INSTITUTE OF TECHNOLOGY & MANAGEMENT
www.itmgoi.in

श्रेष्ठ इंडस्ट्री इन्टरफेस के लिए
CMAI, AICTE & RGPV
द्वारा पुरस्कृत

INSTITUTE OF TECHNOLOGY
& MANAGEMENT
GWALIOR • MP • INDIA

Laboratory Manual

Programming in Python (IT-605)

For

Third Year Students
Department: Information Technology

Department of Information Technology

Vision of IT Department

The Department of Information Technology envisions preparing technically competent problem solvers, researchers, innovators, entrepreneurs, and skilled IT professionals for the development of rural and backward areas of the country for the modern computing challenges.

Mission of the IT Department

- To offer valuable education through an effective pedagogical teaching-learning process.
- To shape technologically strong students for industry, research & higher studies.
- To stimulate the young brain entrenched with ethical values and professional behaviors for the progress of society.

Program Educational Objectives

Graduates will be able to

- Our graduates will show management skills and teamwork to attain employers' objectives in their careers.
- Our graduates will explore the opportunities to succeed in research and/or higher studies.
- Our graduates will apply technical knowledge of Information Technology for innovation and entrepreneurship.
- Our graduates will evolve ethical and professional practices for the betterment of society.

Program Outcomes (POs)

Engineering Graduates will be able to:

1. **Engineering knowledge**: Apply the knowledge of mathematics, science, engineering
Fundamentals, and an engineering specialization to the solution of complex engineering problems.
2. **Problem analysis**: Identify, formulate, review research literature, and analyze complexengineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.
3. **Design/development of solutions**: Design solutions for complex engineering problemsand design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.
4. **Conduct investigations of complex problems**: Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.
5. **Modern tool usage**: Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations.
6. **The engineer and society**: Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.
7. **Environment and sustainability**: Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.
8. **Ethics**: Apply ethical principles and commit to professional ethics and responsibilitiesand norms of the engineering practice.
9. **Individual and team work**: Function effectively as an individual, and as a member orleader in diverse teams, and in multidisciplinary settings.
10. **Communication**: Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.
11. **Project management and finance**: Demonstrate knowledge and understanding of theengineering and management principles and apply these to one’s own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.
12. **Life-long learning**: Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological ch

Course Outcomes

Python (IT-605)

CO1:	Install Python and have knowledge of syntax of Python
CO2 :	Describe the Numbers, Math functions, Strings, List, Tuples and Dictionaries in Python
CO3 :	Express different Decision Making statements and Functions
CO4 :	Develop code in Python using functions, loops etc.
CO5 :	Design GUI Applications in Python and evaluate different database operations Understand and Describe of object oriented programming and discuss advantages over procedure oriented programming

Course	Course Outcomes	CO Attainment	PO1	PO2	PO3	PO4	PO5	PO6	PO7	PO8	PO9	PO10	PO11	PO12	PSO1	PSO2	PSO3
CO1	Install Python and have knowledge of syntax of Python.		2	1	1	0	1	0	0	0	0	0	0	0	1	1	0
CO2	Describe the Numbers, Math functions, Strings, List, Tuples and Dictionaries in Python		2	2	0	0	0	0	0	0	0	0	0	0	0	1	0
CO3	Express different Decision Making statements and Functions		2	2	1	1	1	0	0	0	0	0	0	0	1	0	1
CO4	Develop code in Python using functions, loops etc.		0	1	0	1	0	0	0	0	1	0	1	0	0	1	0
CO5	Design GUI Applications in Python and evaluate different database operations Understand and Describe of object oriented programming and discuss advantages over procedure oriented programming		0	1	0	1	1	0	2	1	0	0	0	0	0	0	1

List of Program

INTRODUCTION TO PYTHON

The Python applications are considered under the section on programming in Python. The following Python applications are covered in detail in this lab based on the RGPV Bhopal syllabus:

1) Console Based Programming
2) OOPs Based Programming
3) GUI Based Programming
4) String
5) List
6) Tuple
7) Dictionary

LAB REQUIREMENTS For Python Programming

- Python 3.7
- PyCharm
- Anaconda

This Interpreter has no special hardware requirements as such. Any System with a minimum 256 MBRAM and any normal processor can use for this lab.

