Introduction

1. **About the work book:**
   This workbook is intended to be used by S.Y.B.Sc. (Computer Science) students for Mathematics practical course (Python Programming) in Semester–III. This workbook is designed by considering all the practical concepts / topics mentioned in syllabus.

2. **The objectives of this workbook are:**
   1) Defining the scope of the course.
   2) To have continuous assessment of the course and students.
   3) Providing ready reference for the students during practical implementation.
   4) Provide more options to students so that they can have good practice before facing the examination.
   5) Catering to the demand of slow and fast learners and accordingly providing the practice assignments to them.

3. Instructions to the students

Please read the following instructions carefully and follow them.

1. Students are expected to carry this book every time they come to the lab for practical.
2. Students should prepare oneself beforehand for the Assignment by reading the relevant material.
3. Instructor will specify which problems to solve in the lab during the allotted slot and student should complete them and get verified by the instructor. However student should spend additional hours in Lab and at home to cover as many problems as possible.
4. Students will be assessed for each exercise on a scale from 0 to 5.

Not Done	0
Incomplete	1
Late Complete	2
Needs Improvement	3
Complete	4
Well Done	5

4. Instruction to the Instructors

1) Explain the assignment and related concepts in around ten minutes using white board if required or by demonstrating the software.
2) You should evaluate each assignment carried out by a student on a scale of 5 as specified above by ticking appropriate box.
3) The value should also be entered on assignment completion page of the respective Lab course.

M.E.S. Abasaheb Garware College, Pune-04.

S.Y.B.Sc.(Comp.Sc.) Mathematics Practical

Certificate

This is to certify that Mr./Ms._____ ,
Roll No. _____ of S.Y.B.Sc.(Comp.Sc.) has satisfactorily completed all the assignments of Mathematics practical in Semester III of the academic year 20 .

Date: Batch In-charge

Internal Examiner External Examiner

Assignment Completion Sheet

Assignment No	Description	Marks (out of 2)
1	Basic Python	
2	Python List, Tuple	
3	Python String	
4	Python Set	
5	Conditional statements and loops in Python	
6	Matrices: Standard matrices, Operations on matrices, Determinant, Inverse, system of Equations	
7	Matrices: Eigenvalues and eigenvectors	
8	Roots of equations: Newton Raphson method	
9	Roots of equations: Regula Falsi method	
10	Trapezoidal rule	
11	Simpson's 1/3 rule	
12	Simpson's 3/8 rule	

Basic Python

Python is an interpreted high-level programming language for general-purpose programming.
Python features a dynamic type system and automatic memory management. It supports multiple programming paradigms,including object-oriented, imperative, functional and procedural, and has a large and comprehensive standard library.

Starting the Interpreter

After installation, the python interpreter lives in the installed directory. By default it is /usr/local/bin/pythonX.X in Linux/Unix and C:\PythonXX in Windows, where the 'X' denotes the version number. To invoke it from the shell or the command prompt we need to add this location in the search path.
Search path is a list of directories (locations) where the operating system searches for executables. For example, in Windows command prompt, we can type set path=%path%;c:\python33 (python33 means version 3.3, it might be different in your case) to add the location to path for that particular session.

Python Use

Python is used by hundreds of thousands of programmers and is used in many places. Python has many standard library which is made up of many functions that come with Python when it is installed. On the Internet there are many other libraries available that make it possible for the Python language to do more things. These libraries make it a powerful language; it can do many different things.

Some things that Python is often used for are:

- Web development
- Game programming
- Desktop GUIs
- Scientific programming
- Network programming.

First Python Program

This is a small example of a Python program. It shows "Hello World!" on the screen.
Type the following code in any text editor or an IDE and save as *helloWorld.py*

```
print("Hello world!")
```

Now at the command window, go to the location of this file. You can use the cd command to change directory.
To run the script, type **python helloWorld.py** in the command window. We should be able to see the output as follows:
Hello world!
If you are using PyScripter, there is a green arrow button on top. Press that button or press Ctrl+F9 on your keyboard to run the program.
In this program we have used the built-in function **print()**, to print out a string to the screen. String is the value inside the quotation marks, i.e. Hello world!. Now try printing out your name by modifying this program.

1. Immediate/Interactive mode

Typing python in the command line will invoke the interpreter in immediate mode. We can directly type in Python expressions and press enter to get the output.

>>>

is the Python prompt. It tells us that the interpreter is ready for our input. Try typing in 1 + 1 and press enter. We get 2 as the output. This prompt can be used as a calculator. To exit this mode type exit() or quit() and press enter.

Type the following text at the Python prompt and press the Enter –

>>> print "Hello World"

2. Script Mode Programming

Invoking the interpreter with a script parameter begins execution of the script and continues until the script is finished. When the script is finished, the interpreter is no longer active.

Let us write a simple Python program in a script. Python files have extension **.py**. Type the following source code in a *test.py* file −

```
print("Hello World")
```

We assume that you have Python interpreter set in PATH variable. Now, try to run this program as follows −

```
$ python test.py
```

This produces the following result −

```
Hello, Python!
```

3. Integrated Development Environment (IDE)

We can use any text editing software to write a Python script file.

We just need to save it with the .py extension. But using an IDE can make our life a lot easier. IDE is a piece of software that provides useful features like code hinting, syntax highlighting and checking, file explorers etc. to the programmer for application development.

Using an IDE can get rid of redundant tasks and significantly decrease the time required for application development.

IDEL is a graphical user interface (GUI) that can be installed along with the Python programming language and is available from the official website.

We can also use other commercial or free IDE according to our preference, the PyScripter IDE can be used for testing. It is free and open source.

Python Comments

In Python, there are two ways to annotate your code.

Single-line comment – Comments are created simply by beginning a line with the hash (#) character, and they are automatically terminated by the end of line.
For example:

```
#This would be a comment in Python
```

Multi-line comment- Comments that span multiple lines – used to explain things in more detail – are created by adding a delimiter (“”””) on each end of the comment.

```
""" This would be a multiline comment
in Python that spans several lines and
describes your code, your day, or anything you want it to
...
"""
```

Indentation

The enforcement of indentation in Python makes the code look neat and clean.

For example:

```
if True:
    print('Hello')
    a=5
```

Incorrect indentation will result into IndentationError.

Standard Data Types

Python has five standard data types −

- Numbers
- String
- List
- Tuple
- Dictionary

Python Numbers: Integers, floating point numbers and complex numbers falls under Python numbers category. They are defined as int, float and complex class in Python.

Python supports four different numerical types −

- int (signed integers)
- long (long integers, they can also be represented in octal and hexadecimal)
- float (floating point real values)
- complex (complex numbers)

Python Strings : Strings in Python are identified as a contiguous set of characters represented in the quotation marks. Python allows for either pairs of single or double quotes.

