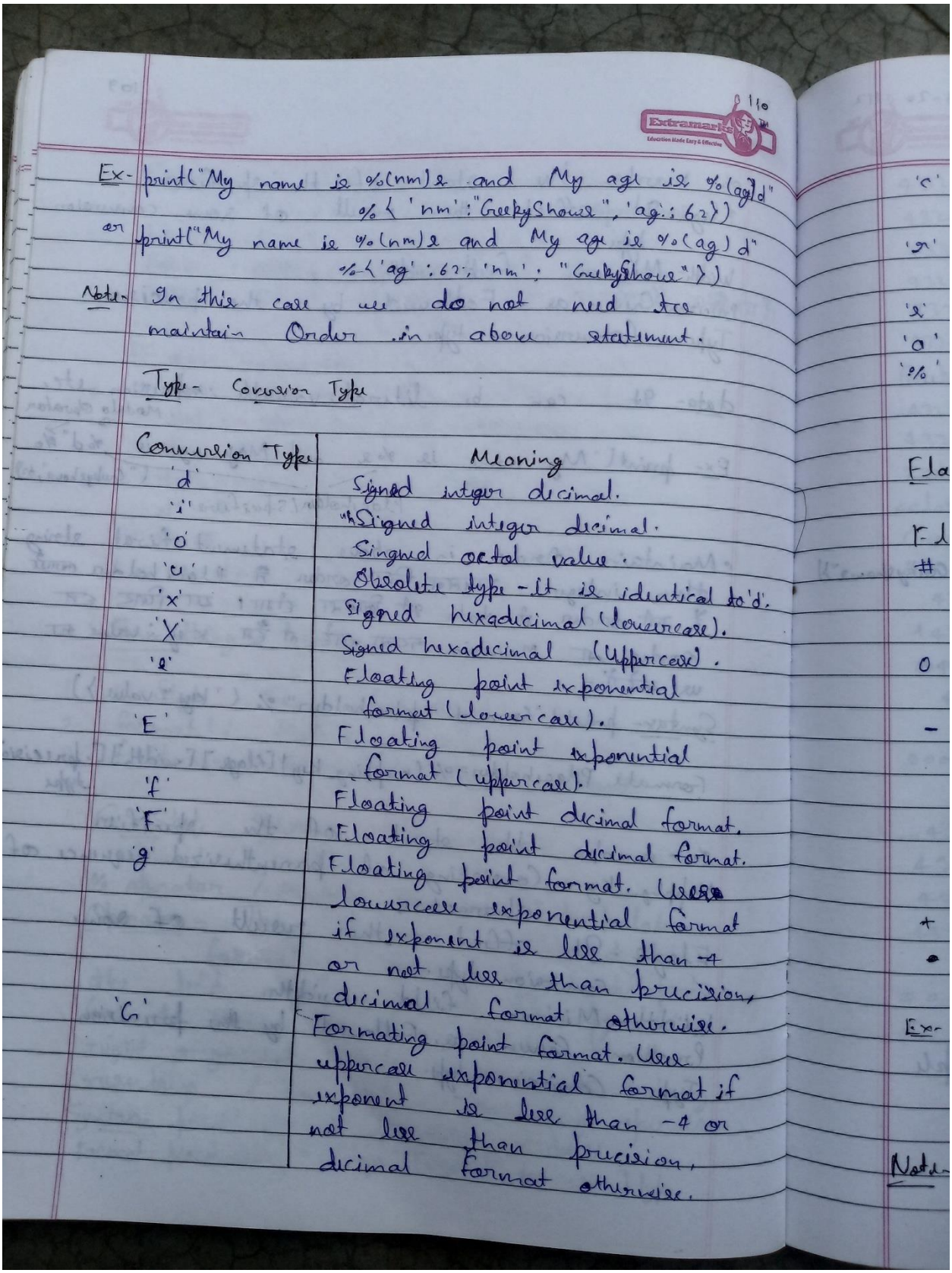
Ex - print("My name is %s and May age is %d" %("Sachin", 62))
Modulo Operator
Placeholder / Specifiers.

• Maintain Order in above statement first string then integer. सबसे पहले order में placeholder लगाये हैं उसी order में data भी लिखना होगा। या फिर हम order को Maintain करना चाहें तो हम key : value का use करते हैं -

Syntax - print("format placeholder" % {'key': value})

Formate Placeholder = % (mapping_key) [flags] [width] [.precision] type

% - marks the start of the specifier
Mapping_key - Consisting of parenthesized sequence of characters.
Flags - It affect the result of some conversion type.
Width - Minimum field width.
Precision - Given as. Followed by the precision
Type - Conversion type.



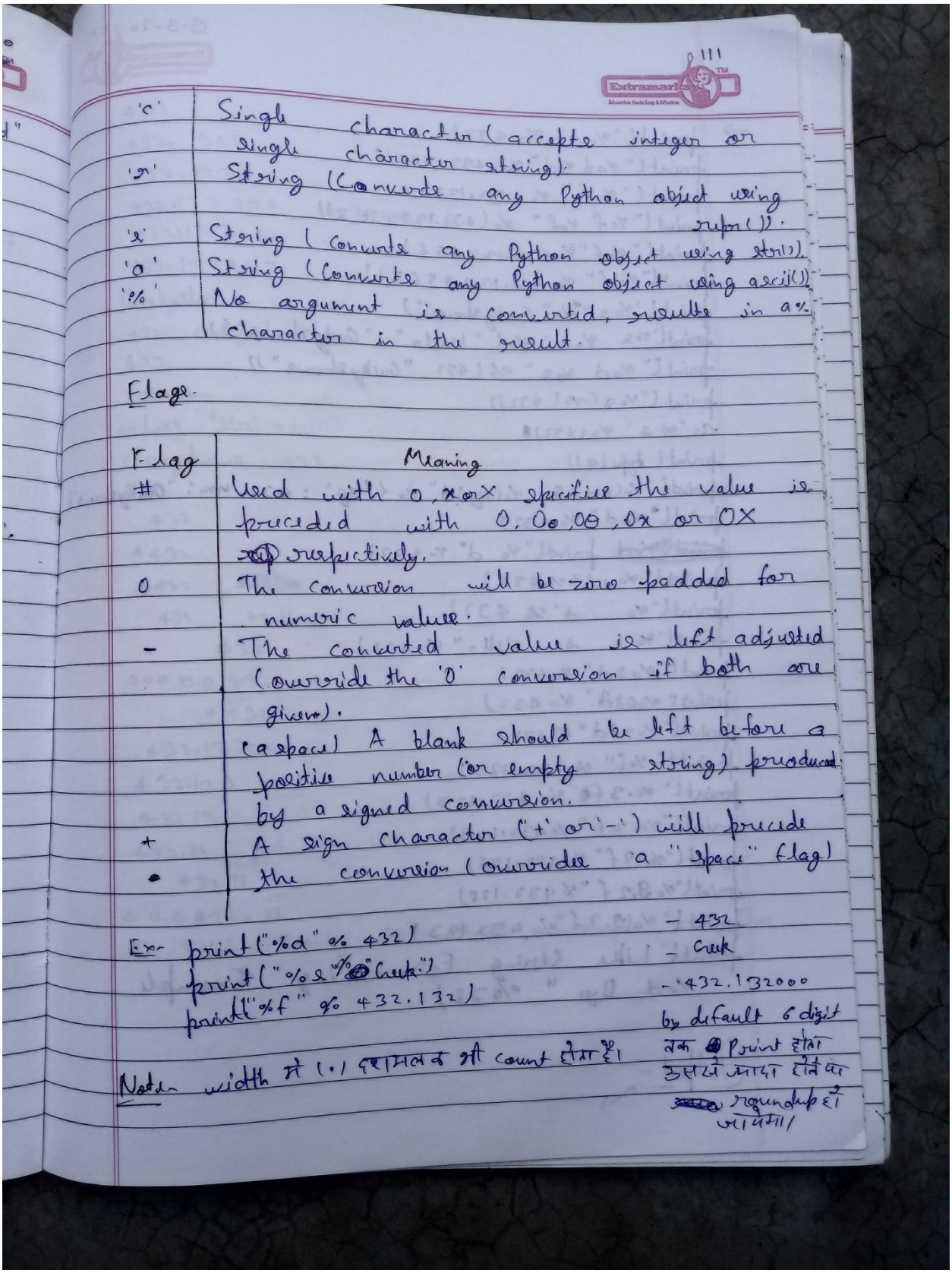
Ex- print("My name is %(nm)s and My age is %(ag)d" %{'nm':"GeekyShows", 'ag':62})

or print("My name is %(nm)s and My age is %(ag)d" %{'ag':62, 'nm':"GeekyShows"})

Note- In this case we do not need to maintain Order in above statement.

Type- Conversion Type

Conversion Type	Meaning
'd'	Signed integer decimal.
'i'	Signed integer decimal.
'o'	Signed octal value.
'u'	Obsolete type - it is identical to 'd'.
'x'	Signed hexadecimal (lowercase).
'X'	Signed hexadecimal (Uppercase).
'e'	Floating point exponential format (lowercase).
'E'	Floating point exponential format (uppercase).
'f'	Floating point decimal format.
'F'	Floating point decimal format.
'g'	Floating point format. Uses lowercase exponential format if exponent is less than -4 or not less than precision, decimal format otherwise.
'G'	Formating point format. Uses uppercase exponential format if exponent is less than -4 or not less than precision, decimal format otherwise.



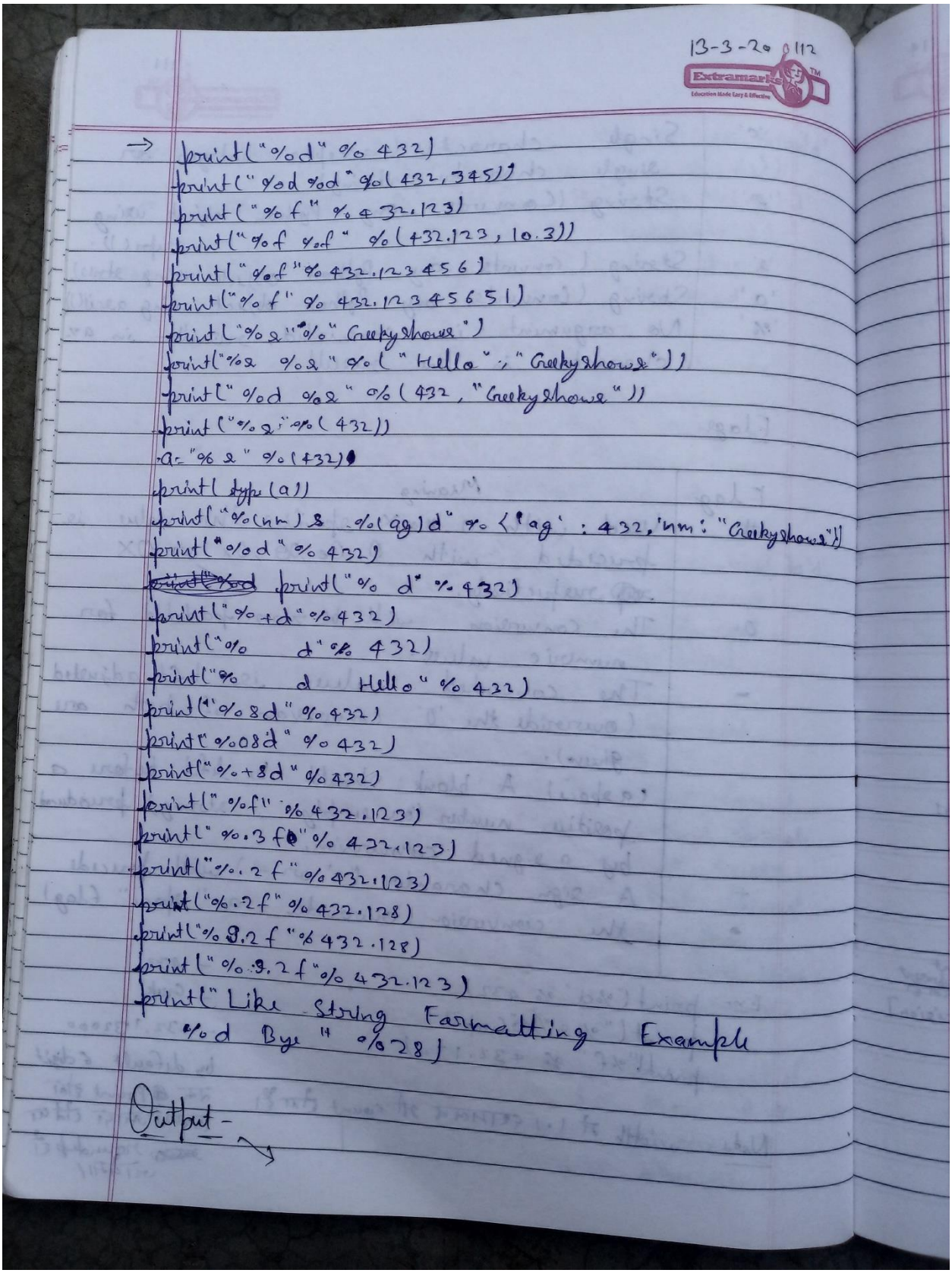
'c'	Single character (accepts integer or single character string).
'r'	String (Converts any Python object using repr()).
's'	String (Converts any Python object using str()).
'a'	String (Converts any Python object using ascii()).
'%'	No argument is converted, results in a % character in the result.

Flags.

Flag	Meaning
#	Used with o, x or X specifies the value is preceded with 0, 0o, 0O, 0x or 0X respectively.
0	The conversion will be zero padded for numeric values.
-	The converted value is left adjusted (overrides the '0' conversion if both are given).
(a space)	A blank should be left before a positive number (or empty string) produced by a signed conversion.
+	A sign character ('+' or '-') will precede the conversion (overrides a "space" flag)

Ex- print("%d" % 432) → - 432
print("%s" % "Geek") → - Geek
print("%f" % 432.132) → - 432.132000 by default 6 digit तक Print होगा उसके आगे होने पर roundup हो जायेगा।

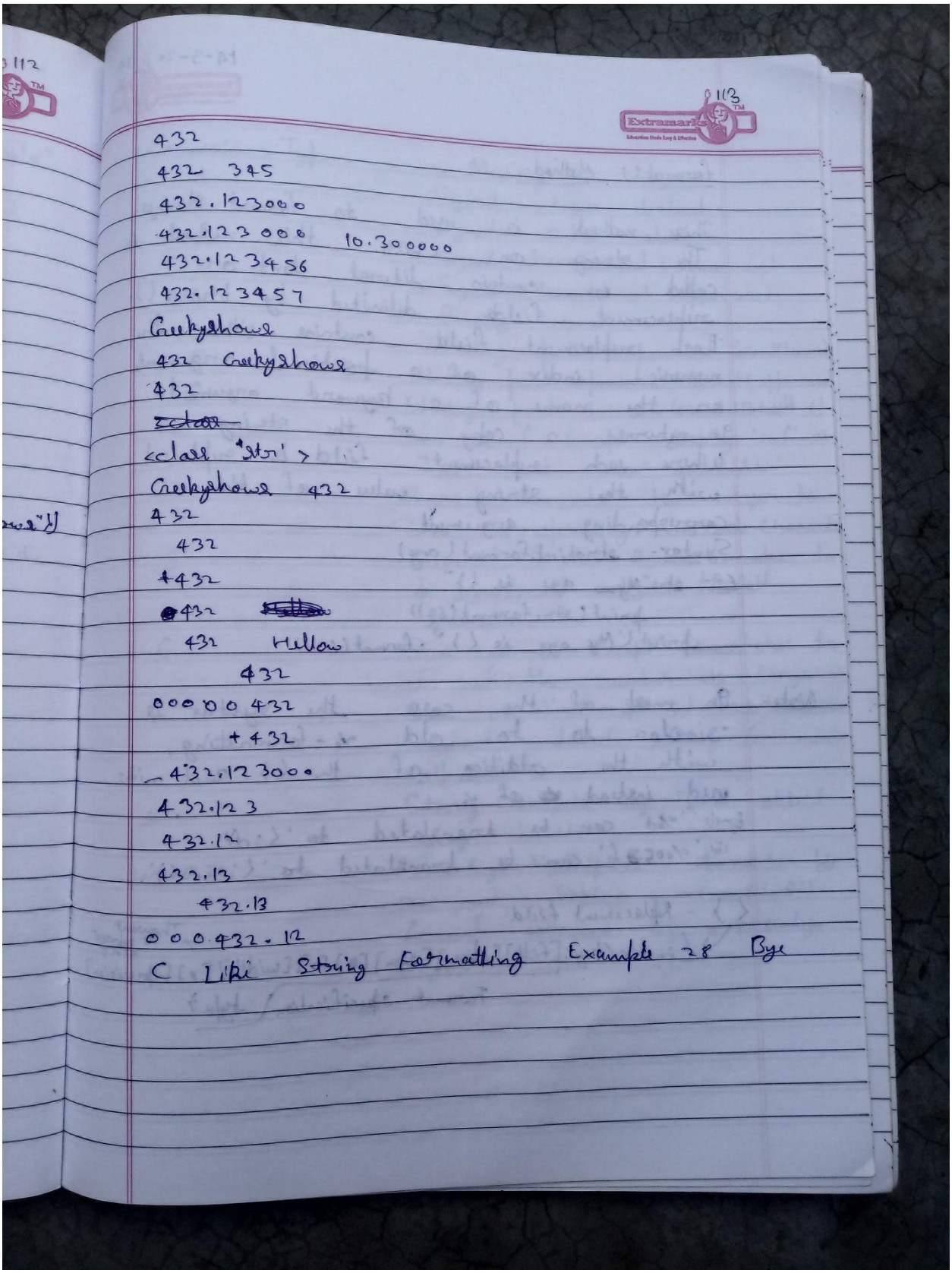
Note- width में (.) फ्लासत भी count होता है।

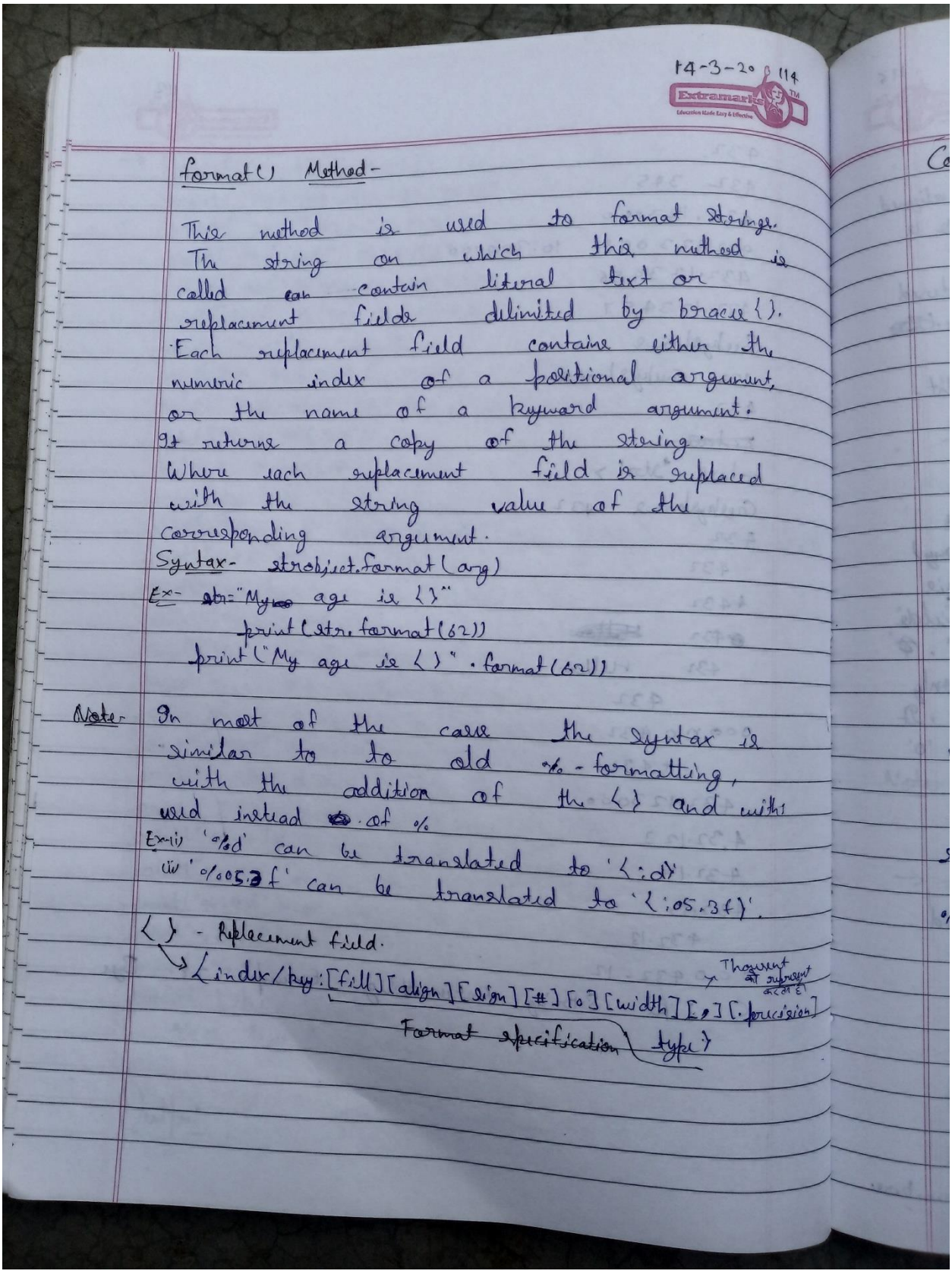


13-3-20 112

```
print("%d" % 432)
print("%d %d" %(432,345))
print("%f" % 432.123)
print("%f %f" %(432.123, 10.3))
print("%f" %432.123456)
print("%f" % 432.12345651)
print("%s" %"Geekyshows")
print("%s %s" %("Hello", "Geekyshows"))
print("%d %s" %(432, "Geekyshows"))
print("%s" %(432))
a="%s" %(432)
print(type(a))
print("%(nm)s %(ag)d" %{'ag': 432, 'nm': "Geekyshows"})
print("%0d" % 432)
print("% d" %432)
print("%+d" %432)
print("%     d" %432)
print("%     d Hello" %432)
print("%8d" %432)
print("%08d" %432)
print("%+8d" %432)
print("%f" %432.123)
print("%.3f" %432.123)
print("%.2f" %432.123)
print("%.2f" %432.128)
print("%8.2f" %432.128)
print("%08.2f" %432.123)
print("Like String Formatting Example
    %d Bye" %28)
```

Output -


432
432 345
432.123000
432.123000 10.300000
432.123456
432.123457
Geekyshows
432 Geekyshows
432
<class 'str'>
Geekyshows 432
432
432
+432
432
432 Hellow
432
00000432
+432
-432.123000
432.123
432.12
432.13
+32.13
000432.12
C Like String Formatting Example 28 Bye



14-3-20 114

Extramarks

format() Method-

This method is used to format strings. The string on which this method is called can contain literal text or replacement fields delimited by braces {}. Each replacement field contains either the numeric index of a positional argument, or the name of a keyword argument. It returns a copy of the string. Where each replacement field is replaced with the string value of the corresponding argument.

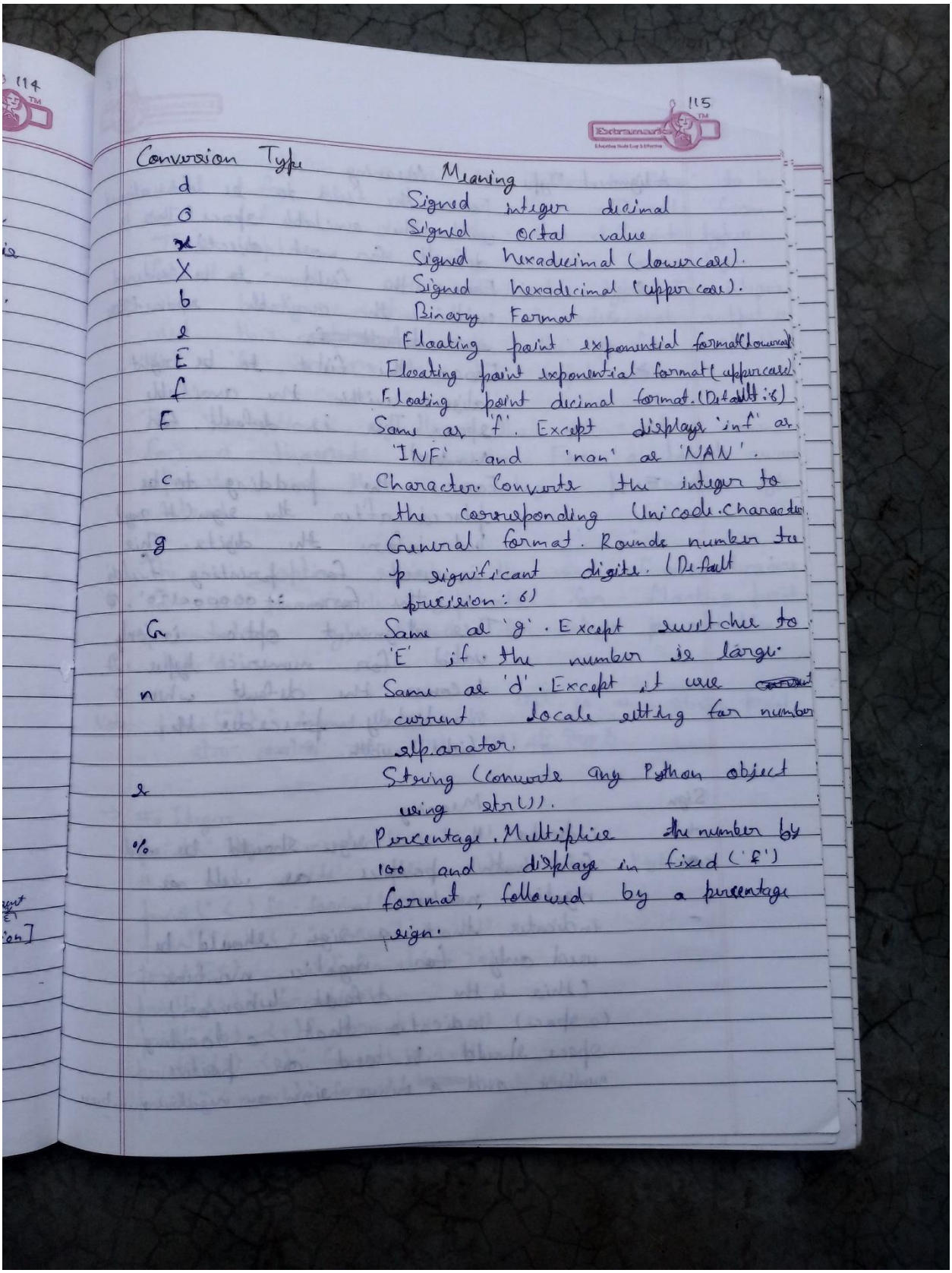
Syntax- strobject.format(arg)

Ex- str="My age is {}"
print(str.format(62))
print("My age is {}".format(62))

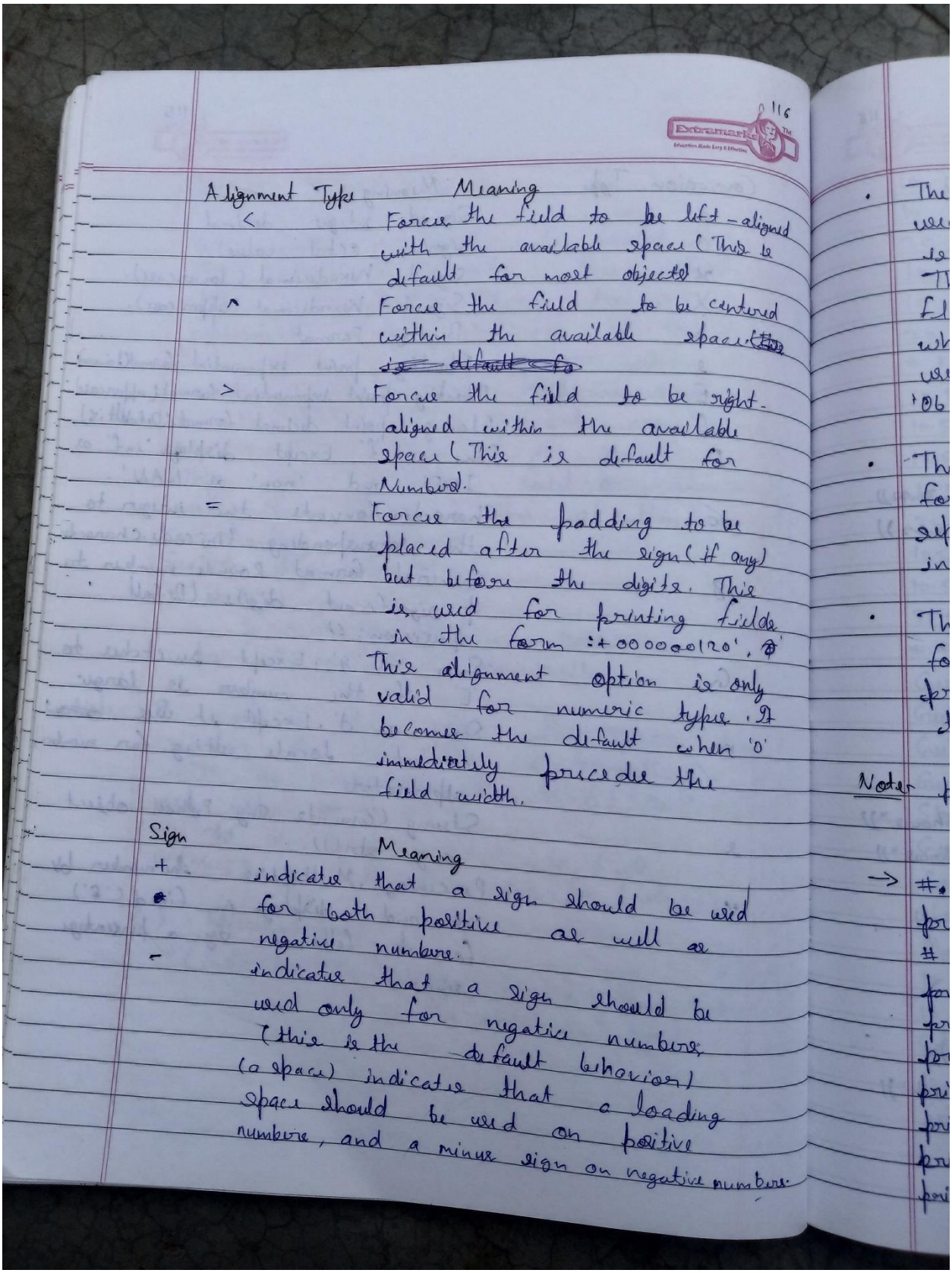
Note- In most of the case the syntax is similar to to old %-formatting, with the addition of the {} and with used instead of %

Ex-i) '%d' can be translated to '{:d}'
ii) '%05.3f' can be translated to '{:05.3f}'.

{} - Replacement field.
↳ {index/key:[fill][align][sign][#][0][width][,][.precision][type]}
Format specification
Thousand separator

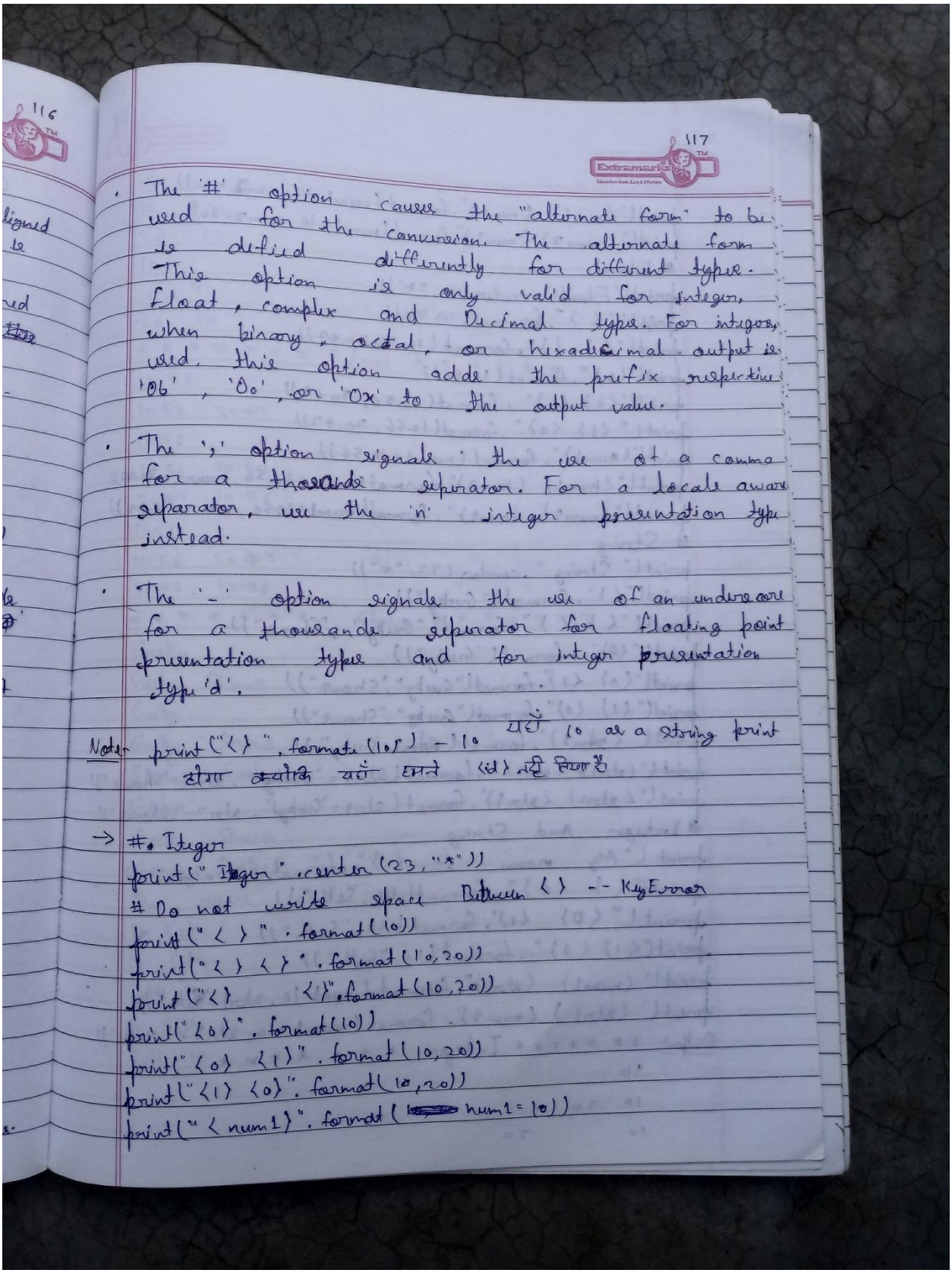


Conversion Type	Meaning
d	Signed integer decimal
o	Signed octal value
x	Signed hexadecimal (lowercase).
X	Signed hexadecimal (uppercase).
b	Binary Format
e	Floating point exponential format(lowercase)
E	Floating point exponential format(uppercase)
f	Floating point decimal format. (Default: 6)
F	Same as 'f'. Except displays 'inf' as 'INF' and 'nan' as 'NAN'.
c	Character. Converts the integer to the corresponding Unicode character
g	General format. Rounds number to p significant digits. (Default precision: 6)
G	Same as 'g'. Except switches to 'E' if the number is large.
n	Same as 'd'. Except it uses current locale setting for number separator.
s	String (converts any Python object using str()).
%	Percentage. Multiplies the number by 100 and displays in fixed ('f') format, followed by a percentage sign.



Alignment Type	Meaning
<	Forces the field to be left-aligned with the available space (This is default for most objects)
^	Forces the field to be centered within the available space.
>	Forces the field to be right-aligned within the available space (This is default for Numbers).
=	Forces the padding to be placed after the sign (if any) but before the digits. This is used for printing fields in the form '+000000120'. This alignment option is only valid for numeric types. It becomes the default when '0' immediately precedes the field width.

Sign	Meaning
+	indicates that a sign should be used for both positive as well as negative numbers.
-	indicates that a sign should be used only for negative numbers (this is the default behavior)
(a space)	indicates that a leading space should be used on positive numbers, and a minus sign on negative numbers.



- The '#' option causes the "alternate form" to be used for the conversion. The alternate form is defined differently for different types. This option is only valid for integer, float, complex and Decimal types. For integers, when binary, octal, or hexadecimal output is used, this option adds the prefix respective '0b', '0o', or '0x' to the output value.

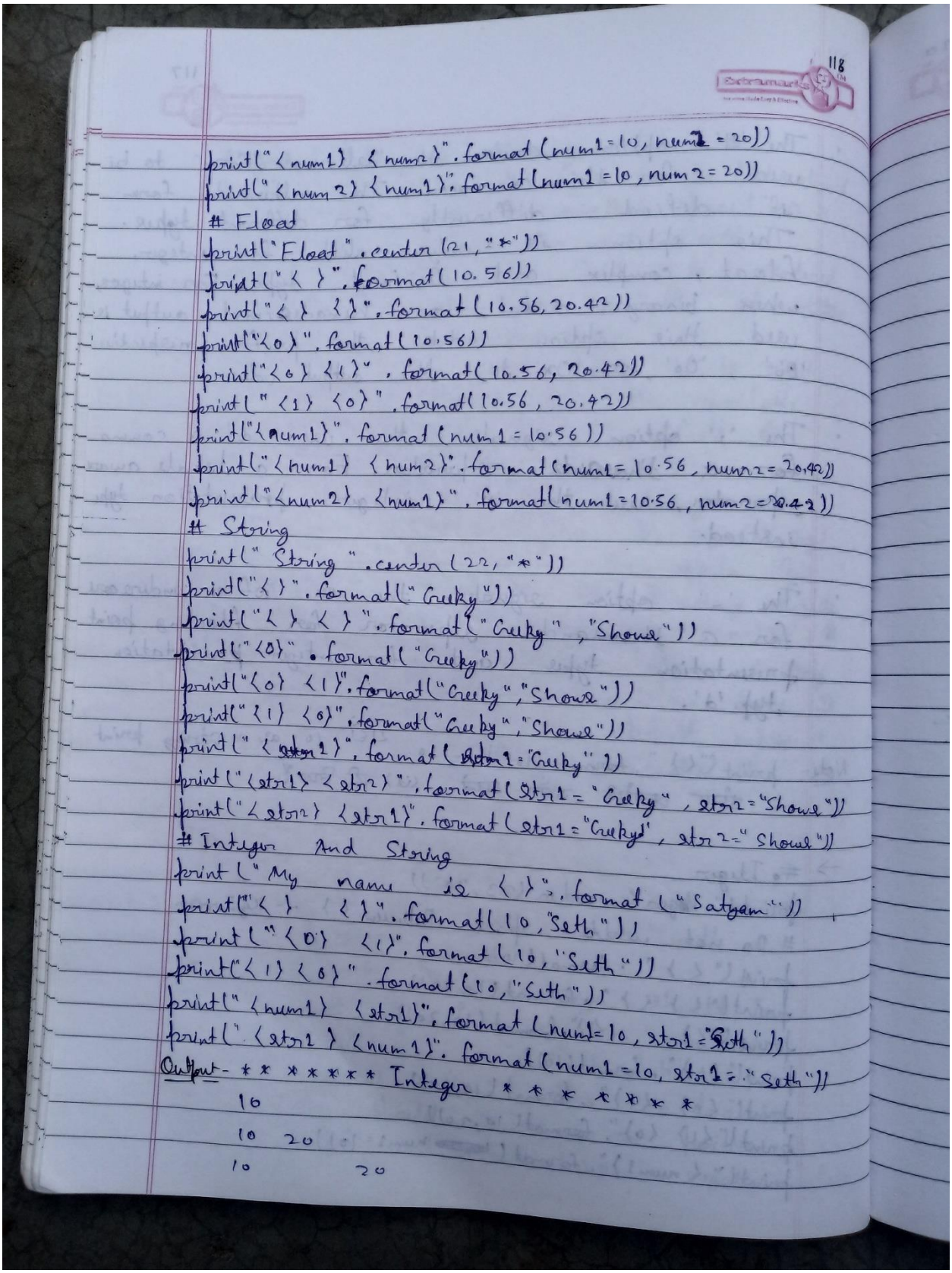
- The ',' option signals the use of a comma for a thousands separator. For a locale aware separator, use the 'n' integer presentation type instead.

- The '_' option signals the use of an underscore for a thousands separator for floating point presentation types and for integer presentation type 'd'.

Note:- print("<>".format(10)) - 10 यहाँ 10 as a string print होगा क्योंकि यहाँ कहीं <0> नहीं लिखा है

→ #. Integer

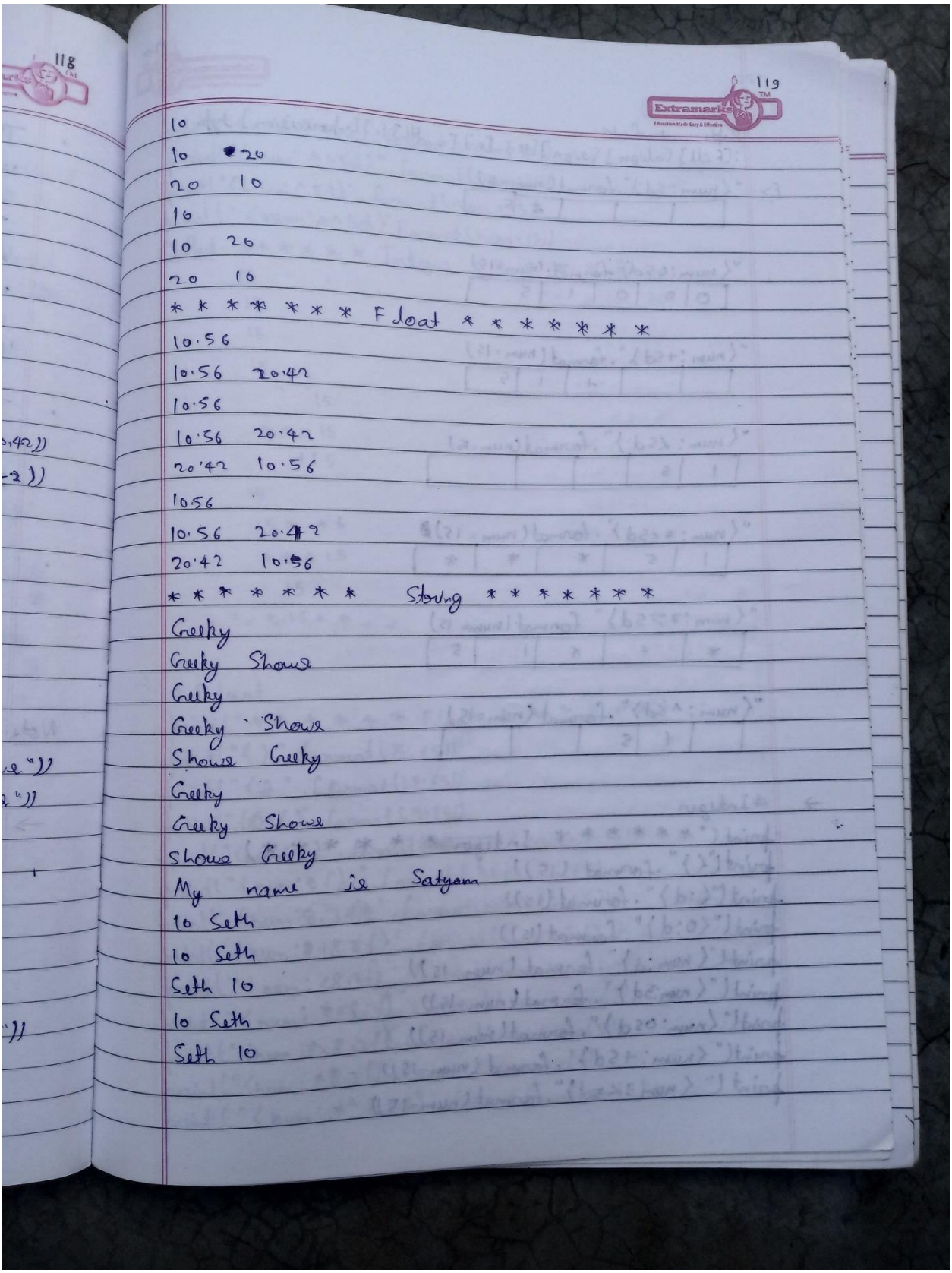
```
print("Integer".center(23,"*"))
# Do not write space Between { } -- KeyError
print("< >".format(10))
print("< > < >".format(10,20))
print("<>  <>".format(10,20))
print("<0>".format(10))
print("<0> <1>".format(10,20))
print("<1> <0>".format(10,20))
print("<num1>".format(num1=10))
```



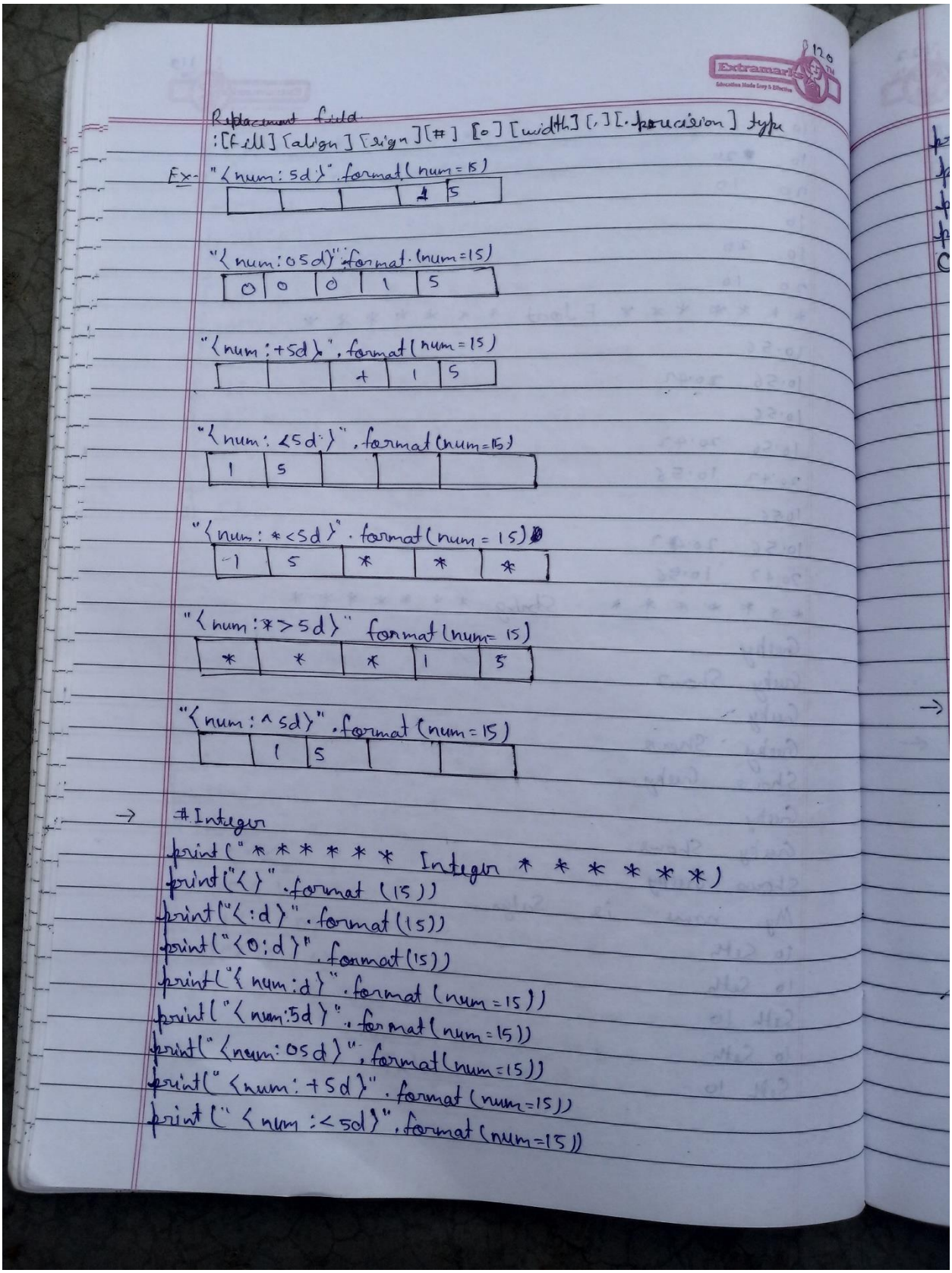
```
print("{num1} {num2}".format(num1=10, num2=20))
print("{num2} {num1}".format(num1=10, num2=20))
# Float
print("Float".center(21, "*"))
print("{}".format(10.56))
print("{} {}".format(10.56, 20.42))
print("{0}".format(10.56))
print("{0} {1}", format(10.56, 20.42))
print("{1} {0}".format(10.56, 20.42))
print("{num1}".format(num1=10.56))
print("{num1} {num2}".format(num1=10.56, num2=20.42))
print("{num2} {num1}".format(num1=10.56, num2=20.42))
# String
print("String".center(22, "*"))
print("{}".format("Geeky"))
print("{} {}".format("Geeky", "Shows"))
print("{0}".format("Geeky"))
print("{0} {1}".format("Geeky", "Shows"))
print("{1} {0}".format("Geeky", "Shows"))
print("{str1}".format(str1="Geeky"))
print("{str1} {str2}".format(str1="Geeky", str2="Shows"))
print("{str2} {str1}".format(str1="Geeky", str2="Shows"))
# Integer And String
print("My name is {}".format("Satyam"))
print("{} {}".format(10, "Seth"))
print("{0} {1}".format(10, "Seth"))
print("{1} {0}".format(10, "Seth"))
print("{num1} {str1}".format(num1=10, str1="Seth"))
print("{str1} {num1}".format(num1=10, str1="Seth"))
```

Output -

```
********** Integer **********
10
10 20
10 20
```



```
10
10  20
20  10
16
10  20
20  10
* * * * * * * Float * * * * * * *
10.56
10.56  20.42
10.56
10.56  20.42
20.42  10.56
10.56
10.56  20.42
20.42  10.56
* * * * * * * String * * * * * * *
Geeky
Geeky Shows
Geeky
Geeky Shows
Shows Geeky
Geeky
Geeky Shows
Shows Geeky
My name is Satyam
10 Seth
10 Seth
Seth 10
10 Seth
Seth 10
```



Replacement field
:[fill][align][sign][#][0][width][,][.precision]type

Ex- "{num:5d}".format(num=15)

			1	5

"{num:05d}".format(num=15)

0	0	0	1	5

"{num:+5d}".format(num=15)

		+	1	5

"{num:<5d}".format(num=15)

1	5			

"{num:*<5d}".format(num=15)

1	5	*	*	*

"{num:*>5d}".format(num=15)

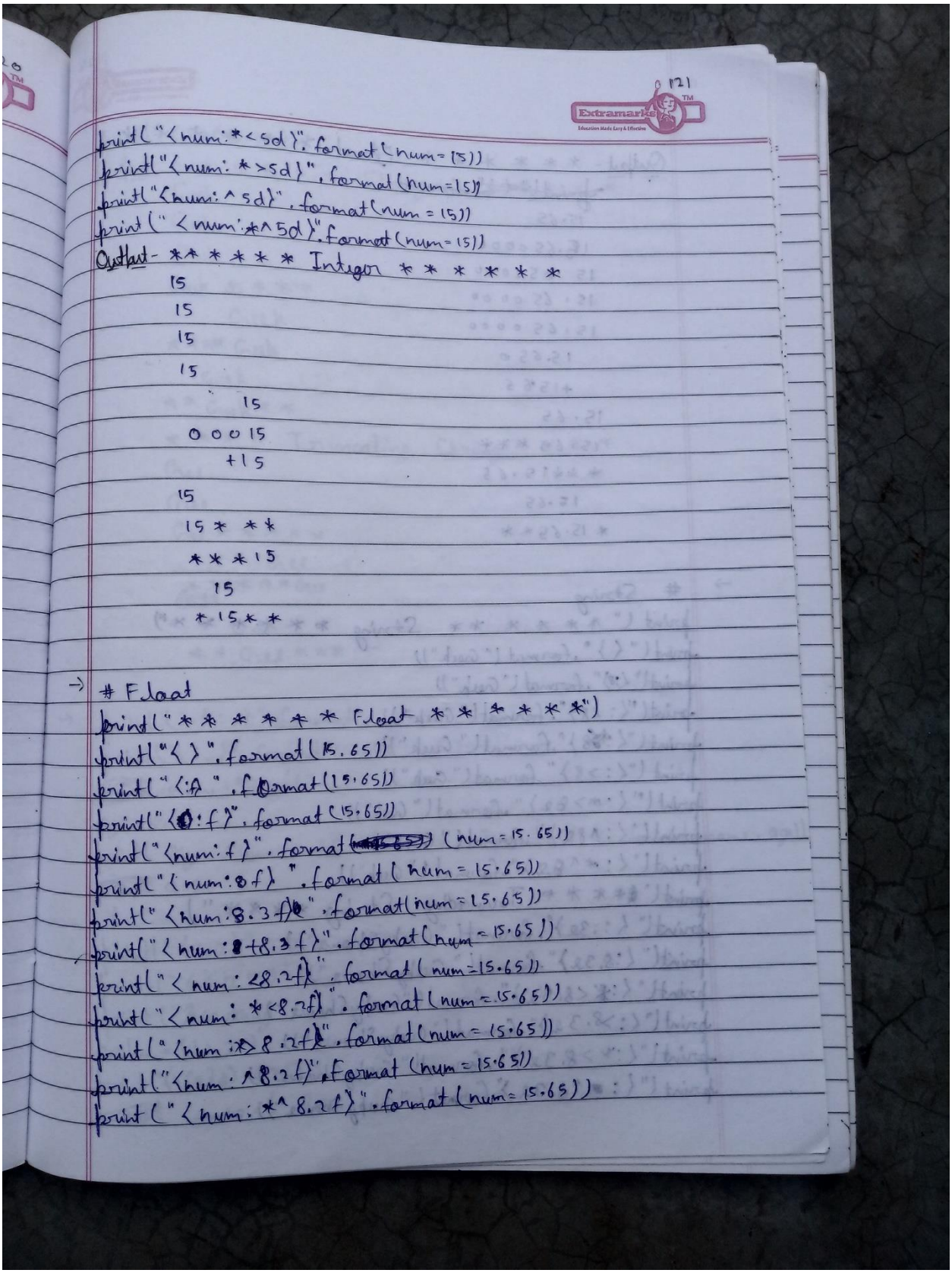
*	*	*	1	5

"{num:^5d}".format(num=15)

	1	5		

→ #Integer

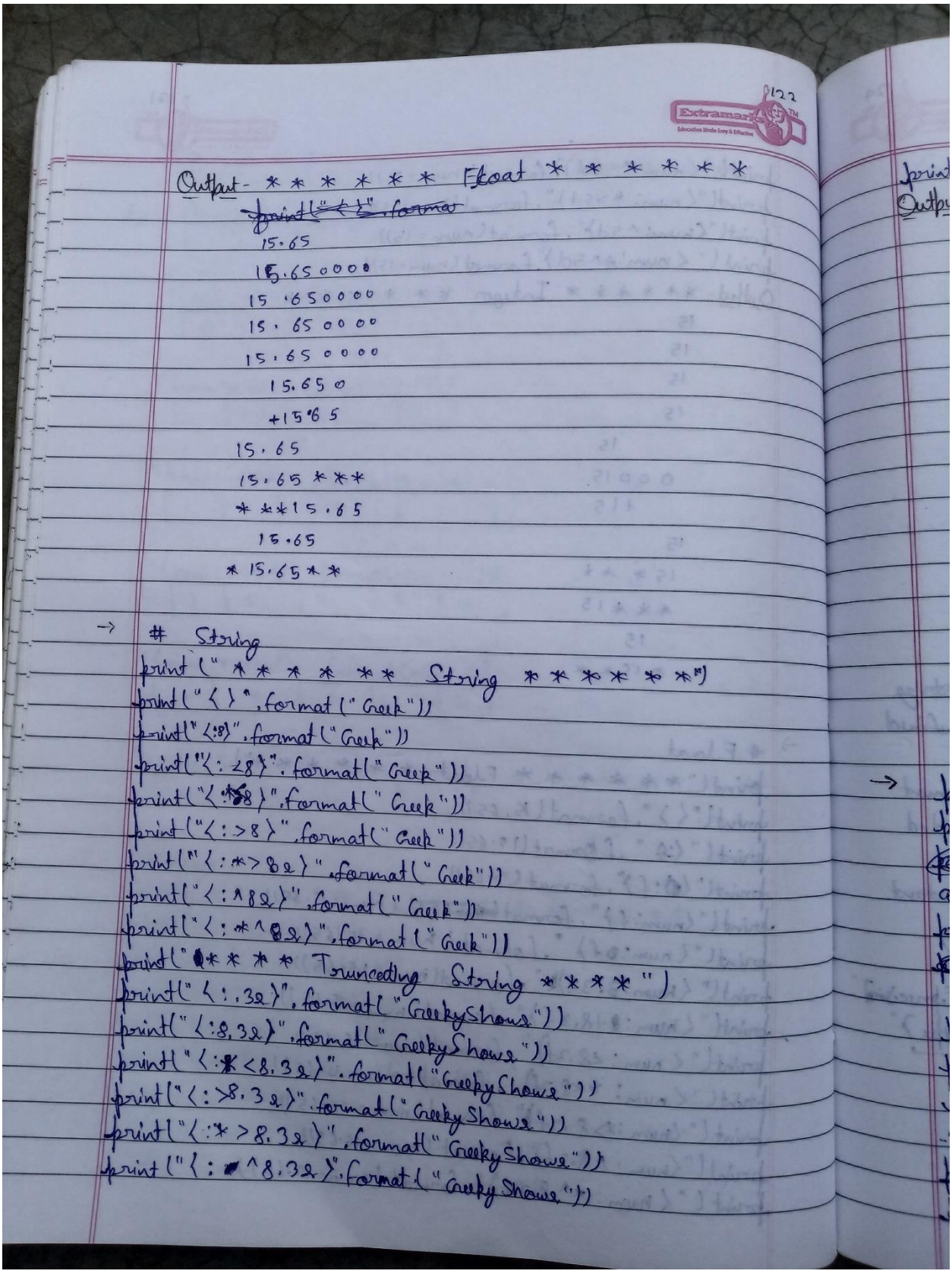
```
print("* * * * * * Integer * * * * * *")
print("{}".format(15))
print("{:d}".format(15))
print("{0:d}".format(15))
print("{num:d}".format(num=15))
print("{num:5d}".format(num=15))
print("{num:05d}".format(num=15))
print("{num:+5d}".format(num=15))
print("{num:<5d}".format(num=15))
```



```
print("{num:*<5d}".format(num=15))
print("{num:*>5d}".format(num=15))
print("{num:^5d}".format(num=15))
print("{num:*^5d}".format(num=15))
Output- ***** Integer *****
15
15
15
15
   15
000 15
  +15
15
15***
***15
 15
*15**
```

→ # Float

```
print("***** Float *****")
print("{}".format(15.65))
print("{:f}".format(15.65))
print("{0:f}".format(15.65))
print("{num:f}".format(num=15.65))
print("{num:.0f}".format(num=15.65))
print("{num:8.3f}".format(num=15.65))
print("{num:+8.3f}".format(num=15.65))
print("{num:<8.2f}".format(num=15.65))
print("{num:*<8.2f}".format(num=15.65))
print("{num:>8.2f}".format(num=15.65))
print("{num:^8.2f}".format(num=15.65))
print("{num:*^8.2f}".format(num=15.65))
```



Output - ****** Float ******

~~print("{:f}".format~~

15.65

15.650000

15.650000

15.650000

15.650000

15.650

+15.65

15.65

15.65***

***15.65

15.65

*15.65**

→ # String

print("***** String *****")

print("{}".format("Geek"))

print("{:8}".format("Geek"))

print("{:<8}".format("Geek"))

print("{:*<8}".format("Geek"))

print("{:>8}".format("Geek"))

print("{:*>8s}".format("Geek"))

print("{:^8s}".format("Geek"))

print("{:*^8s}".format("Geek"))

print("**** Truncating String ****")

print("{:.3s}".format("GeekyShows"))

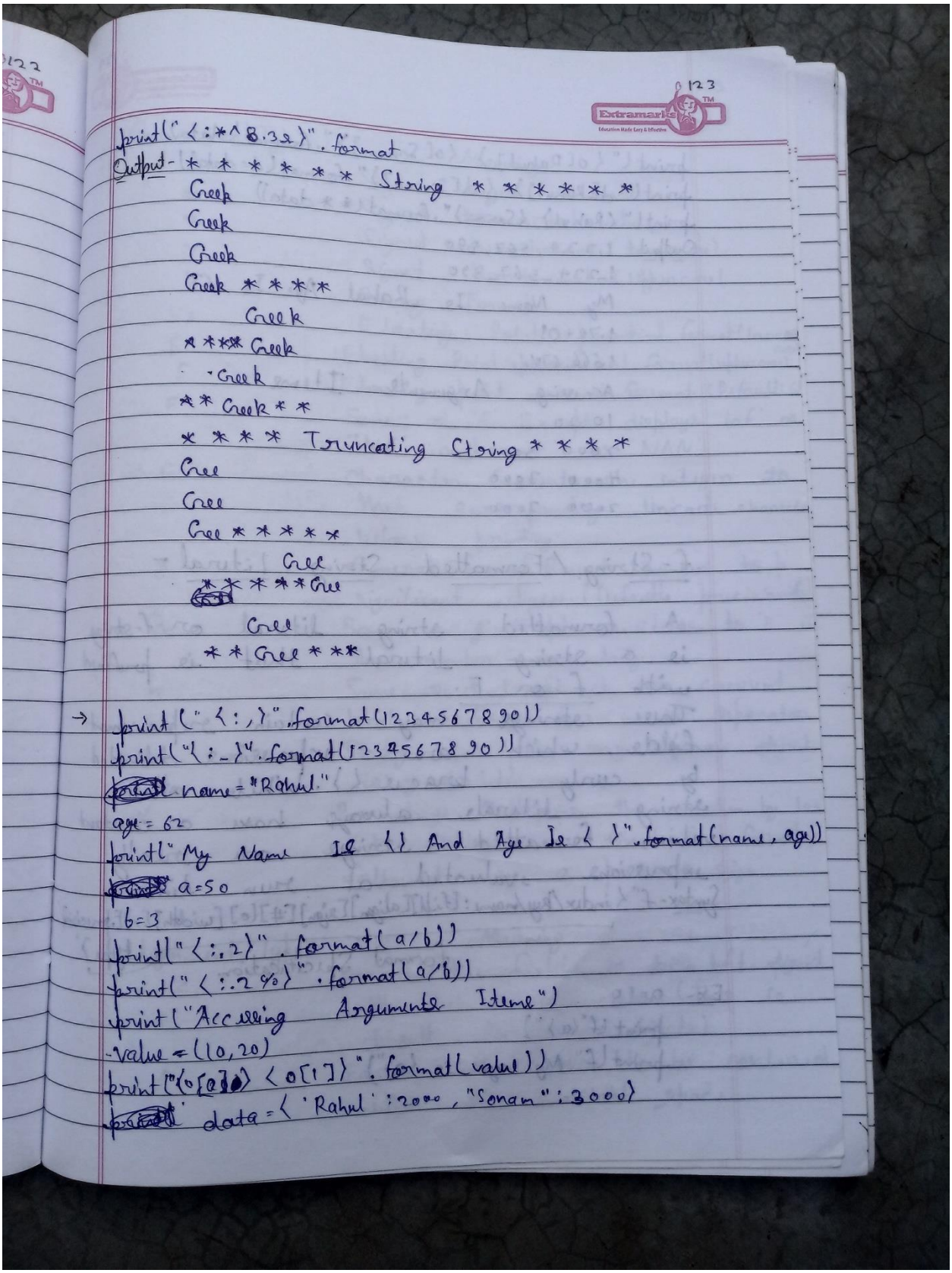
print("{:8.3s}".format("GeekyShows"))

print("{:*<8.3s}".format("GeekyShows"))

print("{:>8.3s}".format("GeekyShows"))

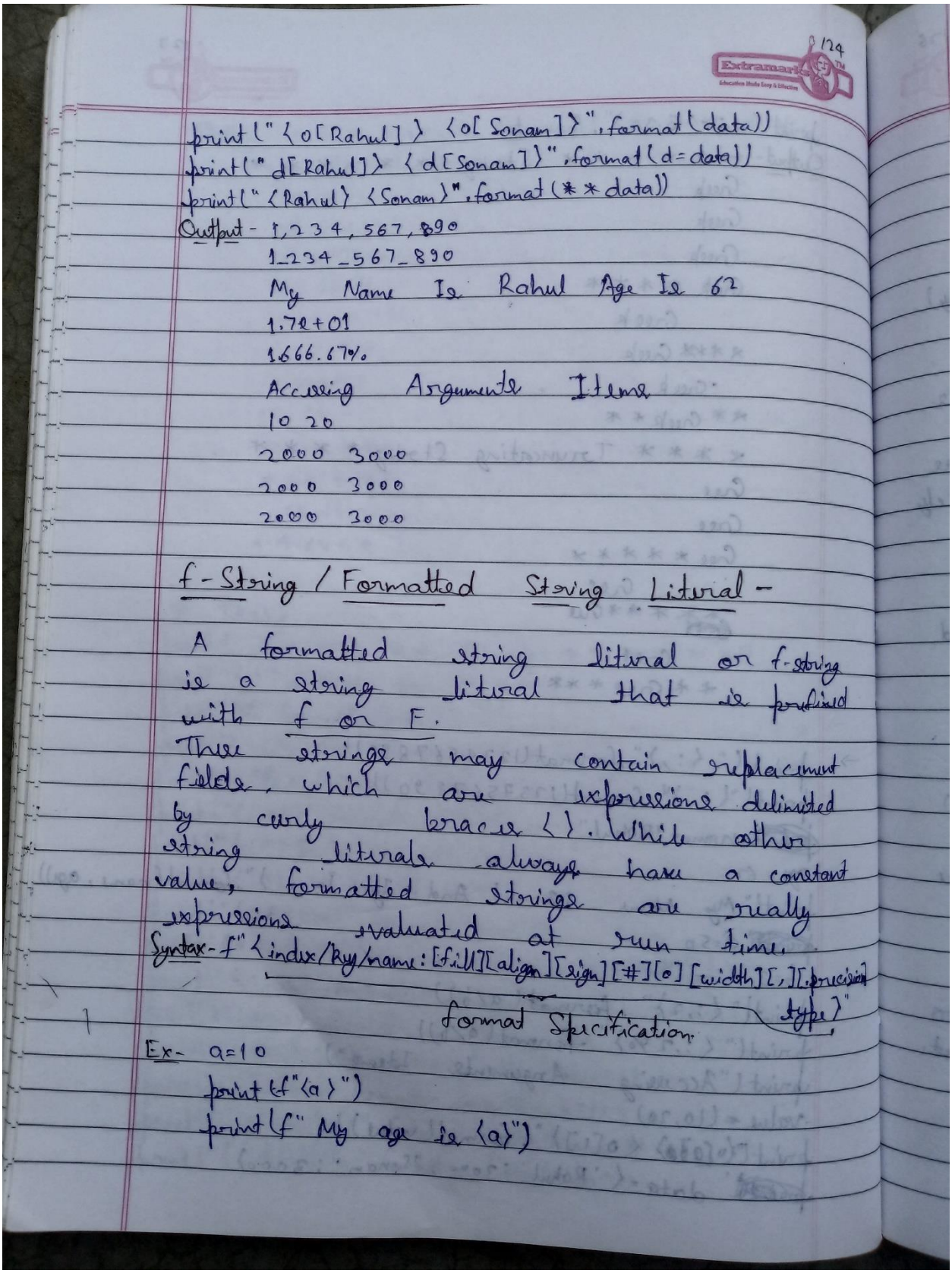
print("{:*>8.3s}".format("GeekyShows"))

print("{:*^8.3s}".format("GeekyShows"))



print("{:*^8.3s}".format
Output- * * * * * * String * * * * * *
Geek
Geek
Geek
Geek * * * *
Geek
* * * * Geek
Geek
* * Geek * *
* * * * Truncating String * * * *
Gee
Gee
Gee * * * * *
Gee
* * * * Gee
Gee
* * Gee * * *

→ print("{:,}".format(1234567890))
print("{:_}".format(1234567890))
name = "Rahul"
age = 62
print("My Name Is {} And Age Is {}".format(name, age))
a=50
b=3
print("{:.2}".format(a/b))
print("{:.2%}".format(a/b))
print("Accessing Arguments Items")
value = (10,20)
print("{0[0]} {0[1]}".format(value))
data = {'Rahul':2000, "Sonam":3000}

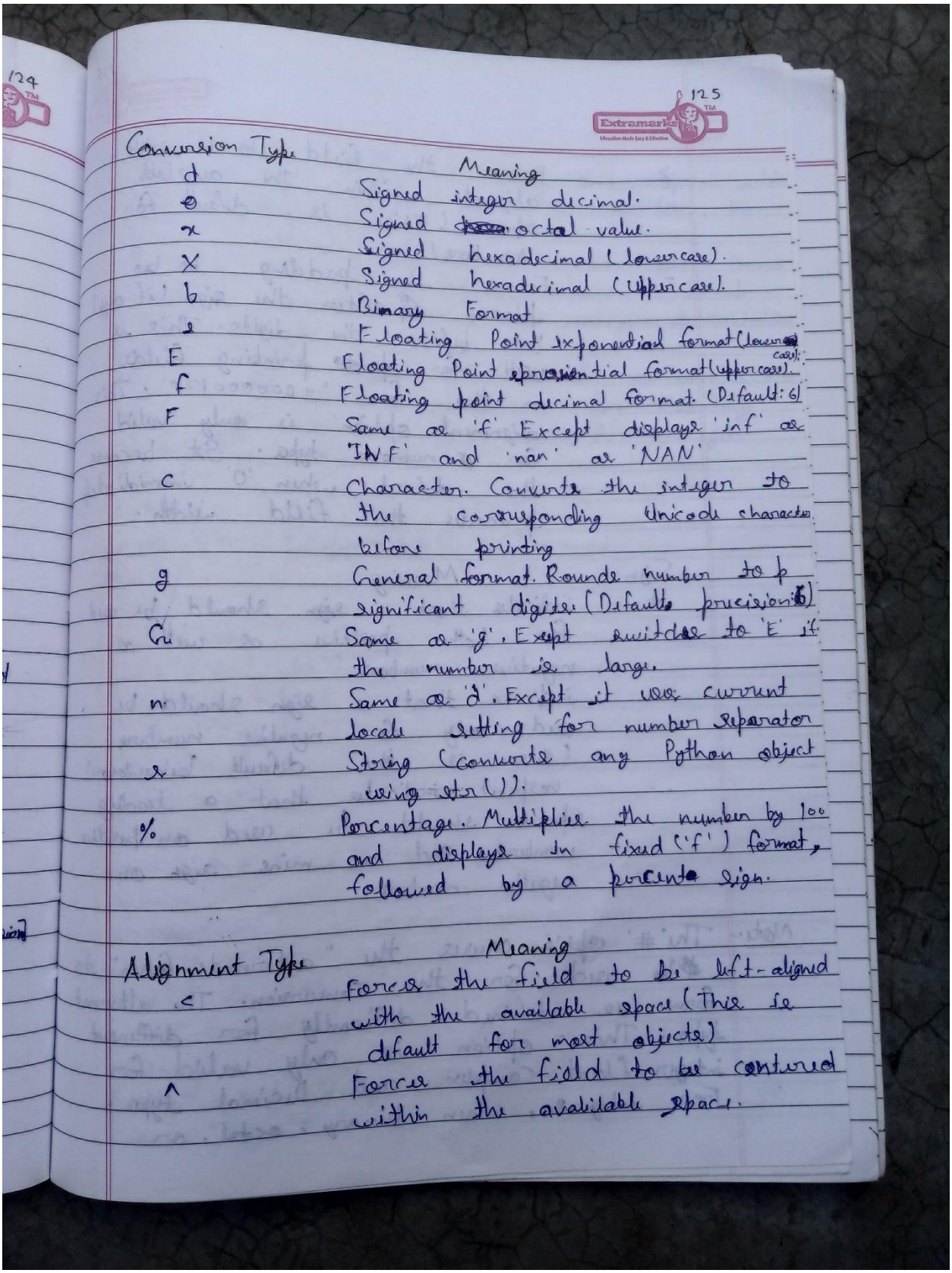


print("{o[Rahul]} {o[Sonam]}".format(data))
print("{d[Rahul]} {d[Sonam]}".format(d=data))
print("{Rahul} {Sonam}".format(**data))
Output - 1,234,567,890
1_234_567_890
My Name Is Rahul Age Is 62
1.7e+01
1666.67%
Accessing Arguments Items
10 20
2000 3000
2000 3000
2000 3000

f-String / Formatted String Literal -

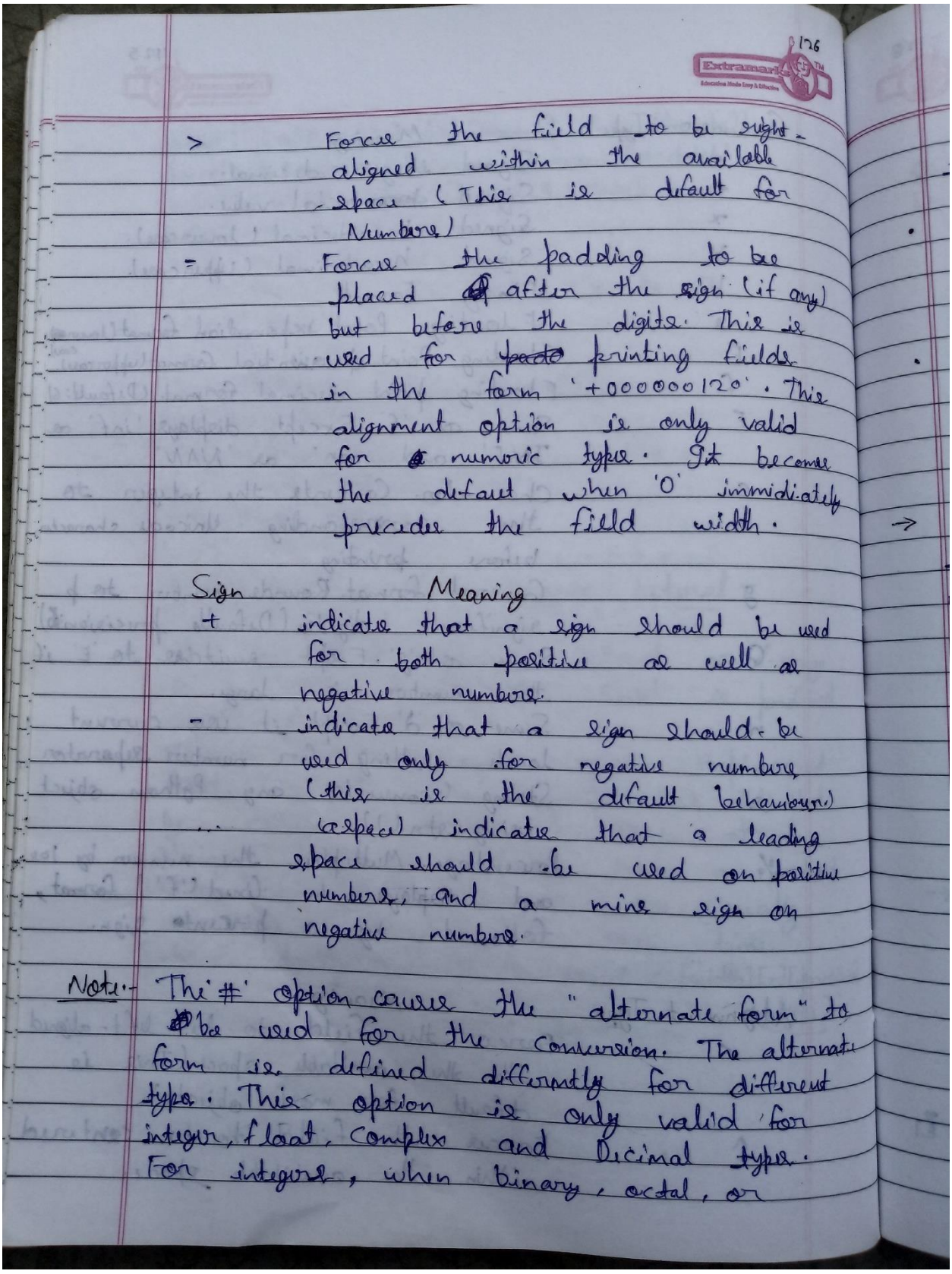
A formatted string literal or f-string is a string literal that is prefixed with f or F.
These strings may contain replacement fields, which are expressions delimited by curly braces {}. While other string literals always have a constant value, formatted strings are really expressions evaluated at run time.
Syntax - f"{index/key/name:[fill][align][sign][#][0][width][,][.precision][type]}"
format Specification

Ex - a=10
print(f"{a}")
print(f"My age is {a}")



Conversion Type	Meaning
d	Signed integer decimal.
o	Signed octal value.
x	Signed hexadecimal (lowercase).
X	Signed hexadecimal (Uppercase).
b	Binary Format
e	Floating Point exponential format (lowercase).
E	Floating Point exponential format (uppercase).
f	Floating point decimal format. (Default: 6)
F	Same as 'f' Except displays 'inf' as 'INF' and 'nan' as 'NAN'
c	Character. Converts the integer to the corresponding Unicode character before printing
g	General format. Rounds number to p significant digits. (Default precision: 6)
G	Same as 'g', Except switches to 'E' if the number is large.
n	Same as 'd'. Except it uses current locale setting for number separator
s	String (converts any Python object using str()).
%	Percentage. Multiplies the number by 100 and displays in fixed ('f') format, followed by a percent sign.

Alignment Type	Meaning
<	Forces the field to be left-aligned with the available space (This is default for most objects)
^	Forces the field to be centered within the available space.

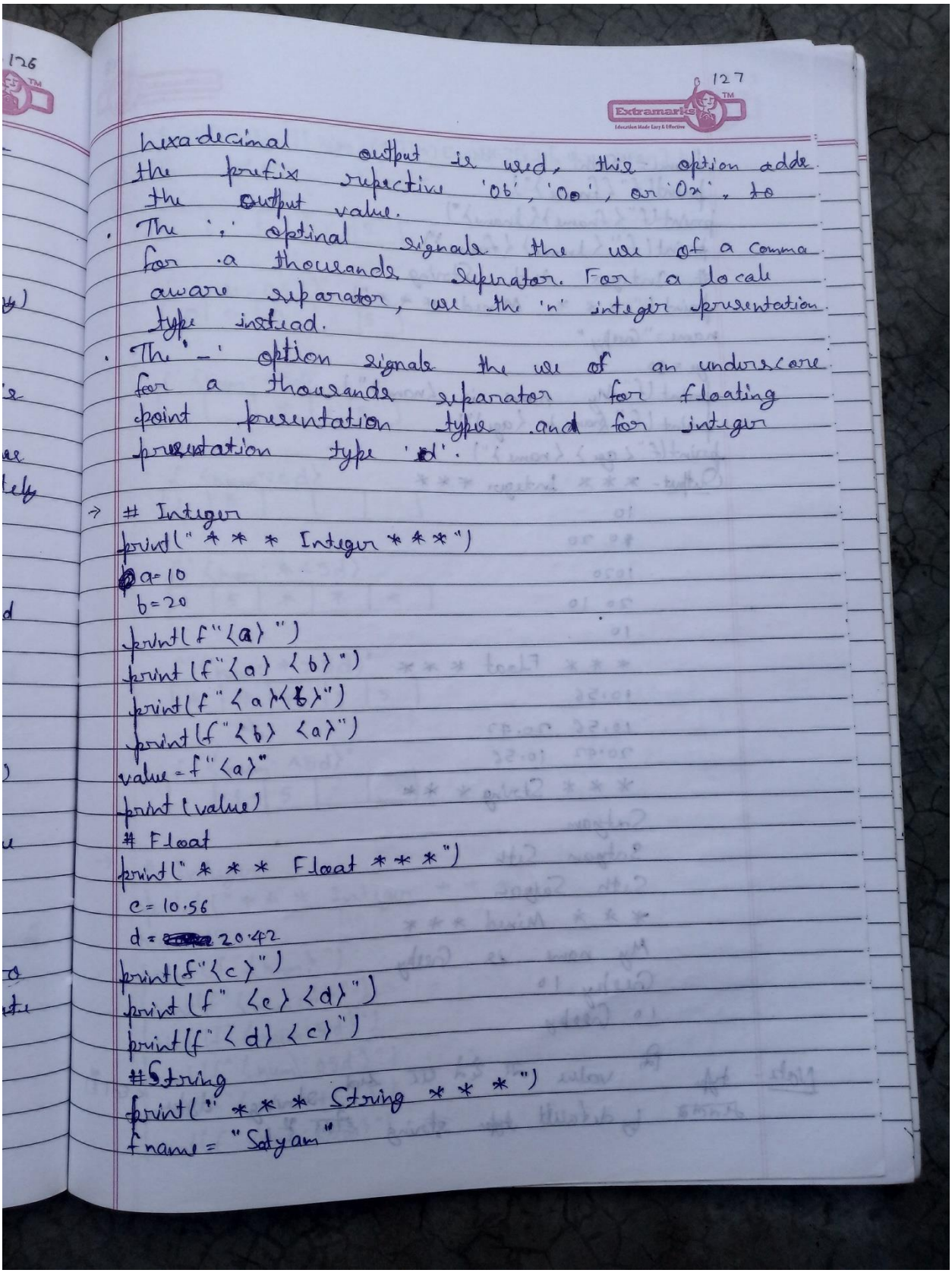


> Force the field to be right-aligned within the available space (This is default for Numbers)

= Force the padding to be placed after the sign (if any) but before the digits. This is used for printing fields in the form '+000000120'. This alignment option is only valid for numeric types. It becomes the default when '0' immidiately precedes the field width.

Sign	Meaning
+	indicate that a sign should be used for both positive as well as negative numbers.
-	indicate that a sign should be used only for negative numbers (this is the default behaviour)
' ' (space)	indicate that a leading space should be used on positive numbers, and a minus sign on negative numbers.

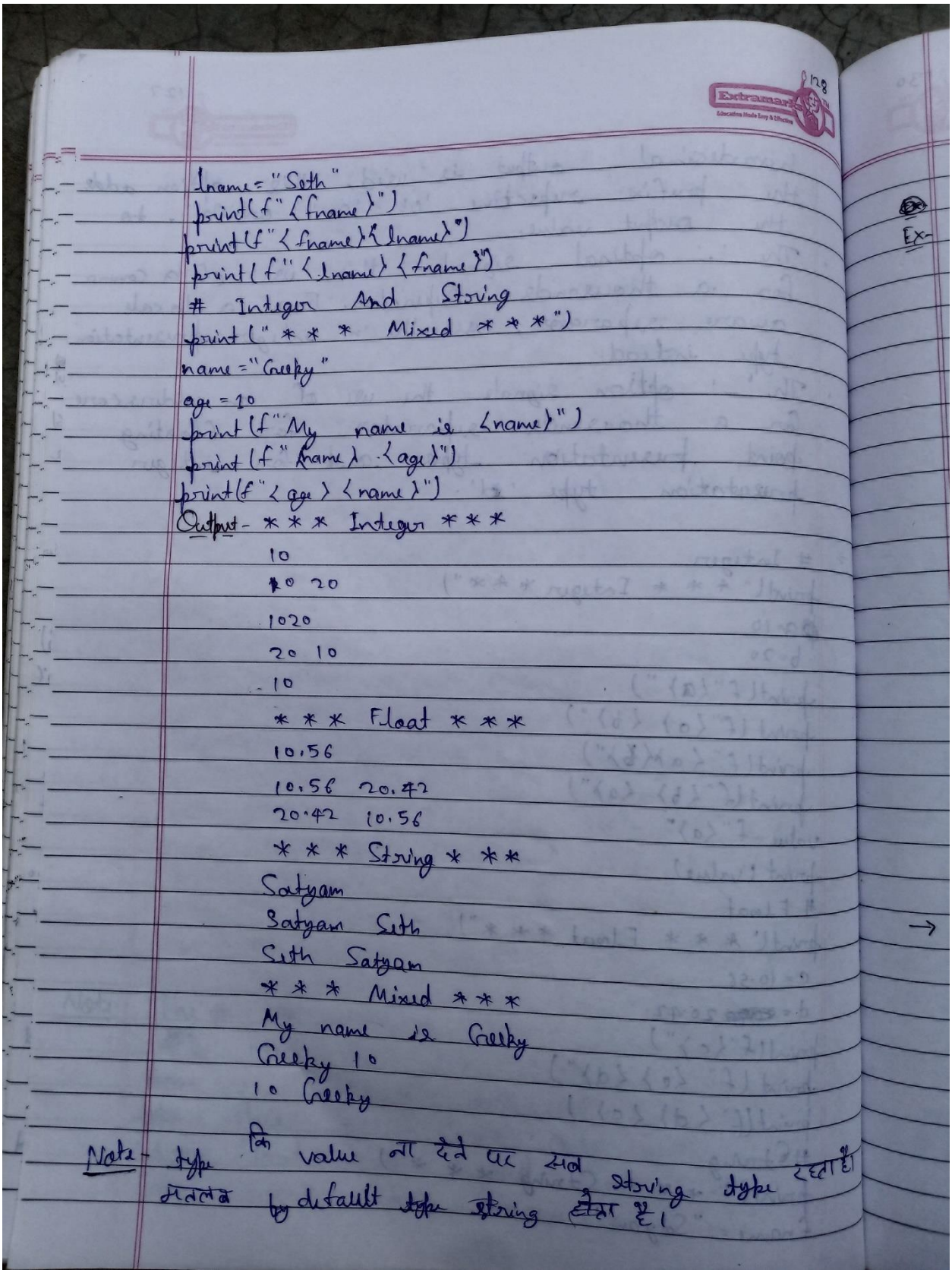
Note:- The '#' option causes the "alternate form" to be used for the conversion. The alternate form is defined differently for different types. This option is only valid for integer, float, complex and Decimal types. For integers, when binary, octal, or



hexadecimal output is used, this option adds the prefix respective '0b', '0o', or '0x', to the output value.