Python

The objective of this exercise is to become familiar with the Python IDE for version 3.X while introducing basic mathematical operations, variable types, and printing options.

Background

An IDE, or integrated development environment, is used by almost all modern programming languages and facilitates the creation, editing, testing, and archiving of programs and modules. The Python IDE is referred to as IDLE, which, like many other things in the language, is a play on the name of one of the members of the British comedy group Monty Python, Eric Idle.
It's important to remember that there are three fundamental categories of simple variables in Python before launching IDLE: integers (whole numbers, which are frequently used as an index), floats (that is, numbers

strings (groups of alphanumeric letters like names, words, or numbers that are not mathematically modified like a part number or zip code) and decimal points (also known as real numbers). A letter must come first in a lawful variable name. After that, there is an optional group of characters, digits, and the underscore. It cannot be a reserved word—that is, a word with a specific meaning in the language, like a command or operator—nor may it contain any other letters or spaces. Variables in Python can be created by just declaring them and giving them a value. As examples, consider:

```
a=2.3
name=”Joe”
```

The equal sign is better understood as "gets." Put otherwise, consider the following examples: "The variable name gets the string Joe" in the second example and "The variable a gets the floating point value 2.3" in the first. These kinds of assignment commands essentially set aside memory on the computer's hard drive for the variable and identify it with its name. After that, it saves the relevant value there for later use.

OBTAINING USER DATA

Objective

Users—those who operate the program, who are not often programmers—must provide data for interactive applications. This exercise looks at the input() method and builds a basic Ohm's Law calculator.

Introduction

The input() function is the most versatile way to get data from the user. Python will wait for the user to enter a string of characters until the user presses the Enter key when it runs this command. After that, a variable is given these characters as a string. Typically, a user prompt—that is, a query given to the user—will be necessary. This can be done by passing this as an argument to the function or by using a separate print statement. Here's a straightforward example that requests the user's name and then prints it back.

```
print( “What is your name? ” )n = input()
print( “Hello”, n )
```

Alternately, this can be shortened with the following:

```
n = input(“What is your name? ”)print( “Hello”, n )
```

It is important to remember that this function always returns a string variable. If the entered data is numeric, it must be turned into either a float or integer. This can be accomplished via the float()and int() functions. For example:

p = float(input("What is your weight in pounds? "))kg = p / 2.2

print("You weigh approximately", kg, "kilograms")

Conditionals: if

Learning about the notion of branching is the aim of this assignment. For straightforward decision-making, we present the if conditional expression. Additionally, we will present the idea of menu-driven programming. We will develop a program to estimate battery life throughout this process.

Introduction

Simple linear or straight-line programs would be a good way to categorize our earlier programs. The program's flow was quite simple: it asked the user for information, asked for guidance, did some calculations using that information, and printed out the relevant results. The idea of branching represents the next degree of complexity. That is, based on specific circumstances, the code's execution path may change. This might be interpreted as the program choosing what action to take. The if statement is the basic conditional expression. This is how it seems.:

if conditional expression:Resulting action

The conditional expression is some manner of test, for example to see if one variable is larger thananother. The tests include = = (same as), != (not same as), >, <, >= and <=. The logical directives and and or are also available. Any legitimate block of Python code is the outcome. It could consist of one line or several lines. Thus, the resulting action is executed if the conditional expression is true. The action is skipped if the expression is false. Program execution begins at the next line following the consequent action block in both scenarios. Note that the resulting action block needs to be indented; this is very crucial. The block's lines must all have the same amount of indentation. Python determines that it is a single block of code in this way.