Example: str=’Hello all’

Python Lists :

Lists are the most versatile of Python's compound data types. A list contains items can be of different data types separated by commas and enclosed within square brackets ([]).

```
list_obj=['table', 59 ,2.69,“chair”]
```

Python Tuples:

A tuple is another sequence immutable data type that is similar to the list. A tuple consists of a number of values separated by commas and enclosed in parentheses (( )).

```
Example:
tuple_obj=(786,2.23, “college” )
t1=('roll_no': 15,'name':’xyz’,'per': 69.88)
```

Python Operators :

Python language supports the following types of operators.

- Arithmetic Operators
- Comparison (Relational) Operators
- Assignment Operators
- Logical Operators
- Bitwise Operators
- Membership Operators
- Identity Operators

i. **Arithmetic Operators**:

The new arithmetic operators in python are,

a) **** (Exponent)-** Performs exponential (power) calculation on operators

Example: a**b =10 to the power 20

b) // **(Floor Division) -** The division of operands where the result is the quotient in which the digits after the decimal point are removed. But if one of the operands is negative, the result is floored, i.e., rounded away from zero (towards negative infinity)

Example: 9//2 = 4 and 9.0//2.0 = 4.0, -11//3 = -4, -11.0//3 = -4.0

ii. Logical operators :

Logical operators are and, or, not operators.

a) and - True if both the operands are true (&)
b) or - True if either of the operands is true (|)
c) not - True if operand is false (complements the operand)

iii. Relational / Comparison operators :

== (equal to), != (not equal to), < (less than),<= (Less than or equal to), > (greater than) and >= (Greater than or equal to) are same as other language relational operators. The new relational operator in python is,

<>- If values of two operands are not equal, then condition becomes true.

Example: (a <> b) is true. This is similar to != operator.

iv. Assignment Operators: The following are assignment operators in python which are same as in C, C++.

=, +=, -=, *=, /=, %=, **=, //=

v. Bitwise Operators: The following are bitwise operators in python which are same as in C,C++.

&(bitwise AND), |(bitwise OR) ,^ (bitwise XOR),~ (bitwise NOT),<<(bitwise left shift), >>(bitwise right shift)

vi. Membership operators :

in and **not in** are the membership operators; used to test whether a value or variable is in a sequence.

in - True if value is found in the sequence
not in - True if value is not found in the sequence

vii. Identity operators :

is and **is not** are the identity operators both are used to check if two values are located on the same part of the memory. Two variables that are equal does not imply that they are identical.

is - True if the operands are identical
is not - True if the operands are not identical

Assignment 1

1) Create variables having following values and display using print command.
   i)x=5, ii)y=10.7, iii) z='Python'
2) Check type of each of the variables mentioned in Q1.
3) Assign values to multiple variables and display one at a time.
   Values: 10, 17.9, 'Cat'
4) Use the following operators to perform operations on the variables a and b.
   a=10, b=3. i) +, ii) - , iii) * iv) / , v) ** , vi) % , vii) //
5) Evaluate the following expressions:
   i) 1+2*3 ii) (1+2)*3
   iii) 1+1*3+3*4-4*5 iv) (1+1)*(3+3)*4-4+5
   v) 5+(4-2)*2+6%2-4+3-(5-3)/1.4
6) Use the operator + on the strings a='xy' and b='pq'.
7) Write a Python code to concatenate two strings "Hello" and 'World'.
8) Write a Python code to replicate the string 'Mathematics' 6 times without space and then with space.
9) Display '-' 10 times.
10) Given two variables a and b with values 50 and 20 respectively, verify the comparison operators in Python.
   (!= not equal, == equal, < less than,
   > greater than, <= less than or equal to, >= greater than or equal to)
11) Given two variables a and b with values 55 and 30 respectively, verify the logical operators (& and, | or) for the conditions
   i) a>40, b>40;
   ii) a>0, b>0
12) Compute: 2*3, 2+3, 11/5, absolute value of (3-10), absolute value of (7-4)
13) Find i) ceil(1.001) ii) ceil(2.79) iii) floor(1.001) iv)floor(2.79)
   v) 5! vi) 7! vii) gcd of 10 and 125 viii) gcd of 2 and 17
14) Find the value(s) of the following using math module in Python. $i)\ \sin\left(\frac{\pi}{4}\right)$,
   $ii)\ \cos(\pi)$, $iii)\ \tan\left(\frac{\pi}{6}\right)$, $iv)\ \sinh(\pi)$, v) $\cosh\left(\frac{\pi}{2}\right)$ $vi)\ \exp(5)$,
   $vii)\ \log(100)$ to base 10, 2 and e. viii) square root of 144, ix) 2.8^{th} power of 3.4
   x) 4^{th} power of 121

Assignment Evaluation

0: Not Done [ ]	1: Incomplete [ ]	2: Late Complete [ ]
3: Needs Improvement [ ]	4: Complete [ ]	5: Well Done [ ]

Signature of Instructor

Assignment No. 2 : Python Lists and Tuples

Python Lists :

Lists are the most versatile of Python's compound data types. A list contains items can be of different data types separated by commas and enclosed within square brackets ([]).

```
list_obj=['table', 59 ,2.69,“chair”]
```

Python Tuples:

A tuple is another sequence immutable data type that is similar to the list. A tuple consists of a number of values separated by commas and enclosed in parentheses (( )).

Example:

```
tuple_obj=(786,2.23, “college” )

t1=('roll_no': 15, 'name':’xyz’, 'per': 69.88)
```

A tuple is a sequence of immutable Python objects. Tuples are sequences, just like lists. The differences between tuples and lists are, the tuples cannot be changed unlike lists and tuples use parentheses, whereas lists use square brackets.

Creating a list/tuple is as simple as putting different comma-separated values. Put these comma-separated values between parentheses. For example −

```
tup1 = ('physics', 'chemistry', 1997, 2000);
tup2 = (1, 2, 3, 4, 5 );
tup3 = "a", "b", "c", "d";
```

The empty list/tuple is written as two parentheses containing nothing −

```
Lst1=[]
tup1 = ()
```

Indices of list and tuple indices start at 0, and they can be sliced, concatenated, and so on.

Accessing Values in List / Tuple:

To access values in tuple, use the square brackets for slicing along with the index or indices to obtain value available at that index. For example −

```
#!/usr/bin/python

Lst1 = ['physics', 'chemistry', 1997, 2000]
tup2 = (1, 2, 3, 4, 5, 6, 7 );

print("Lst1[0]: ", Lst1[0])
print("tup2[1:5]: ", tup2[1:5])
```

When the above code is executed, it produces the following result −

```
Lst1[0]:      physics
tup2[1:5]:      [2, 3, 4, 5]
```

Basic Lists/Tuples Operations

Lists/Tuples respond to the + and * operators as concatenation and repetition, the result is a new list/tuple.