- The ',' optinal signals the use of a comma for a thousands separator. For a locale aware separator, use the 'n' integer presentation type instead.
- The '_' option signals the use of an underscore for a thousands separator for floating point presentation type and for integer presentation type 'd'.

```
# Integer
print("*** Integer ***")
a=10
b=20
print(f"{a}")
print(f"{a} {b}")
print(f"{a}{b}")
print(f"{b} {a}")
value=f"{a}"
print(value)
# Float
print("*** Float ***")
c=10.56
d=20.42
print(f"{c}")
print(f"{c} {d}")
print(f"{d} {c}")
#String
print("*** String ***")
fname="Satyam"
```

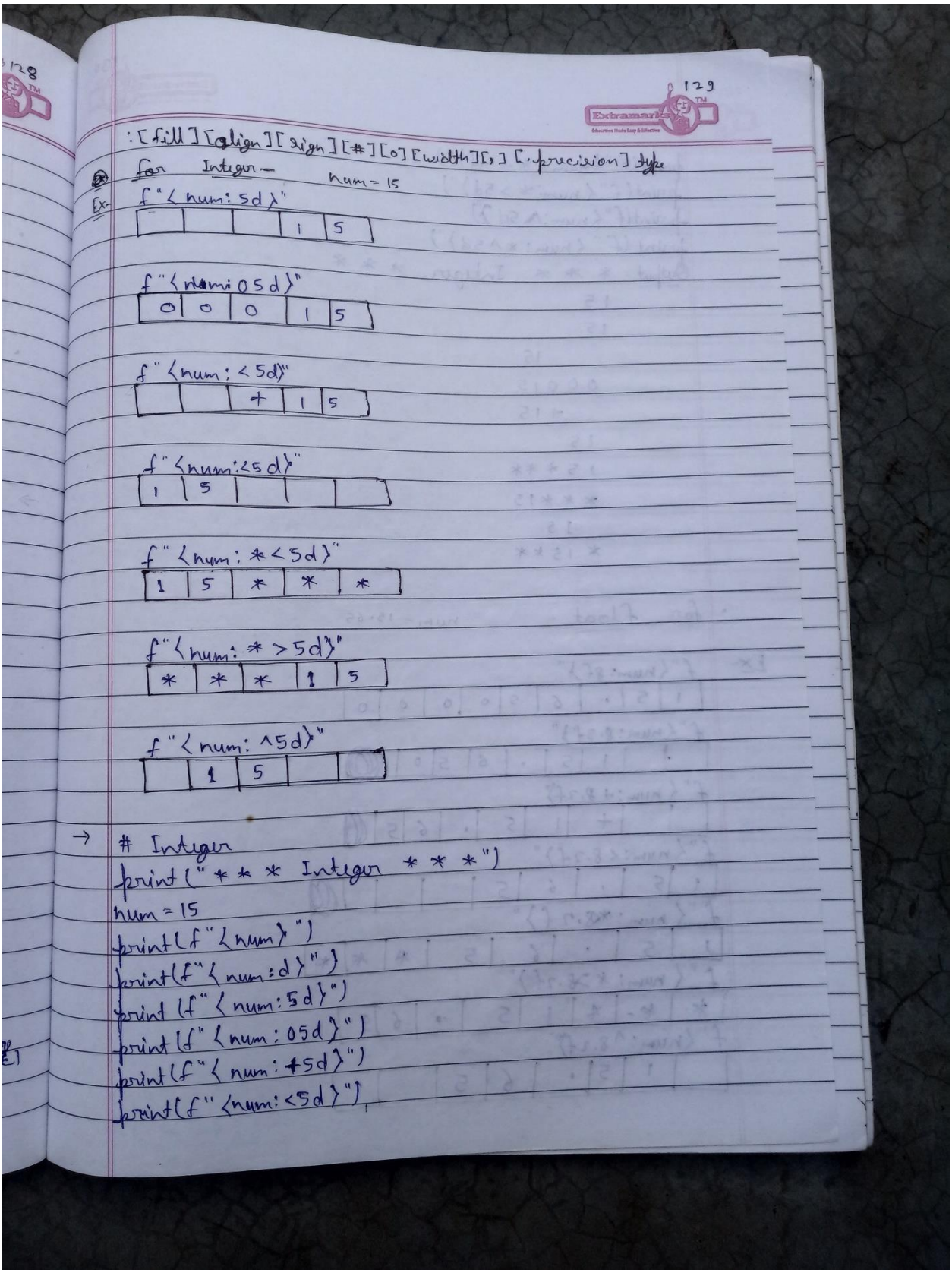


```
lname = "Seth"
print(f"{fname}")
print(f"{fname}{lname}")
print(f"{lname}{fname}")
# Integer And String
print("*** Mixed ***")
name = "Geeky"
age = 10
print(f"My name is {name}")
print(f"{name} {age}")
print(f"{age} {name}")
```

Output -

```
*** Integer ***
10
10 20
1020
20 10
10
*** Float ***
10.56
10.56 20.42
20.42 10.56
*** String ***
Satyam
Satyam Seth
Seth Satyam
*** Mixed ***
My name is Geeky
Geeky 10
10 Geeky
```

Note - type कि value ना देने पर यह string type रहता है।
यानि कि by default type string होता है।



:[fill][align][sign][#][0][width][,][.precision]type

for Integer — num = 15

Ex- f"{num:5d}"

			1	5

f"{num:05d}"

0	0	0	1	5

f"{num:<5d}"

			1	5

f"{num:<5d}"

1	5			

f"{num:*<5d}"

1	5	*	*	*

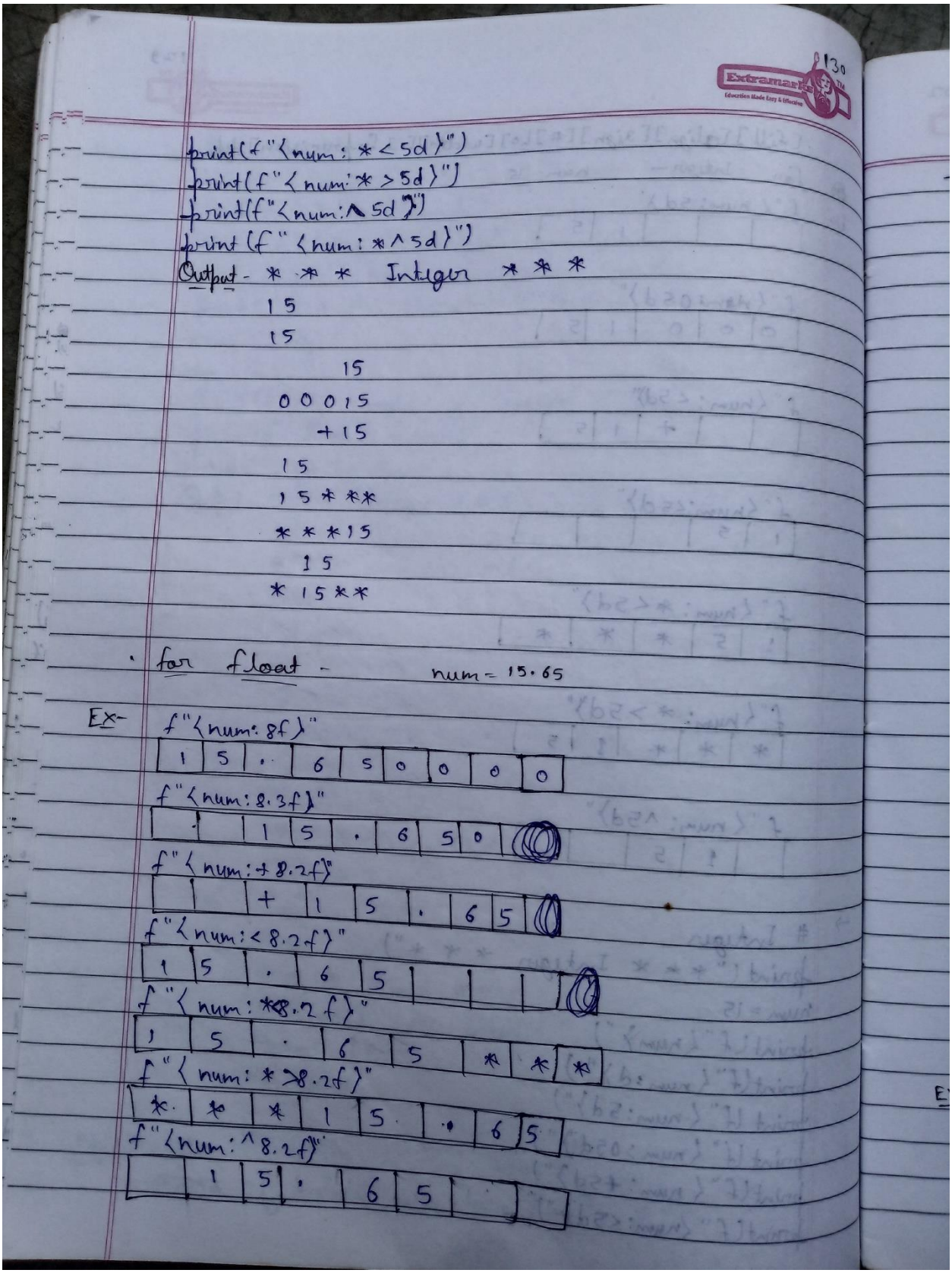
f"{num:*>5d}"

*	*	*	1	5

f"{num:^5d}"

	1	5		

```
# Integer
print("*** Integer ***")
num = 15
print(f"{num}")
print(f"{num:d}")
print(f"{num:5d}")
print(f"{num:05d}")
print(f"{num:+5d}")
print(f"{num:<5d}")
```



```
print(f"{num:*<5d}")
print(f"{num:*>5d}")
print(f"{num:^5d}")
print(f"{num:*^5d}")
```

Output - * * * Integer * * *

```
15
15
   15
00015
  +15
15
15***
***15
 15
*15**
```

- for float - num = 15.65

Ex- f"{num:8f}"

1	5	.	6	5	0	0	0	0

f"{num:8.3f}"

	1	5	.	6	5	0

f"{num:+8.2f}"

		+	1	5	.	6	5

f"{num:<8.2f}"

1	5	.	6	5			

f"{num:*8.2f}"

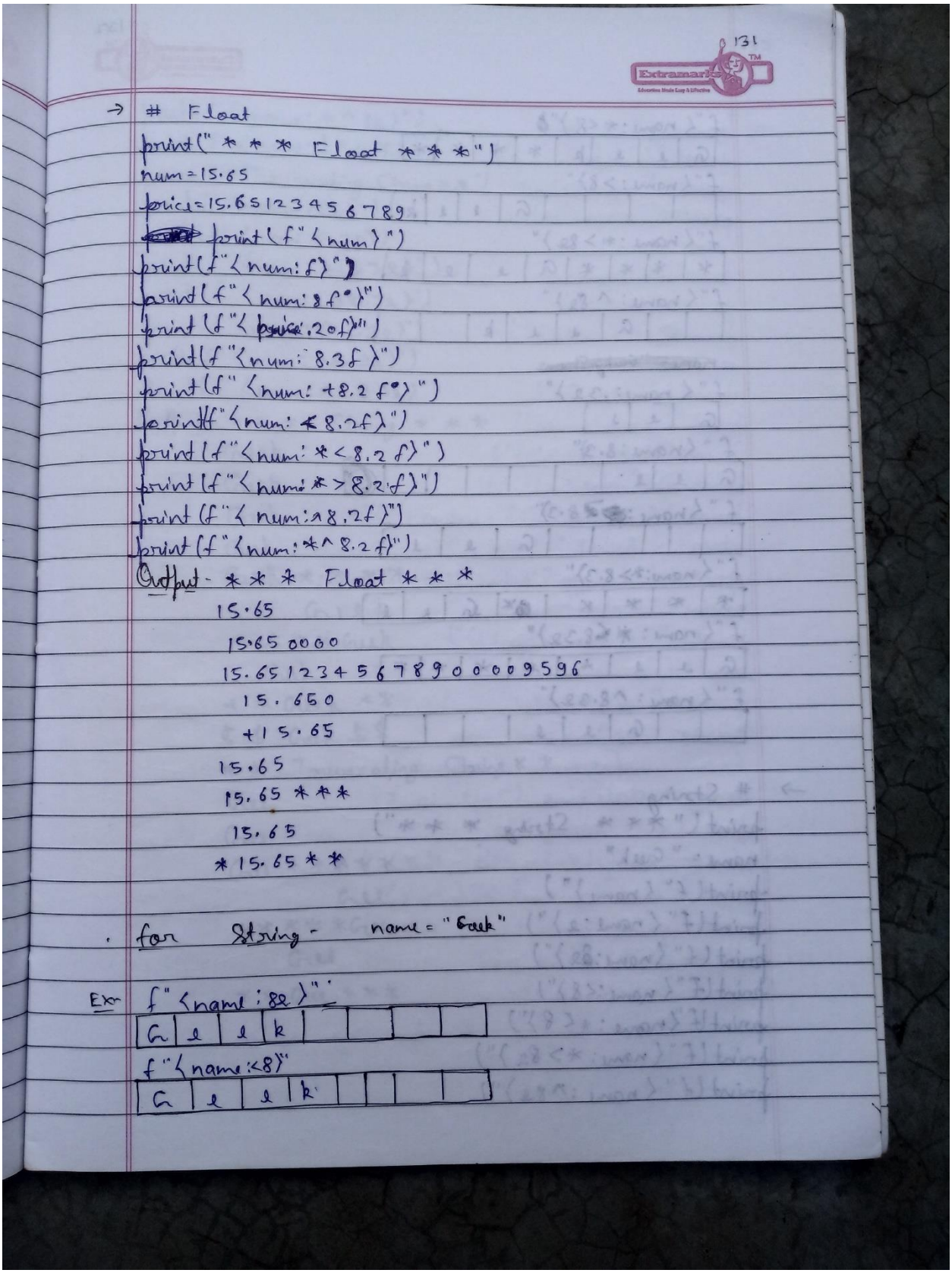
1	5	.	6	5	*	*	*

f"{num:*>8.2f}"

*	*	*	1	5	.	6	5

f"{num:^8.2f}"

	1	5	.	6	5		



```
# Float
print("*** Float ***")
num = 15.65
price = 15.651234567890
print(f"{num}")
print(f"{num:f}")
print(f"{num:8f}")
print(f"{price:20f}")
print(f"{num:8.3f}")
print(f"{num:+8.2f}")
print(f"{num:=8.2f}")
print(f"{num:*<8.2f}")
print(f"{num:*>8.2f}")
print(f"{num:^8.2f}")
print(f"{num:*^8.2f}")
```

Output -

```
*** Float ***
15.65
15.650000
15.651234567890000009596
 15.650
+15.65
15.65
15.65***
 15.65
*15.65**
```

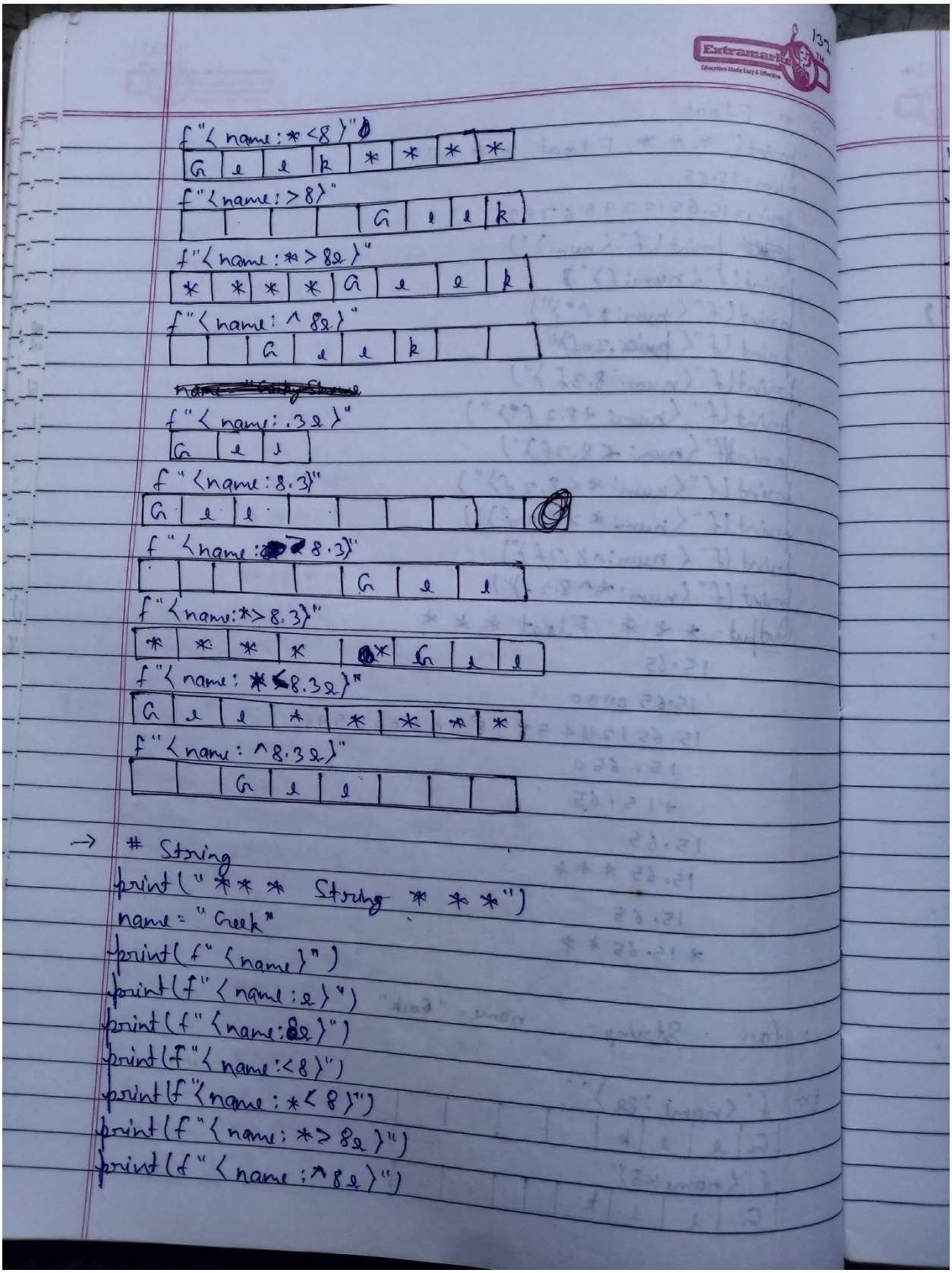
. for String - name = "Geek"

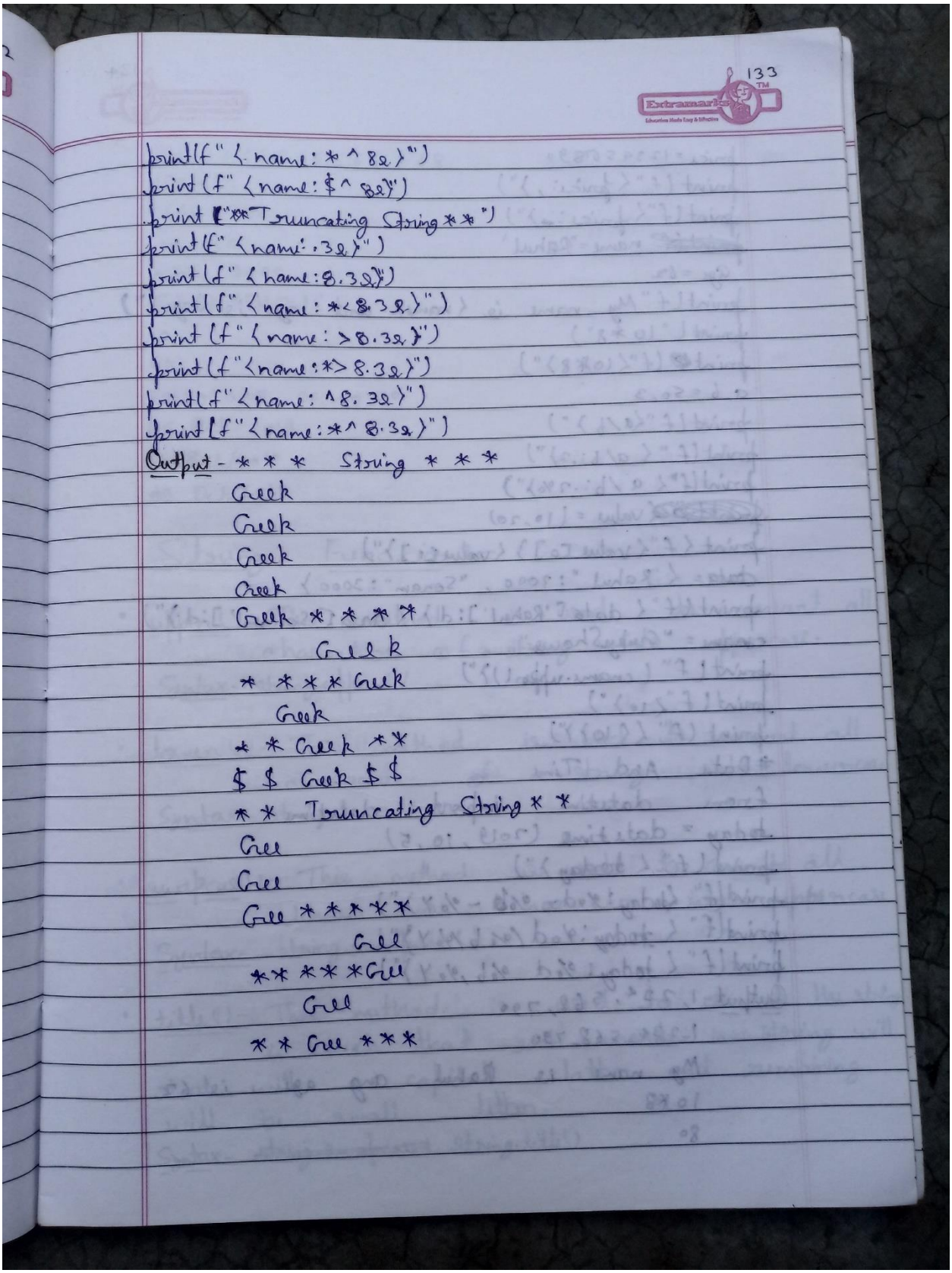
Ex f"{name:8s}"

G	e	e	k				

f"{name:<8}"

G	e	e	k				

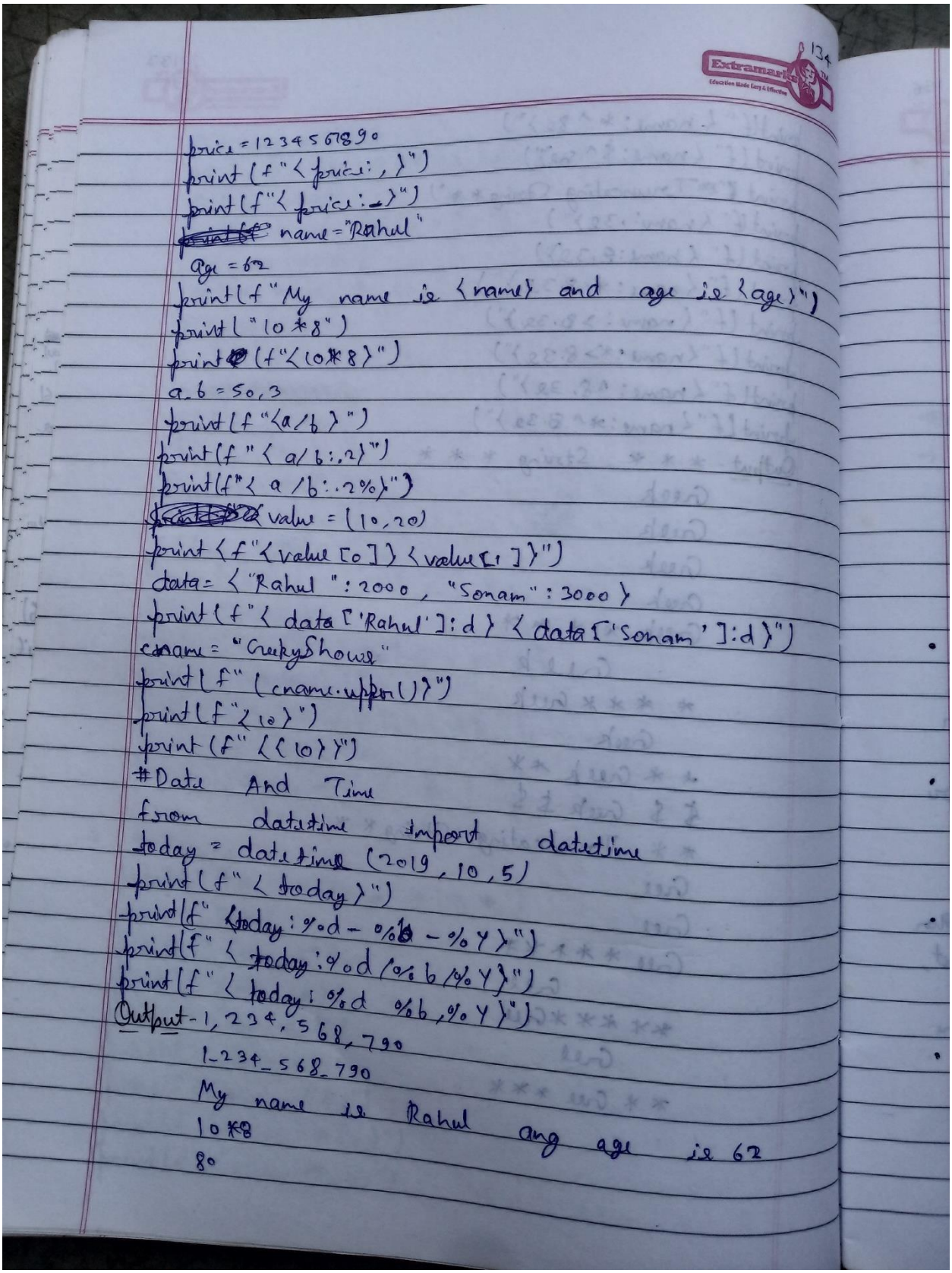

f"{name:*<8}"
Geek****
f"{name:>8}"
Geek
f"{name:*>8s}"
****Geek
f"{name:^8s}"
Geek
f"{name:.3s}"
Gee
f"{name:8.3}"
Gee
f"{name:8.3}"
Gee
f"{name:*>8.3}"
*****Gee
f"{name:*<8.3s}"
Gee*****
f"{name:^8.3s}"
Gee
# String
print("*** String ***")
name = "Geek"
print(f"{name}")
print(f"{name:s}")
print(f"{name:8s}")
print(f"{name:<8}")
print(f"{name:*<8}")
print(f"{name:*>8s}")
print(f"{name:^8s}")



```
print(f"{name:*^82}")
print(f"{name:$^82}")
print("**Truncating String**")
print(f"{name:.3g}")
print(f"{name:8.3g}")
print(f"{name:*<8.3g}")
print(f"{name:>8.3g}")
print(f"{name:*>8.3g}")
print(f"{name:^8.3g}")
print(f"{name:*^8.3g}")
```

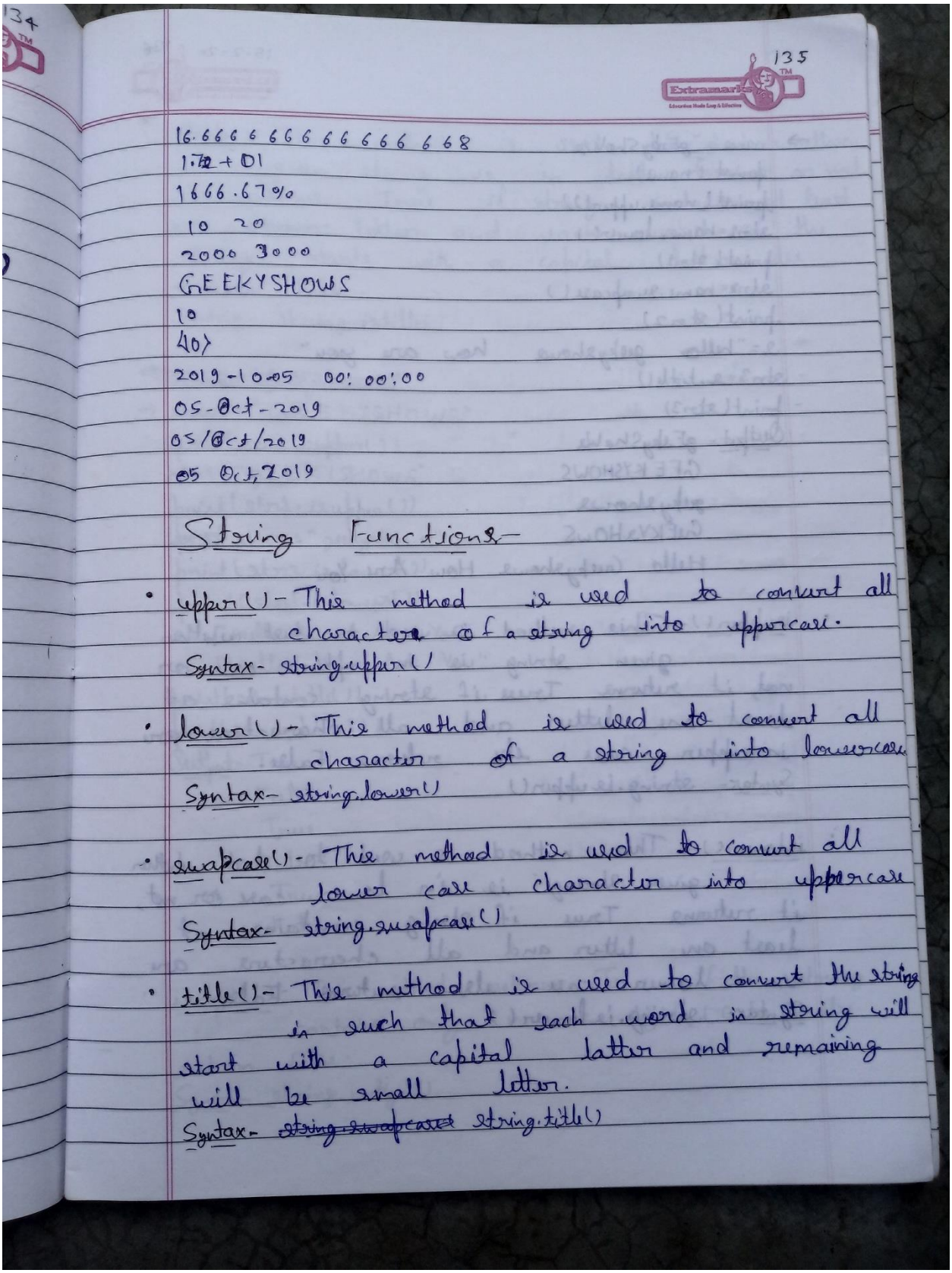
Output -

```
***  String  ***
Geek
Geek
Geek
Geek
Geek ****
    Geek
* ***Geek
  Geek
**Geek**
$$ Geek $$
** Truncating String **
Gee
Gee
Gee *****
    Gee
*** ***Gee
  Gee
** Gee ***
```



```
price = 1234567890
print(f"{price:,}")
print(f"{price:_}")
name = "Rahul"
age = 62
print(f"My name is {name} and age is {age}")
print("10*8")
print(f"{10*8}")
a,b = 50,3
print(f"{a/b}")
print(f"{a/b:.2}")
print(f"{a/b:.2%}")
value = (10,20)
print(f"{value[0]} {value[1]}")
data = {"Rahul":2000, "Sonam":3000}
print(f"{data['Rahul']:d} {data['Sonam']:d}")
cname = "GeekyShows"
print(f"{cname.upper()}")
print(f"{10}")
print(f"{{10}}")
#Date And Time
from datetime import datetime
today = datetime(2019,10,5)
print(f"{today}")
print(f"{today:%d-%b-%Y}")
print(f"{today:%d/%b/%Y}")
print(f"{today:%d %b,%Y}")
```

Output-
1,234,568,790
1_234_568_790
My name is Rahul ang age is 62
10*8
80



134

135

16.666666666666668
1.7e+01
1666.67%
10 20
2000 3000
GEEKYSHOWS
10
40
2019-10-05 00:00:00
05-Oct-2019
05/Oct/2019
05 Oct, 2019

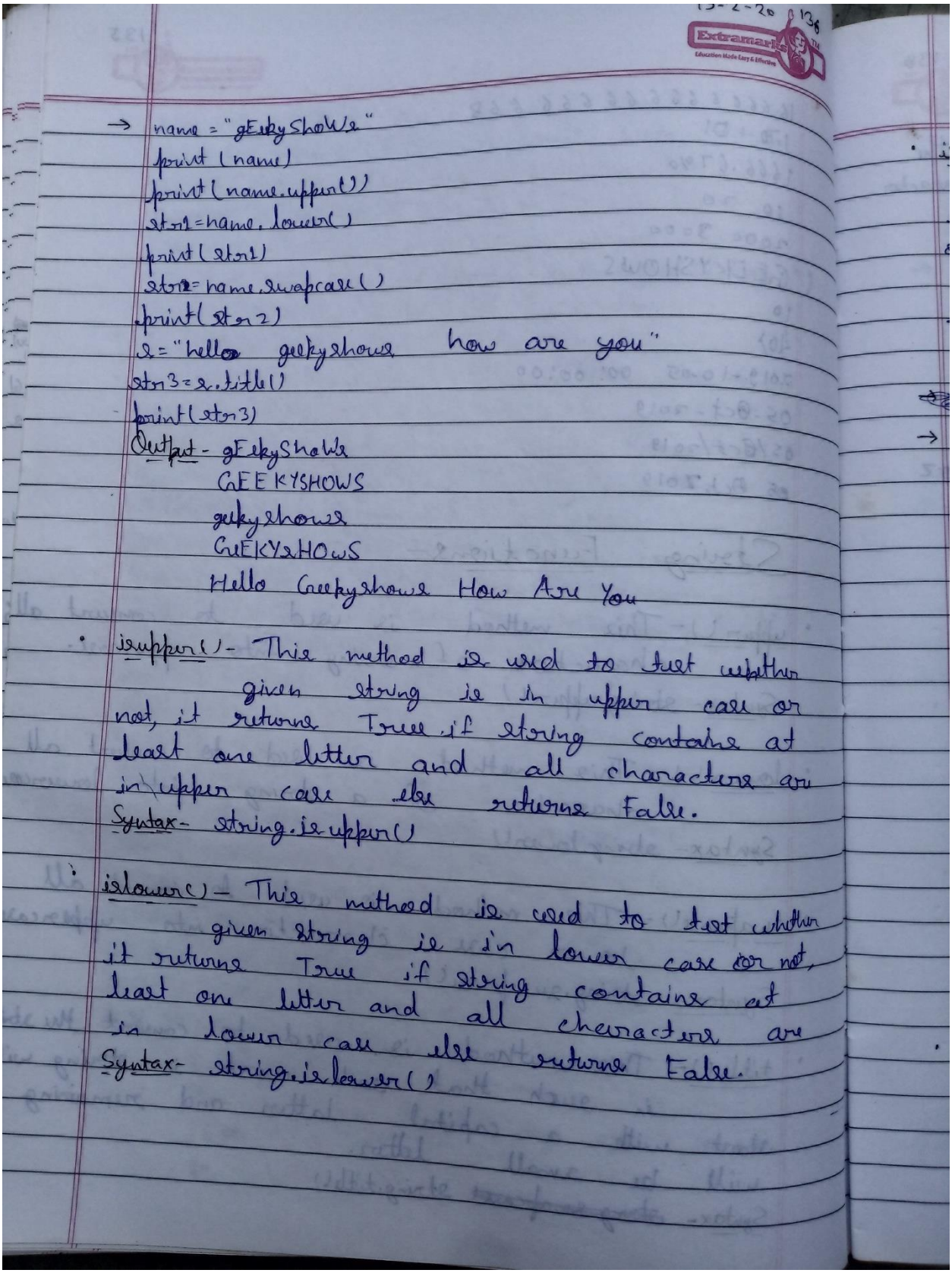
String Functions-

- **upper()** - This method is used to convert all characters of a string into uppercase.
  Syntax- string.upper()

- **lower()** - This method is used to convert all character of a string into lowercase.
  Syntax- string.lower()

- **swapcase()** - This method is used to convert all lower case character into uppercase.
  Syntax- string.swapcase()

- **title()** - This method is used to convert the string in such that each word in string will start with a capital letter and remaining will be small letter.
  Syntax- ~~string.swapcase~~ string.title()

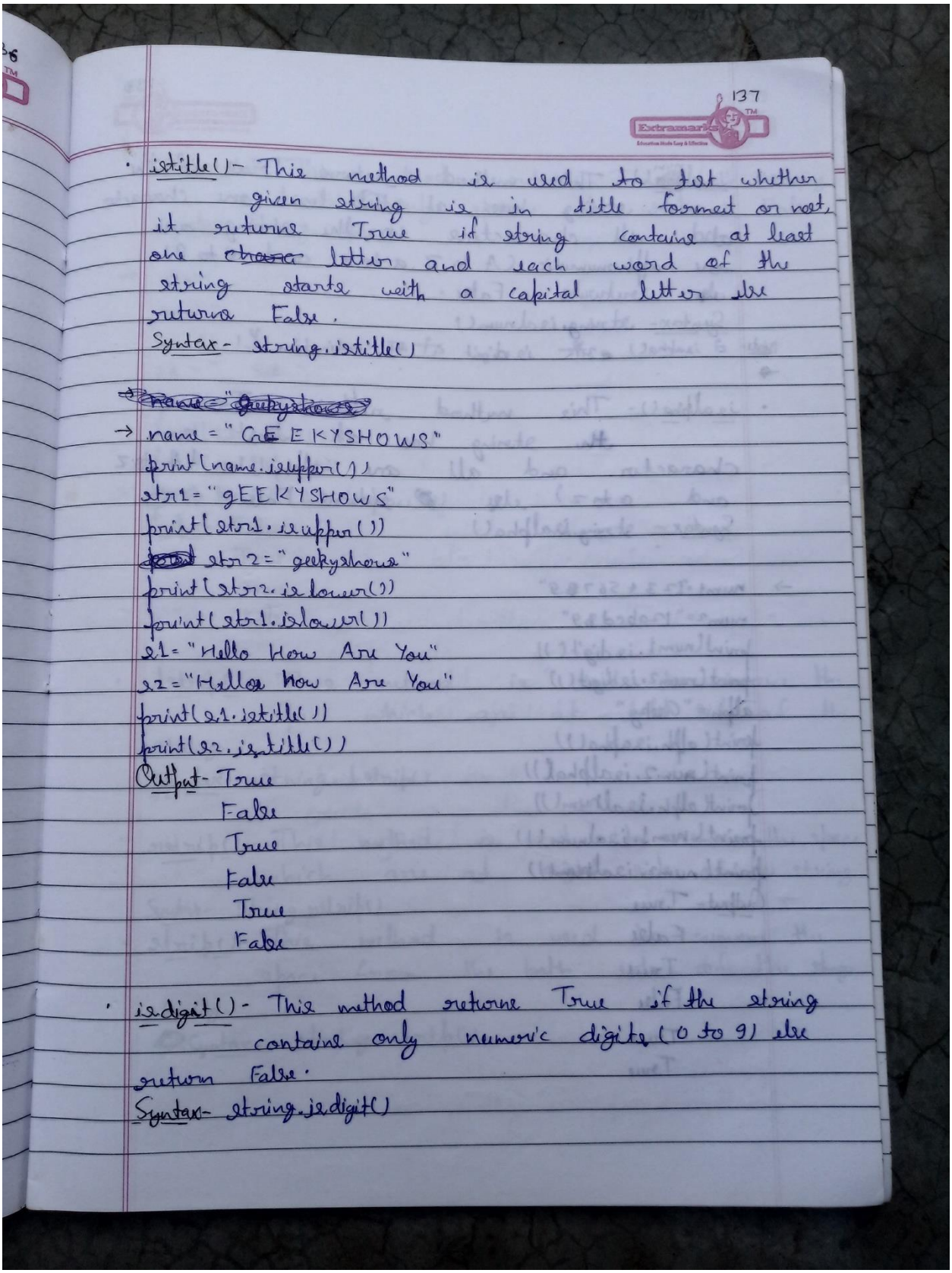


→
```
name = "gEekyShoWs"
print (name)
print (name.upper())
str1 = name.lower()
print (str1)
str2 = name.swapcase()
print (str2)
s = "hello geekyshows how are you"
str3 = s.title()
print (str3)
```

Output - gEekyShoWs
GEEKYSHOWS
geekyshows
GeEKYsHOwS
Hello Geekyshows How Are You

- isupper() - This method is used to test whether given string is in upper case or not, it returns True if string contains at least one letter and all characters are in upper case else returns False.
  Syntax - string.isupper()

- islower() - This method is used to test whether given string is in lower case or not, it returns True if string contains at least one letter and all characters are in lower case else returns False.
  Syntax - string.islower()



137

- **istitle()** - This method is used to tst whether given string is in title format or not, it returns True if string contains at least one ~~chara~~ letter and each word of the string starts with a capital letter else returns False.

Syntax - string.istitle()

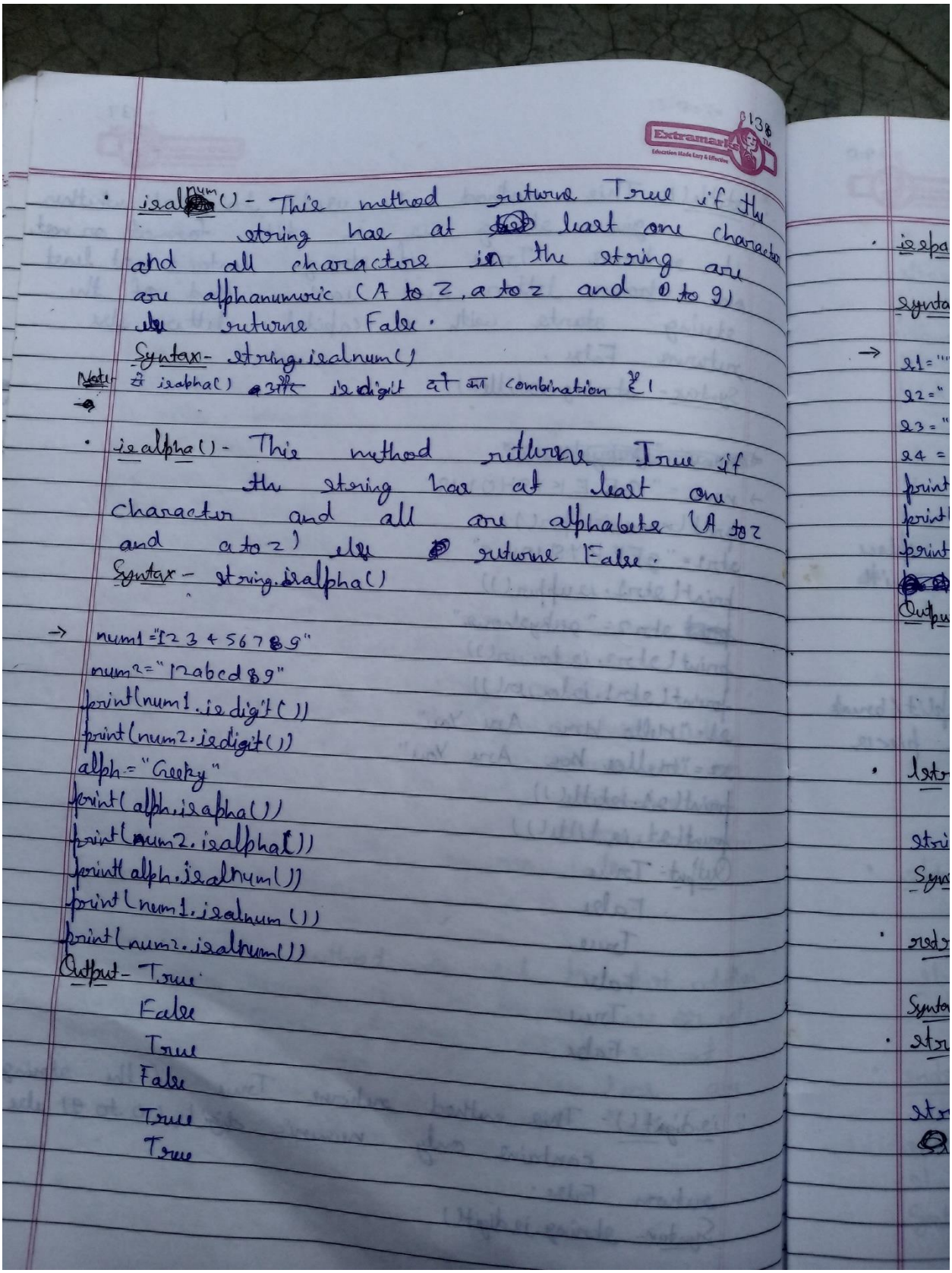
~~name = "geekyshows"~~

→ name = "GEEKYSHOWS"
print(name.isupper())
str1 = "gEEKYSHOWS"
print(str1.isupper())
~~print~~ str2 = "geekyshows"
print(str2.islower())
print(str1.islower())
s1 = "Hello How Are You"
s2 = "Hello how Are You"
print(s1.istitle())
print(s2.istitle())

Output - True
False
True
False
True
False

- **isdigit()** - This method returns True if the string contains only numeric digits (0 to 9) else return False.

Syntax - string.isdigit()



- isalnum() - This method returns True if the string has at least one character and all characters in the string are are alphanumeric (A to Z, a to z and 0 to 9) else returns False.

Syntax - string.isalnum()

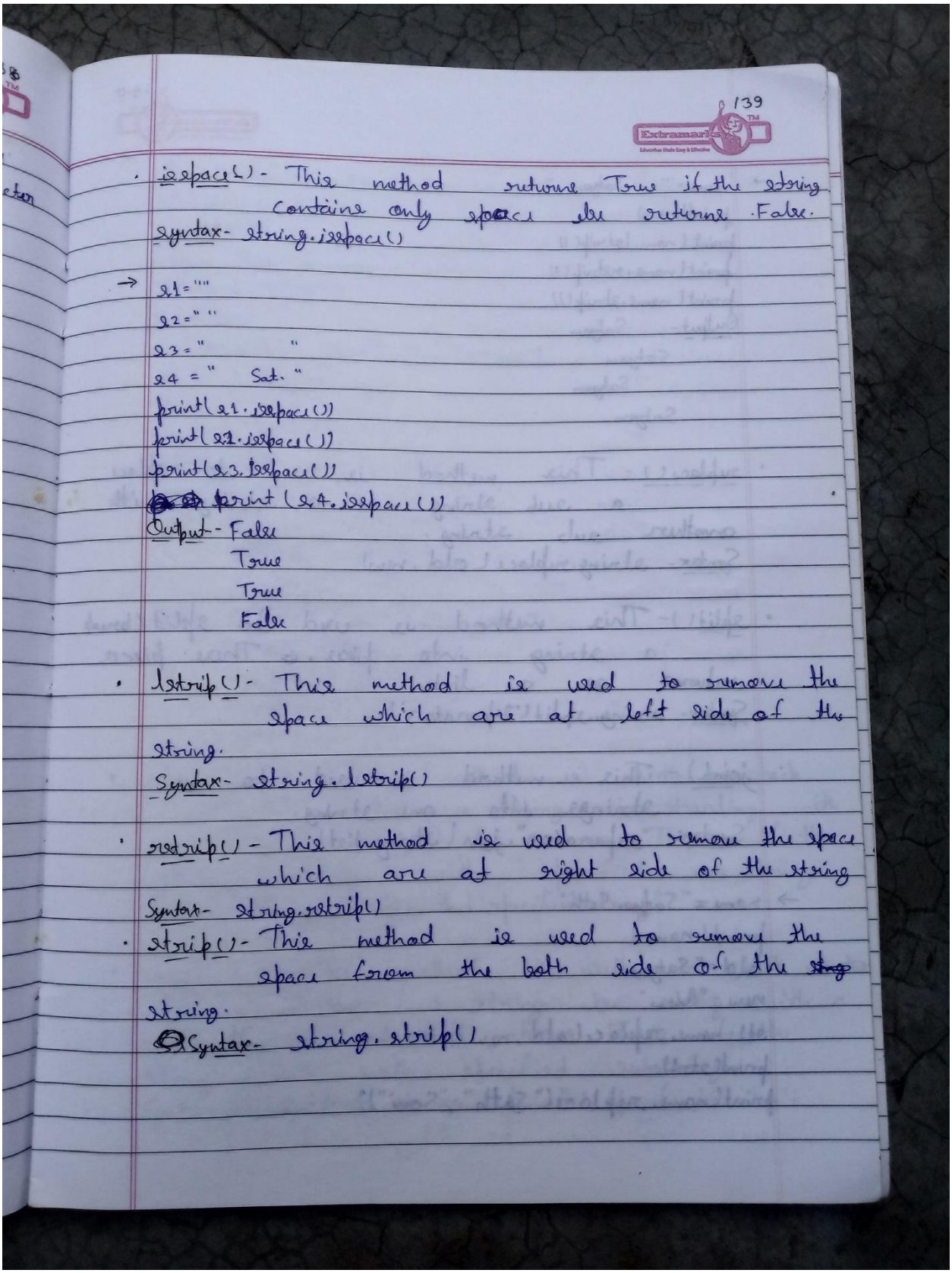
Note - It isalpha() and isdigit() ka combination hai.

- isalpha() - This method returns True if the string has at least one character and all are alphabets (A to Z and a to z) else returns False.

Syntax - string.isalpha()

```
num1 = "1 2 3 4 5 6 7 8 9"
num2 = "1 2 a b c d 8 9"
print(num1.isdigit())
print(num2.isdigit())
alph = "Geeky"
print(alph.isalpha())
print(num2.isalpha())
print(alph.isalnum())
print(num1.isalnum())
print(num2.isalnum())
```

Output - True
False
True
False
True
True



139

- isspace() - This method returns True if the string contains only spaces else returns False.

Syntax- string.isspace()

→
```
s1=""
s2=" "
s3="    "
s4 = "  Sat. "
print(s1.isspace())
print(s2.isspace())
print(s3.isspace())
print(s4.isspace())
```

Output- False
True
True
False

- lstrip() - This method is used to remove the space which are at left side of the string.

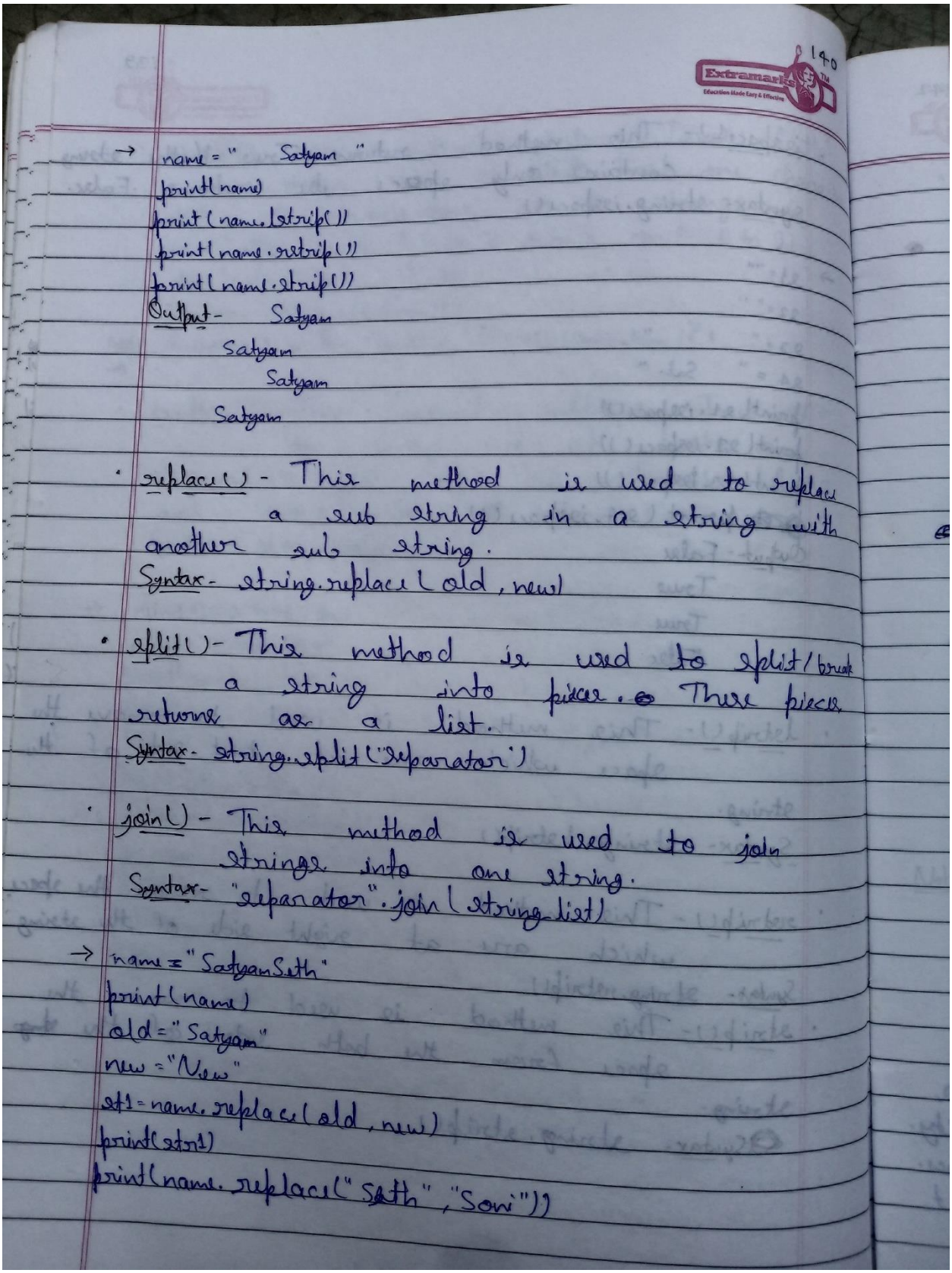
Syntax- string.lstrip()

- rstrip() - This method is used to remove the space which are at right side of the string

Syntax- string.rstrip()

- strip() - This method is used to remove the space from the both side of the string.

Syntax- string.strip()



```
name = "   Satyam "
print(name)
print(name.lstrip())
print(name.rstrip())
print(name.strip())
```

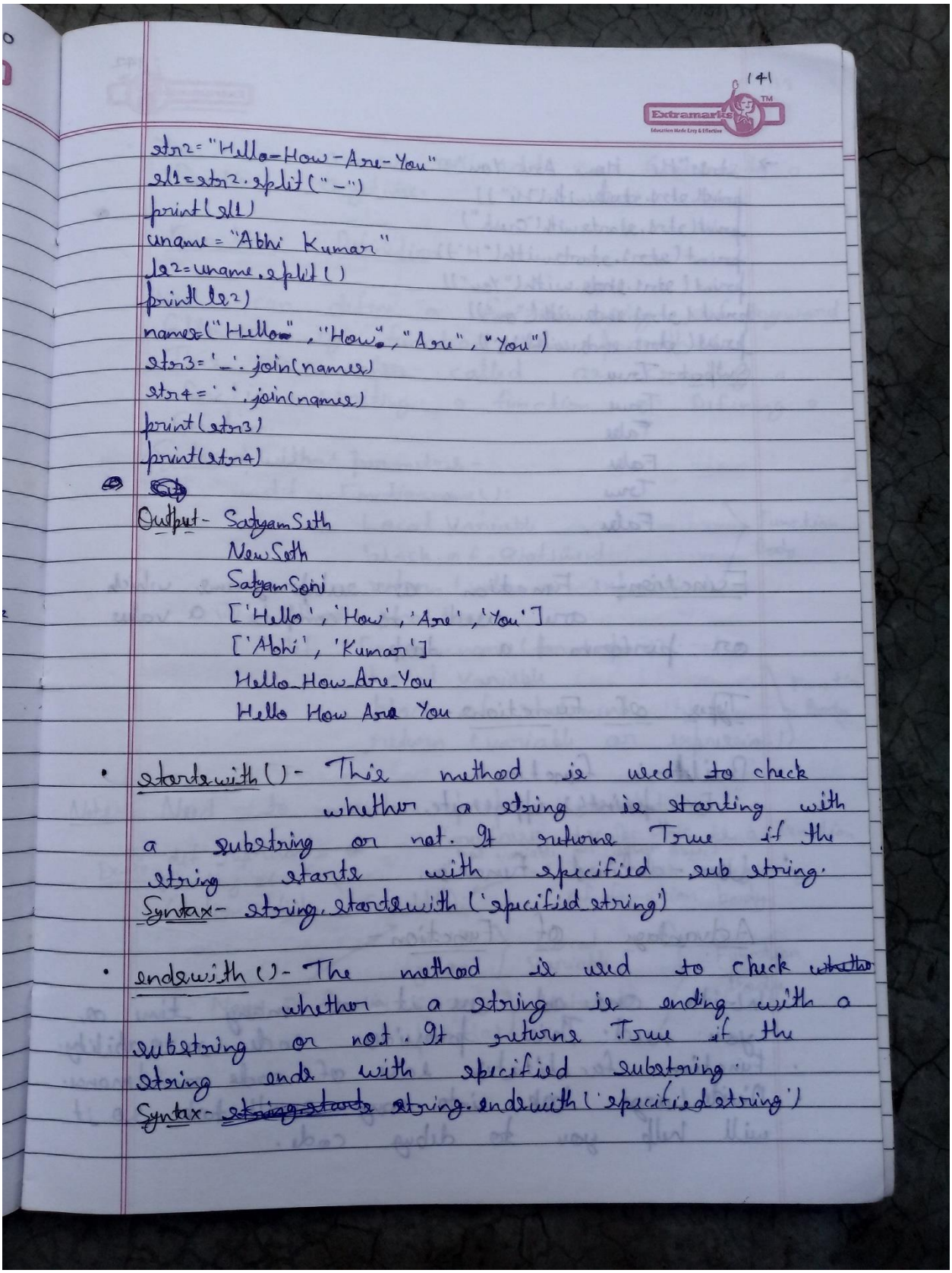
Output- Satyam
Satyam
Satyam
Satyam

- replace() - This method is used to replace a sub string in a string with another sub string.
  Syntax- string.replace(old, new)

- split() - This method is used to split/break a string into pieces. These pieces returns as a list.
  Syntax- string.split('separator')

- join() - This method is used to join strings into one string.
  Syntax- "separator".join(string_list)

```
name = "SatyamSeth"
print(name)
old = "Satyam"
new = "New"
str1 = name.replace(old, new)
print(str1)
print(name.replace("Seth", "Soni"))
```

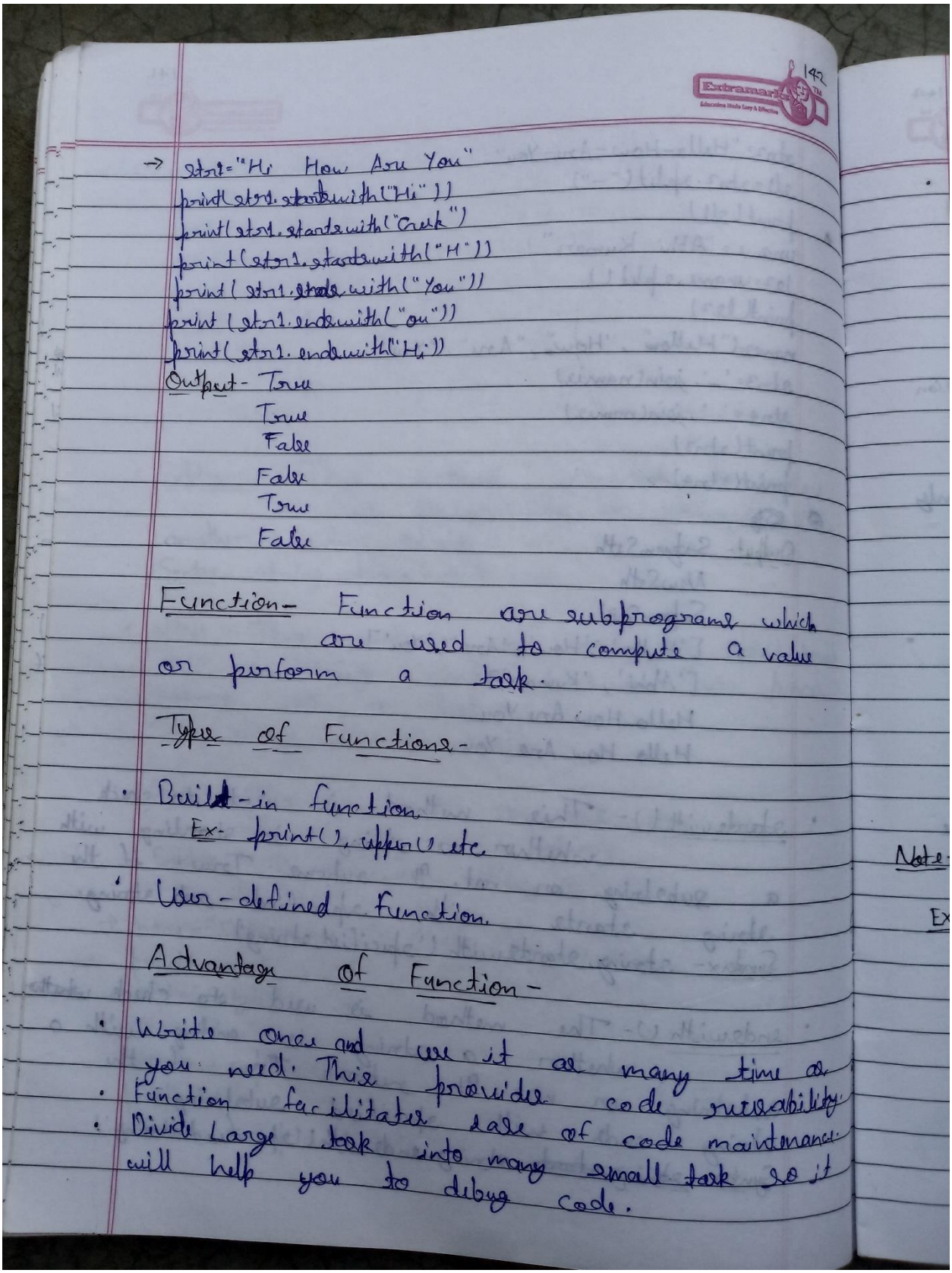


```
str2 = "Hello-How-Are-You"
sl1 = str2.split("-")
print(sl1)
uname = "Abhi Kumar"
sl2 = uname.split()
print(sl2)
names = ("Hello", "How", "Are", "You")
str3 = '_'.join(names)
str4 = ' '.join(names)
print(str3)
print(str4)
```

Output - SatyamSeth
NewSeth
SatyamSoni
['Hello', 'How', 'Are', 'You']
['Abhi', 'Kumar']
Hello_How_Are_You
Hello How Are You

- startswith() - This method is used to check whether a string is starting with a substring or not. It returns True if the string starts with specified sub string.
  Syntax - string.startswith('specified string')

- endswith() - The method is used to check whether a string is ending with a substring or not. It returns True if the string ends with specified substring.
  Syntax - string.endswith('specified string')



→ str1 = "Hi How Are You"
print(str1.startswith("Hi"))
print(str1.startswith("Check"))
print(str1.startswith("H"))
print(str1.endswith("You"))
print(str1.endswith("ou"))
print(str1.endswith("Hi"))
Output - True
True
False
False
True
False

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

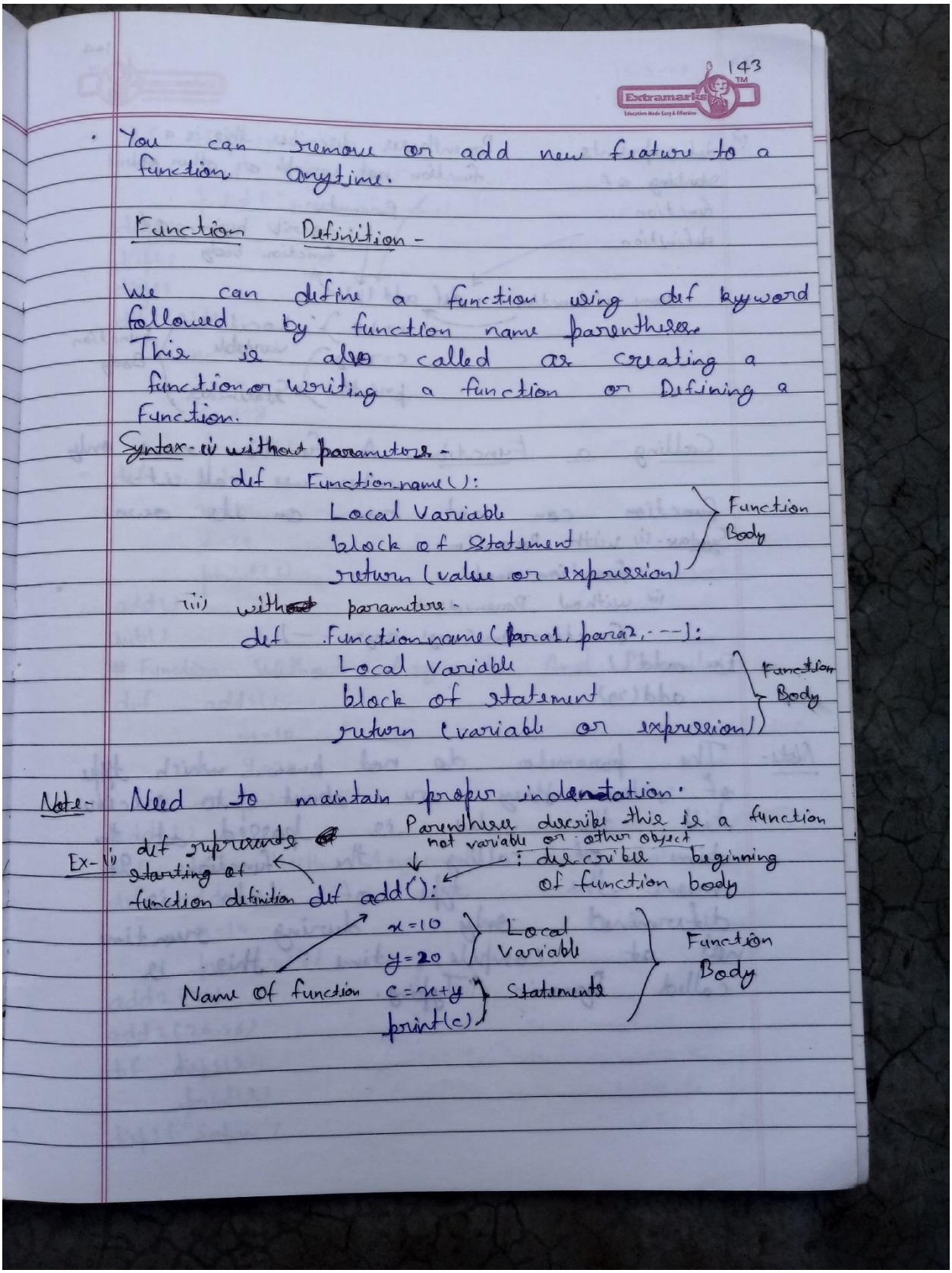
Type of Functions -

- Build-in function
  Ex- print(), upper() etc.
- User-defined function.

Advantage of Function -

- Write once and use it as many time as you need. This provides code reusability.
- Function facilitates ease of code maintenance.
- Divide Large task into many small task so it will help you to debug code.

Note -
Ex



143

- You can remove or add new feature to a function anytime.

Function Definition -

We can define a function using def keyword followed by function name parentheses. This is also called as creating a function or writing a function or Defining a Function.

Syntax - i) without parameters -

```
def Function_name():
    Local Variable
    block of Statement
    return (value or expression)
```

Function Body

ii) with parameters -

```
def Function_name(para1, para2, ---):
    Local Variable
    block of statement
    return (variable or expression)
```

Function Body

Note - Need to maintain proper indentation.

Ex- i)

```
def add():
    x=10
    y=20
    c=x+y
    print(c)
```

def represents starting of function definition

Parentheses describe this is a function not variable or other object

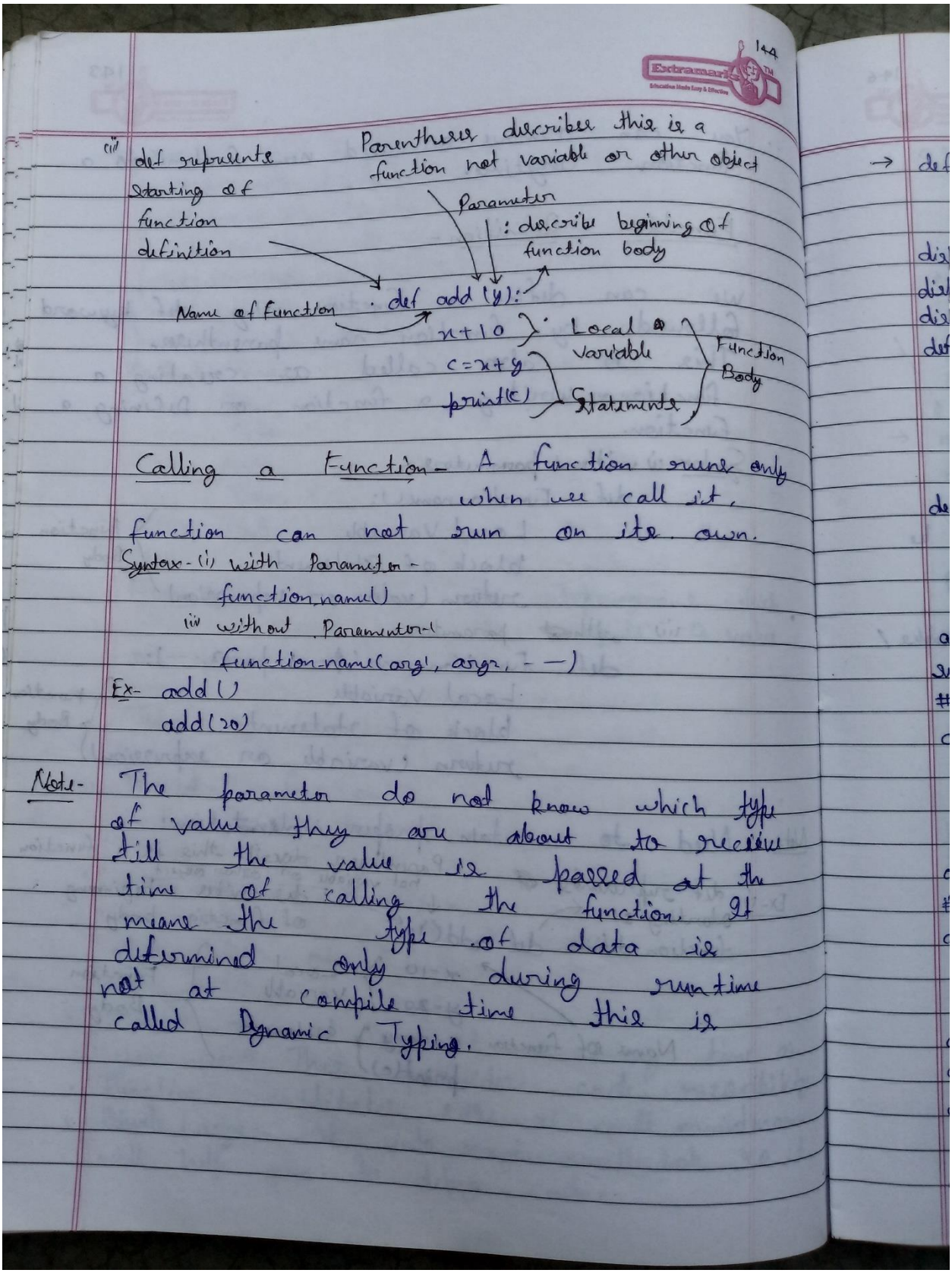
: describe beginning of function body

Name of function

x=10, y=20 } Local Variable

c=x+y, print(c) } Statements

Function Body



Calling a Function - A function runs only when we call it, function can not run on its own.

Syntax - (i) with Parameter -

function_name()

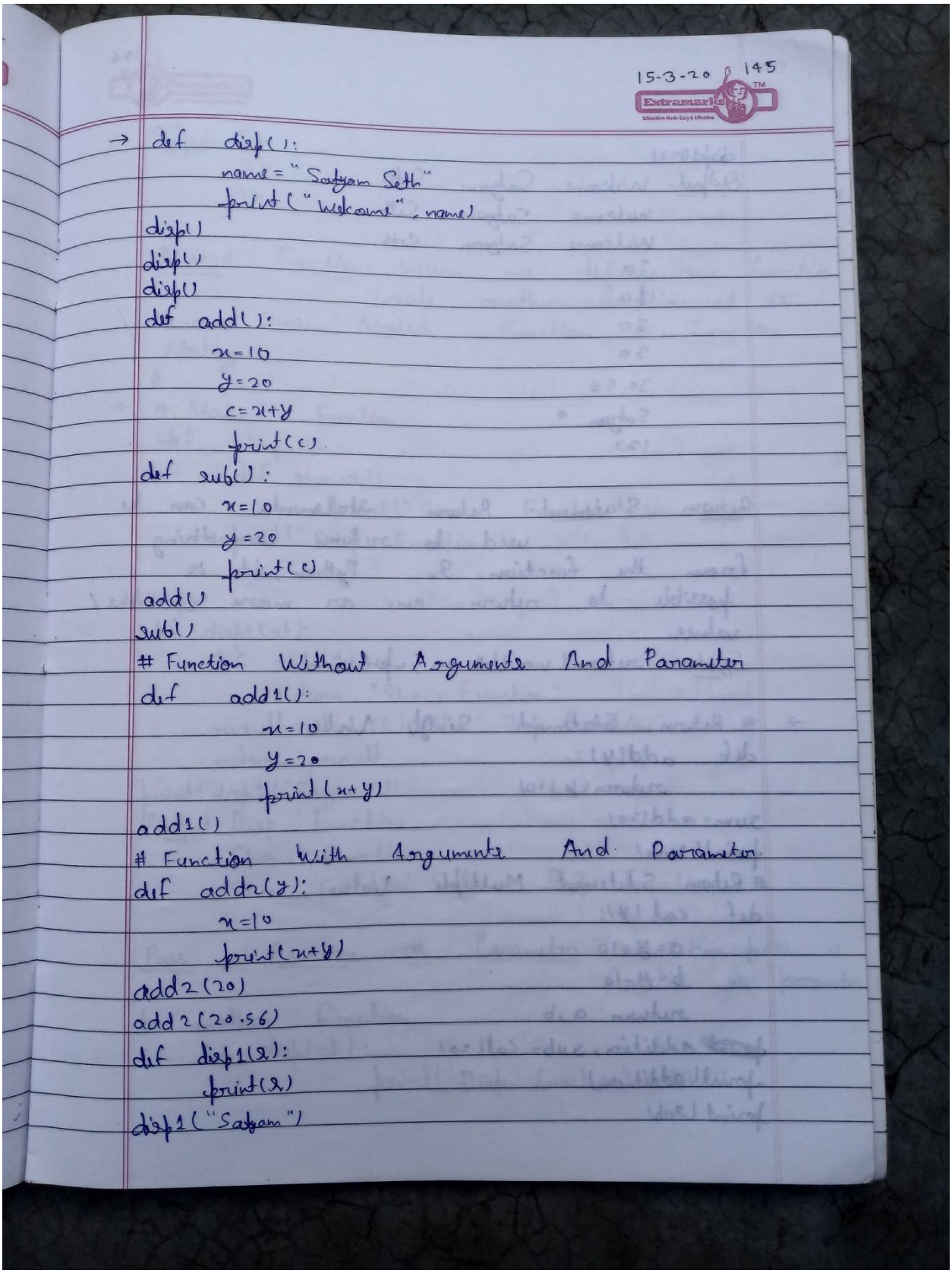
(ii) without Parameter -

function_name(arg1, arg2, - - -)

Ex- add()

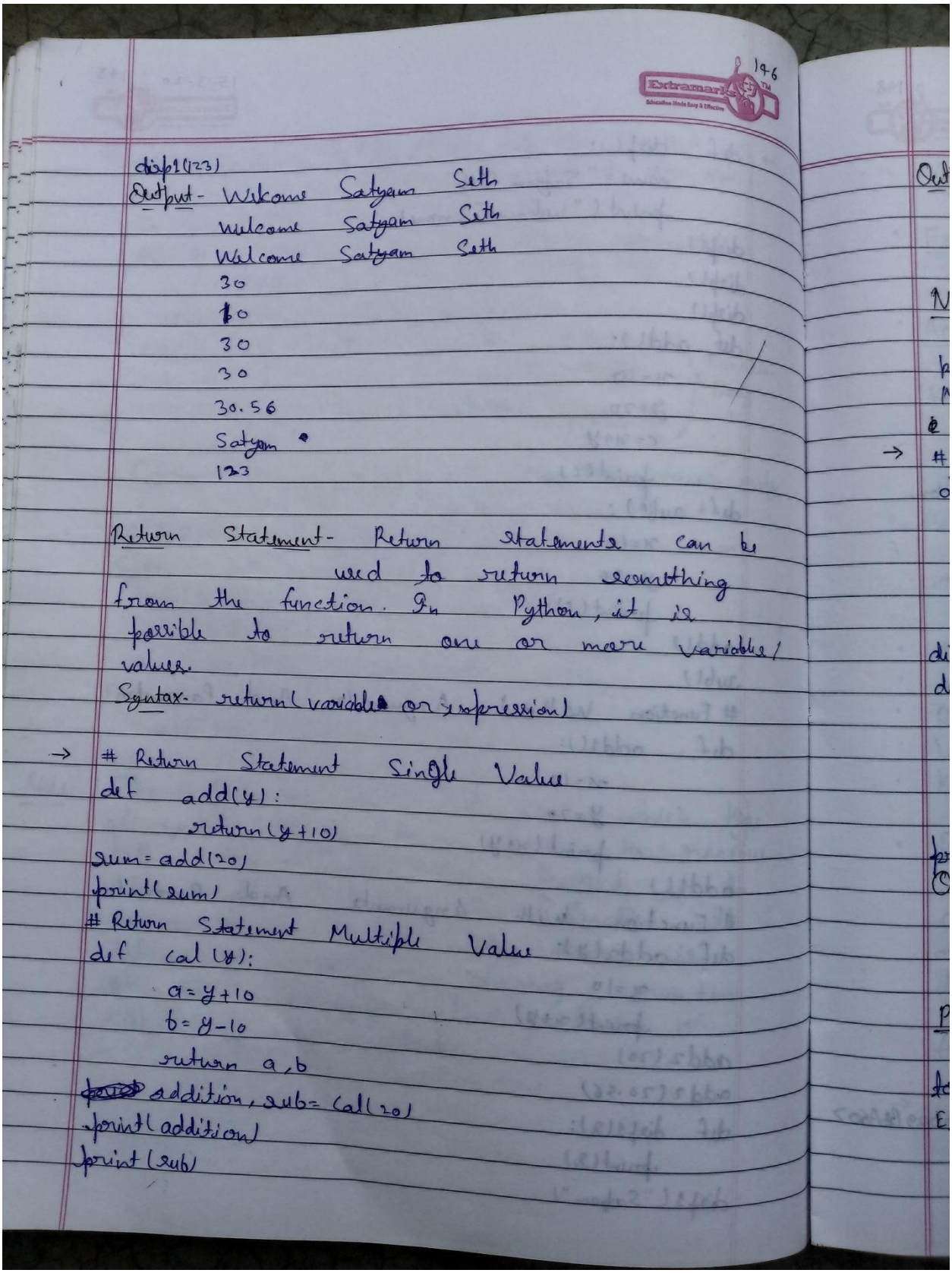
add(20)

Note- The parameter do not know which type of value they are about to receive till the value is passed at the time of calling the function. It means the type of data is determined only during runtime not at compile time this is called Dynamic Typing.



15-3-20 145

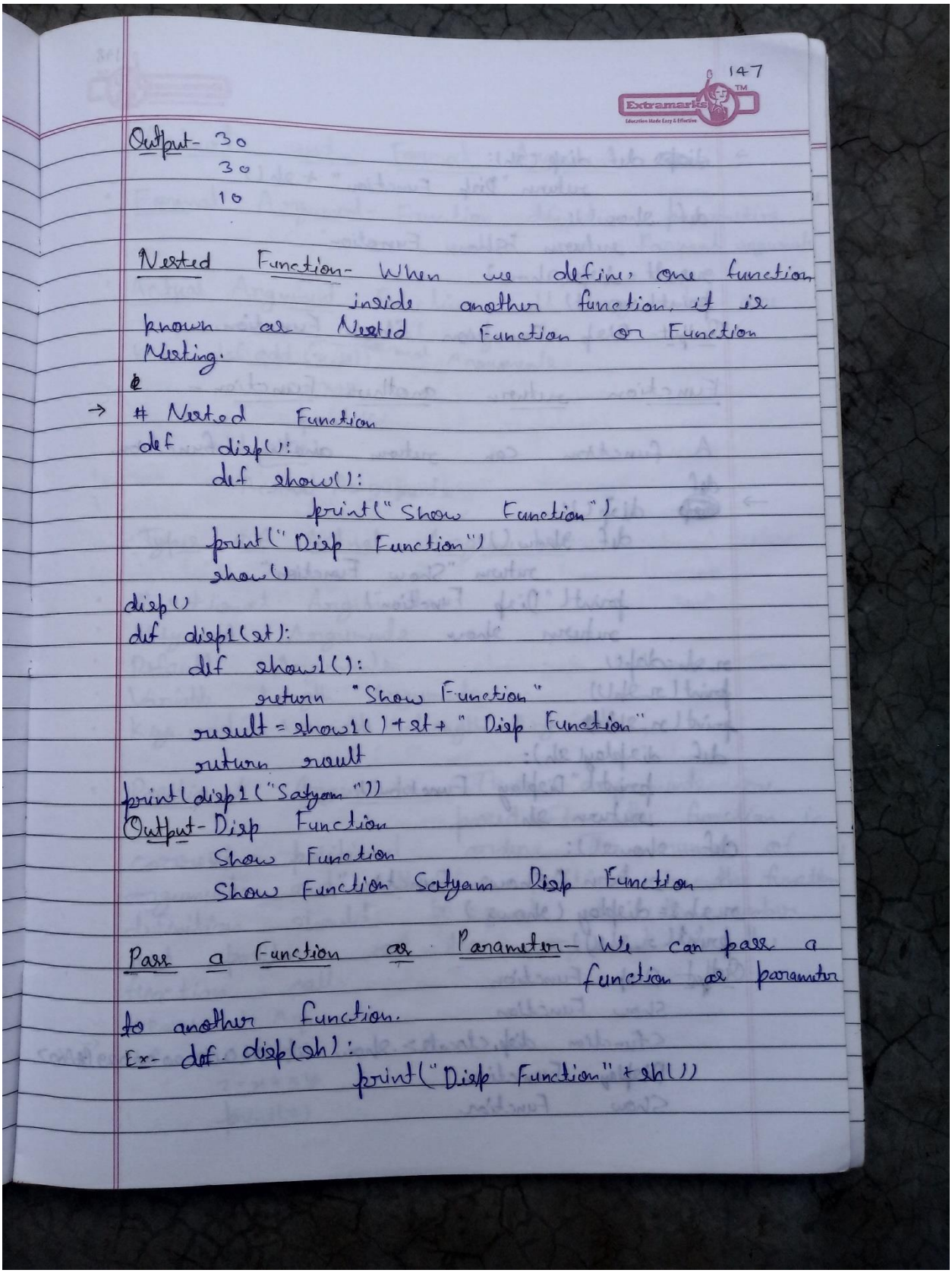
```
def disp():
    name = "Satyam Seth"
    print("Welcome", name)
disp()
disp()
disp()
def add():
    x=10
    y=20
    c=x+y
    print(c)
def sub():
    x=10
    y=20
    print(c)
add()
sub()
# Function Without Arguments And Paramater
def add1():
    x=10
    y=20
    print(x+y)
add1()
# Function With Arguments And Paramater
def add2(y):
    x=10
    print(x+y)
add2(20)
add2(20.56)
def disp1(x):
    print(x)
disp1("Satyam")
```



display(123)
Output- Welcome Satyam Seth
Welcome Satyam Seth
Welcome Satyam Seth
30
10
30
30
30.56
Satyam
123

Return Statement- Return statements can be used to return something from the function. In Python, it is possible to return one or more variables/values.
Syntax- return(variable or expression)

→ # Return Statement Single Value
def add(y):
    return(y+10)
sum=add(20)
print(sum)
# Return Statement Multiple Value
def cal(y):
    a=y+10
    b=y-10
    return a,b
addition,sub=cal(20)
print(addition)
print(sub)



147

Output- 30
30
10

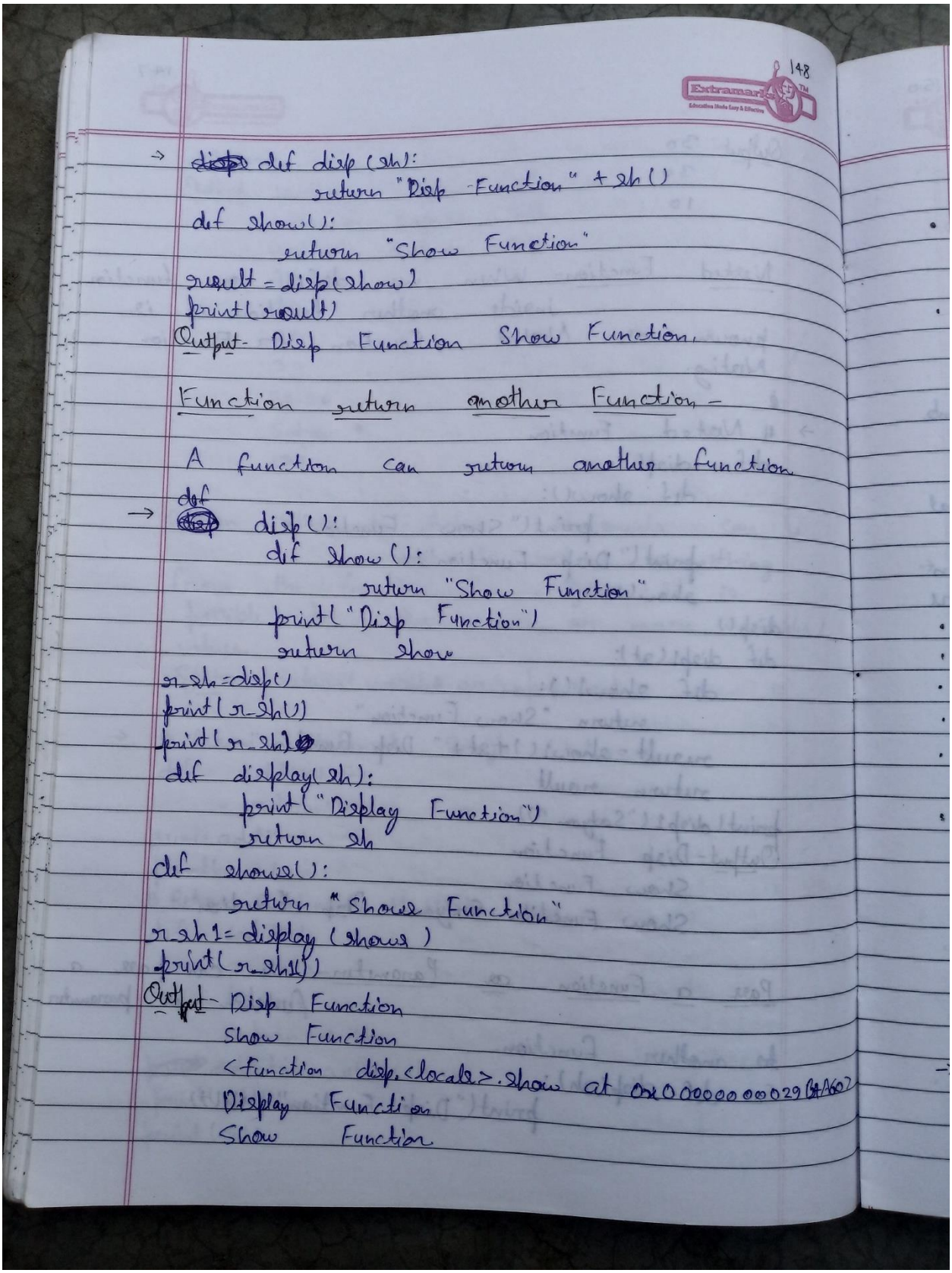
Nested Function- When we define one function inside another function, it is known as Nested Function or Function Nesting.

→ # Nested Function
def disp():
    def show():
        print("Show Function")
    print("Disp Function")
    show()
disp()
def disp1(st):
    def show1():
        return "Show Function"
    result = show1() + st + " Disp Function"
    return result
print(disp1("Satyam"))

Output- Disp Function
Show Function
Show Function Satyam Disp Function

Pass a Function as Paramater- We can pass a function as paramater to another function.

Ex- def disp(sh):
        print("Disp Function" + sh())



```
def disp(sh):
    return "Disp Function" + sh()
def show():
    return "Show Function"
result = disp(show)
print(result)
```

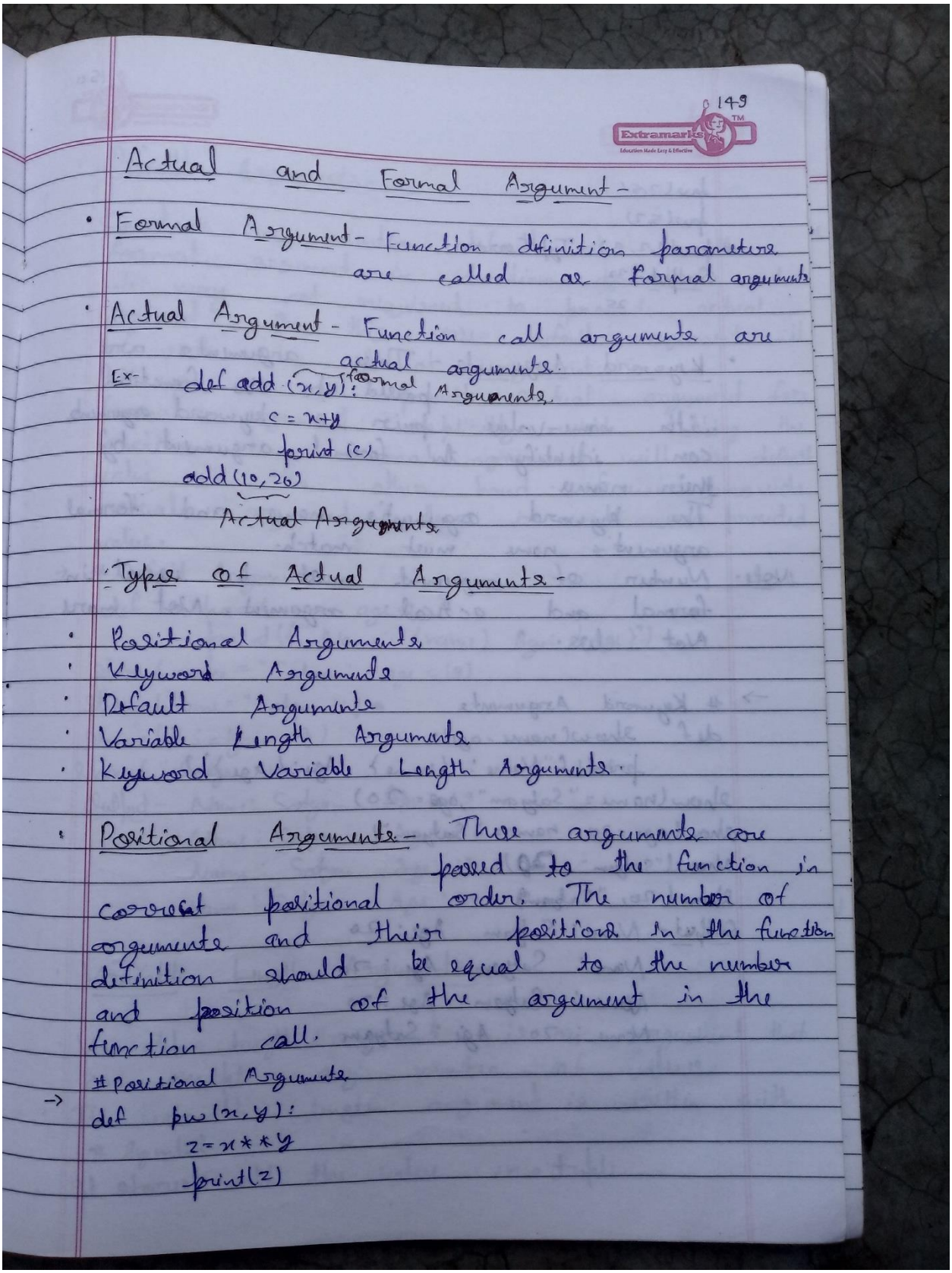
Output- Disp Function Show Function.

Function return another Function -

A function can return another function.

```
def disp():
    def show():
        return "Show Function"
    print("Disp Function")
    return show
r_sh = disp()
print(r_sh())
print(r_sh)
def display(sh):
    print("Display Function")
    return sh
def shows():
    return "Shows Function"
r_sh1 = display(shows)
print(r_sh1())
```

Output- Disp Function
Show Function
<function disp.<locals>.show at 0x0000000029BA60>
Display Function
Show Function



149

Actual and Formal Argument -

- **Formal Argument** - Function definition parameters are called as formal arguments
- **Actual Argument** - Function call arguments are actual arguments.

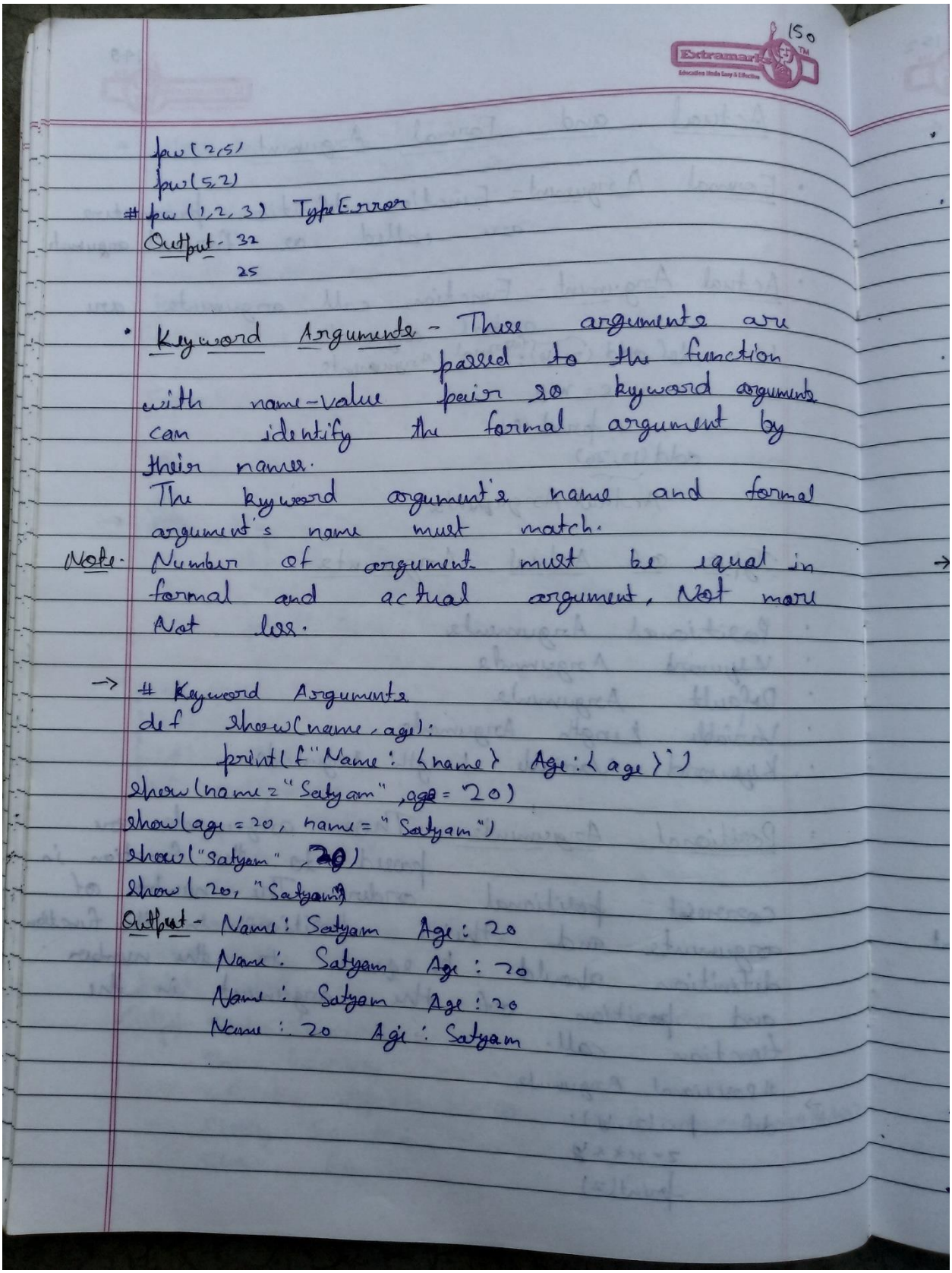
Ex- def add (x, y): (formal Arguments)
    c = x+y
    print (c)
add (10, 20)
(Actual Arguments)

Types of Actual Arguments -

- Positional Arguments
- Keyword Arguments
- Default Arguments
- Variable Length Arguments
- Keyword Variable Length Arguments.

- **Positional Arguments** - These arguments are passed to the function in correct positional order. The number of arguments and their positions in the function definition should be equal to the number and position of the argument in the function call.

→
```
# Positional Arguments
def pw(x, y):
    z = x**y
    print(z)
```

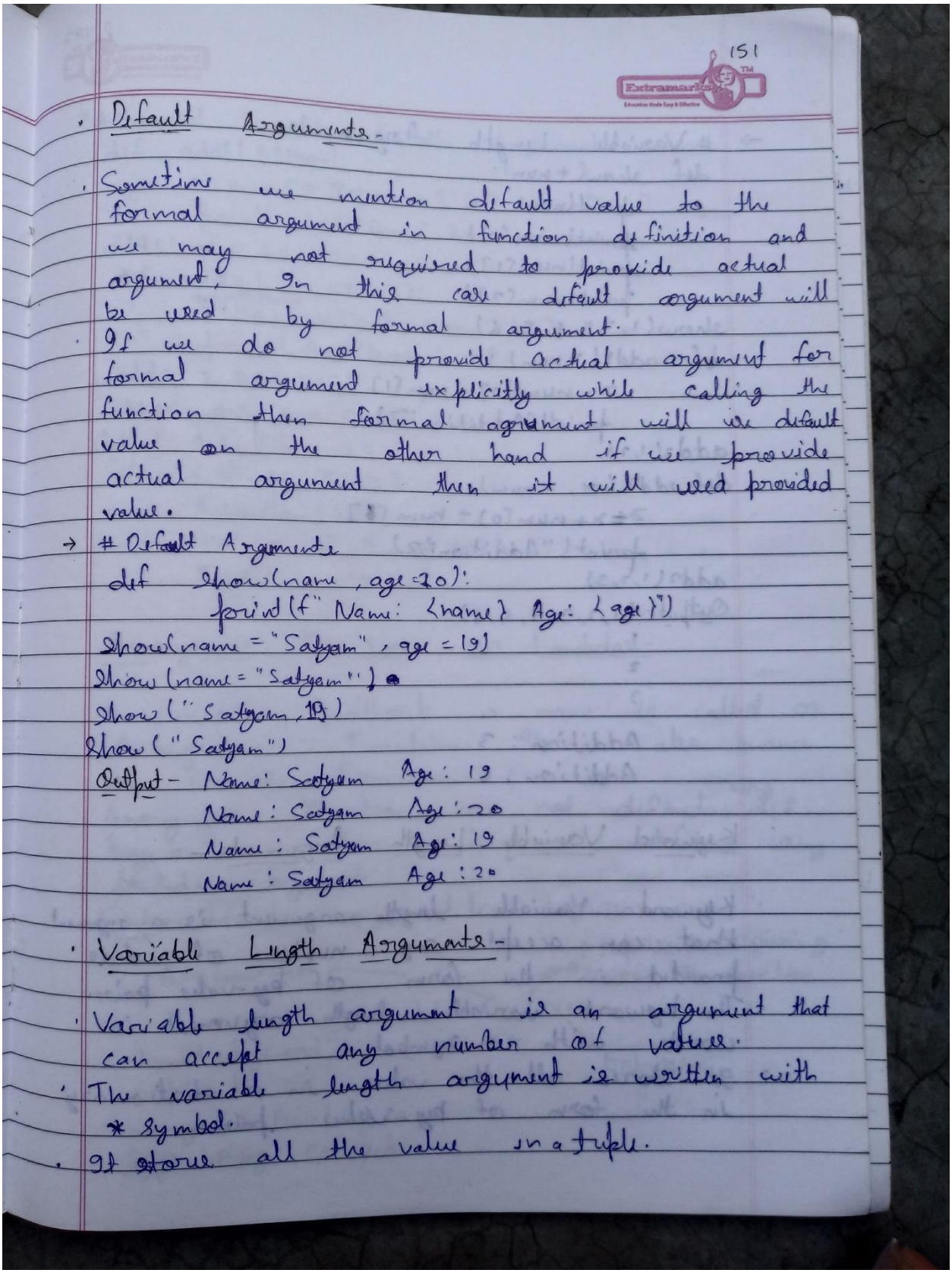


fun(2,5)
fun(5,2)
#fun(1,2,3) TypeError
Output - 32
25

- Keyword Arguments - These arguments are passed to the function with name-value pair so keyword arguments can identify the formal argument by their names.
The keyword argument's name and formal argument's name must match.