As an example, suppose we'd like to test to see if variable A is larger than variable B. If it is, we'dlike to print out the message: "It's bigger". After this, we want to print out the message "Done", whether or not A was larger.

if A > B:

print("It's bigger")print("Done")

Because the second print statement is not indented, it is not part of the block, therefore it is alwaysexecuted. If the second print statement had been indented instead, then “Done” would

only be printed if A was larger than B. A common beginner’s syntax error is to forget the colon at the endof the if statement.
For another example, consider that you have a floating point variable named T that represents a

computed time in hours. Instead of printing this out as hours with a fraction, you prefer to presentit as hours and minutes. A floor divide can be used to obtain the whole hours:
h = T // 1.0

Similarly, a modulo can be used to obtain the fractional portion, which when multiplied by 60.0 will yield the minutes:
m = (T % 1.0) * 60.0

So, you could print out the result as follows:

print(“The time is”, h, “hours and”, m, “minutes”)

Of course, what it the minutes portion works out to zero? Reading something like “The time is 5 hours and 0 minutes” looks a little strange. We’d prefer to leave off the “and 0 minutes” portion.This can be achieved with a simple set of if tests:
if m != 0.0:

print(“The time is”, h, “hours and”, m, “minutes”)if m == 0.0:
print(“The time is”, h, “hours”)

This sort of “one-or-the-other” construct is fairly common. To make life a little simpler, we can use the else clause:
if m != 0.0:

print(“The time is”, h, “hours and”, m, “minutes”)

else:

print(“The time is”, h, “hours”)

If m is non-zero, the full print statement is used, otherwise (else) the simplified version is used. Enter the completed program below and try it with several different values, some whole numbers,others not, and inspect the results:
T = float(input(“Please enter a time value: ”))h = T // 1.0
m = (T % 1.0) * 60.0

if m != 0.0:

```
print( “The time is”, h, “hours and”, m, “minutes” )
else:

        print( “The time is”, h, “hours” )
  print( “Done!” )
```

If you look carefully, you might note that under certain circumstances the printout may still beless than satisfactory (hint: what about seconds?). How might this issue be fixed?

ITERATION

Objective

This exercise aims to familiarize you with the idea of iteration, which is often referred to as looping. We will also look into how to make basic text-based graphs. In the process, a program will be developed to demonstrate the Maximum Power Transfer Theorem..

Introduction

A very potent computing tool is the capacity to repeat a set of instructions with controlled variance. In Python, there are several methods for doing this, each with advantages and disadvantages. The while loop is the initial loop control structure. This appears to be an if statement at first glance:

```
while control expression:statement block
```

The statement block will be repeated as long as the control expression is true, potentially consisting of several lines. This block needs to be indented, much like the if statement. The control expressions are typically straightforward variable tests, much like those found in if statements. Furthermore, in order to prevent the loop from trying to run indefinitely, it is crucial that one or more of the variables used in the control expression change during the looping process. As an illustration:

```
x=1
while x<10:
print( x )
```

A second technique to create a loop is through the for statement. The template

looks similar to thewhile structure:
for variable in value list:
statement block

value list can be a simple listing of values such as:
for x in 1,3,25,17:

For example:
for x in range(5):
print(x)
The code above will print the numbers 0, 1, 2, 3 and 4. Separate starting and ending values may alsobe used:

for x in range(3,7):
print(x)

This will print out the values 3, 4, 5, and 6. Next, an increment value may be included:

for x in range(3,11,2):
print(x)

TUPLES

Objective
The aim of this exercise is to become familiar with tuples and have a little fun besides.