Python Expression	**Results**	**Description**
len(1, 2, 3) len([1,2,3])	3	Length
(1, 2, 3) +(4, 5, 6) [1, 2, 3] +[4, 5, 6]	(1, 2, 3, 4, 5, 6) [1, 2, 3, 4, 5, 6]	Concatenation
'Hi!',* 4	'Hi!','Hi!','Hi!','Hi!'	Repetition
3 in (1, 2, 3) 3 in [1,2,3]	True	Membership
for x in (1, 2, 3): print x	1 2 3	Iteration
min(1,2,3) min([1,2,3])	1	Minimum
max(1,2,3) max([1,2,3])	3	Maximum

Updating Lists

To insert an element in a list at the end of the list, append is used.
Lst_name.append(element)

To insert an element k in a list at i^{th} index, insert is used.
Lst_name.insert(i, k)

```
l=[1,2,3,4]
l.append(5)
l
[1, 2, 3, 4, 5]
l.insert(2,7)
l
[1, 2, 7, 3, 4, 5]
```

To remove an element k at i^{th} index, remove is used.
Lst_name.remove(i)

```
l
```

```
[1, 2, 7, 3, 4, 5]
```

```
l.remove(3)
```

```
l
```

```
[1, 2, 7, 4, 5]
```

Updating Tuples

Tuples are immutable which means you cannot update or change the values of tuple elements. You are able to take portions of existing tuples to create new tuples as the following example demonstrates –

```
tup1 = (12, 34.56);
tup2 = ('abc', 'xyz');

#  Following action is not valid for tuples
#  tup1[0] = 100;

#  So let's create a new tuple as follows
tup3 = tup1 + tup2;
print tup3
```

When the above code is executed, it produces the following result −

```
(12, 34.56, 'abc', 'xyz')
```

Delete Tuple Elements

Removing individual tuple elements is not possible. There is, of course, nothing wrong with putting together another tuple with the undesired elements discarded.

To explicitly remove an entire tuple, just use the **del** statement. For example:

```
tup = ('physics', 'chemistry', 1997, 2000);

print tup
del tup;
print "After deleting tup : "
print tup
```

This produces the following result. Note an exception raised, this is because after **del tup** tuple does not exist any more −

```
('physics', 'chemistry', 1997, 2000)
After deleting tup :
Traceback (most recent call last):
  File "test.py", line 9, in <module>
    print tup;
NameError: name 'tup' is not defined
```

Built-in Tuple Functions
Python includes the following tuple functions :

Function	Description
all()	Return True if all elements of the tuple are true (or if the tuple is empty).
any()	Return True if any element of the tuple is true. If the tuple is empty, return False.
enumerate()	Return an enumerate object. It contains the index and value of all the items of tuple as pairs.
len()	Return the length (the number of items) in the tuple.
max()	Return the largest item in the tuple.
min()	Return the smallest item in the tuple
sorted()	Take elements in the tuple and return a new sorted list (does not sort the tuple itself).
sum()	Return the sum of all elements in the tuple.
tuple()	Convert an iterable (list, string, set, dictionary) to a tuple.

Assignment 2- Lists/ Tuples

Lists:

1. Declare a list having following elements.

   i) 1,2,3,4,5 ii) a, b, c, d, e iii) Hello, s, 2023, 1.374

2. Print all the elements of from the lists given in question 1.

3. Declare a list having elements integers from 1 to 8.

i) Extract elements 1 to 6 from the list in a sequential order.

ii) Extract the elements 2,3,4,5 from a list.

4. Declare a list having elements Maths, 100, 12.45, 2000. Print value at index 2. Update the value at index 2 by 245. Print the updated list.
5. Delete the second and third value from the list mentioned in the question 4.
6. Find lengths of the lists in questions 3 and 4.
7. Concatenate the lists given in questions 3 and 4.
8. Declare a list having elements Thursday, Friday, 2022, Python, b.

   Replicate the element 2022 four times. Replicate the element Python three times.
9. Check whether 2021 belongs to the list given in question 8.
10. Check whether Thursday belongs to the list given in question 8.
11. Print the elements of the list [1,2,3,4] using for loop.
12. Declare two lists with elements 1,2,3,4,5 and 1,6,4,8,9. i) Find the elements from the lists with minimum value. ii) Find the elements from the lists with maximum value. iii) Append the element 11 to both the lists. iii) Insert the element 225 in both the lists at index 2.
13. Remove 4 from the first and 9 from the second list.
14. Declare a list with elements h, H, 2022,H, 2023, a, b, H. Print the count of number of occurrences of H in this list. Also find the number of occurrences of 2022.

Tuples:

1. Declare a tuple having elements i) hello, a, 20, 30 ii) 1, 2, 3, 4, 5, 6.

2. Declare an empty tuple.

3. Declare a tuple having only one element 20.

4. Access elements (one by one) of the tuples given in question 1.

5. Extract the elements 2,3,4,5 from 2nd tuple given in question 1.

6. Concatenate the tuples given in question 1.

Assignment Evaluation

0: Not Done [ ]	1: Incomplete [ ]	2: Late Complete [ ]
3: Needs Improvement [ ]	4: Complete [ ]	5: Well Done [ ]

Signature of Instructor

Assignment No. 3 : Python strings

A string is a sequence of characters. A character is simply a symbol. For example, the English language has 26 characters. Computers do not deal with characters, they deal with numbers (binary). Even though you may see characters on your screen, internally it is stored and manipulated as a combination of 0's and 1's.

This conversion of character to a number is called encoding, and the reverse process is decoding. ASCII and Unicode are some of the popular encoding used.

In Python, string is a sequence of Unicode character. Unicode was introduced to include every character in all languages and bring uniformity in encoding.

We can create them simply by enclosing characters in quotes. Python treats single quotes the same as double quotes. Creating strings is as simple as assigning a value to a variable. For example –

```
var1 = 'Hello World!'
var2 = "Python Programming"
```

Accessing Values in Strings

Python does not support a character type; these are treated as strings of length one, thus also considered a substring.