Note: Number of argument must be equal in formal and actual argument, Not more Not less.

→ # Keyword Arguments
def show(name, age):
print(f"Name: {name} Age: {age}")
show(name = "Satyam", age = 20)
show(age = 20, name = "Satyam")
show("satyam", 20)
show(20, "Satyam")
Output - Name: Satyam Age: 20
Name: Satyam Age: 20
Name: Satyam Age: 20
Name: 20 Age: Satyam



151

Default Arguments -

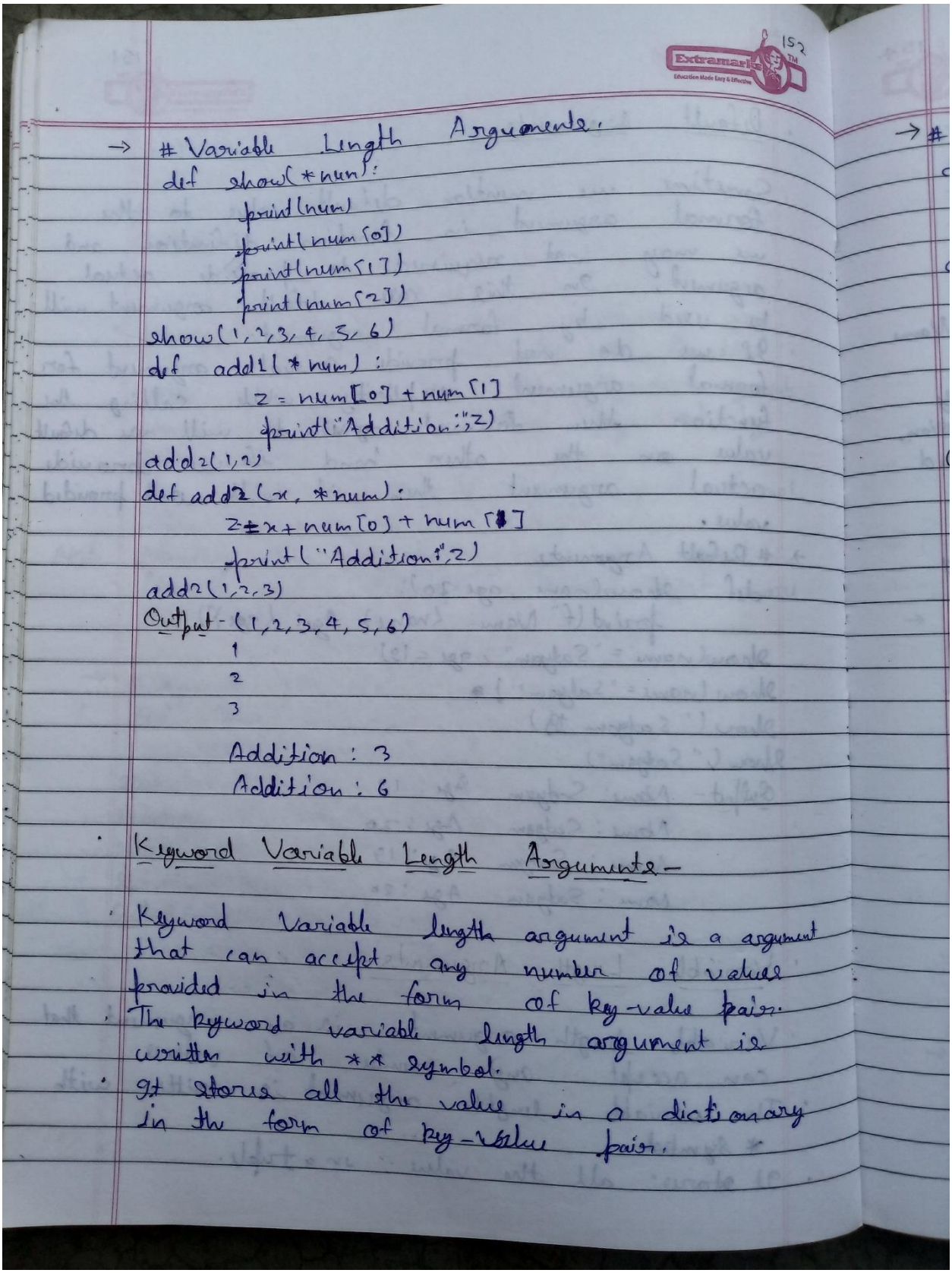
- Sometime we mention default value to the formal argument in function definition and we may not required to provide actual argument, in this case default argument will be used by formal argument.
- If we do not provide actual argument for formal argument explicitly while calling the function then formal argument will use default value on the other hand if we provide actual argument then it will used provided value.

```
# Default Arguments
def show(name, age=20):
    print(f"Name: {name} Age: {age}")
show(name = "Satyam", age = 19)
show(name = "Satyam")
show("Satyam", 19)
show("Satyam")
```

Output -
Name: Satyam Age: 19
Name: Satyam Age: 20
Name: Satyam Age: 19
Name: Satyam Age: 20

Variable Length Arguments -

- Variable length argument is an argument that can accept any number of values.
- The variable length argument is written with * symbol.
- It store all the value in a tuple.



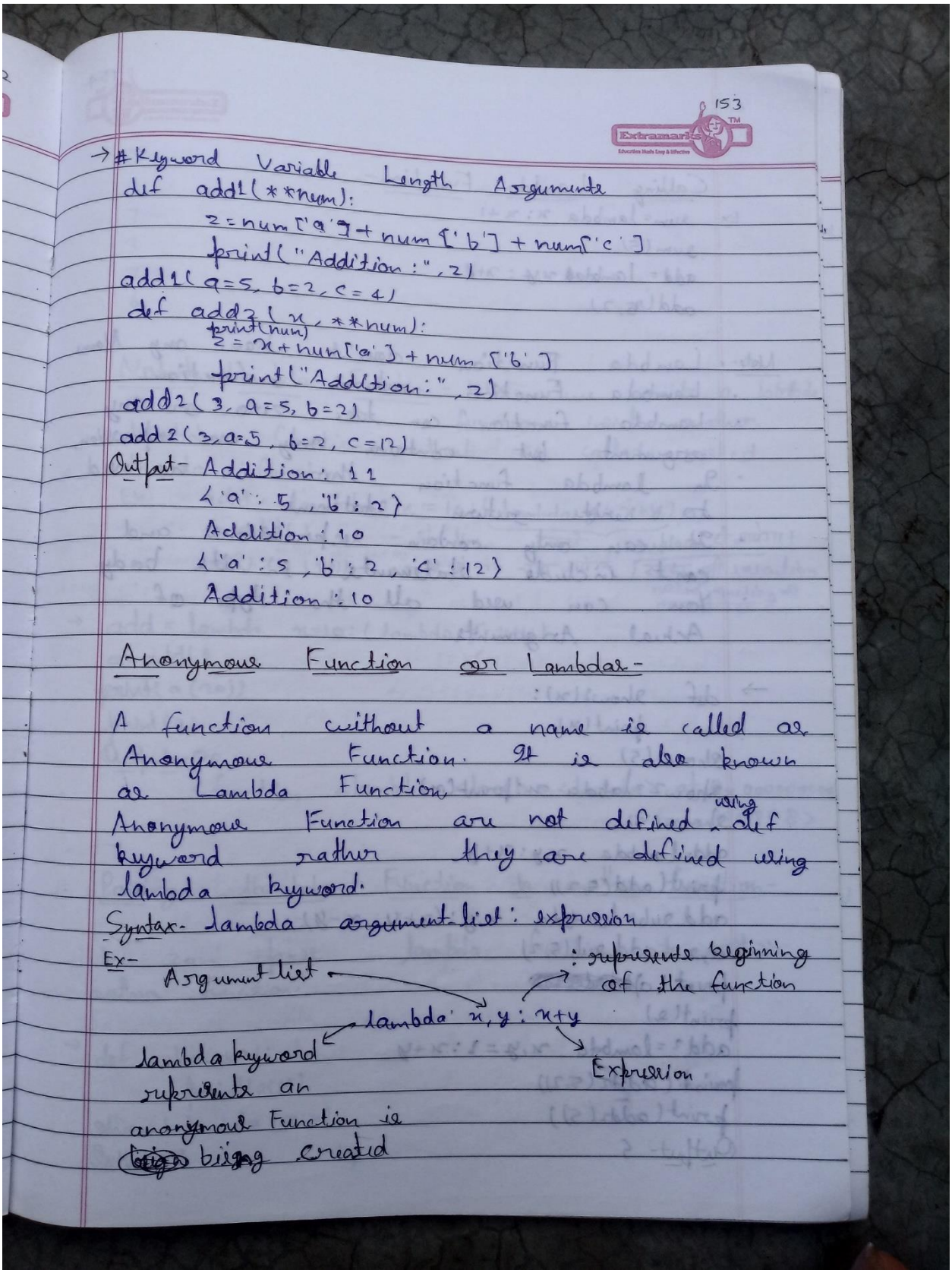
```
# Variable Length Arguments.
def show(*num):
    print(num)
    print(num[0])
    print(num[1])
    print(num[2])
show(1,2,3,4,5,6)
def add1(*num):
    z = num[0] + num[1]
    print("Addition:",z)
add1(1,2)
def add2(x, *num):
    z = x + num[0] + num[1]
    print("Addition:",z)
add2(1,2,3)
```

Output -

```
(1,2,3,4,5,6)
1
2
3
Addition : 3
Addition : 6
```

Keyword Variable Length Arguments -

- Keyword Variable length argument is a argument that can accept any number of values provided in the form of key-value pair.
- The keyword variable length argument is written with ** symbol.
- It stores all the value in a dictionary in the form of key-value pair.



→ # Keyword Variable Length Arguments

```
def add1(**num):
    z = num['a'] + num['b'] + num['c']
    print("Addition :", z)
add1(a=5, b=2, c=4)
def add2(x, **num):
    print(num)
    z = x + num['a'] + num['b']
    print("Addition :", z)
add2(3, a=5, b=2)
add2(3, a=5, b=2, c=12)
```

Output- Addition : 11
{'a': 5, 'b': 2}
Addition : 10
{'a': 5, 'b': 2, 'c': 12}
Addition : 10

Anonymous Function or Lambda-

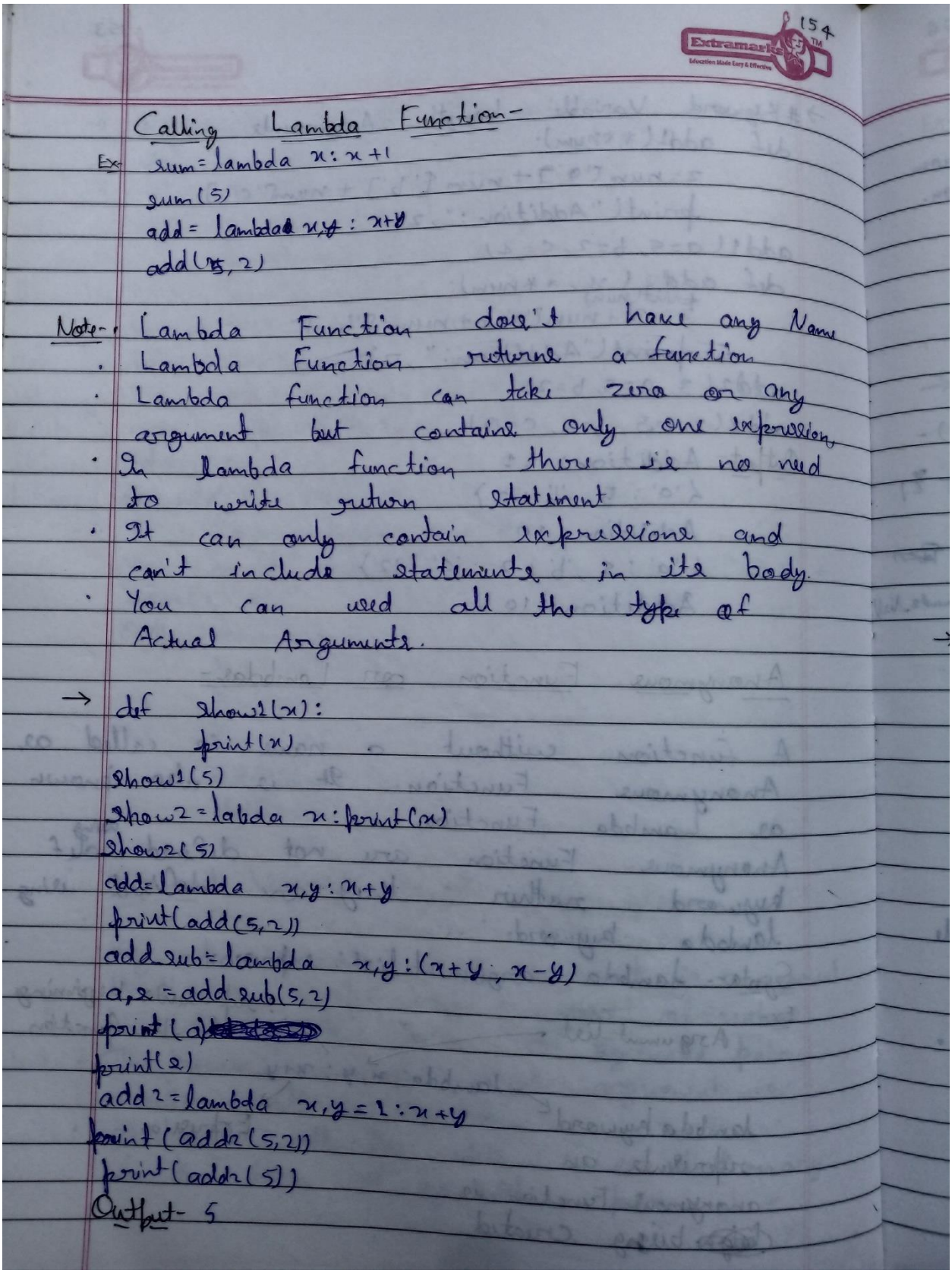
A function without a name is called as Anonymous Function. It is also known as Lambda Function.
Anonymous Function are not defined using def keyword rather they are defined using lambda keyword.

Syntax- lambda argument-list : expression

Ex-

```
lambda x, y : x+y
```

- Argument list
- : represents beginning of the function
- Expression
- lambda keyword represents an anonymous Function is being created



154

Calling Lambda Function-

Ex sum = lambda x : x + 1
sum(5)
add = lambda x,y : x+y
add(5, 2)

Note-
- Lambda Function doesn't have any Name
- Lambda Function returns a function
- Lambda function can take zero or any argument but contains only one expression
- In lambda function there is no need to write return statement
- It can only contain expressions and can't include statements in its body.
- You can used all the type of Actual Arguments.

→
```
def show1(x):
    print(x)
show1(5)
show2 = labda x : print(x)
show2(5)
add = lambda x,y : x+y
print(add(5,2))
add_sub = lambda x,y : (x+y, x-y)
a,s = add_sub(5,2)
print(a)
print(s)
add2 = lambda x,y=1 : x+y
print(add2(5,2))
print(add2(5))
```
Output- 5

5
7
7
3
7
6

Nested Lambda Function - When we write a lambda function inside another lambda function that is called nested lambda function.

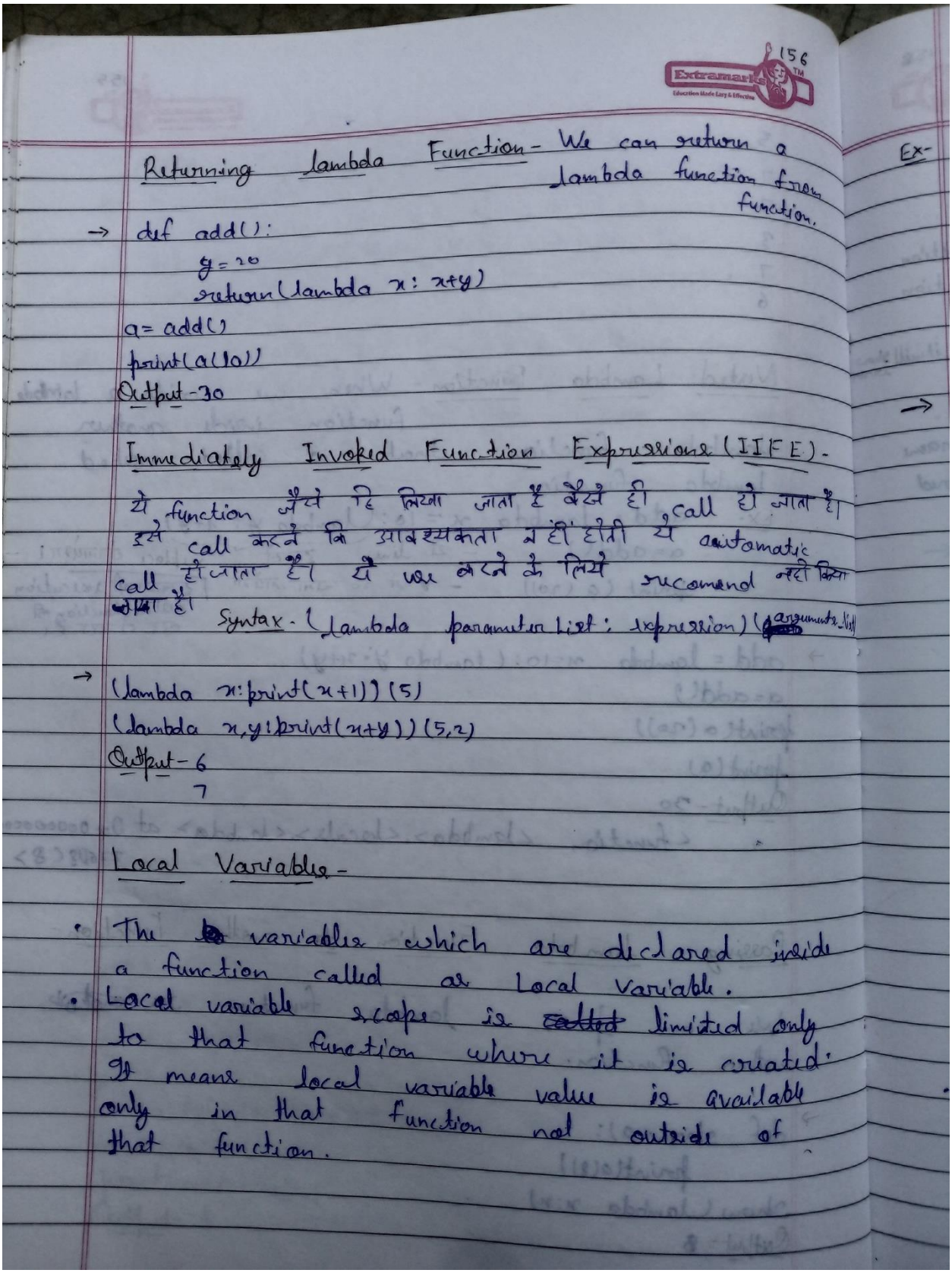
Ex. add = lambda x=10:(lambda y: x+y)
a=add() - यहां line इससे replace हो जायेगा।
print(a(20)) - y में 20 पास जायेगा इसका भी execution nested function की तरह होता है।

→
```
add = lambda x=10:(lambda y: x+y)
a=add()
print(a(20))
print(a)
```
Output - 30
<function <lambda>.<locals>.<lambda> at 0x000000000226D8C8>

Passing lambda Function to another Function -

We can pass lambda function to another function.

→
```
def show(a):
    print(a(8))
show(lambda x:x)
```
Output - 8



156

Returning lambda Function - We can return a lambda function from function.

```
def add():
    y=20
    return(lambda x: x+y)
a=add()
print(a(10))
```

Output - 30

Immediately Invoked Function Expressions (IIFE) -

यें function जैसे ही लिखा जाता है वैसे ही call हो जाता है। इसे call करने कि आवश्यकता नहीं होती यें automatic call हो जाता है। यें use करने के लिये recomend नहीं किया जाता है।

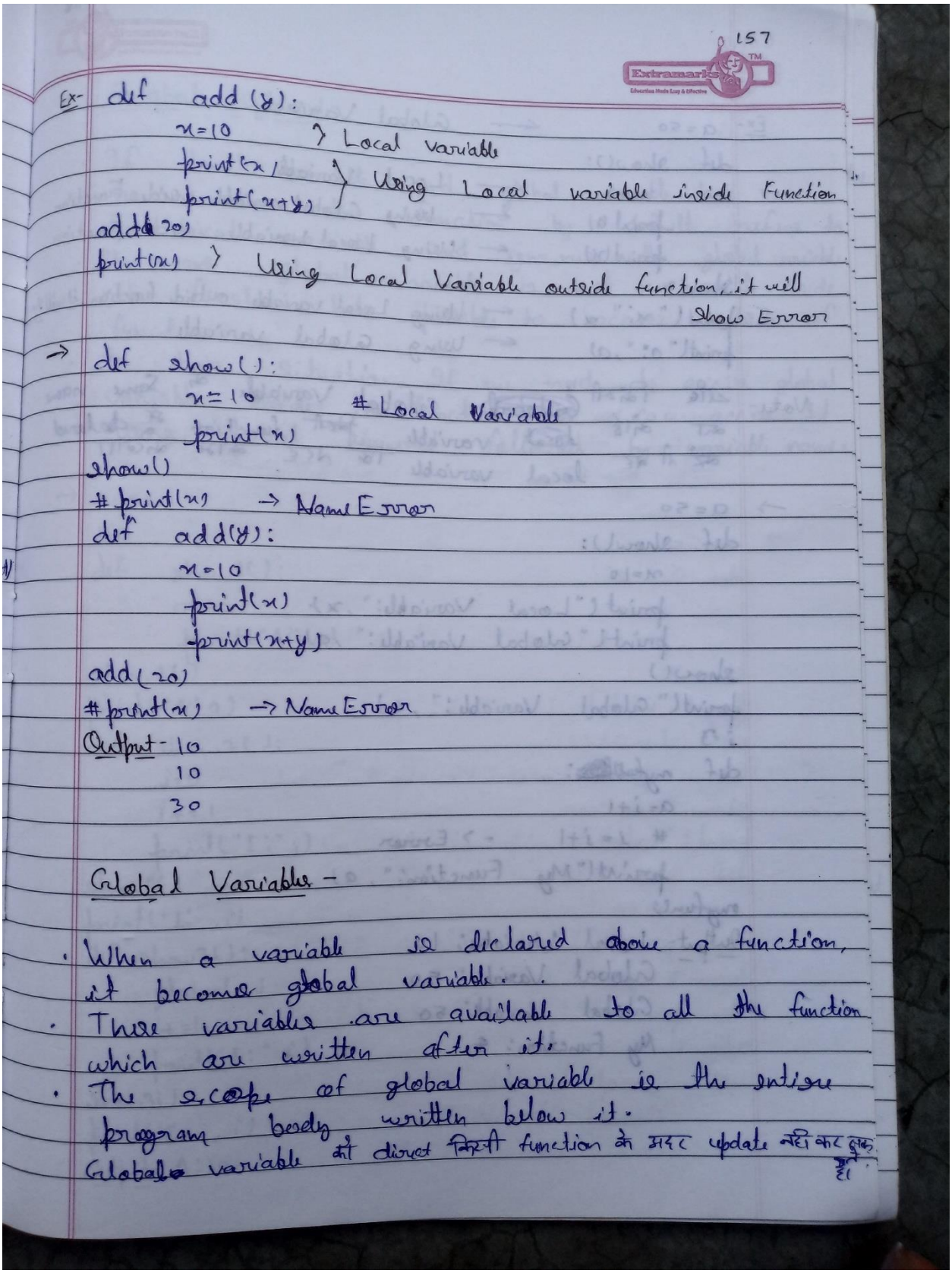
Syntax - (lambda parameter list : expression)(argument_list)

```
(lambda x:print(x+1))(5)
(lambda x,y:print(x+y))(5,2)
```

Output - 6
7

Local Variables -

- The variables which are declared inside a function called as Local variable.
- Local variable scope is limited only to that function where it is created. It means local variable value is available only in that function not outside of that function.



157

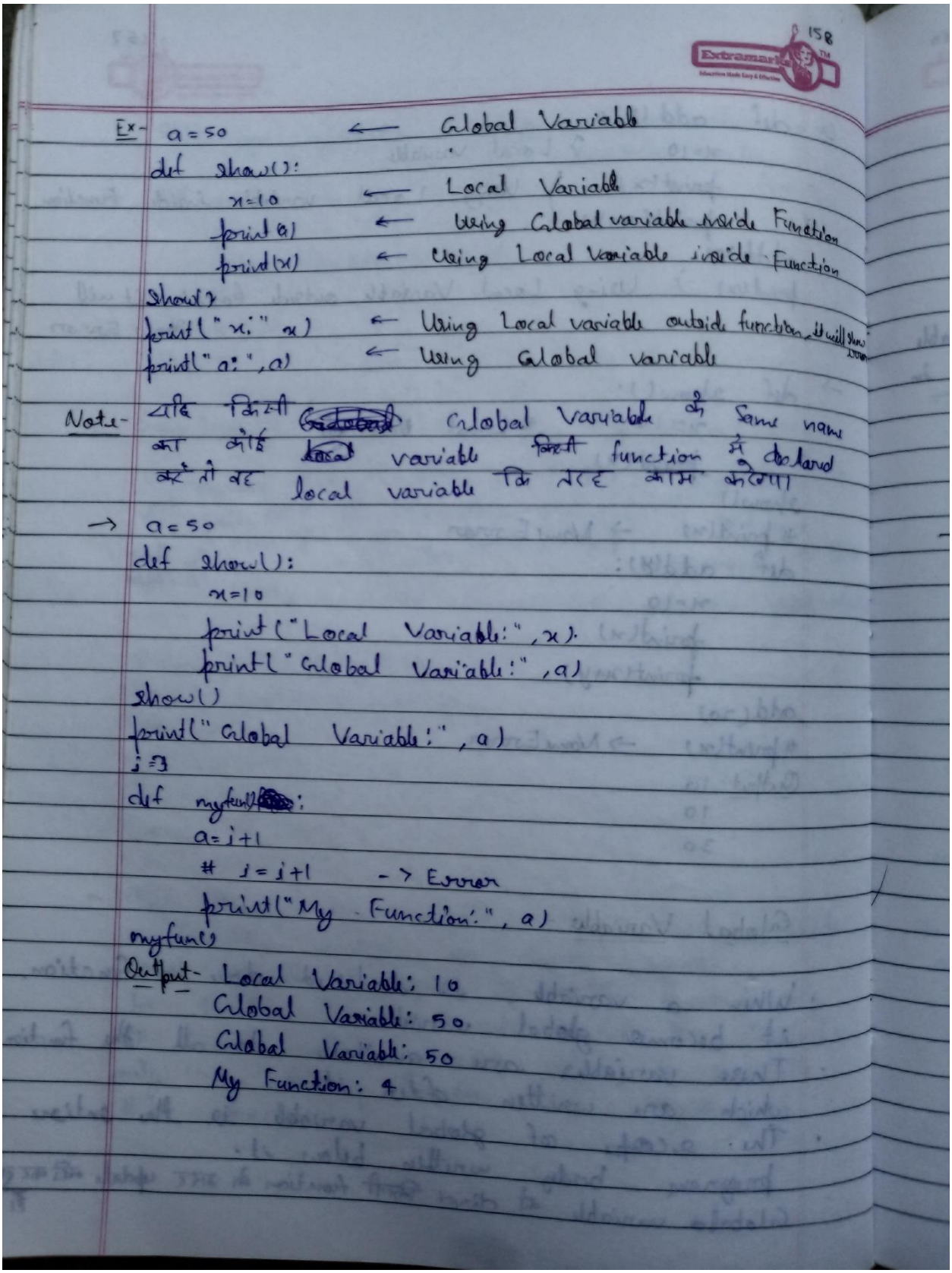
Ex- def add(y):
    x=10 } Local variable
    print(x) } Using Local variable inside Function
    print(x+y)
add(20)
print(x) } Using Local Variable outside function, it will show Error

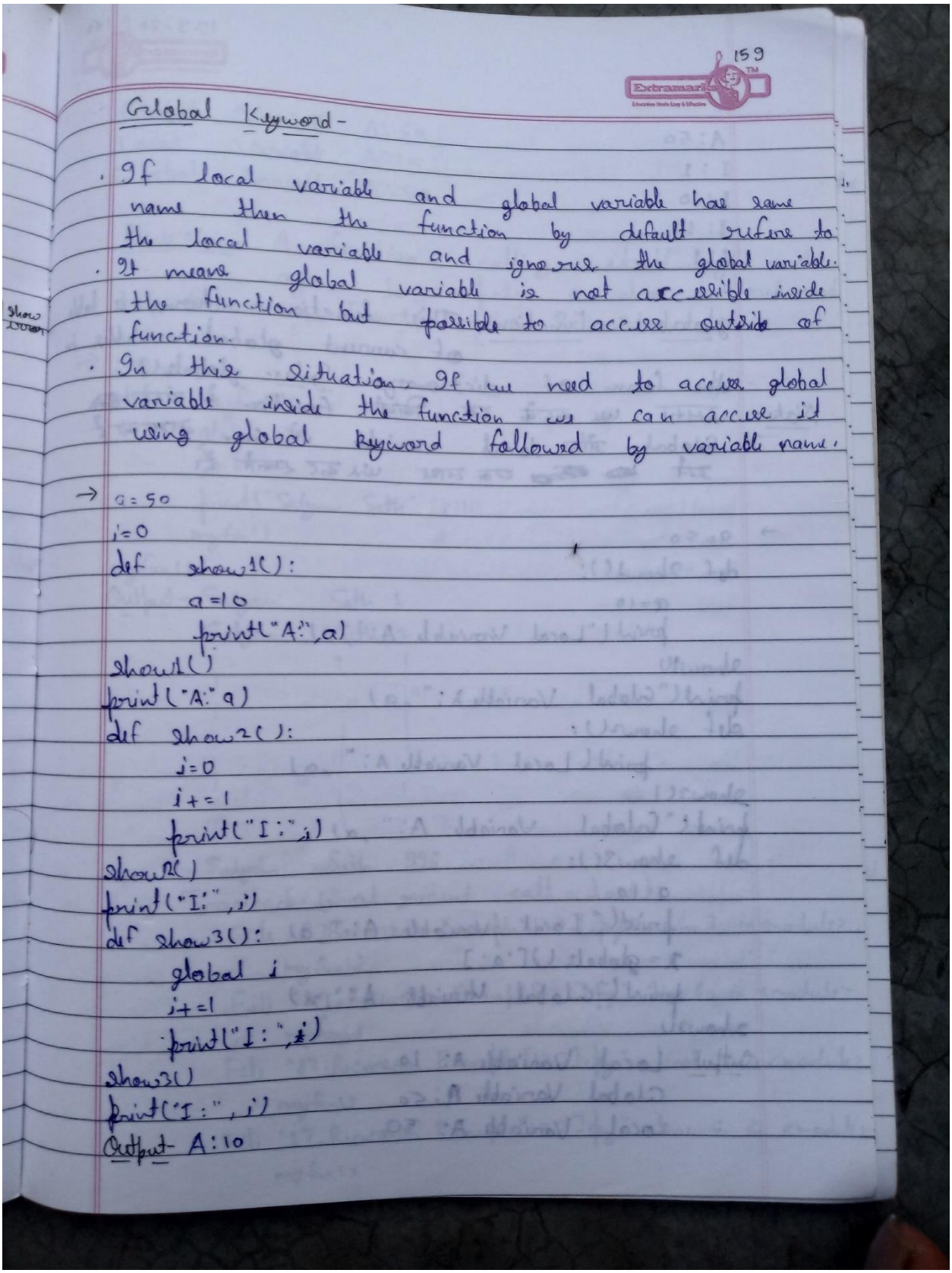
→ def show():
    x=10 # Local Variable
    print(x)
show()
#print(x) → NameError
def add(y):
    x=10
    print(x)
    print(x+y)
add(20)
#print(x) → NameError
Output- 10
10
30

Global Variable -

- When a variable is declared above a function, it becomes global variable.
- These variables are available to all the function which are written after it.
- The scope of global variable is the entire program body written below it.

Global variable को direct किसी function के अंदर update नहीं कर सकते हैं।


158
Extramarks
Ex- a=50 ← Global Variable
def show():
x=10 ← Local Variable
print(a) ← Using Global variable inside Function
print(x) ← Using Local Variable inside Function
show()
print("x:" x) ← Using Local variable outside function, it will show error
print("a:",a) ← Using Global variable
Note- यदि किसी Global Variable के same name का कोई local variable function में declared है तो वह local variable ही वरीयता लेगा।
→ a=50
def show():
x=10
print("Local Variable:",x)
print("Global Variable:",a)
show()
print("Global Variable:",a)
i=3
def myfunc():
a=i+1
# i=i+1 -> Error
print("My Function:",a)
myfunc()
Output- Local Variable: 10
Global Variable: 50
Global Variable: 50
My Function: 4



159

Global Keyword-

- If local variable and global variable has same name then the function by default refers to the local variable and ignores the global variable.
- It means global variable is not accessible inside the function but possible to access outside of function.
- In this situation, If we need to access global variable inside the function we can access it using global keyword followed by variable name.

```
a=50
i=0
def show1():
    a=10
    print("A:",a)
show1()
print("A:",a)
def show2():
    i=0
    i+=1
    print("I:",i)
show2()
print("I:",i)
def show3():
    global i
    i+=1
    print("I:",i)
show3()
print("I:",i)
```

Output- A:10

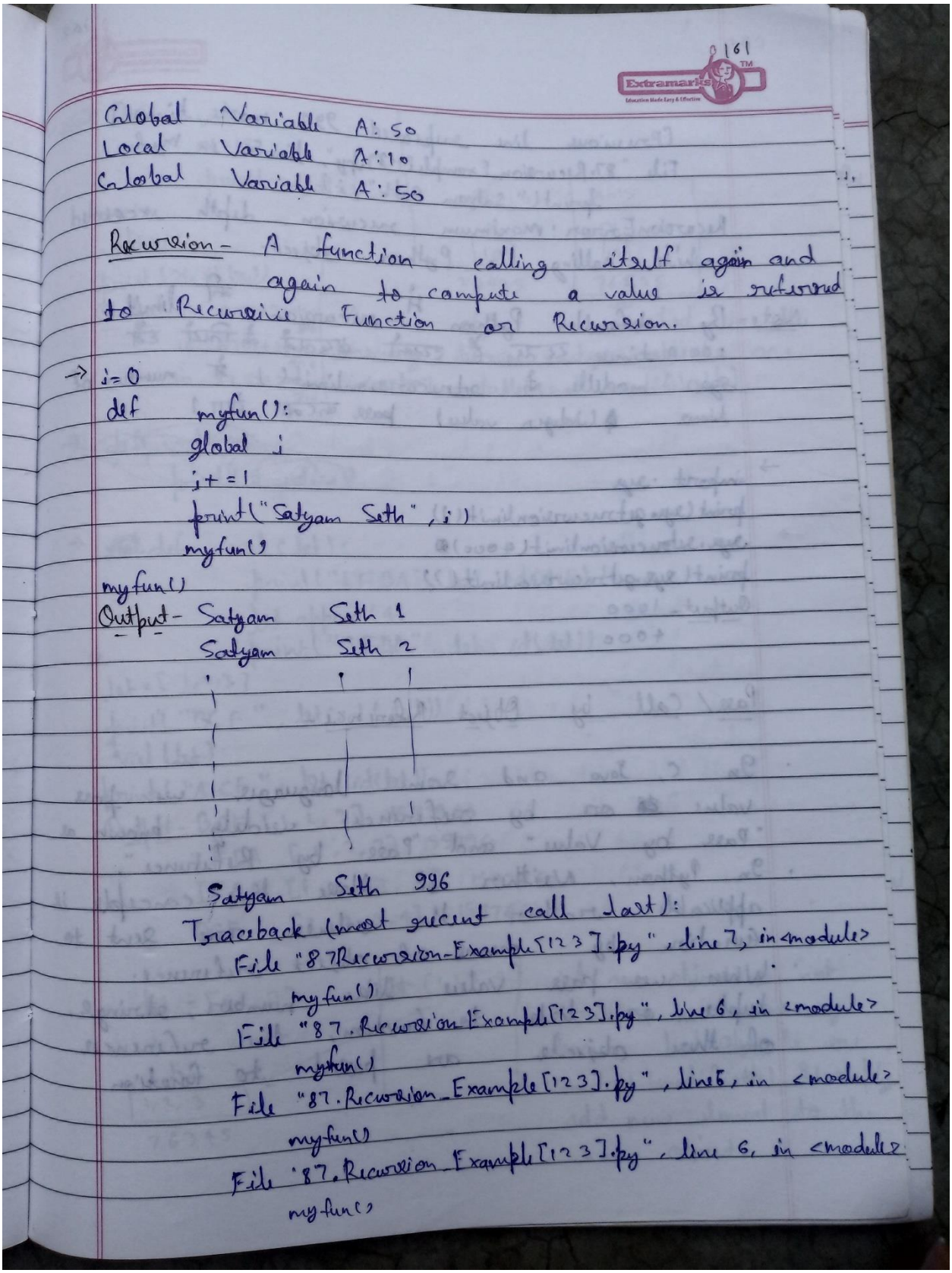
A: 50
I : 1
I : 0
I: 1
I: 1

globals() Function- This function returns a table of current global variables in the form of dictionary. Syntax- globals()

Note- इसका use करके हम किसी function के अंदर Global और local variable जो same नाम का है उन्हें एक साथ use कर सकते हैं।

→
```
a=50
def show1():
    a=10
    print("Local Variable A:", a)
show1()
print("Global Variable A:", a)
def show2():
    print("Local Variable A:", a)
show2()
print("Global Variable A:", a)
def show3():
    a=10
    print("Local Variable A:", a)
    x=globals()['a']
    print("Global Variable A:", x)
show3()
```

Output- Local Variable A: 10
Global Variable A: 50
Local Variable A: 50



161

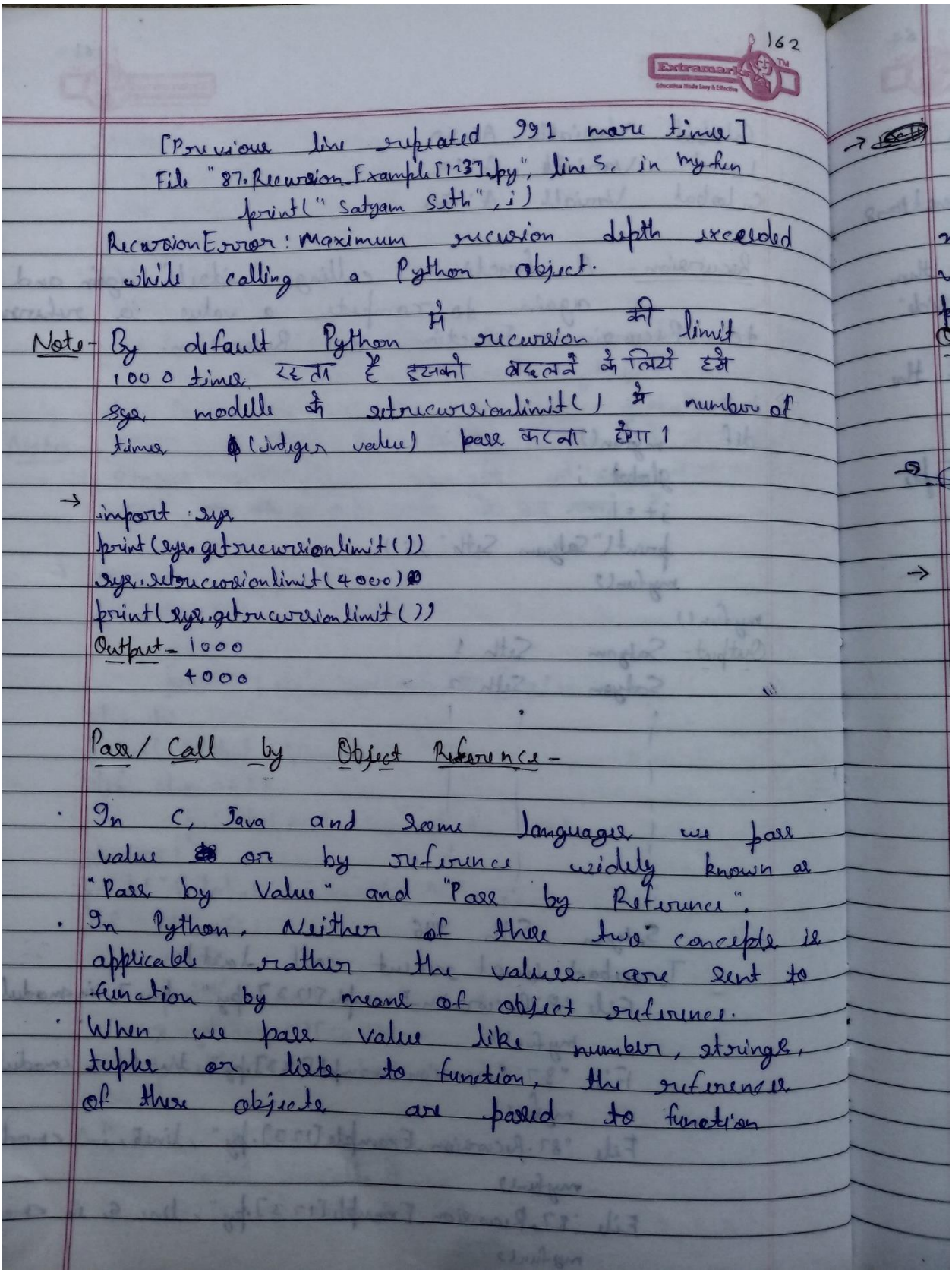
Global Variable A: 50
Local Variable A: 10
Global Variable A: 50

Recursion - A function calling itself again and again to compute a value is returned to Recursive Function or Recursion.

```
i=0
def myfun():
    global i
    i+=1
    print("Satyam Seth", i)
    myfun()
myfun()
```

Output - Satyam Seth 1
Satyam Seth 2

Satyam Seth 996
Traceback (most recent call last):
File "87Recursion-Example[123].py", line 7, in <module>
myfun()
File "87.Recursion Example[123].py", line 6, in <module>
myfun()
File "87.Recursion_Example[123].py", line 6, in <module>
myfun()
File "87.Recursion_Example[123].py", line 6, in <module>
myfun()



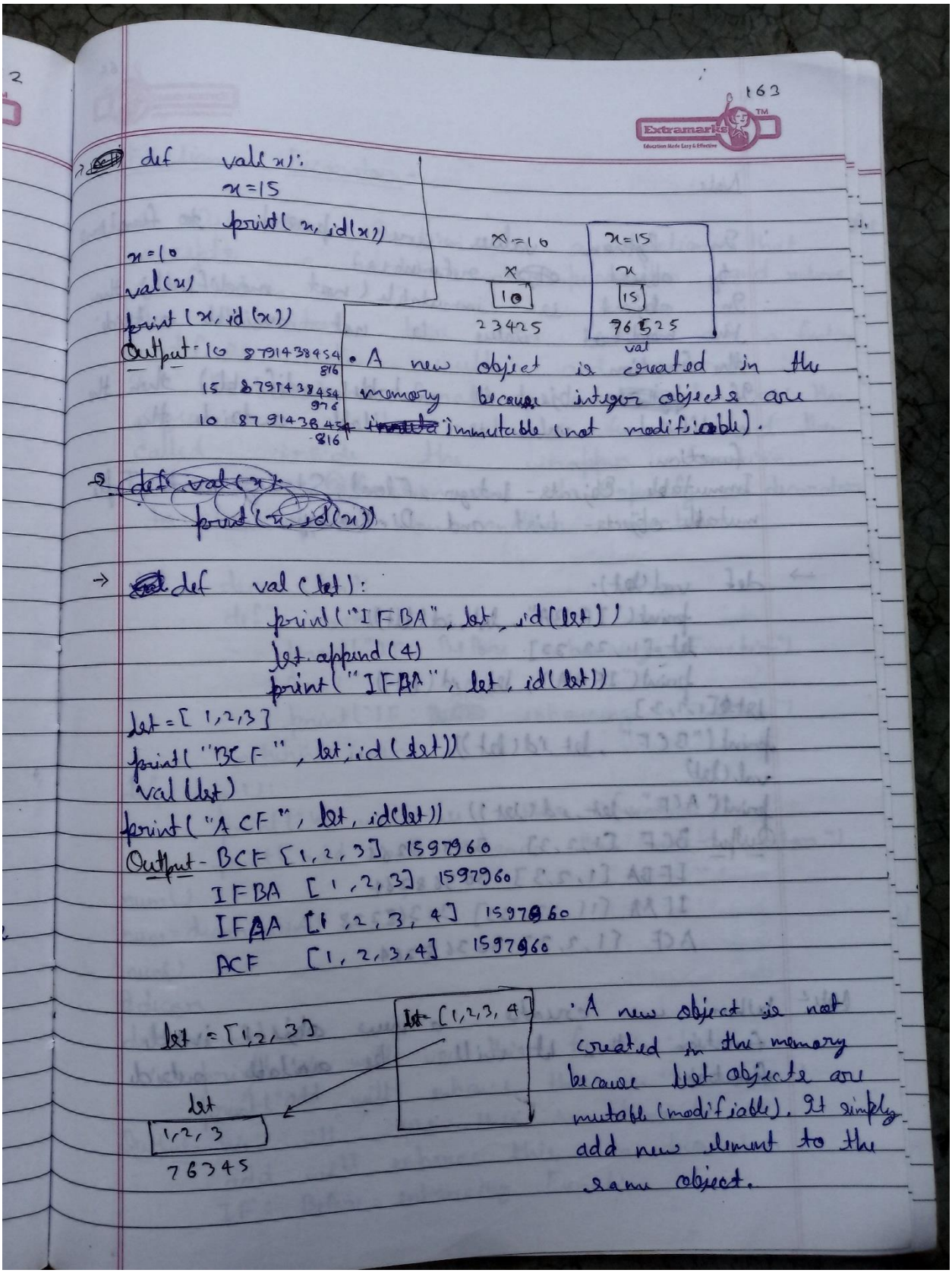
[Previous line repeated 991 more times]
File "87.Recursion_Example[1-37].py", line 5, in myfun
print("Satyam Seth", i)
RecursionError: maximum recursion depth exceeded while calling a Python object.

Note- By default Python में recursion की limit 1000 time रहता है इसको बढ़ाने के लिये हमे sys. module के setrecursionlimit() में number of time (integer value) pass करना होगा।

→
```
import sys
print(sys.getrecursionlimit())
sys.setrecursionlimit(4000)
print(sys.getrecursionlimit())
```
Output- 1000
4000

Pass / Call by Object Reference -

- In C, Java and some language we pass value or by reference widely known as "Pass by Value" and "Pass by Reference".
- In Python, Neither of these two concepts is applicable rather the values are sent to function by means of object reference.
- When we pass value like number, strings, tuples or lists to function, the references of these objects are passed to function



```
def val(x):
    x=15
    print(x, id(x))
x=10
val(x)
print(x, id(x))
```

Output: 10 8791438454 816
15 8791438454 976
10 8791438454 816

x=10 → 23425 (10)
x=15 → 76525 (15) (val)

• A new object is created in the memory because integer objects are immutable (not modifiable).

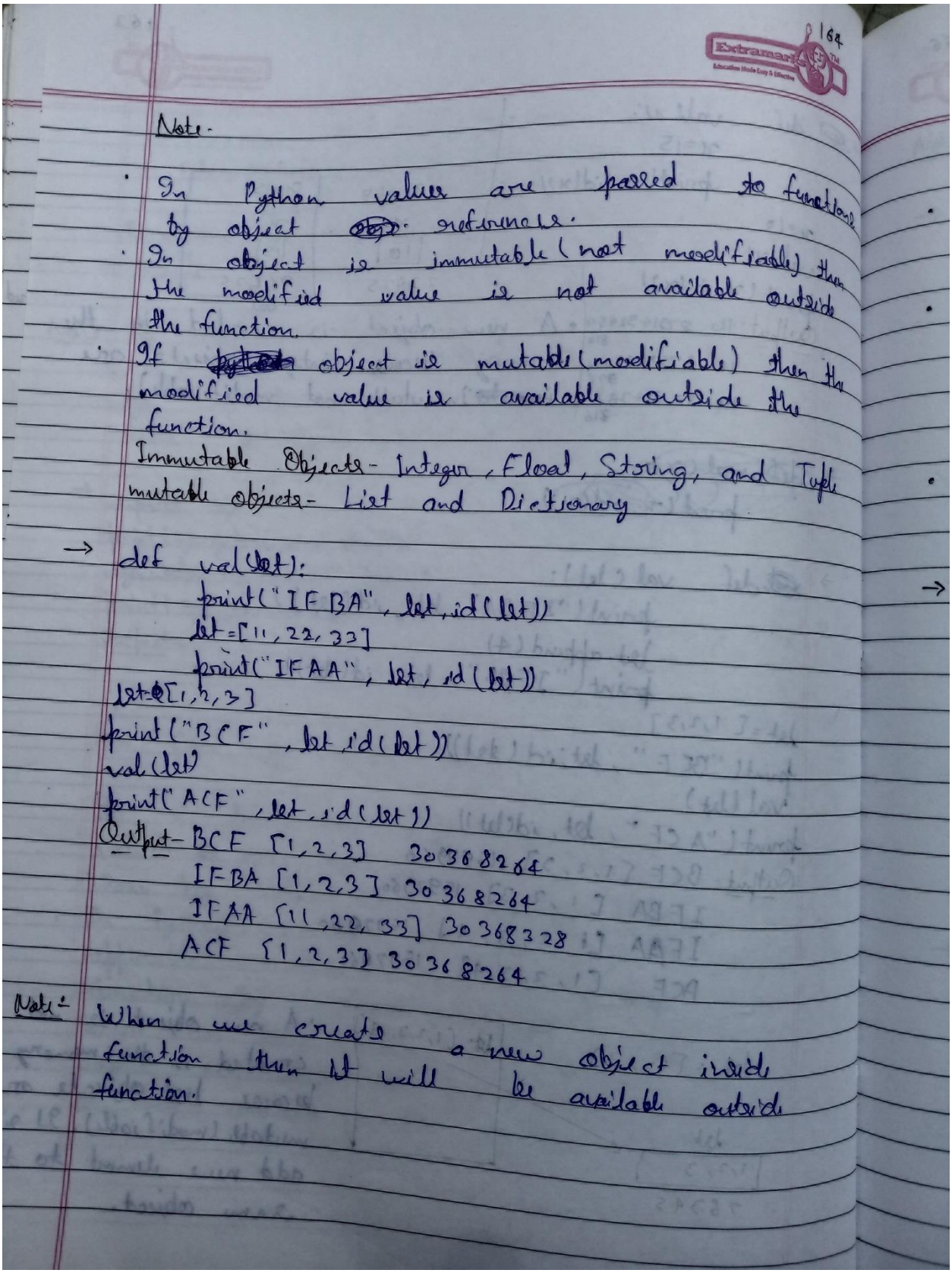
```
def val(x):
    print(x, id(x))
```

```
def val(let):
    print("IFBA", let, id(let))
    let.append(4)
    print("IFAA", let, id(let))
let=[1,2,3]
print("BCF", let, id(let))
val(let)
print("ACF", let, id(let))
```

Output - BCF [1,2,3] 1597960
IFBA [1,2,3] 1597960
IFAA [1,2,3,4] 1597960
ACF [1,2,3,4] 1597960

let=[1,2,3] → 76345 (let) (1,2,3)
let [1,2,3,4]

• A new object is not created in the memory because list objects are mutable (modifiable). It simply add new element to the same object.



Note.

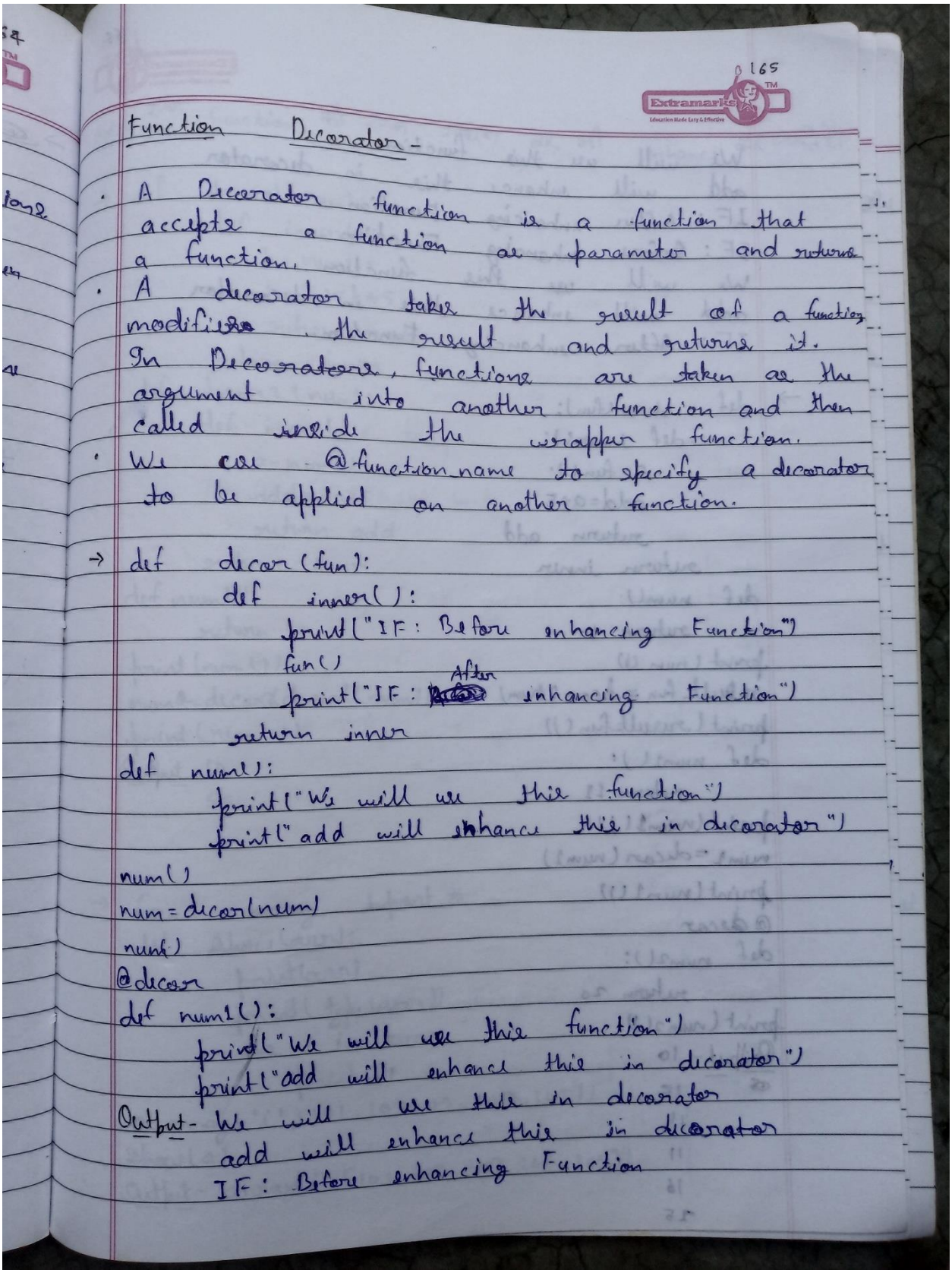
- In Python values are passed to functions by object reference.
- If object is immutable (not modifiable) then the modified value is not available outside the function.
- If object is mutable (modifiable) then the modified value is available outside the function.

Immutable Objects - Integer, Float, String, and Tuple
mutable objects - List and Dictionary

```
→ def val(lst):
      print("IFBA", lst, id(lst))
      lst=[11,22,33]
      print("IFAA", lst, id(lst))
  lst=[1,2,3]
  print("BCF", lst, id(lst))
  val(lst)
  print("ACF", lst, id(lst))
```

Output- BCF [1,2,3] 30368264
IFBA [1,2,3] 30368264
IFAA [11,22,33] 30368328
ACF [1,2,3] 30368264

Note:- When we create a new object inside function then it will be available outside function.

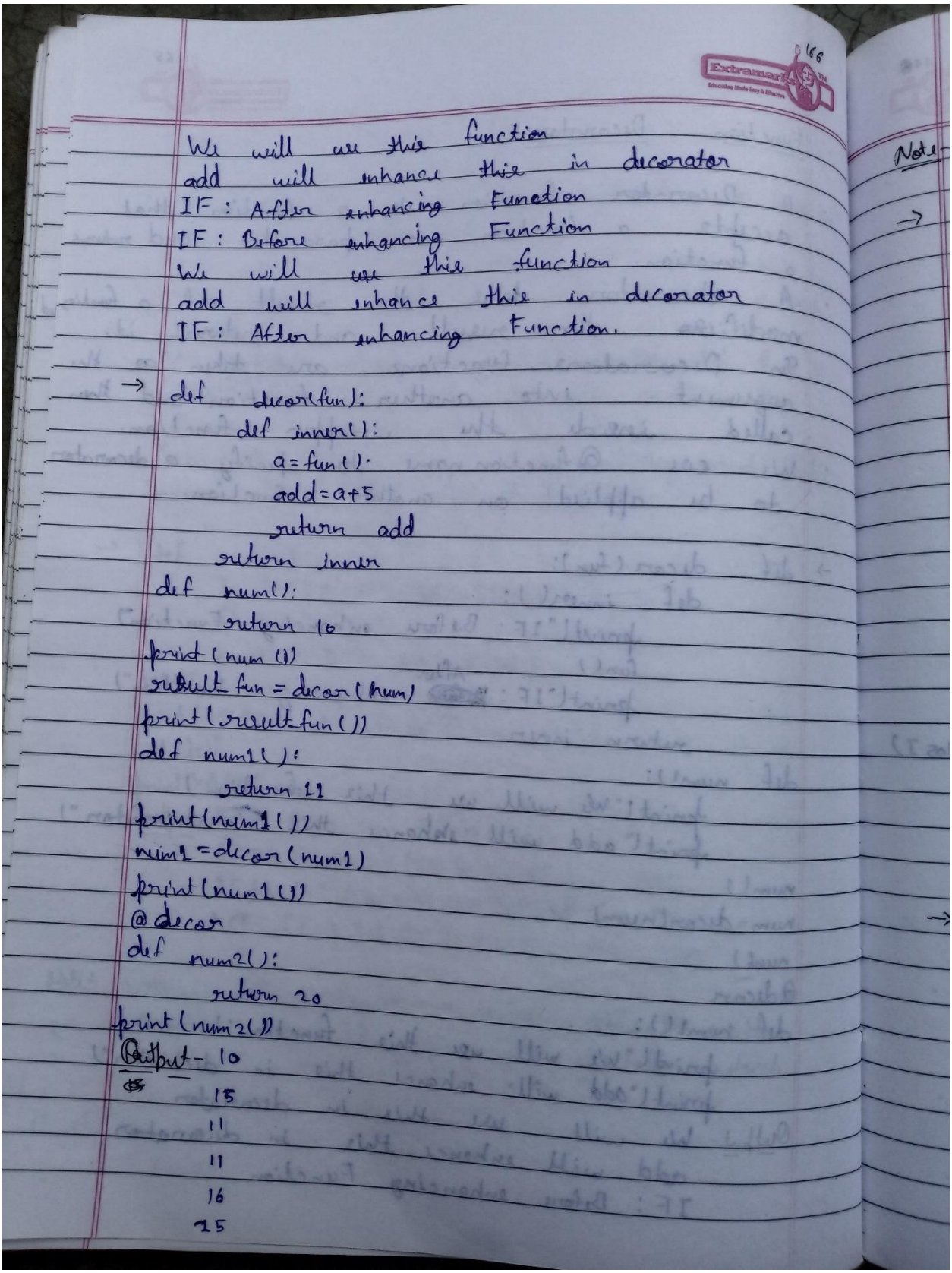


Function Decorator -

- A Decorator function is a function that accepts a function as parameter and returns a function.
- A decorator takes the result of a function, modifies the result and returns it. In Decorators, functions are taken as the argument into another function and then called inside the wrapper function.
- We use @function_name to specify a decorator to be applied on another function.

```
→ def decor(fun):
      def inner():
          print("IF: Before enhancing Function")
          fun()
          print("IF: After enhancing Function")
      return inner
def num():
    print("We will use this function")
    print("add will enhance this in decorator")
num()
num = decor(num)
num()
@decor
def num1():
    print("We will use this function")
    print("add will enhance this in decorator")
```

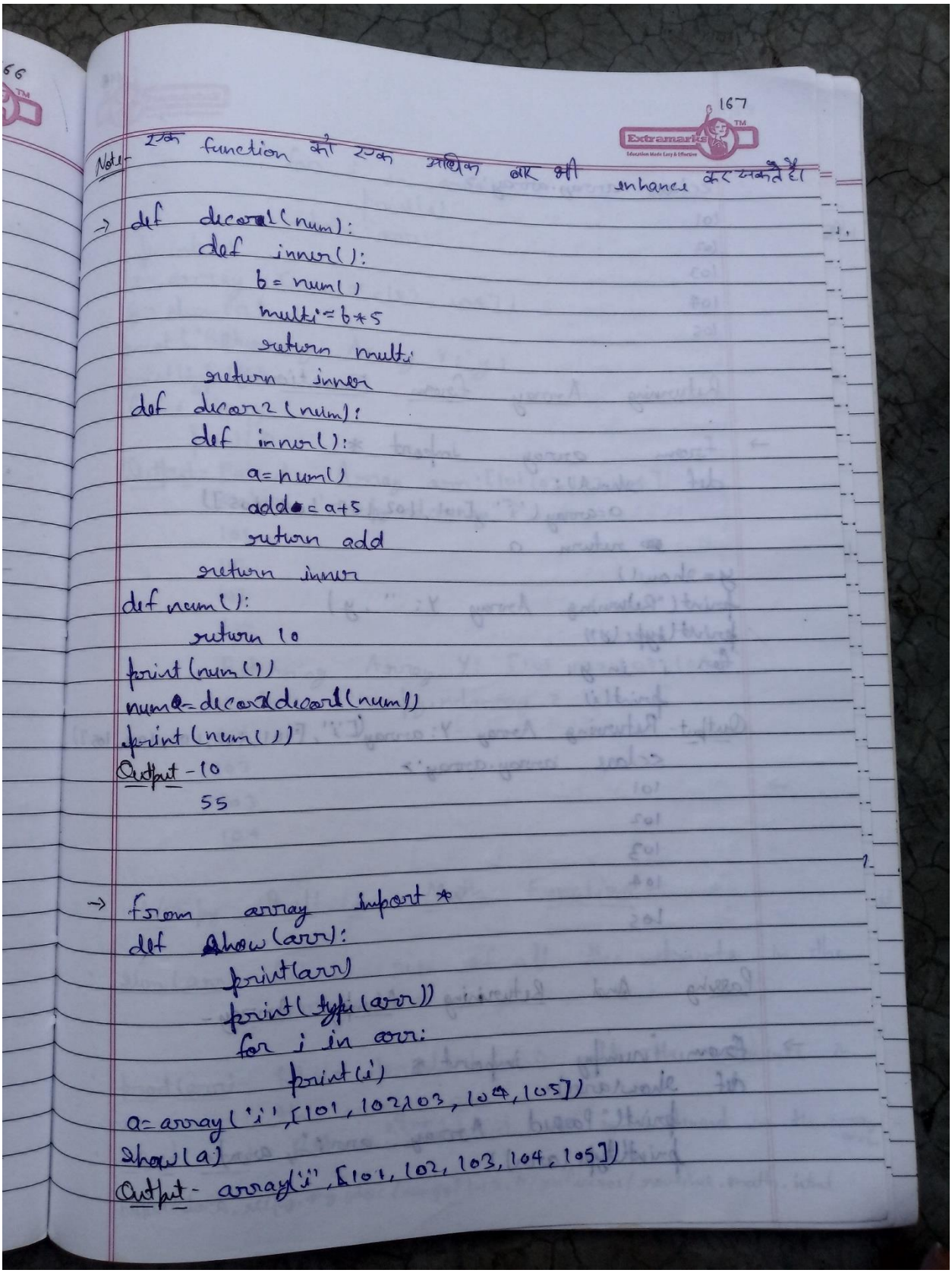
Output - We will use this in decorator
add will enhance this in decorator
IF: Before enhancing Function



We will use this function
add will inhance this in decorator
IF : After inhancing Function
IF : Before inhancing Function
We will use this function
add will inhance this in decorator
IF : After inhancing Function.

```
def decor(fun):
    def inner():
        a=fun():
        add=a+5
        return add
    return inner
def num():
    return 10
print(num())
result_fun=decor(num)
print(result_fun())
def num1():
    return 11
print(num1())
num1=decor(num1)
print(num1())
@decor
def num2():
    return 20
print(num2())
```

Output -
15
11
11
16
25



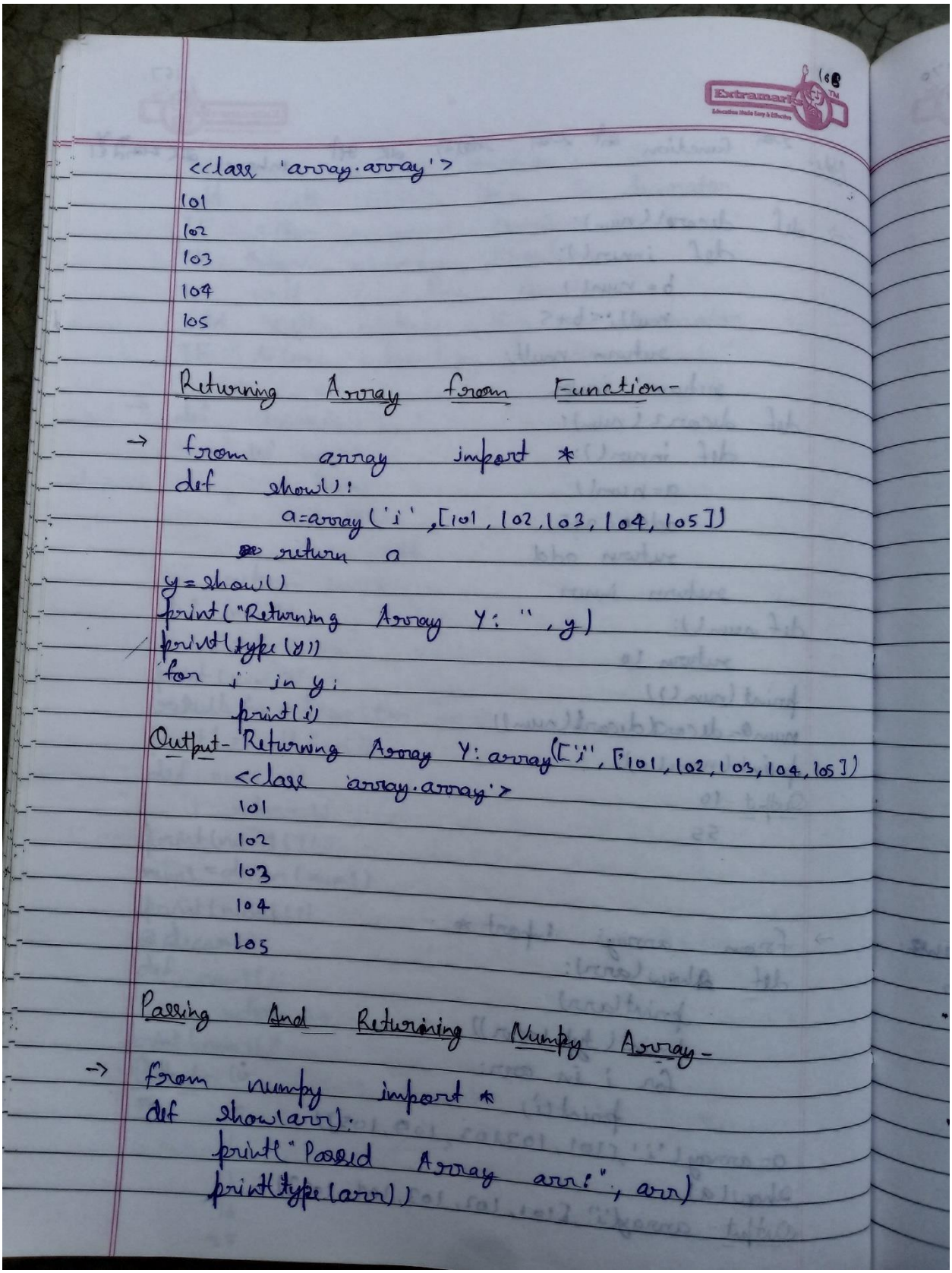
Note:- एक function को एक से अधिक बार भी enhance कर सकते हैं।

```
→ def decor1(num):
      def inner():
          b = num()
          multi = b*5
          return multi
      return inner
  def decor2(num):
      def inner():
          a = num()
          add = a+5
          return add
      return inner
  def num():
      return 10
  print(num())
  num = decor1(decor1(num))
  print(num())
```

Output - 10
55

```
→ from array import *
  def show(arr):
      print(arr)
      print(type(arr))
      for i in arr:
          print(i)
  a = array('i',[101,102,103,104,105])
  show(a)
```

Output - array('i', [101, 102, 103, 104, 105])



```
<class 'array.array'>
101
102
103
104
105
```

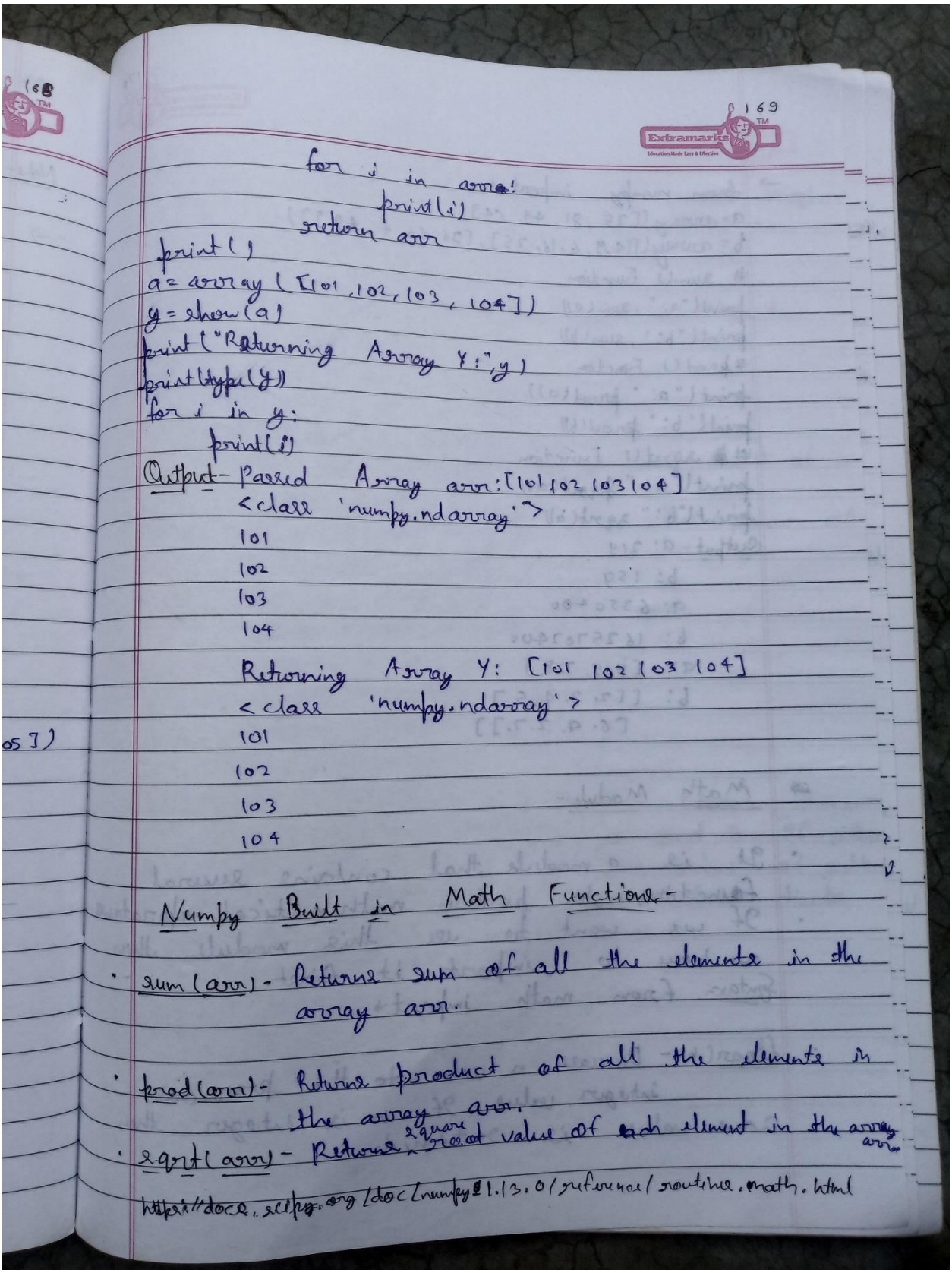
Returning Array from Function-

→
```
from array import *
def show():
    a=array('i',[101,102,103,104,105])
    return a
y=show()
print("Returning Array Y: ",y)
print(type(y))
for i in y:
    print(i)
```

Output- Returning Array Y: array('i',[101,102,103,104,105])
```
<class 'array.array'>
101
102
103
104
105
```

Passing And Returning Numpy Array-

→
```
from numpy import *
def show(arr):
    print("Passed Array arr:",arr)
    print(type(arr))
```



```
    for i in arr:
        print(i)
    return arr
print()
a = array([101,102,103,104])
y = show(a)
print("Returning Array Y:",y)
print(type(y))
for i in y:
    print(i)
```

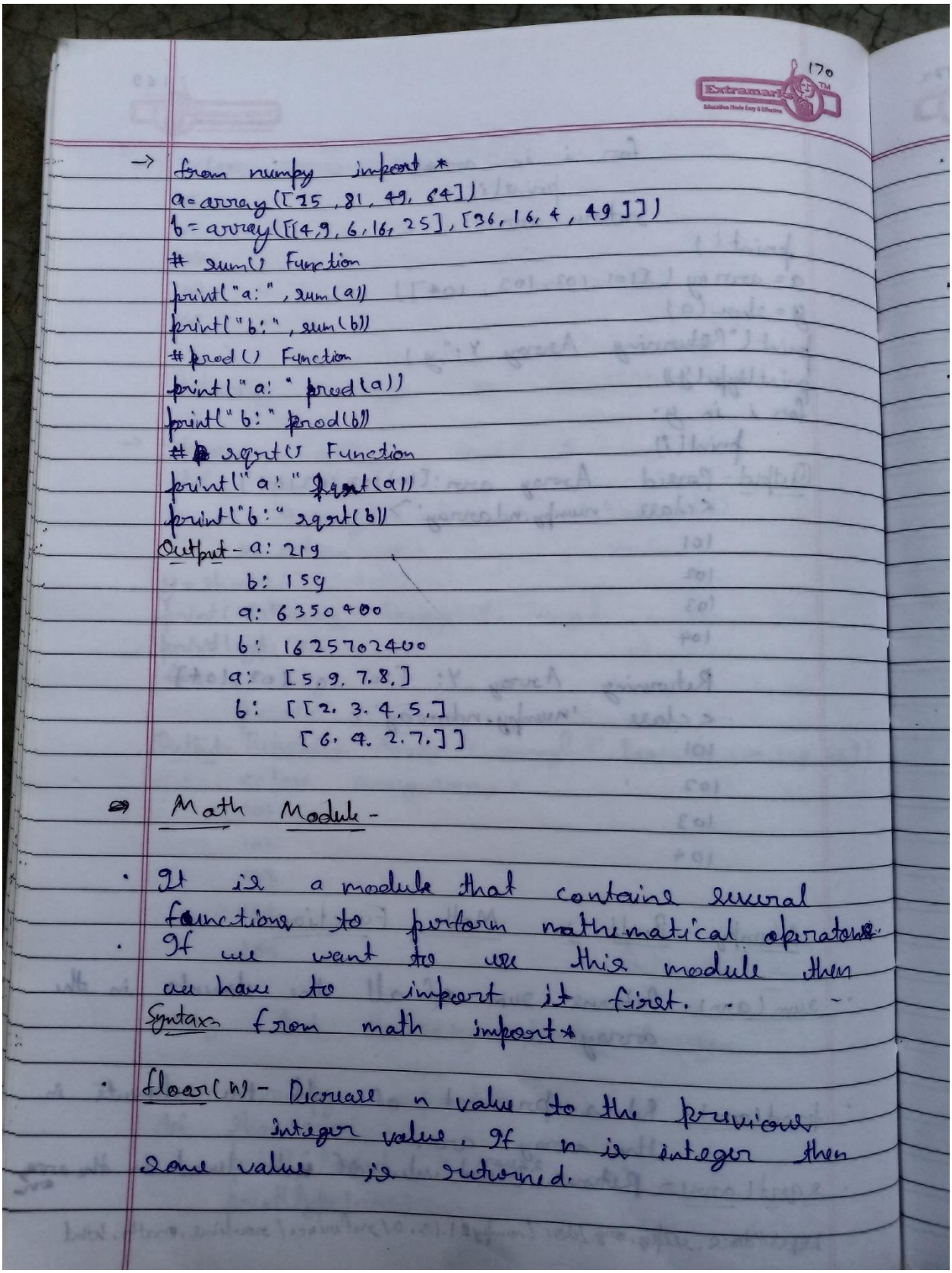
Output- Passed Array arr:[101 102 103 104]
<class 'numpy.ndarray'>
101
102
103
104

Returning Array Y: [101 102 103 104]
<class 'numpy.ndarray'>
101
102
103
104

Numpy Built in Math Functions-

- sum(arr) - Returns sum of all the elements in the array arr.
- prod(arr) - Returns product of all the elements in the array arr.
- sqrt(arr) - Returns square root value of each element in the array arr.

https://docs.scipy.org/doc/numpy-1.13.0/reference/routines.math.html



→
```
from numpy import *
a=array([25,81,49,64])
b=array([[4,9,6,16,25],[36,16,4,49]])
# sum() Function
print("a:",sum(a))
print("b:",sum(b))
#prod() Function
print("a:" prod(a))
print("b:" prod(b))
# sqrt() Function
print("a:" sqrt(a))
print("b:" sqrt(b))
```

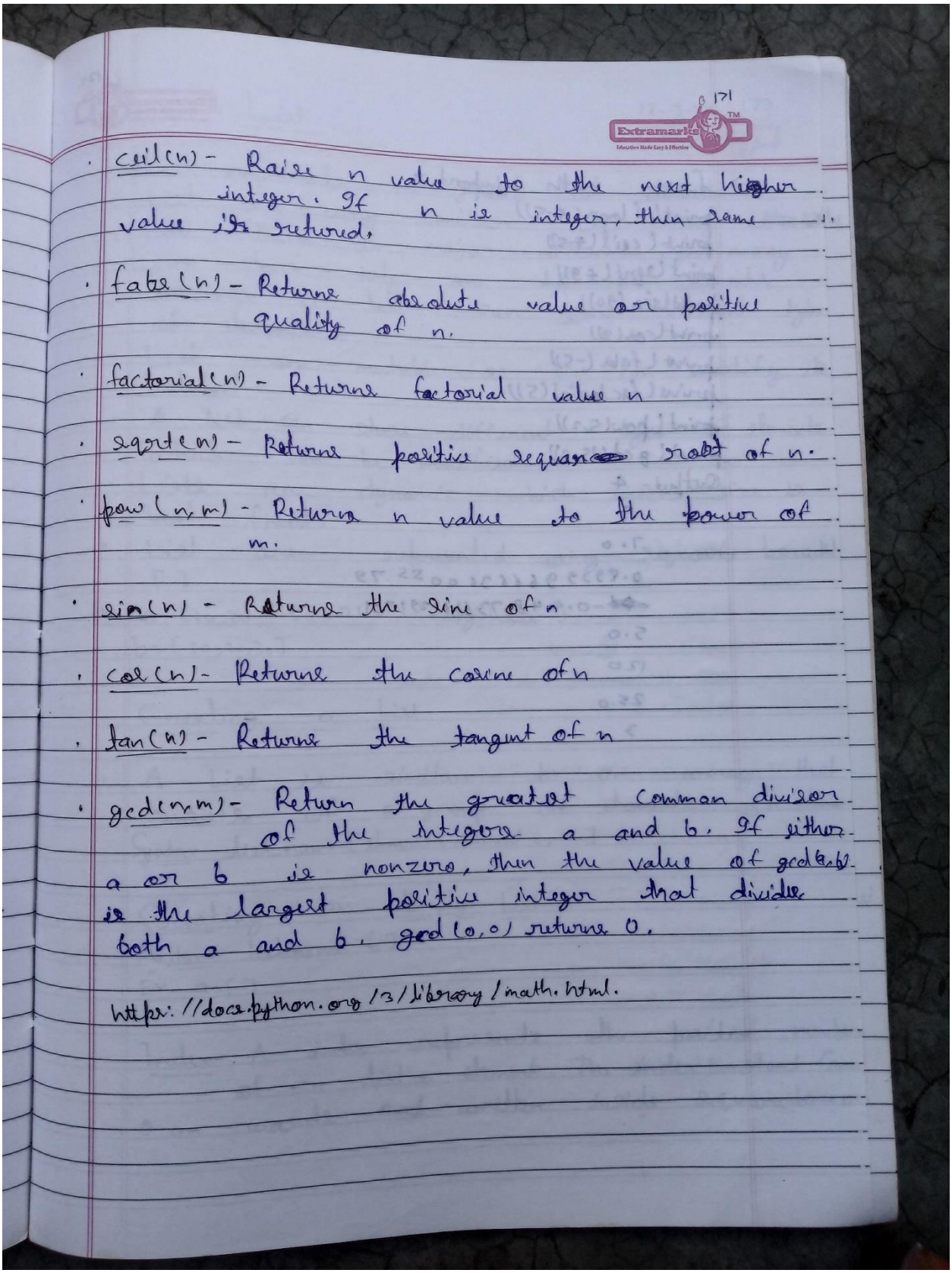
Output - a: 219
b: 159
a: 6350400
b: 16257024000
a: [5. 9. 7. 8.]
b: [[2. 3. 4. 5.]
[6. 4. 2. 7.]]

Math Module -

- It is a module that contains several functions to perform mathematical operations.
- If we want to use this module then we have to import it first.

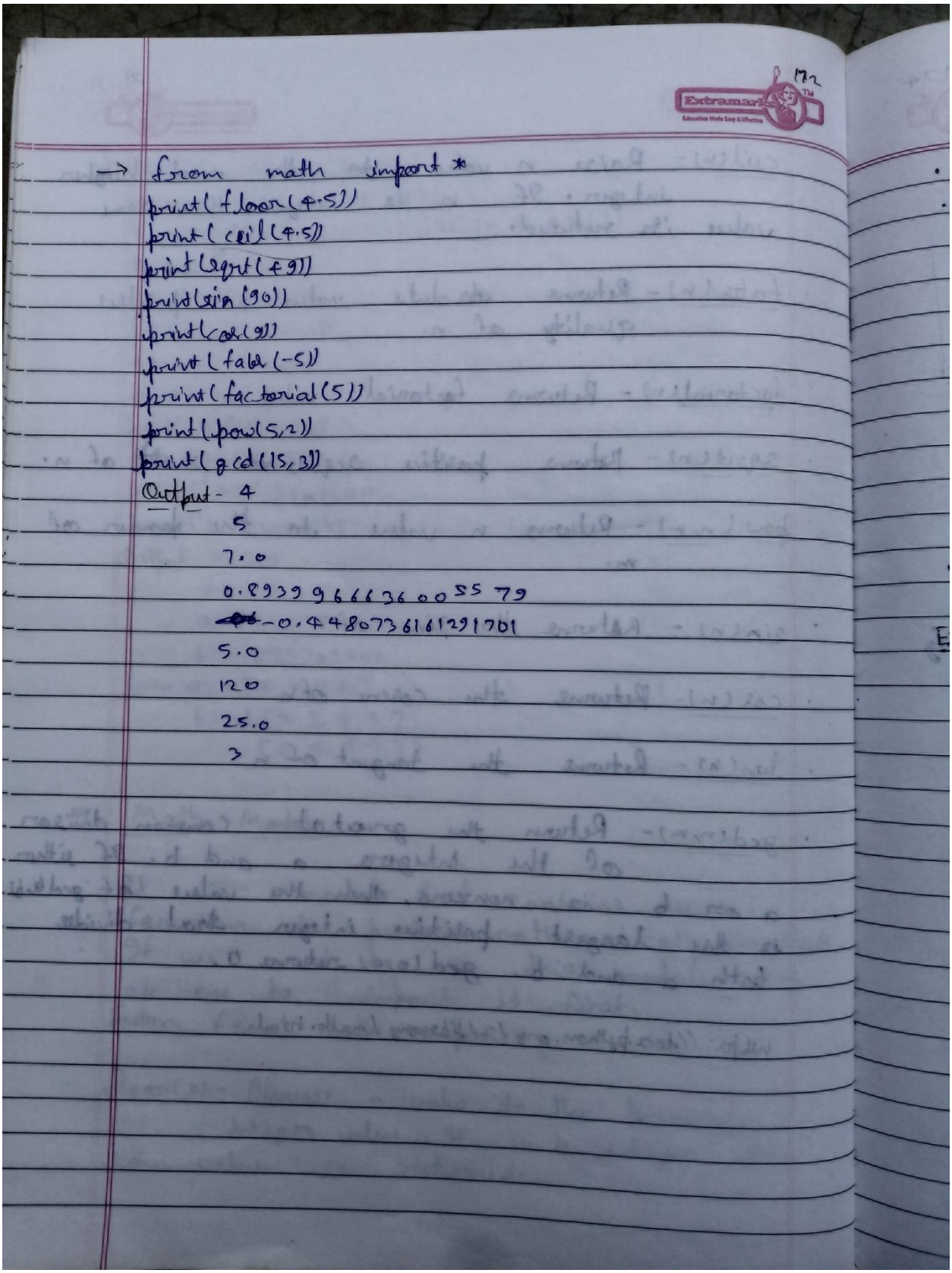
Syntax- from math import *

- floor(n) - Decrease n value to the previous integer value. If n is integer then same value is returned.



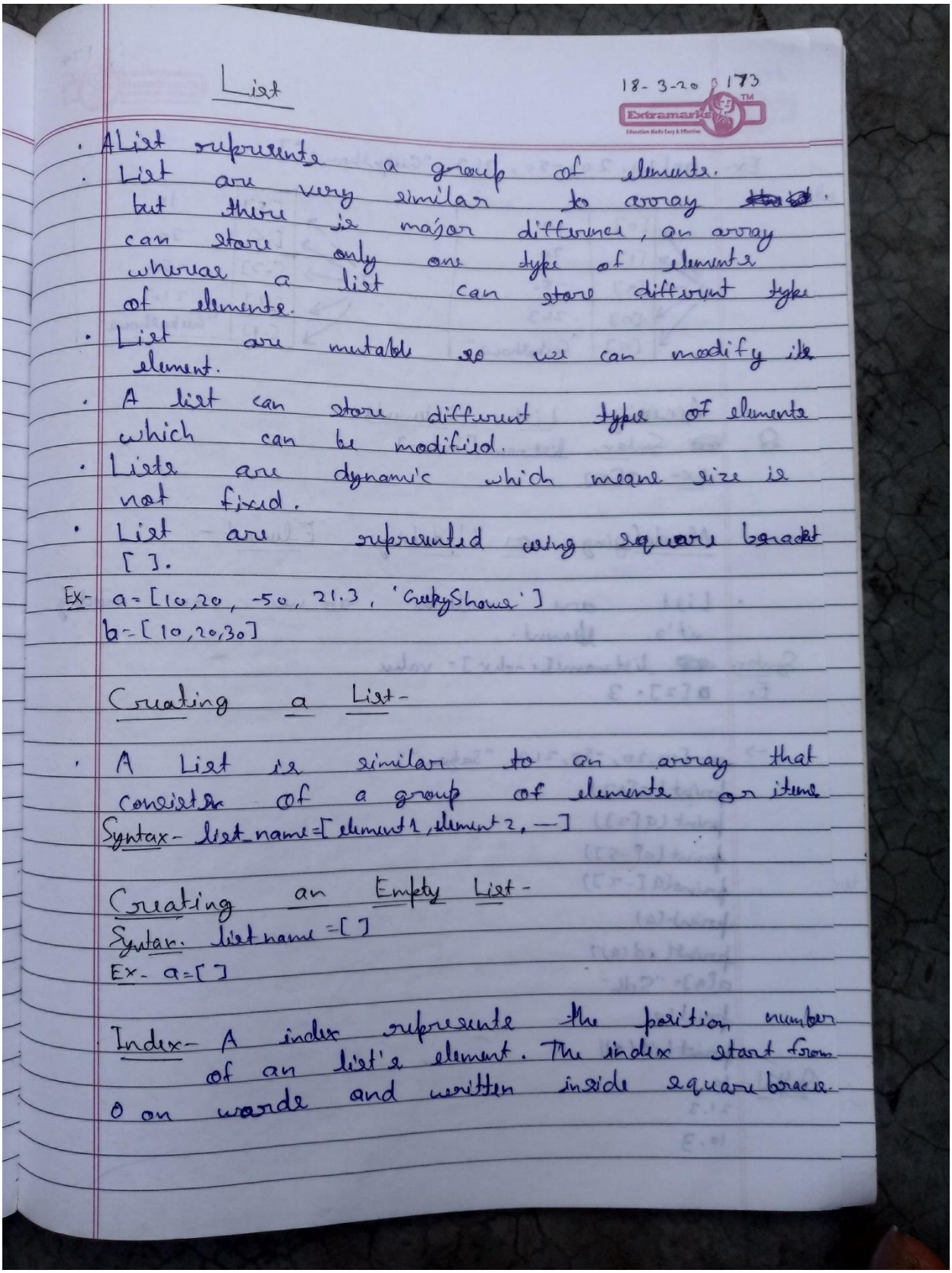
- ceil(n) - Raise n value to the next higher integer. If n is integer, then same value is returned.
- fabs(n) - Returns absolute value or positive quality of n.
- factorial(n) - Returns factorial value n
- sqrt(n) - Returns positive square root of n.
- pow(n,m) - Returns n value to the power of m.
- sin(n) - Returns the sine of n
- cos(n) - Returns the cosine of n
- tan(n) - Returns the tangent of n
- gcd(n,m) - Return the greatest common divisor of the integers a and b. If either a or b is nonzero, then the value of gcd(a,b) is the largest positive integer that divides both a and b. gcd(0,0) returns 0.

http://docs.python.org/3/library/math.html.



→ from math import *
print(floor(4.5))
print(ceil(4.5))
print(sqrt(49))
print(sin(90))
print(cos(0))
print(fabs(-5))
print(factorial(5))
print(pow(5,2))
print(gcd(15,3))

Output - 4
5
7.0
0.8939966636005579
-0.4480736161291701
5.0
120
25.0
3



List

18-3-20 173

- A List represents a group of elements.
- List are very similar to array but there is major difference, an array can store only one type of element whereas a list can store different type of elements.
- List are mutable so we can modify its element.
- A list can store different type of elements which can be modified.
- Lists are dynamic which means size is not fixed.
- List are represented using square bracket [ ].

Ex- a=[10,20,-50,21.3,'GeekyShows']
b=[10,20,30]

Creating a List-

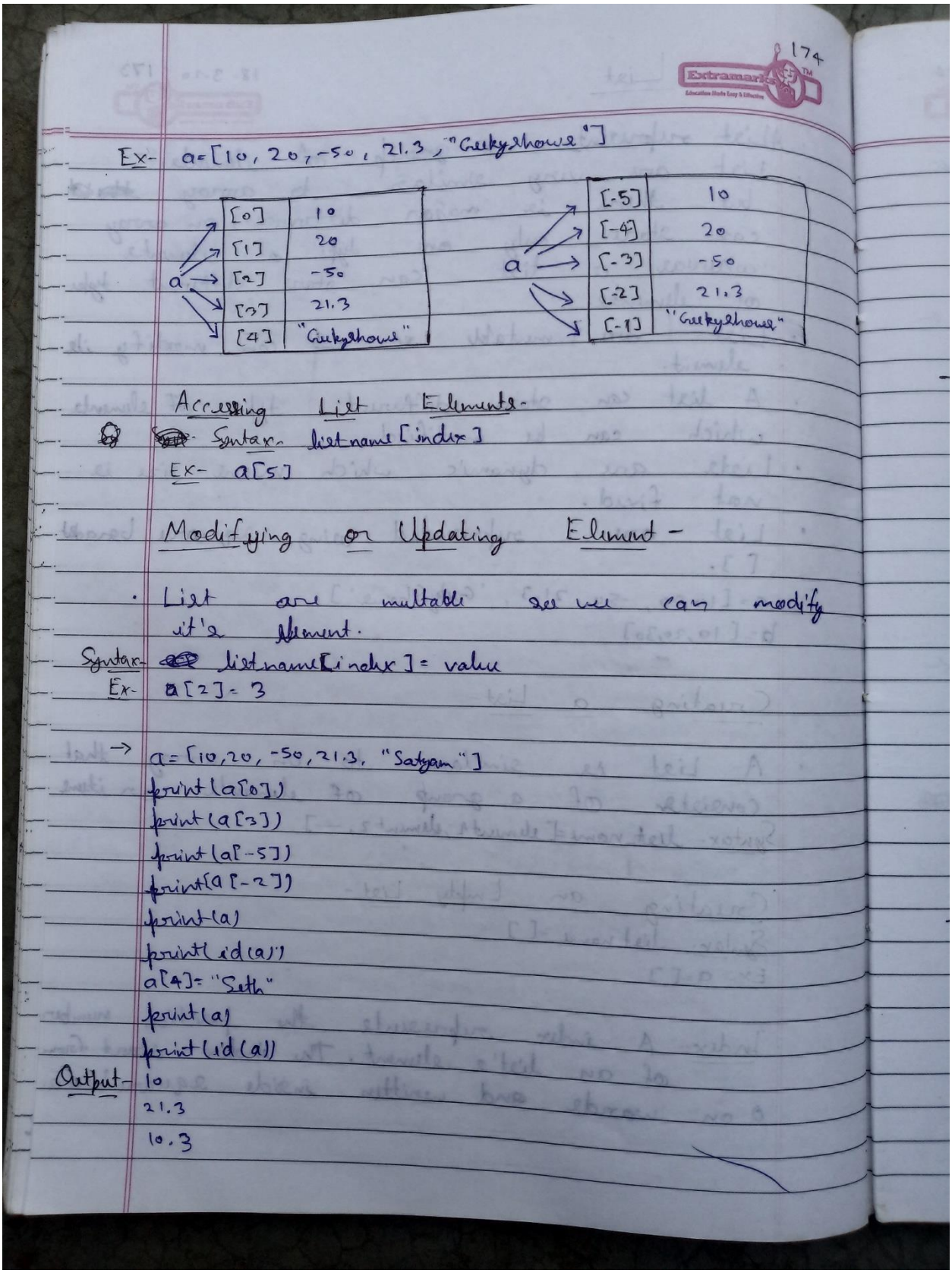
- A List is similar to an array that consists of a group of elements or items

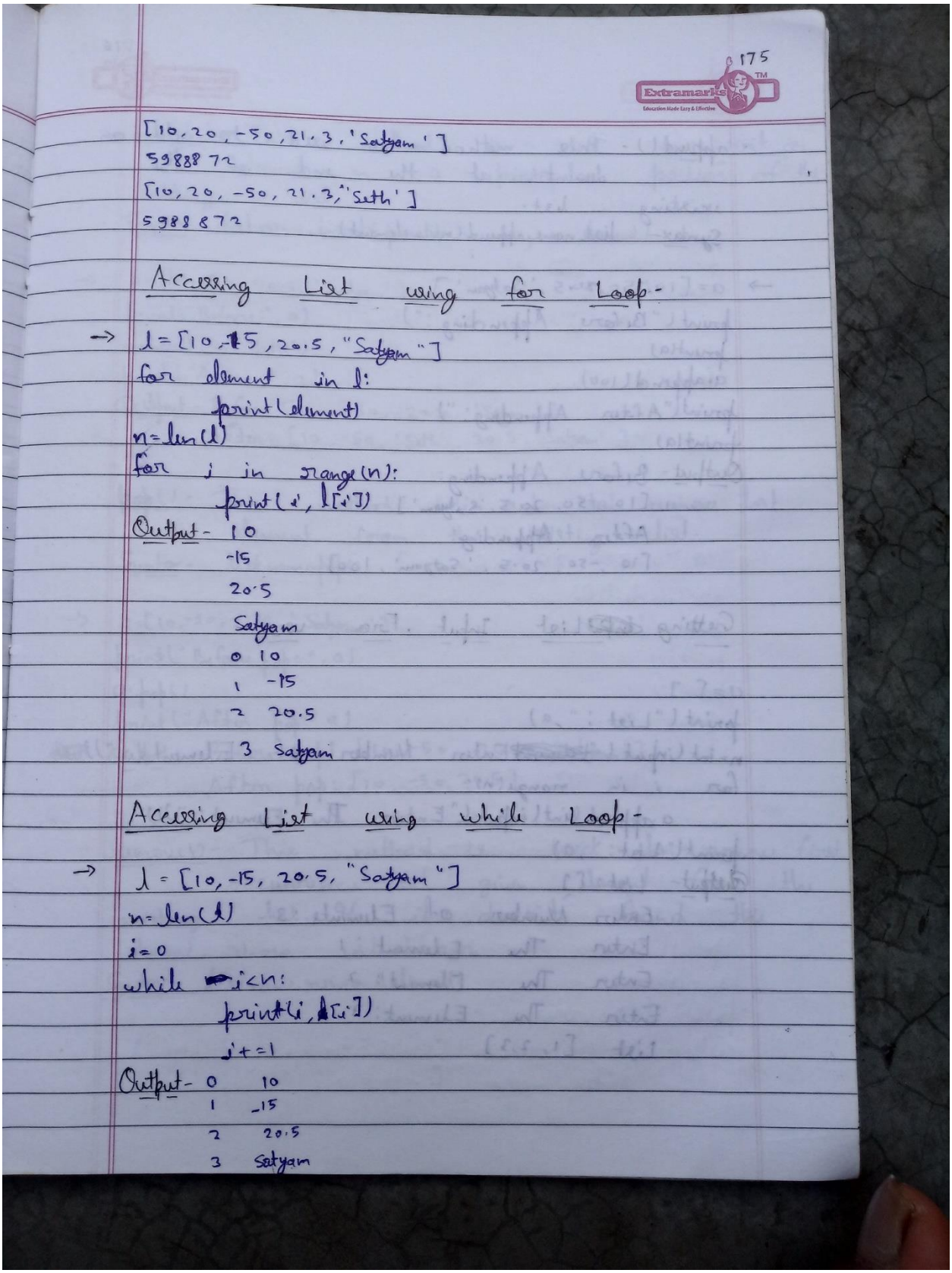
Syntax- list_name=[element1, element2, —]

Creating an Empty List-

Syntax. listname=[ ]
Ex- a=[ ]

Index- A index represents the position number of an list's element. The index start from 0 onwards and written inside square brace.


174
Extramarks
Ex- a=[10, 20, -50, 21.3, "Geekyshows"]
a
[0] 10
[1] 20
[2] -50
[3] 21.3
[4] "Geekyshows"
a
[-5] 10
[-4] 20
[-3] -50
[-2] 21.3
[-1] "Geekyshows"
Accessing List Elements-
Syntax- listname[index]
Ex- a[5]
Modifying or Updating Element -
• List are multable so we can modify it's element.
Syntax- listname[index] = value
Ex- a[2] = 3
→ a=[10,20,-50,21.3,"Satyam"]
print(a[0])
print(a[3])
print(a[-5])
print(a[-2])
print(a)
print(id(a))
a[4]="Seth"
print(a)
print(id(a))
Output- 10
21.3
10.3



175

[10, 20, -50, 21.3, 'Satyam']
59888 72
[10, 20, -50, 21.3, 'Seth']
5988872

Accessing List using for Loop -

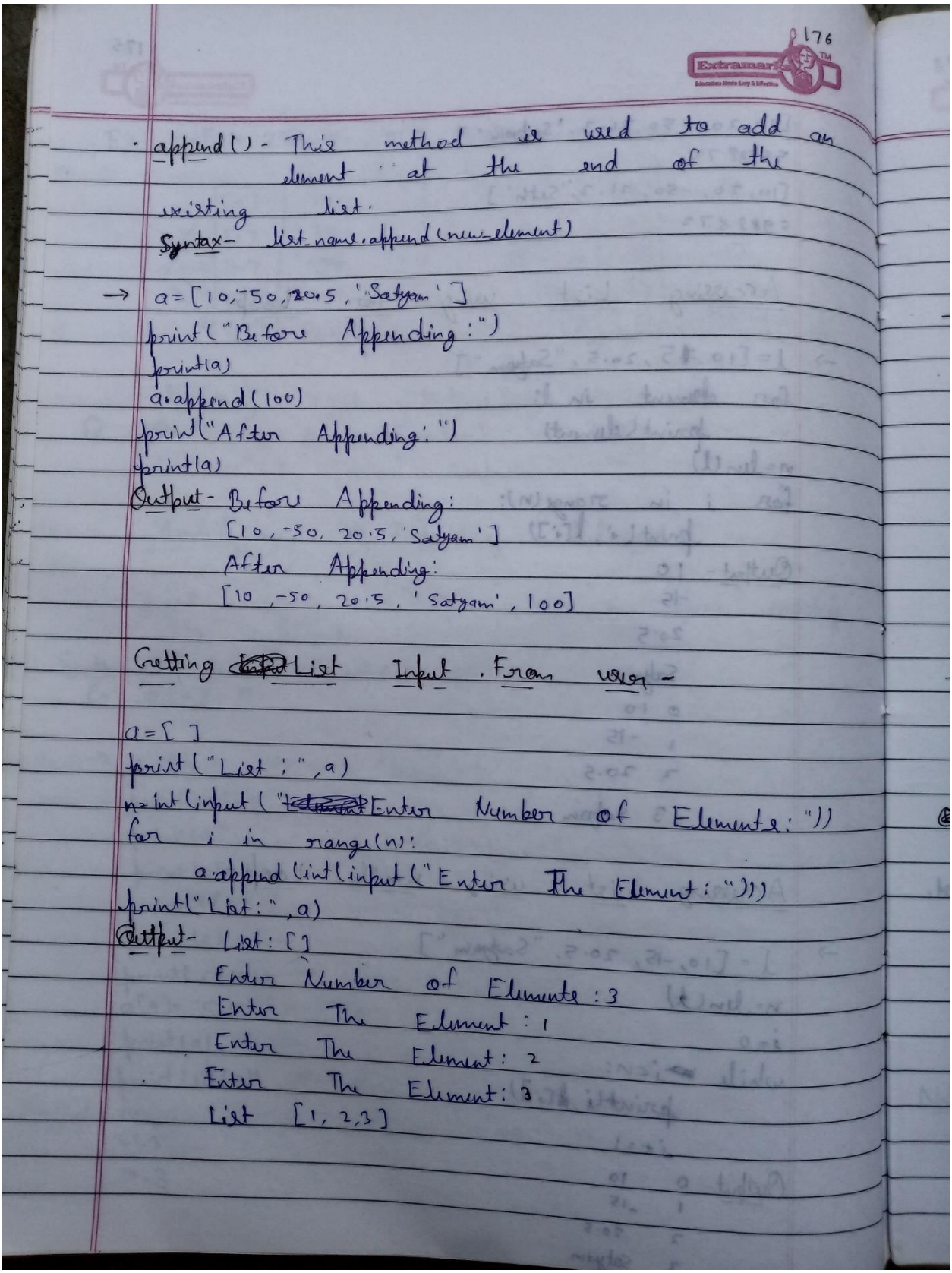
→
```
l = [10, -15, 20.5, "Satyam"]
for element in l:
    print(element)
n = len(l)
for i in range(n):
    print(i, l[i])
```

Output -
```
10
-15
20.5
Satyam
0 10
1 -15
2 20.5
3 Satyam
```

Accessing List using while Loop -

→
```
l = [10, -15, 20.5, "Satyam"]
n = len(l)
i = 0
while i<n:
    print(i, l[i])
    i += 1
```

Output -
```
0 10
1 -15
2 20.5
3 Satyam
```



176

- append() - This method is used to add an element at the end of the existing list.

Syntax- list_name.append(new_element)

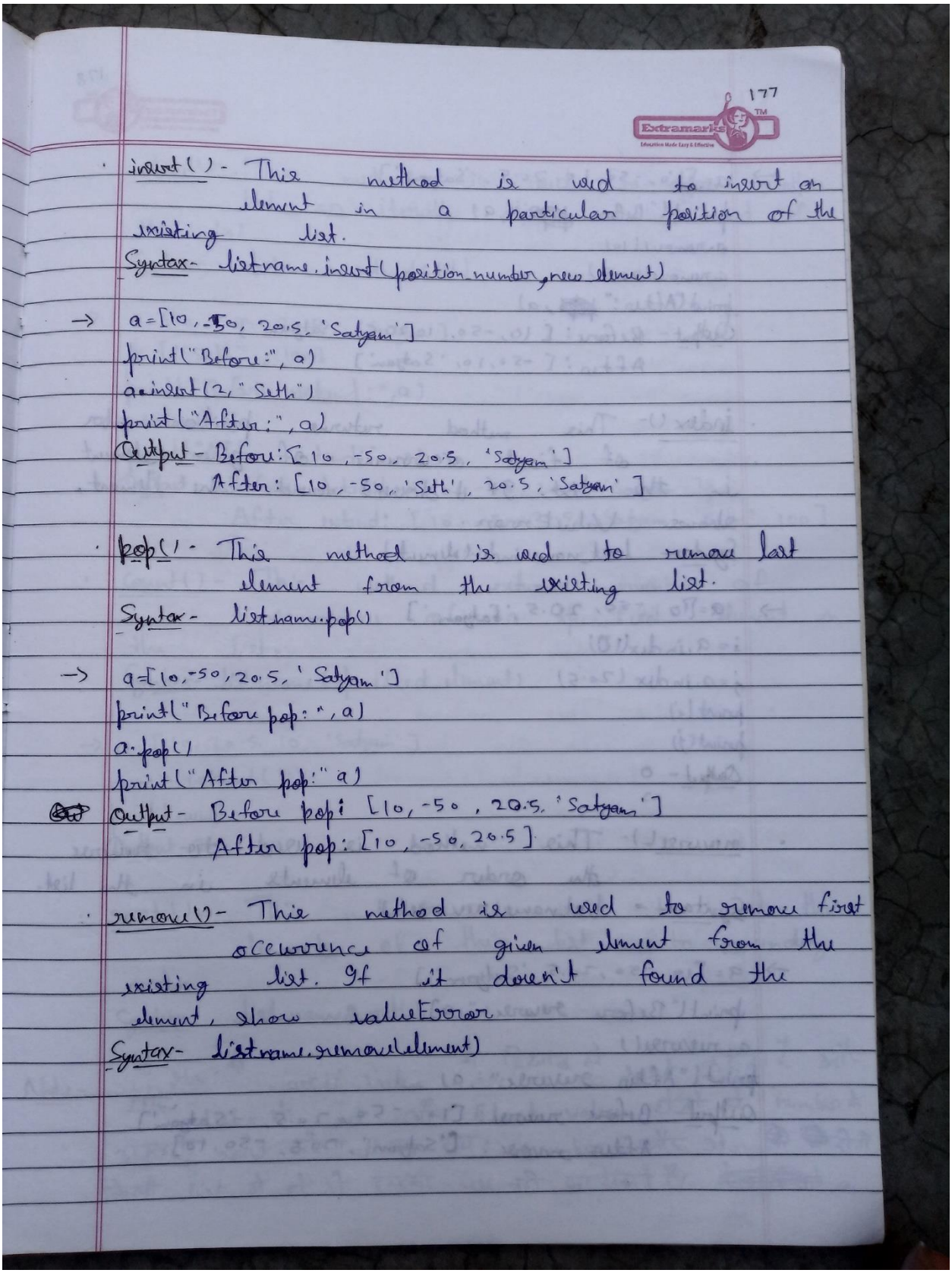
→ a=[10,-50,20.5,'Satyam']
print("Before Appending:")
print(a)
a.append(100)
print("After Appending:")
print(a)

Output- Before Appending:
[10,-50,20.5,'Satyam']
After Appending:
[10,-50,20.5,'Satyam',100]

Getting List Input From user -

a=[]
print("List:",a)
n=int(input("Enter Number of Elements:"))
for i in range(n):
    a.append(int(input("Enter The Element:")))
print("List:",a)

Output- List:[]
Enter Number of Elements:3
Enter The Element:1
Enter The Element:2
Enter The Element:3
List [1,2,3]



177

- **insert()** - This method is used to insert an element in a particular position of the existing list.

Syntax- listname.insert(position_number, new element)

→
```
a=[10, -50, 20.5, 'Satyam']
print("Before:", a)
a.insert(2, "Seth")
print("After:", a)
```
Output - Before: [10, -50, 20.5, 'Satyam']
After: [10, -50, 'Seth', 20.5, 'Satyam']

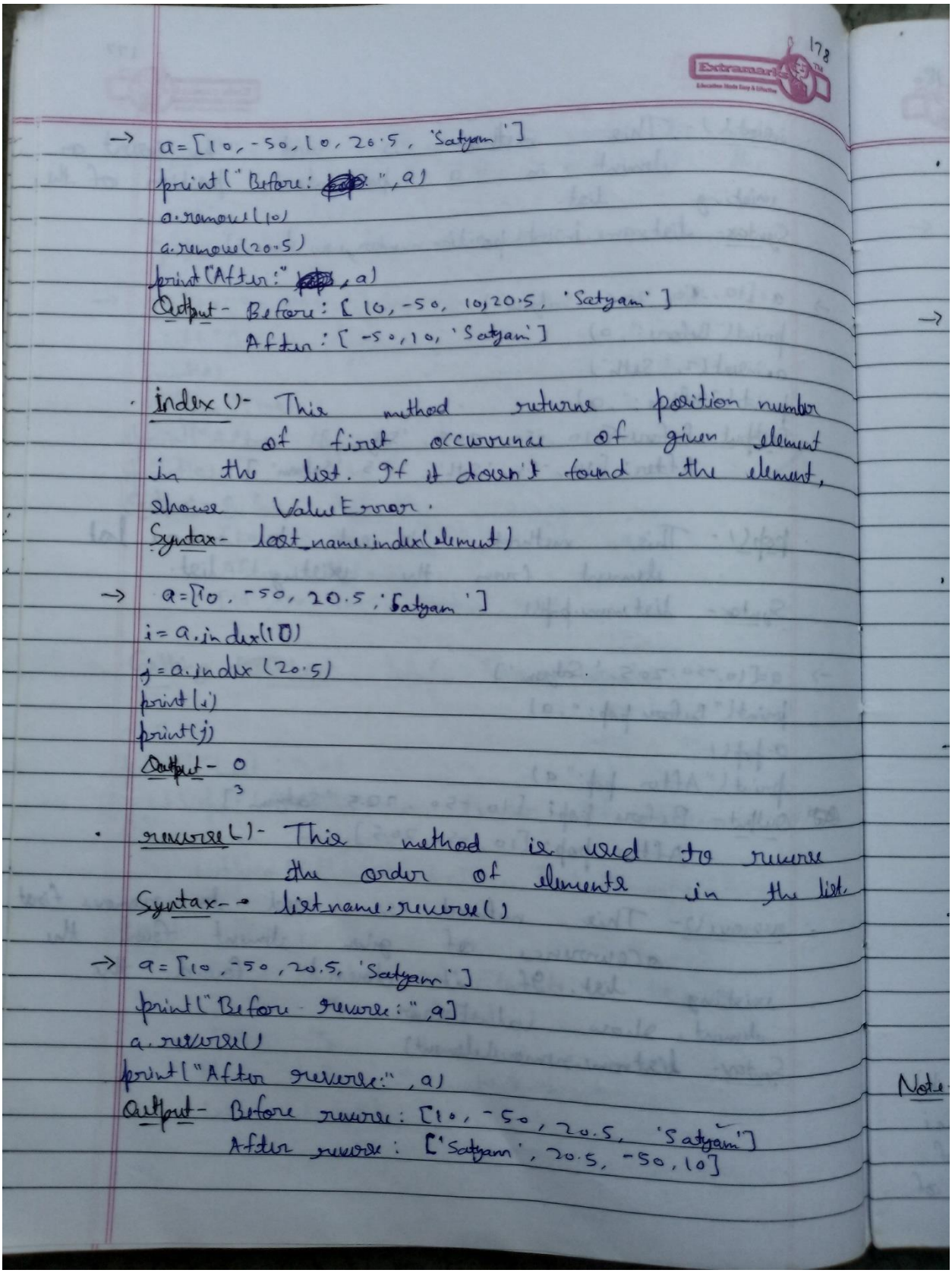
- **pop()** - This method is used to remove last element from the existing list.

Syntax- listname.pop()

→
```
a=[10, -50, 20.5, 'Satyam']
print("Before pop:", a)
a.pop()
print("After pop:" a)
```
Output - Before pop: [10, -50, 20.5, 'Satyam']
After pop: [10, -50, 20.5]

- **remove()** - This method is used to remove first occurrence of given element from the existing list. If it doesn't found the element, show valueError

Syntax- listname.remove(element)



```
a=[10,-50,10,20.5,'Satyam']
print("Before: ",a)
a.remove(10)
a.remove(20.5)
print("After:",a)
```

Output - Before: [10, -50, 10, 20.5, 'Satyam']
After: [-50, 10, 'Satyam']

- **index()** - This method returns position number of first occurrence of given element in the list. If it doesn't found the element, shows ValueError.

Syntax - list_name.index(element)

```
a=[10,-50,20.5,'Satyam']
i=a.index(10)
j=a.index(20.5)
print(i)
print(j)
```

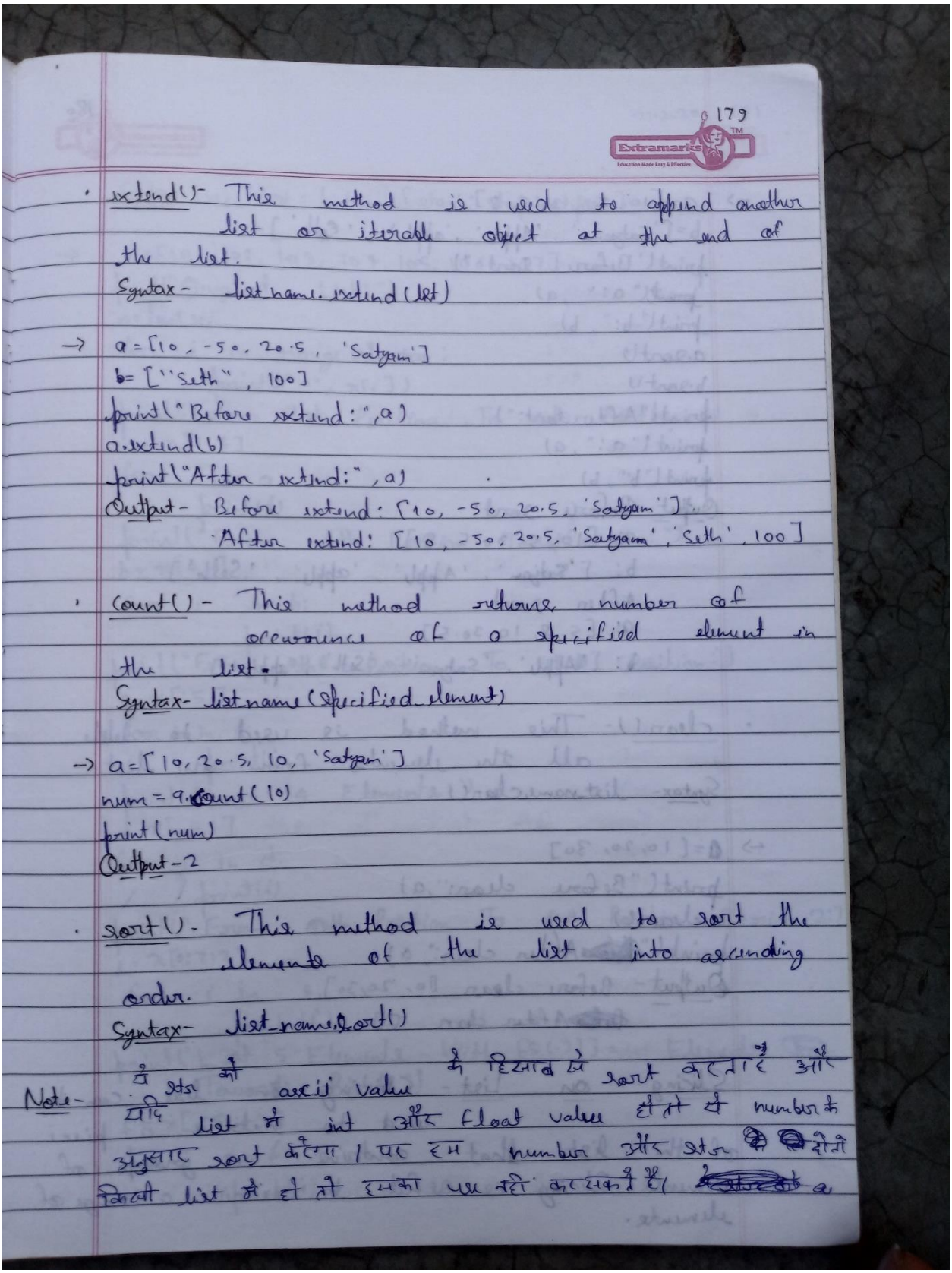
Output - 0
3

- **reverse()** - This method is used to reverse the order of elements in the list.

Syntax - list_name.reverse()

```
a=[10,-50,20.5,'Satyam']
print("Before reverse:",a)
a.reverse()
print("After reverse:",a)
```

Output - Before reverse: [10, -50, 20.5, 'Satyam']
After reverse: ['Satyam', 20.5, -50, 10]



179

- extend() - This method is used to append another list or iterable object at the end of the list.

Syntax - list_name.extend(list)

→ a = [10, -50, 20.5, 'Satyam']
b = ["Seth", 100]
print("Before extend:", a)
a.extend(b)
print("After extend:", a)
Output - Before extend: [10, -50, 20.5, 'Satyam']
After extend: [10, -50, 20.5, 'Satyam', 'Seth', 100]

- count() - This method returns number of occurance of a specified element in the list.

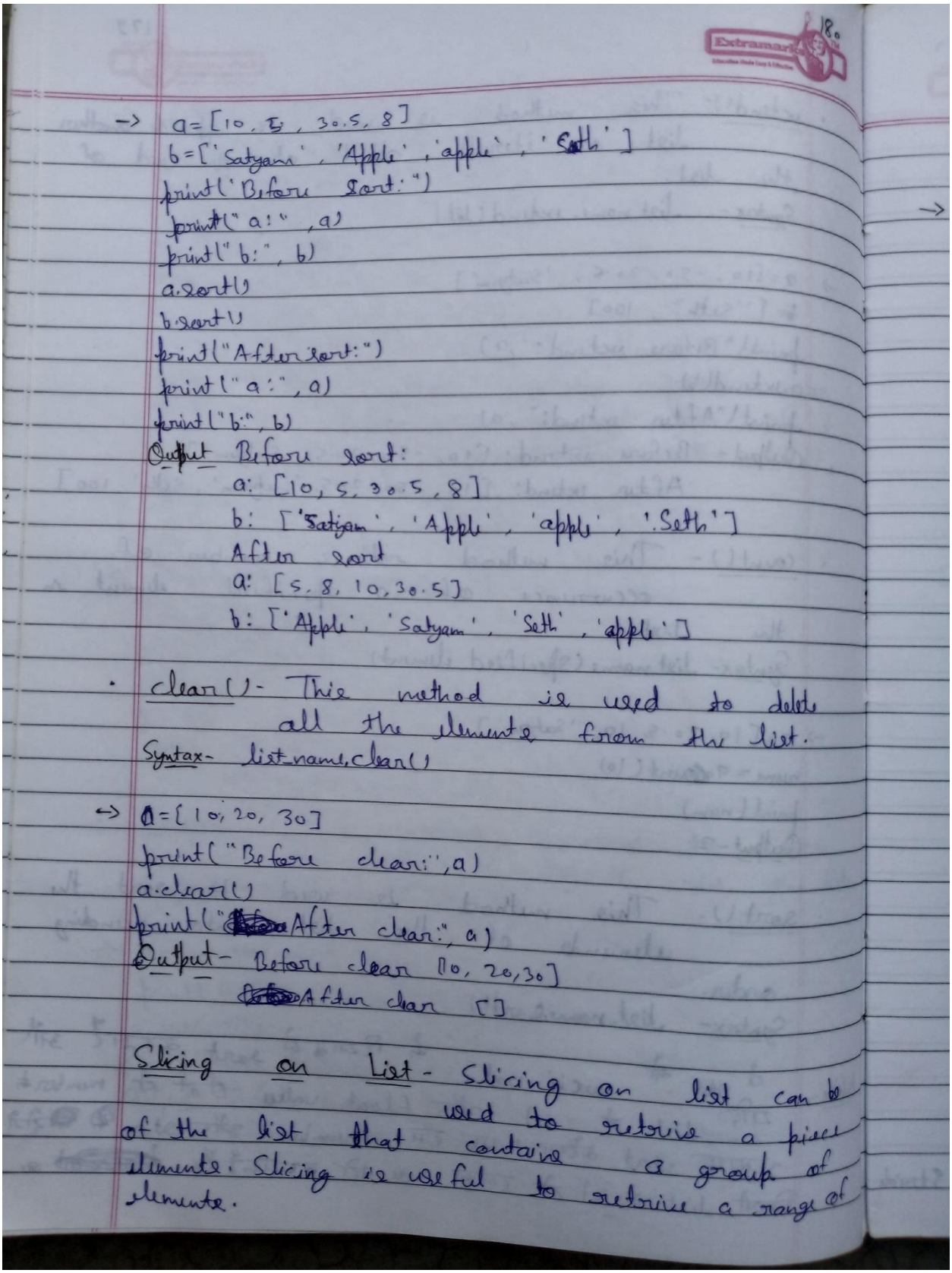
Syntax - list_name(Specified_element)

→ a = [10, 20.5, 10, 'Satyam']
num = a.count(10)
print(num)
Output - 2

- sort() - This method is used to sort the elements of the list into ascending order.

Syntax - list_name.sort()

Note - यदि list में str को ascii value के हिसाब से sort करेगा और int और float values होंगे तो number के अनुसार sort करेगा। पर हम number और str को दोनों किसी list में हों तो इसको use नहीं कर सकते हैं।



```
a=[10, 5, 30.5, 8]
b=['Satyam', 'Apple', 'apple', 'Seth']
print('Before sort:')
print("a:", a)
print("b:", b)
a.sort()
b.sort()
print("After sort:")
print("a:", a)
print("b:", b)
```

Output Before sort:

a: [10, 5, 30.5, 8]

b: ['Satyam', 'Apple', 'apple', 'Seth']

After sort

a: [5, 8, 10, 30.5]

b: ['Apple', 'Satyam', 'Seth', 'apple']

- clear() - This method is used to delete all the elements from the list.

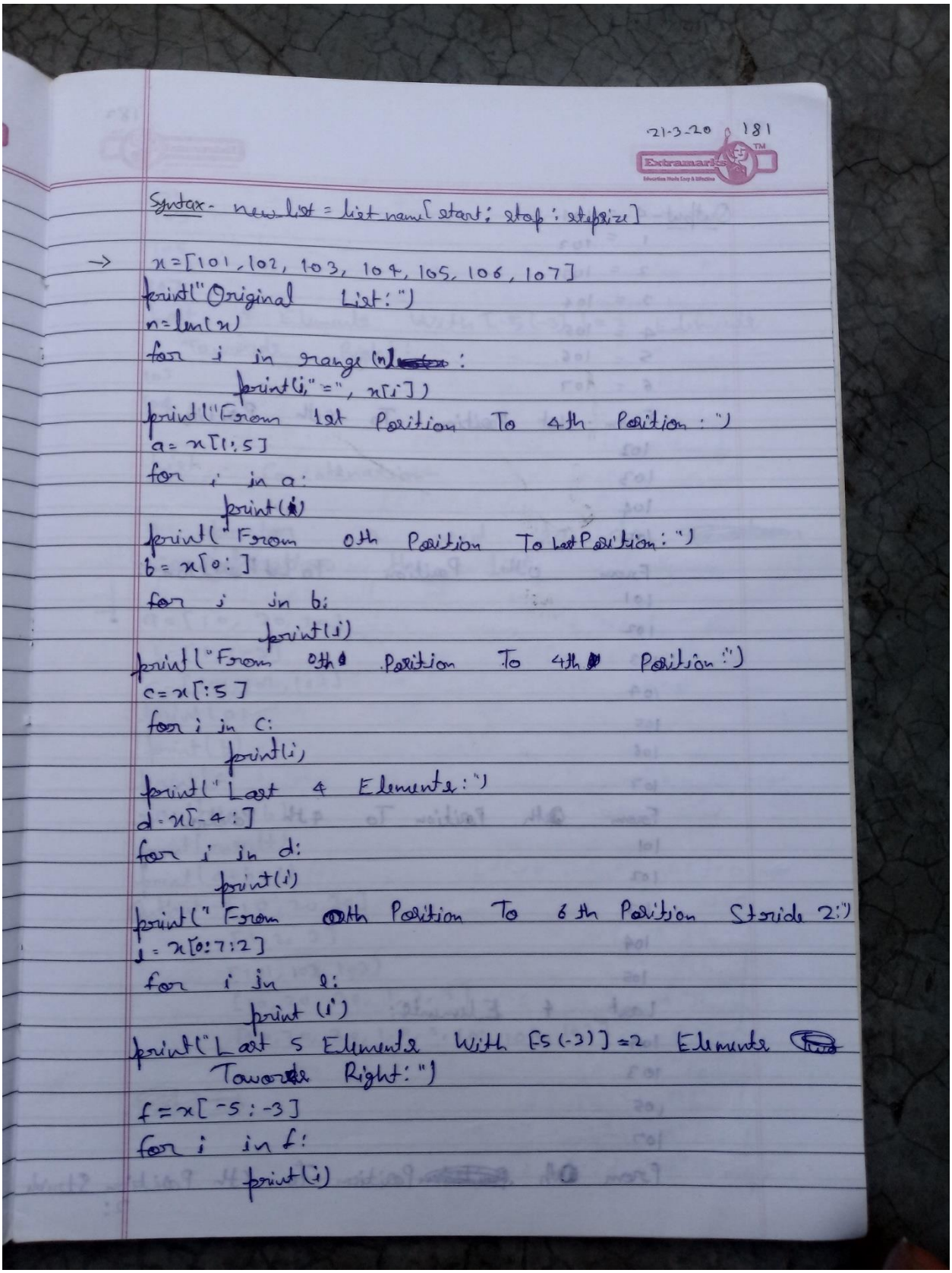
Syntax- list_name.clear()

```
a=[10, 20, 30]
print("Before clear:", a)
a.clear()
print("After clear:", a)
```

Output- Before clear [10, 20, 30]

After clear []

Slicing on List - Slicing on list can be used to retrive a piece of the list that contains a group of elements. Slicing is useful to retrive a range of elements.

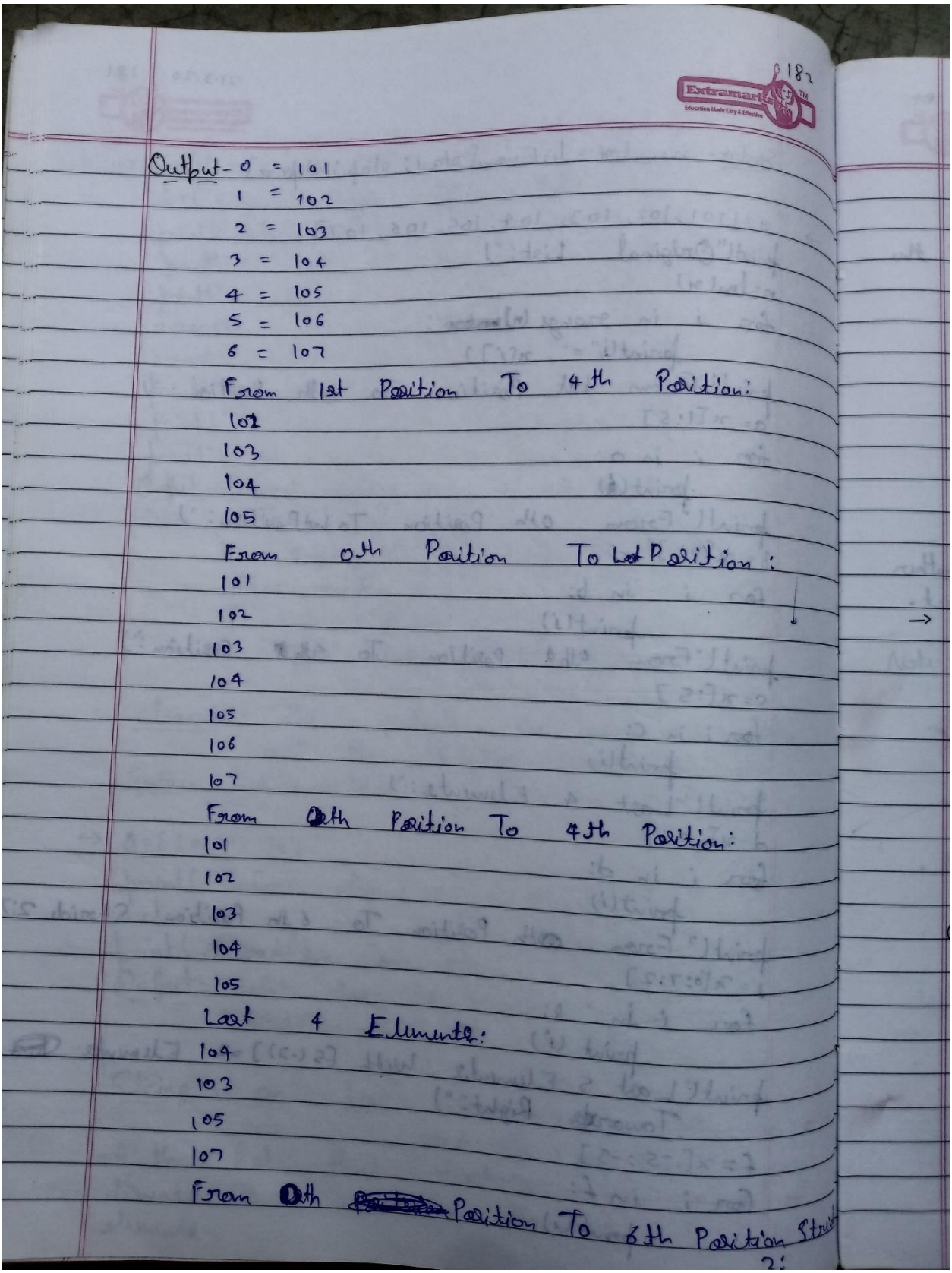


21-3-20 181

Extramarks

Syntax- new_list = list_name[start : stop : stepsize]

→
```
x=[101,102,103,104,105,106,107]
print("Original List:")
n=len(x)
for i in range(n):
    print(i,"=",x[i])
print("From 1st Position To 4th Position:")
a=x[1:5]
for i in a:
    print(i)
print("From 0th Position To Last Position:")
b=x[0:]
for i in b:
    print(i)
print("From 0th Position To 4th Position:")
c=x[:5]
for i in c:
    print(i)
print("Last 4 Elements:")
d=x[-4:]
for i in d:
    print(i)
print("From 0th Position To 6th Position Stride 2:")
e=x[0:7:2]
for i in e:
    print(i)
print("Last 5 Elements With [5:(-3)]=2 Elements Towards Right:")
f=x[-5:-3]
for i in f:
    print(i)
```



182

Output- 0 = 101
1 = 102
2 = 103
3 = 104
4 = 105
5 = 106
6 = 107

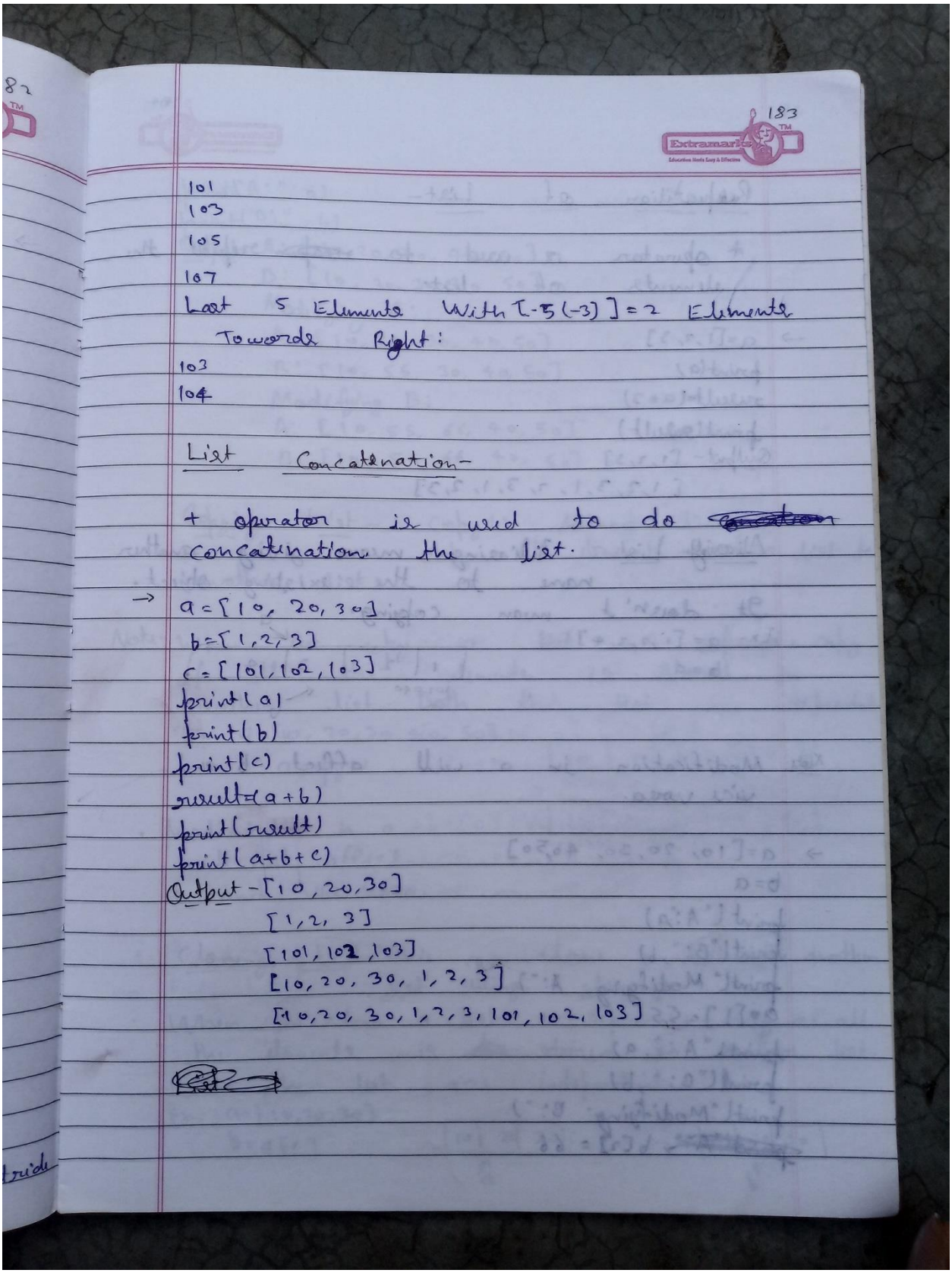
From 1st Position To 4th Position:
102
103
104
105

From 0th Position To Last Position:
101
102
103
104
105
106
107

From 0th Position To 4th Position:
101
102
103
104
105

Last 4 Elements:
104
103
105
107

From 0th [redacted] Position To 6th Position Stride 2:



101
103
105
107
Last 5 Elements With [-5 (-3)] = 2 Elements Towards Right:
103
104

List Concatenation-

+ operator is used to do concatenation the list.

→
```
a=[10, 20, 30]
b=[1,2,3]
c=[101,102,103]
print(a)
print(b)
print(c)
result=(a+b)
print(result)
print(a+b+c)
```
Output - [10, 20, 30]
[1, 2, 3]
[101, 102, 103]
[10, 20, 30, 1, 2, 3]
[10, 20, 30, 1, 2, 3, 101, 102, 103]

184

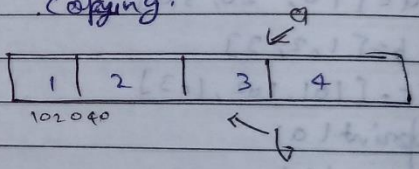
Repetition of List -

* operator is used to repeat the elements of list.

→
```
a=[1,2,3]
print(a)
result=(a*3)
print(result)
```
Output - [1,2,3]
[1,2,3,1,2,3,1,2,3]

Aliasing List - Aliasing means giving another name to the existing object. It doesn't mean copying.

Ex- a = [1,2,3,4]
b = a



Note

Note - Modification in a will affect b and vice versa.

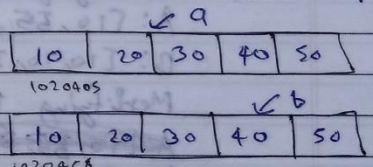
→
```
a=[10, 20, 30, 40, 50]
b=a
print("A:",a)
print("B:",b)
print("Modifying A:")
a[1]=55
print("A:",a)
print("B:",b)
print("Modifying B:")
b[2]=66
```

print("A:", a)
print("B:", b)
Output- A: [10, 20, 30, 40, 50]
B: [10, 20, 30, 40, 50]
Modifying A:
A: [10, 55, 30, 40, 50]
B: [10, 55, 30, 40, 50]
Modifying B:
A: [10, 55, 66, 40, 50]
B: [10, 55, 66, 40, 50]

Copying List - copy() Method is used to copy all elements of a list to another list.

Note- When we copy a list a separate copy of all the elements is stored in another list. Both the list are independent.

Ex- a=[10, 20, 30, 40, 50]
b= a.copy()

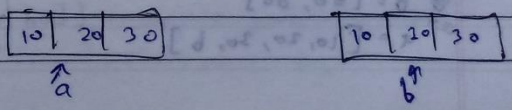


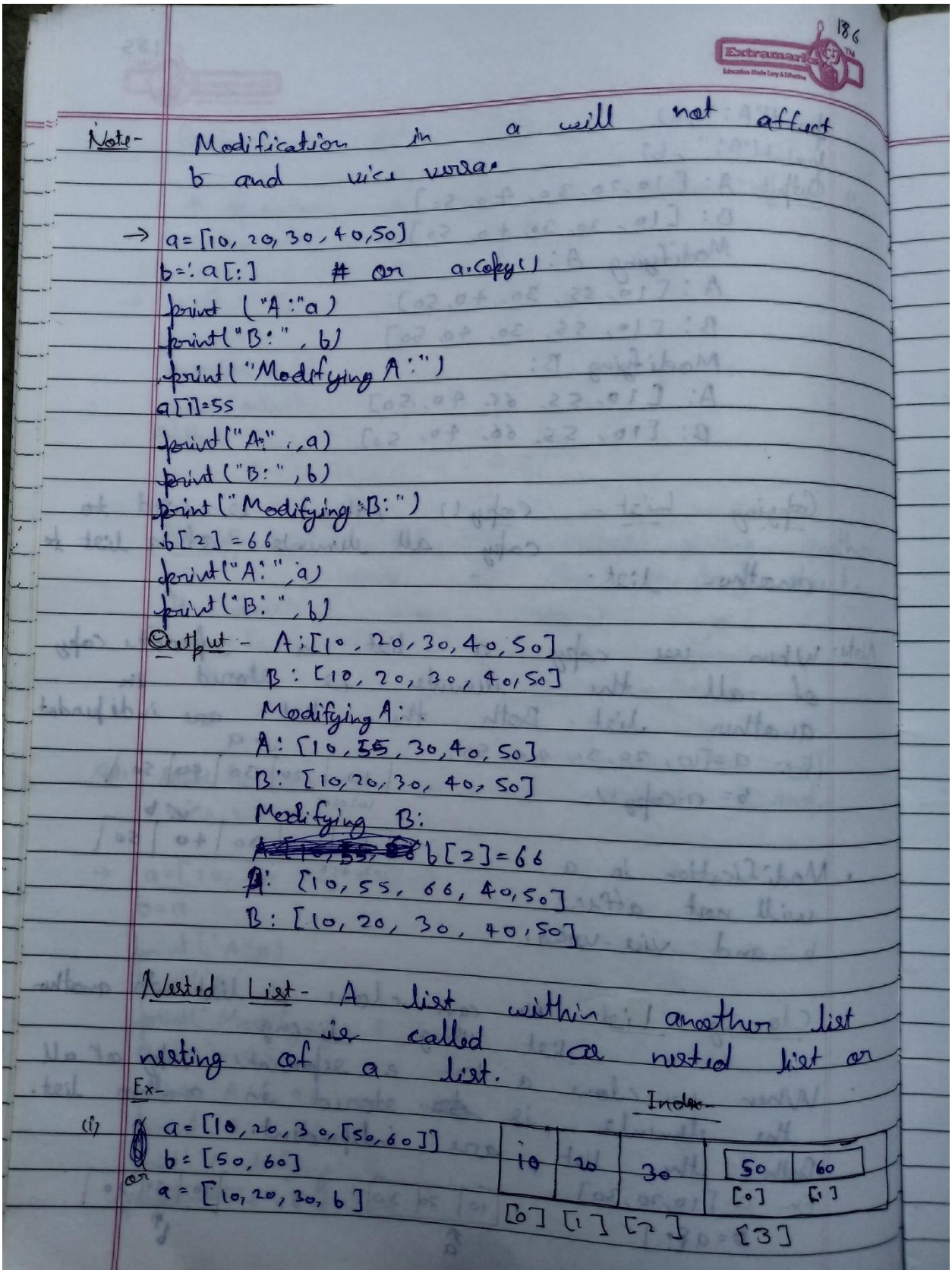
- Modification in a will not affect b and vice-versa.

Cloning List - We can clone a list into another list using slicing

- When we clone a list a separate copy of all the elements is stored in another list. Both the list are independent.

Ex- a=[10,20,30]
b=a[:]





Note- Modification in a will not affect b and vice versa.

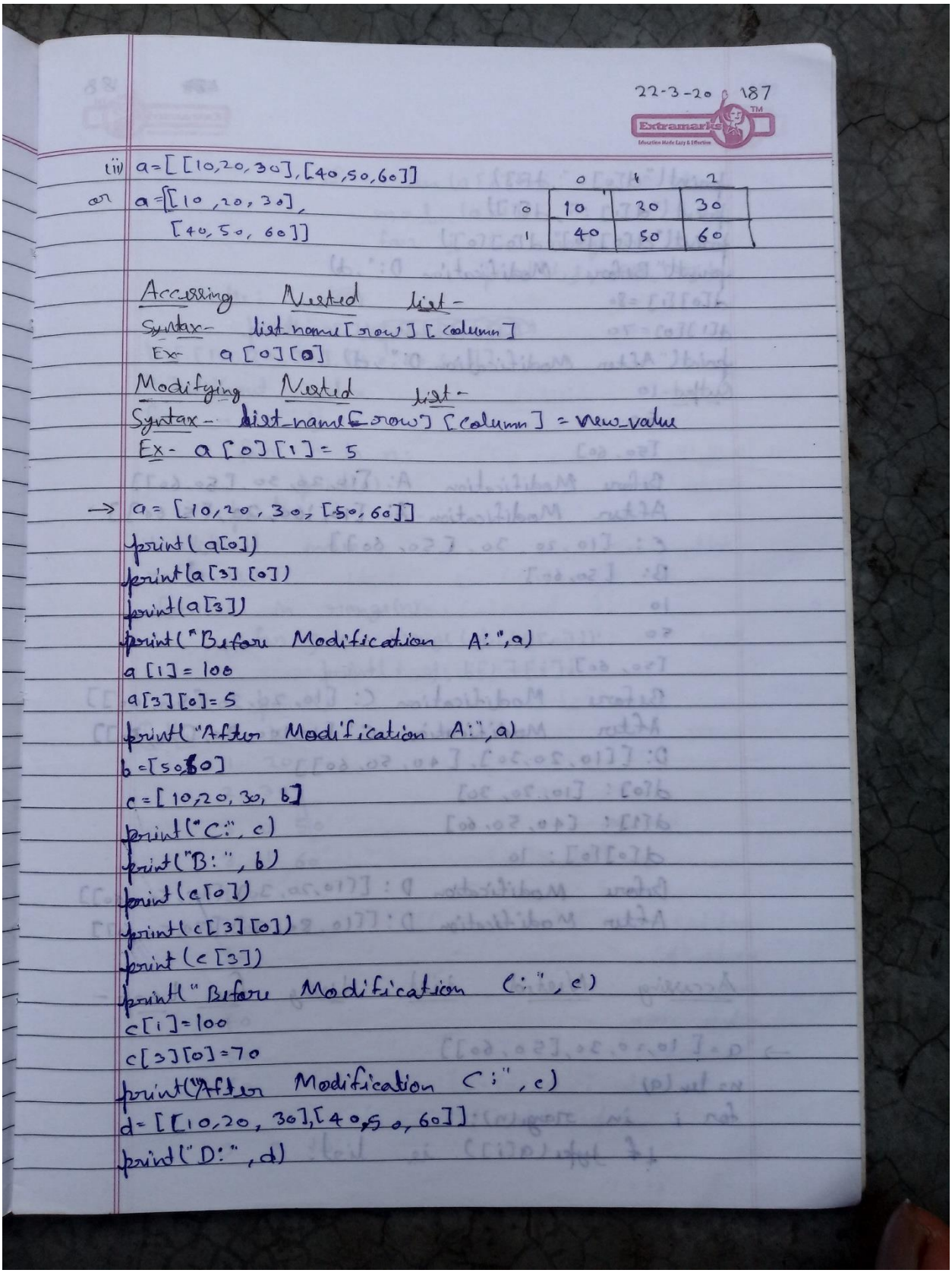
→
```
a=[10,20,30,40,50]
b=a[:]      # or a.copy()
print("A:",a)
print("B:",b)
print("Modifying A:")
a[1]=55
print("A:",a)
print("B:",b)
print("Modifying B:")
b[2]=66
print("A:",a)
print("B:",b)
```

Output - A:[10,20,30,40,50]
B:[10,20,30,40,50]
Modifying A:
A:[10,55,30,40,50]
B:[10,20,30,40,50]
Modifying B:
b[2]=66
A:[10,55,66,40,50]
B:[10,20,30,40,50]

Nested List - A list within another list is called as nested list or nesting of a list.

Ex-

(i)
```
a=[10,20,30,[50,60]]
b=[50,60]
or
a=[10,20,30,b]
```



22-3-20 187

(iii) a=[[10,20,30],[40,50,60]]

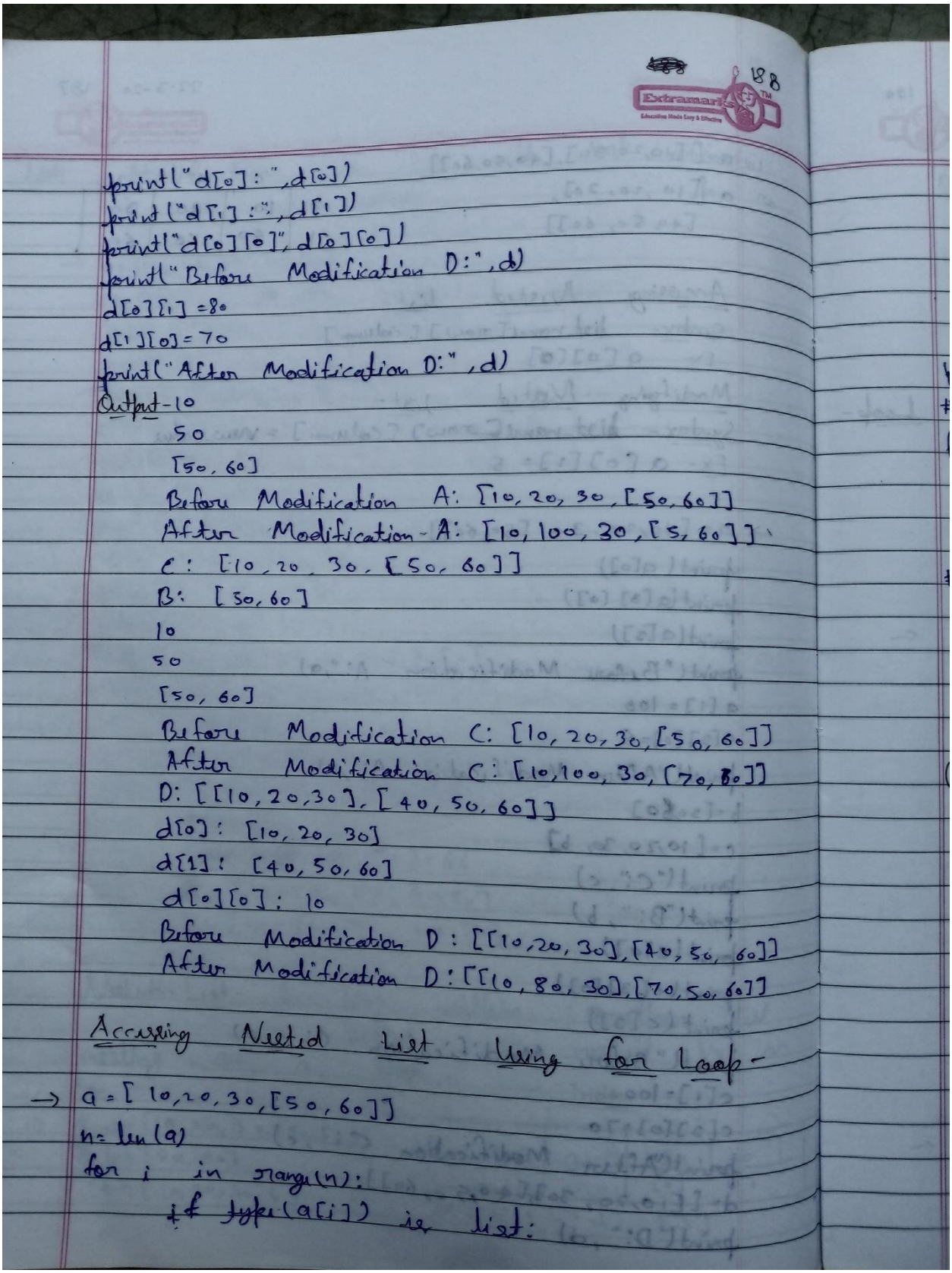
or a=[[10,20,30],
[40,50,60]]

	0	1	2
0	10	20	30
1	40	50	60

Accessing Nested list -
Syntax - list_name[row][column]
Ex - a[0][0]

Modifying Nested list -
Syntax - list_name[row][column] = new_value
Ex - a[0][1] = 5

→
```
a = [10,20,30,[50,60]]
print(a[0])
print(a[3][0])
print(a[3])
print("Before Modification A:",a)
a[1] = 100
a[3][0]=5
print("After Modification A:",a)
b=[50,60]
c=[10,20,30,b]
print("C:",c)
print("B:",b)
print(c[0])
print(c[3][0])
print(c[3])
print("Before Modification C:",c)
c[1]=100
c[3][0]=70
print("After Modification C:",c)
d=[[10,20,30],[40,50,60]]
print("D:",d)
```

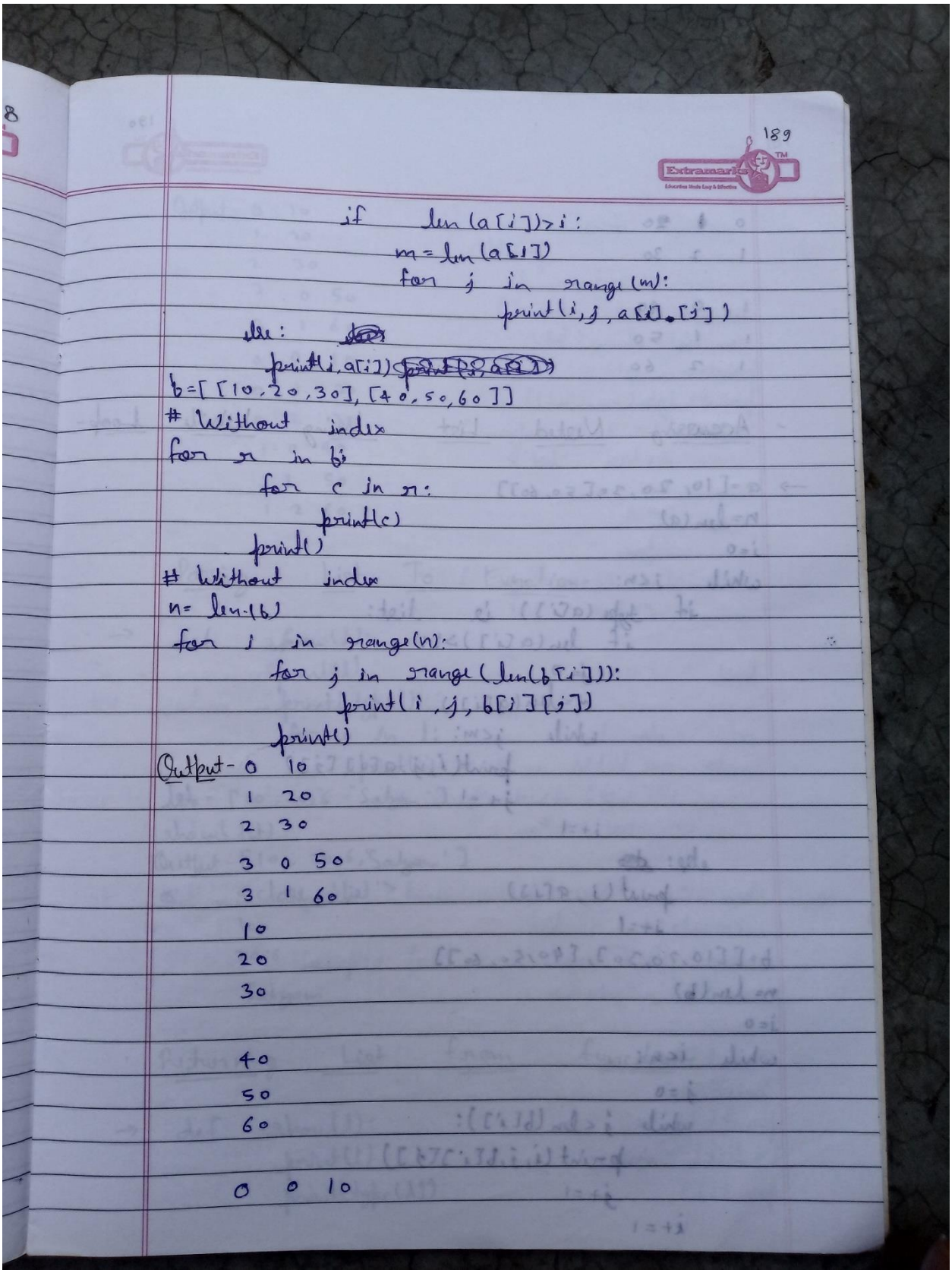


```
print("d[0]:",d[0])
print("d[1]:",d[1])
print("d[0][0]",d[0][0])
print("Before Modification D:",d)
d[0][1]=80
d[1][0]=70
print("After Modification D:",d)
```

Output-10
50
[50, 60]
Before Modification A: [10, 20, 30, [50, 60]]
After Modification - A: [10, 100, 30, [5, 60]]
C: [10, 20, 30, [50, 60]]
B: [50, 60]
10
50
[50, 60]
Before Modification C: [10, 20, 30, [50, 60]]
After Modification C: [10, 100, 30, [70, 80]]
D: [[10, 20, 30], [40, 50, 60]]
d[0]: [10, 20, 30]
d[1]: [40, 50, 60]
d[0][0]: 10
Before Modification D: [[10, 20, 30], [40, 50, 60]]
After Modification D: [[10, 80, 30], [70, 50, 60]]

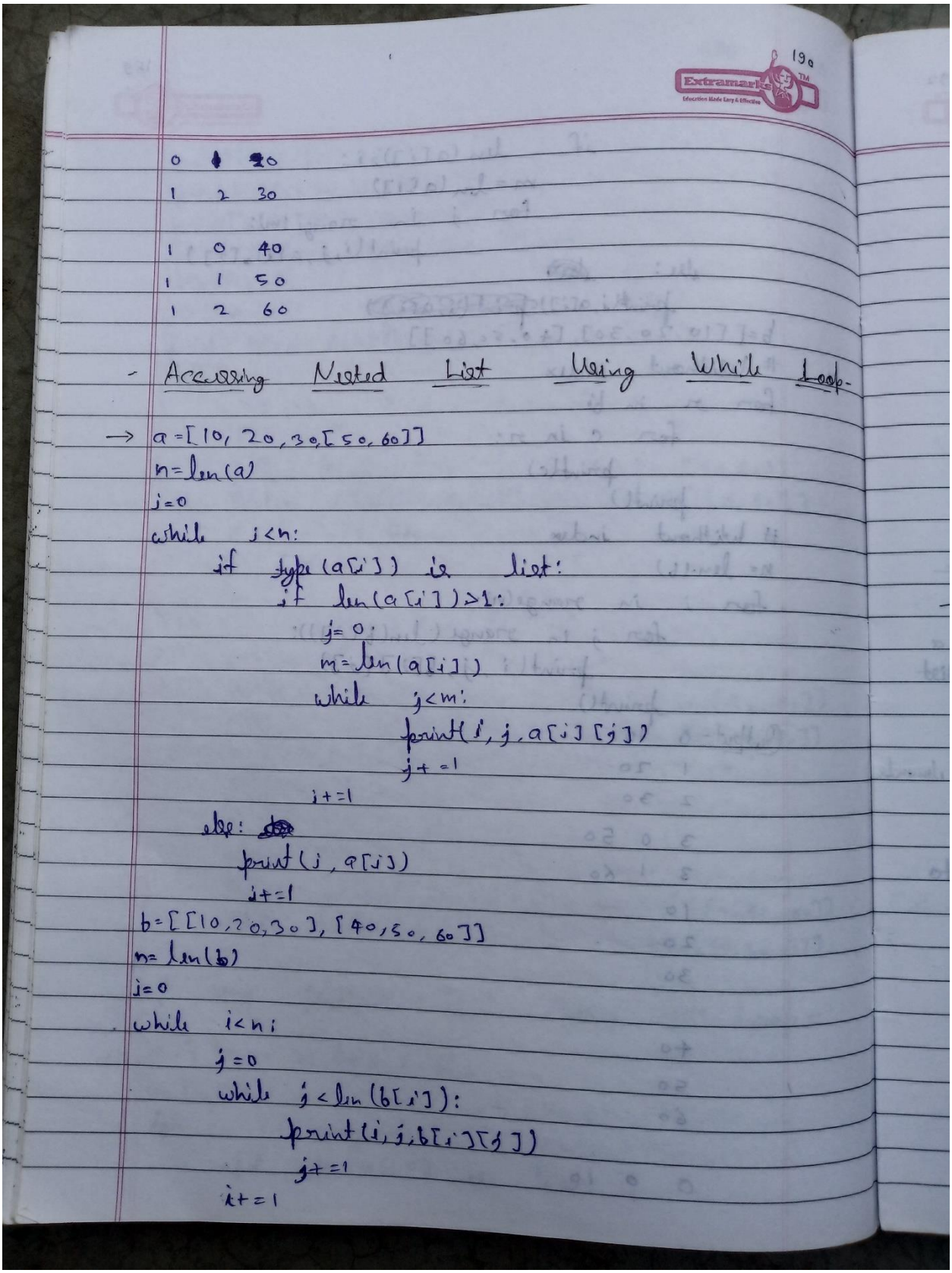
Accessing Nested List Using for Loop-

→
```
a=[10,20,30,[50,60]]
n=len(a)
for i in range(n):
    if type(a[i]) is list:
```



```
                if len(a[i])>i:
                    m=len(a[i])
                    for j in range(m):
                        print(i,j,a[i][j])
        else:
            print(i,a[i])
b=[[10,20,30],[40,50,60]]
# Without index
for r in b:
    for c in r:
        print(c)
    print()
# Without index
n=len(b)
for i in range(n):
    for j in range(len(b[i])):
        print(i,j,b[i][j])
    print()
```

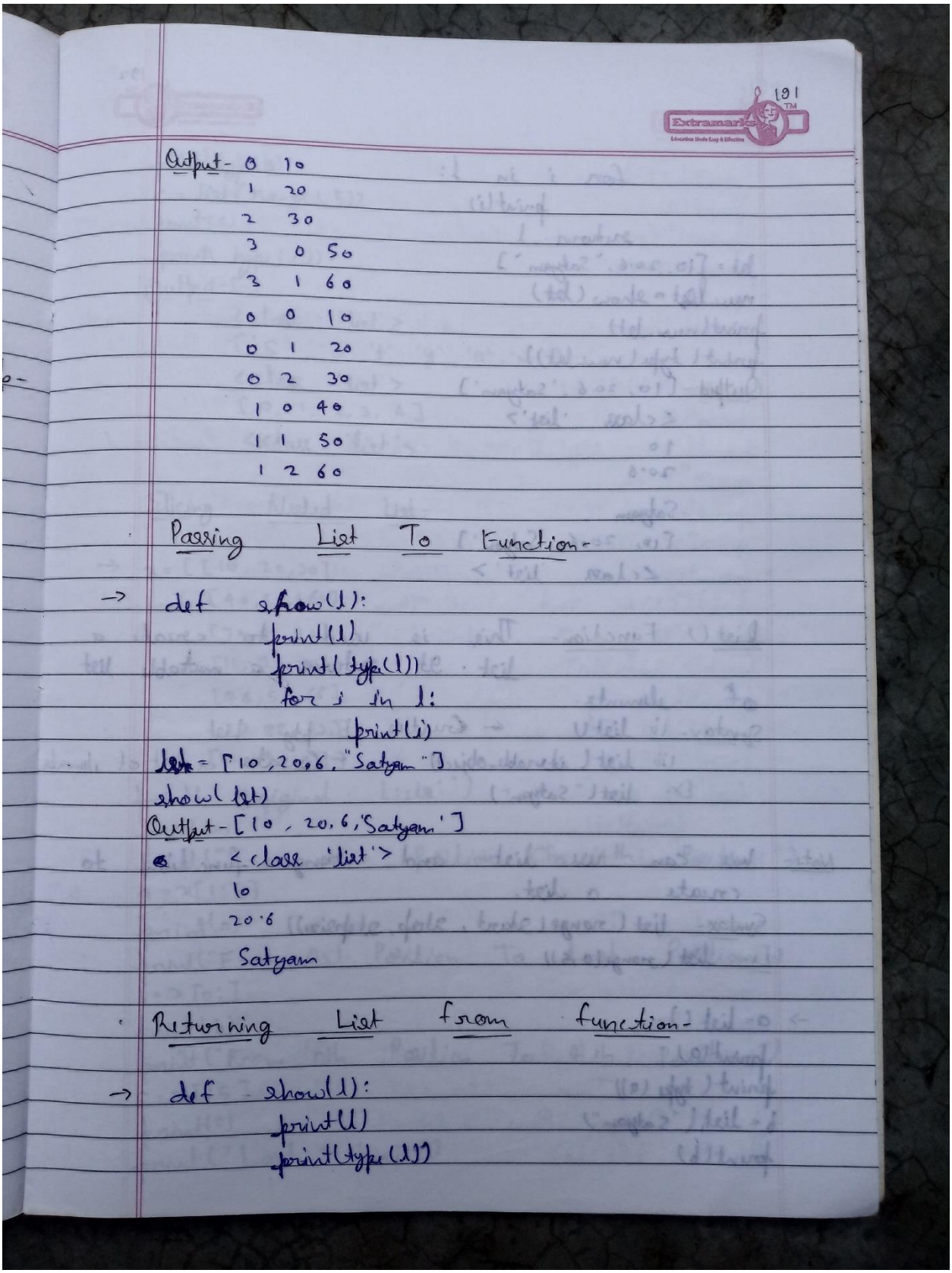
Output -
```
0 10
1 20
2 30
3 0 50
3 1 60
10
20
30

40
50
60

0 0 10
```



```
0  1  20
1  2  30

1  0  40
1  1  50
1  2  60
```

- Accessing Nested List Using While Loop-

→
```
a=[10, 20, 30,[50, 60]]
n=len(a)
i=0
while i<n:
    if type(a[i]) is list:
        if len(a[i])>1:
            j=0
            m=len(a[i])
            while j<m:
                print(i, j, a[i][j])
                j+=1
        i+=1
    else:
        print(i, a[i])
        i+=1
b=[[10,20,30],[40,50,60]]
n=len(b)
i=0
while i<n:
    j=0
    while j<len(b[i]):
        print(i, j, b[i][j])
        j+=1
    i+=1
```



Output -
```
0 10
1 20
2 30
3 0 50
3 1 60
0 0 10
0 1 20
0 2 30
1 0 40
1 1 50
1 2 60
```

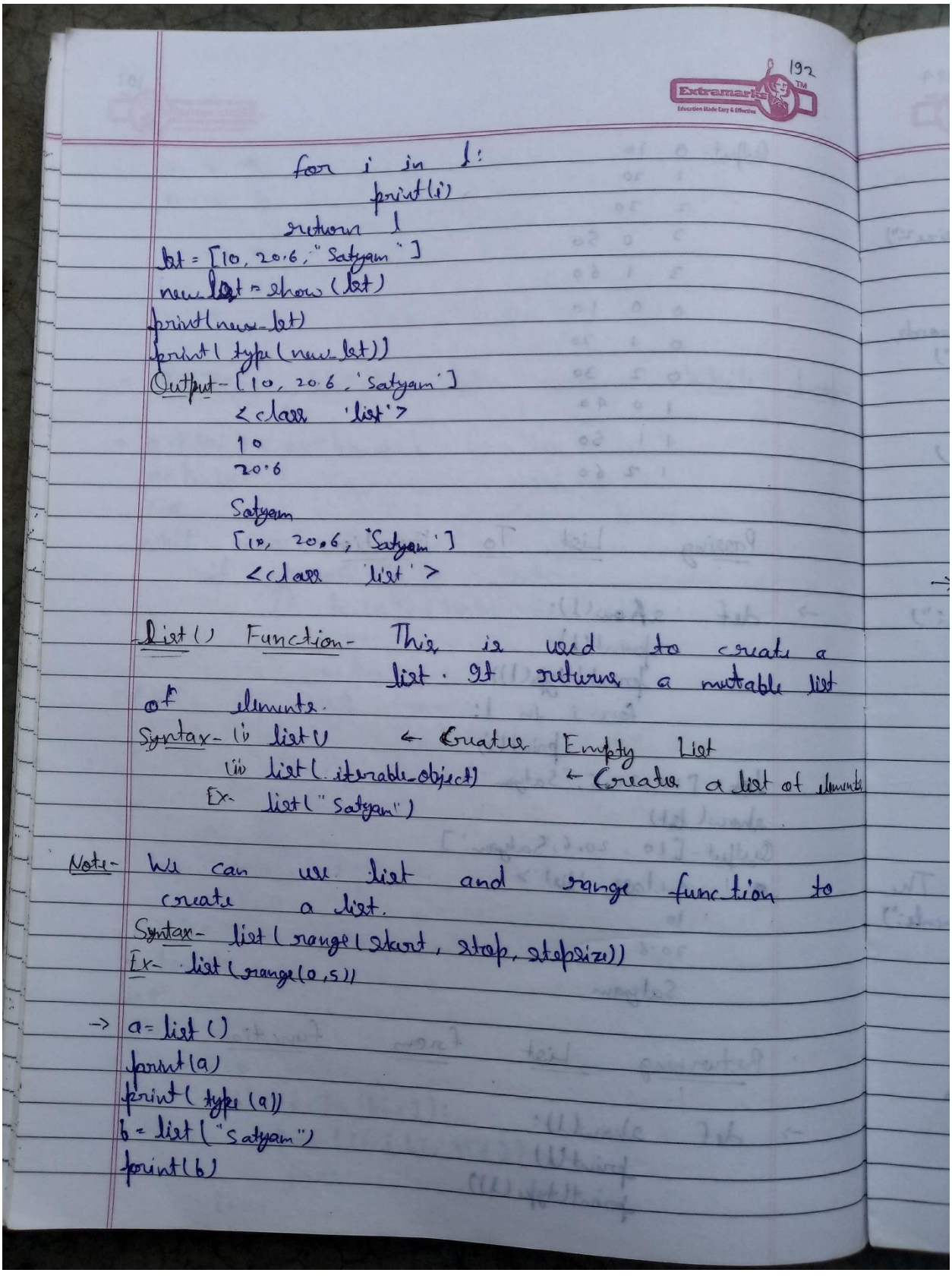
Passing List To Function -

→
```
def show(l):
    print(l)
    print(type(l))
    for i in l:
        print(i)
lst = [10, 20.6, "Satyam"]
show(lst)
```

Output -
```
[10, 20.6, 'Satyam']
<class 'list'>
10
20.6
Satyam
```

Returning List from function -

→
```
def show(l):
    print(l)
    print(type(l))
```



```
        for i in l:
            print(i)
        return l
lst = [10, 20.6, "Satyam"]
new_lst = show(lst)
print(new_lst)
print(type(new_lst))
```

Output-
```
[10, 20.6, 'satyam']
<class 'list'>
10
20.6
Satyam
[10, 20.6, 'Satyam']
<class 'list'>
```

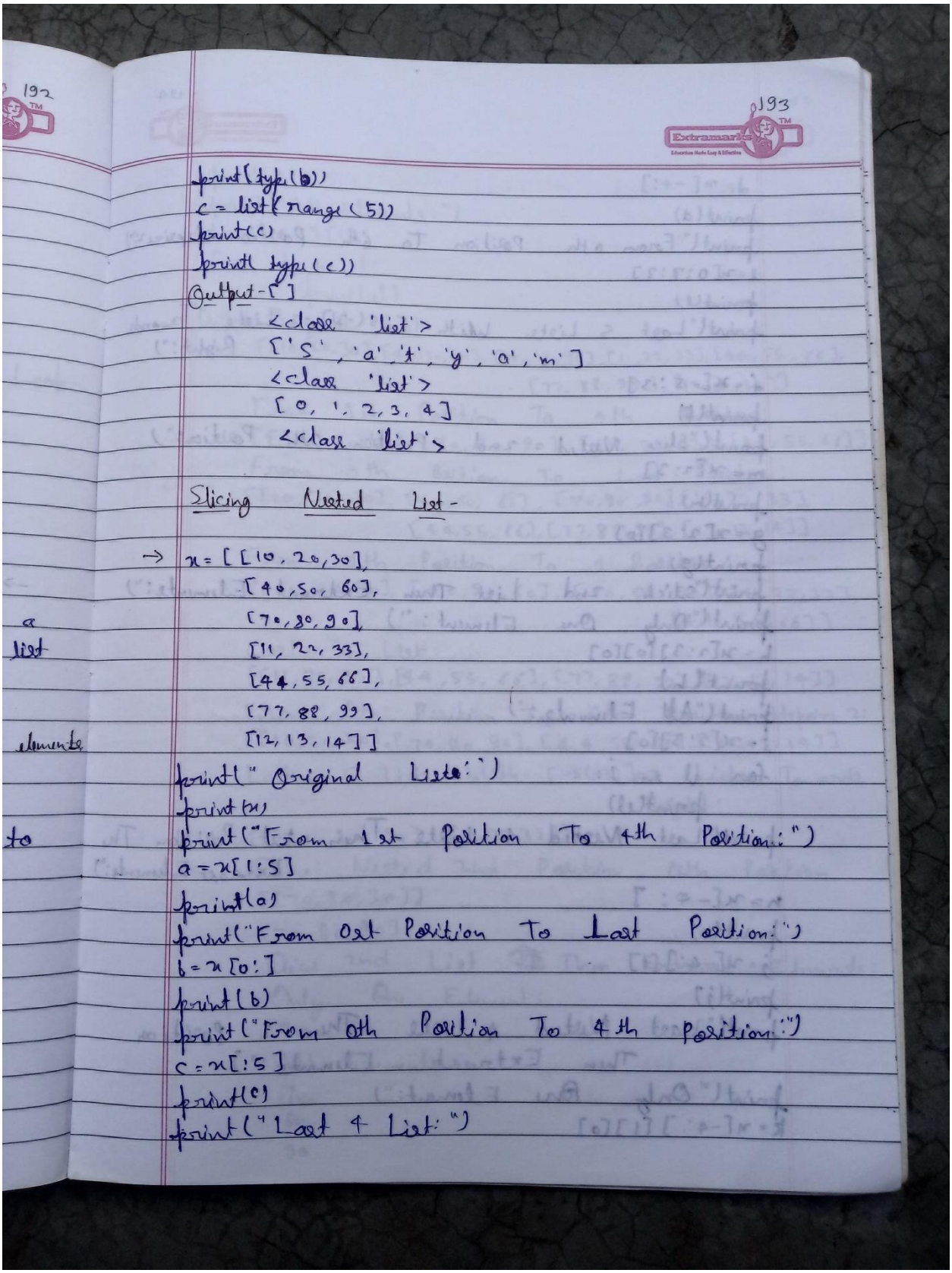
list() Function- This is used to create a list. It returns a mutable list of elements.

Syntax- (i) list() ← Creates Empty List
(ii) list(iterable_object) ← Create a list of elements
Ex- list("Satyam")

Note- We can use list and range function to create a list.

Syntax- list(range(start, stop, stepsize))
Ex- list(range(0,5))

→
```
a = list()
print(a)
print(type(a))
b = list("satyam")
print(b)
```

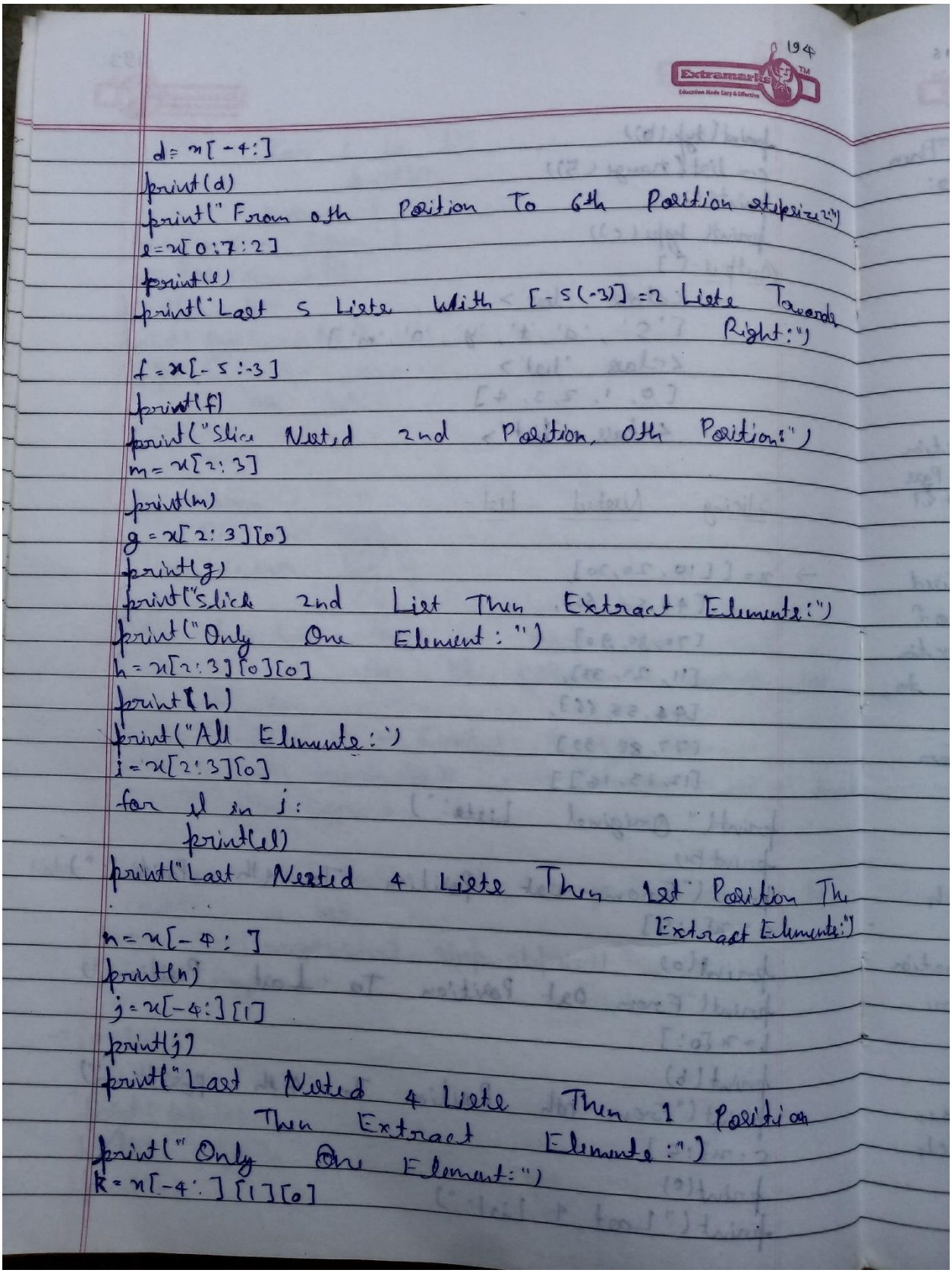


```
print(type(b))
c = list(range(5))
print(c)
print(type(c))
Output-[]
        <class 'list'>
        ['S', 'a', 't', 'y', 'a', 'm']
        <class 'list'>
        [0, 1, 2, 3, 4]
        <class 'list'>

Slicing Nested List -

→ x = [[10, 20, 30],
       [40, 50, 60],
       [70, 80, 90],
       [11, 22, 33],
       [44, 55, 66],
       [77, 88, 99],
       [12, 13, 14]]
print("Original Lists:")
print(x)
print("From 1st Position To 4th Position:")
a = x[1:5]
print(a)
print("From 0st Position To Last Position:")
b = x[0:]
print(b)
print("From 0th Position To 4th Position:")
c = x[:5]
print(c)
print("Last 4 List:")
```

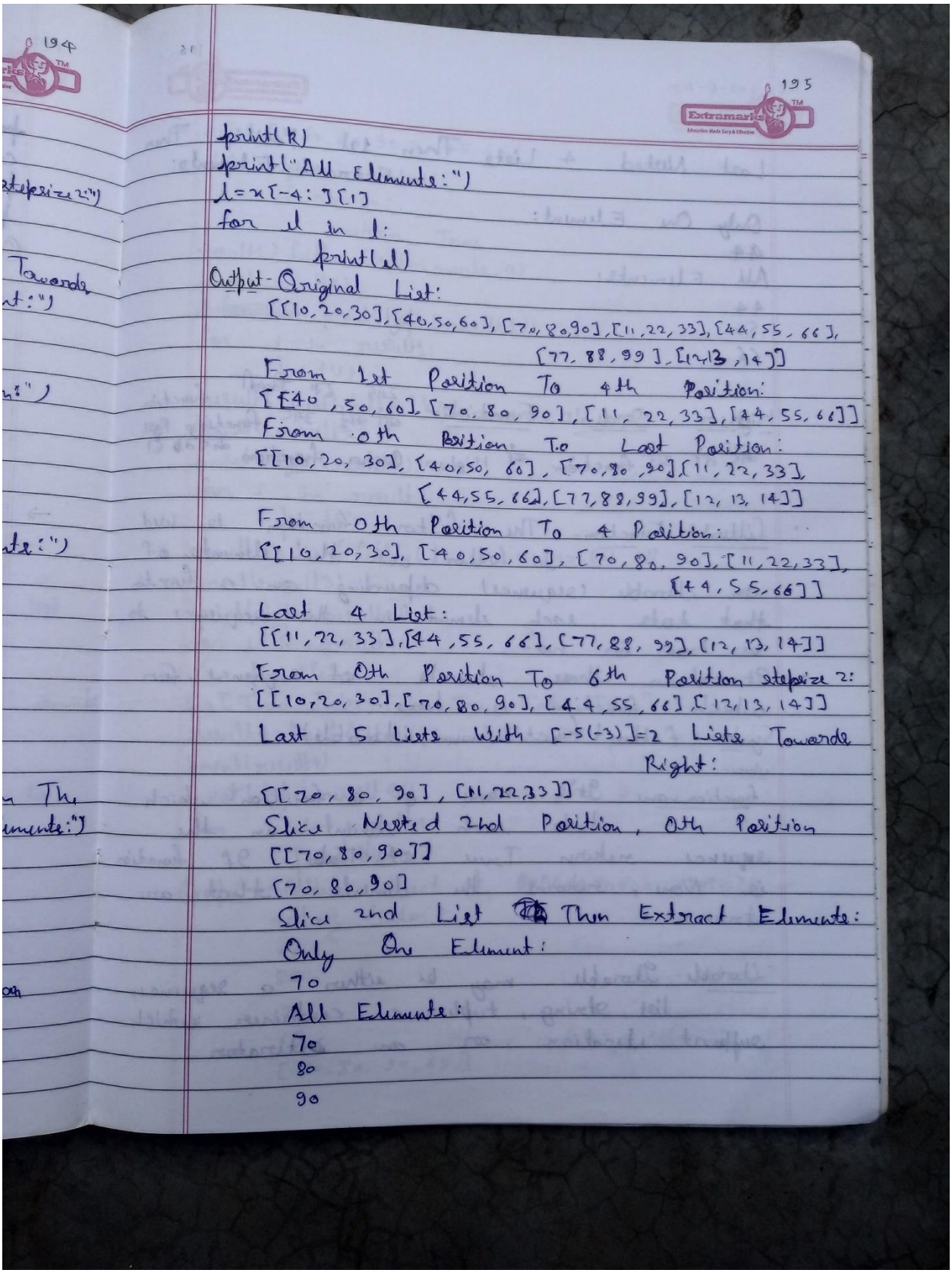
a list

elements

to



```
d = x[-4:]
print(d)
print("From 0th Position To 6th Position stepsize 2:")
e = x[0:7:2]
print(e)
print("Last 5 Lists With [-5(-3)] =2 Lists Towards Right:")
f = x[-5:-3]
print(f)
print("Slice Nested 2nd Position, 0th Position:")
m = x[2:3]
print(m)
g = x[2:3][0]
print(g)
print("Slick 2nd List Then Extract Elements:")
print("Only One Element:")
h = x[2:3][0][0]
print(h)
print("All Elements:")
i = x[2:3][0]
for el in i:
    print(el)
print("Last Nested 4 Lists Then 1st Position Th Extract Elements:")
n = x[-4:]
print(n)
j = x[-4:][1]
print(j)
print("Last Nested 4 Lists Then 1 Position Then Extract Elements:")
print("Only One Element:")
k = x[-4:][1][0]
```



194

print(k)
print("All Elements:")
l=x[-4:][1]
for d in l:
    print(d)
Output- Original List:
[[10,20,30],[40,50,60],[70,80,90],[11,22,33],[44,55,66],
[77,88,99],[12,13,14]]
From 1st Position To 4th Position:
[[40,50,60],[70,80,90],[11,22,33],[44,55,66]]
From 0th Position To Last Position:
[[10,20,30],[40,50,60],[70,80,90],[11,22,33],
[44,55,66],[77,88,99],[12,13,14]]
From 0th Position To 4 Position:
[[10,20,30],[40,50,60],[70,80,90],[11,22,33],
[44,55,66]]
Last 4 List:
[[11,22,33],[44,55,66],[77,88,99],[12,13,14]]
From 0th Position To 6th Position stepsize 2:
[[10,20,30],[70,80,90],[44,55,66],[12,13,14]]
Last 5 Lists With [-5(-3)]=2 Lists Towards Right:
[[70,80,90],[11,22,33]]
Slice Nested 2nd Position, 0th Position
[[70,80,90]]
[70,80,90]
Slice 2nd List Then Extract Elements:
Only One Element:
70
All Elements:
70
80
90

195

196

Last Nested 4 Lists Then 1st Position Then Extract Elements:

Only One Element:
44
All Elements:
44
55
66

Higher Order Function - यदि हम किसी function को कोई और function pass करें तो उस function को Higher Order function कहते हैं।

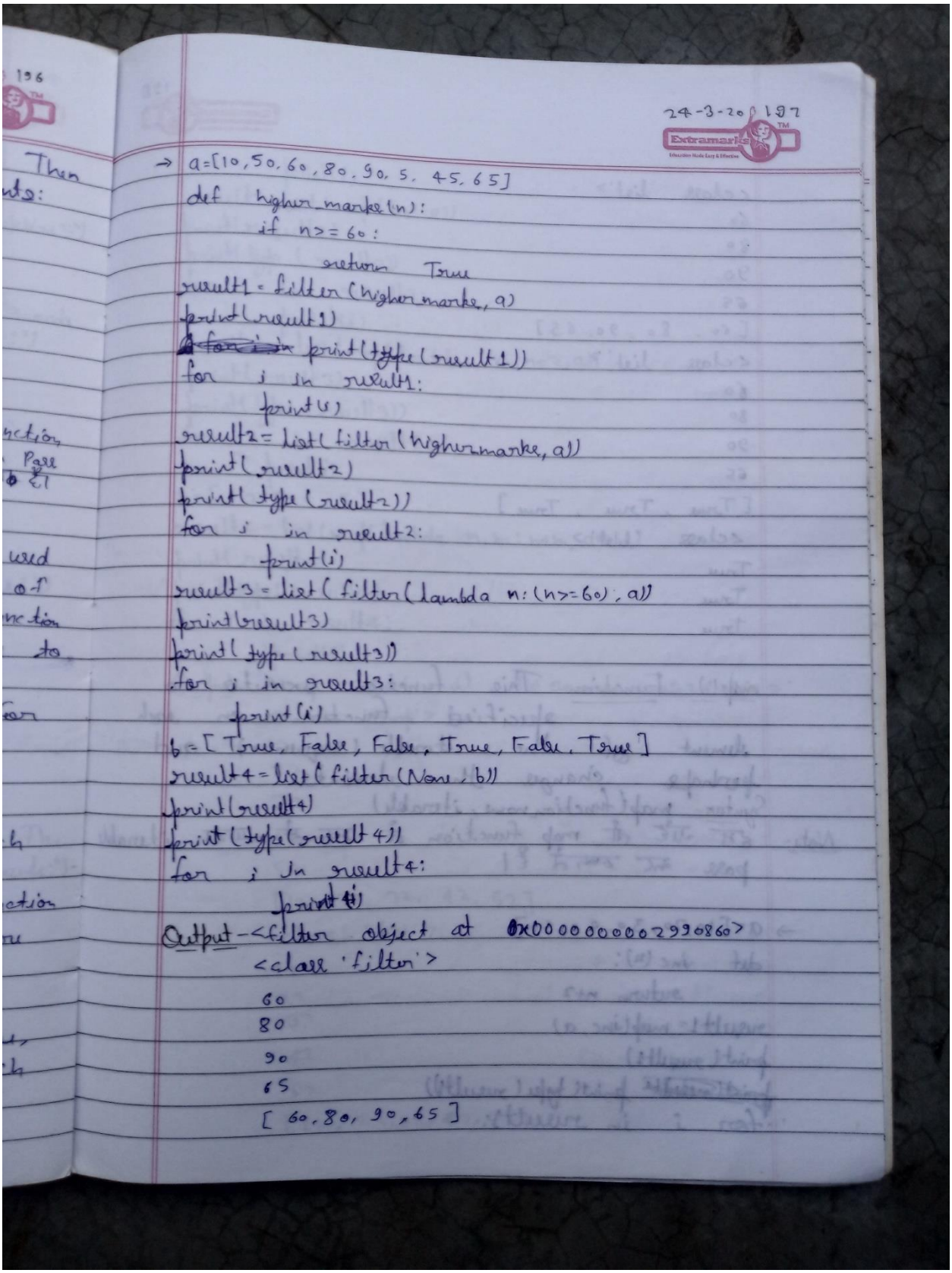
- filter() Function - The filter function is used to filter out the elements of an iterable (sequence) depending on a function that tests each element in the sequence to be true or not.
- It return those elements of sequence for which function is true.