To access substrings, use the square brackets for slicing along with the index or indices to obtain your substring. For example −

```
#!/usr/bin/python

var1 = 'Hello World!'
var2 = "Python Programming"

print "var1[0]: ", var1[0]
print "var2[1:5]: ", var2[1:5]
```

Updating Strings

You can "update" an existing string by *re*assigning a variable to another string. The new value can be related to its previous value or to a completely different string altogether. For example −

```
#!/usr/bin/python

var1 = 'Hello World!'

print "Updated String :- ", var1[:6] + 'Python'
```

When the above code is executed, it produces the following result −

```
Updated String :-  Hello Python
```

String Special Operators

Assume string variable **a** holds 'Hello' and variable **b** holds 'Python', then −

Operator	Description	Example
+	Concatenation - Adds values on either side of the operator	a + b will give HelloPython
*	Repetition - Creates new strings, concatenating multiple copies of the same string	a*2 will give -HelloHello
[]	Slice - Gives the character from the given Index	a[1] will give e
[:]	Range Slice - Gives the characters from the given range	a[1:4] will give ell
in	Membership - Returns true if a character exists in the given string	H in a will give 1
not in	Membership - Returns true if a character does not exist in the given string	M not in a will give 1
%	Format - Performs String formatting	See at next section

String Formatting Operator

One of Python's coolest features is the string format operator %. This operator is unique to strings and makes up for the pack of having functions from C's printf family. Following is a simple example.

```
#!/usr/bin/python
print "My name is %s and weight is %d kg!" % ('Surya ', 65)

When the above code is executed, it produces the following result −
My name is Surya and weight is 65 kg!
```

Here is the list of complete set of symbols which can be used along with % −

Format Symbol	Conversion
%c	Character
%s	string conversion via str prior to formatting
%i	signed decimal integer
%d	signed decimal integer
%u	unsigned decimal integer
%o	octal integer
%x	hexadecimal integer *lowercaseletters*
%X	hexadecimal integer *UPPERcaseletters*

%e	exponential notation *withlowercase′e′*
%E	exponential notation *withUPPERcase′E′*
%f	floating point real number
%g	the shorter of %f and %e
%G	the shorter of %f and %E

Other supported symbols and functionality are listed in the following table –

Symbol	**Functionality**
*	argument specifies width or precision
-	left justification
+	display the sign
<sp>	leave a blank space before a positive number
#	add the octal leading zero ′0′ or hexadecimal leading '0x' or '0X',
	depending on whether 'x' or 'X' were used.
0	pad from left with zeros *insteadofspaces*
%	'%%' leaves you with a single literal '%'
Var	mapping variable *dictionary arguments*
m.n.	m is the minimum total width and n is the number of digits to display after the decimal point *ifappl*.

Triple Quotes

Python's triple quotes comes to the rescue by allowing strings to span multiple lines, including verbatim NEWLINEs, TABs, and any other special characters.

The syntax for triple quotes consists of three consecutive **single or double** quotes.

```
#!/usr/bin/python
para_str = """this is a long string that is made up of several lines and non-printable characters such as TAB ( \t ) and they will show up that way when displayed. NEWLINEs within the string, whether explicitly given like this within the brackets [ \n ], or just a NEWLINE within the variable assignment will also show up. """
print para_str
```

When the above code is executed, it produces the following result. Note how every single special character has been converted to its printed form, right down to the last NEWLINE at the end of the string between the "up." and closing triple quotes. Also note that NEWLINEs occur either with an explicit carriage return at the end of a line or its escape code \n −

```
this is a long string that is made up of
several lines and non-printable characters such as
TAB ( ) and they will show up that way when displayed. NEWLINEs within the string, whether explicitly given like this within the brackets [
], or just a NEWLINE within
the variable assignment will also show up.
```

Raw strings do not treat the backslash as a special character at all. Every character you put into a raw string stays the way you wrote it −

```
#!/usr/bin/python
print 'C:\\nowhere'
```

When the above code is executed, it produces the following result −

```
C:\nowhere
```

Built-in String Methods (Most of these methods will not be needed in Mathematics practical)

Python includes the following built-in methods to manipulate strings –

Sr. No.	Methods	Description
1	**Capitalize()**	Capitalizes first letter of string
2	**count(str, beg= 0,end=len(string))**	Counts how many times str occurs in string or in a substring of string if starting index beg and ending index end are given.
3	**endswith(suffix, beg=0, end=len(string))**	Determines if string or a substring of string (if starting index beg and ending index end are given) ends with suffix; returns true if so and false otherwise.
4	**isalnum()**	Returns true if string has at least 1 character and all characters are alphanumeric and false otherwise.
5	**isalpha()**	Returns true if string has at least 1 character and all characters are alphabetic

		and false otherwise.
6	**isdigit()**	Returns true if string contains only digits and false otherwise.
7	**islower()**	Returns true if string has at least 1 cased character and all cased characters are in lowercase and false otherwise.
8	**isnumeric()**	Returns true if a unicode string contains only numeric characters and false otherwise.
9	**isspace()**	Returns true if string contains only whitespace characters and false otherwise.
10	**istitle()**	Returns true if string is properly "titlecased" and false otherwise.
11	**isupper()**	Returns true if string has at least one cased character and all cased characters are in uppercase and false otherwise.
12	**join(seq)**	Merges (concatenates) the string representations of elements in sequence seq into a string, with separator string.
13	**len(string)**	Returns the length of the string
14	**ljust(width[, fillchar])**	Returns a space-padded string with the original string left-justified to a total of width columns.
15	**lower()**	Converts all uppercase letters in string to lowercase.
16	**lstrip()**	Removes all leading whitespace in string.
17	**maketrans()**	Returns a translation table to be used in translate function.
18	**max(str)**	Returns the max alphabetical character from the string str.
19	**min(str)**	Returns the min alphabetical character from the string str.
20	**replace(old, new [, max])**	Replaces all occurrences of old in string with new or at most max occurrences if max given.
21	**rfind(str, beg=0,end=len(string))**	Same as find(), but search backwards in string.
22	**rjust(width,[, fillchar])**	Returns a space-padded string with the

		original string right-justified to a total of width columns.
23	**rstrip()**	Removes all trailing whitespace of string.
24	**split(str="", num=string.count(str))**	Splits string according to delimiter str (space if not provided) and returns list of substrings; split into at most num substrings if given.
25	**splitlines(num=string.count('\n'))**	Splits string at all (or num) NEWLINEs and returns a list of each line with NEWLINEs removed.
26	**swapcase()**	Inverts case for all letters in string.
27	**title()**	Returns "titlecased" version of string, that is, all words begin with uppercase and the rest are lowercase.
28	**translate(table, deletechars="")**	Translates string according to translation table str(256 chars), removing those in the del string.
29	**upper()**	Converts lowercase letters in string to uppercase.
30	**zfill (width)**	Returns original string leftpadded with zeros to a total of width characters; intended for numbers, zfill() retains any sign given (less one zero).
31	**isdecimal()**	Returns true if a unicode string contains only decimal characters and false otherwise.

Assignment 3 - Strings

1. Declare the following strings. i) Mathematics ii) hello world
2. Find length of each string mentioned above in question1 and print characters with index value.
3. Access specific elements from above strings using slicing syntax. Print Ma, orl.
4. Access the contents of the string given in question 1 using for loop.
5. Access the contents of the string given in question 1 using while loop.
6. Declare two strings fruit and Friday. Compare these two strings using comparison operators ==, !=, <, >, <=, >=.

7. Concatenate the two strings given in question 6.
8. Replicate the first and second strings 4 times and 7 times respectively.
9. Declare a string PYTHON. Print this string using slicing syntax. Print PYTH using slicing syntax.
10. Declare a string Mathematics. Write a python code to covert this string to uppercase (lowercase).

Assignment Evaluation

0: Not Done [ ]	1: Incomplete [ ]	2: Late Complete [ ]
3: Needs Improvement [ ]	4: Complete [ ]	5: Well Done [ ]

Signature of Instructor

Assignment No. 4 : Python Sets

A set is an unordered collection of items. Every element is unique (no duplicates) and must be immutable (which cannot be changed).

However, the set itself is mutable. We can add or remove items from it.

Sets can be used to perform mathematical set operations like union, intersection, symmetric difference etc.

How to create a set?

A set is created by placing all the items (elements) inside curly braces {}, separated by comma or by using the built-in function set().

It can have any number of items and they may be of different types (integer, float, tuple, string etc.). But a set cannot have a mutable element, like list, set or dictionary, as its element.

Example

```
# set of integers
my_set = {1, 2, 3}
print(my_set)
# set of mixed datatypes
my_set = {1.0, "Hello", (1, 2, 3)}
print(my_set)
```

Creating an empty set is a bit tricky.

Empty curly braces {} will make an empty dictionary in Python. To make a set without any elements we use the set() function without any argument.

```
# initialize a with {}
a = {}
# check data type of a
# Output: <class 'dict'>
print(type(a))
# initialize a with set()
a = set()
# check data type of a
```

```
# Output: <class 'set'>

print(type(a))
```

How to change a set in Python?

Sets are mutable. But since they are unordered, indexing have no meaning.
We cannot access or change an element of set using indexing or slicing. Set does not support it.
We can add single element using the add() method and multiple elements using the update() method. The update() method can take tuples, lists, strings or other sets as its argument. In all cases, duplicates are avoided.

```
# initialize my_set
my_set = {1,3}
print(my_set)

# if you uncomment line 9,
# you will get an error
# TypeError: 'set' object does not support indexing

#my_set[0]

# add an element
# Output: {1, 2, 3}
my_set.add(2)
print(my_set)

# add multiple elements
# Output: {1, 2, 3, 4}
my_set.update([2,3,4])
print(my_set)

# add list and set
# Output: {1, 2, 3, 4, 5, 6, 8}
my_set.update([4,5], {1,6,8})
print(my_set)
```

How to remove elements from a set?

A particular item can be removed from set using methods, discard() and remove().

The only difference between the two is that, while using discard() if the item does not exist in the set, it remains unchanged. But remove() will raise an error in such condition.

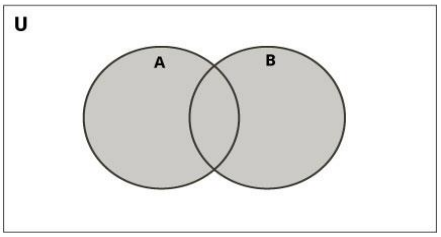
```
my_set = {1, 3, 4, 5, 6}
print(my_set)
# discard an element
# Output: {1, 3, 5, 6}
my_set.discard(4)
print(my_set)
# remove an element
# Output: {1, 3, 5}
my_set.remove(6)
print(my_set)
# discard an element
# not present in my_set
# Output: {1, 3, 5}
my_set.discard(2)
print(my_set)
```

Python Set Operations

Sets can be used to carry out mathematical set operations like union, intersection, difference and symmetric difference. We can do this with operators or methods. Let us consider the following two sets for the following operations.

```
>>> A ={1,2,3,4,5}
>>> B ={4,5,6,7,8}
```

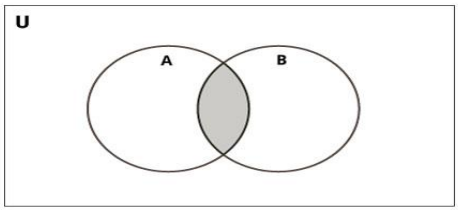
Set Union



Union of *A* and *B* is a set of all elements from both sets.

Union is performed using | operator. Same can be accomplished using the method union()

Set Intersection



Intersection of *A* and *B* is a set of elements that are common in both sets.

Intersection is performed using & operator. Same can be accomplished using the method intersection().

A = {1, 2, 3, 4, 5}

B = {4, 5, 6, 7, 8}

#union

print(A | B) OR A.union(B)

InterSection

print(A & B) OR A.intersection(B)

Method	Description
add()	Add an element to a set
clear()	Remove all elements form a set
copy()	Return a shallow copy of a set
difference()	Return the difference of two or more sets as a new set
difference_update()	Remove all elements of another set from this set
discard()	Remove an element from set if it is a member. (Do nothing if the element is not in set)

intersection()	Return the intersection of two sets as a new set
intersection_update()	Update the set with the intersection of itself and another
isdisjoint()	Return True if two sets have a null intersection
issubset()	Return True if another set contains this set
issuperset()	Return True if this set contains another set
pop()	Remove and return an arbitary set element. Raise KeyErrorif the set is empty
remove()	Remove an element from a set. If the element is not a member, raise a KeyError
symmetric_difference()	Return the symmetric difference of two sets as a new set
symmetric_difference_update()	Update a set with the symmetric difference of itself and another
union()	Return the union of sets in a new set
update()	Update a set with the union of itself and others

Built-in Functions with Set

Built-in functions like all(), any(), enumerate(), len(), max(), min(), sorted(), sum() etc. are commonly used with set to perform different tasks.

Function	Description
all()	Return True if all elements of the set are true (or if the set is empty).
any()	Return True if any element of the set is true. If the set is empty, return False.
enumerate()	Return an enumerate object. It contains the index and value of all the items of

	set as a pair.
len()	Return the length (the number of items) in the set.
max()	Return the largest item in the set.
min()	Return the smallest item in the set.
sorted()	Return a new sorted list from elements in the set(does not sort the set itself).
sum()	Return the sum of all elements in the set.

Assignment 4

1. Declare a list of elements. 1, 2, 3, 4, 5, 2. Create a set using this list.
2. Declare a set having elements. 1, 2, 3, 2, 4, 5, 2 and find its cardinality.
3. Declare empty set.
4. Declare two sets P and Q having elements 1, 2, 3, 5, 8 and 2, 3, 5, 7, 11, 13 respectively.
   i) Add 13 to set P.
   ii) Remove 13 from set Q.
   iii) Find union of P&Q.
   iv) Intersection of P&Q.
   v) Find difference of the two sets.
   vi) Find symmetric difference of A&B.
5. Declare two sets A and B having elements 1, 2, 3, 4 and 1,2 respectively.
   i) Check whether B is a subset of A.
   ii) Check whether A is a subset of B.
   iii) Check whether A is a super set of B.
   iv) Check whether B is a super set of A.
6. Check whether 1 belongs to A. Also check whether 12 is an element of B.