Syntax - filter(function_name, iterable)

where -

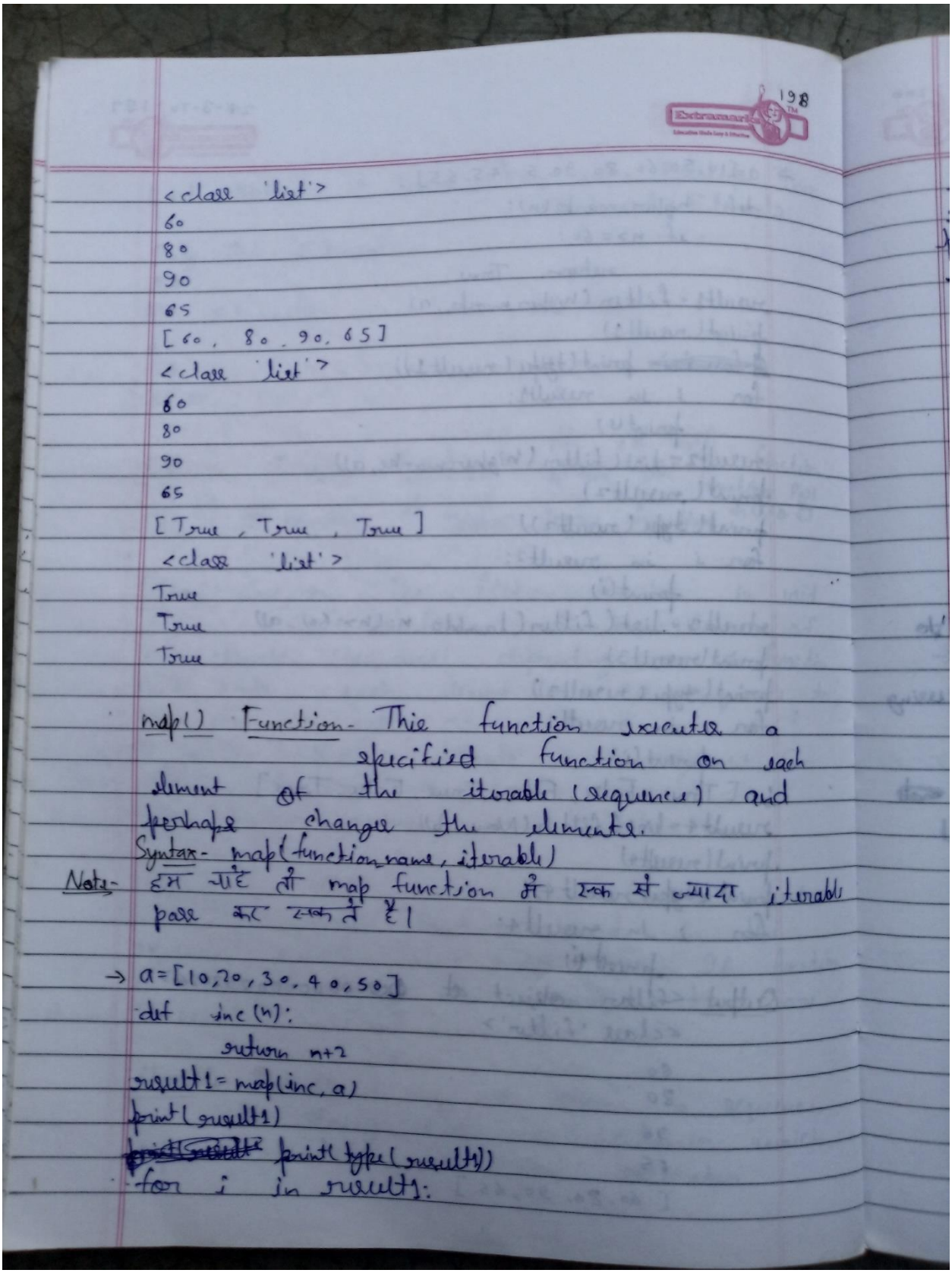
function_name - It's name of a function which tests each element in the sequence return True or False. If function is None, returns the elements that are true.

iterable - Iterable may be either a sequence, list, string, tuple, a container which support iteration, or an iterator



```
a=[10,50,60,80,90,5,45,65]
def higher_marks(n):
    if n>=60:
        return True
result1=filter(higher_marks,a)
print(result1)
print(type(result1))
for i in result1:
    print(i)
result2=list(filter(higher_marks,a))
print(result2)
print(type(result2))
for i in result2:
    print(i)
result3=list(filter(lambda n:(n>=60),a))
print(result3)
print(type(result3))
for i in result3:
    print(i)
b=[True,False,False,True,False,True]
result4=list(filter(None,b))
print(result4)
print(type(result4))
for i in result4:
    print(i)
```

Output -
```
<filter object at 0x0000000002990860>
<class 'filter'>
60
80
90
65
[60,80,90,65]
```



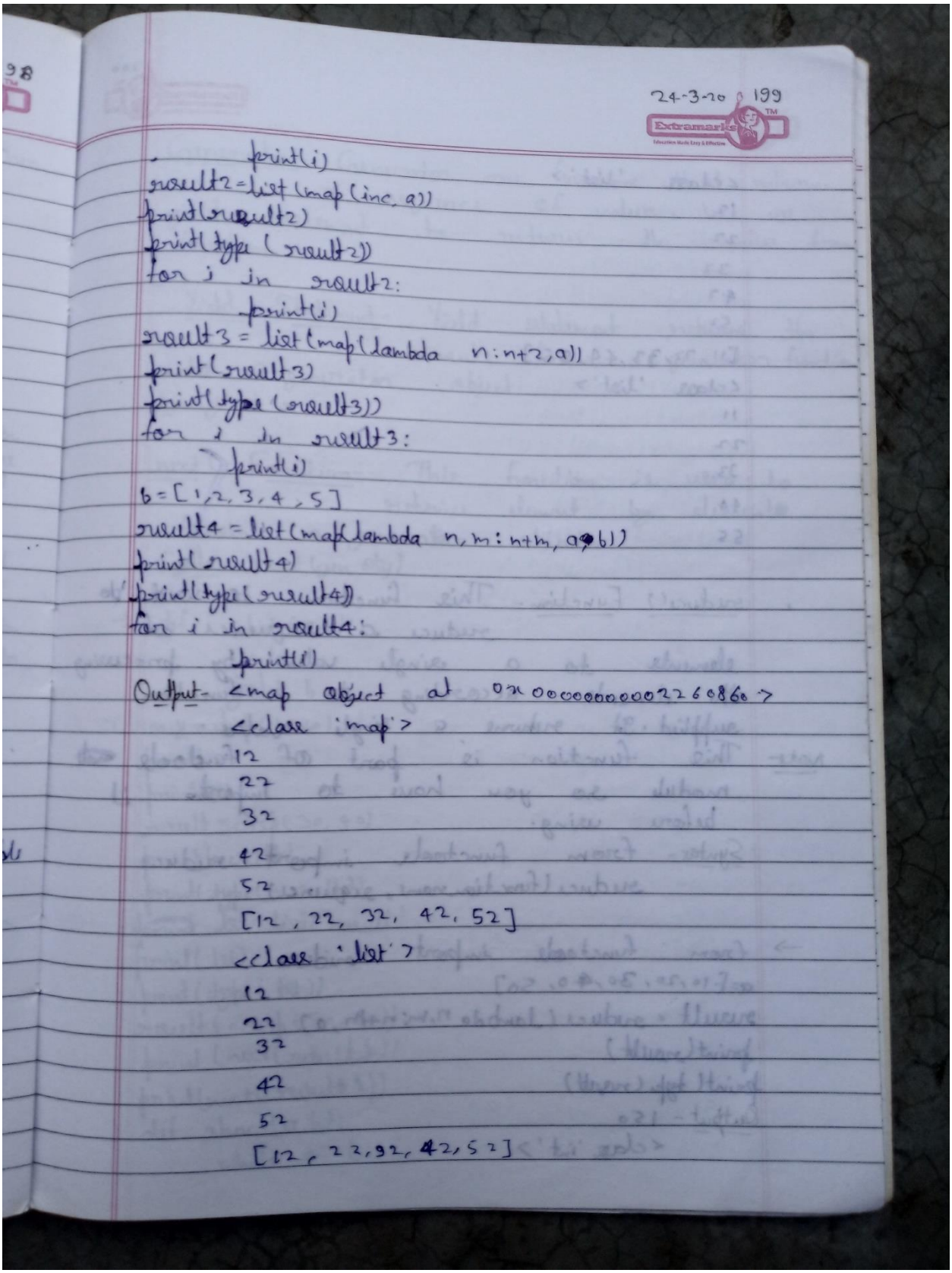
```
<class 'list'>
60
80
90
65
[60, 80, 90, 65]
<class 'list'>
60
80
90
65
[True, True, True]
<class 'list'>
True
True
True
```

map() Function- This function executes a specified function on each element of the iterable (sequence) and perhaps change the elements.

Syntax- map(function_name, iterable)

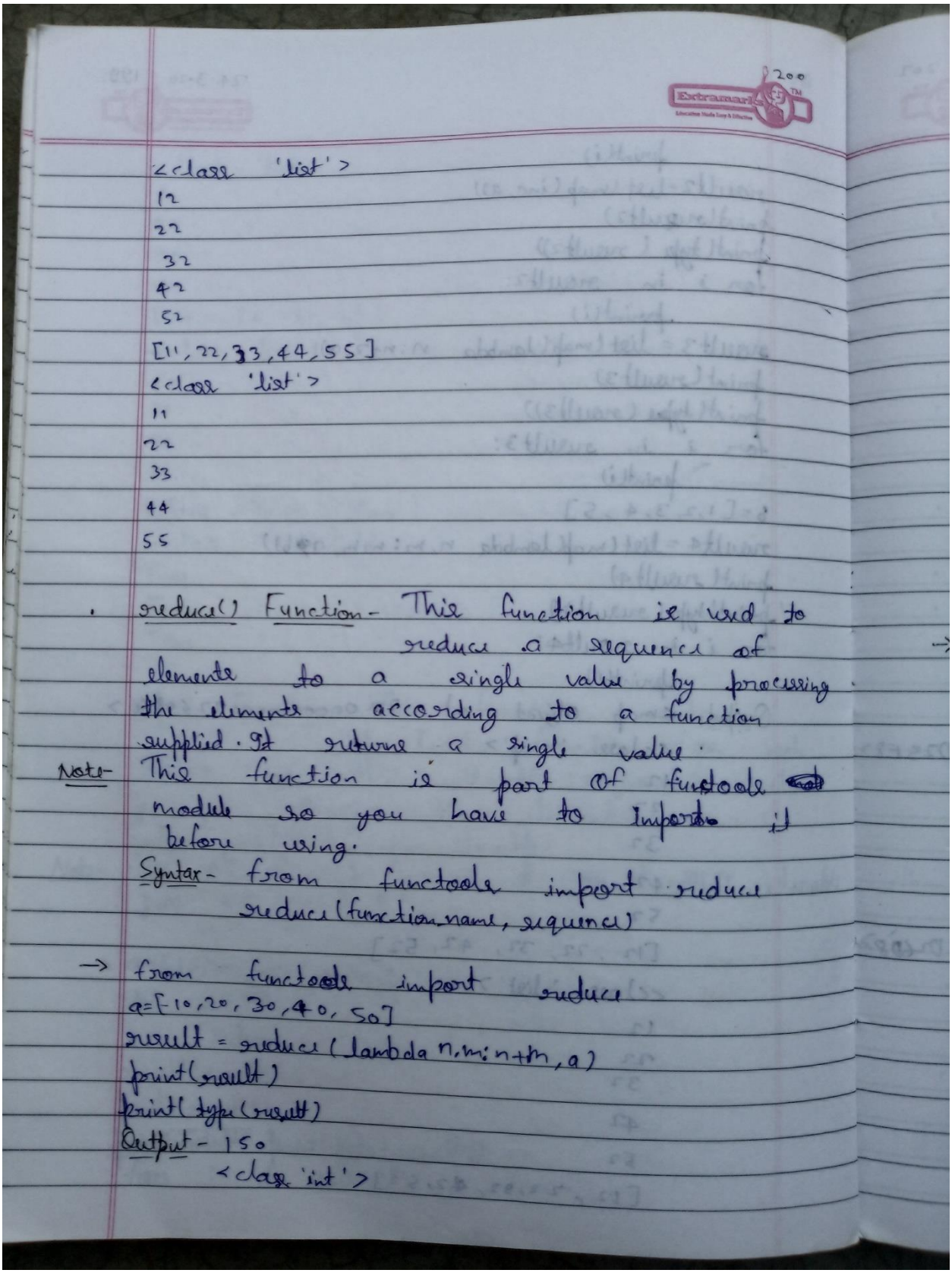
Note- हम जब map function में एक से ज्यादा iterable pass करते हैं।

→
```
a=[10,20,30,40,50]
def inc(n):
    return n+2
result1=map(inc, a)
print(result1)
print(type(result1))
for i in result1:
```



```
        print(i)
result2 = list(map(inc, a))
print(result2)
print(type(result2))
for i in result2:
        print(i)
result3 = list(map(lambda n:n+2,a))
print(result3)
print(type(result3))
for i in result3:
        print(i)
b=[1,2,3,4,5]
result4 = list(map(lambda n,m:n+m, a,b))
print(result4)
print(type(result4))
for i in result4:
        print(i)
```

Output- <map object at 0x000000000226D860>
<class 'map'>
12
22
32
42
52
[12, 22, 32, 42, 52]
<class 'list'>
12
22
32
42
52
[12, 22, 32, 42, 52]



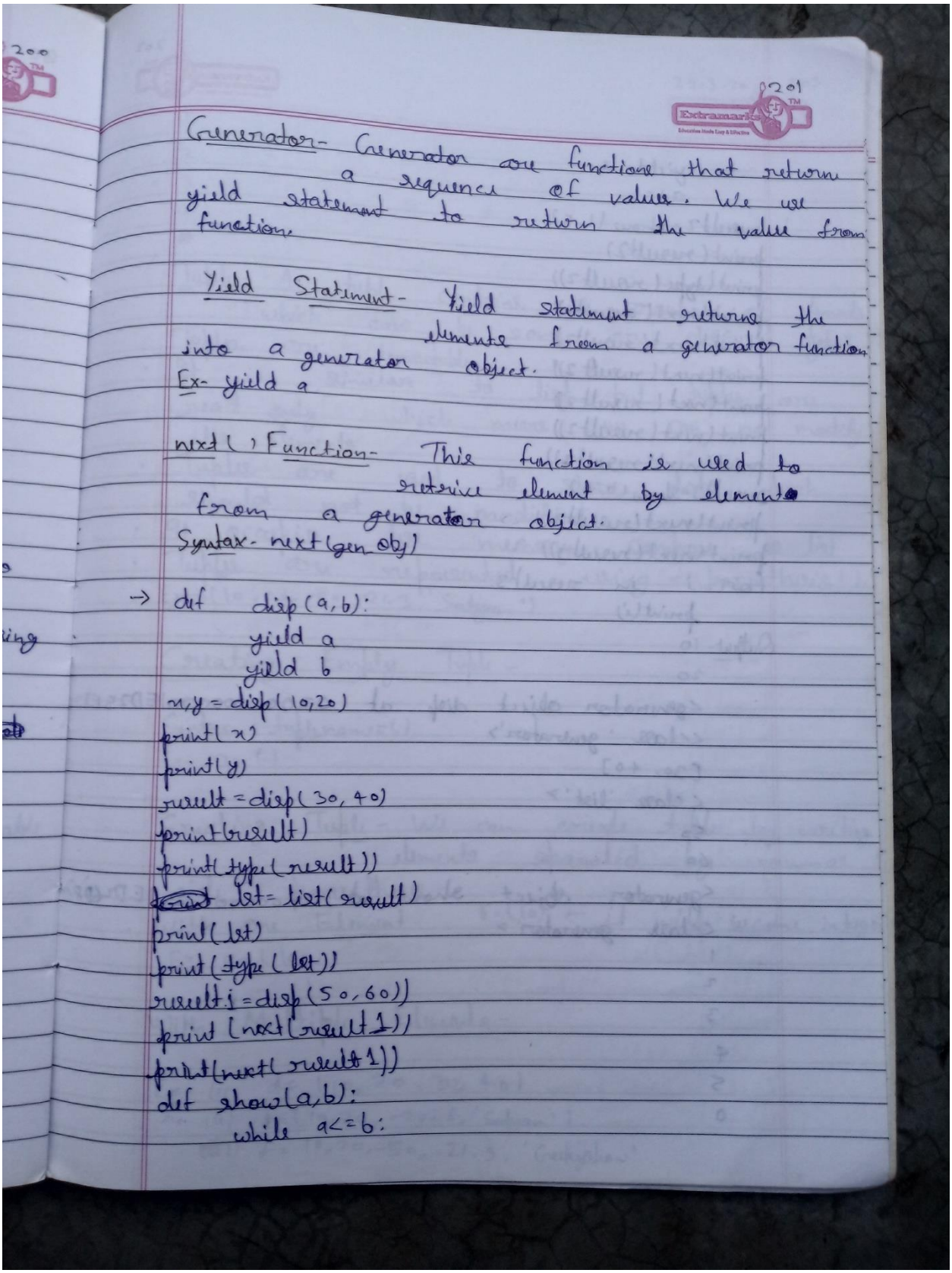
```
<class 'list'>
12
22
32
42
52
[11, 22, 33, 44, 55]
<class 'list'>
11
22
33
44
55
```

- reduce() Function - This function is used to reduce a sequence of elements to a single value by processing the elements according to a function supplied. It returns a single value.

Note - This function is part of functools module so you have to import it before using.

Syntax - from functools import reduce
reduce(function_name, sequence)

→
```
from functools import reduce
a=[10,20,30,40,50]
result = reduce(lambda n,m:n+m, a)
print(result)
print(type(result))
```
Output - 150
<class 'int'>

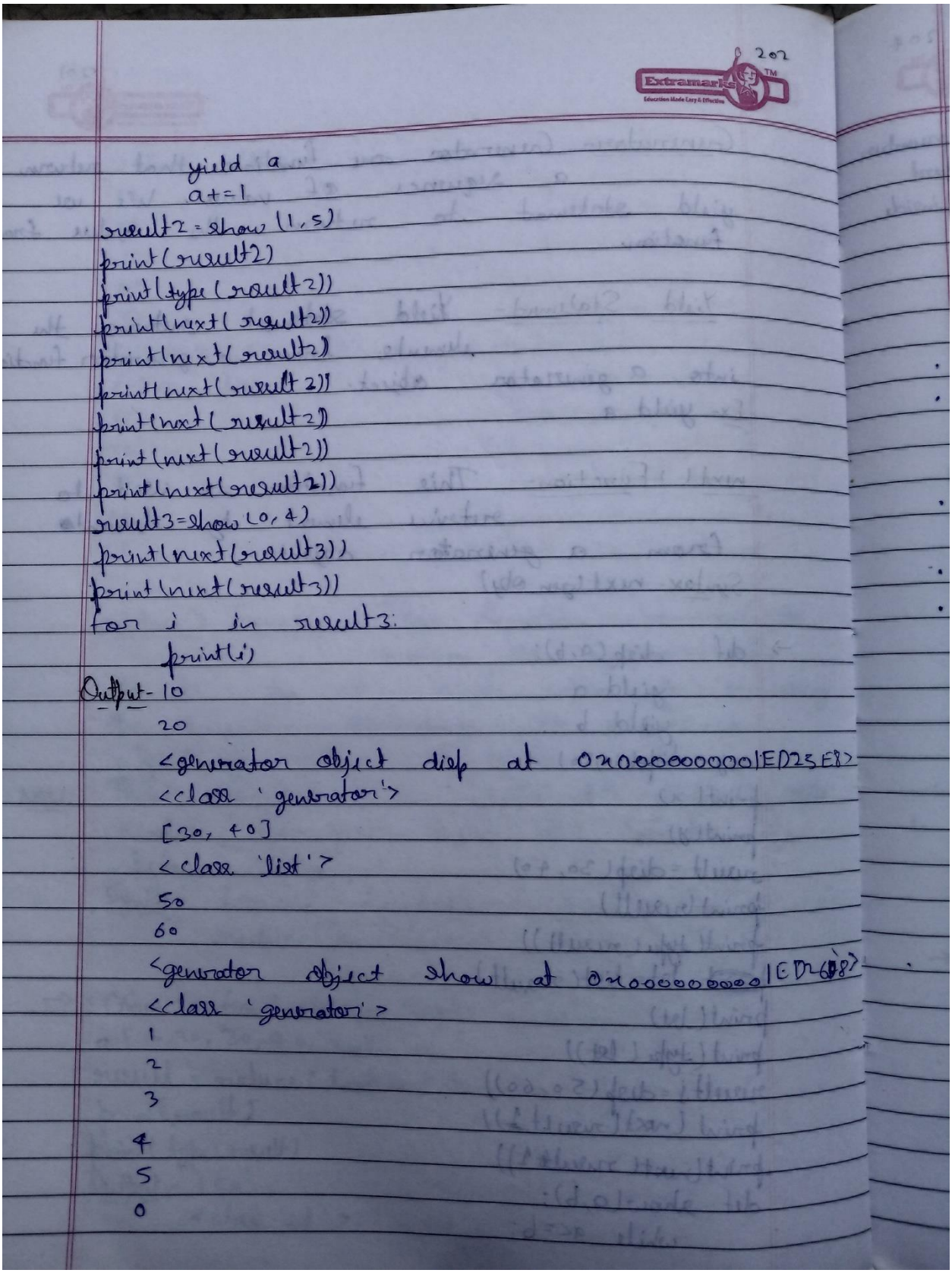


Generator- Generator are functions that return a sequence of values. We use yield statement to return the value from function.

Yield Statement- Yield statement returns the elements from a generator function into a generator object.
Ex- yield a

next() Function- This function is used to retrive element by element from a generator object.
Syntax- next(gen_obj)

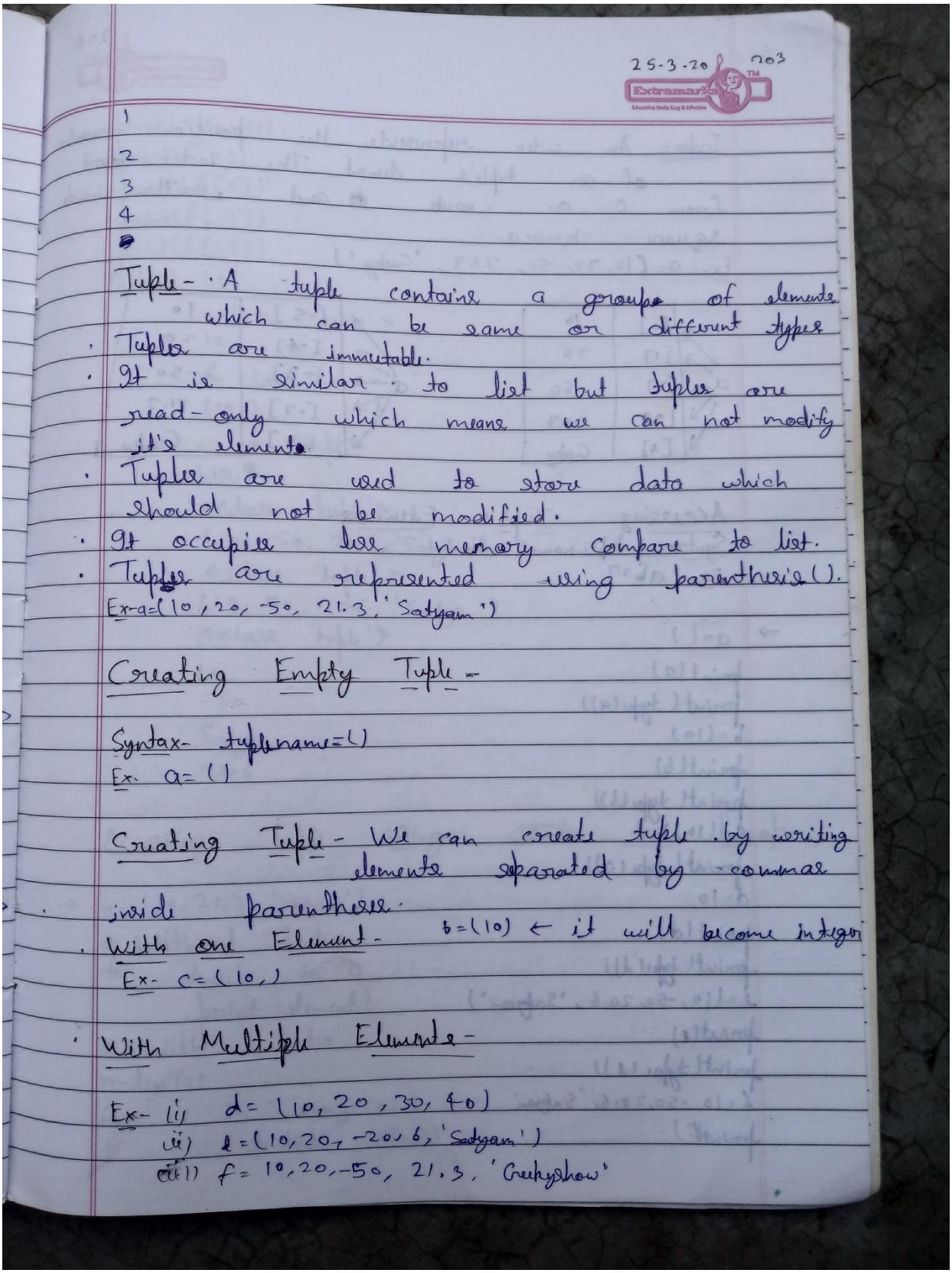
```
def disp(a,b):
    yield a
    yield b
x,y = disp(10,20)
print(x)
print(y)
result = disp(30,40)
print(result)
print(type(result))
lst = list(result)
print(lst)
print(type(lst))
result.i = disp(50,60)
print(next(result.i))
print(next(result.i))
def show(a,b):
    while a<=b:
```



```
        yield a
        a+=1
result2 = show(1,5)
print(result2)
print(type(result2))
print(next(result2))
print(next(result2))
print(next(result2))
print(next(result2))
print(next(result2))
print(next(result2))
result3=show(0,4)
print(next(result3))
print(next(result3))
for i in result3:
    print(i)
```

Output-

```
10
20
<generator object disp at 0x000000000EED23E8>
<class 'generator'>
[30, 40]
<class 'list'>
50
60
<generator object show at 0x000000000EED2468>
<class 'generator'>
1
2
3
4
5
0
```



25-3-20 203

Extramarks

1
2
3
4
5

Tuple - A tuple contains a groups of elements which can be same or different types

- Tuples are immutable.
- It is similar to list but tuples are read-only which means we can not modify it's elements.
- Tuples are used to store data which should not be modified.
- It occupies less memory compare to list.
- Tuples are represented using parenthesis ().

Ex- a=(10, 20, -50, 21.3, 'Satyam')

Creating Empty Tuple -

Syntax- tuplename=()
Ex. a=()

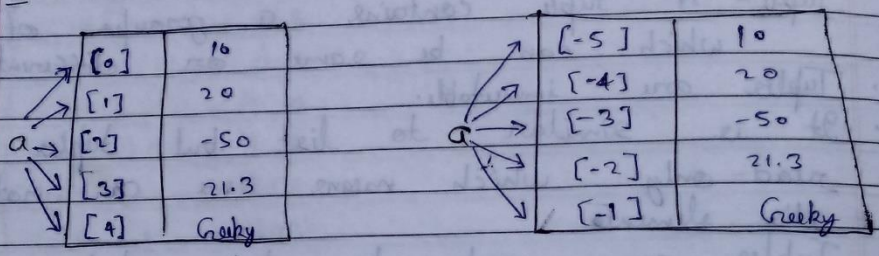
Creating Tuple - We can create tuple by writing elements separated by commas inside parenthesis.

- With one Element - b=(10) ← it will become integer
  Ex- c=(10,)
- With Multiple Elements -

Ex- (i) d=(10, 20, 30, 40)
(ii) e=(10, 20, -20, 6, 'Satyam')
(iii) f=10, 20, -50, 21.3, 'Geekyshow'

Index- An index represents the position number of an tuple's element. The index start from 0 or words and written inside square braces.

Ex. a=(10, 20, -50, 21.3, 'Geeky')

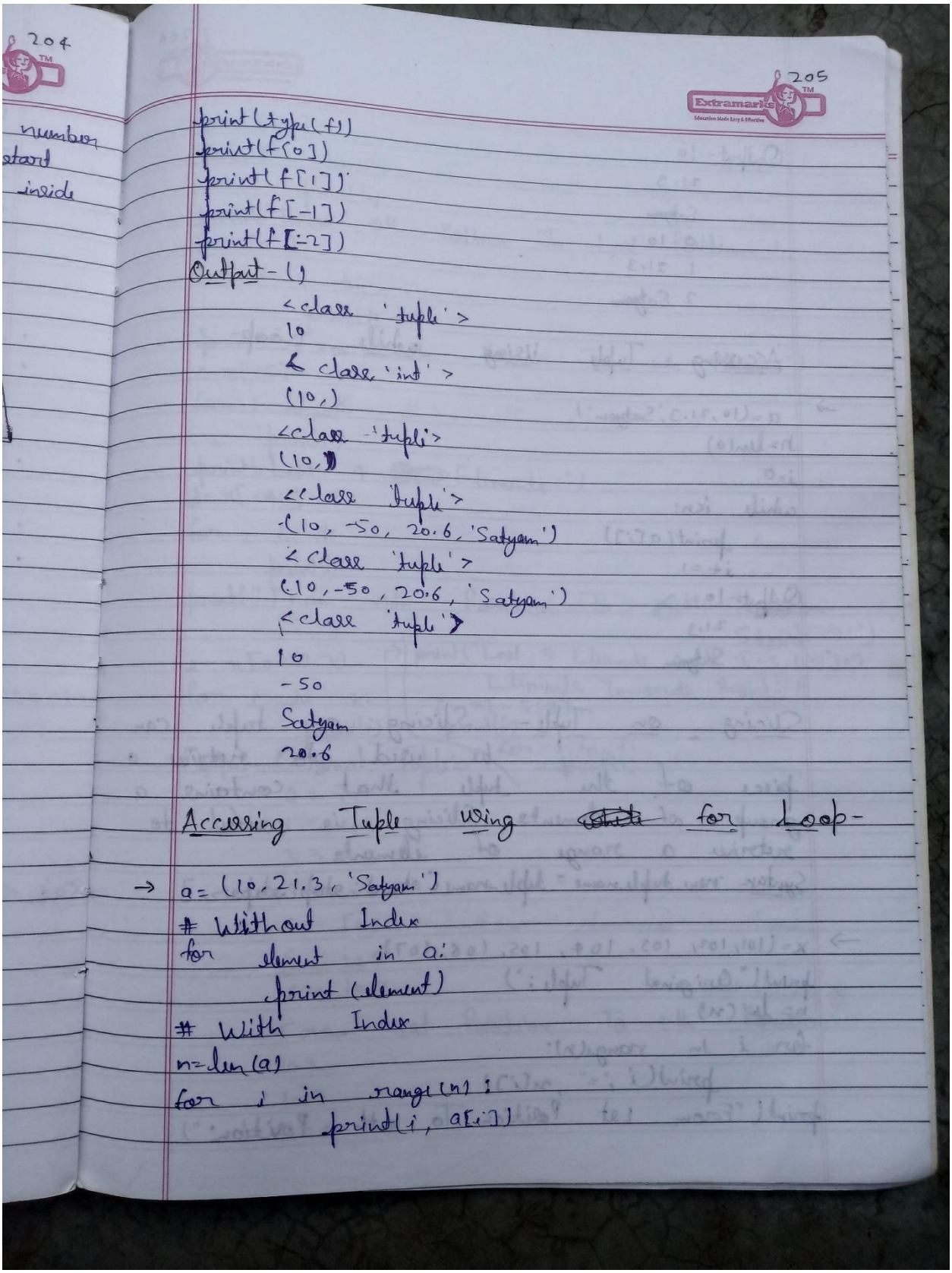


Accessing Tuple's Elements-

Syntax- tuple-name[index]

Ex. a[2]

```
a=()
print(a)
print(type(a))
b=(10)
print(b)
print(type(b))
c=(10,)
print(type(c))
d=10,
print(d)
print(type(d))
e=(10,-50,20.6,'Satyam')
print(e)
print(type(e))
f=10,-50,20.6,'Satyam'
print(f)
```



204

number
start
inside

205

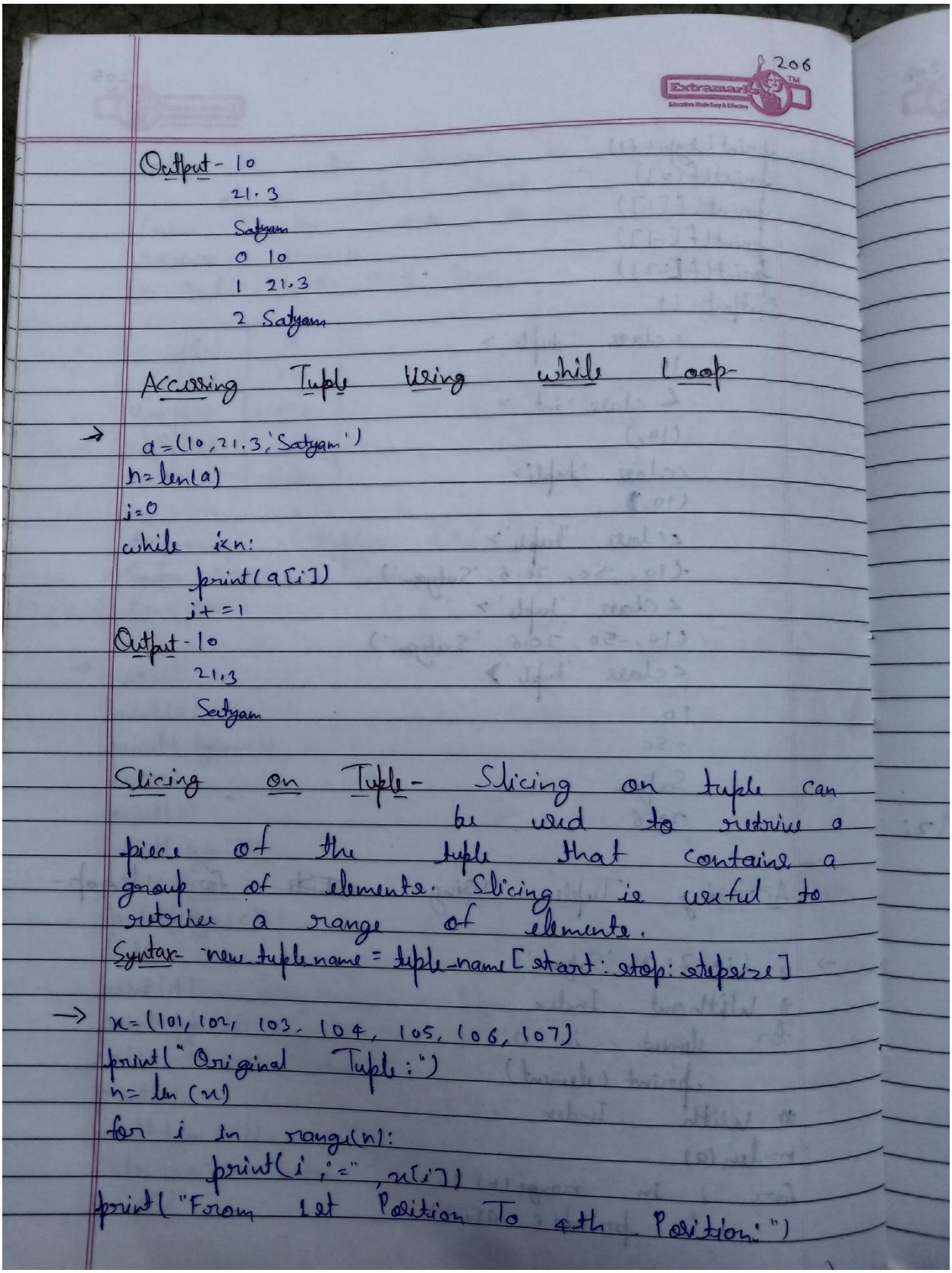
```
print(type(f))
print(f[0])
print(f[1])
print(f[-1])
print(f[-2])
```

Output -

```
<class 'tuple'>
10
<class 'int'>
(10,)
<class 'tuple'>
(10,)
<class 'tuple'>
(10, -50, 20.6, 'Satyam')
<class 'tuple'>
(10, -50, 20.6, 'Satyam')
<class 'tuple'>
10
-50
Satyam
20.6
```

Accessing Tuple using for Loop -

```
a = (10, 21.3, 'Satyam')
# Without Index
for element in a:
    print(element)
# With Index
n = len(a)
for i in range(n):
    print(i, a[i])
```



206

Output - 10
21.3
Satyam
0 10
1 21.3
2 Satyam

Accessing Tuple Using while Loop-

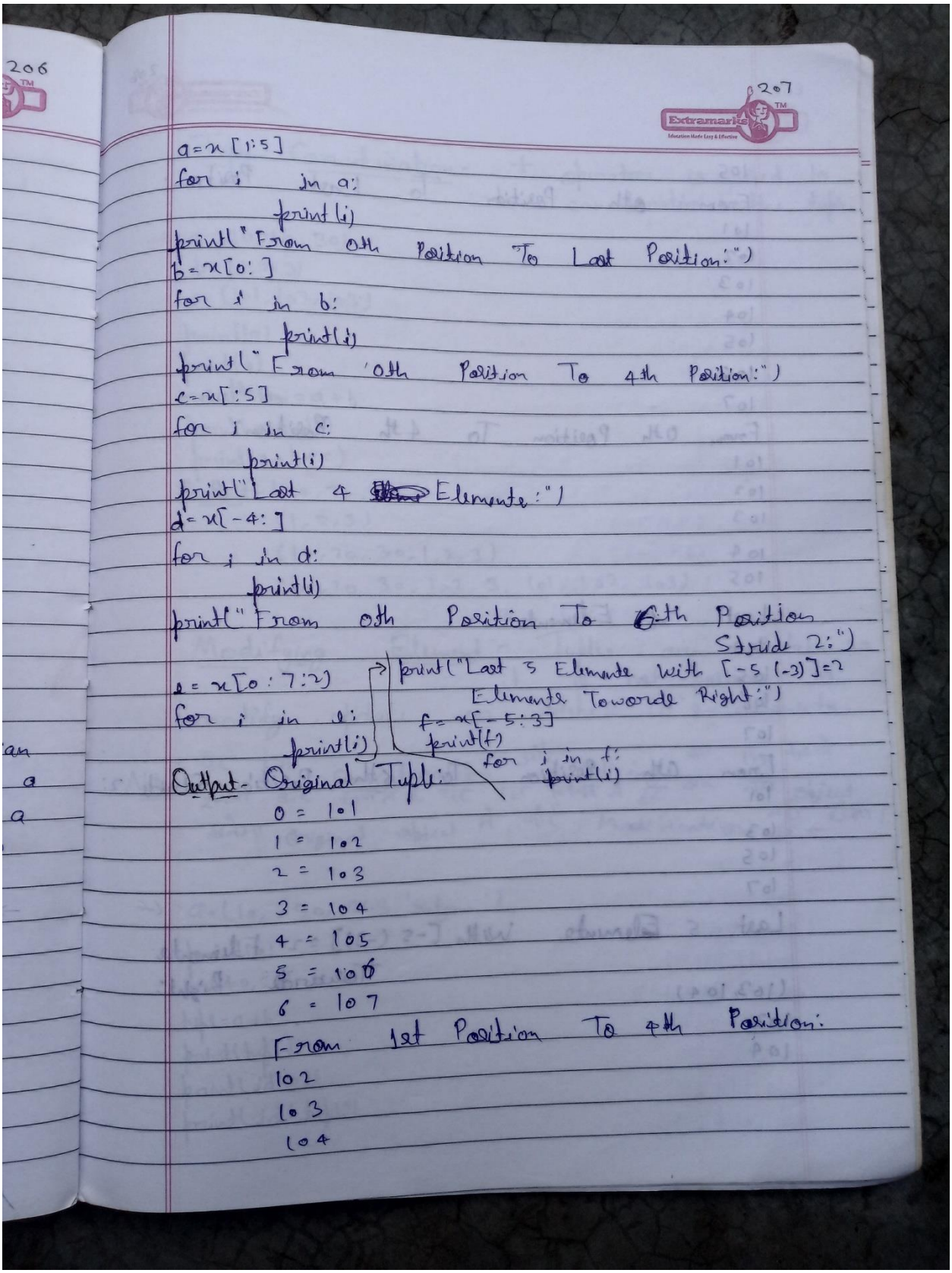
→
```
a=(10,21.3,'Satyam')
n=len(a)
i=0
while i<n:
    print(a[i])
    i+=1
```

Output - 10
21.3
Satyam

Slicing on Tuple - Slicing on tuple can be used to retrive a piece of the tuple that contains a group of elements. Slicing is useful to retrive a range of elements.

Syntax- new_tuple_name = tuple_name[start:stop:stepsize]

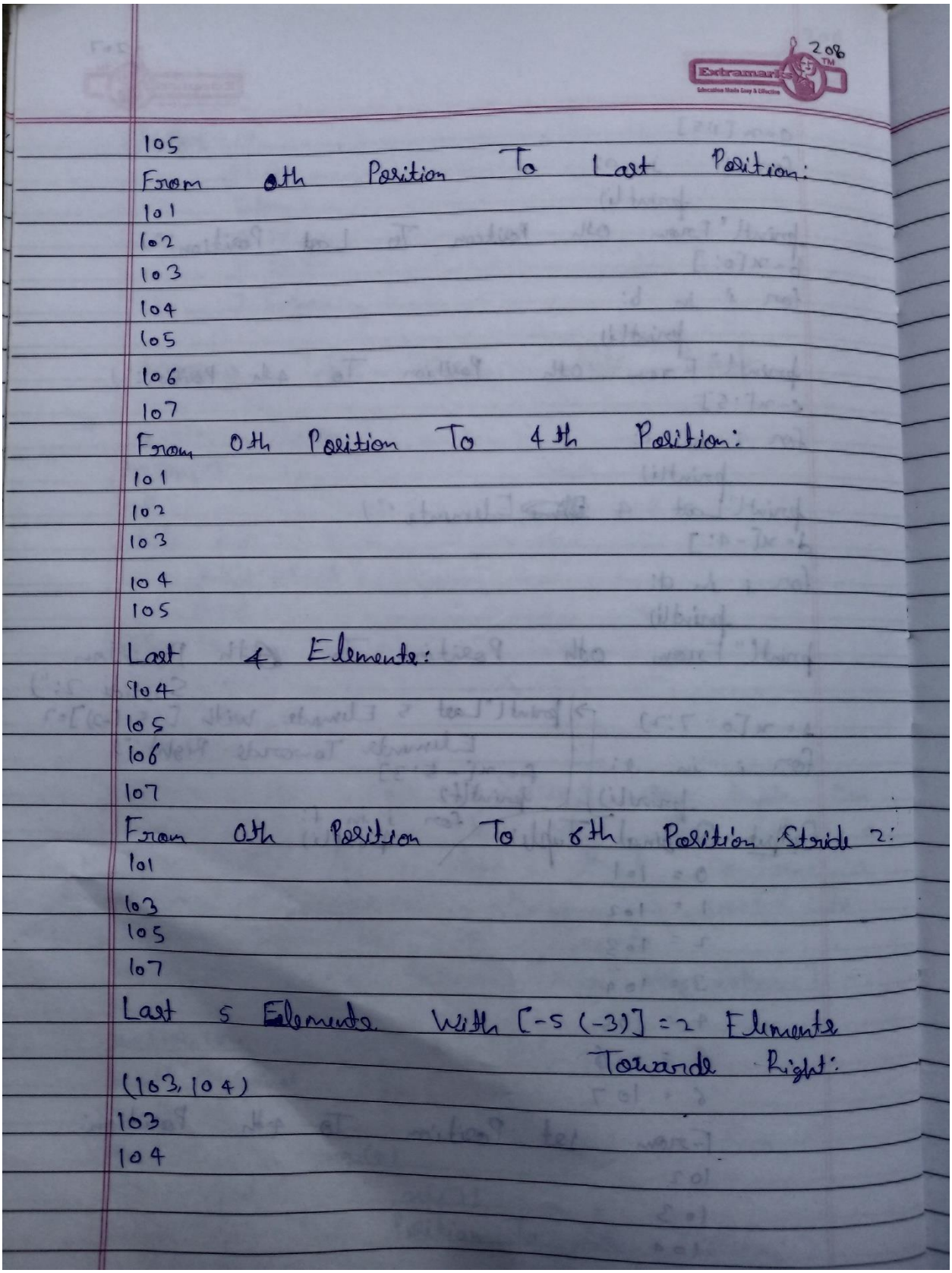
→
```
x=(101,102,103,104,105,106,107)
print("Original Tuple:")
n=len(x)
for i in range(n):
    print(i,":=",x[i])
print("From 1st Position To 4th Position:")
```



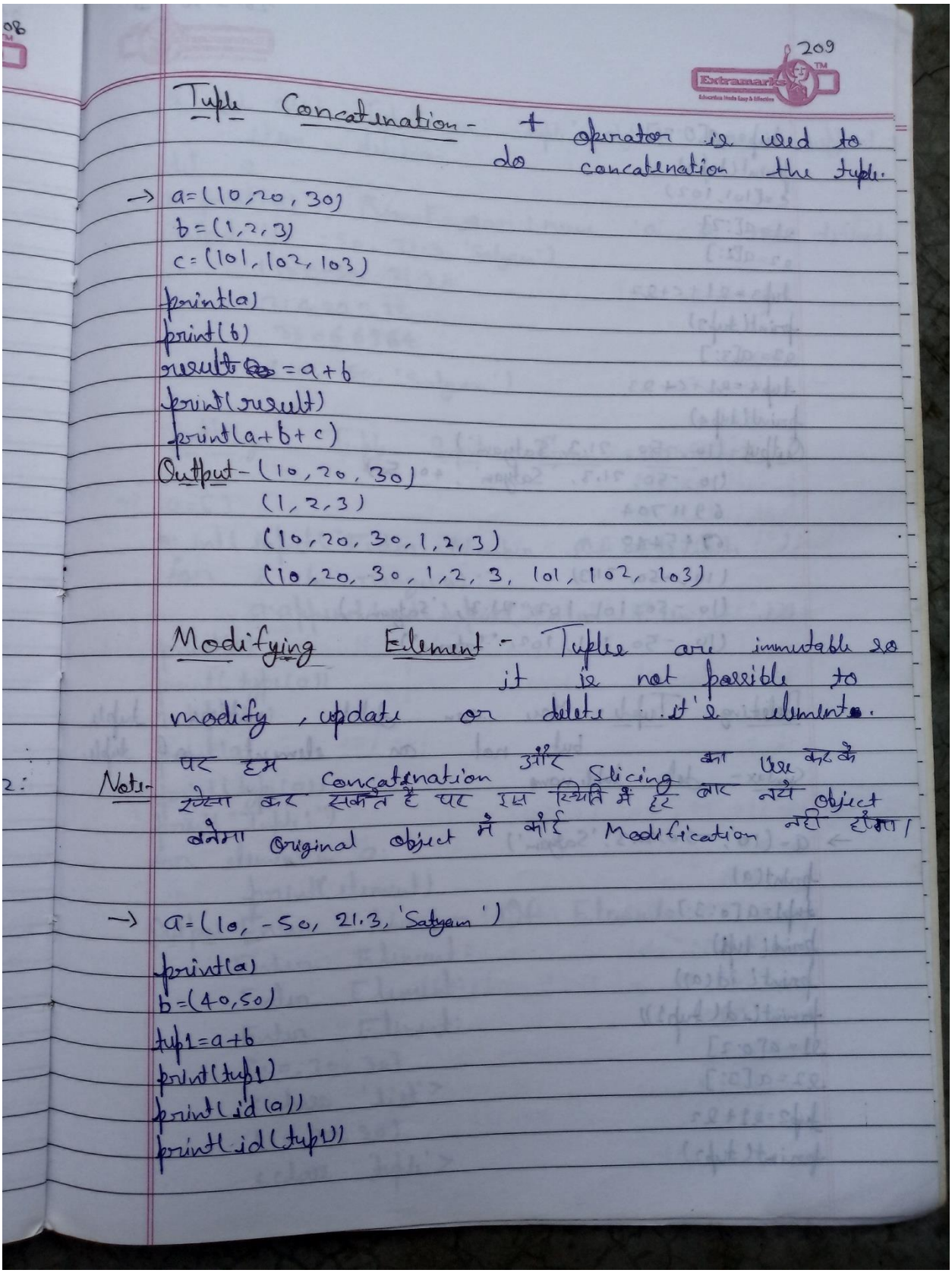
```
a=x[1:5]
for i in a:
    print(i)
print("From 0th Position To Last Position:")
b=x[0:]
for i in b:
    print(i)
print("From 0th Position To 4th Position:")
c=x[:5]
for i in c:
    print(i)
print("Last 4 Elements:")
d=x[-4:]
for i in d:
    print(i)
print("From 0th Position To 6th Position
                        Stride 2:")
e=x[0:7:2]
for i in e:
    print(i)
print("Last 5 Elements with [-5:(-3)]=2
        Elements Towards Right:")
f=x[-5:3]
print(f)
for i in f:
    print(i)
```

Output- Original Tuple:

```
0 = 101
1 = 102
2 = 103
3 = 104
4 = 105
5 = 106
6 = 107
From 1st Position To 4th Position:
102
103
104
```



105
From 0th Position To Last Position:
101
102
103
104
105
106
107
From 0th Position To 4th Position:
101
102
103
104
105
Last 4 Elements:
104
105
106
107
From 0th Position To 8th Position Stride 2:
101
103
105
107
Last 5 Elements With [-5 (-3)] = 2 Elements Towards Right:
(103, 104)
103
104



209

Tuple Concatenation - + operator is used to do concatenation the tuple.

→
```
a=(10,20,30)
b=(1,2,3)
c=(101,102,103)
print(a)
print(b)
result=a+b
print(result)
print(a+b+c)
```
Output - (10,20,30)
(1,2,3)
(10,20,30,1,2,3)
(10,20,30,1,2,3,101,102,103)

Modifying Element - Tuples are immutable so it is not possible to modify, update or delete it's elements.

Note - पर हम Concatenation और Slicing का use करके देखा कर सकते है पर इस स्थिति में हर बार नया object बनेगा Original object में कोई Modification नहीं होगा।

→
```
a=(10,-50,21.3,'Satyam')
print(a)
b=(40,50)
tup1=a+b
print(tup1)
print(id(a))
print(id(tup1))
```

26-3-20 26

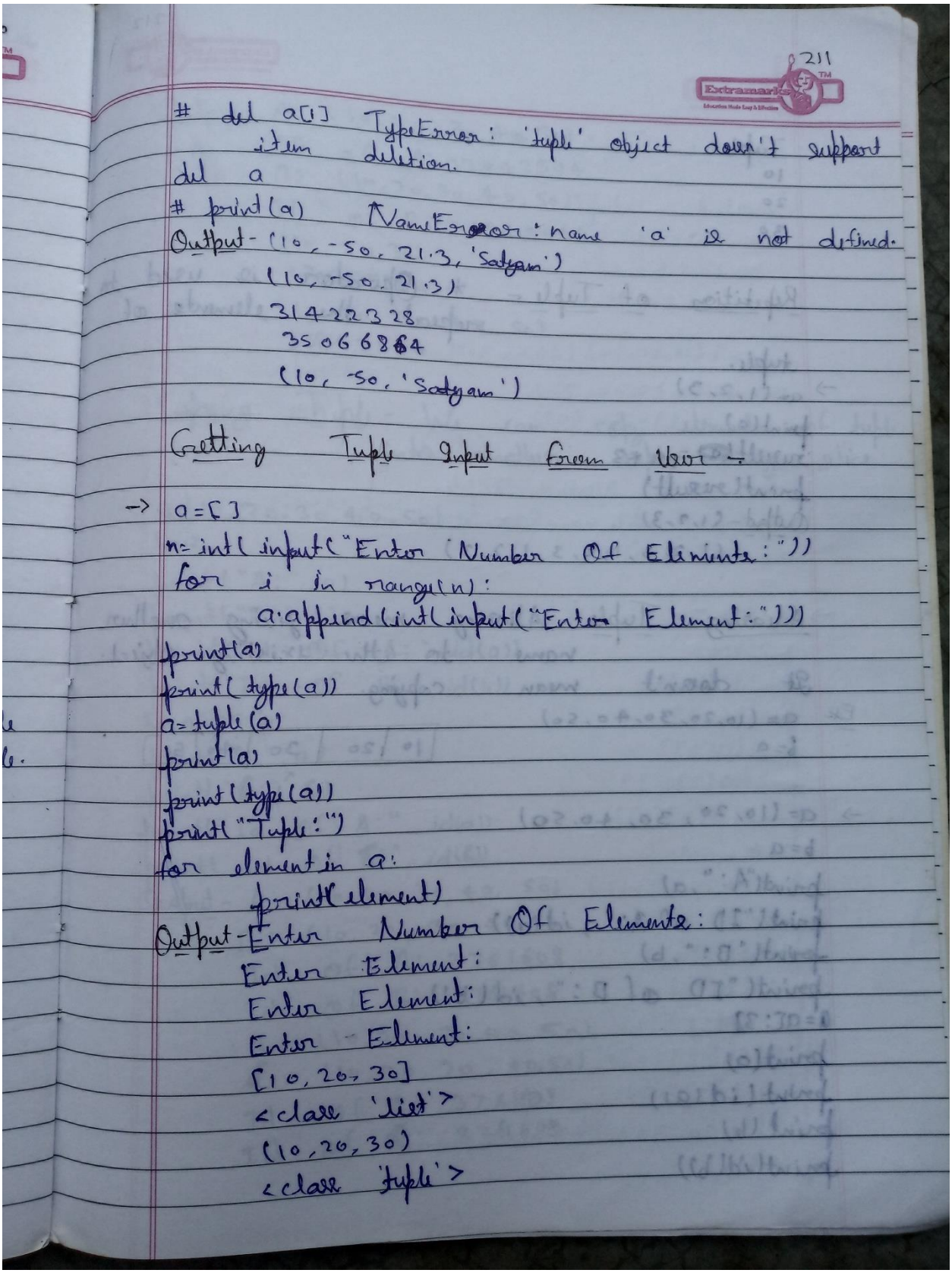
```
tup2=a[0:3]
print(tup2)
c=(101,102)
a1=a[:2]
a2=a[2:]
tup3=a1+c+a2
print(tup3)
a3=a[3:]
tup4=a1+c+a3
print(tup4)
```

Output- (10, -50, 21.3, 'Satyam')
(10, -50, 21.3, 'Satyam', 40, 50)
6911704
6745448
(10, -50, 21.3)
(10, -50, 101, 102, 21.3, 'Satyam')
(10, -50, 101, 102, 'Satyam')

Deleting Tuple- You can delete entire tuple but not an element of tuple.

Syntax- del tuple_name

→
```
a=(10,-50,21.3,'Satyam')
print(a)
tup1=a[0:3]
print(tup1)
print(id(a))
print(id(tup1))
a1=a[0:2]
a2=a[3:]
tup2=a1+a2
print(tup2)
```



```
# del a[1] TypeError: 'tuple' object doesn't support
    item deletion.
del a
# print(a)  NameError: name 'a' is not defined.
Output- (10, -50, 21.3, 'Satyam')
        (10, -50, 21.3)
        31422328
        35066884
        (10, -50, 'Satyam')

Getting Tuple Input From User :-

-> a=[]
n= int(input("Enter Number Of Elements:"))
for i in range(n):
    a.append(int(input("Enter Element:")))
print(a)
print(type(a))
a=tuple(a)
print(a)
print(type(a))
print("Tuple:")
for element in a:
    print(element)
Output- Enter Number Of Elements:
        Enter Element:
        Enter Element:
        Enter Element:
        [10, 20, 30]
        <class 'list'>
        (10, 20, 30)
        <class 'tuple'>
```

212

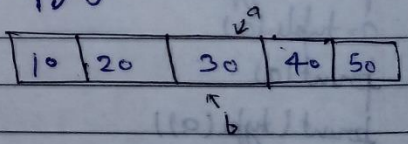
Tuple:
10
20
30

Repetition of Tuple - * Operator is used to repeat the elements of tuple.

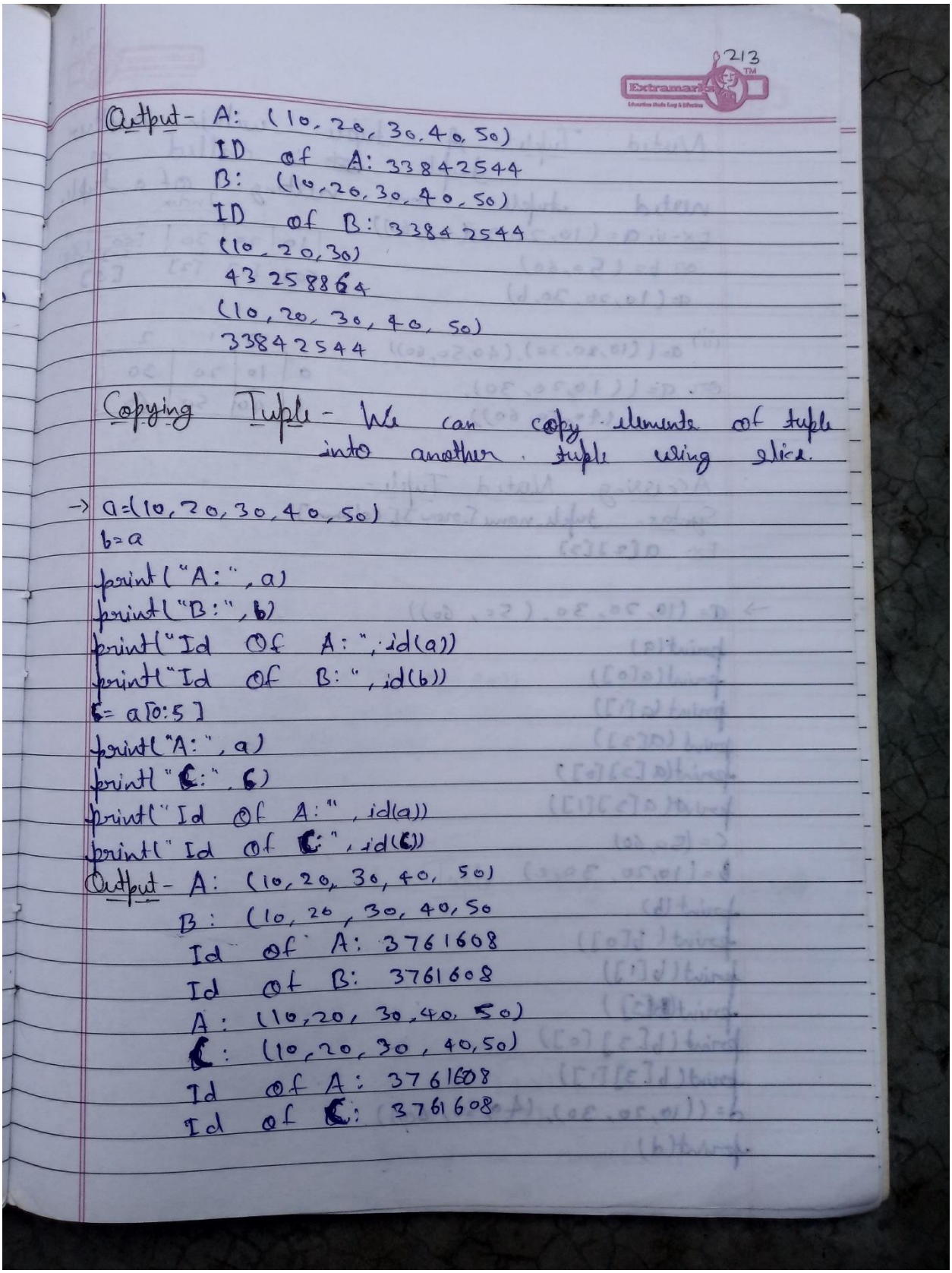
→
```
a=(1,2,3)
print(a)
result = a*3
print(result)
```
Output - (1,2,3)
(1,2,3,1,2,3,1,2,3)

Aliasing Tuple - Aliasing means giving another name to the existing object. It doesn't mean copying.

Ex- a=(10,20,30,40,50)
b=a



→
```
a=(10,20,30,40,50)
b=a
print("A:",a)
print("ID of A:",id(a))
print("B:",b)
print("ID of B:",id(b))
a=a[:3]
print(a)
print(id(a))
print(b)
print(id(b))
```

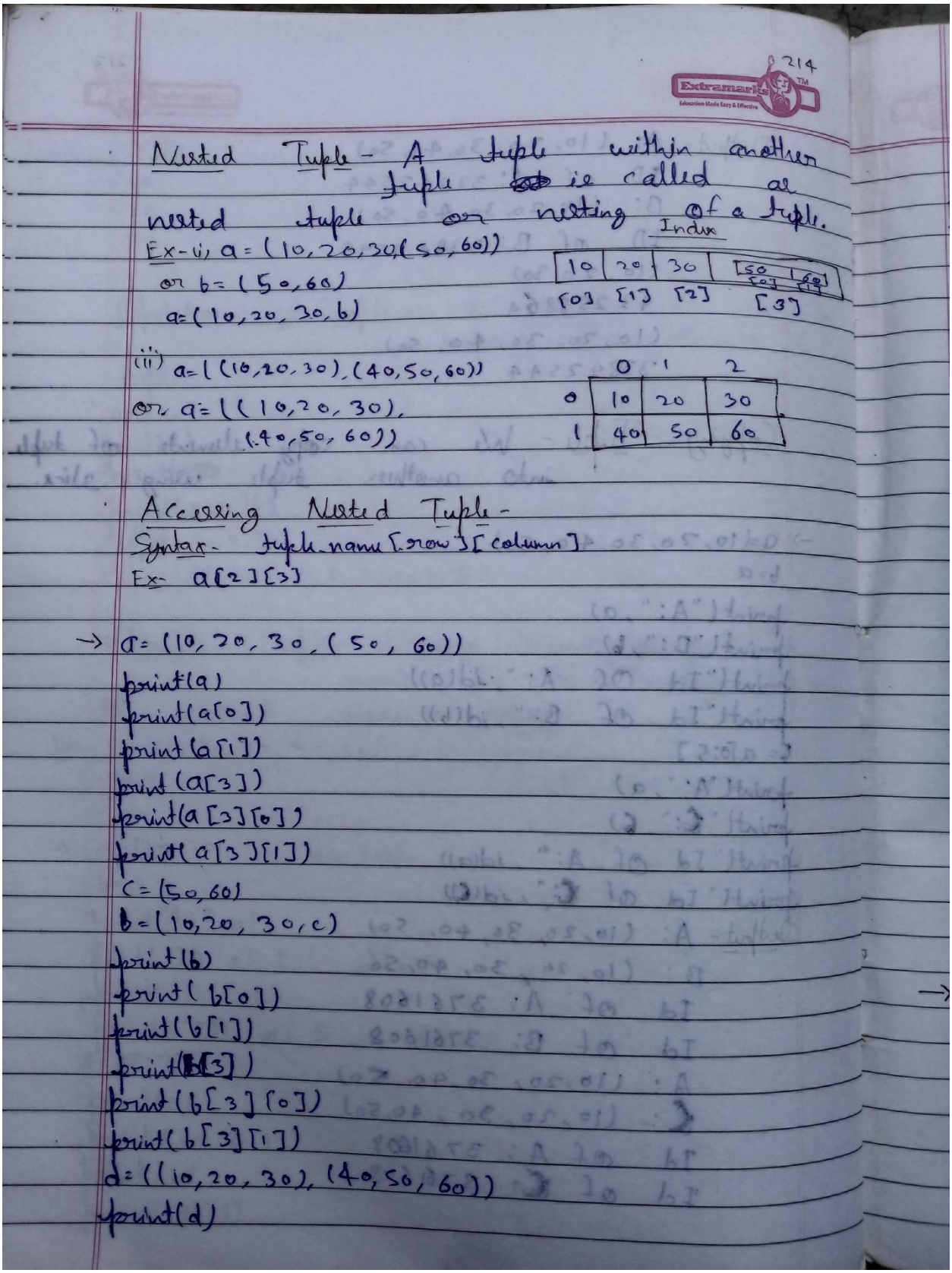


Output- A: (10, 20, 30, 40, 50)
ID of A: 33842544
B: (10, 20, 30, 40, 50)
ID of B: 33842544
(10, 20, 30)
43258864
(10, 20, 30, 40, 50)
33842544

Copying Tuple - We can copy elements of tuple into another tuple using slice.

```
a=(10,20,30,40,50)
b=a
print("A:",a)
print("B:",b)
print("Id Of A:",id(a))
print"Id Of B:",id(b))
c= a[0:5]
print("A:",a)
print"C:",c)
print("Id Of A:",id(a))
print(" Id Of C:",id(c))
```

Output- A: (10, 20, 30, 40, 50)
B: (10, 20, 30, 40, 50
Id of A: 3761608
Id of B: 3761608
A: (10, 20, 30, 40, 50)
C: (10, 20, 30, 40, 50)
Id of A: 3761608
Id of C: 3761608



Nested Tuple - A tuple within another tuple is called as nested tuple or nesting of a tuple.

Ex- (i) a = (10, 20, 30, (50, 60))
or b = (50, 60)
a = (10, 20, 30, b)

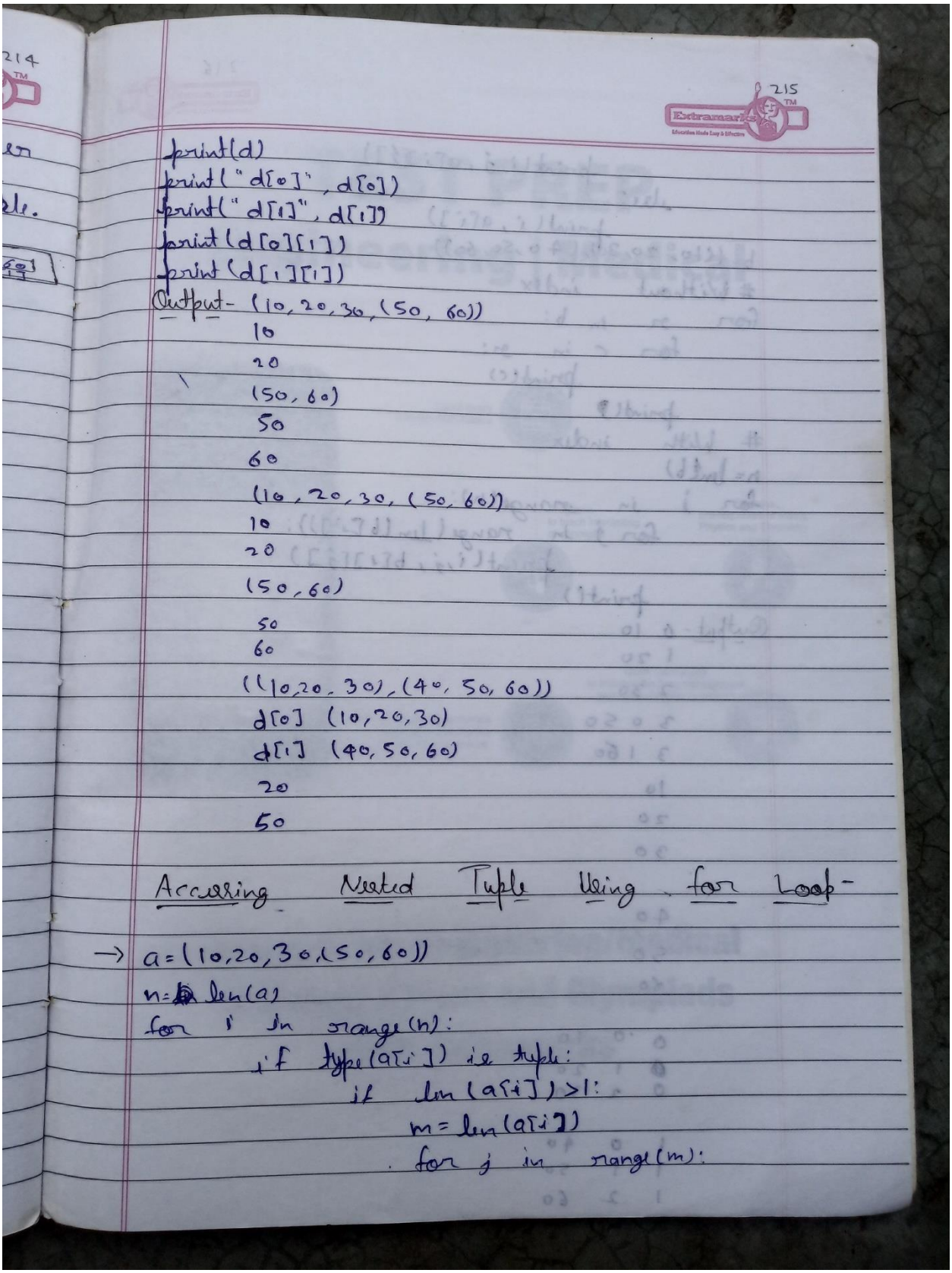
10	20	30	[50 | 60]
[0]	[1]	[2]	[3]

(ii) a = ((10, 20, 30), (40, 50, 60))
or a = ((10, 20, 30),
(40, 50, 60))

	0	1	2
0	10	20	30
1	40	50	60

Accessing Nested Tuple -
Syntax - tuple_name[row][column]
Ex- a[2][3]

```
a = (10, 20, 30, (50, 60))
print(a)
print(a[0])
print(a[1])
print(a[3])
print(a[3][0])
print(a[3][1])
c = (50, 60)
b = (10, 20, 30, c)
print(b)
print(b[0])
print(b[1])
print(b[3])
print(b[3][0])
print(b[3][1])
d = ((10, 20, 30), (40, 50, 60))
print(d)
```

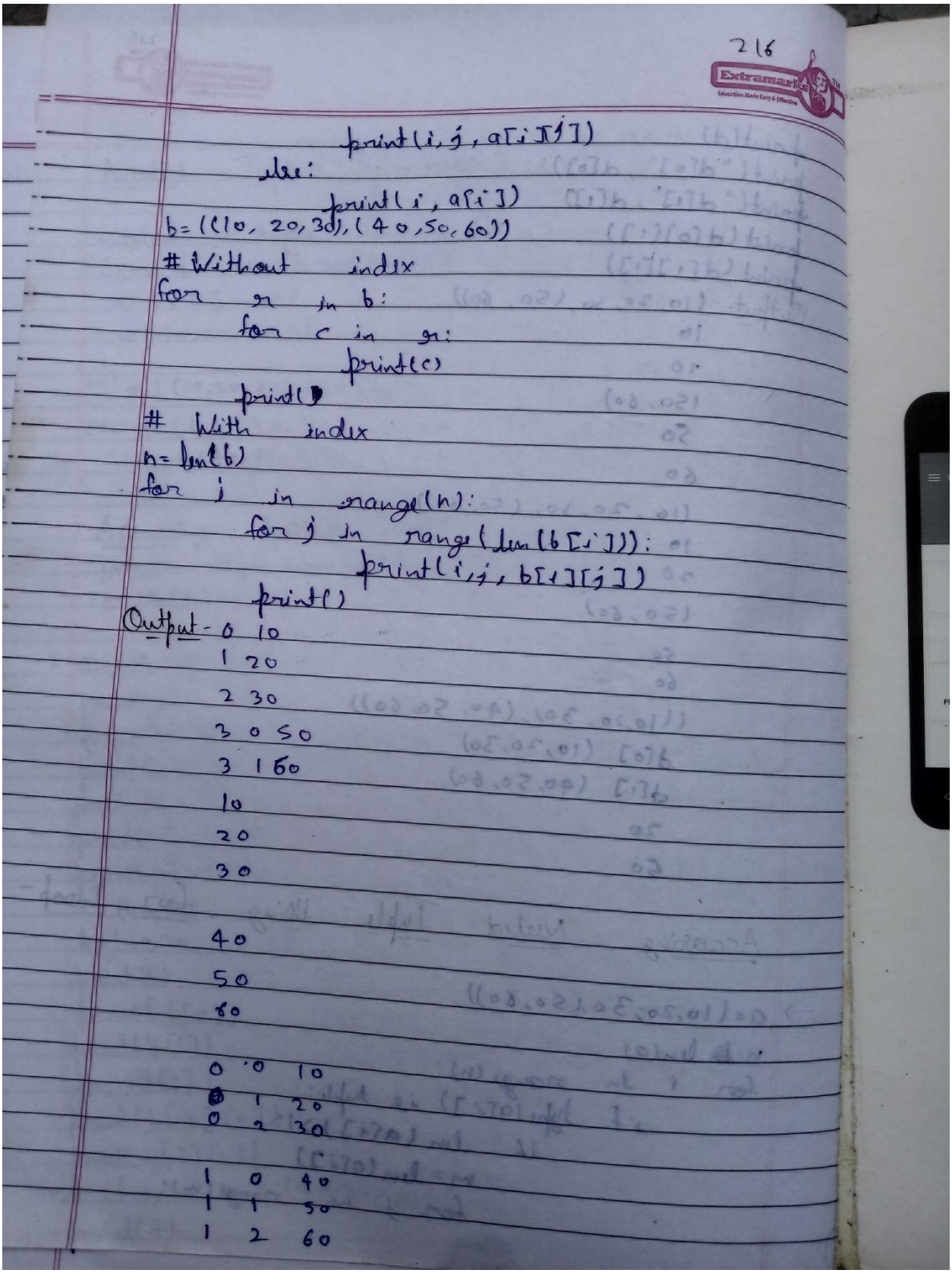


```
print(d)
print("d[0]", d[0])
print("d[1]", d[1])
print(d[0][1])
print(d[1][1])
```

Output-
```
(10, 20, 30, (50, 60))
10
20
(50, 60)
50
60
(10, 20, 30, (50, 60))
10
20
(50, 60)
50
60
((10,20, 30), (40, 50, 60))
d[0] (10, 20, 30)
d[1] (40, 50, 60)
20
50
```

Accessing Nested Tuple Using for Loop-

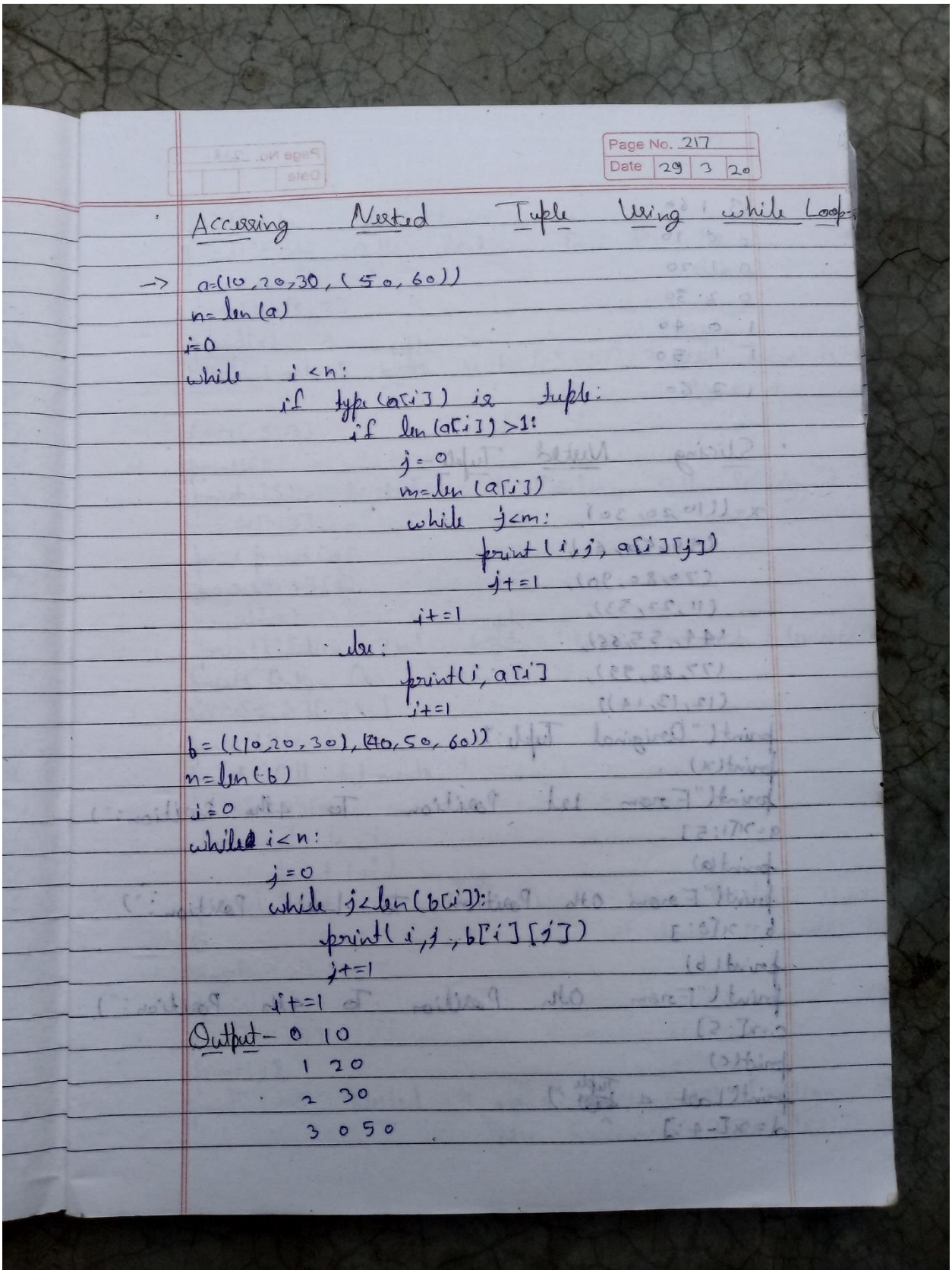
→
```
a=(10,20,30,(50,60))
n=len(a)
for i in range(n):
    if type(a[i]) is tuple:
        if len(a[i])>1:
            m=len(a[i])
            for j in range(m):
```



```
                print(i,j,a[i][j])
        else:
                print(i,a[i])
b=((10,20,30),(40,50,60))
# Without index
for x in b:
    for c in x:
        print(c)
    print()
# With index
n=len(b)
for i in range(n):
    for j in range(len(b[i])):
        print(i,j,b[i][j])
    print()
```

Output -

```
0 10
1 20
2 30
3 0 50
3 1 60
10
20
30

40
50
60

0 0 10
0 1 20
0 2 30

1 0 40
1 1 50
1 2 60
```

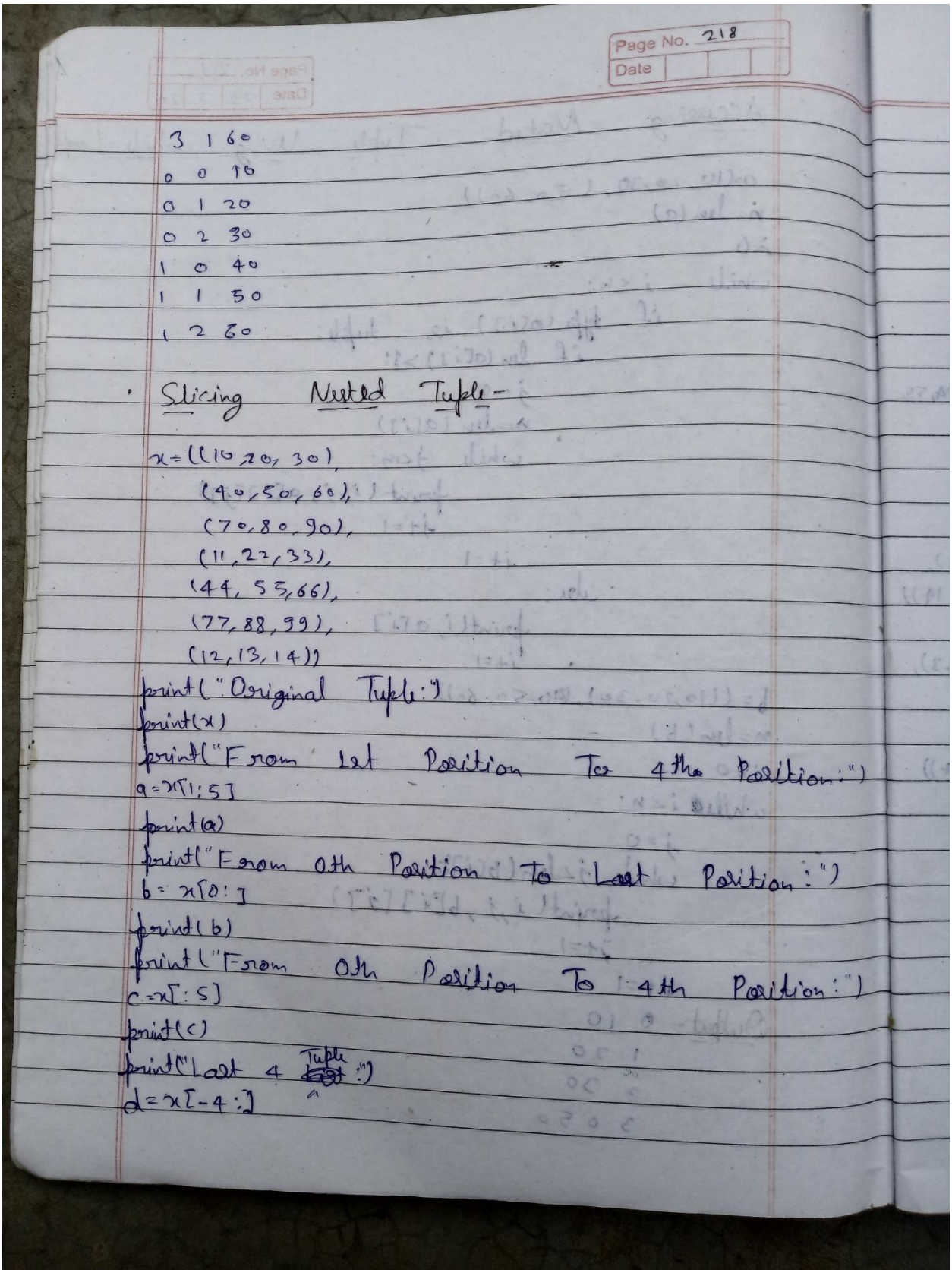


Page No. 217
Date 29 3 20

Accessing Nested Tuple using while Loop:

```
a=(10,20,30,(50,60))
n=len(a)
i=0
while i<n:
    if type(a[i]) is tuple:
        if len(a[i])>1:
            j=0
            m=len(a[i])
            while j<m:
                print(i,j,a[i][j])
                j+=1
        i+=1
    else:
        print(i,a[i])
        i+=1
b=((10,20,30),(40,50,60))
n=len(b)
i=0
while i<n:
    j=0
    while j<len(b[i]):
        print(i,j,b[i][j])
        j+=1
    i+=1
```

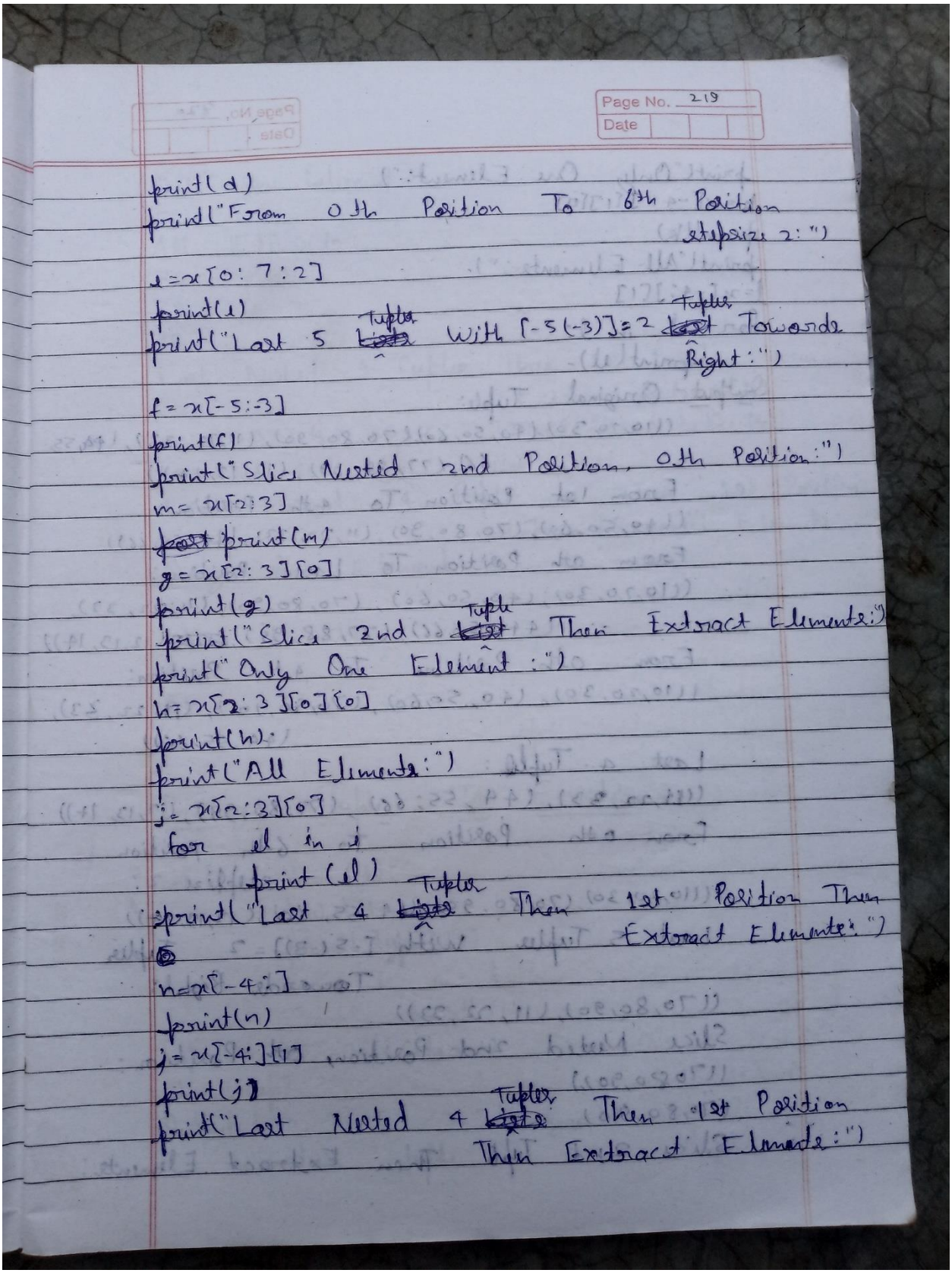
Output-
```
0 10
1 20
2 30
3 0 50
```



```
3 1 60
0 0 10
0 1 20
0 2 30
1 0 40
1 1 50
1 2 60
```

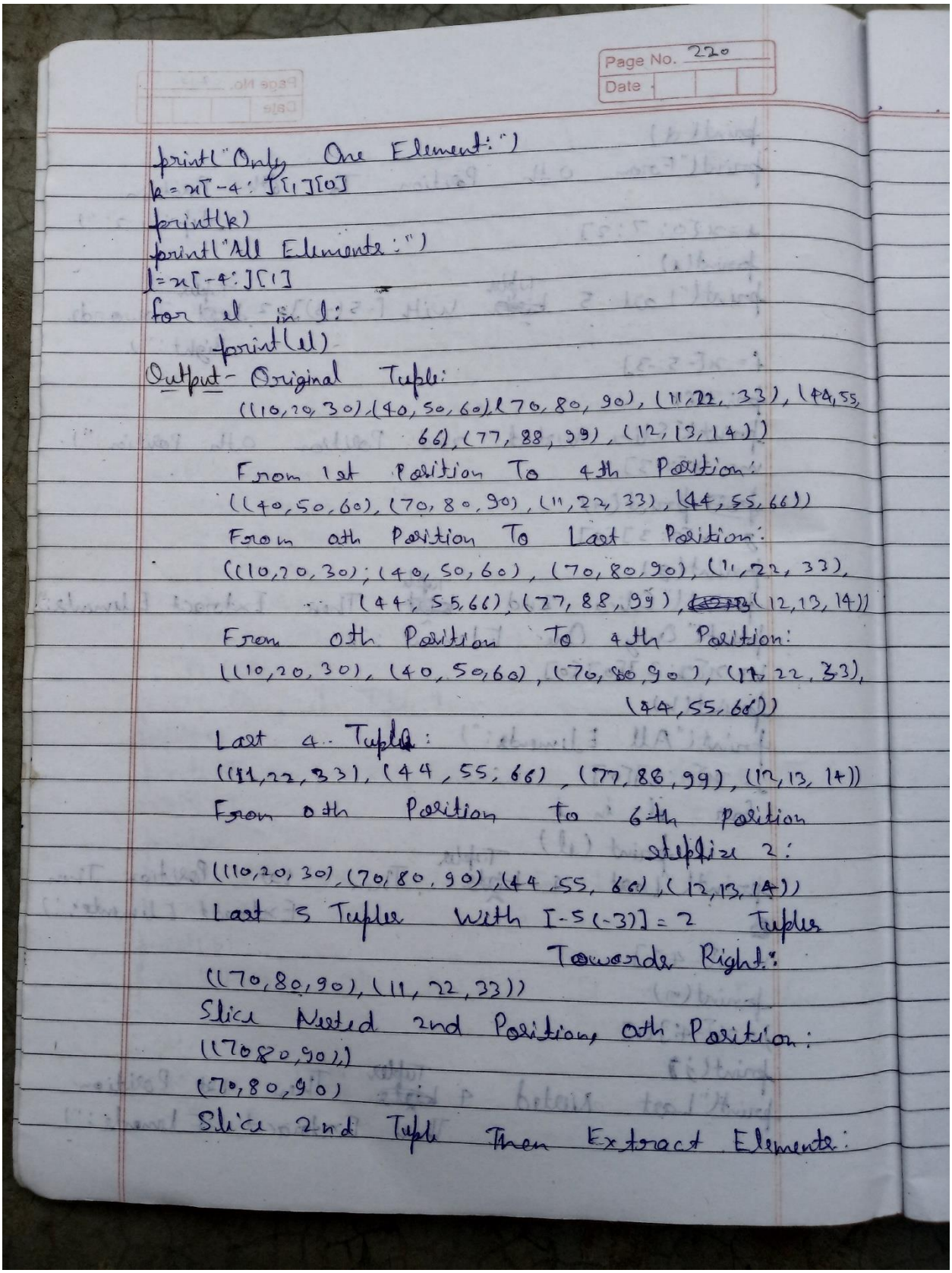
- Slicing Nested Tuple -

```
x=((10,20,30),
   (40,50,60),
   (70,80,90),
   (11,22,33),
   (44,55,66),
   (77,88,99),
   (12,13,14))
print("Original Tuple:")
print(x)
print("From 1st Position To 4th Position:")
a=x[1:5]
print(a)
print("From 0th Position To Last Position:")
b=x[0:]
print(b)
print("From 0th Position To 4th Position:")
c=x[:5]
print(c)
print("Last 4 Tuple:")
d=x[-4:]
```



Page No. 219
Date

```
print(d)
print("From 0th Position To 6th Position stepsize 2:")
d=x[0:7:2]
print(d)
print("Last 5 Tuples With [-5(-3)]=2 Towards Right:")
f=x[-5:-3]
print(f)
print("Slice Nested 2nd Position, 0th Position:")
m=x[2:3]
print(m)
g=x[2:3][0]
print(g)
print("Slice 2nd Tuple Then Extract Elements:")
print("Only One Element:")
h=x[2:3][0][0]
print(h)
print("All Elements:")
i=x[2:3][0]
for el in i
    print(el)
print("Last 4 Tuples Then 1st Position Then Extract Elements:")
n=x[-4:]
print(n)
j=x[-4:][1]
print(j)
print("Last Nested 4 Tuples Then 1st Position Then Extract Elements:")
```



Page No. 220
Date

```
print("Only One Element:")
k = x[-4:][1][0]
print(k)
print("All Elements:")
l = x[-4:][1]
for el in l:
    print(el)
```

Output - Original Tuple:
((10,20,30),(40,50,60),(70,80,90),(11,22,33),(44,55,66),(77,88,99),(12,13,14))

From 1st Position To 4th Position:
((40,50,60),(70,80,90),(11,22,33),(44,55,66))

From 0th Position To Last Position:
((10,20,30),(40,50,60),(70,80,90),(11,22,33),(44,55,66),(77,88,99),(12,13,14))

From 0th Position To 4th Position:
((10,20,30),(40,50,60),(70,80,90),(11,22,33),(44,55,66))

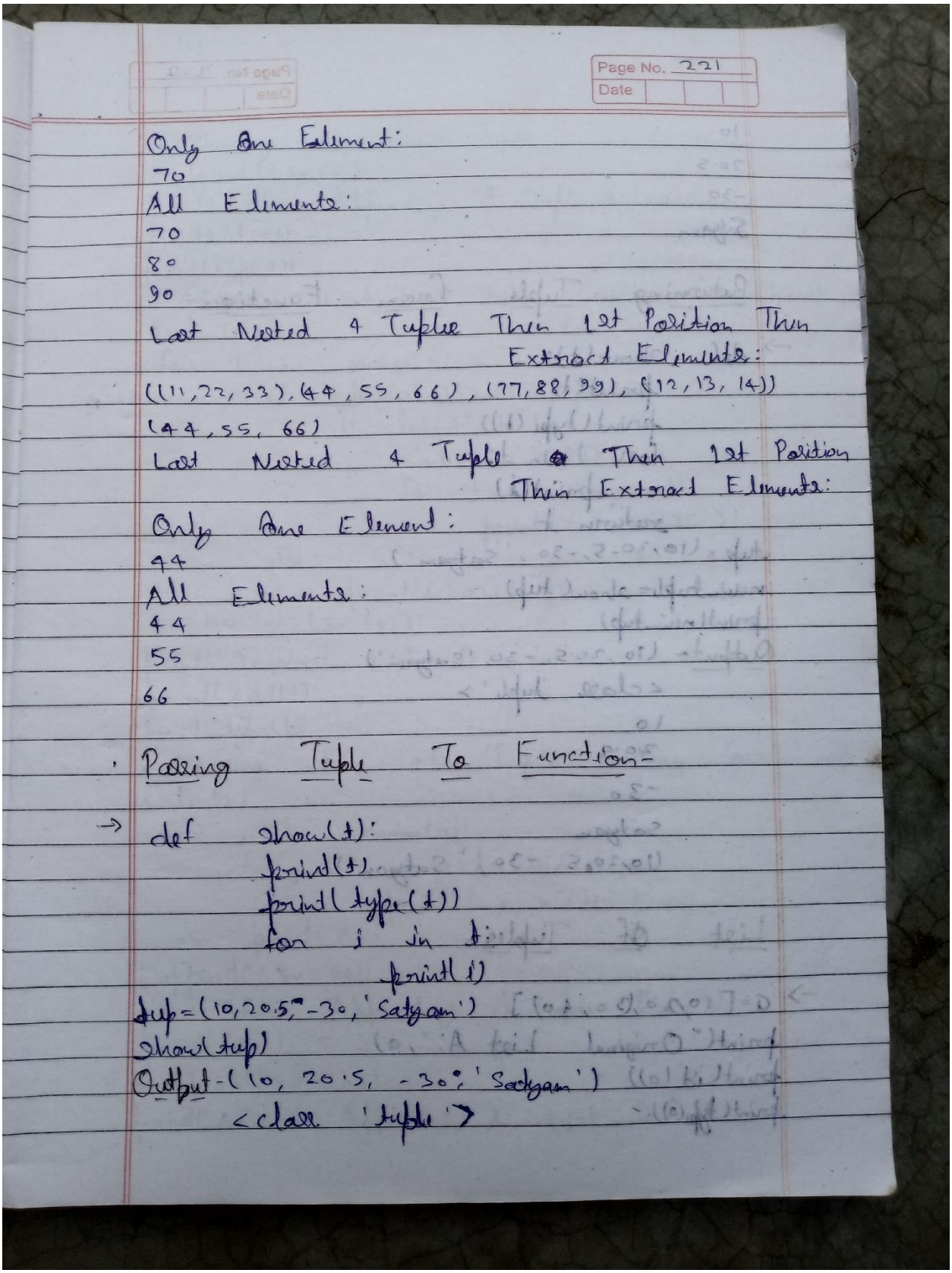
Last 4 Tuple:
((11,22,33),(44,55,66),(77,88,99),(12,13,14))

From 0th Position to 6th Position step size 2:
((10,20,30),(70,80,90),(44,55,66),(12,13,14))

Last 5 Tuples With [-5(-3)] = 2 Tuples Towards Right:
((70,80,90),(11,22,33))

Slice Nested 2nd Position, 0th Position:
((70,80,90),)
(70,80,90)

Slice 2nd Tuple Then Extract Elements:

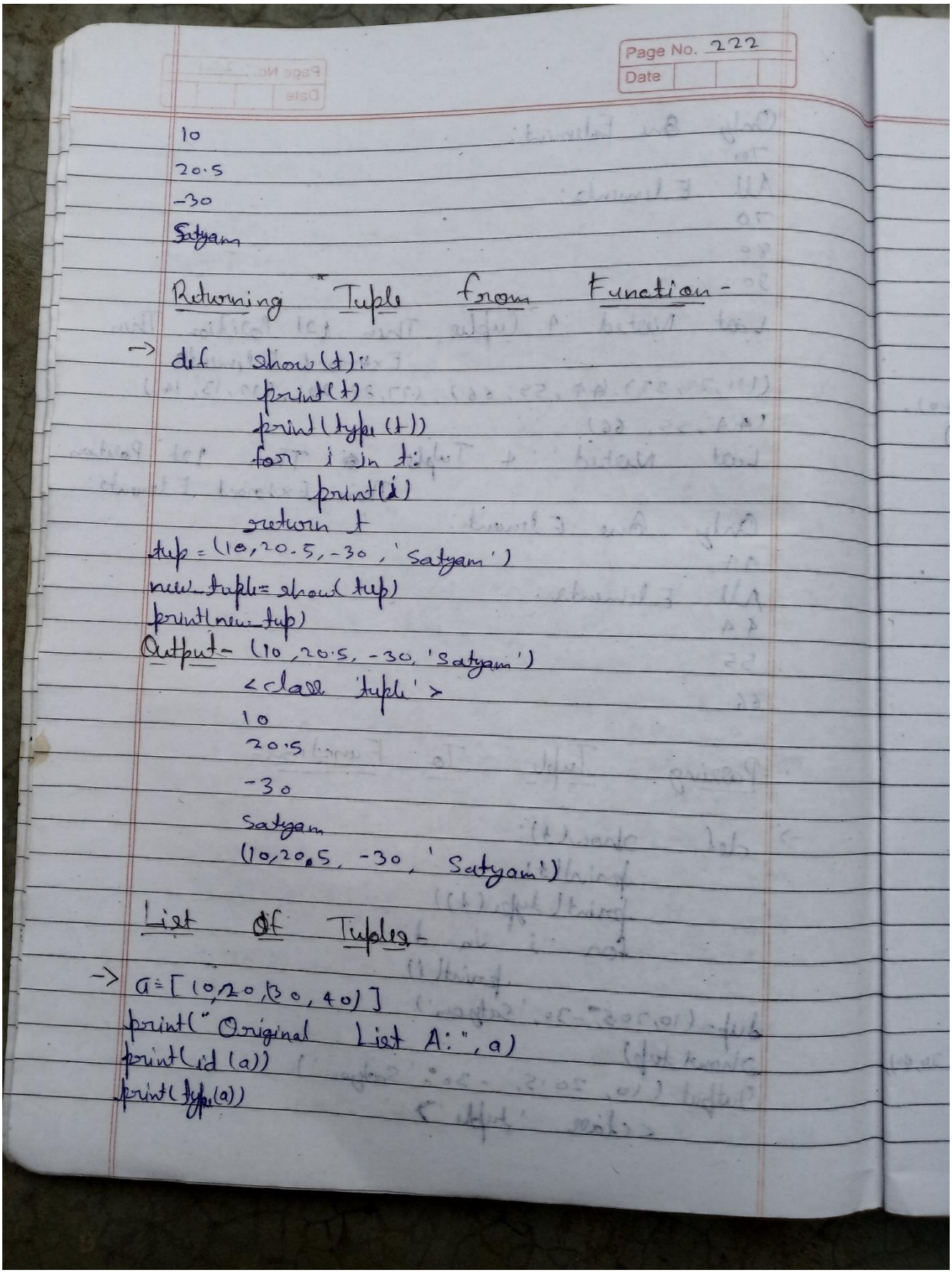


Page No. 221
Date

Only One Element:
70
All Elements:
70
80
90
Last Nested 4 Tuples Then 1st Position Then Extract Elements:
((11,22,33),44,55,66),(77,88,99),(12,13,14))
(44,55,66)
Last Nested 4 Tuple Then 1st Position Then Extract Elements:
Only One Element:
44
All Elements:
44
55
66

Passing Tuple To Function-

→
```
def show(t):
    print(t)
    print(type(t))
    for i in t:
        print(i)
tup=(10,20.5,-30,'Satyam')
show(tup)
```
Output-(10, 20.5, -30, 'Satyam')
<class 'tuple'>



Page No. 222
Date

10
20.5
-30
Satyam

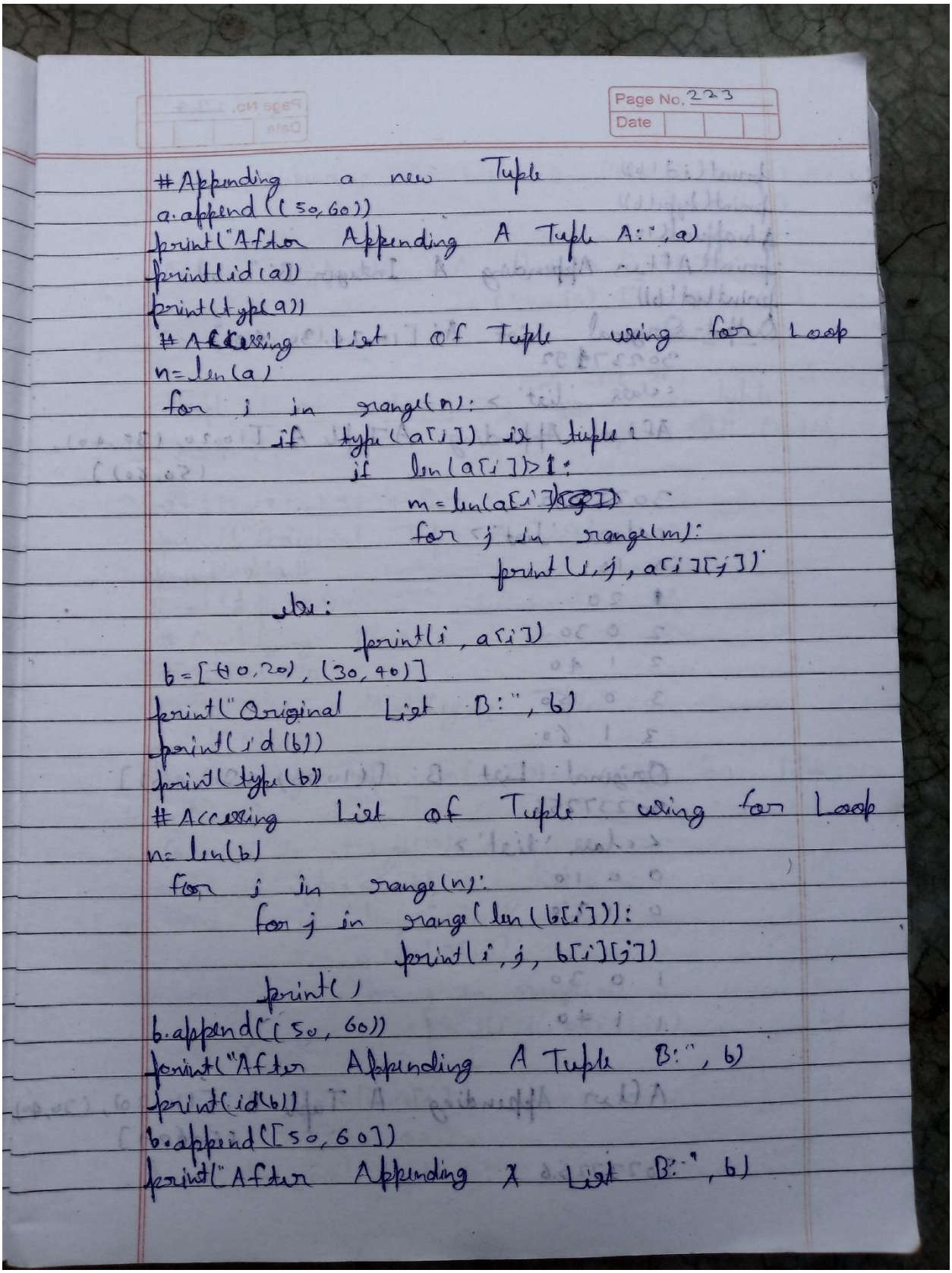
Returning Tuple from Function -

```
-> def show(t):
       print(t)
       print(type(t))
       for i in t:
           print(i)
       return t
tup = (10, 20.5, -30, 'Satyam')
new_tuple = show(tup)
print(new_tup)
```

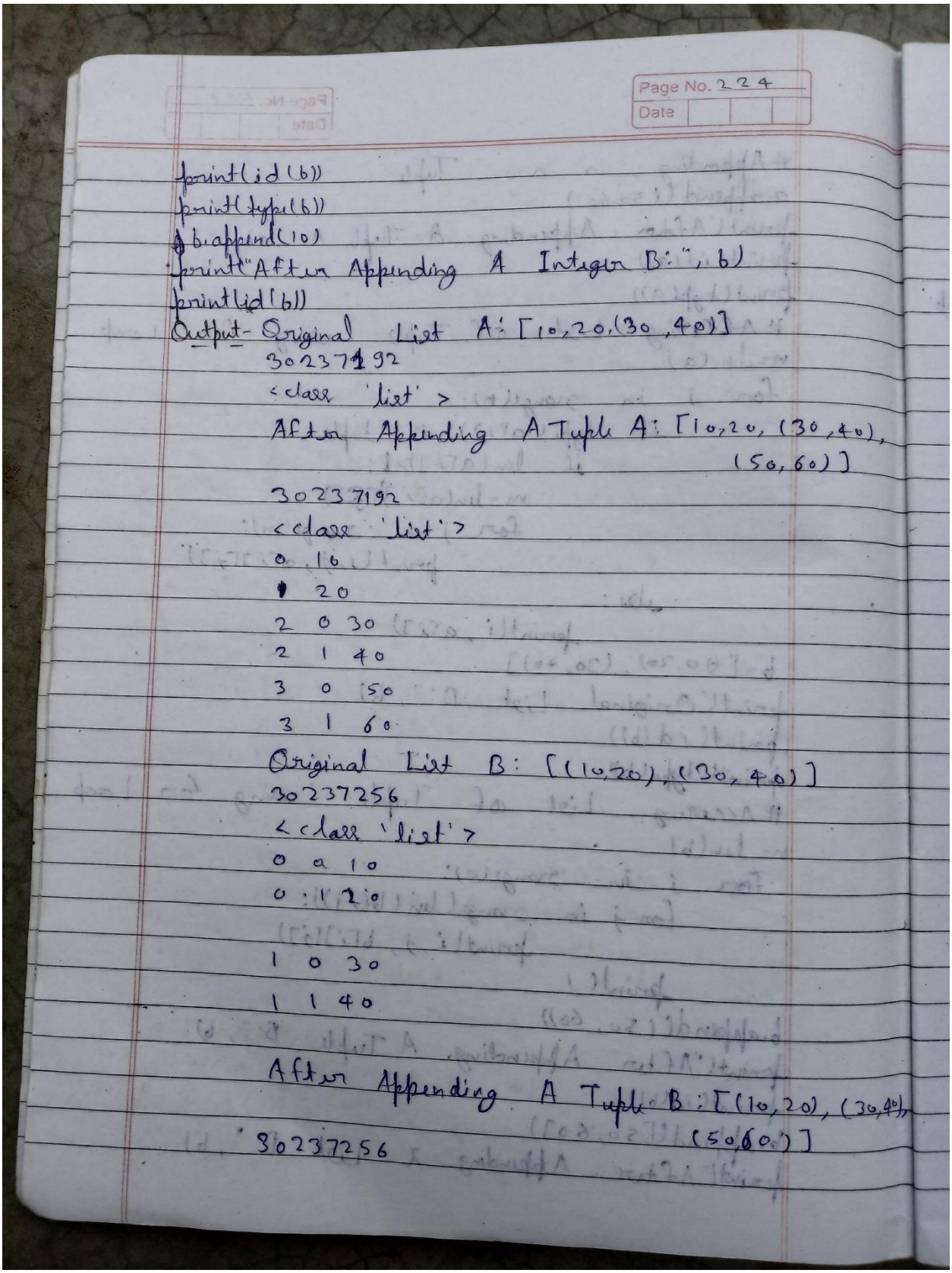
Output - (10, 20.5, -30, 'Satyam')
<class 'tuple'>
10
20.5
-30
Satyam
(10, 20.5, -30, 'Satyam')

List Of Tuples -

```
-> a = [(10, 20, 30, 40)]
print("Original List A:", a)
print(id(a))
print(type(a))
```



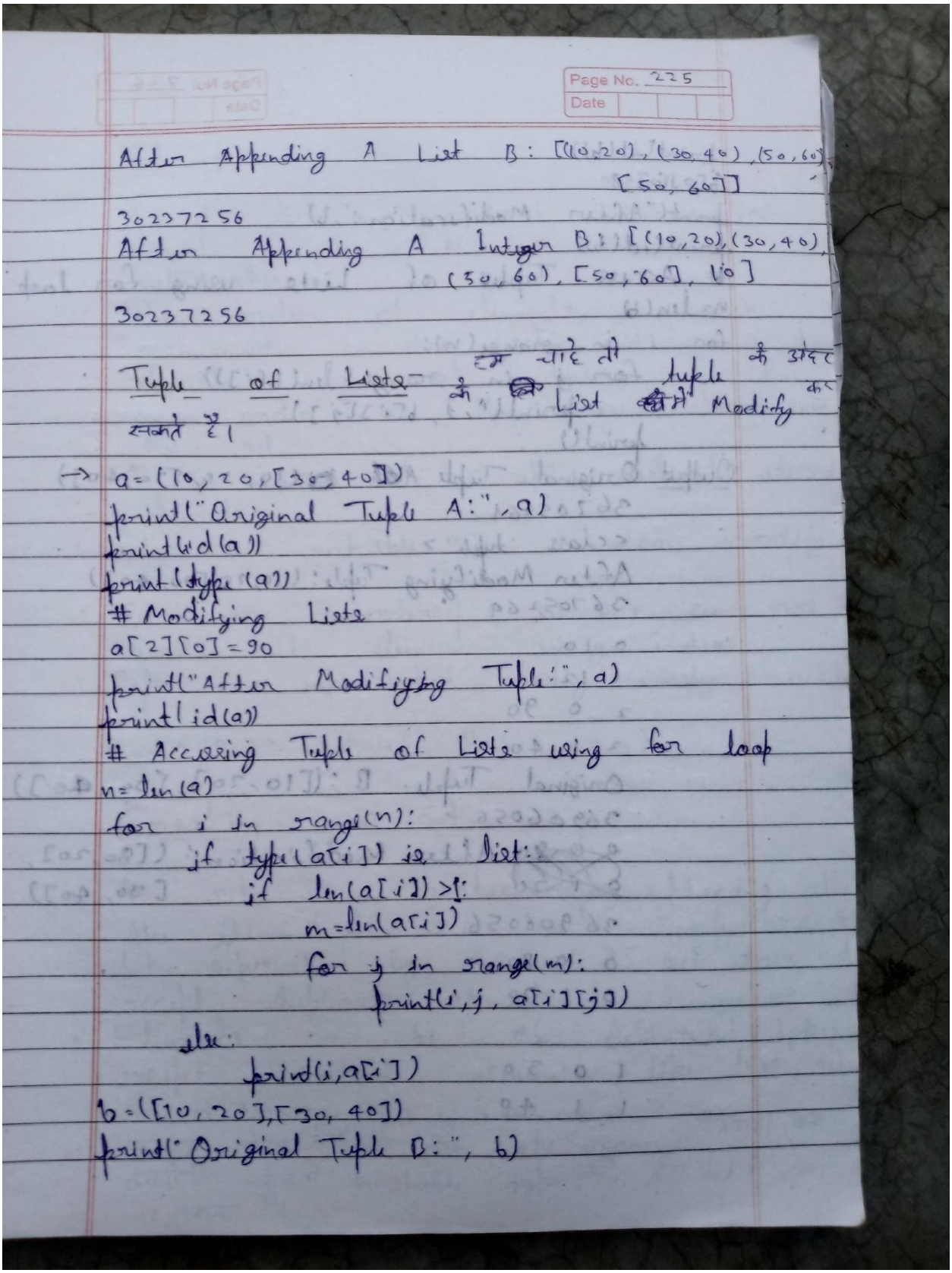
```
#Appending a new Tuple
a.append((50,60))
print("After Appending A Tuple A:",a)
print(id(a))
print(type(a))
#Accessing List of Tuple using for Loop
n=len(a)
for i in range(n):
    if type(a[i]) is tuple:
        if len(a[i])>1:
            m=len(a[i])
            for j in range(m):
                print(i,j,a[i][j])
    else:
        print(i,a[i])
b=[(10,20),(30,40)]
print("Original List B:",b)
print(id(b))
print(type(b))
#Accessing List of Tuple using for Loop
n=len(b)
for i in range(n):
    for j in range(len(b[i])):
        print(i,j,b[i][j])
    print()
b.append((50,60))
print("After Appending A Tuple B:",b)
print(id(b))
b.append([50,60])
print("After Appending A List B:",b)
```



```
print(id(b))
print(type(b))
b.append(10)
print("After Appending A Integer B:", b)
print(id(b))
```

Output- Original List A: [10,20,(30,40)]

```
30237192
<class 'list'>
After Appending A Tuple A: [10,20,(30,40),
                            (50,60)]
30237192
<class 'list'>
0 10
1 20
2 0 30
2 1 40
3 0 50
3 1 60
Original List B: [(10,20),(30,40)]
30237256
<class 'list'>
0 0 10
0 1 20

1 0 30
1 1 40

After Appending A Tuple B: [(10,20),(30,40),
                            (50,60)]
30237256
```

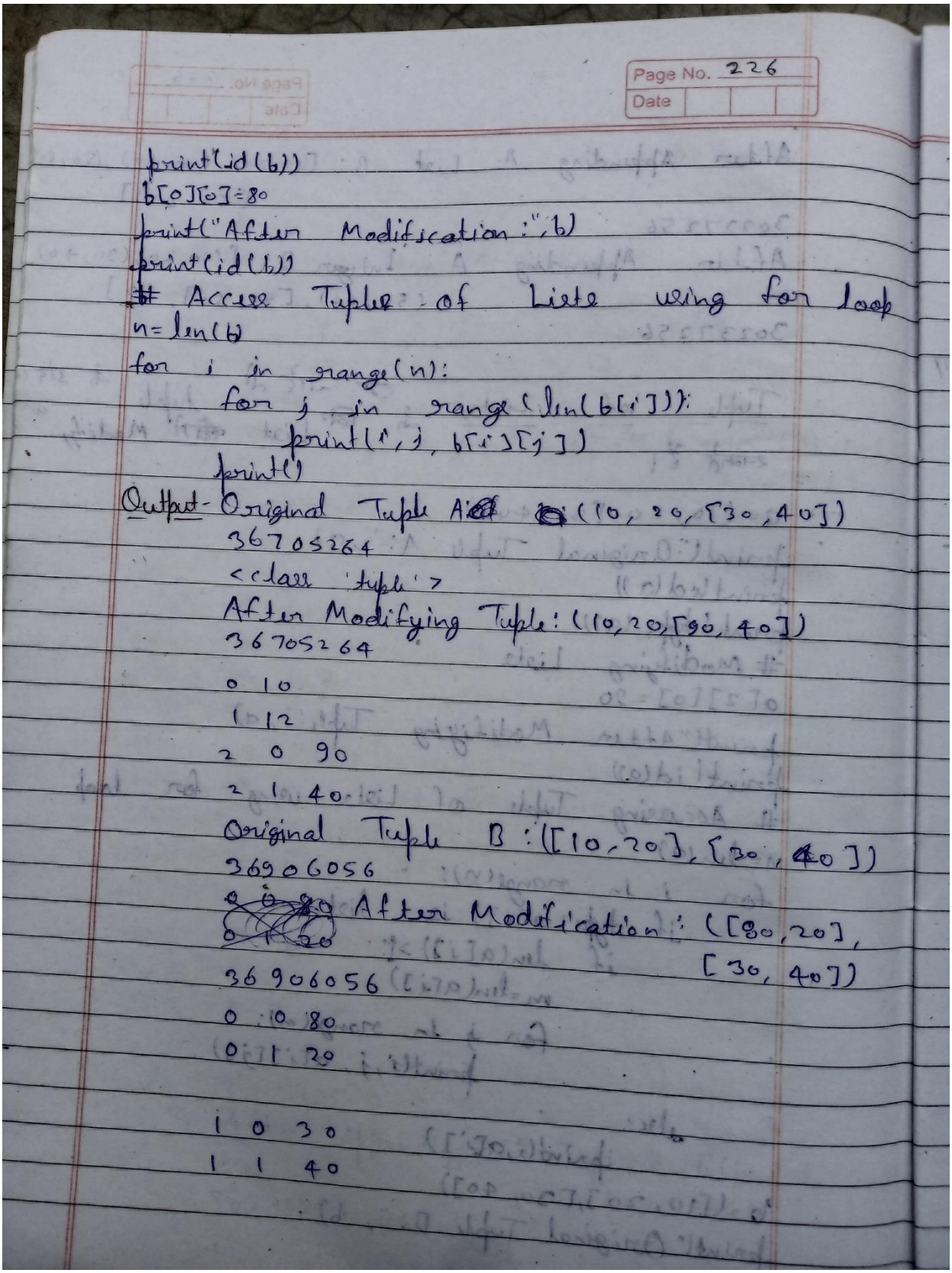


Page No. 225
Date

After Appending A List B: [(10,20), (30,40), (50,60), [50,60]]
30237256
After Appending A Integer B: [(10,20), (30,40), (50,60), [50,60], 10]
30237256

Tuple of Lists- में चाहे तो tuple के अंदर List को Modify कर सकते हैं।

→
```
a=(10,20,[30,40])
print("Original Tuple A:",a)
print(id(a))
print(type(a))
# Modifying Lists
a[2][0]=90
print("After Modifying Tuple:",a)
print(id(a))
# Accessing Tuple of Lists using for loop
n=len(a)
for i in range(n):
    if type(a[i]) is list:
        if len(a[i])>1:
            m=len(a[i])
            for j in range(m):
                print(i,j,a[i][j])
    else:
        print(i,a[i])
b=([10,20],[30,40])
print("Original Tuple B:",b)
```



```
print(id(b))
b[0][0]=80
print("After Modification:",b)
print(id(b))
# Access Tuple of Lists using for loop
n=len(b)
for i in range(n):
    for j in range(len(b[i])):
        print(i,j,b[i][j])
    print()
```

Output- Original Tuple A: (10, 20, [30, 40])

```
36705264
<class 'tuple'>
After Modifying Tuple: (10, 20, [90, 40])
36705264
0 10
1 12
2 0 90
2 1 40
```

Original Tuple B: ([10, 20], [30, 40])

```
36906056
```

After Modification: ([80, 20], [30, 40])

```
36906056
0 0 80
0 1 20

1 0 30
1 1 40
```

Page No. 227

Set

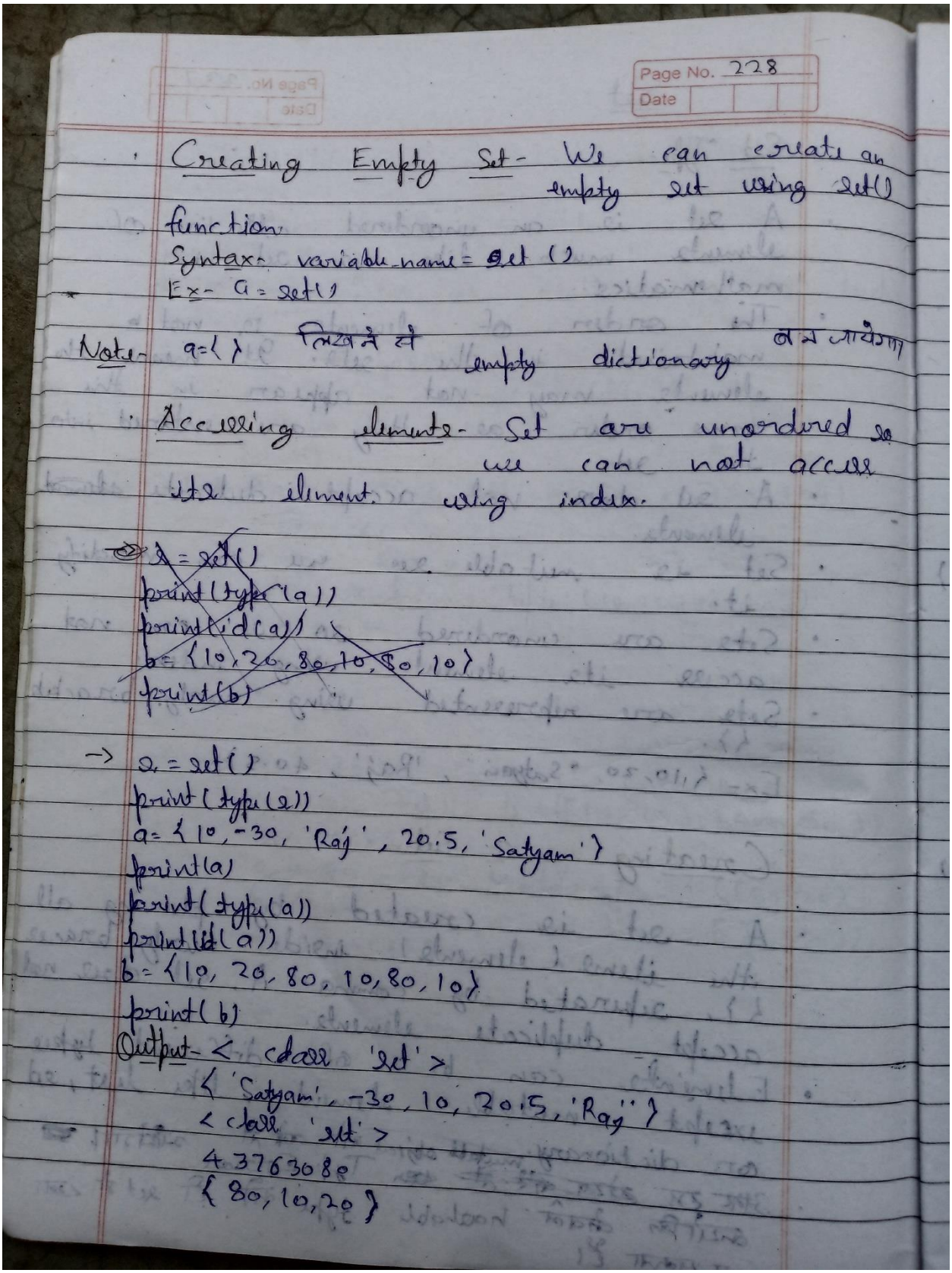
Set Type-

- A set is an unordered collection of elements much like a set in mathematics.
- The order of elements is not maintained in the sets. It means the elements may not appear in the same order as they are entered into the set.
- A set does not accept dulicate elements.
- Set is mutable so we can modify it.
- Sets are unordered so we can not access its element using index.
- Sets are represented using curly brackets {}.

Ex- {10, 20, "Satyam", 'Raj', 40.5}

Creating a Set-

- A set is created by placing all the items (elements) inside curly braces {}, separated by comma. A set does not accept duplicate elements.
- Elements can be of different types except mutable element, like list, set or dictionary.
- अगर हम mutable object को रखे तो TypeError आयेगा। क्योंकि केवल hashable type को ही set मे रखा जा सकता है।



Page No. 228
Date

- Creating Empty Set - We can create an empty set using set() function.

Syntax - variable_name = set()

Ex - a = set()

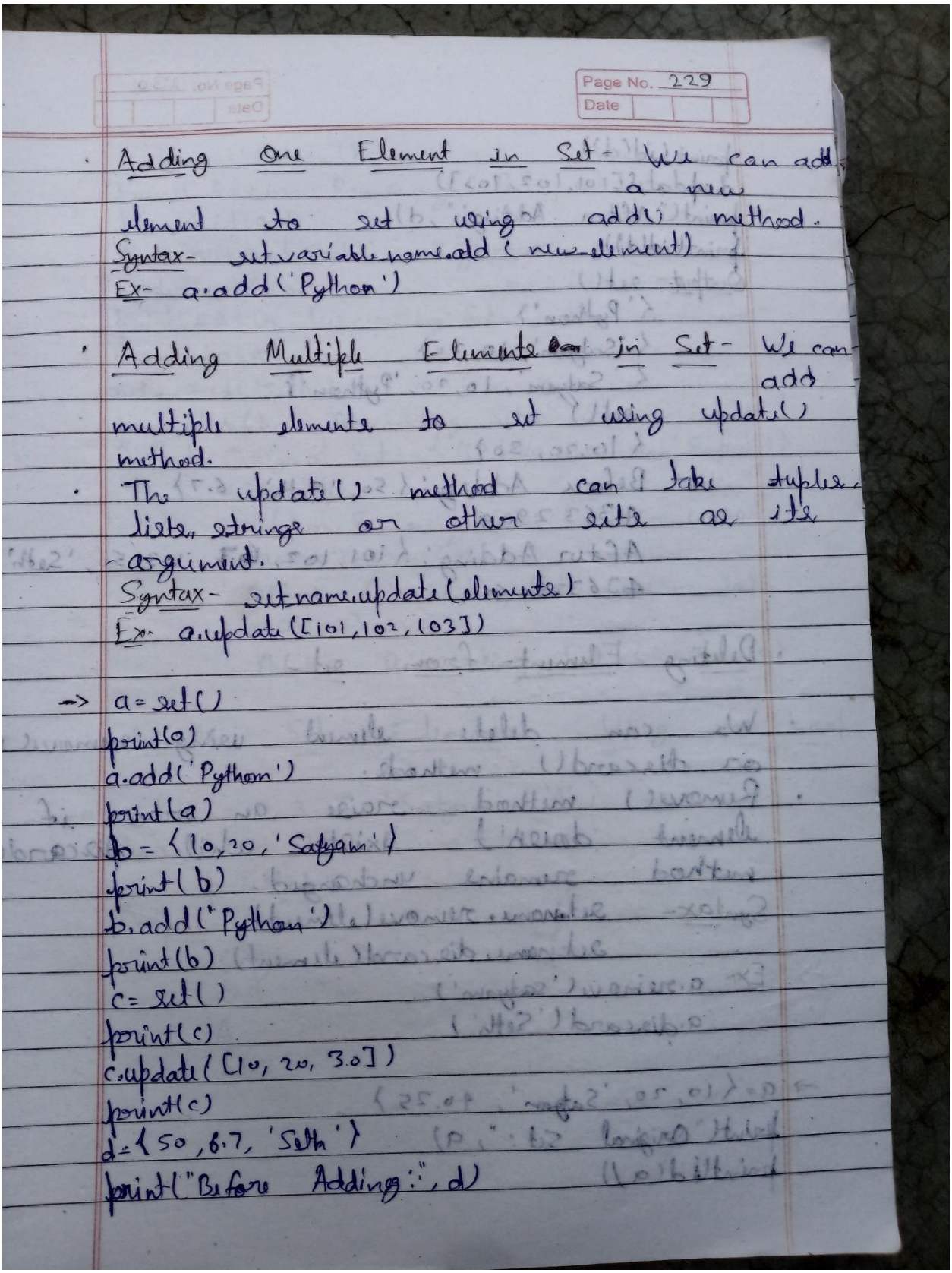
Note - a={} लिखने से empty dictionary बन जायेगा

- Accessing elements - Set are unordered so we can not access its element using index.

~~a = set()~~
~~print(type(a))~~
~~print(id(a))~~
~~b = {10, 20, 80, 10, 80, 10}~~
~~print(b)~~

→ a = set()
print(type(a))
a = {10, -30, 'Raj', 20.5, 'Satyam'}
print(a)
print(type(a))
print(id(a))
b = {10, 20, 80, 10, 80, 10}
print(b)

Output - <class 'set'>
{'Satyam', -30, 10, 20.5, 'Raj'}
<class 'set'>
4376308
{80, 10, 20}



Page No. 229
Date

- Adding One Element in Set - We can add a new element to set using add() method.
  Syntax- set_variable_name.add(new_element)
  Ex- a.add('Python')

- Adding Multiple Elements in Set - We can add multiple elements to set using update() method.
- The update() method can take tuples, lists, strings or other sets as its argument.
  Syntax- setname.update(elements)
  Ex- a.update([101,102,103])

```
a=set()
print(a)
a.add('Python')
print(a)
b={10,20,'Satyam'}
print(b)
b.add('Python')
print(b)
c=set()
print(c)
c.update([10,20,3.0])
print(c)
d={50,6.7,'Seth'}
print("Before Adding:",d)
```

Page No. 230
Date

```
print(id(d))
d.update([101,102,103])
print("After Adding:",d)
print(id(d))
```

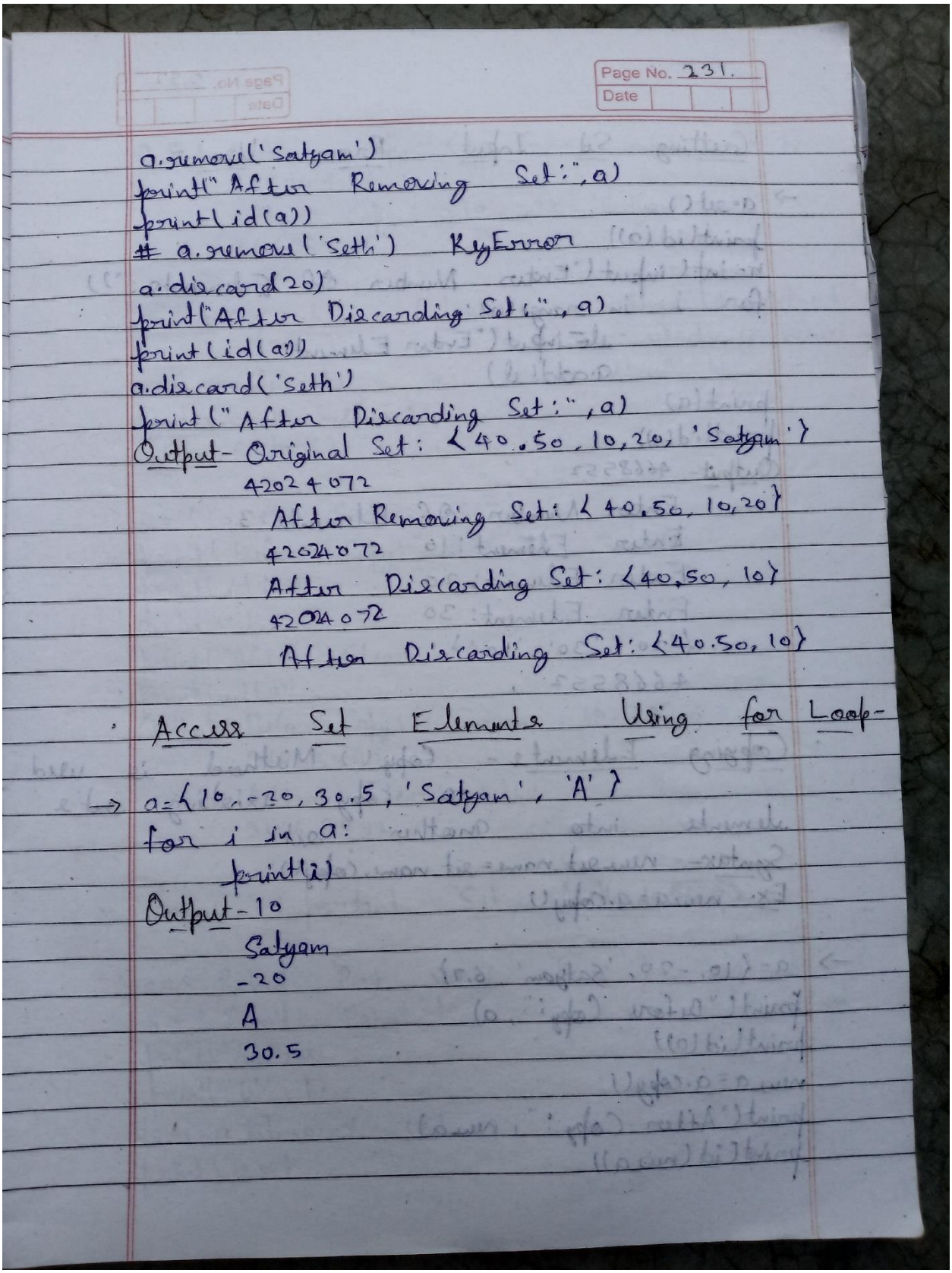
Output- set()
{'Python'}
{'Satyam', 10, 20}
{'Satyam', 10, 20, 'Python'}
set()
{10, 20, 30}
Before Adding: {50, 'Seth', 6.7}
43632904
After Adding: {101, 102, 6.7, 103, 50, 'Seth'}
43632904

- **Deleting Element from set-**
- We can delete element using remove() or discard() methods.
- Remove() method raise an error if element doesn't exists while discard() method remains unchanged.

Syntax- setname.remove(element)
setname.discard(element)

Ex- a.remove('satyam')
a.discard('Seth')

→ a={10, 20, 'Satyam', 40.25}
print("Original Set:", a)
print(id(a))



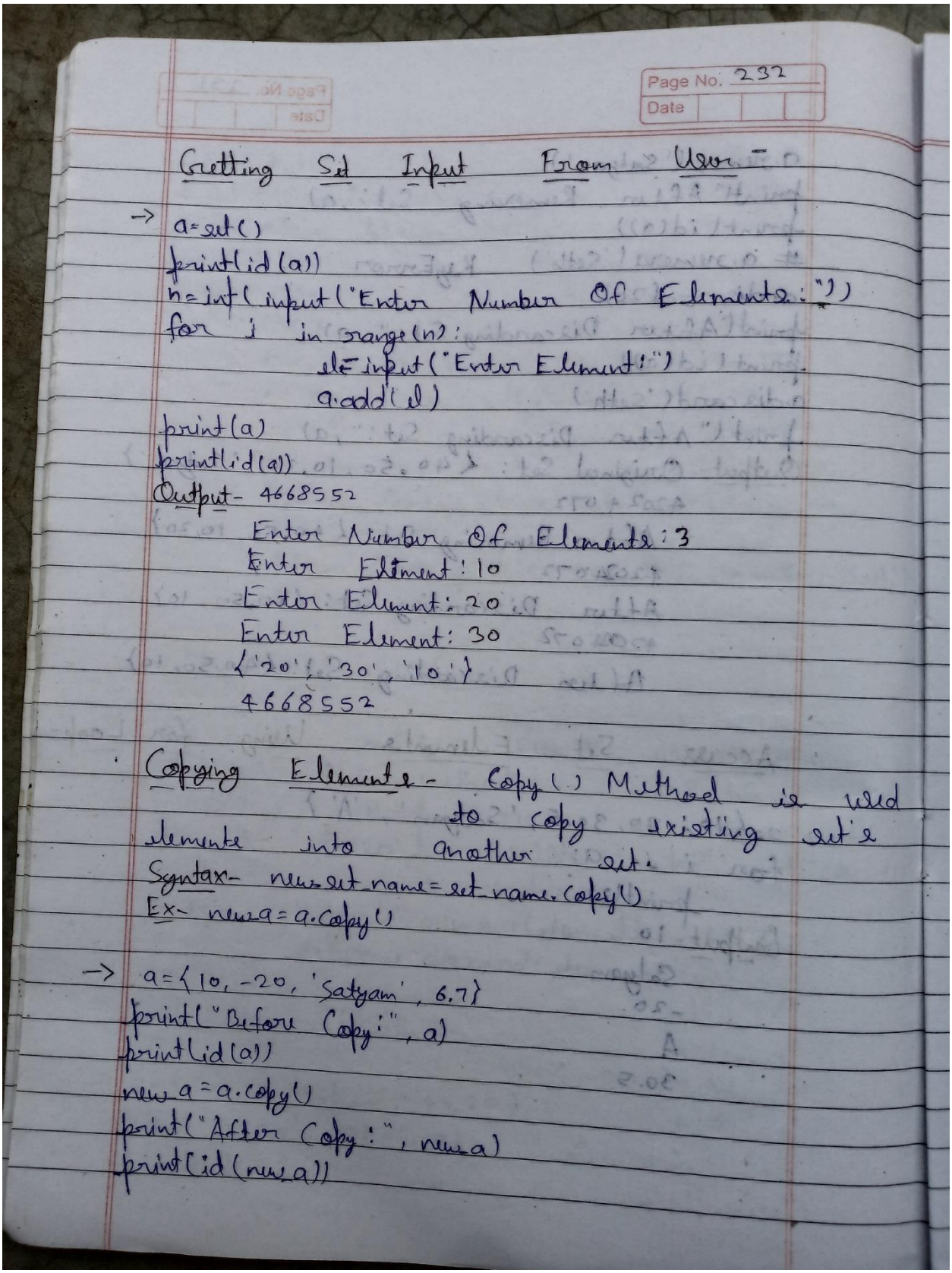
```
a.remove('Satyam')
print("After Removing Set:",a)
print(id(a))
# a.remove('Seth')  KeyError
a.discard(20)
print("After Discarding Set:",a)
print(id(a))
a.discard('Seth')
print("After Discarding Set:",a)
```

Output- Original Set: {40,50,10,20,'Satyam'}
42024072
After Removing Set: {40,50,10,20}
42024072
After Discarding Set: {40,50,10}
42024072
After Discarding Set: {40,50,10}

Access Set Elements Using for Loop-

→
```
a={10,-20,30.5,'Satyam','A'}
for i in a:
    print(i)
```

Output- 10
Satyam
-20
A
30.5



Page No. 232
Date

Getting Set Input From User -

```
a=set()
print(id(a))
n=int(input("Enter Number Of Elements:"))
for i in range(n):
    el=input("Enter Element:")
    a.add(el)
print(a)
print(id(a))
```

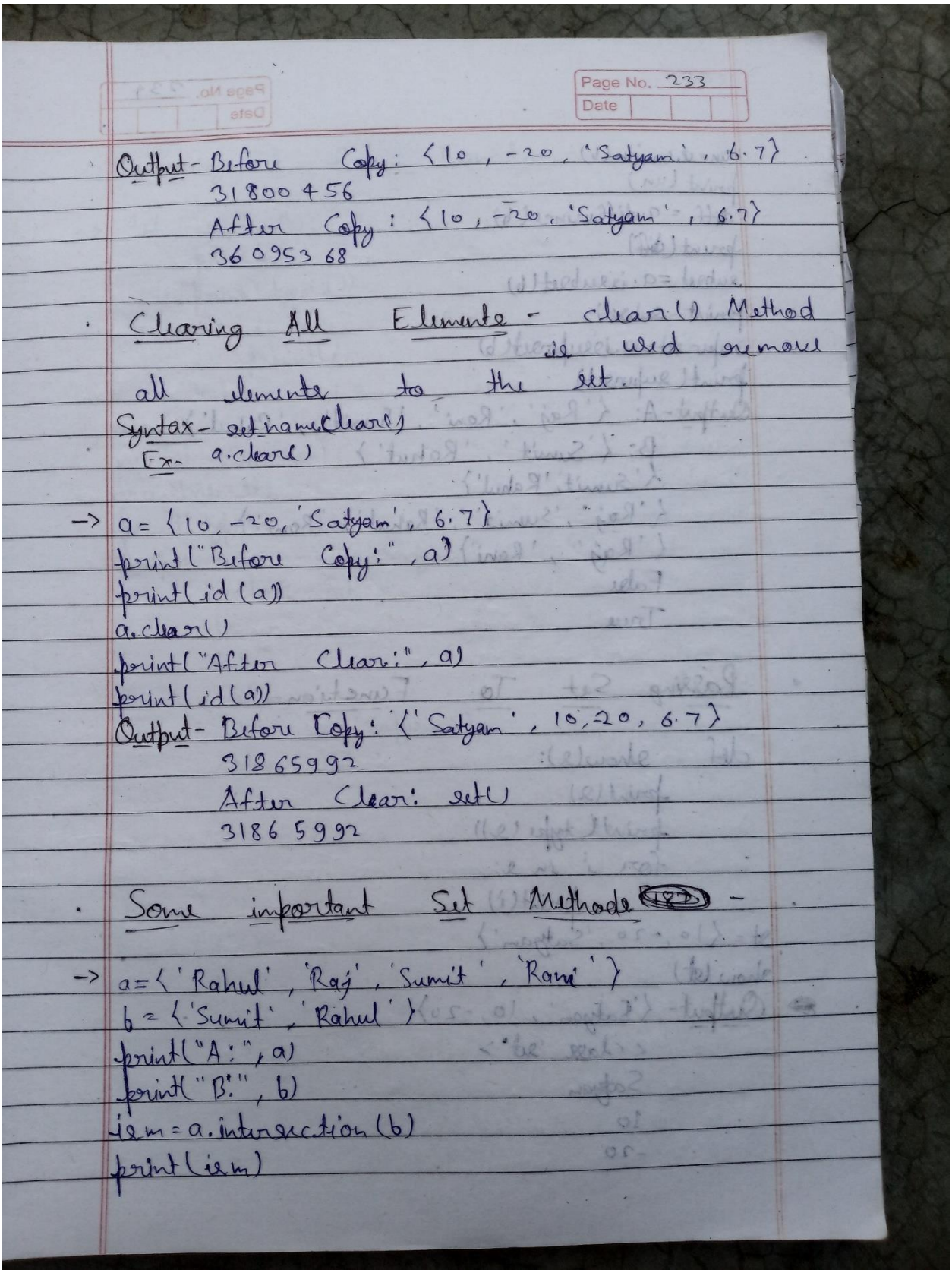
Output-
```
4668552
Enter Number Of Elements:3
Enter Element:10
Enter Element:20
Enter Element:30
{'20','30','10'}
4668552
```

- **Copying Elements** - copy() Method is used to copy existing set's elements into another set.

Syntax- new_set_name=set_name.copy()

Ex- new_a=a.copy()

```
a={10,-20,'satyam',6.7}
print("Before Copy:",a)
print(id(a))
new_a=a.copy()
print("After Copy:",new_a)
print(id(new_a))
```



Page No. 233
Date

Output- Before Copy: {10, -20, 'Satyam', 6.7}
31800456
After Copy: {10, -20, 'Satyam', 6.7}
36095368

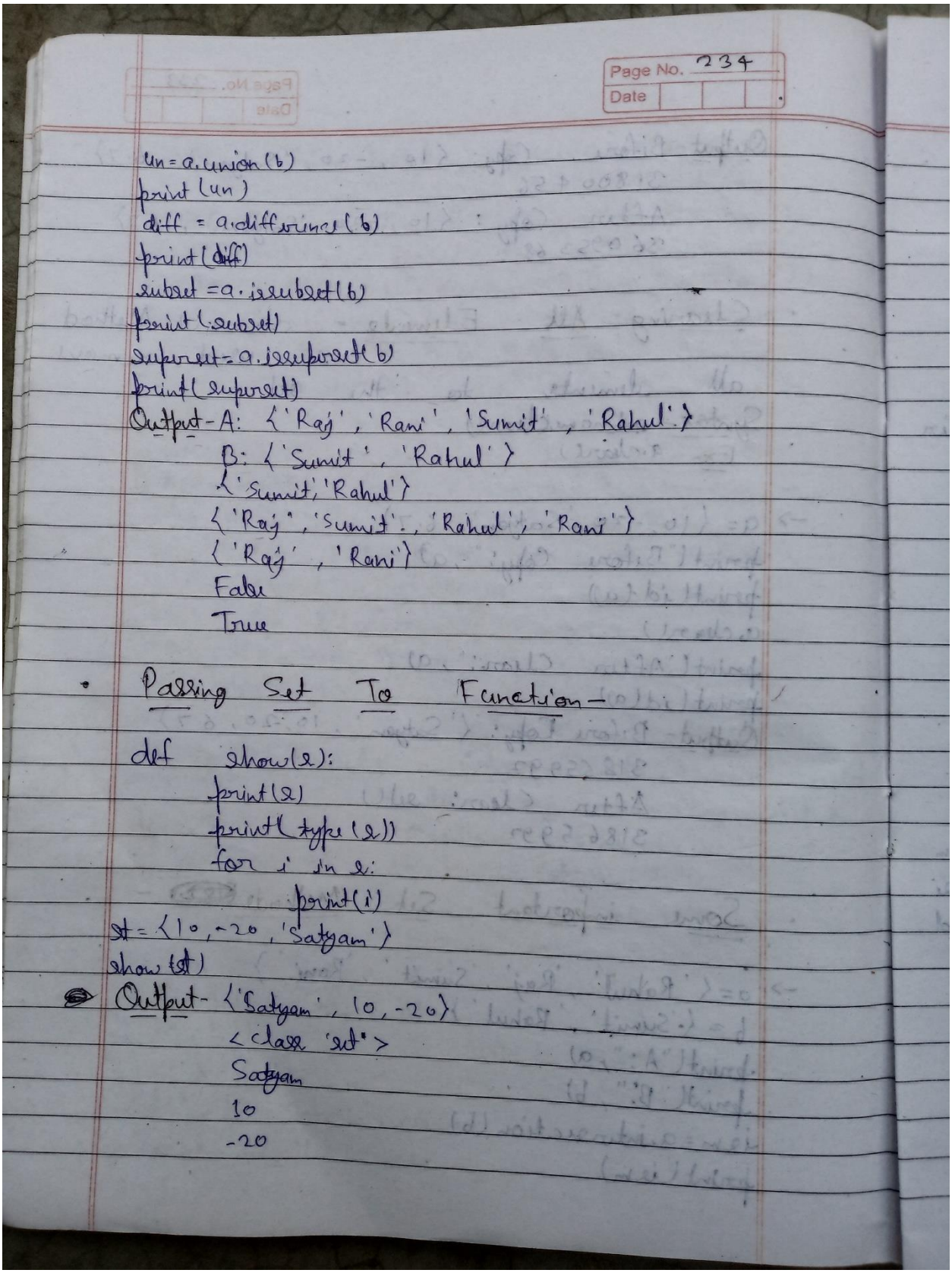
- Clearing All Elements - clear() Method is used remove all elements to the set.

Syntax- set_name.clear()
Ex- a.clear()

-> a = {10, -20, 'Satyam', 6.7}
print("Before Copy:", a)
print(id(a))
a.clear()
print("After Clear:", a)
print(id(a))
Output- Before Copy: {'Satyam', 10, 20, 6.7}
31865992
After Clear: set()
31865992

- Some important Set Methods -

-> a = {'Rahul', 'Raj', 'Sumit', 'Rani'}
b = {'Sumit', 'Rahul'}
print("A:", a)
print("B:", b)
isn = a.intersection(b)
print(isn)



Page No. 234
Date

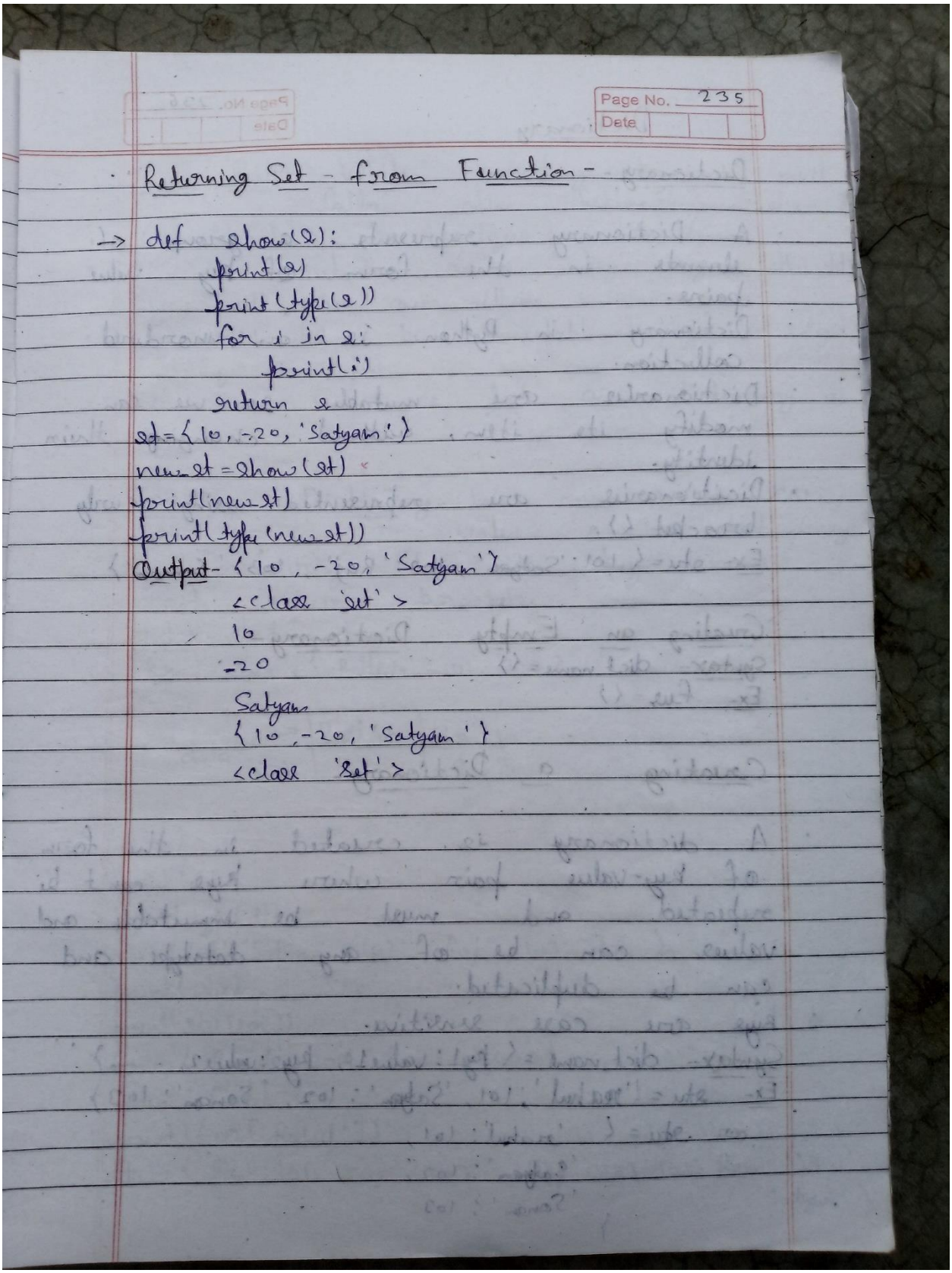
```
un = a.union(b)
print(un)
diff = a.difference(b)
print(diff)
subset = a.issubset(b)
print(subset)
superset = a.issuperset(b)
print(superset)
```

Output- A: {'Raj', 'Rani', 'Sumit', 'Rahul'}
B: {'Sumit', 'Rahul'}
{'Sumit', 'Rahul'}
{'Raj', 'Sumit', 'Rahul', 'Rani'}
{'Raj', 'Rani'}
False
True

- **Passing Set To Function-**

```
def show(s):
    print(s)
    print(type(s))
    for i in s:
        print(i)
st = {10, -20, 'Satyam'}
show(st)
```

Output- {'Satyam', 10, -20}
<class 'set'>
Satyam
10
-20

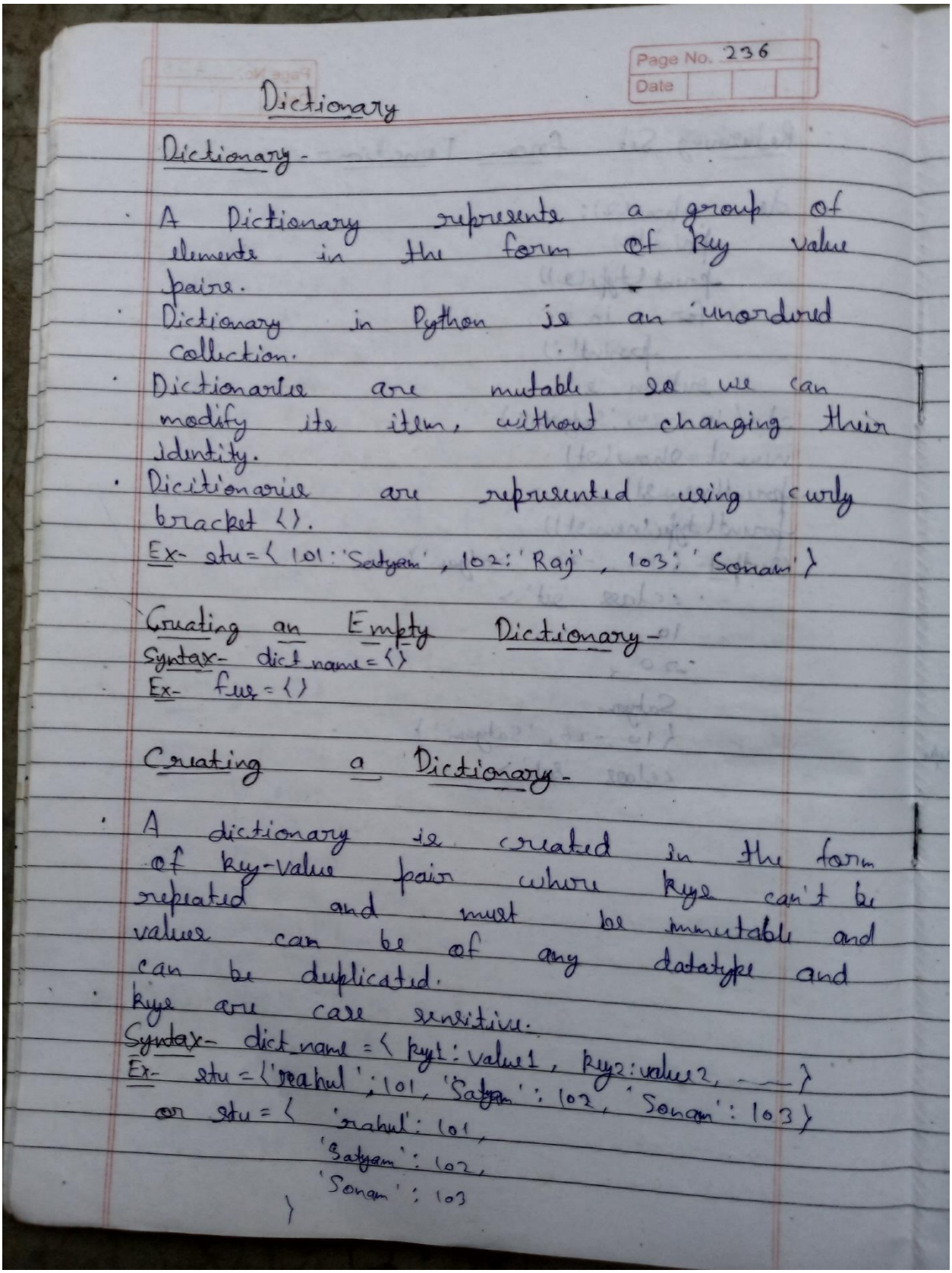


Page No. 235
Date

Returning Set - from Function -

```
def show(a):
    print(a)
    print(type(a))
    for i in a:
        print(i)
    return a
st={10,-20,'Satyam'}
new_st=show(st)
print(new_st)
print(type(new_st))
```

Output -
```
{10, -20, 'Satyam'}
<class 'set'>
10
-20
Satyam
{10, -20, 'Satyam'}
<class 'set'>
```



Page No. 236
Date

Dictionary

Dictionary -

- A Dictionary represents a group of elements in the form of key value pairs.
- Dictionary in Python is an unordered collection.
- Dictionaries are mutable so we can modify its item, without changing their identity.
- Dicitionaries are represented using curly bracket {}.

Ex- stu = {101: 'Satyam', 102: 'Raj', 103: 'Sonam'}

Creating an Empty Dictionary -

Syntax- dict_name = {}

Ex- fun = {}

Creating a Dictionary -

- A dictionary is created in the form of key-value pair where keys can't be repeated and must be immutable and values can be of any datatype and can be duplicated.
- Keys are case sensitive.

Syntax- dict_name = { key1: value1, key2: value2, ... }

Ex- stu = {'rahul': 101, 'Satyam': 102, 'Sonam': 103}

or stu = {
'rahul': 101,
'Satyam': 102,
'Sonam': 103
}

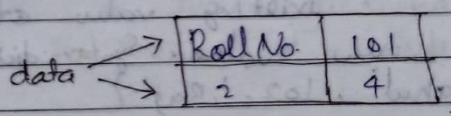
Page No. 237
Date

- Key Rules - While writting key we must follow the following rules -
- Key should be unique.
- If we mention some key again, the old key will be overwritten.
- Key should be immutable type i.e. integer, string or tuple.
- We can not use list or dictionary as key.

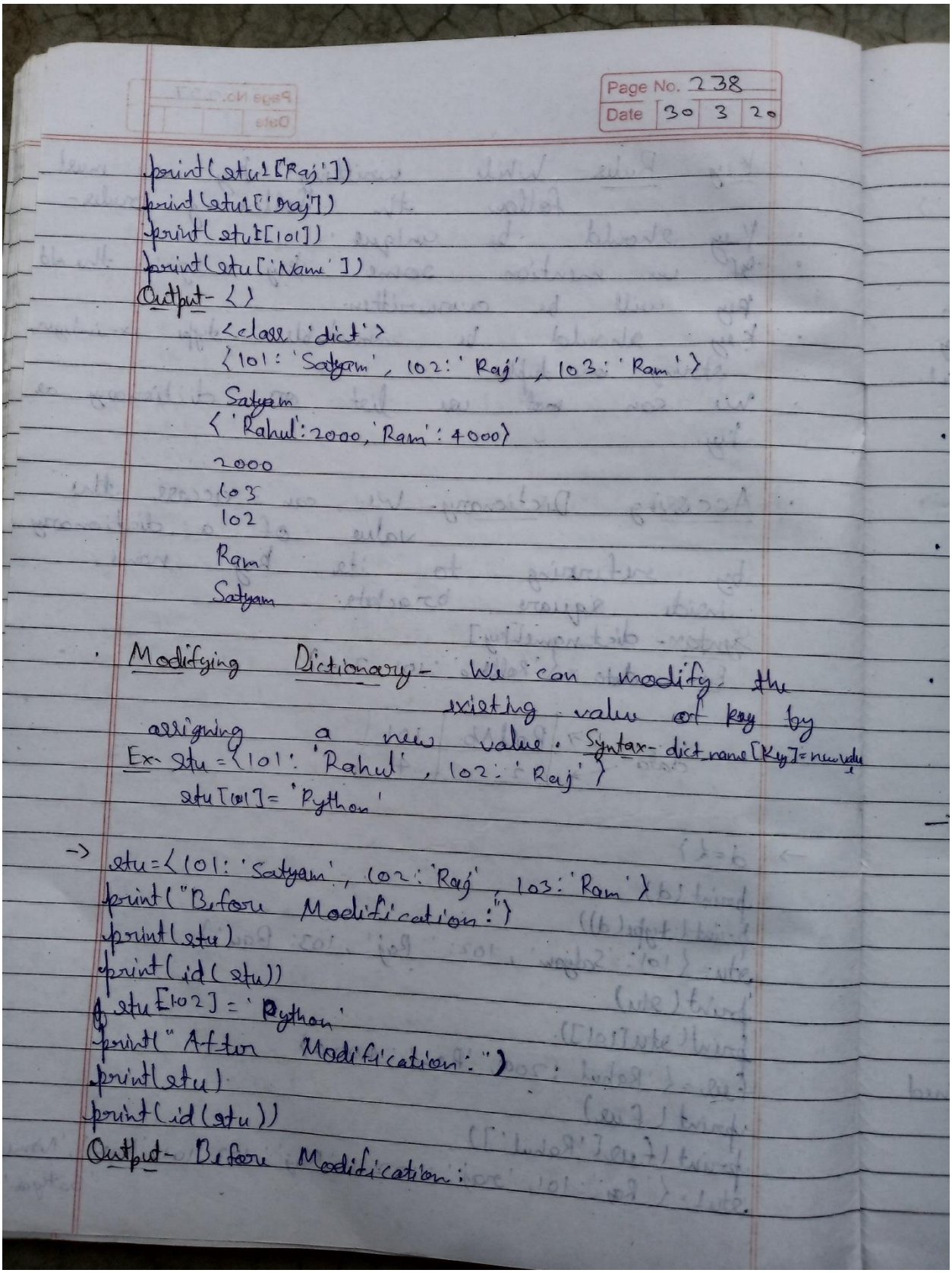
- Accessing Dictionary - We can access the value of a dictionary by referring to its key name, inside square brackets.

Syntax - dict_name[key]

Ex. data = {'RollNo' : 101, 2:4}



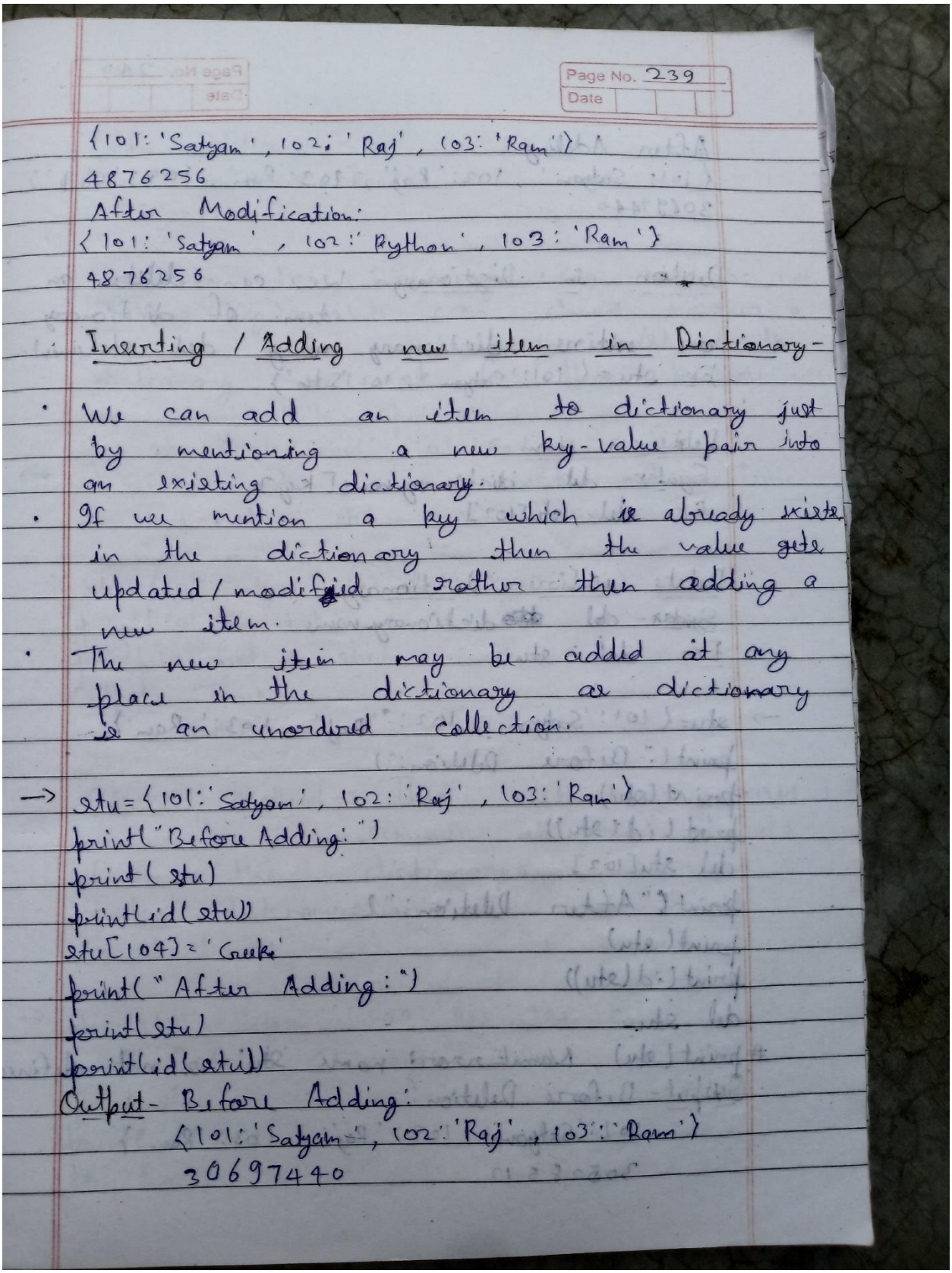
→
```
d={}
print(d)
print(type(d))
stu={101:'Satyam', 102:'Raj', 103:'Ram'}
print(stu)
print(stu[101])
fee={'Rahul':2000, 'Ram':4000}
print(fee)
print(fee['Rahul'])
stu1={'Raj':101, 'raj':102, 'Raj':103, 101:'Ram', 'Name':'Satyam'}
```



print(stu1['Raj'])
print(stu1['Raj'])
print(stu1[101])
print(stu['Name'])
Output- {}
<class 'dict'>
{101: 'Satyam', 102: 'Raj', 103: 'Ram'}
Satyam
{'Rahul': 2000, 'Ram': 4000}
2000
103
102
Ram
Satyam

- Modifying Dictionary- We can modify the existing value of key by assigning a new value. Syntax- dict_name[Key]=new_value
Ex- stu={101: 'Rahul', 102: 'Raj'}
stu[101]='Python'

-> stu={101: 'Satyam', 102: 'Raj', 103: 'Ram'}
print("Before Modification:")
print(stu)
print(id(stu))
stu[102]='Python'
print("After Modification:")
print(stu)
print(id(stu))
Output- Before Modification:



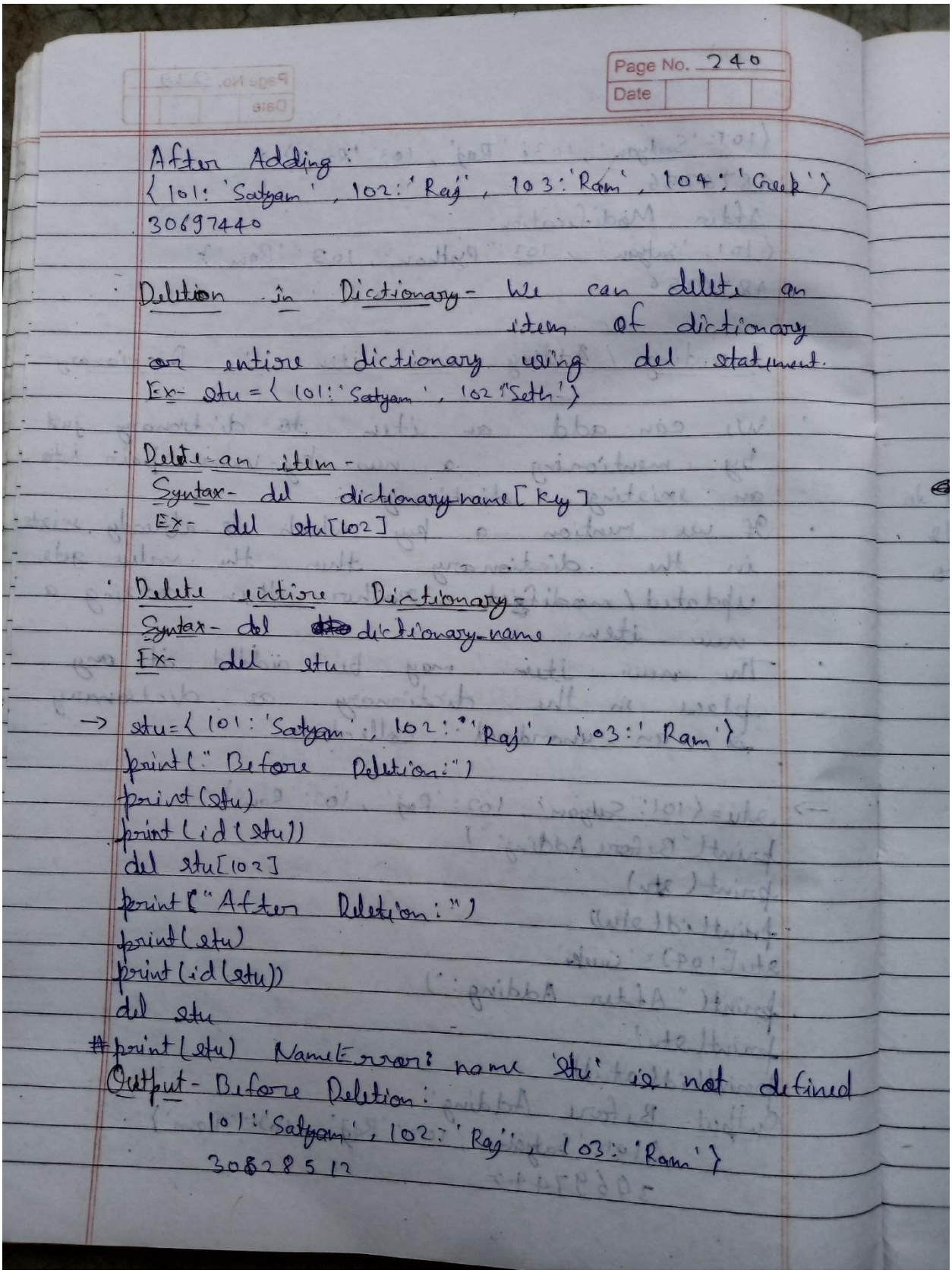
{101: 'Satyam', 102: 'Raj', 103: 'Ram'}
4876256
After Modification:
{101: 'Satyam', 102: 'Python', 103: 'Ram'}
4876256

Inserting / Adding new item in Dictionary-

- We can add an item to dictionary just by mentioning a new key-value pair into an existing dictionary.
- If we mention a key which is already exists in the dictionary then the value gets updated / modified rather then adding a new item.
- The new item may be added at any place in the dictionary as dictionary is an unordered collection.

```
stu={101:'Satyam', 102:'Raj', 103:'Ram'}
print("Before Adding:")
print(stu)
print(id(stu))
stu[104]='Geeky'
print("After Adding:")
print(stu)
print(id(stu))
```

Output- Before Adding:
{101: 'Satyam', 102: 'Raj', 103: 'Ram'}
30697440



Page No. 240
Date

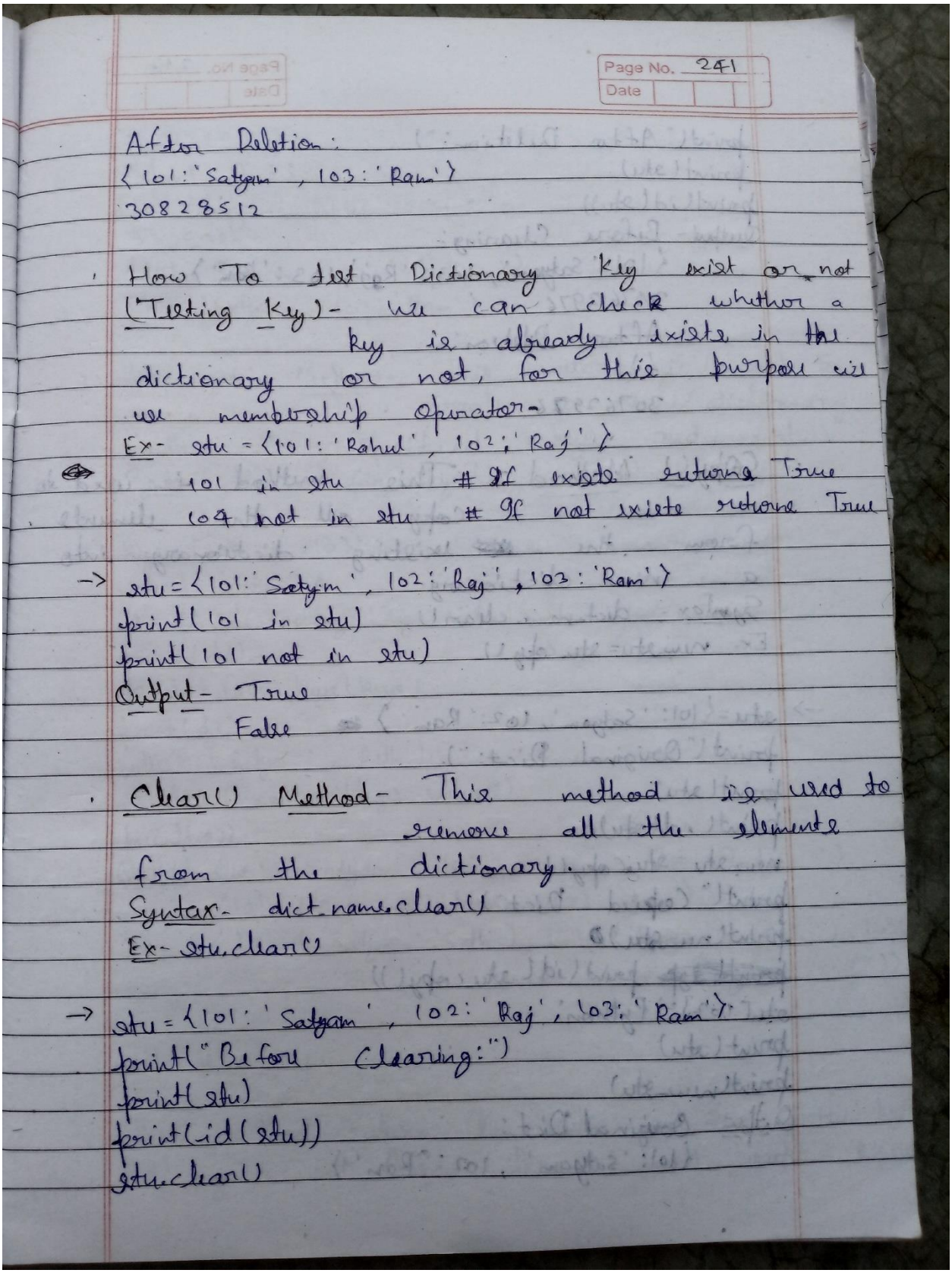
After Adding:
{101: 'Satyam', 102: 'Raj', 103: 'Ram', 104: 'Geek'}
30697440

Deletion in Dictionary - We can delete an item of dictionary or entire dictionary using del statement.
Ex- stu = {101: 'Satyam', 102 'Seth'}

Delete an item -
Syntax- del dictionary-name[Key]
Ex- del stu[102]

Delete entire Dictionary -
Syntax- del dictionary-name
Ex- del stu

→ stu={101: 'Satyam', 102: 'Raj', 103: 'Ram'}
print(" Before Deletion:")
print(stu)
print(id(stu))
del stu[102]
print("After Deletion:")
print(stu)
print(id(stu))
del stu
#print(stu) NameError: name 'stu' is not defined
Output - Before Deletion:
{101: 'Satyam', 102: 'Raj', 103: 'Ram'}
30828512



Page No. 241
Date

After Deletion:
{101: 'Satyam', 103: 'Ram'}
30828512

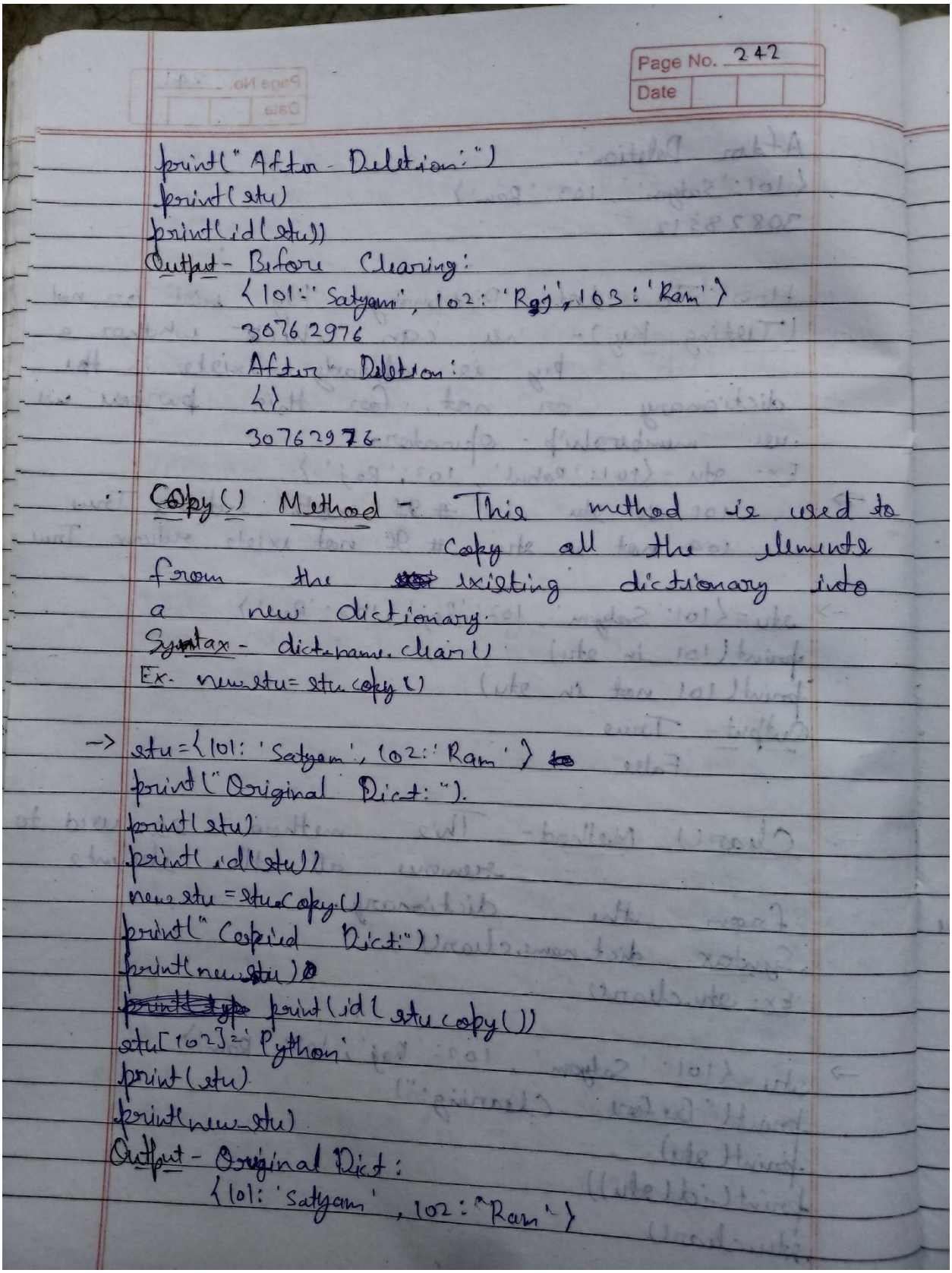
- How To Test Dictionary key exist or not (Testing Key)- We can check whether a key is already existe in the dictionary or not, for this purpose we use membership operator-

Ex- stu = {101: 'Rahul', 102: 'Raj'}
101 in stu # If exists returns True
104 not in stu # If not exists returns True

-> stu = {101: 'Satyam', 102: 'Raj', 103: 'Ram'}
print(101 in stu)
print(101 not in stu)
Output- True
False

- clear() Method- This method is used to remove all the elements from the dictionary.
Syntax- dict_name.clear()
Ex- stu.clear()

-> stu = {101: 'Satyam', 102: 'Raj', 103: 'Ram'}
print("Before Clearing:")
print(stu)
print(id(stu))
stu.clear()

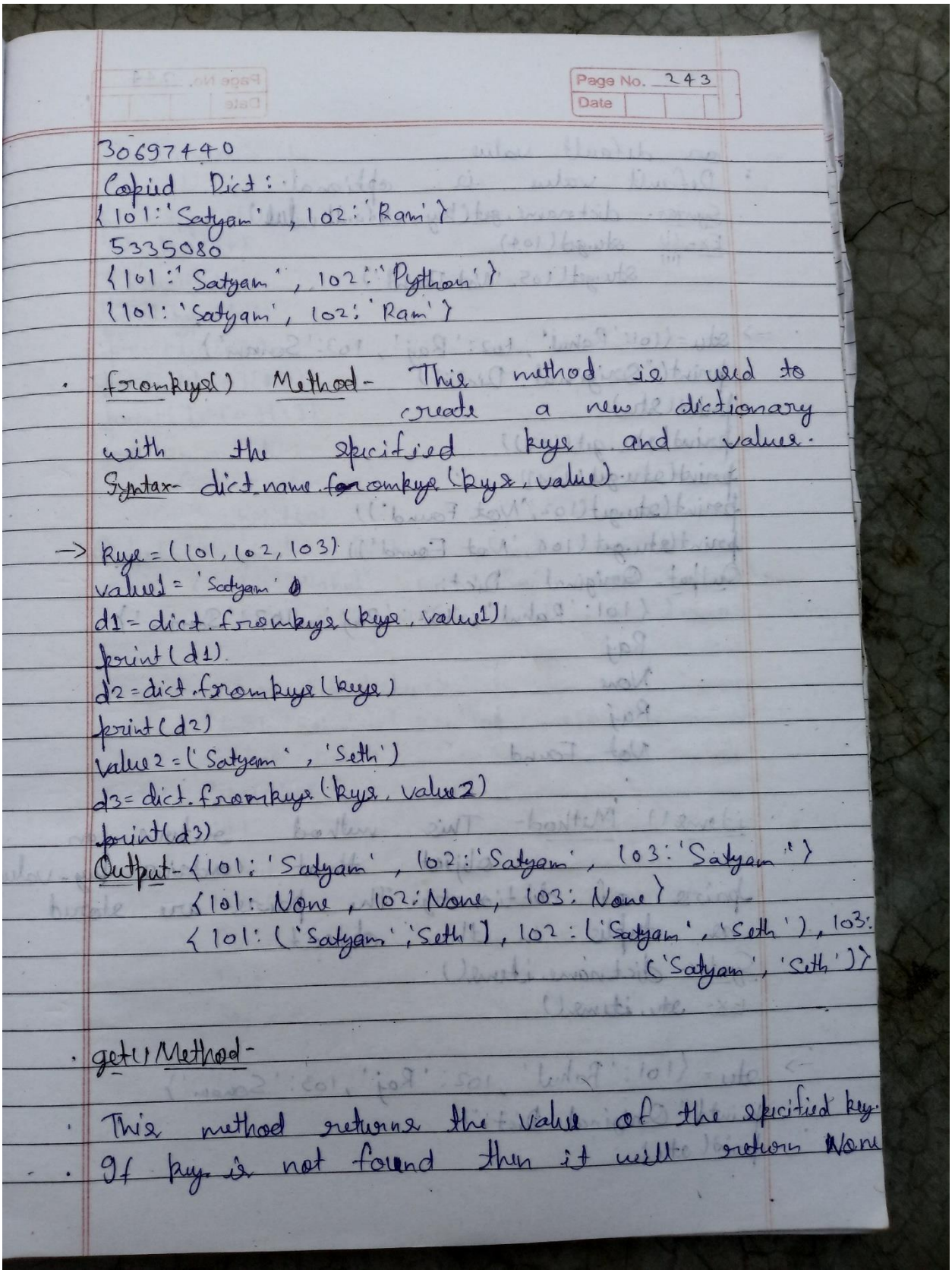


Page No. 242
Date

print("After Deletion:")
print(stu)
print(id(stu))
Output - Before Clearing:
{101: 'Satyam', 102: 'Raj', 103: 'Ram'}
30762976
After Deletion:
{}
30762976

Copy() Method - This method is used to copy all the elements from the existing dictionary into a new dictionary.
Syntax - dict_name.clear()
Ex. new_stu = stu.copy()

-> stu = {101: 'Satyam', 102: 'Ram'}
print("Original Dict:")
print(stu)
print(id(stu))
new_stu = stu.copy()
print("Copied Dict:")
print(new_stu)
print(id(stu.copy()))
stu[102] = 'Python'
print(stu)
print(new_stu)
Output - Original Dict:
{101: 'Satyam', 102: 'Ram'}



30697440
Copied Dict:
{101: 'Satyam', 102: 'Ram'}
5335080
{101: 'Satyam', 102: 'Python'}
{101: 'Satyam', 102: 'Ram'}

- fromkys() Method - This method is used to create a new dictionary with the specified keys and values.