Assignment Evaluation

0: Not Done [ ] 1: Incomplete [ ] 2: Late Complete [ ]

3: Needs Improvement [ ] 4: Complete [ ] 5: Well Done [ ]

Signature of Instructor

Assignment No. 5 – Conditional statements and Loops

Decision making Statement

Python programming language provides following types of decision making statements.

i. **If statement**: It is similar to that of other languages

Syntax

```
if expression:
    statement(s)
```

ii. **IF...ELIF...ELSE Statements:**

Syntax

```
if expression:
    statement(s)
else:
    statement(s)
```

iii. **nested IF statements:**

In a nested **if** construct, you can have an **if...elif...else** construct inside another **if...elif...else** construct.

Syntax

```
if expression1:
    statement(s)
if expression2:
    statement(s)
elif expression3:
    statement(s)
else:
    statement(s)
elif expression4:
    statement(s)
else:
    statement(s)
```

Python – Loops

i. while loop:

A **while** loop statement in Python programming language repeatedly executes a target statement as long as a given condition is true.

Syntax-

```
while expression:
    statement(s)
```

Example:

```
count=0
while(count <3):
    print('The count is:', count)
    count= count +1
```

ii. for loop:

It has the ability to iterate over the items of any sequence, such as a list or a string.

Syntax

```
for iterating_var  in sequence:
                statements(s)
```

Example:

```
for x in List1:

    print x
```

Assignment 5

1. Using input function accept name of the user and print it.
2. Using input function accept and surname separately and print the full name.
3. Accept two integers a and b from the user and print addition of a and b.
   (Hint: a=int(input("input an integer a="))
4. Accept an integer and print its cube.
5. Accept an expression 2*3 – 7 +20 from the user and evaluate it. (x=eval(input('Enter an expression: ') print(x))
6. Print 'Python' 10 times using while loop.
7. Print the integers 1 to 15 in ascending order using while loop.
8. Print the integers 1 to 15 in descending order using while loop.
9. Accept a positive integer x from the user and print factorial of x. (Use while loop)
10. Accept a positive integer x from the user and print table of x. (Use while loop)
11. Declare a list containing all colors of a rainbow. Print all colors using for loop.

12. Repeat question 11 for x=[2, 3.7, ‘India’].
13. Print the numbers from 1 to 20 in ascending order using for loop. (Use range function)

```
for i in range(1,20,1):
    print(i)
```

14. Print the numbers from 31 to 40 in ascending order using for loop. (Use range function)
15. Print the integers from 100 to 150 which are divisible by 5.
16. Print the integers less than 23 which are not divisible by 3.
17. Print the integers from 100 to 150 which are divisible by 3 and 7.

Assignment Evaluation

0: Not Done [ ]	1: Incomplete [ ]	2: Late Complete [ ]
3: Needs Improvement [ ]	4: Complete [ ]	5: Well Done [ ]

Signature of Instructor

Assignment 6 – Operations on Matrices, System of linear equations

Python does not have a built in data structure for matrix/ multidimensional arrays. We use NumPy Array for it. NumPy is a package for scientific computing which has support for a powerful N-dimensional array object.

NumPy provides multidimensional array of numbers (which is actually an object). Let's take an example:

import numpy as np

a = np.array([1, 2, 3])

print(a) # Output: [1, 2, 3]

print(type(a)) # Output: <class 'numpy.ndarray'>

There are several ways to create a matrix using NumPy.

Creating numPy matrix

np.matrix will create a matrix. Dimension of a matrix can be found using shape command.

```
: import numpy as np

: np.matrix([5,6,8])

: matrix([[5, 6, 8]])

: a=np.matrix([5,6,7])
  print(a)

 [[5 6 7]]

: np.matrix([[1],[2],[3],[4]])

: matrix([[1],
          [2],
          [3],
          [4]])
```

Attribute shape is used to find dimension (number of rows and columns) of a matrix.

```
: #To find dimension of a matrix
  b=np.matrix([[1],[2],[3],[4]])
  b.shape

: (4, 1)
```

To create special matrices like identity matrix, diagonal matrix, matrix with all entries equal to 1, numpy has built in functions eye, diag and ones respectively.

```
[8]: #Write a matrix of size 4x6 with all entries equal to 1
     np.ones(24).reshape(4,6)
```

```
[8]: array([[1., 1., 1., 1., 1., 1.],
           [1., 1., 1., 1., 1., 1.],
           [1., 1., 1., 1., 1., 1.],
           [1., 1., 1., 1., 1., 1.]])
```

```
[9]: #Write a matrix of size 4x6 with all entries equal to 2.5 i.e. constant matrix
     ↪of size 4x6 with entries 2.5
     c=np.ones(24).reshape(4,6)
     c=2.5*c
     print(c)
```

```
[[2.5 2.5 2.5 2.5 2.5 2.5]
 [2.5 2.5 2.5 2.5 2.5 2.5]
 [2.5 2.5 2.5 2.5 2.5 2.5]
 [2.5 2.5 2.5 2.5 2.5 2.5]]
```

```
[10]: #Generate a diagonal matrix with diagonal entries 1,2,6
      np.diag([1,2,6])
```

```
[10]: array([[1, 0, 0],
            [0, 2, 0],
            [0, 0, 6]])
```

For addition, subtraction and multiplication of two matrices, the operators +, - and * are used. Operator ** can be used for finding power of a matrix.

```
A+B
```

```
matrix([[ 6,  8],
        [10, 12]])
```

```
A-B
```

```
matrix([[-4, -4],
        [-4, -4]])
```

```
A*B
```

```
matrix([[19, 22],
        [43, 50]])
```

To insert a row or a column in a matrix insert is used.

np.insert(target_matrix, index, matrix-tobe-inserted, flag)

flag=0 if row is to be inserted in target_matrix

flag=1 if column is to be inserted in target matrix

```
import numpy as np
A=np.matrix([[1,2,0],[3,5,4],[0,1,0]])
B=np.matrix([[5,67,8]])
```

```
C=np.insert(A,0,B,0)
print(C)
```

```
[[ 5 67  8]
 [ 1  2  0]
 [ 3  5  4]
 [ 0  1  0]]
```

```
D=np.insert(A,2,B,1)
print(D)
```

```
[[ 1  2  5  0]
 [ 3  5 67  4]
 [ 0  1  8  0]]
```

Numpy has a strong **linalg** submodule which can be used for finding determinant, inverse, rank, eigenvalues and eigenvectors of a matrix.