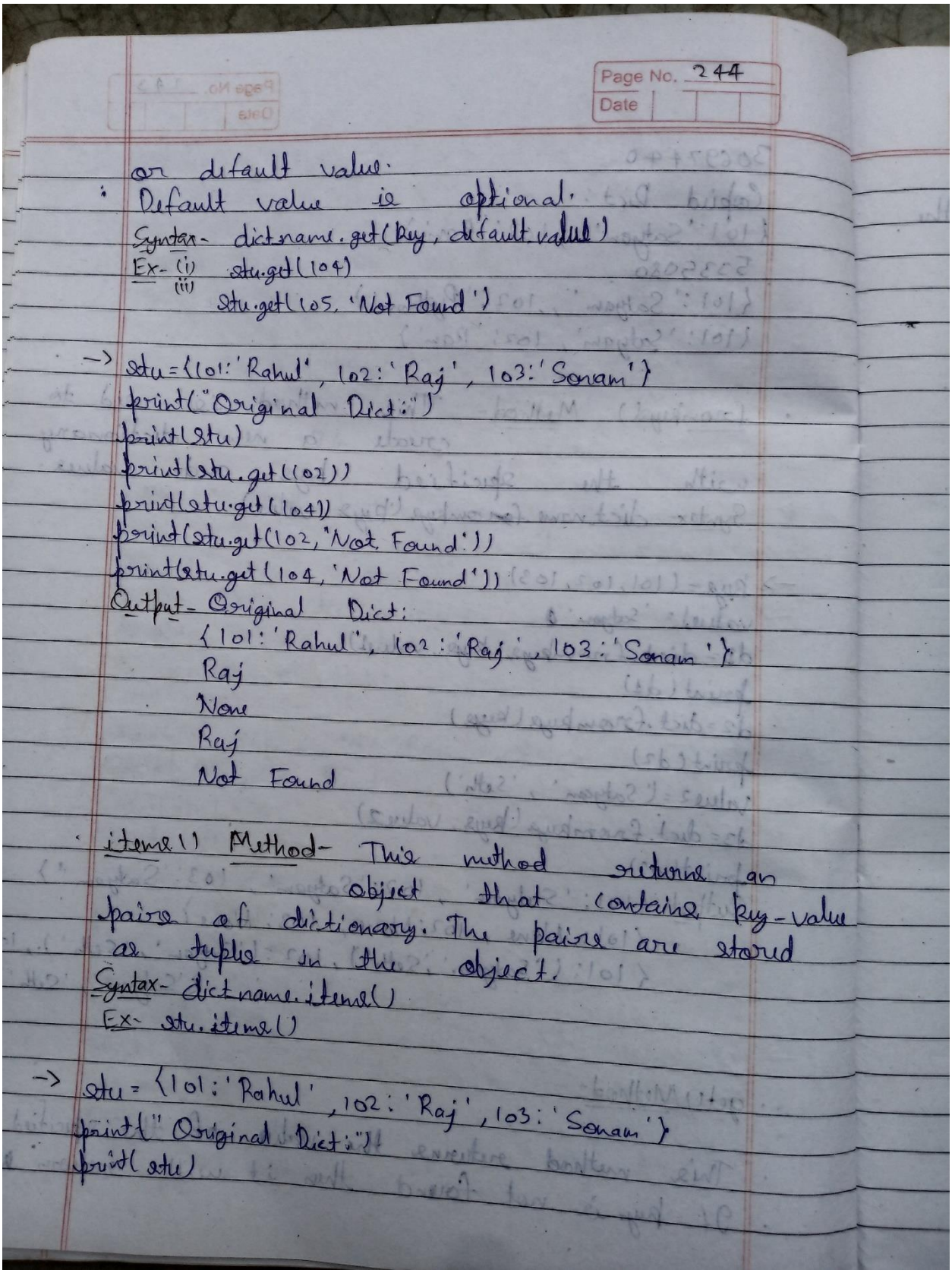
Syntax - dict_name.fromkys(keys, value)

```
-> Keys = (101, 102, 103)
value1 = 'Satyam'
d1 = dict.fromkys(keys, value1)
print(d1)
d2 = dict.fromkys(keys)
print(d2)
value2 = ('Satyam', 'Seth')
d3 = dict.fromkys(keys, value2)
print(d3)
```

Output - {101: 'Satyam', 102: 'Satyam', 103: 'Satyam'}
{101: None, 102: None, 103: None}
{101: ('Satyam', 'Seth'), 102: ('Satyam', 'Seth'), 103: ('Satyam', 'Seth')}

- get() Method -

- This method returns the value of the specified key.
- If key is not found then it will return None



or default value.

: Default value is optional.

Syntax- dictname.get(key, default value)

Ex- (i) stu.get(104)

(ii) stu.get(105, 'Not Found')

-> stu={101:'Rahul', 102:'Raj', 103:'Sonam'}
print("Original Dict:")
print(stu)
print(stu.get(102))
print(stu.get(104))
print(stu.get(102,"Not Found"))
print(stu.get(104,'Not Found'))

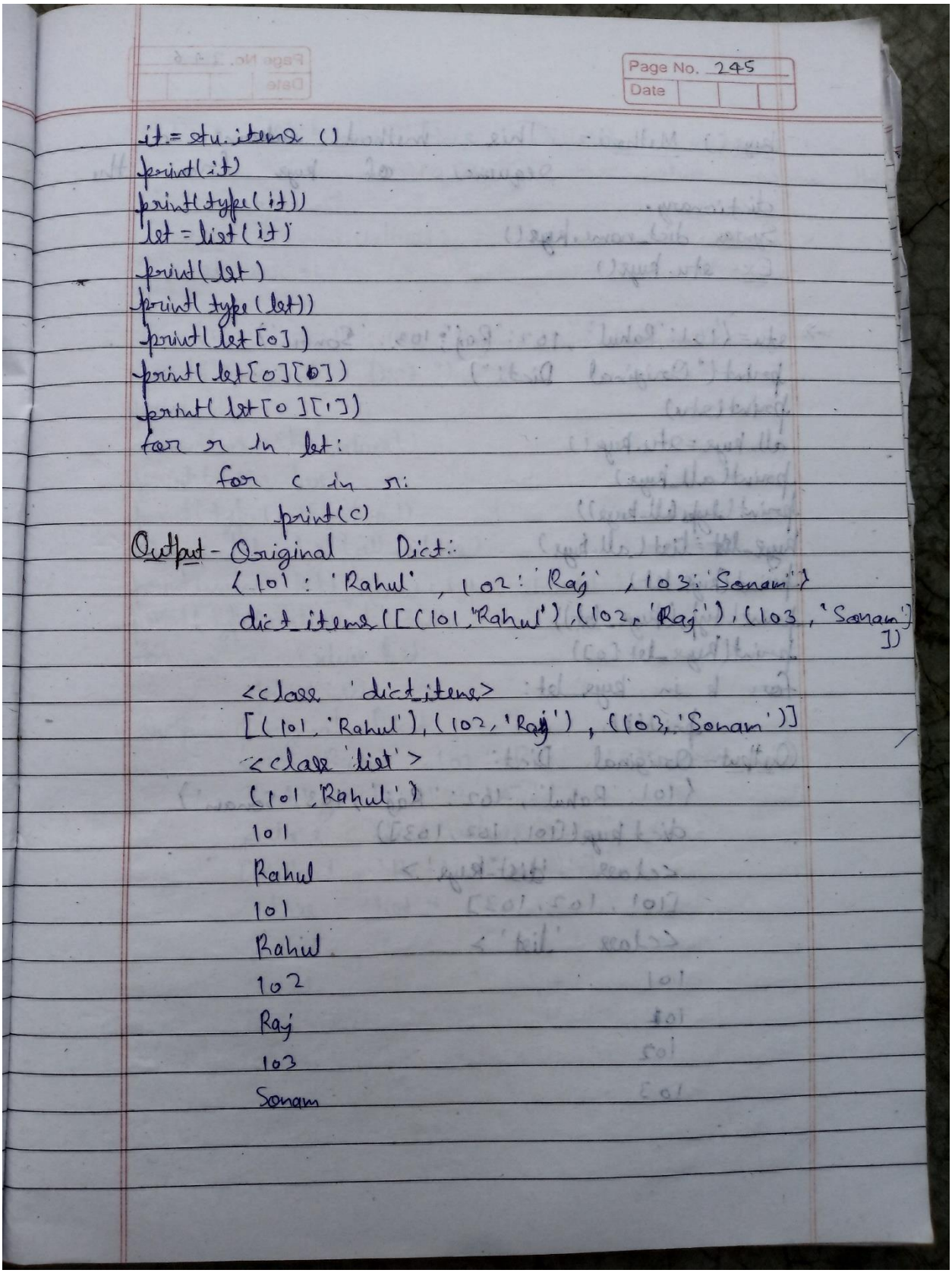
Output- Original Dict:
{101:'Rahul', 102:'Raj', 103:'Sonam'}
Raj
None
Raj
Not Found

items() Method- This method returns an object that contains key-value pairs of dictionary. The pairs are stored as tuples in the object.

Syntax- dictname.items()

Ex- stu.items()

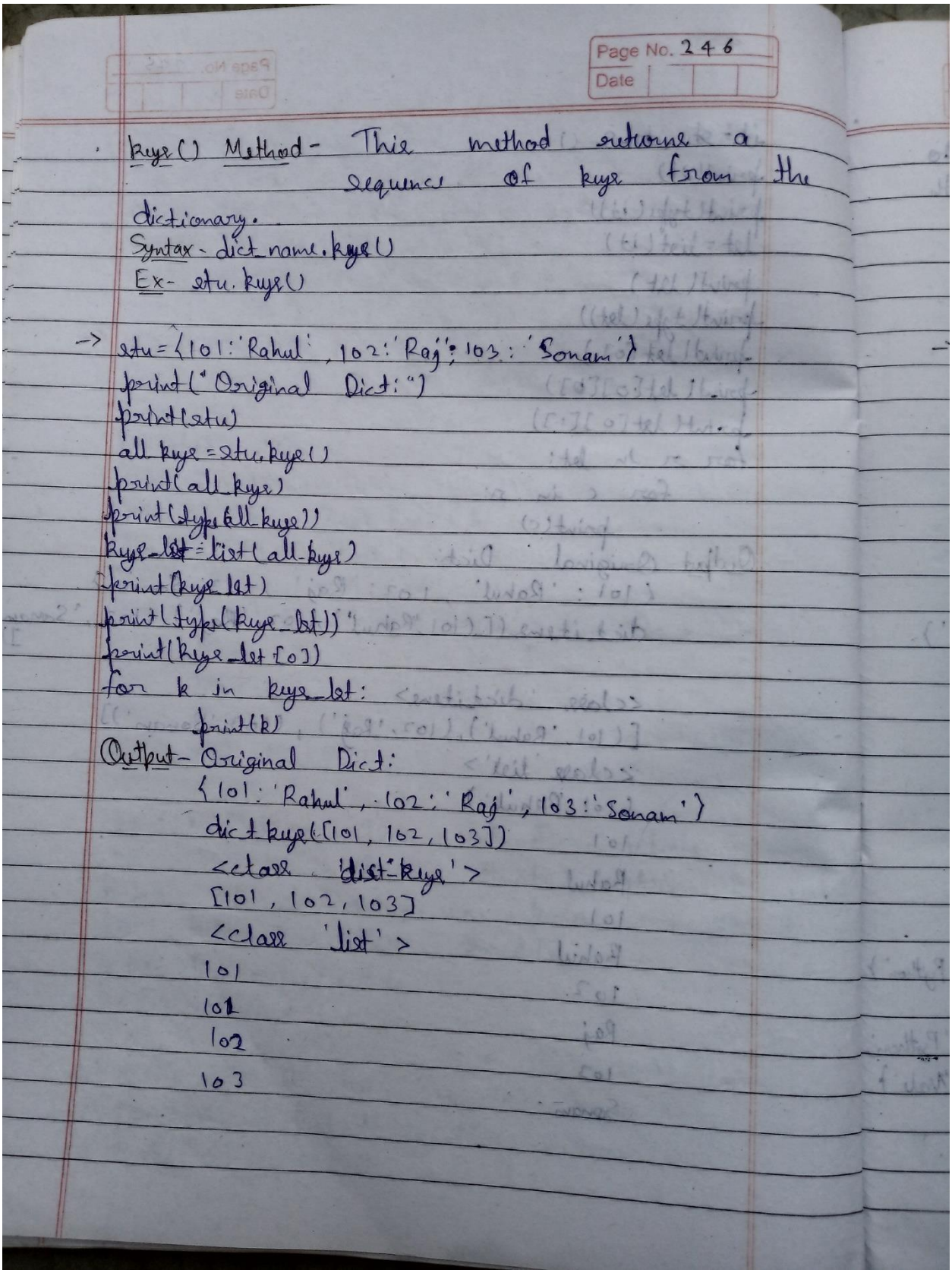
-> stu={101:'Rahul', 102:'Raj', 103:'Sonam'}
print("Original Dict:")
print(stu)



```
it = stu.items()
print(it)
print(type(it))
lst = list(it)
print(lst)
print(type(lst))
print(lst[0])
print(lst[0][0])
print(lst[0][1])
for r in lst:
    for c in r:
        print(c)
```

Output - Original Dict:
{101: 'Rahul', 102: 'Raj', 103: 'Sonam'}
dict_items([(101, 'Rahul'), (102, 'Raj'), (103, 'Sonam')])

<class 'dict_items'>
[(101, 'Rahul'), (102, 'Raj'), (103, 'Sonam')]
<class 'list'>
(101, 'Rahul')
101
Rahul
101
Rahul
102
Raj
103
Sonam



Page No. 246
Date

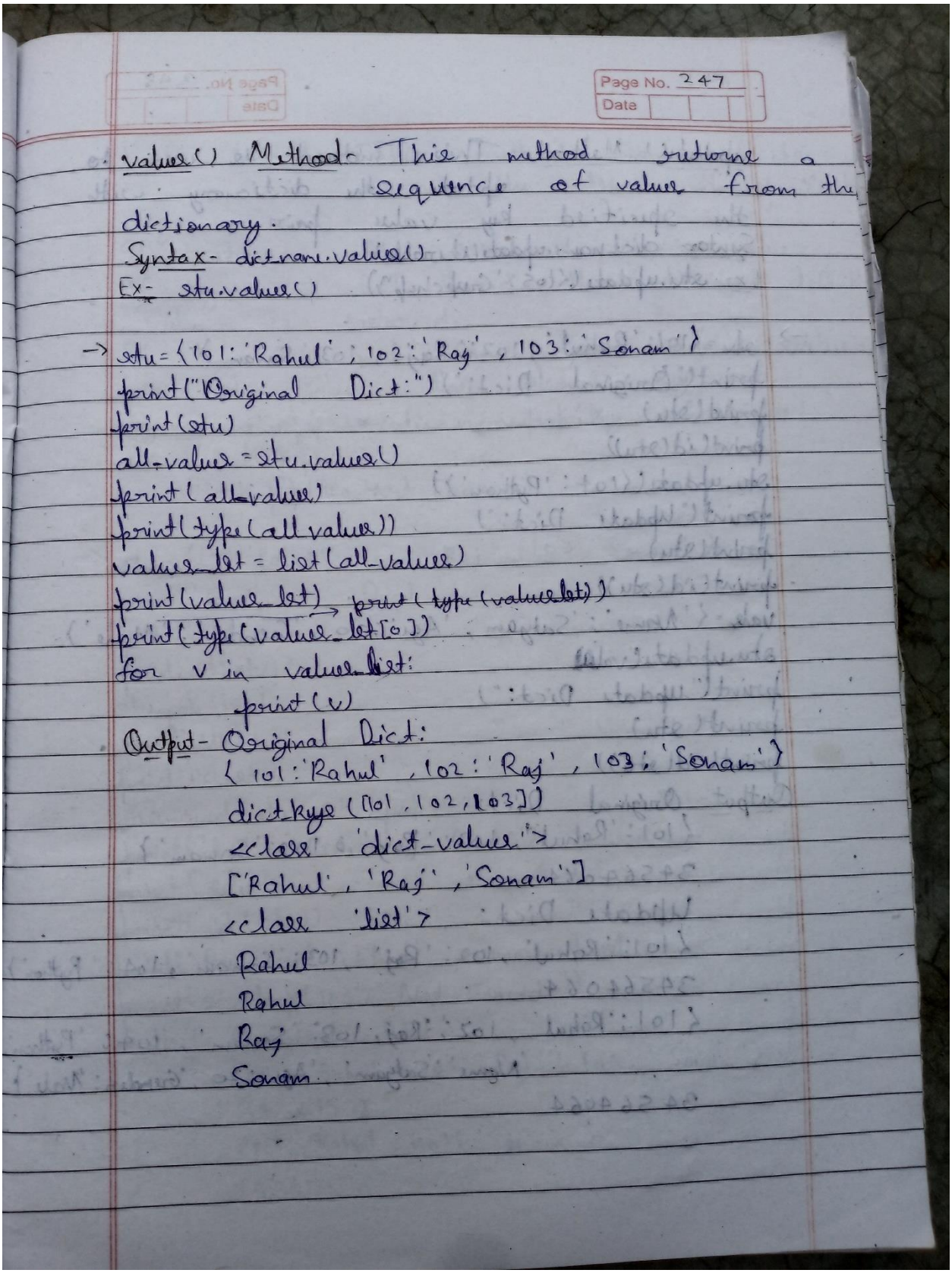
- keys() Method - This method returns a sequence of keys from the dictionary.

Syntax - dict_name.keys()

Ex - stu.keys()

```
-> stu={101:'Rahul',102:'Raj',103:'Sonam'}
print("Original Dict:")
print(stu)
all_keys=stu.keys()
print(all_keys)
print(type(all_keys))
keys_lst=list(all_keys)
print(keys_lst)
print(type(keys_lst))
print(keys_lst[0])
for k in keys_lst:
    print(k)
```

Output -
```
Original Dict:
{101:'Rahul',102:'Raj',103:'Sonam'}
dict_keys([101,102,103])
<class 'dict_keys'>
[101,102,103]
<class 'list'>
101
101
102
103
```



Page No. 247
Date

values() Method- This method returns a sequence of values from the dictionary.

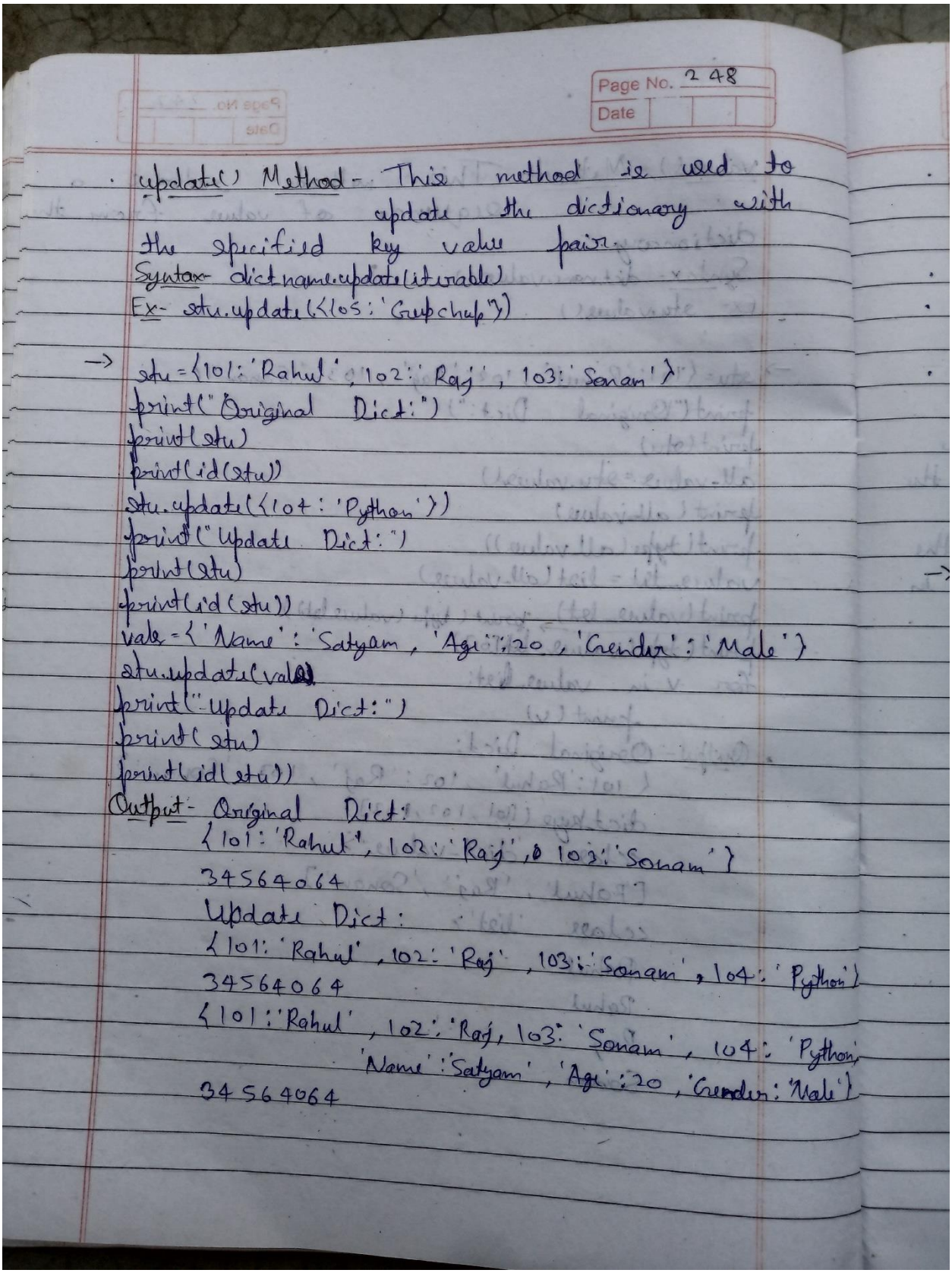
Syntax- dictname.values()

Ex- stu.values()

```
stu={101:'Rahul',102:'Raj',103:'Sonam'}
print("Original Dict:")
print(stu)
all_values=stu.values()
print(all_values)
print(type(all_values))
values_lst=list(all_values)
print(values_lst)
print(type(values_lst[0]))
for v in values_list:
    print(v)
```

Output- Original Dict:

```
{101:'Rahul',102:'Raj',103:'Sonam'}
dict_type([101,102,103])
<class 'dict_values'>
['Rahul','Raj','Sonam']
<class 'list'>
Rahul
Rahul
Raj
Sonam
```



Page No. 248
Date

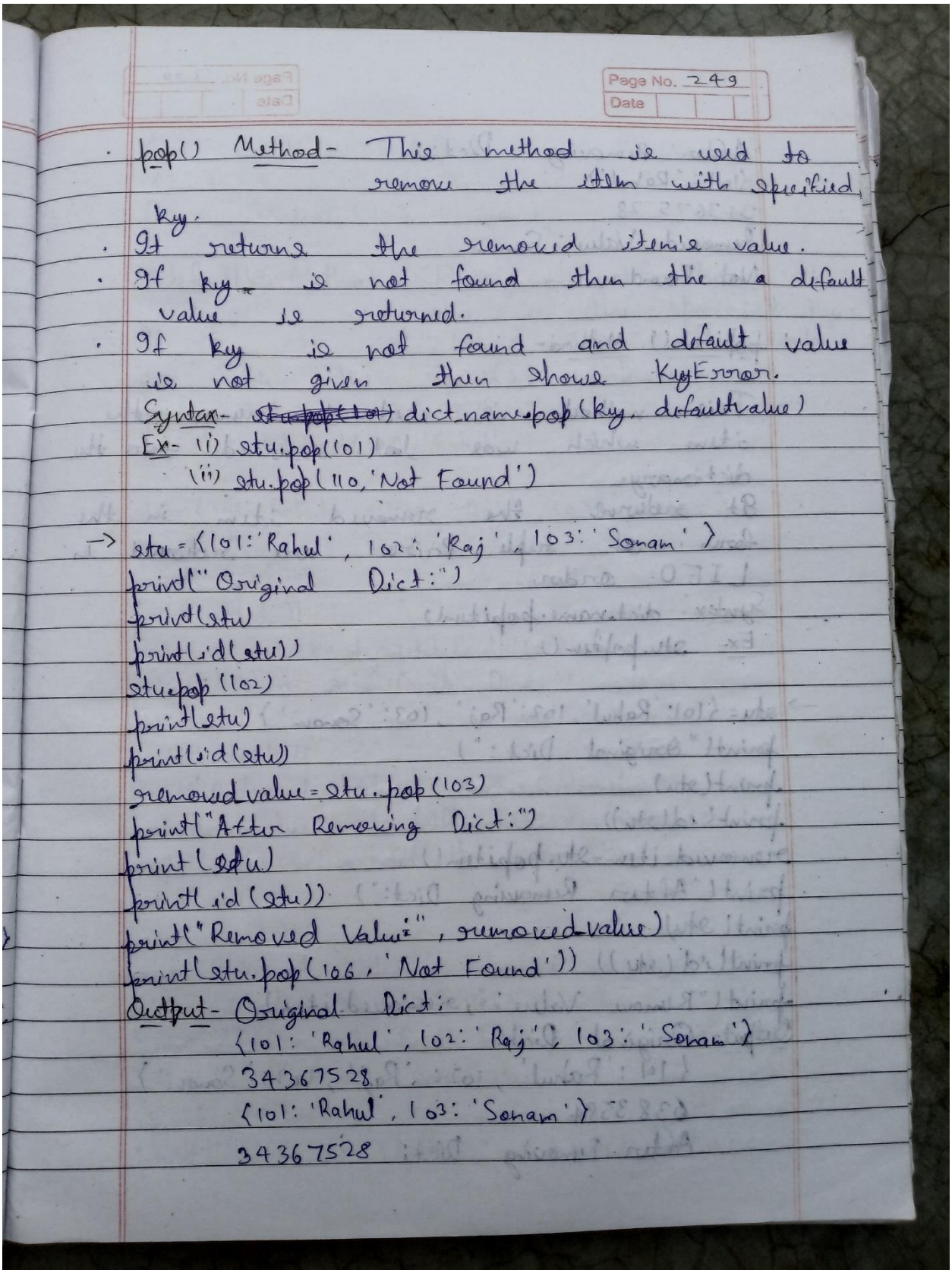
- update() Method - This method is used to update the dictionary with the specified key value pair.

Syntax- dictname.update(iterable)

Ex- stu.update({105:'Gupchup'})

→
```
stu={101:'Rahul',102:'Raj',103:'Sonam'}
print("Original Dict:")
print(stu)
print(id(stu))
stu.update({104:'Python'})
print("Update Dict:")
print(stu)
print(id(stu))
val={'Name':'Satyam','Age':20,'Gender':'Male'}
stu.update(val)
print("Update Dict:")
print(stu)
print(id(stu))
```

Output- Original Dict:
{101:'Rahul',102:'Raj',103:'Sonam'}
34564064
Update Dict:
{101:'Rahul',102:'Raj',103:'Sonam',104:'Python'}
34564064
{101:'Rahul',102:'Raj',103:'Sonam',104:'Python',
'Name':'Satyam','Age':20,'Gender':'Male'}
34564064



Page No. 249
Date

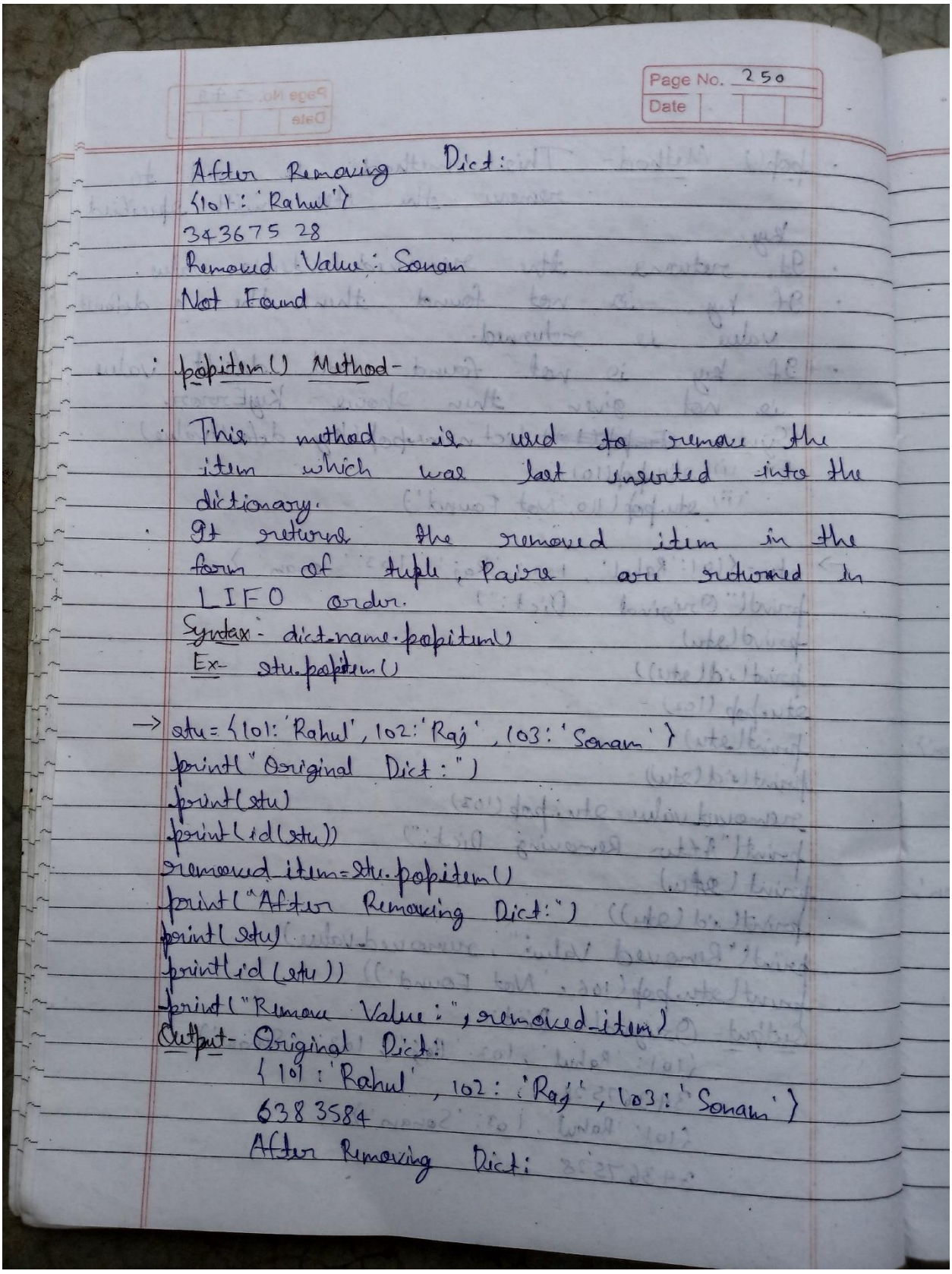
- pop() Method- This method is used to remove the item with specified key.
- It returns the removed item's value.
- If key is not found then the a default value is returned.
- If key is not found and default value is not given then shows KeyError.

Syntax- ~~stu.pop(key)~~ dict_name.pop(key, defaultvalue)

Ex- (i) stu.pop(101)
(ii) stu.pop(110, 'Not Found')

→
```
stu={101:'Rahul', 102:'Raj', 103:'Sonam'}
print("Original Dict:")
print(stu)
print(id(stu))
stu.pop(102)
print(stu)
print(id(stu))
removed_value=stu.pop(103)
print("After Removing Dict:")
print(stu)
print(id(stu))
print("Removed Value:", removed_value)
print(stu.pop(106, 'Not Found'))
```

Output- Original Dict:
{101: 'Rahul', 102: 'Raj', 103: 'Sonam'}
34367528
{101: 'Rahul', 103: 'Sonam'}
34367528



Page No. 250
Date

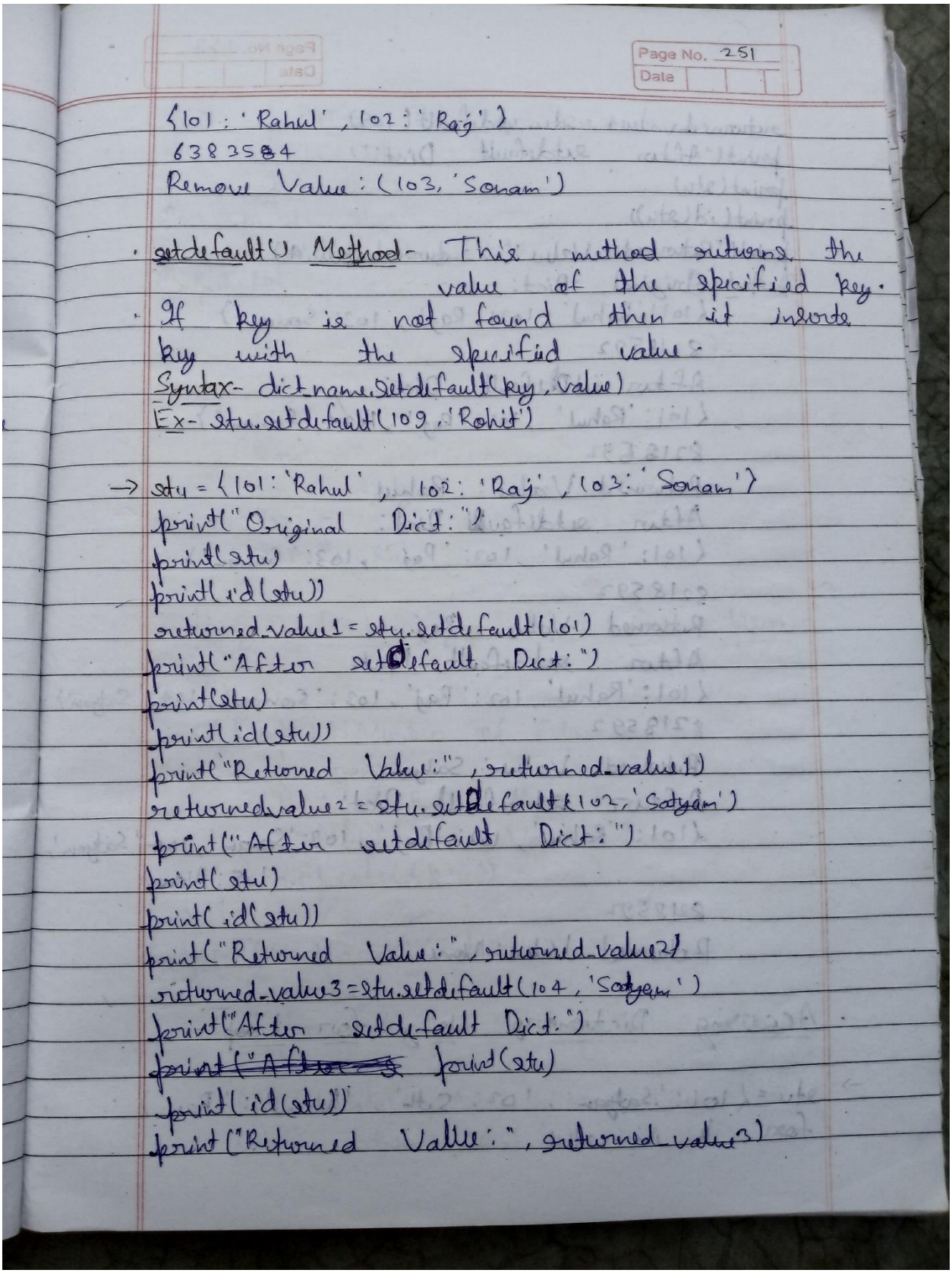
After Removing Dict:
{101: 'Rahul'}
34367528
Removed Value: Sonam
Not Found

- popitem() Method-
  - This method is used to remove the item which was last inserted into the dictionary.
  - It returns the removed item in the form of tuple, Pairs are returned in LIFO order.

Syntax- dict_name.popitem()
Ex- stu.popitem()

```
→ stu={101:'Rahul',102:'Raj',103:'Sonam'}
print("Original Dict:")
print(stu)
print(id(stu))
removed_item=stu.popitem()
print("After Removing Dict:")
print(stu)
print(id(stu))
print("Remove Value:",removed_item)
```

Output- Original Dict:
{101: 'Rahul', 102: 'Raj', 103: 'Sonam'}
6383584
After Removing Dict:



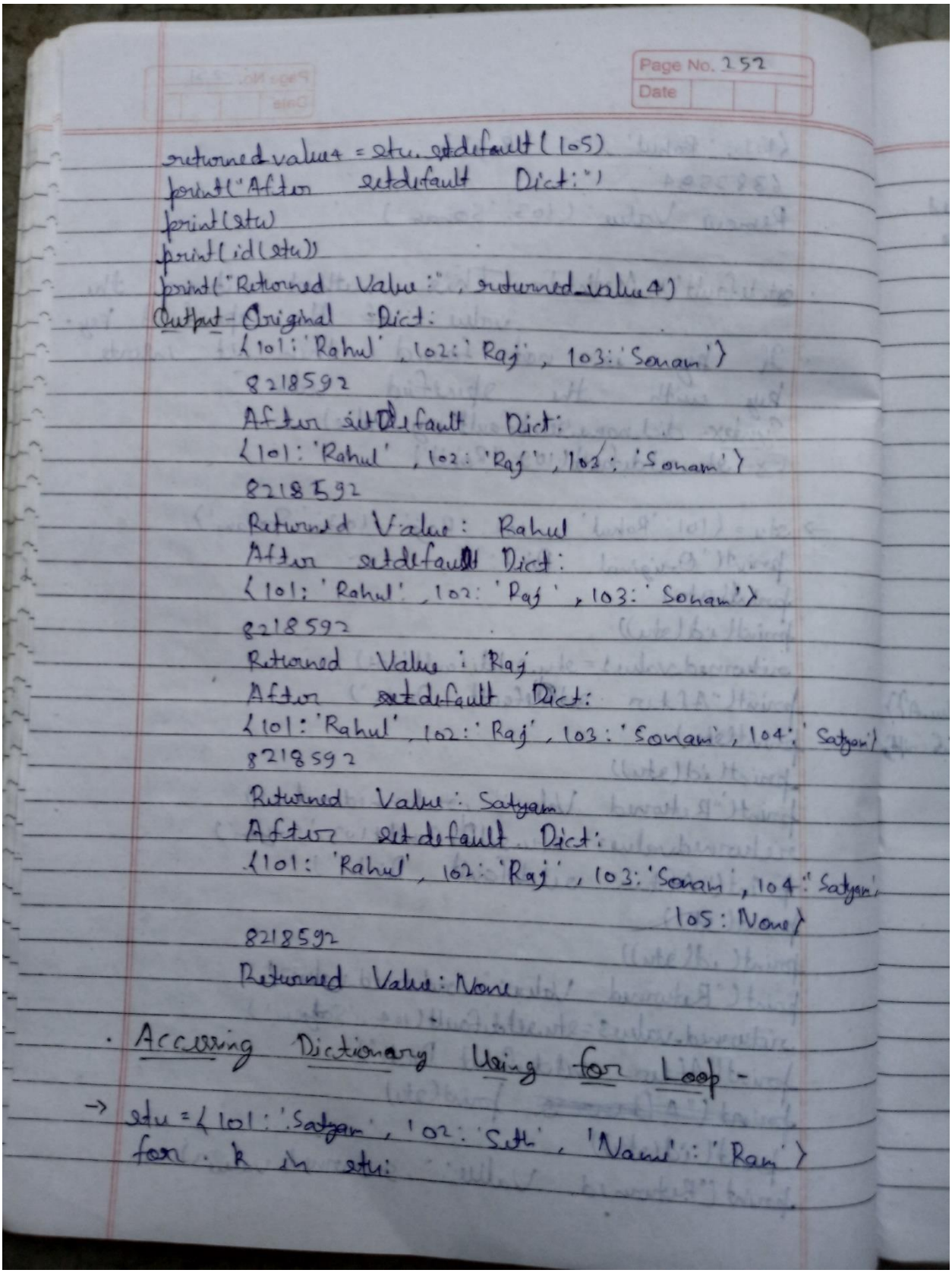
Page No. 251
Date

{101: 'Rahul', 102: 'Raj'}
6383584
Remove Value: (103, 'Sonam')

- setdefault() Method - This method returns the value of the specified key.
- If key is not found then it inserts key with the specified value.

Syntax - dict_name.setdefault(key, value)
Ex - stu.setdefault(109, 'Rohit')

```
stu = {101: 'Rahul', 102: 'Raj', 103: 'Sonam'}
print("Original Dict:")
print(stu)
print(id(stu))
returned_value1 = stu.setdefault(101)
print("After setdefault Dict:")
print(stu)
print(id(stu))
print("Returned Value:", returned_value1)
returned_value2 = stu.setdefault(102, 'Satyam')
print("After setdefault Dict:")
print(stu)
print(id(stu))
print("Returned Value:", returned_value2)
returned_value3 = stu.setdefault(104, 'Satyam')
print("After setdefault Dict:")
print(stu)
print(id(stu))
print("Returned Value:", returned_value3)
```

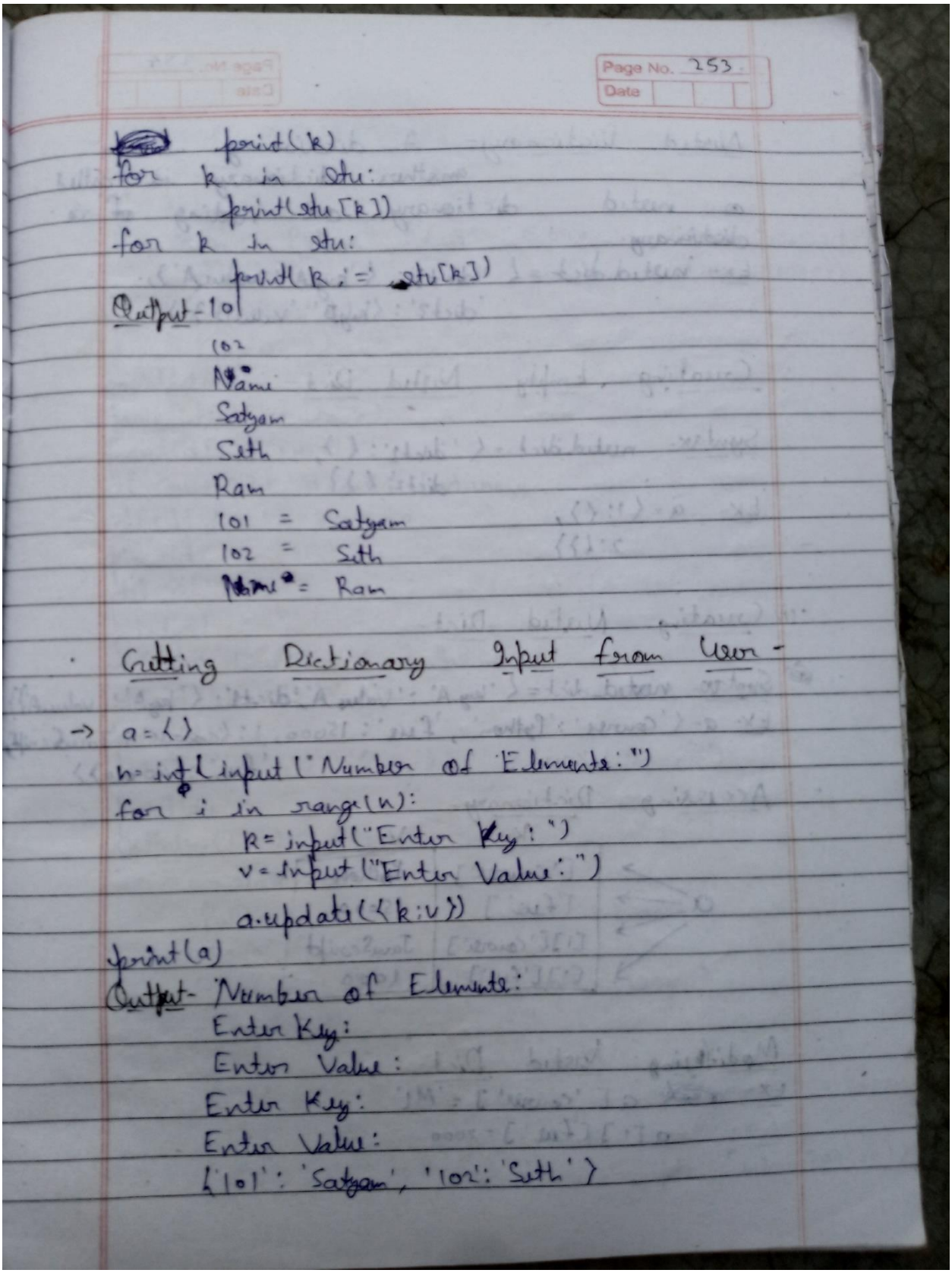


returned_value4 = stu.setdefault(105)
print("After setdefault Dict:")
print(stu)
print(id(stu))
print("Returned Value:", returned_value4)

Output - Original Dict:
{101: 'Rahul', 102: 'Raj', 103: 'Sonam'}
8218592
After setdefault Dict:
{101: 'Rahul', 102: 'Raj', 103: 'Sonam'}
8218592
Returned Value: Rahul
After setdefault Dict:
{101: 'Rahul', 102: 'Raj', 103: 'Sonam'}
8218592
Returned Value: Raj
After setdefault Dict:
{101: 'Rahul', 102: 'Raj', 103: 'Sonam', 104: 'Satyam'}
8218592
Returned Value: Satyam
After setdefault Dict:
{101: 'Rahul', 102: 'Raj', 103: 'Sonam', 104: 'Satyam', 105: None}
8218592
Returned Value: None

· Accessing Dictionary Using for Loop -

→ stu = {101: 'Satyam', 102: 'Seth', 'Name': 'Ram'}
for k in stu:



Page No. 253
Date

```
print(k)
for k in stu:
    print(stu[k])
for k in stu:
    print(k,':',stu[k])
```

Output - 101
102
Name
Satyam
Seth
Ram
101 = Satyam
102 = Seth
Name = Ram

- Getting Dictionary Input from User -

→
```
a={}
n=int(input("Number of Elements:"))
for i in range(n):
    k=input("Enter Key:")
    v=input("Enter Value:")
    a.update({k:v})
print(a)
```

Output - Number of Elements:
Enter Key:
Enter Value:
Enter Key:
Enter Value:
{'101': 'Satyam', '102': 'Seth'}

Page No. 254
Date

- **Nested Dictionary-** A dictionary within another dictionary is called as nested dictionary or nesting of a dictionary.

Ex- nested_dict = { 'dict1': { 'key_A': 'value_A' },
'dict2': {'keyB': 'valueB' } }

- **Creating Empty Nested Dict-**

Syntax- nested_dict = { 'dict1': { },
'dict2': { } }

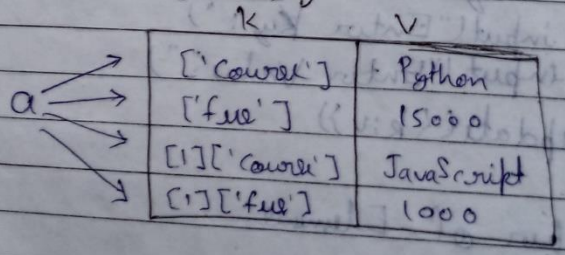
Ex- a = { 1: { },
2: { } }

- (i) **Creating Nested Dict-**

Syntax- nested_dict = { 'key_A' : 'Value_A', 'dict1': { 'keyA' : value_A }

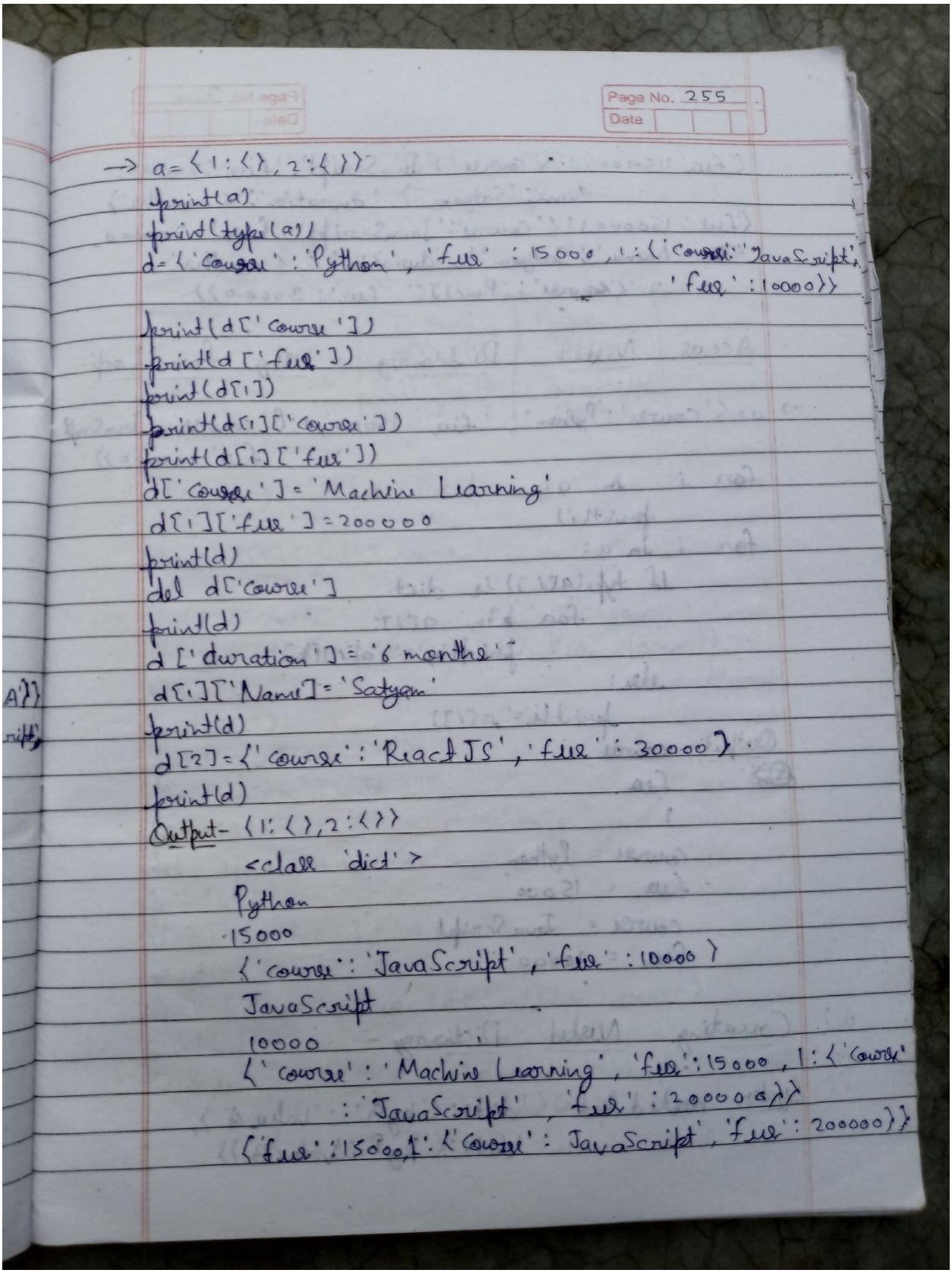
Ex- a = { 'course' : 'Python' , 'fee' : 15000, 1: {'course' : JavaScript,
'fee' : 10000} }

- **Accessing Dictionary-**



- **Modifying Nested Dict-**

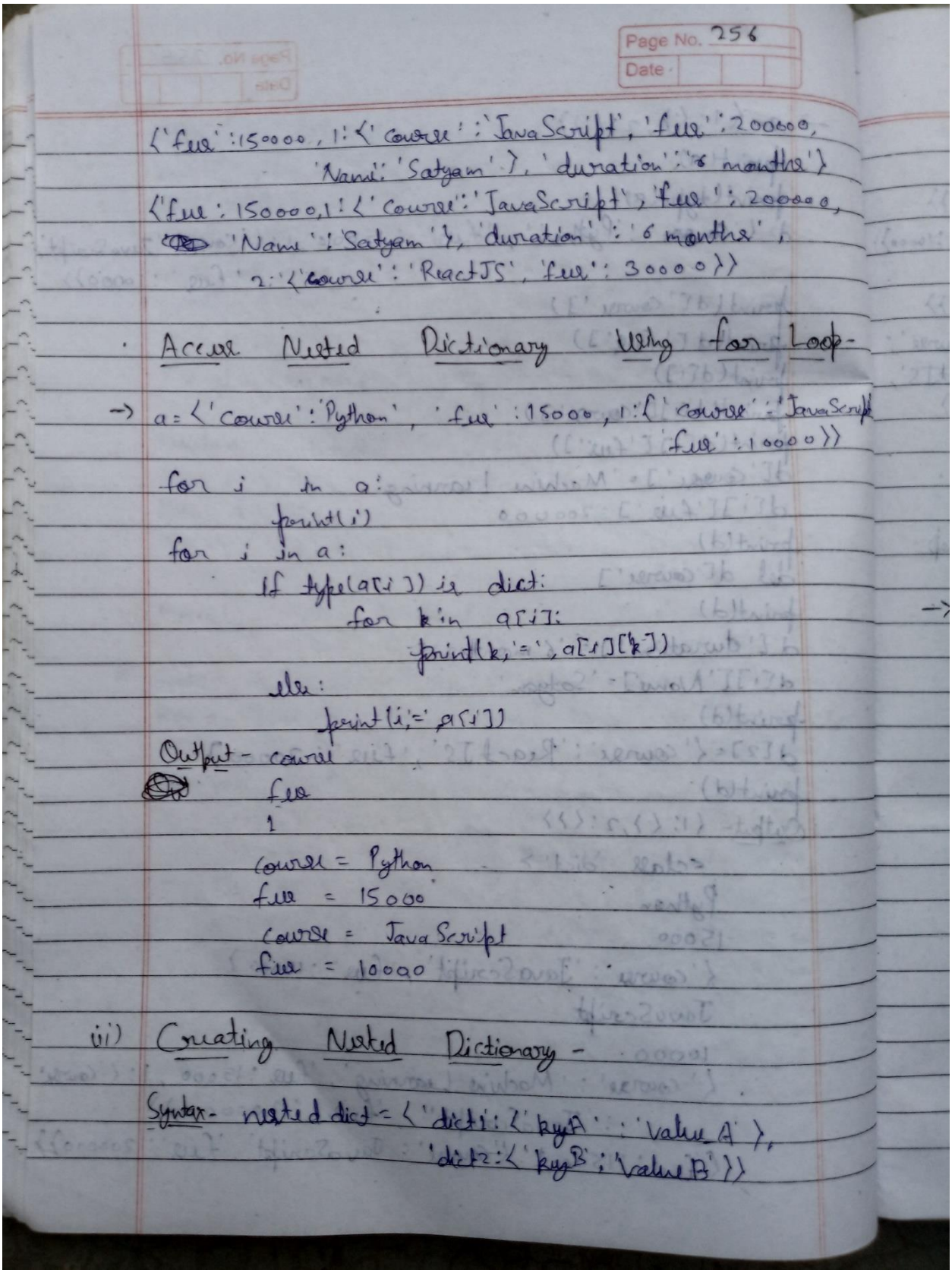
Ex- a['course'] = 'ML'
a[1]['fee'] = 2000



```
a={1:{},2:{}}
print(a)
print(type(a))
d={'course':'Python','fee':15000,1:{'course':'JavaScript',
                                    'fee':10000}}

print(d['course'])
print(d['fee'])
print(d[1])
print(d[1]['course'])
print(d[1]['fee'])
d['course']='Machine Learning'
d[1]['fee']=200000
print(d)
del d['course']
print(d)
d['duration']='6 months'
d[1]['Name']='Satyam'
print(d)
d[2]={'course':'ReactJS','fee':30000}
print(d)
```

Output-

```
{1:{},2:{}}
<class 'dict'>
Python
15000
{'course':'JavaScript','fee':10000}
JavaScript
10000
{'course':'Machine Learning','fee':15000,1:{'course'
:'JavaScript','fee':200000}}
{'fee':15000,1:{'course':JavaScript,'fee':200000}}
```



{'fee':15000, 1:{'course':'JavaScript', 'fee':20000, 'Name':'Satyam'}, 'duration':'6 months'}
{'fee':15000, 1:{'course':'JavaScript', 'fee':20000, 'Name':'Satyam'}, 'duration':'6 months', 2:{'course':'ReactJS', 'fee':30000}}

- Access Nested Dictionary Using for Loop -

→ a={'course':'Python', 'fee':15000, 1:{'course':'JavaScript', 'fee':10000}}

```
for i in a:
    print(i)
for i in a:
    if type(a[i]) is dict:
        for k in a[i]:
            print(k,'=',a[i][k])
    else:
        print(i,'=',a[i])
```

Output - course
fee
1
course = Python
fee = 15000
course = JavaScript
fee = 10000

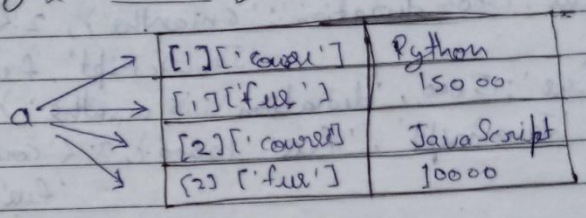
ii) Creating Nested Dictionary -

Syntax- nested dict = {'dict1':{'key1':'value1'}, 'dict2':{'key2':'value2'}}

Page No. 257
Date

Ex- a= {1:{'course':'Python', 'fee':15000},
2:{'course':'JavaScript', 'fee':10000}}

Accessing a Nested Dictionary



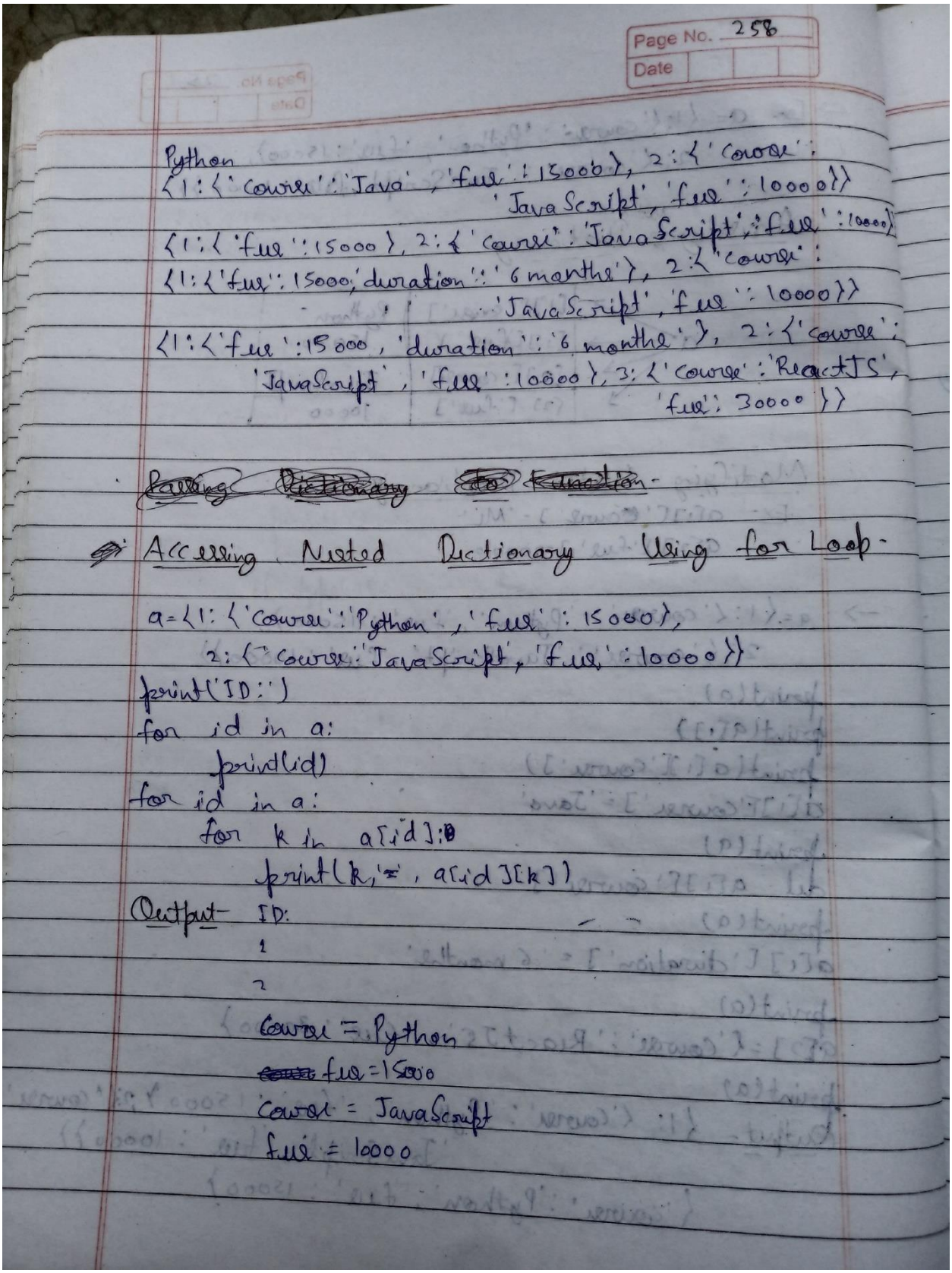
Modifying Nested Dictionary-

Ex- a[1]['course']='ML'
a[2]['fee']=20000

-> a={1:{'course':'Python', 'fee':15000},
2:{'course':'JavaScript', 'fee':10000}}

```
print(a)
print(a[1])
print(a[1]['course'])
a[1]['course']='Java'
print(a)
del a[1]['course']
print(a)
a[1]['duration']='6 months'
print(a)
a[3]={'course':'ReactJS', 'fee':30000}
print(a)
```

Output- {1:{'course':'Python', 'fee':15000}, 2:{'course':'JavaScript', 'fee':10000}}
{'course':'Python', 'fee':15000}



Page No. 258
Date

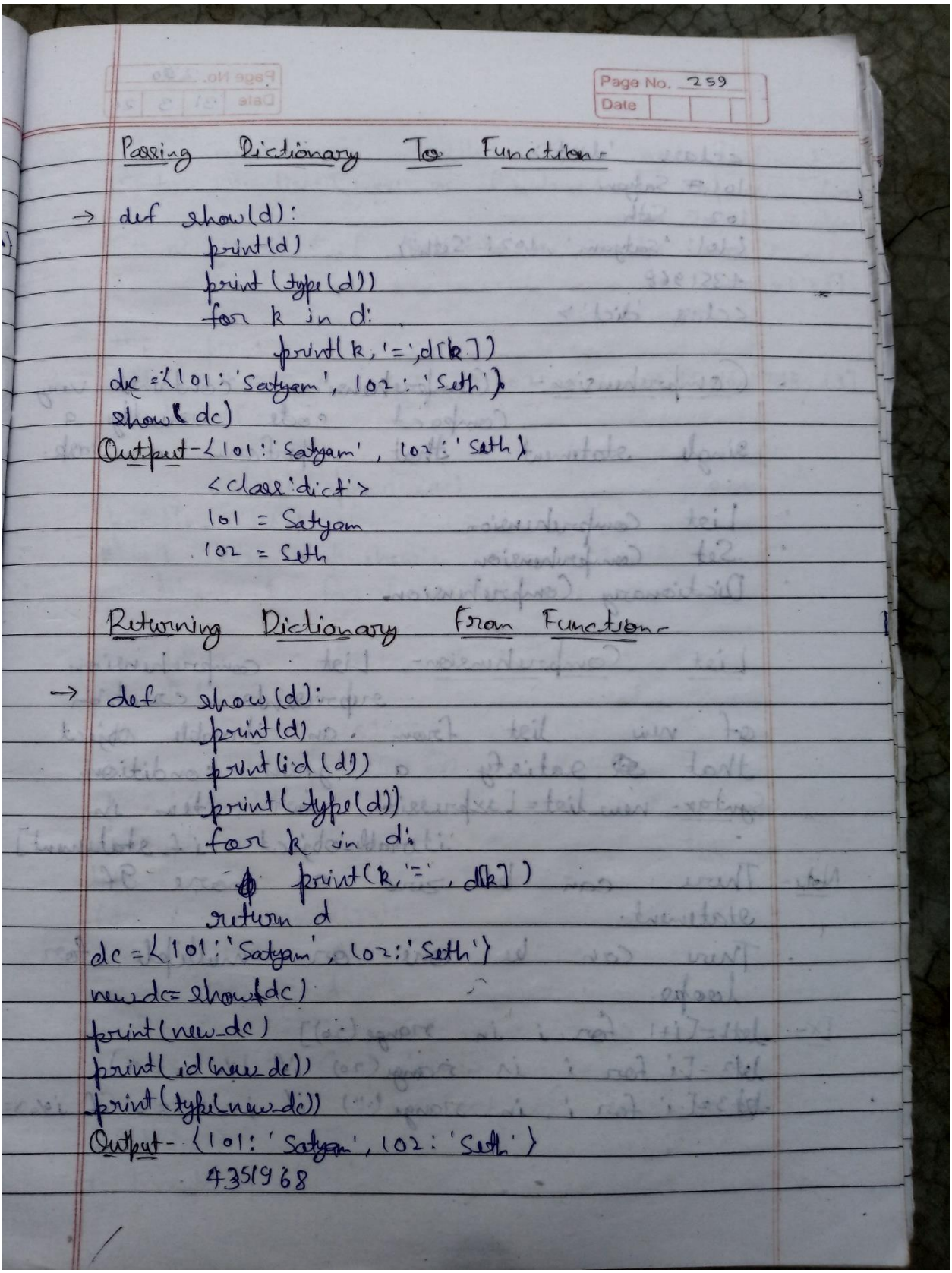
Python
{1:{'course':'Java','fee':15000}, 2:{'course':'JavaScript','fee':10000}}
{1:{'fee':15000}, 2:{'course':'JavaScript','fee':10000}}
{1:{'fee':15000,'duration':'6 months'}, 2:{'course':'JavaScript','fee':10000}}
{1:{'fee':15000,'duration':'6 months'}, 2:{'course':'JavaScript','fee':10000}, 3:{'course':'ReactJS','fee':30000}}

~~Passing Dictionary to Function~~

Accessing Nested Dictionary Using for Loop.

```
a={1:{'course':'Python','fee':15000},
   2:{'course':'JavaScript','fee':10000}}
print('ID:')
for id in a:
    print(id)
for id in a:
    for k in a[id]:
        print(k,'=',a[id][k])
```

Output
```
ID:
1
2
course = Python
fee = 15000
course = JavaScript
fee = 10000
```



Page No. 259
Date

Passing Dictionary To Function-

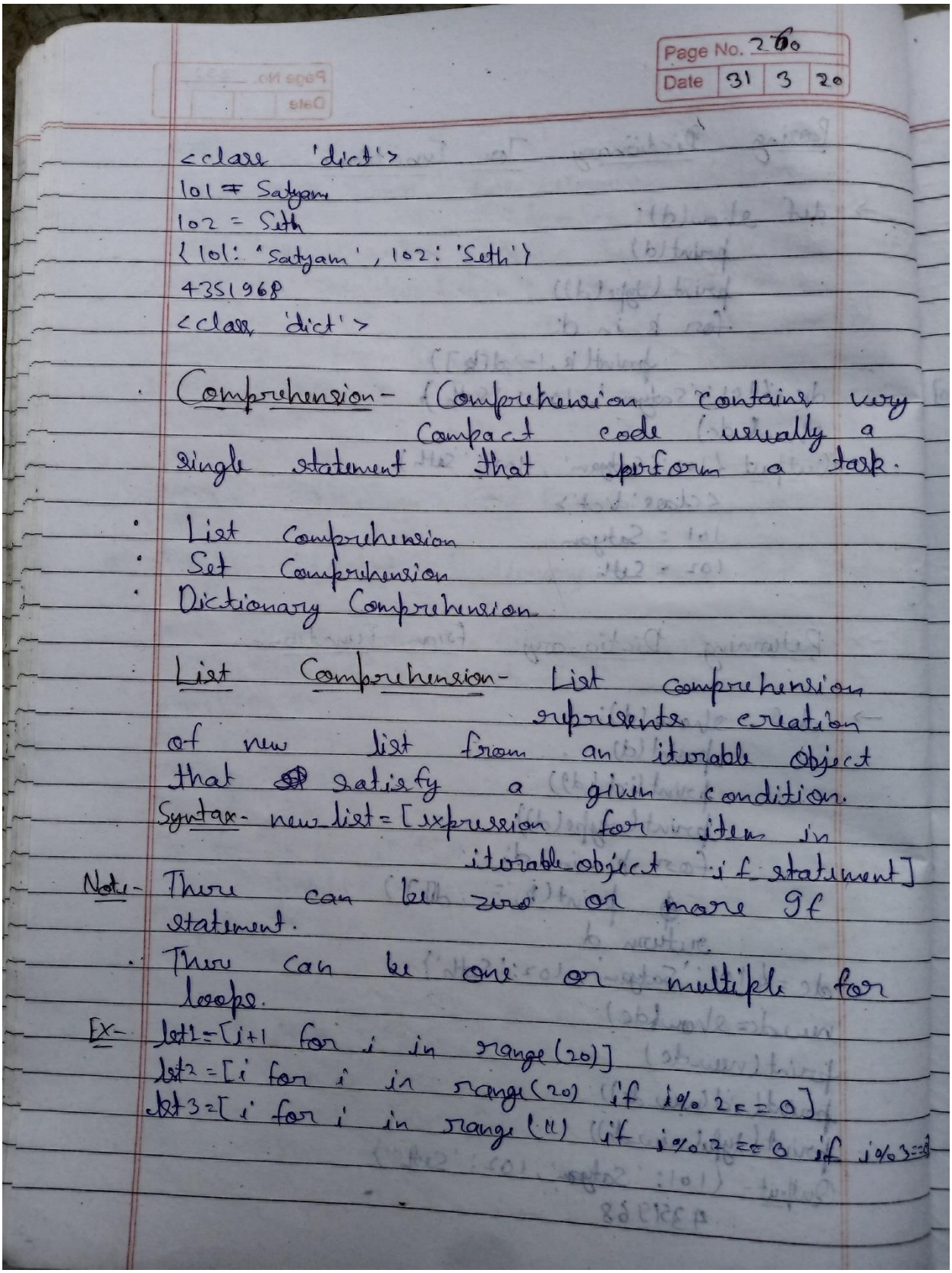
```
→ def show(d):
      print(d)
      print(type(d))
      for k in d:
          print(k,'=',d[k])
dc={101:'Satyam',102:'Seth'}
show(dc)
```

Output- {101:'Satyam', 102:'Seth'}
<class 'dict'>
101 = Satyam
102 = Seth

Returning Dictionary From Function-

```
→ def show(d):
      print(d)
      print(id(d))
      print(type(d))
      for k in d:
          print(k,'=',d[k])
      return d
dc={101:'Satyam',102:'Seth'}
new_dc=show(dc)
print(new_dc)
print(id(new_dc))
print(type(new_dc))
```

Output- {101:'Satyam', 102:'Seth'}
43519368



Page No. 260
Date 31 3 20

<class 'dict'>
101 = Satyam
102 = Seth
{101: 'Satyam', 102: 'Seth'}
4351968
<class 'dict'>

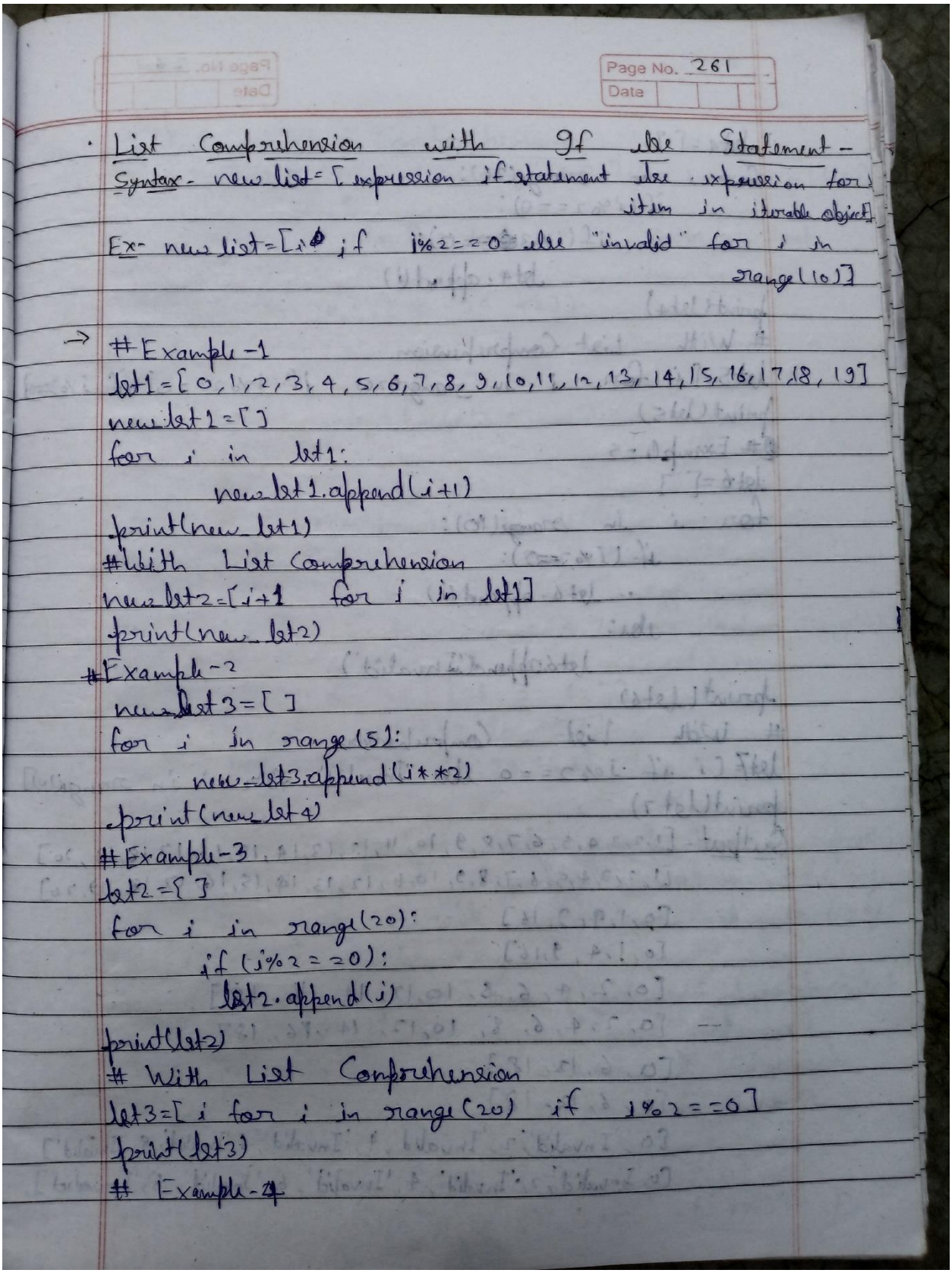
- Comprehension- Comprehension contains very Compact code usually a single statement that perform a task.
  - List Comprehension
  - Set Comprehension
  - Dictionary Comprehension

List Comprehension- List comprehension represents creation of new list from an iterable object that satisfy a given condition.

Syntax- new_list = [expression for item in iterable_object if statement]

Note- There can be zero or more if statement.
- There can be one or multiple for loops.

Ex- lst1 = [i+1 for i in range(20)]
lst2 = [i for i in range(20) if i%2==0]
lst3 = [i for i in range(11) if i%2==0 if i%3==0]



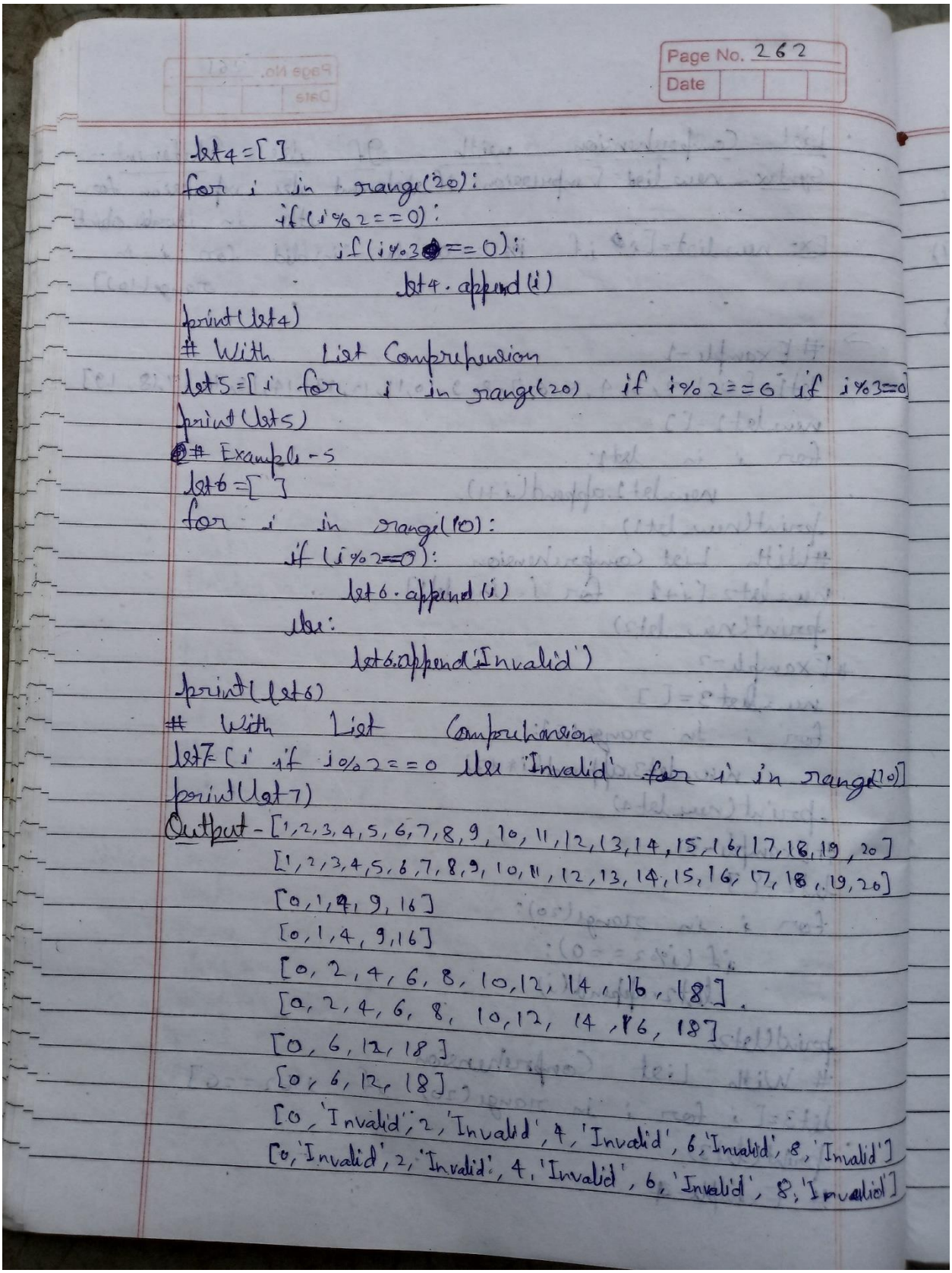
Page No. 261
Date

- List Comprehension with If else Statement -

Syntax - new list = [expression if statement else expression for item in iterable object]

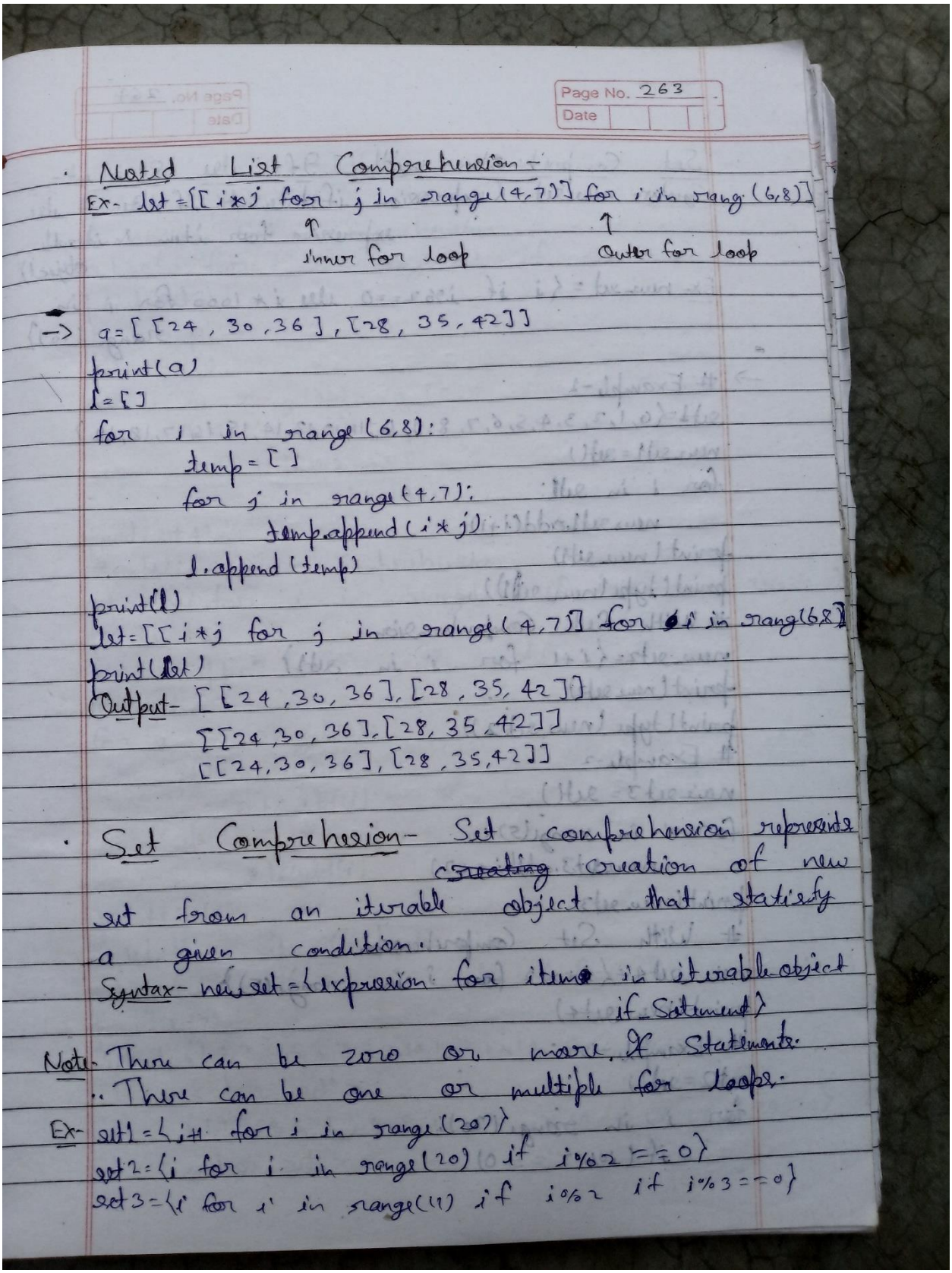
Ex- new list = [i if i%2==0 else "invalid" for i in range(10)]

→
```
#Example-1
lst1=[0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19]
new_lst1=[]
for i in lst1:
    new_lst1.append(i+1)
print(new_lst1)
#With List Comprehension
new_lst2=[i+1 for i in lst1]
print(new_lst2)
#Example-2
new_lst3=[]
for i in range(5):
    new_lst3.append(i**2)
print(new_lst4)
#Example-3
lst2=[]
for i in range(20):
    if(i%2==0):
        lst2.append(i)
print(lst2)
# With List Comprehension
lst3=[i for i in range(20) if i%2==0]
print(lst3)
# Example-4
```



```
lst4=[]
for i in range(20):
    if(i%2==0):
        if(i%3==0):
            lst4.append(i)
print(lst4)
# With List Comprehension
lst5=[i for i in range(20) if i%2==0 if i%3==0]
print(lst5)
# Example-5
lst6=[]
for i in range(10):
    if(i%2==0):
        lst6.append(i)
    else:
        lst6.append('Invalid')
print(lst6)
# With List Comprehension
lst7=[i if i%2==0 else 'Invalid' for i in range(10)]
print(lst7)
```

Output - [1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20]
[1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20]
[0,1,4,9,16]
[0,1,4,9,16]
[0,2,4,6,8,10,12,14,16,18]
[0,2,4,6,8,10,12,14,16,18]
[0,6,12,18]
[0,6,12,18]
[0,'Invalid',2,'Invalid',4,'Invalid',6,'Invalid',8,'Invalid']
[0,'Invalid',2,'Invalid',4,'Invalid',6,'Invalid',8,'Invalid']



Page No. 263
Date

· Nested List Comprehension -

Ex- lst = [[i*j for j in range(4,7)] for i in rang(6,8)]

inner for loop — Outer for loop

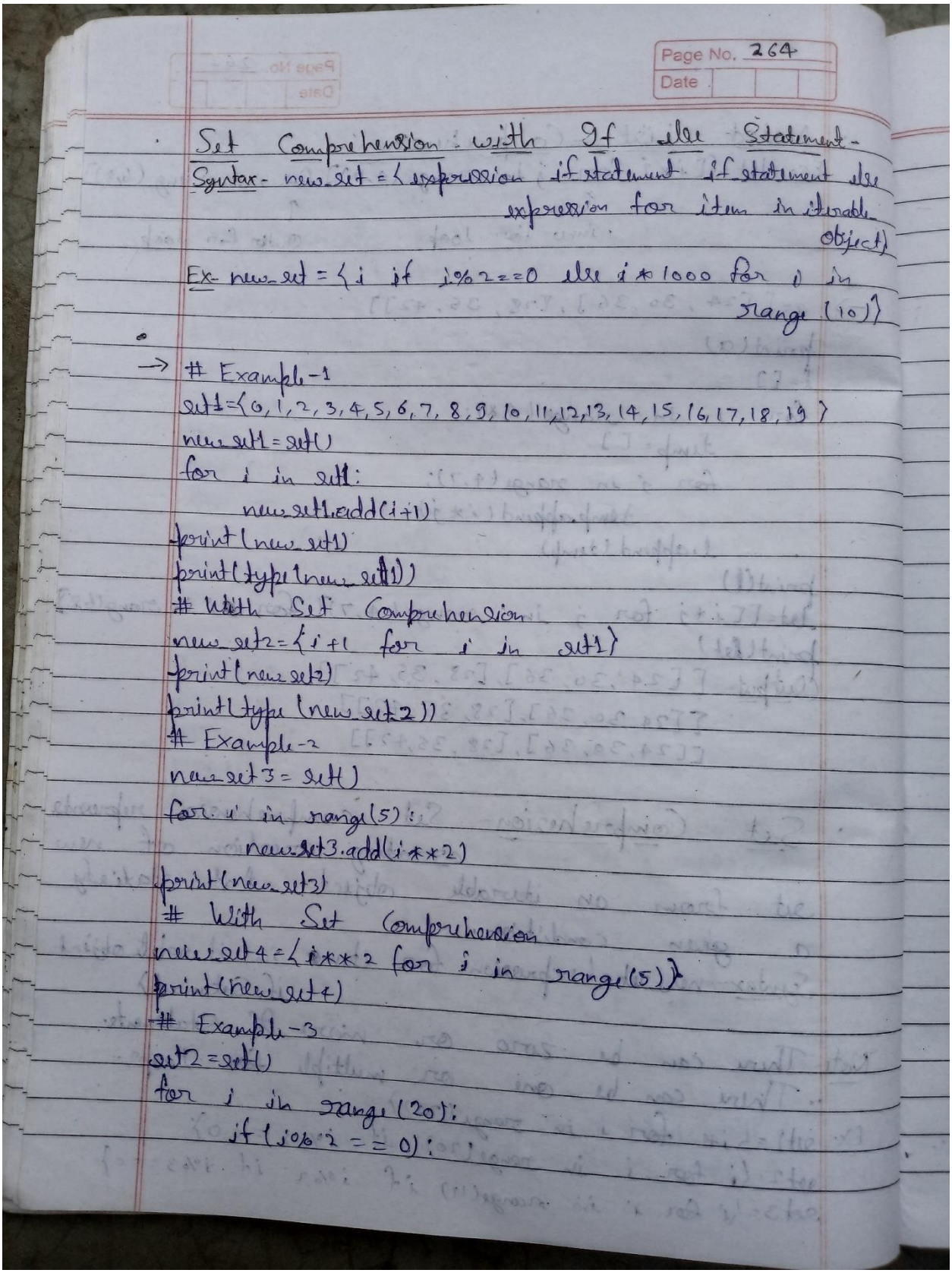
-> a = [[24, 30, 36], [28, 35, 42]]
print(a)
l = []
for i in range(6,8):
    temp = []
    for j in range(4,7):
        temp.append(i*j)
    l.append(temp)
print(l)
lst = [[i*j for j in range(4,7)] for i in rang(6,8)]
print(lst)
Output- [[24, 30, 36], [28, 35, 42]]
[[24, 30, 36], [28, 35, 42]]
[[24, 30, 36], [28, 35, 42]]

· Set Comprehension- Set comprehension represents ~~creating~~ creation of new set from an iterable object that satisfy a given condition.

Syntax- new set = {expression for item in iterable object if Statement}

Note- There can be zero or more if Statements.
.. There can be one or multiple for loops.