To use linalg module first import it.

import numpy.linalg as la

Function	**Description**
la.det(A)	Finds determinant of A
la.inv(A)	Finds inverse of A, if it exists
la.transpose(A) (OR A.T)	Finds transpose of A
la.matrix_rank	Finds rank of A

```
import numpy.linalg as la
```

```
#Find inverse of B
la.inv(B)
```

```
matrix([[-4. ,  3. ],
        [ 3.5, -2.5]])
```

```
la.det(A)
```

```
-2.0000000000000004
```

```
#Find transpose of A
np.transpose(A)
```

```
matrix([[1, 3],
        [2, 4]])
```

```
#Find transpose of A
A.T
```

```
matrix([[1, 3],
        [2, 4]])
```

Solving a system of linear equations

To solve a system of linear equation Ax=B, solve command in linalg is used.

import numpy.linalg as la

la.solve(A,B) # Output is solution of Ax=B

```
# x+y+z=3, x-y+z=1, x-y-z=-1#
A=np.matrix([[1,1,1],[1,-1,1],[1,-1,-1]])
B=np.matrix([[3],[1],[-1]])
la.solve(A,B)
```

```
matrix([[1.],
        [1.],
        [1.]])
```

```
# x+y=3, x-y=1 #
A=np.matrix([[1,1],[1,-1]])
B=np.matrix([[3],[1]])
la.solve(A,B)
```

```
matrix([[2.],
        [1.]])
```

Matrix inversion method

To solve a system of linear equation Ax=B, where A is a square matrix and determinant of A is nonzero, matrix inversion method can also be used.

```
import numpy as np
A=np.matrix([[1,2,-1],[5,6,-1],[0,1,1]])
```

```
B=np.matrix([[1],[-2],[4]])
```

```
la.det(A)
```

```
-7.999999999999998
```

```
x=la.inv(A)*B
```

```
x
```

```
matrix([[-3.625],
        [ 2.875],
        [ 1.125]])
```

Assignment 6 – Matrices, System of Equations

1. Create following matrices:
   i) Row matrix of order 1x3.
   ii) Column matrix of order 4x1.
   iii) Identity matrix of order 5.
   iv) Zero matrix of size 4x3.
   v) Matrix of ones 5x6
   vi) diagonal matrix
2. Consider the matrices A and B. Perform following operation on them. $A = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}, B = \begin{bmatrix} 5 & 6 \\ 7 & 8 \end{bmatrix}$.
3. Delete second row from A.
4. Delete the first column from B.
5. Consider the matrix $C = \begin{bmatrix} 1 & 0 & 0 \\ 3 & 4 & 6 \\ 1 & 3 & 2 \end{bmatrix}$. Insert a row [1, -1, 0] as 2nd row to C. Then insert [0, 1, 2, 0] as 3rd column.
6. Find transpose of the matrices A, B and C given above.
7. Find determinant of A, B and C. Also find rank of each of these matrices.
8. Find inverse of A, B and C, if it exists.
9. Solve the following system of equations. x+y+z=3, x-y+z=1, x-y-z=-1. (Use solve

command)

10. Solve the following system of equations. x+y=3, x-y=1. (Use solve command)
11. Find solutions of systems of equation by matrix inversion method. (Ax=B → x= A^{-1} B)
12. Consider the system of equations Cx=D, where $C = \begin{bmatrix} 1 & 0 & 0 \\ 3 & 4 & 6 \\ 1 & 3 & 2 \end{bmatrix}, D = \begin{bmatrix} 3 \\ -1 \\ 8 \end{bmatrix}$. Find rank of C and rank of the augmented matrix [C, D]. Hence check whether Cx=D is solvable.

Assignment Evaluation

0: Not Done [ ]	1: Incomplete [ ]	2: Late Complete [ ]
3: Needs Improvement [ ]	4: Complete [ ]	5: Well Done [ ]

Signature of Instructor

Assignment No. 7 : Eigenvalues and Eigenvectors

If A is a square matrix of order A, then eigenvalues of A are roots of equations $|\lambda I - A| = 0$. A vector x, which satisfies $Ax = \lambda x$ is an eigenvector of A corresponding to eigenvalue λ.

To find eigenvalues and eigenvectors

u,v=la.eig(A) # Output : u = eigenvalues of A, v=eigenvectors of A

```
import numpy.linalg as la
u,v=la.eig(A)
print(u)
print(v)
#u stores eigenvalues and v stores eigenvecors
```

```
[-0.53112887  7.53112887  0.        ]
[[-0.79402877 -0.2928046  -0.57735027]
 [ 0.60788018 -0.9561723   0.57735027]
 [ 0.          0.          0.57735027]]
```

```
B=np.matrix([[2,27,0],[0,4,40],[0,3,30]])
B
```

```
matrix([[ 2, 27,  0],
        [ 0,  4, 40],
        [ 0,  3, 30]])
```

```
#Find eigenvalues of B
u,v=la.eig(B)
print(u)
```

```
[ 2.  0. 34.]
```

To check whether given matrix is diagonalizable

To check whether matrix A is diagonalizable, first find (matrix of) eigenvectors of A, say P.

If $P^{-1}AP$ is a diagonal matrix then A is diagonalizable.

```
C=np.matrix([[1,1,1],[0,1,1],[0,0,1]])
print(C)
```

```
[[1 1 1]
 [0 1 1]
 [0 0 1]]
```

```
q,p=la.eig(C)
```

```
p
```

```
matrix([[ 1.00000000e+00, -1.00000000e+00,  1.00000000e+00],
        [ 0.00000000e+00,  2.22044605e-16, -2.22044605e-16],
        [ 0.00000000e+00,  0.00000000e+00,  4.93038066e-32]])
```

```
la.inv(p)*C*p
```

```
matrix([[1.00000000e+00, 2.22044605e-16, 9.86076132e-32],
        [0.00000000e+00, 1.00000000e+00, 2.22044605e-16],
        [0.00000000e+00, 0.00000000e+00, 1.00000000e+00]])
```

Assignment 7

1. Find the eigenvalues of the matrix $A = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$.
2. Find eigenvalues and eigenvectors of $B = \begin{bmatrix} 1 & 2 & 0 \\ -7 & 5 & 6 \\ 0 & 3 & 1 \end{bmatrix}$.
3. Find the matrix P which diagonalizes the matrix A.
4. Find the matrix P which diagonalizes the matrix A.
5. Check whether C=$\begin{bmatrix} 1 & 1 & 3 \\ 4 & 0 & 1 \\ 5 & -2 & -1 \end{bmatrix}$ is diagonalizable.
6. For the given matrix D find eigenvalues and eigenvectors of D. Aldo check whether D is diagonalizable. $D = \begin{bmatrix} 1 & 12 & 0 \\ -8 & 5 & 5 \\ 0 & 0 & 1 \end{bmatrix}$

Assignment Evaluation

0: Not Done [ ]	1: Incomplete [ ]	2: Late Complete [ ]
3: Needs Improvement [ ]	4: Complete [ ]	5: Well Done [ ]

Signature of Instructor

Assignment No. 8 : Functions in Python

A function is a block of organized, reusable code that is used to perform a single, related action. Functions gives modularity and reusing of code in a program.
There are many built-in functions like in Python but you can also create your own functions. These functions are called user-defined functions.