Ex- set1 = {i+1 for i in range(20)}
set2 = {i for i in range(20) if i%2 == 0}
set3 = {i for i in range(11) if i%2 if i%3 == 0}

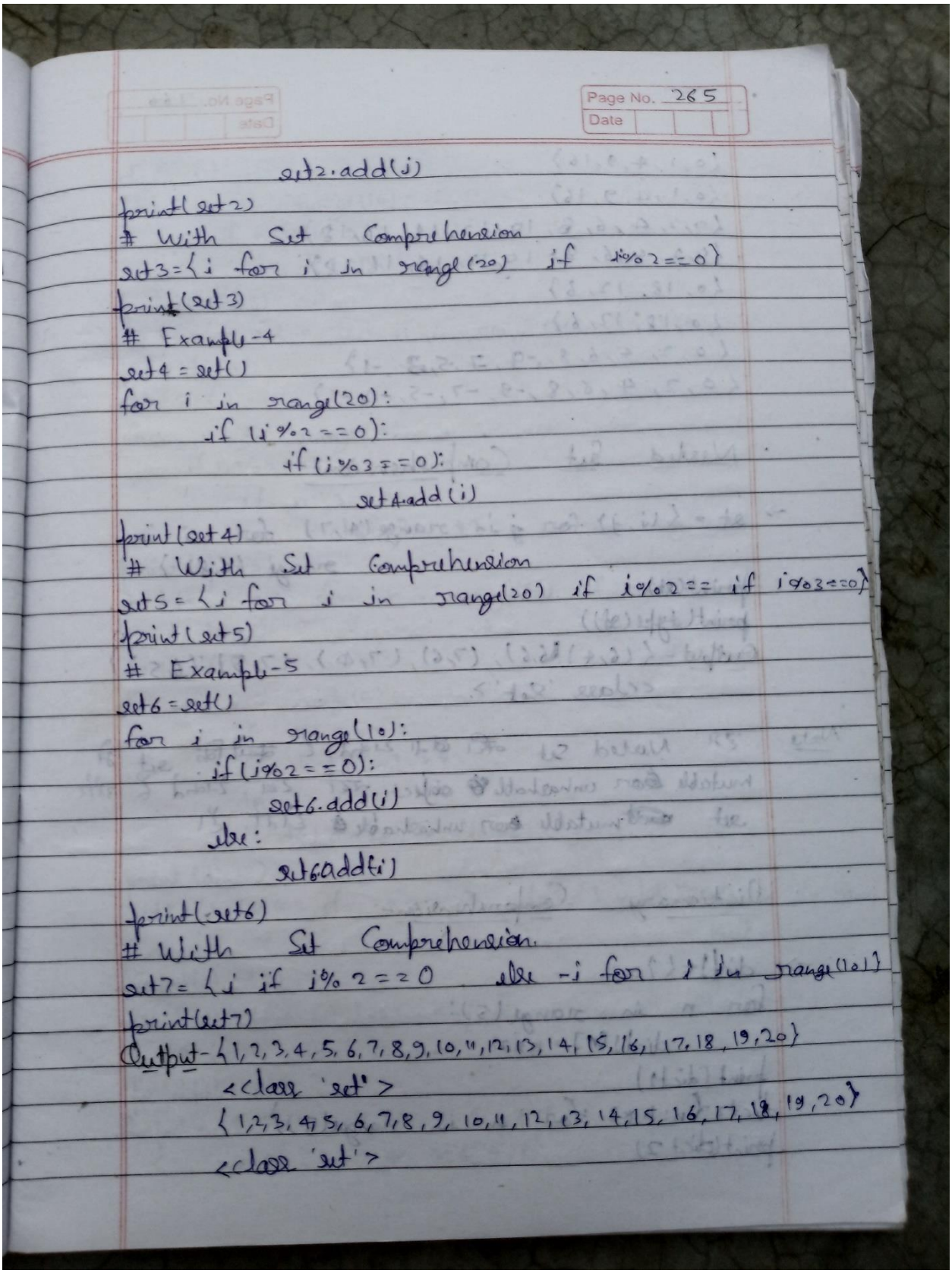


Page No. 264
Date

Set Comprehension with If else Statement -
Syntax - new_set = { expression if statement if statement else expression for item in iterable object }

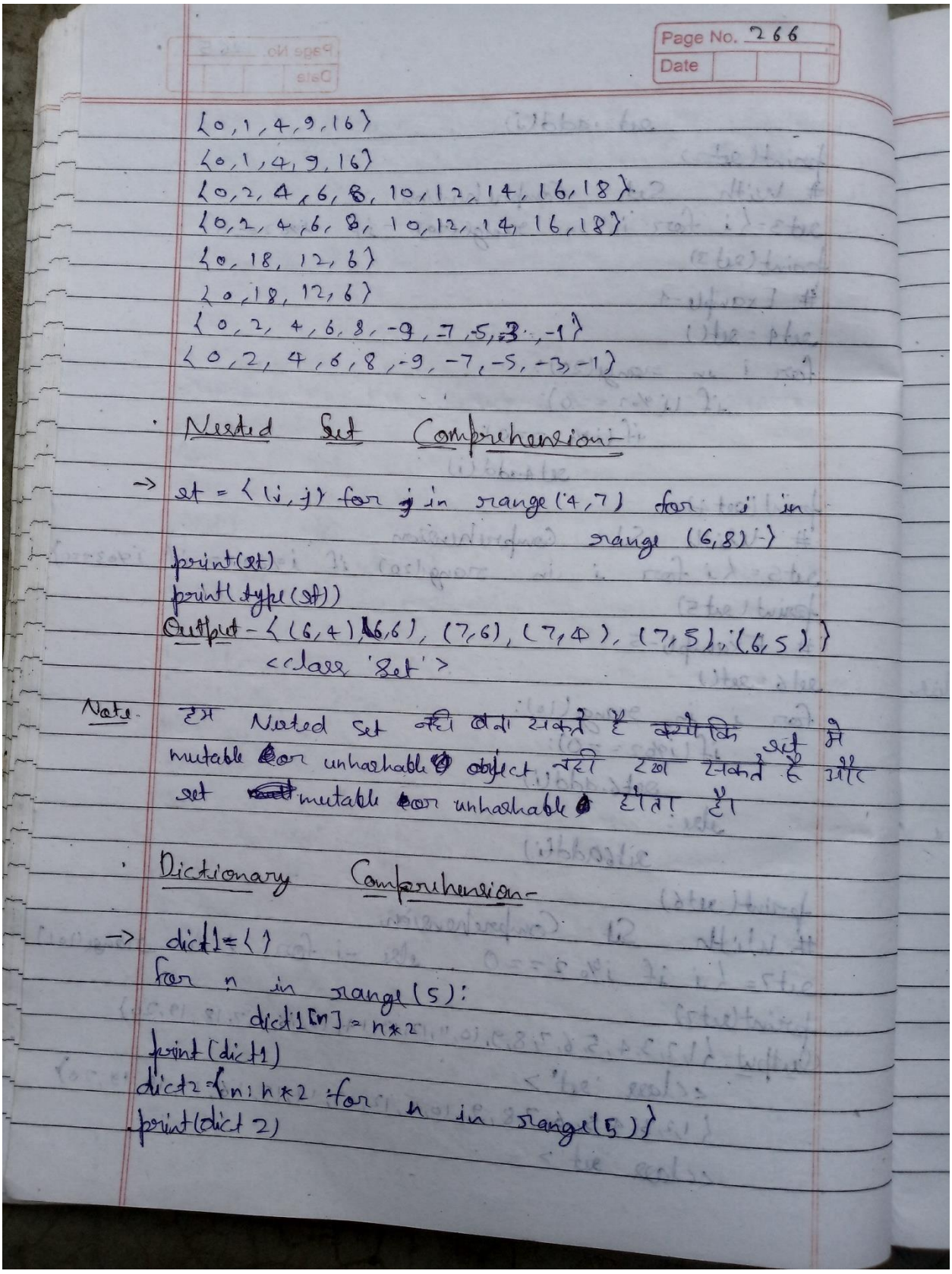
Ex - new_set = { i if i%2==0 else i*1000 for i in range(10) }

→
```
# Example-1
set1={0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19}
new_set1=set()
for i in set1:
    new_set1.add(i+1)
print(new_set1)
print(type(new_set1))
# With Set Comprehension
new_set2={i+1 for i in set1}
print(new_set2)
print(type(new_set2))
# Example-2
new_set3=set()
for i in range(5):
    new_set3.add(i**2)
print(new_set3)
# With Set Comprehension
new_set4={i**2 for i in range(5)}
print(new_set4)
# Example-3
set2=set()
for i in range(20):
    if(i%2==0):
```



```
                set2.add(i)
print(set2)
# With Set Comprehension
set3={i for i in range(20) if i%2==0}
print(set3)
# Example-4
set4=set()
for i in range(20):
    if(i%2==0):
        if(i%3==0):
            set4.add(i)
print(set4)
# With Set Comprehension
set5={i for i in range(20) if i%2== if i%3==0}
print(set5)
# Example-5
set6=set()
for i in range(10):
    if(i%2==0):
        set6.add(i)
    else:
        set6.add(i)
print(set6)
# With Set Comprehension.
set7={i if i%2==0 else -i for i in range(10)}
print(set7)
```

Output-
```
{1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20}
<class 'set'>
{1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20}
<class 'set'>
```



Page No. 266
Date

{0, 1, 4, 9, 16}
{0, 1, 4, 9, 16}
{0, 2, 4, 6, 8, 10, 12, 14, 16, 18}
{0, 2, 4, 6, 8, 10, 12, 14, 16, 18}
{0, 18, 12, 6}
{0, 18, 12, 6}
{0, 2, 4, 6, 8, -9, -7, -5, -3, -1}
{0, 2, 4, 6, 8, -9, -7, -5, -3, -1}

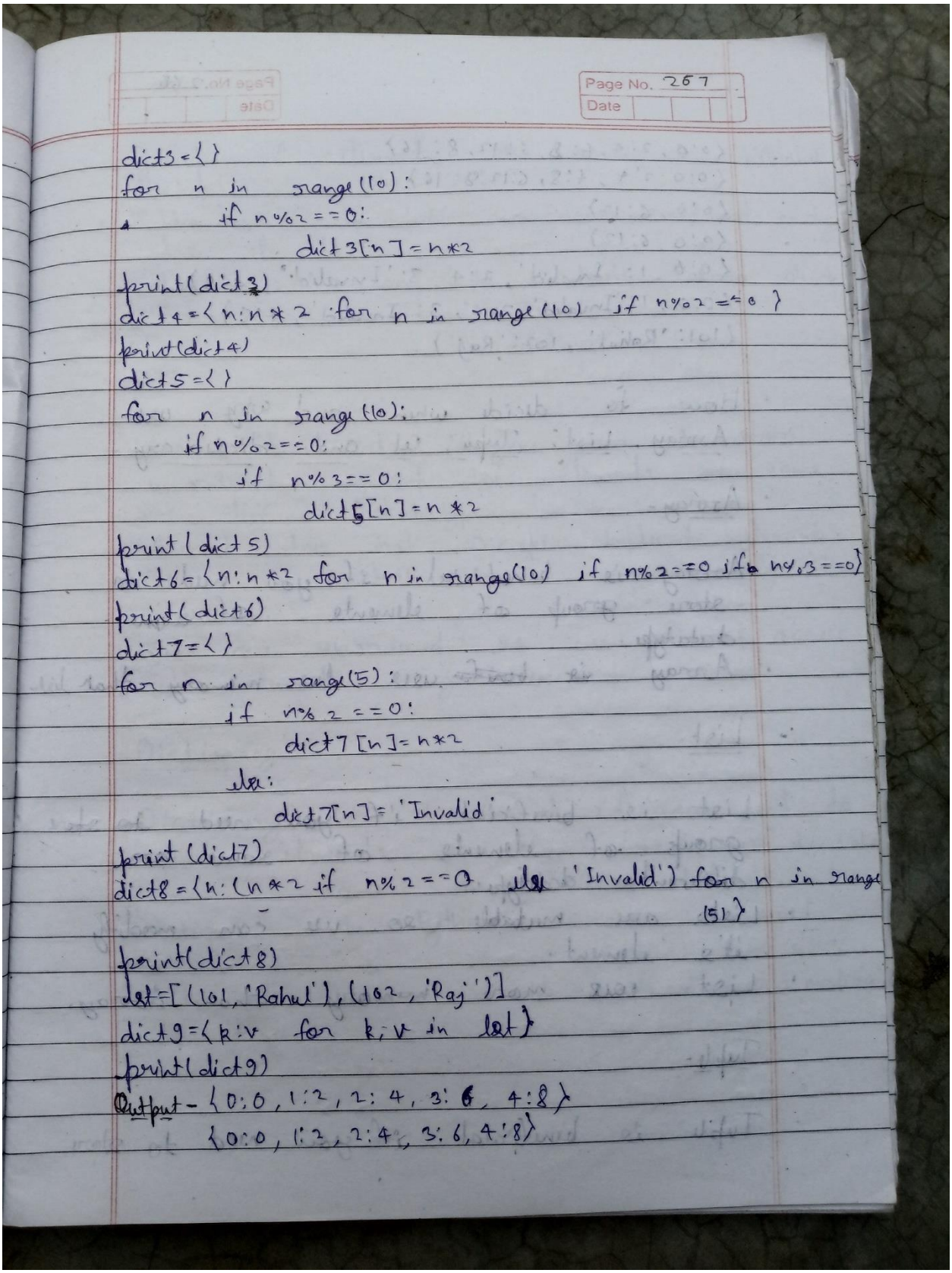
Nested Set Comprehension-

-> st = {(i, j) for j in range(4, 7) for i in range(6, 8)}
print(st)
print(type(st))
Output - {(6, 4), (6, 6), (7, 6), (7, 4), (7, 5), (6, 5)}
<class 'set'>

Note - इस Nested Set नहीं बना सकते है क्योंकि set मे mutable or unhashable object नहीं रख सकते है और set mutable or unhashable होता है।

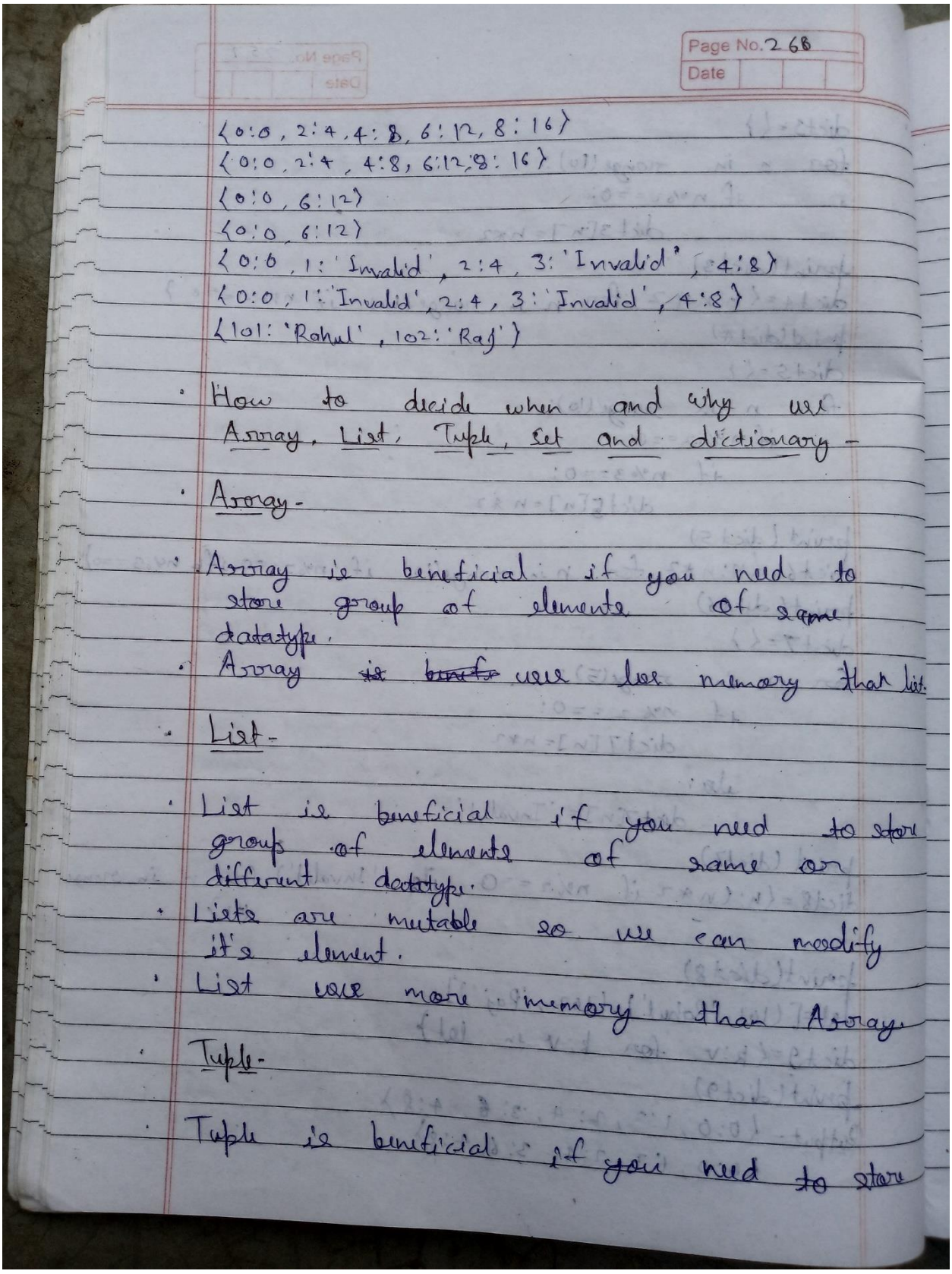
Dictionary Comprehension-

-> dict1 = {}
for n in range(5):
    dict1[n] = n*2
print(dict1)
dict2 = {n: n*2 for n in range(5)}
print(dict2)



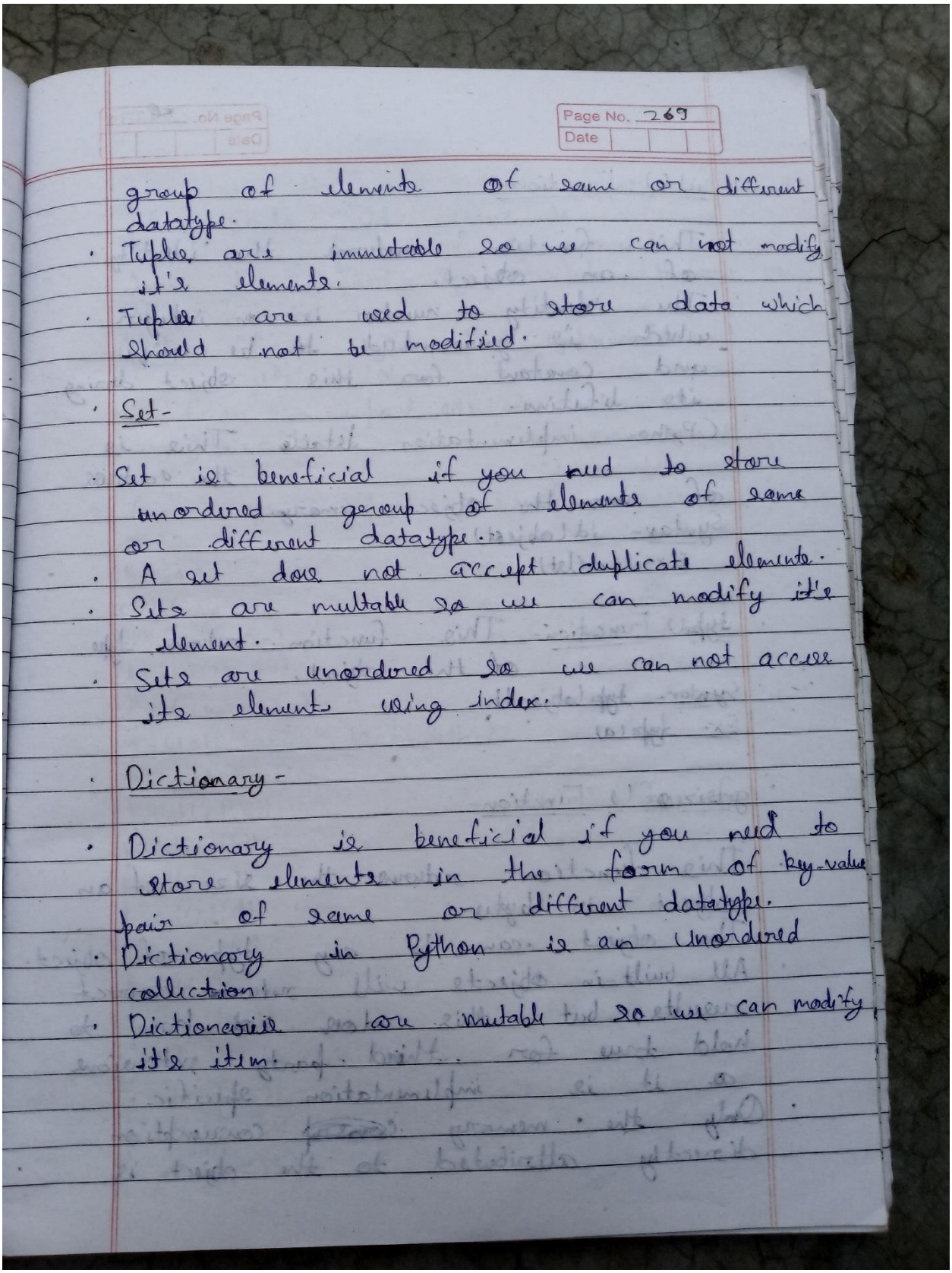
```
dict3={}
for n in range(10):
    if n%2==0:
        dict3[n]=n*2
print(dict3)
dict4={n:n*2 for n in range(10) if n%2==0}
print(dict4)
dict5={}
for n in range(10):
    if n%2==0:
        if n%3==0:
            dict5[n]=n*2
print(dict5)
dict6={n:n*2 for n in range(10) if n%2==0 if n%3==0}
print(dict6)
dict7={}
for n in range(5):
    if n%2==0:
        dict7[n]=n*2
    else:
        dict7[n]='Invalid'
print(dict7)
dict8={n:(n*2 if n%2==0 else 'Invalid') for n in range(5)}
print(dict8)
lst=[(101,'Rahul'),(102,'Raj')]
dict9={k:v for k,v in lst}
print(dict9)
```

Output - {0:0, 1:2, 2:4, 3:6, 4:8}
{0:0, 1:2, 2:4, 3:6, 4:8}



{0:0, 2:4, 4:8, 6:12, 8:16}
{0:0, 2:4, 4:8, 6:12, 8:16}
{0:0, 6:12}
{0:0, 6:12}
{0:0, 1:'Invalid', 2:4, 3:'Invalid', 4:8}
{0:0, 1:'Invalid', 2:4, 3:'Invalid', 4:8}
{101:'Rahul', 102:'Raj'}

- How to decide when and why use Array, List, Tuple, Set and dictionary -
- Array -
- Array is beneficial if you need to store group of elements of same datatype.
- Array ~~is~~ ~~benefit~~ use less memory than list.
- List -
- List is beneficial if you need to store group of elements of same or different datatype.
- Lists are mutable so we can modify it's element.
- List use more memory than Array.
- Tuple -
- Tuple is beneficial if you need to store



Page No. 269
Date

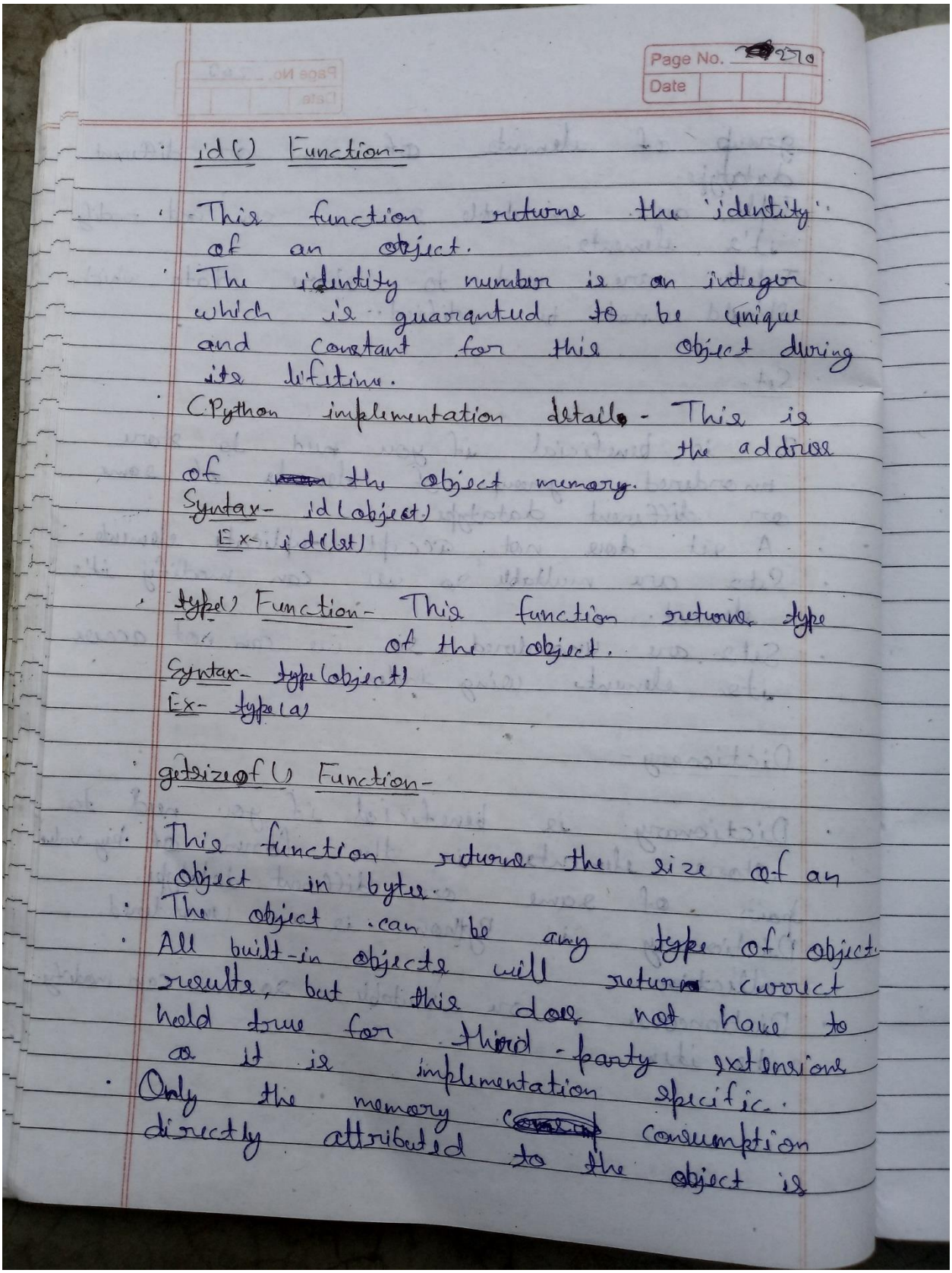
group of elements of same or different datatype.
- Tuples are immutable so we can not modify it's elements.
- Tuples are used to store data which should not be modified.

- Set -

- Set is beneficial if you need to store unordered group of elements of same or different datatype.
- A set does not accept duplicate elements.
- Sets are mutable so we can modify it's element.
- Sets are unordered so we can not access its elements using index.

- Dictionary -

- Dictionary is beneficial if you need to store elements in the form of key-value pair of same or different datatype.
- Dictionary in Python is an unordered collection.
- Dictionaries are mutable so we can modify it's item.



id() Function-

- This function returns the 'identity' of an object.
- The identity number is an integer which is guaranteed to be unique and constant for this object during its lifetime.
- CPython implementation detail - This is the address of the object memory.

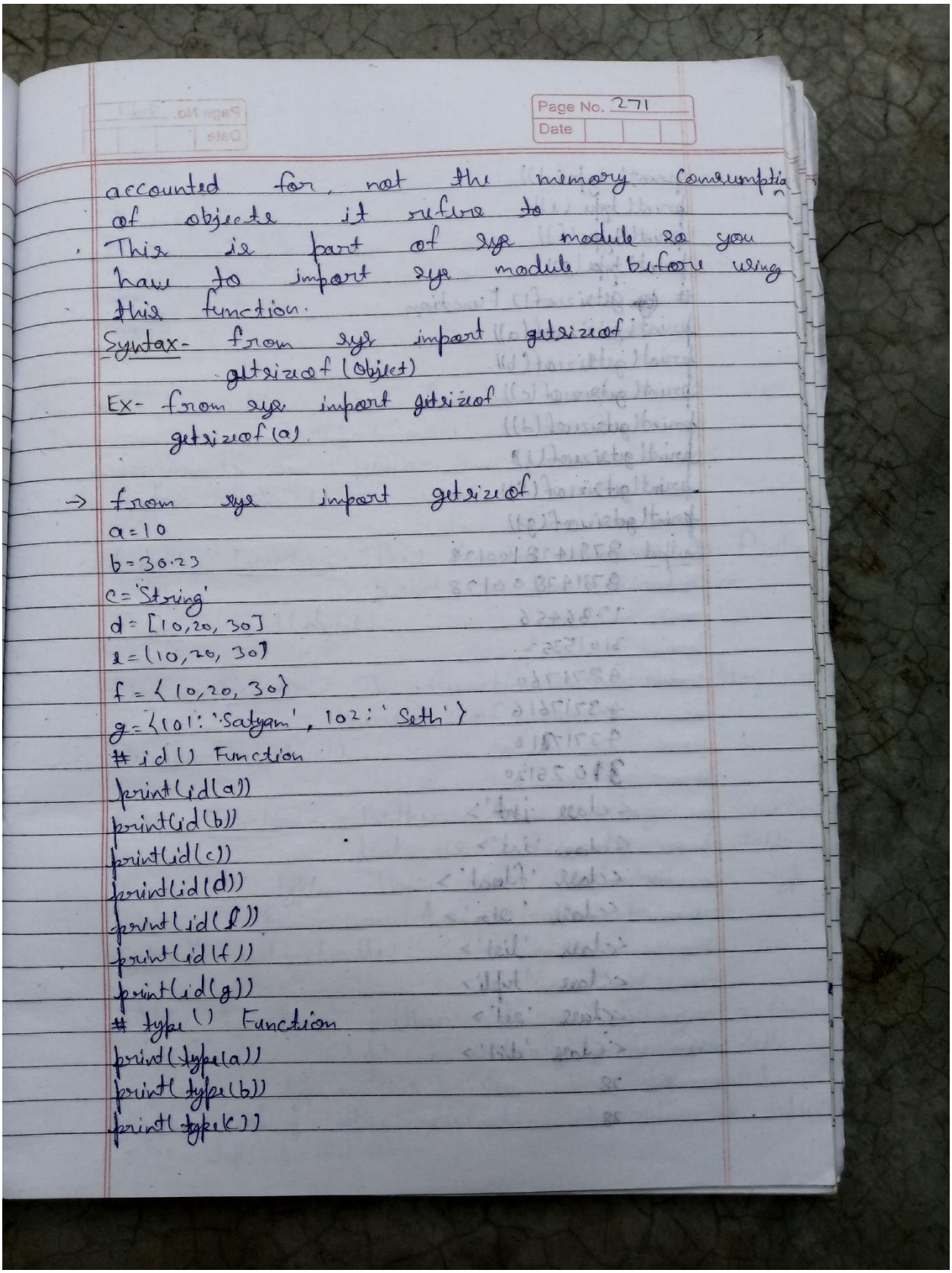
Syntax- id(object)
Ex- id(lst)

- type() Function- This function returns type of the object.

Syntax- type(object)
Ex- type(a)

- getsizeof() Function-

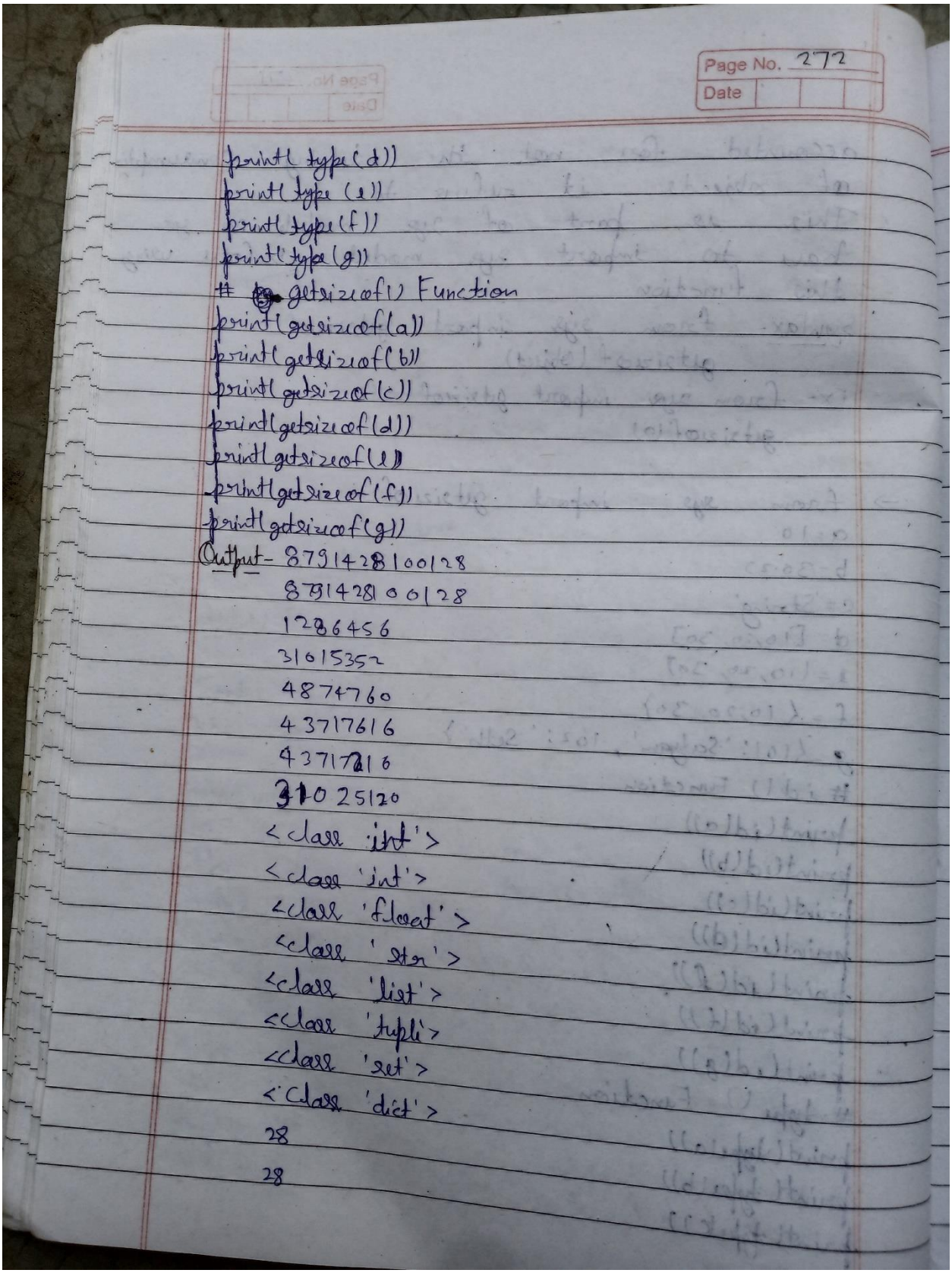
- This function returns the size of an object in bytes.
- The object can be any type of object.
- All built-in objects will return correct results, but this does not have to hold true for third-party extensions as it is implementation specific.
- Only the memory consumption directly attributed to the object is



Page No. 271
Date

accounted for, not the memory consumption of objects it refers to.
This is part of sys module so you have to import sys module before using this function.
Syntax- from sys import getsizeof
getsizeof(object)
Ex- from sys import getsizeof
getsizeof(a)

→ from sys import getsizeof
a=10
b=30.23
c='String'
d=[10,20,30]
t=(10,20,30)
f={10,20,30}
g={101:'Satyam', 102:'Seth'}
# id() Function
print(id(a))
print(id(b))
print(id(c))
print(id(d))
print(id(t))
print(id(f))
print(id(g))
# type() Function
print(type(a))
print(type(b))
print(type(c))

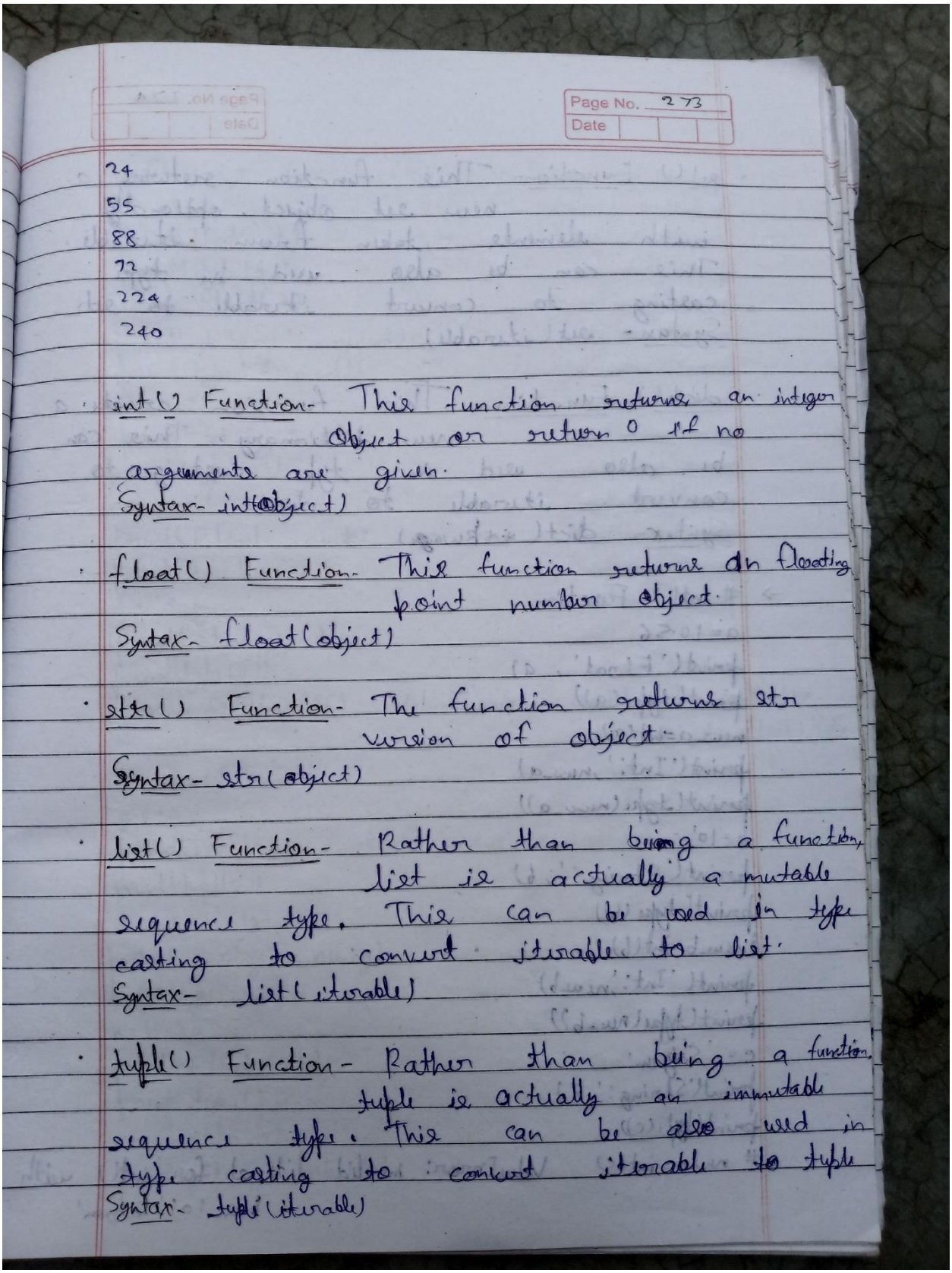


Page No. 272
Date

```
print( type(d))
print( type (e))
print type(f))
print type(g))
# getsizeof() Function
print(getsizeof(a))
print(getsizeof(b))
print(getsizeof(c))
print(getsizeof(d))
print(getsizeof(e))
print(getsizeof(f))
print(getsizeof(g))
```

Output- 879142881 00128

```
879142881 00128
1286456
3101535
4874760
43717616
43717216
3102 5120
<class 'int'>
<class 'int'>
<class 'float'>
<class 'str'>
<class 'list'>
<class 'tuple'>
<class 'set'>
<class 'dict'>
28
28
```



Page No. 273
Date

24
55
88
72
224
240

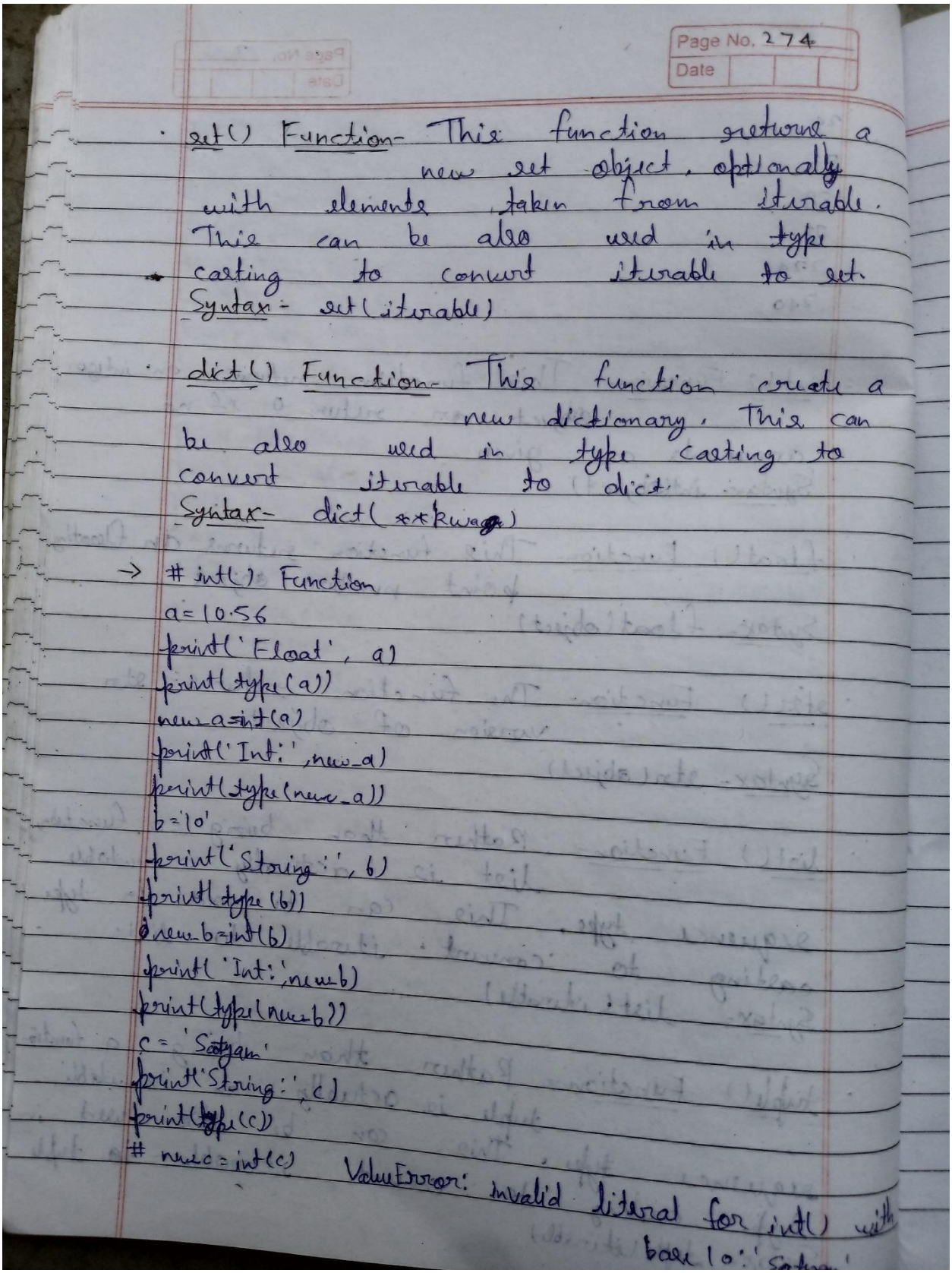
- int() Function- This function returns an integer object or return 0 if no arguments are given.
  Syntax- int(object)

- float() Function- This function returns an floating point number object.
  Syntax- float(object)

- str() Function- The function returns str version of object.
  Syntax- str(object)

- list() Function- Rather than being a function, list is actually a mutable sequence type. This can be used in type casting to convert iterable to list.
  Syntax- list(iterable)

- tuple() Function- Rather than being a function, tuple is actually an immutable sequence type. This can be also used in type casting to convert iterable to tuple
  Syntax- tuple(iterable)

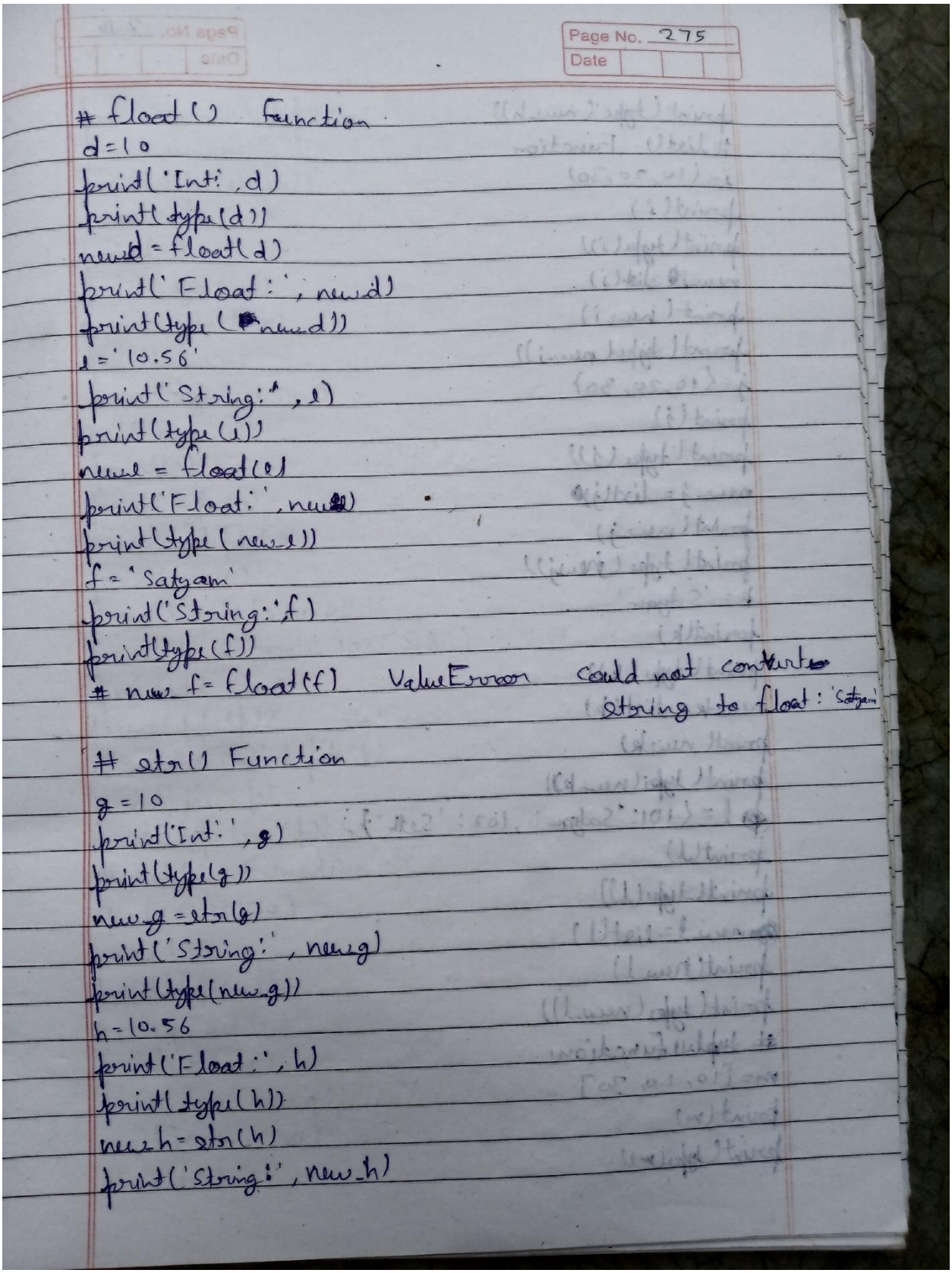


Page No. 274
Date

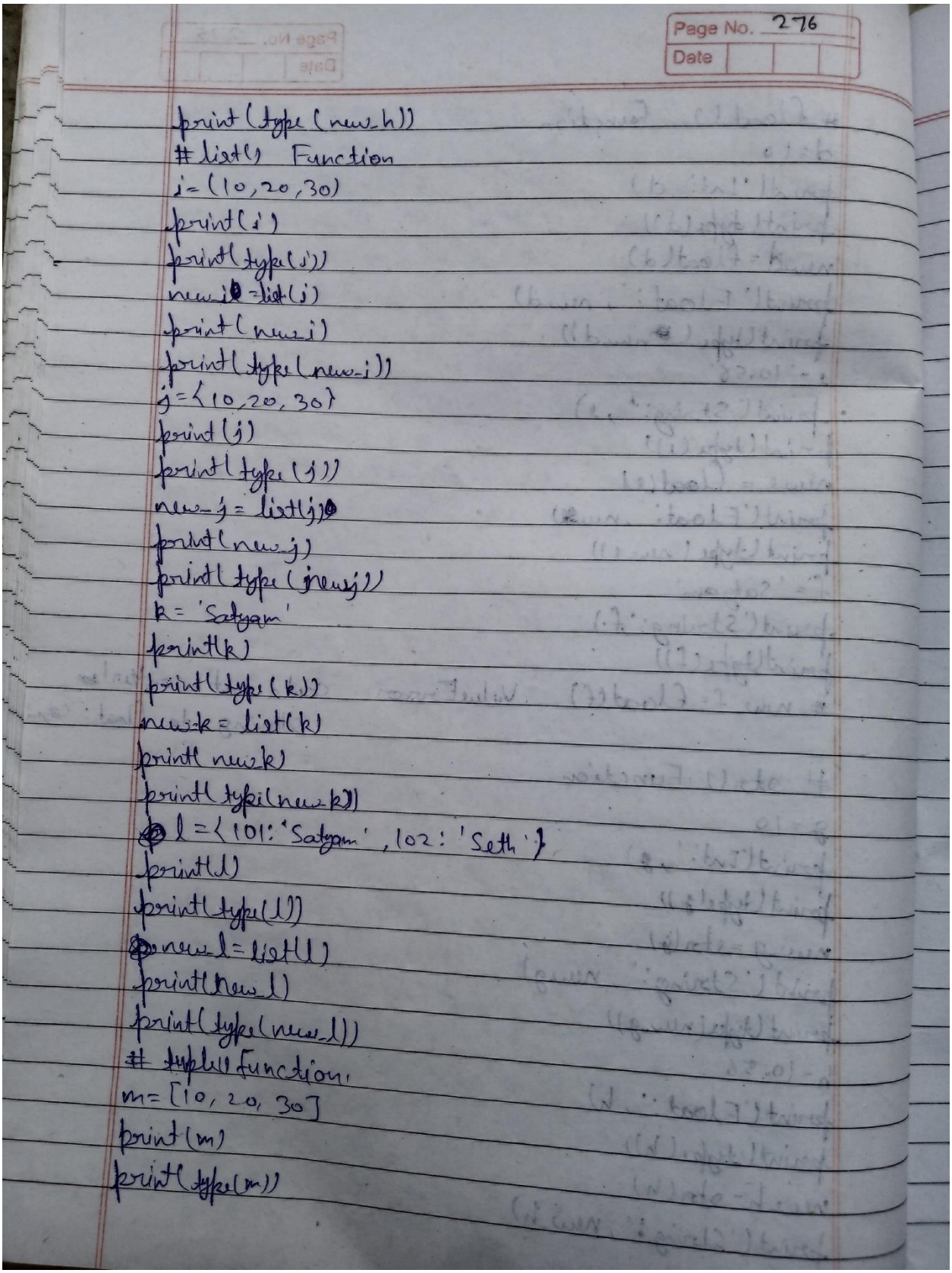
- set() Function- This function returns a new set object, optionally with elements taken from iterable. This can be also used in type casting to convert iterable to set.
  Syntax- set(iterable)

- dict() Function- This function create a new dictionary. This can be also used in type casting to convert iterable to dict.
  Syntax- dict(**kwargs)

→
```
# int() Function
a=10.56
print('Float', a)
print(type(a))
new_a=int(a)
print('Int:', new_a)
print(type(new_a))
b='10'
print('String:', b)
print(type(b))
new_b=int(b)
print('Int:', new_b)
print(type(new_b))
c='Satyam'
print('String:', c)
print(type(c))
# new_c=int(c)  ValueError: invalid literal for int() with base 10: 'Satyam'
```

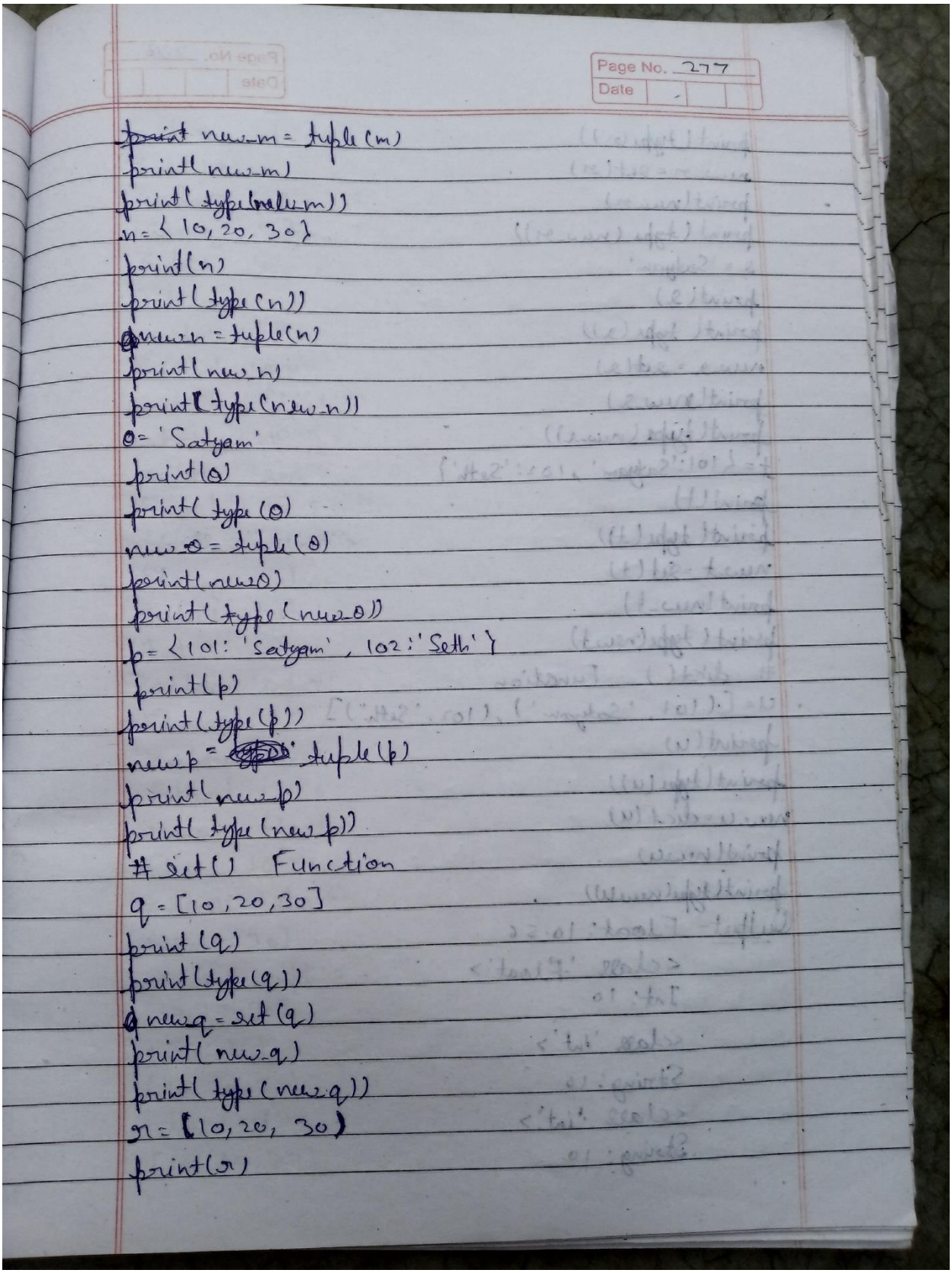


```
# float() Function
d=10
print('Int:',d)
print(type(d))
newd=float(d)
print('Float:',newd)
print(type(newd))
l='10.56'
print('String:',l)
print(type(l))
newl=float(l)
print('Float:',newl)
print(type(newl))
f='Satyam'
print('String:',f)
print(type(f))
# newf=float(f)  ValueError could not convert string to float: 'Satyam'

# str() Function
g=10
print('Int:',g)
print(type(g))
newg=str(g)
print('String:',newg)
print(type(newg))
h=10.56
print('Float:',h)
print(type(h))
newh=str(h)
print('String:',newh)
```

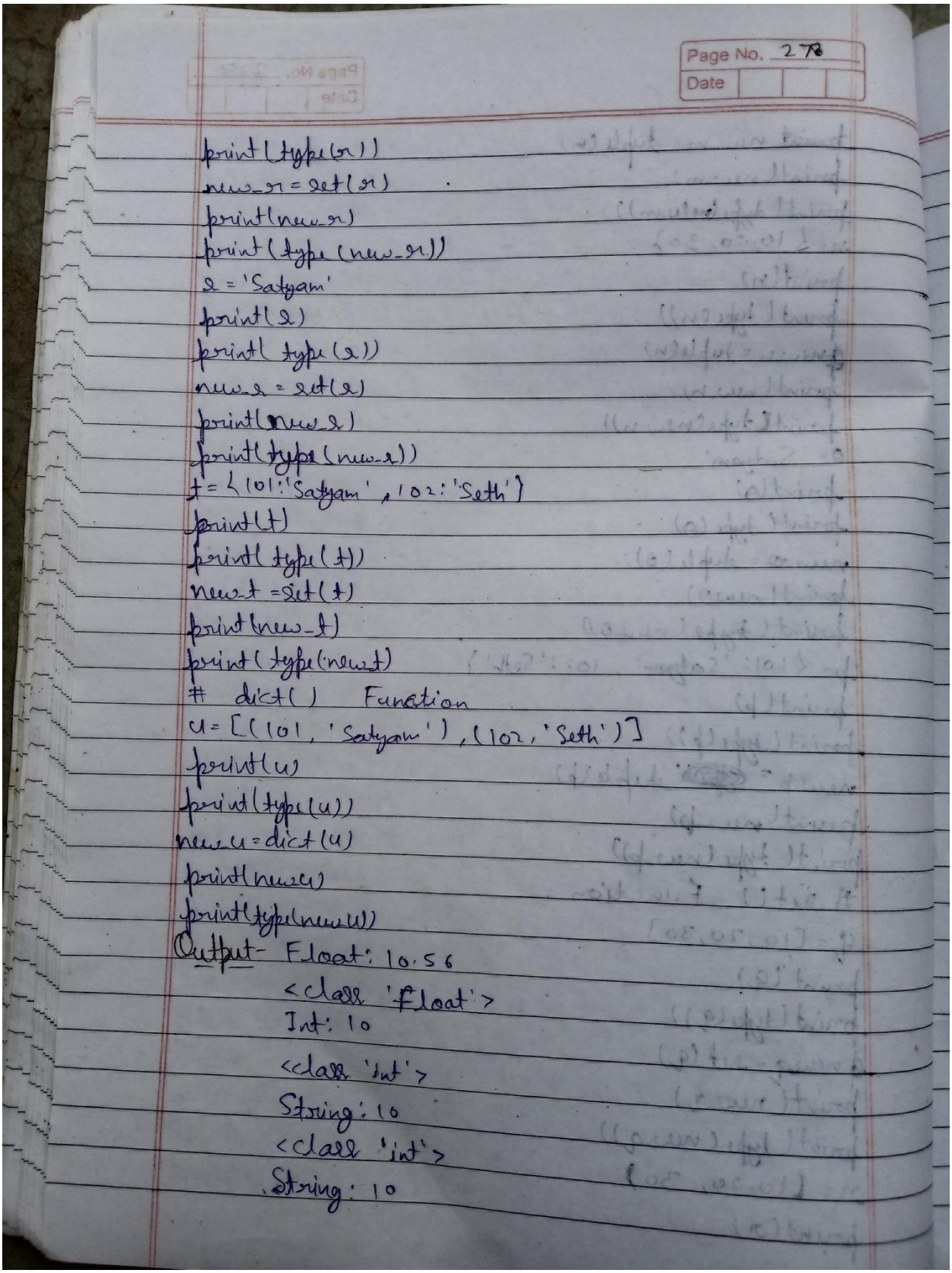


Page No. 276
Date

```
print(type(new_h))
# list() Function
i=(10,20,30)
print(i)
print(type(i))
new_i=list(i)
print(new_i)
print(type(new_i))
j={10,20,30}
print(j)
print(type(j))
new_j=list(j)
print(new_j)
print(type(new_j))
k='Satyam'
print(k)
print(type(k))
new_k=list(k)
print(new_k)
print(type(new_k))
l={101:'Satyam',102:'Seth'}
print(l)
print(type(l))
new_l=list(l)
print(new_l)
print(type(new_l))
# tuple() Function
m=[10,20,30]
print(m)
print(type(m))
```



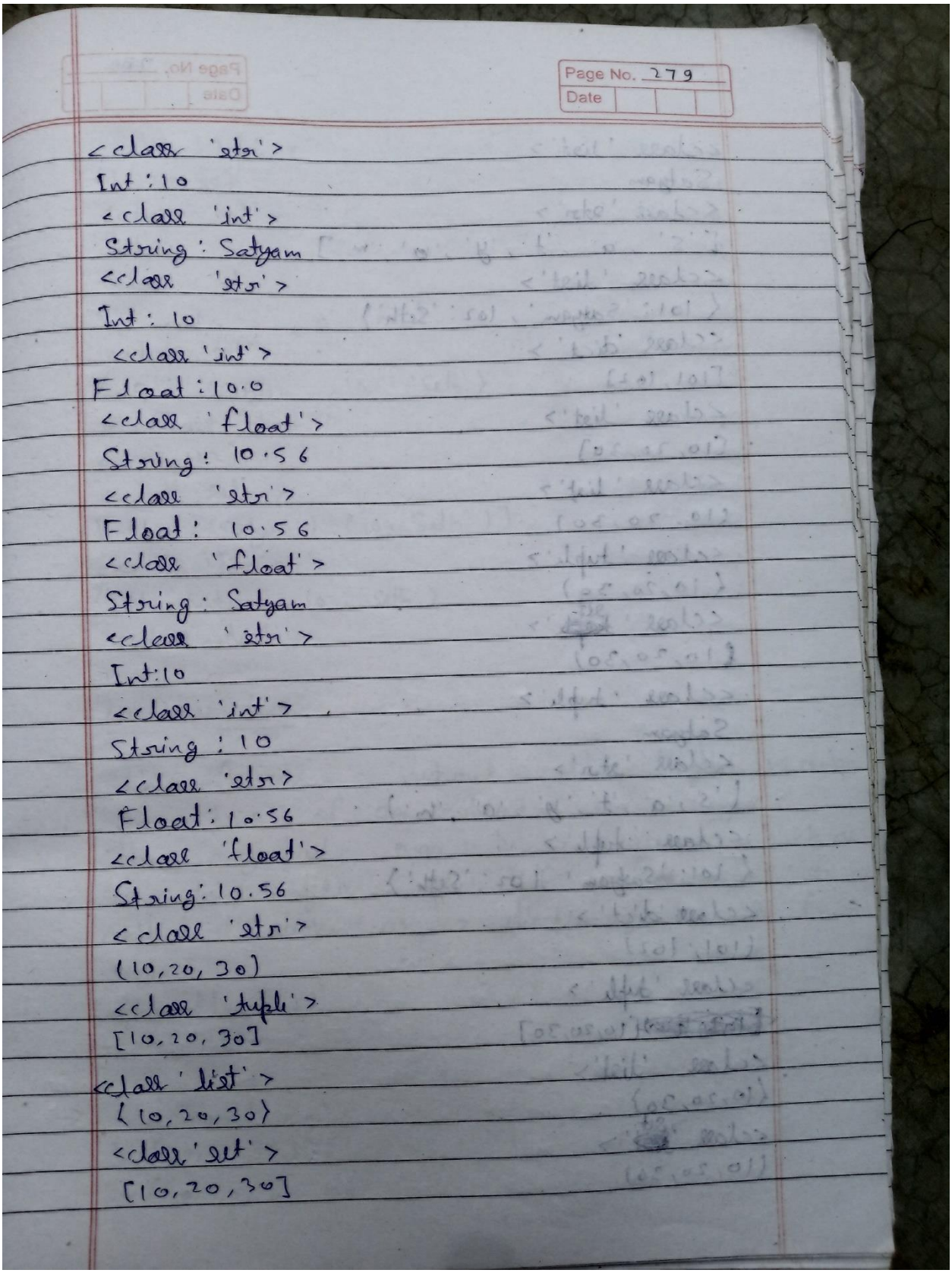
```
print new_m = tuple(m)
print(new_m)
print(type(new_m))
n = {10, 20, 30}
print(n)
print(type(n))
new_n = tuple(n)
print(new_n)
print(type(new_n))
o = 'Satyam'
print(o)
print(type(o))
new_o = tuple(o)
print(new_o)
print(type(new_o))
p = {101: 'Satyam', 102: 'Seth'}
print(p)
print(type(p))
new_p = tuple(p)
print(new_p)
print(type(new_p))
# set() Function
q = [10, 20, 30]
print(q)
print(type(q))
new_q = set(q)
print(new_q)
print(type(new_q))
r = {10, 20, 30}
print(r)
```



Page No. 278
Date

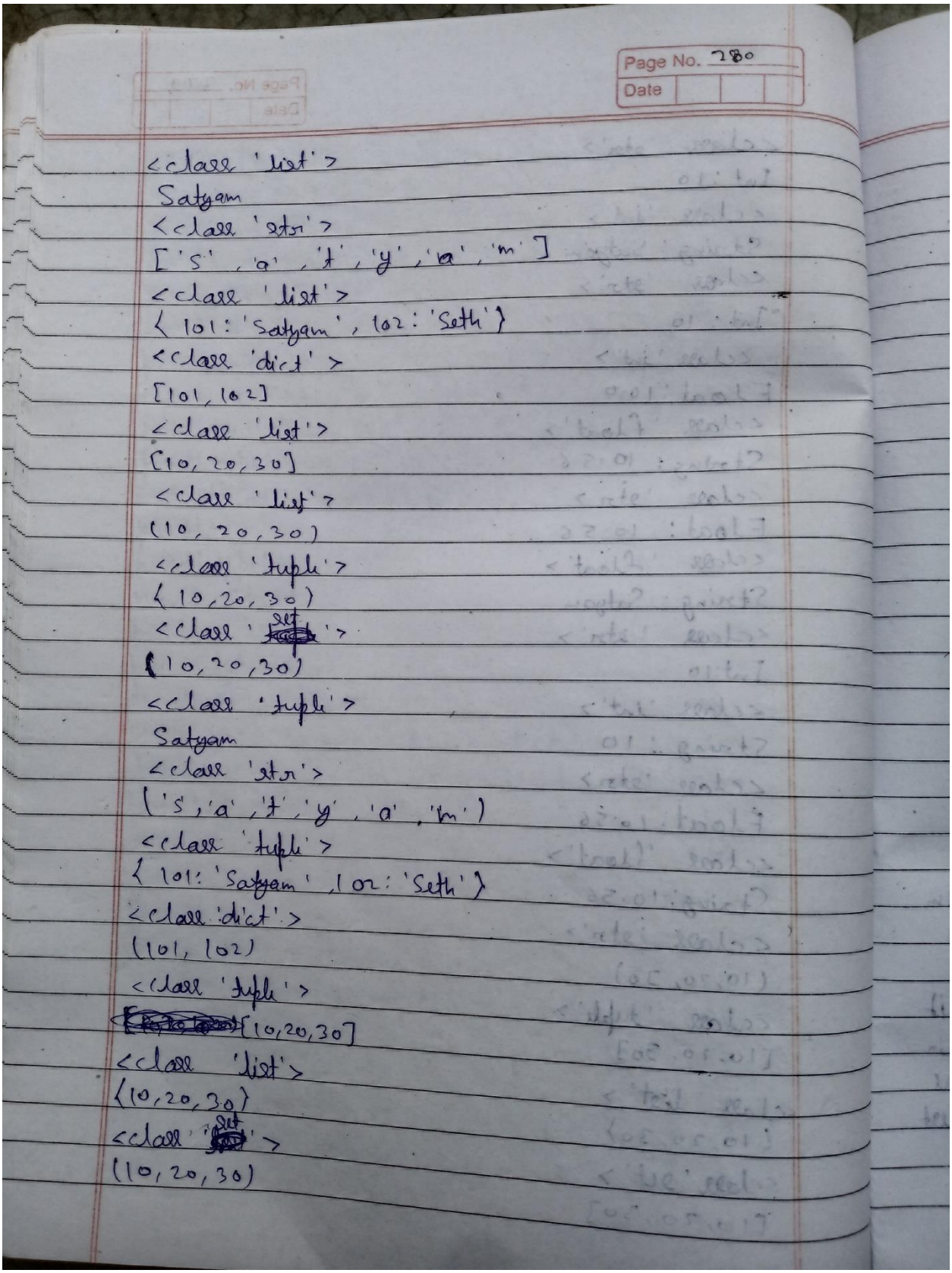
```
print(type(x))
new_x = set(x)
print(new_x)
print(type(new_x))
s = 'Satyam'
print(s)
print(type(s))
new_s = set(s)
print(new_s)
print(type(new_s))
t = {101:'Satyam', 102:'Seth'}
print(t)
print(type(t))
new_t = set(t)
print(new_t)
print(type(new_t)
# dict() Function
u = [(101, 'Satyam'), (102, 'Seth')]
print(u)
print(type(u))
new_u = dict(u)
print(new_u)
print(type(new_u))
```

Output- Float: 10.56
<class 'float'>
Int: 10
<class 'int'>
String: 10
<class 'int'>
String: 10



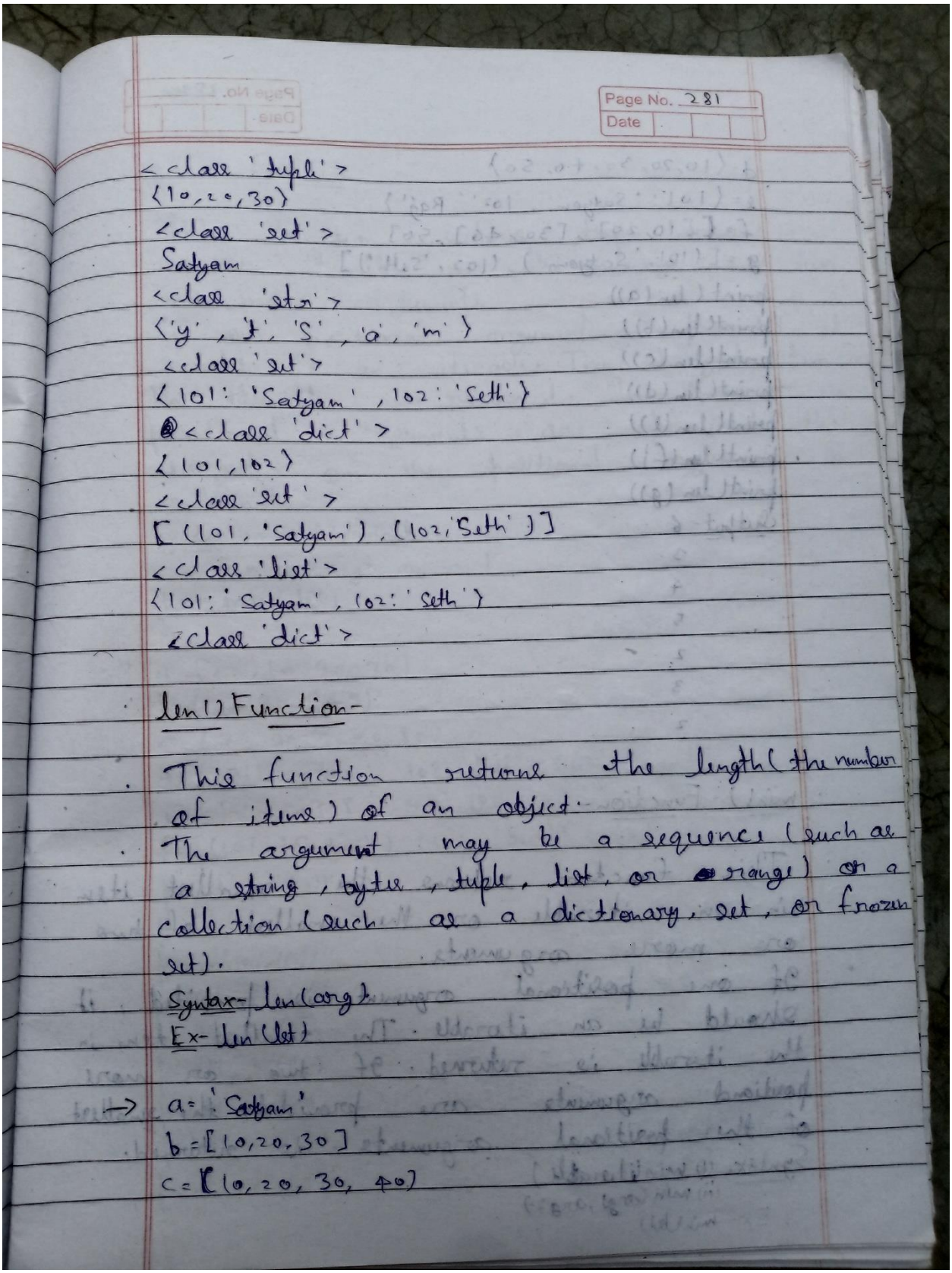
Page No. 279
Date

```
<class 'str'>
Int : 10
<class 'int'>
String : Satyam
<class 'str'>
Int : 10
<class 'int'>
Float : 10.0
<class 'float'>
String : 10.56
<class 'str'>
Float : 10.56
<class 'float'>
String : Satyam
<class 'str'>
Int : 10
<class 'int'>
String : 10
<class 'str'>
Float : 10.56
<class 'float'>
String : 10.56
<class 'str'>
(10, 20, 30)
<class 'tuple'>
[10, 20, 30]
<class 'list'>
{10, 20, 30}
<class 'set'>
[10, 20, 30]
```



Page No. 280
Date

```
<class 'list'>
Satyam
<class 'str'>
['S', 'a', 't', 'y', 'a', 'm']
<class 'list'>
{101: 'Satyam', 102: 'Seth'}
<class 'dict'>
[101, 102]
<class 'list'>
[10, 20, 30]
<class 'list'>
(10, 20, 30)
<class 'tuple'>
{10, 20, 30}
<class 'set'>
{10, 20, 30}
<class 'tuple'>
Satyam
<class 'str'>
('S', 'a', 't', 'y', 'a', 'm')
<class 'tuple'>
{101: 'Satyam', 102: 'Seth'}
<class 'dict'>
(101, 102)
<class 'tuple'>
[10, 20, 30]
<class 'list'>
{10, 20, 30}
<class 'set'>
(10, 20, 30)
```



Page No. 281
Date

```
<class 'tuple'>
(10,20,30)
<class 'set'>
Satyam
<class 'str'>
('y', 't', 'S', 'a', 'm')
<class 'set'>
{101: 'Satyam', 102: 'Seth'}
<class 'dict'>
{101,102}
<class 'set'>
[(101, 'Satyam'), (102, 'Seth')]
<class 'list'>
{101: 'Satyam', 102: 'Seth'}
<class 'dict'>
```

len() Function-

This function returns the length (the number of items) of an object.
The argument may be a sequence (such as a string, bytes, tuple, list, or range) or a collection (such as a dictionary, set, or frozen set).

Syntax- len(arg)
Ex- len(lst)

```
a='Satyam'
b=[10,20,30]
c=[10,20,30,40]
```

Page No. 282
Date

```
d={10,20,30,40,50}
l={101:'Satyam',102:'Raj'}
f=[[10,20],[30,40],50]
g=[(101,'Satyam'),(102,'Seth')]
print(len(a))
print(len(b))
print(len(c))
print(len(d))
print(len(l))
print(len(f))
print(len(g))
```

Output - 6
3
4
5
2
3
2

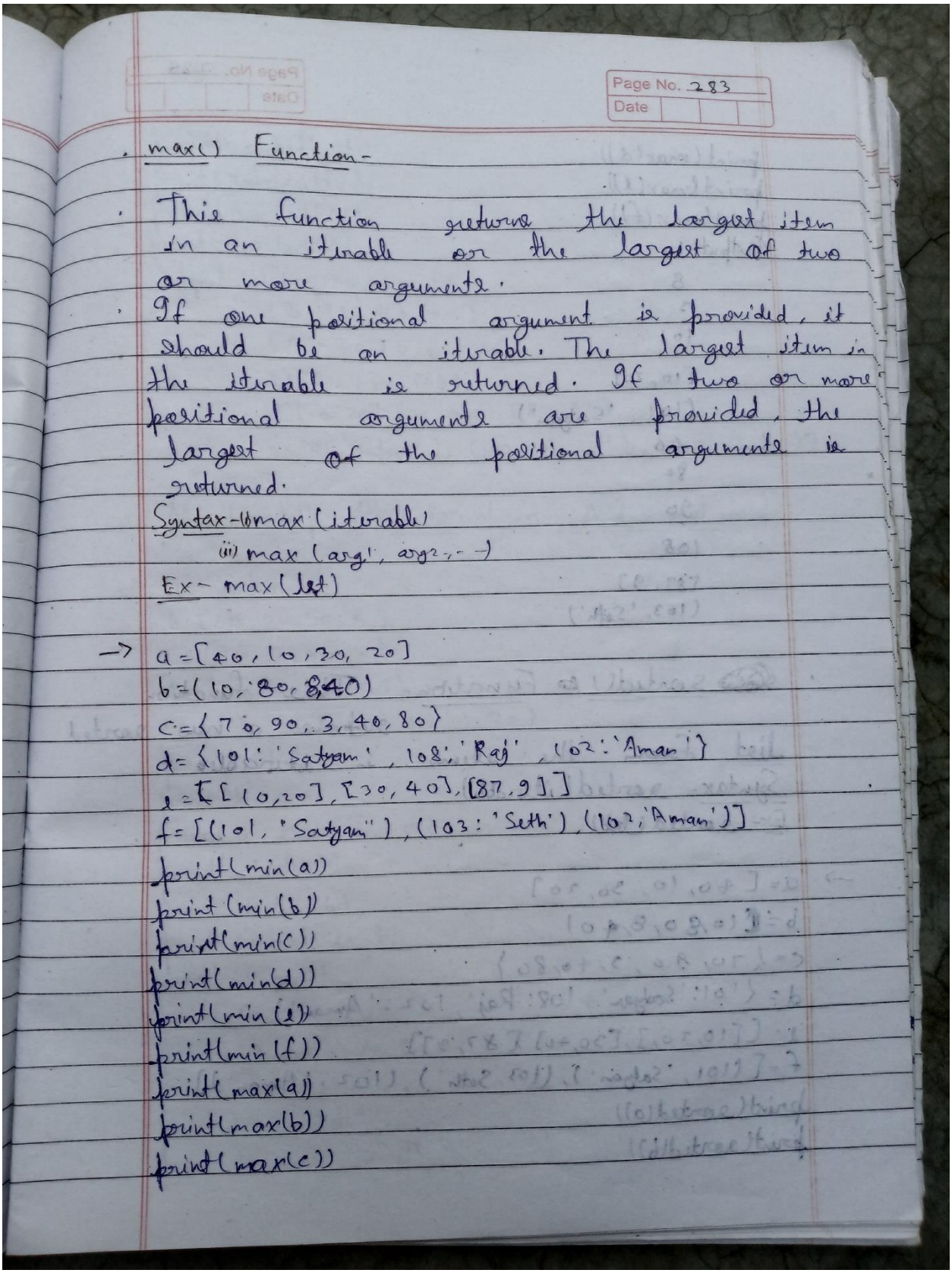
min() Function -

This function returns the smallest item in an iterable or the smallest of two or more arguments.

If one positional argument is provided, it should be an iterable. The smallest item in the iterable is returned. If two or more positional arguments are provided, the smallest of the positional arguments is returned.

Syntax - (i) min(iterable)
(ii) min(arg1, arg2)

Ex - min(a)



Page No. 283
Date

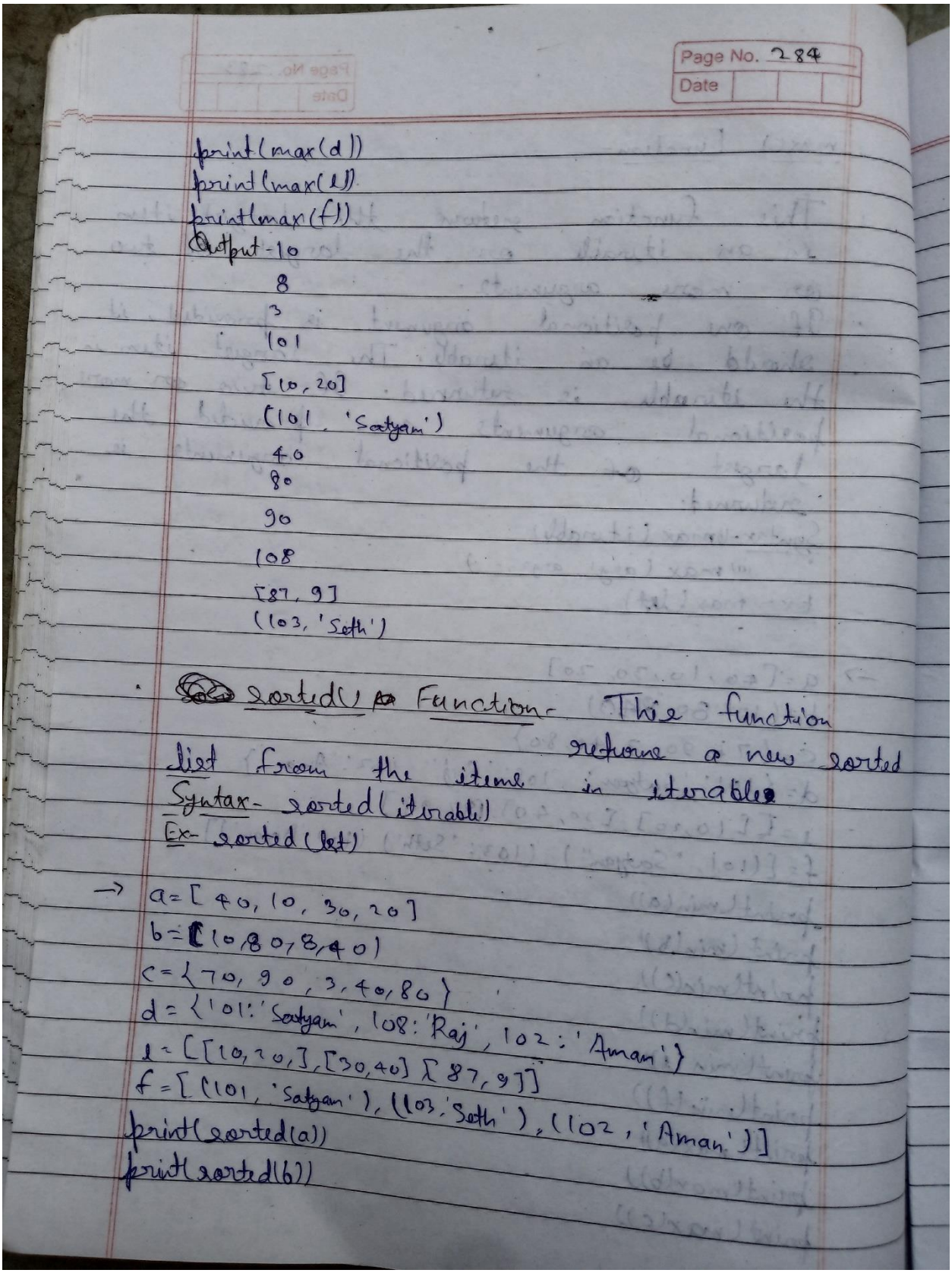
. max() Function -

. This function returns the largest item in an iterable or the largest of two or more arguments.
. If one positional argument is provided, it should be an iterable. The largest item in the iterable is returned. If two or more positional arguments are provided, the largest of the positional arguments is returned.

Syntax - (i) max(iterable)
(ii) max(arg1, arg2, ...)

Ex - max(list)

```
a=[40,10,30,20]
b=(10,80,8,40)
c={70,90,3,40,80}
d={101:'Satyam',108:'Raj',102:'Aman'}
e=[[10,20],[30,40],[87,9]]
f=[(101,'Satyam'),(103:'Seth'),(102,'Aman')]
print(min(a))
print(min(b))
print(min(c))
print(min(d))
print(min(e))
print(min(f))
print(max(a))
print(max(b))
print(max(c))
```



Page No. 284
Date

```
print(max(d))
print(max(l))
print(max(f))
```

Output - 10
8
3
101
[10, 20]
(101, 'Satyam')

```
40
80
90
108
[87, 9]
(103, 'Seth')
```

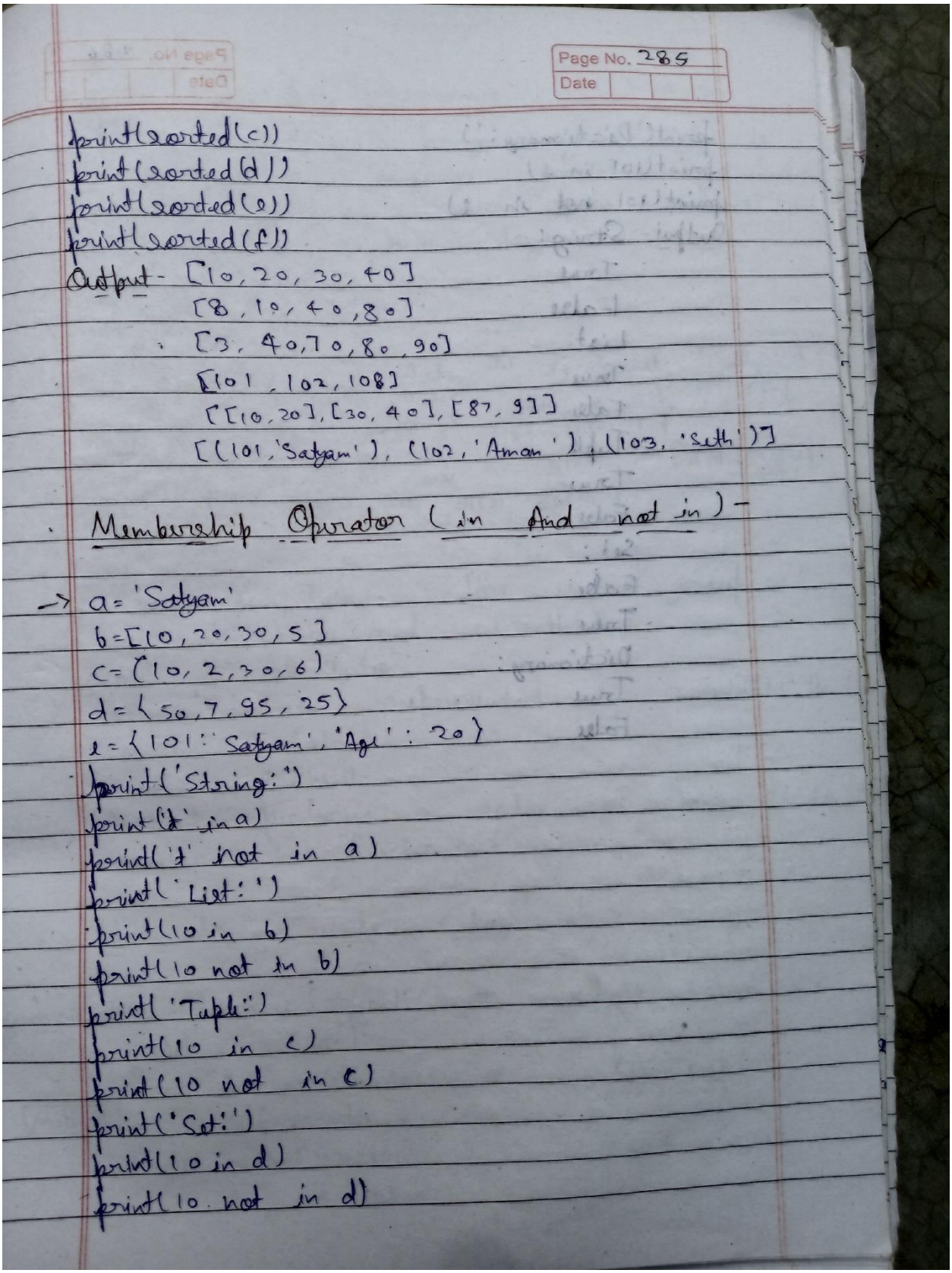
- sorted() Function - This function returns a new sorted list from the items in iterable.

Syntax - sorted(iterable)

Ex - sorted(let)

→

```
a=[40, 10, 30, 20]
b=(10,80,8,40)
c={70, 90, 3, 40, 80}
d={101:'Satyam', 108:'Raj', 102:'Aman'}
l=[[10,20,],[30,40] [87,9]]
f=[(101,'Satyam'),(103,'Seth'),(102,'Aman')]
print(sorted(a))
print(sorted(b))
```



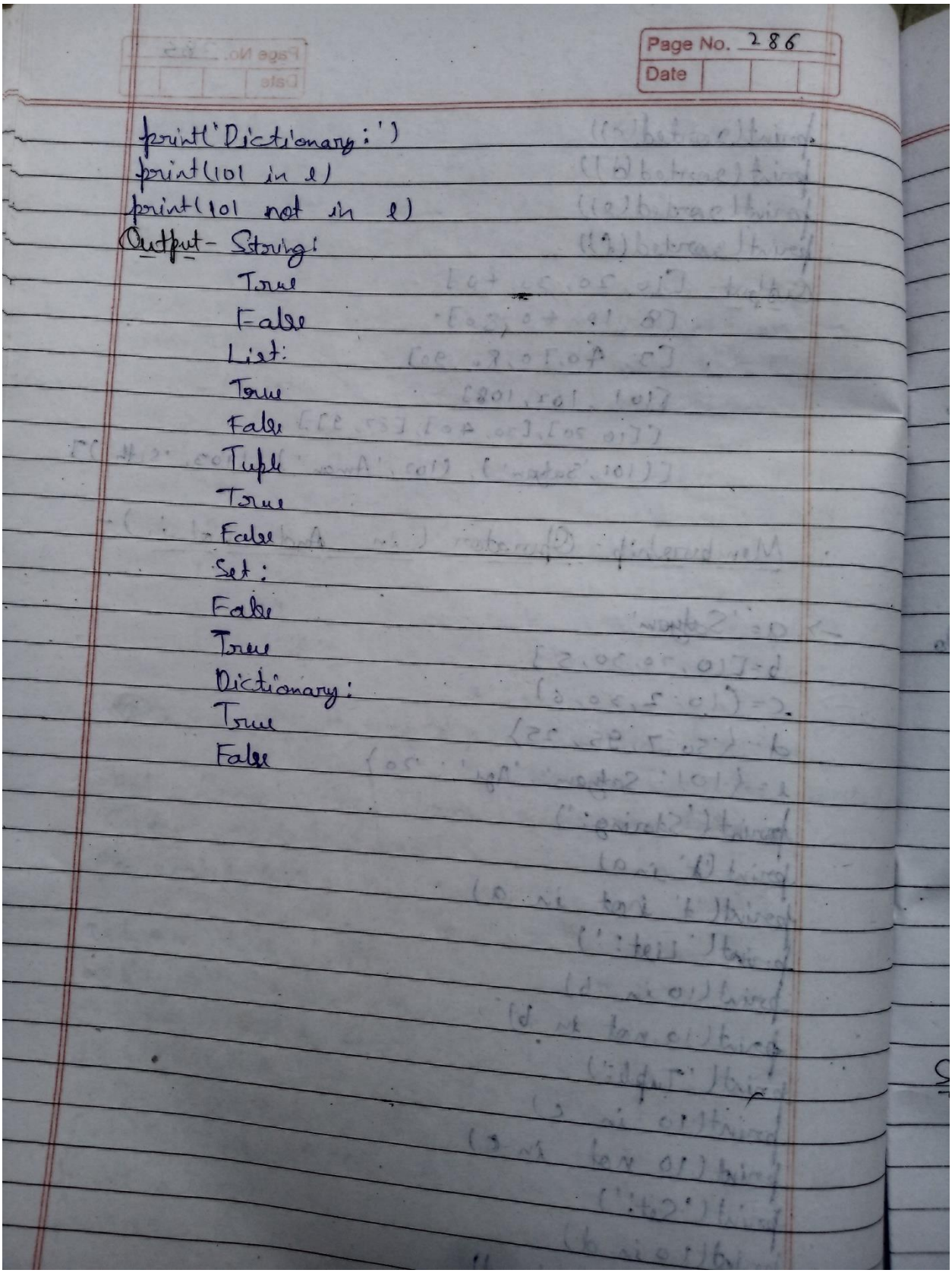
Page No. 285
Date

```
print(sorted(c))
print(sorted(d))
print(sorted(e))
print(sorted(f))
```

Output - [10, 20, 30, 40]
[8, 10, 40, 80]
[3, 40, 70, 80, 90]
[101, 102, 108]
[[10, 20], [30, 40], [87, 9]]
[(101, 'Satyam'), (102, 'Aman'), (103, 'Seth')]

Membership Operator (in And not in) -

```
a = 'Satyam'
b = [10, 20, 30, 5]
c = (10, 2, 30, 6)
d = {50, 7, 95, 25}
e = {101: 'Satyam', 'Age': 20}
print('String:')
print('t' in a)
print('t' not in a)
print('List:')
print(10 in b)
print(10 not in b)
print('Tuple:')
print(10 in c)
print(10 not in c)
print('Set:')
print(10 in d)
print(10 not in d)
```



Page No. 286
Date

```
print('Dictionary:')
print(101 in l)
print(101 not in l)
```

Output- String:
True
False
List:
True
False
Tuple
True
False
Set:
False
True
Dictionary:
True
False