How to define a function
SYNTAX:

```
def functionname( parameters ):
    function_body
    return [expression]
```

The function blocks starts with the keyword ***def*** followed by the function name and parentheses and colon(:).
The input parameters or arguments should be placed within these parentheses. You can also define parameters inside these parentheses.
The code block within every function starts with a colon (:) and is indented.
The statement return [expression] exits a function, the return statement with no arguments means return Nothing.

Consider the following function:

```
def revno(n):
    sum=0
    while (n>0):
        rem=n%10
        n=n/10
        sum=(sum*10)+rem
     return sum

print ("The reverse of 54321 is :",revno(54321))
```

Calling a function
Once the function is defined, you can execute it by calling it from another function or directly from the Python prompt. In the above code the function is called from print statement.

Function Arguments:
There are the following types of formal arguments:
1. Required arguments
2. Keyword arguments
3. Default arguments
4. Variable-length arguments

Required arguments
Required arguments are the arguments passed to a function in correct positional order. Here, the number of arguments in the function call should match exactly with the function definition.

```
#!/usr/bin/python
def display( str ):
        printstr;
        return;
# call the function
display("hello");
```

Keyword Arguments:

Keyword arguments are related to the function calls. When you use keyword arguments in a function call, the caller identifies the arguments by the parameter name. In this we can change the order of arguments or may skip it.

```
#!/usr/bin/python
def display( classname, roll_no ):
        print "Class: ", classname;
        print "Roll_no ", roll_no;
        return;
# call the function
display(roll_no=10, classname="SYBScCS" );
```

Return Statement:

The statement return [expression] exits a function, passing back an expression to the caller.

```
def checkprime(n):
        if n>1 :
            for i in range(2,n):
                if(n%i)==0:
                    return 0
         return 1

a=checkprime(10)
if a==1:
print " no is prime"
else:
print " no is not prime"
```

Functions returning multiple values:

```
def display(x, y):
        return x * 3, y * 4
a, b = display(5, 4)
print a
print b
```

Scope of Variables:

All variables in a program may not be accessible at all locations in that program. It depends on

where you have declared a variable. The scope of a variable is the portion of the program where you can access it. There are two basic scopes of variables in Python:

- Global variables
- Local variables

Variables that are defined inside a function body are having local scope, and defined outside the function body have a global scope.

Recursive Functions

The function which calls itself is a recursive function. Python allows us to write recursive functions.

Recursive function for factorial of a number:

```
def fact(n):
        if(n==1):
            return 1
        else:
            return n* fact(n-1)
print fact(4)
```

Assignment 8 – Functions

1. Write a Python function which accepts an integer x from a user and prints "POSITIVE" if x is positive, else prints "NEGATIVE".
2. Write a Python function which accepts length and breadth of a rectangle and prints its area.
3. Write a Python function which accepts, principal, rate of interest and no of years for investment and calculates total interest earned. Also print the amount an investor gets**.**
4. Write a python function which accepts radius as a parameter and returns area of the circle.
5. Write a Python function to find a root of an equation f(x)=0 with given initial approximation x0 using Newton Raphson Method. (Maximum number of iterations n)

   Algorithm:
   Step 1: Write a Python function to evaluate the function f(x)
   Step 2: Write a Python function to evaluate derivative of f(x), say df(x)
   Step 3:Write Python function to evaluate xnew = xold – f(xold)/df(xold). Use for loop to perform n iterations of this formula.
   Each iteration calculates xnew from xold. Xnew is displayed and this xnew becomes xold for the next iteration.

Assignment Evaluation

0: Not Done [ ]	1: Incomplete [ ]	2: Late Complete [ ]
3: Needs Improvement [ ]	4: Complete [ ]	5: Well Done [ ]

Signature of Instructor

Assignment No. 9 – Regula Falsi Method

Write a Python function to find a root of an equation f(x)=0 in the interval [a, b] using Regula Falsi Method. (Maximum number of iterations n)

Algorithm:

Step 1: Write a Python function to evaluate f(x)
Step 2: Verify that f(a), f(b) have opposite signs, else print error message and Stop.
Step 3: i=1 # loop counter
Step 4: Calculate c= (a*f(b) – b*f(a))/ (f(b) – f(a)).
Step 5: If f(a)*f(c) <0, then set b=c, else set a=c.
Step 6: print c #as approximate root.
Step 7: i=i+1.
Step 8: If i<=n, then go to Step 4, else Stop.

Assignment Evaluation

0: Not Done [ ]	1: Incomplete [ ]	2: Late Complete [ ]
3: Needs Improvement [ ]	4: Complete [ ]	5: Well Done [ ]

Signature of Instructor

Assignment No. 10 – Numerical Integration Trapezoidal Rule

Write a Python function to evaluate $\int_{x0}^{xn} f(x)dx$ using Trapezoidal rule, given (x0,y0), (x1,y1), …, (xn,yn).

Algorithm:

Step 1: Write a function with x, y as arguments.

Step 2: Find n= len(y) - 1 and h=x[1]-x[0]

Step 3: Implement the formula $Ans = \frac{h}{2}((y0 + yn) + 2(y1 + y2 + \cdots + y_{n-1}))$

Assignment Evaluation

0: Not Done [ ]	1: Incomplete [ ]	2: Late Complete [ ]
3: Needs Improvement [ ]	4: Complete [ ]	5: Well Done [ ]

Signature of Instructor

Assignment No. 11 – Numerical Integration Simpson's 1/3 Rule

Write a Python function to evaluate $\int_{x0}^{xn} f(x)dx$ using Trapezoidal rule, given (x0,y0), (x1,y1), …, (xn,yn).

Algorithm:

Step 1: Write a function with x, y as arguments.

Step 2: Find n= len(y) - 1 and h=x[1]-x[0]

Step 3: Implement the formula

$$Ans = \frac{h}{3}((y0 + yn) + 2(y2 + y4 + \cdots + y_{n-2}) + 4(y1 + y3 + \cdots + y_{n-1}))$$

Assignment Evaluation

0: Not Done [ ]	1: Incomplete [ ]	2: Late Complete [ ]
3: Needs Improvement [ ]	4: Complete [ ]	5: Well Done [ ]

Signature of Instructor

Assignment No. 12 – Numerical Integration Simpson's 3/8 Rule

Write a Python function to evaluate $\int_{x0}^{xn} f(x)dx$ using Trapezoidal rule, given (x0,y0), (x1,y1), …, (xn,yn).

Algorithm:

Step 1: Write a function with x, y as arguments.

Step 2: Find n= len(y) - 1 and h=x[1]-x[0]

Step 3: Implement the formula

$$Ans = \frac{3h}{8}((y0 + yn) + 2(y3 + y6 + \cdots + y_{n-3}) + 3(y1 + y2 + y4 + y5 + \cdots + y_{n-2} + y_{n-1}))$$

Assignment Evaluation

0: Not Done [ ]	1: Incomplete [ ]	2: Late Complete [ ]
3: Needs Improvement [ ]	4: Complete [ ]	5: Well Done [ ]

Signature of Instructor