Introduction
In general, there are two sorts of variables in Python: compound types, which store many instances of an item, and simple types, which include single things, such as integers and floats. We refer to these as sequences. Because a string is composed of a group of separate characters, it is a particular kind of sequence. Strings can have individual characters or groups of characters extracted from them, even though they are frequently handled as a single entity (a technique known as slicing). The tuple is another kind of sequence; it can be thought of as a contraction of multiple. A set of floats or ints can be contained in a basic tuple. It might also include a collection of sequences, such strings or even more tuples. .. Take into consideration, for illustration, a tuple containing several voltage settings. It could have this initialization.:

V = (3.5, 2.0, 4.5, 6.0, 50.0, 10.0)

The sequence is defined with enclosing parentheses and the individual items are separated by commas. Square brackets are used to access any given item or slice with the initial item at location 0, as shown in the examples below:

print(V[1])

print(V[4])

These yield 2.0 and 50.0, respectively. A slice refers to a range of locations. Two values are specifiedseparated by a colon: The first is the starting point while the second is the ending point (which is itself not included). It is also worth noting that a slice is itself a sequence and therefore will be printedwith surrounding parentheses. For example:
print(V[1:4])

This prints (2.0, 4.5, 6.0) If the start point is left off, it is assumed to be 0. Thus,
print(V[:4])

yields (3.5, 2.0, 4.5, 6.0).

Finally, a third argument may be added which indicates an increment. For
example:print(V[0:4:2])
yields (3.5, 4.5).

You can determine the number of items in a sequence by using the len()
function:print(len(V))

FUNCTIONS, LIST AND FILES

Objective

This activity aims to achieve three main goals: The first is to learn how to use functions written by programmers; the second is to learn how to use lists; and the third is to investigate ways to read and work with data from external files.

Introduction - Functions

In Python, these functions must be defined (or imported if they're in a module) before they are called within the main program. They may or may not have arguments and they may or may not return a value (possibly more than one). If these functions prove to be widely applicable, they may be placed into custom modules so that they can be used conveniently in other programs. Programmer defined functions are a very useful way to compartmentalize and reuse code. Once a bit of code has been created that performs a certain task, it can be reused repeatedly and makes the subsequent code more readable.

In contrast, consider a common function like round(). The variable you want to round and the number of places to round to are its two required inputs. It gives back a single value, which is the argument's rounded version. As an illustration:
x = round(y, 2)

y and 2 are the arguments to the function and it returns a result which we then assign to the variablex (that is, x gets y rounded to 2 places). Suppose you wish to create a function that produced (returned)the parallel equivalent of two resistors. The function would look something like this, using product-sum rule:

```
def parallel( r1, r2 ):
rp = r1*r2/(r1+r2)return rp
```

Introduction - lists

The second item of interest is the list. Lists are sequences like tuples, but unlike tuples, lists are mutable, that is, the elements within a list maybe changed. Tuples are best thought of as a collectionof constants in comparison. Lists are defined using square brackets [] instead of using parentheses() like a tuple. Like tuples and strings, lists are accessed by using square brackets []. Lists can be sliced just like tuples.

```
T = (12,43,17)      # This is a tuple definition
L = [12,43,17]      # This is a list definition
print( T[0] )       # print first element of tuple
print( L[0] )       # print first element of list
```

Introduction - Files

Keeping data in a software or expecting the user to enter it again every time the application is used are often impractical. Just think of how useless a word processor would be if you couldn't save documents anywhere other than inside the software. Here's when the idea of files becomes relevant. In the end, programmers typically only need to accomplish a few things with files: You can move around in them (for example, to skip unnecessary sections), read from them, write to them, open them (that is, obtain access to them, frequently exclusively, by using a suitable filename and/or path), and release or close them (so that other programs can acquire access). Any software that works with files must open and shut them; however, depending on what the program requires from the file, it will determine whether or not it is read from, written to, and moved inside. Only opening, reading, and closing will be used in the following practice. Additionally, files can be classified as either text or binary. Text files are typically simpler to use, but binary files have the potential to be more powerful. We will just examine text files in the upcoming experiment. While binary files are typically unreadable by standard text editors, these files contain strings that can be viewed with any text editor.
In order to gain access to a file, it must first be opened:

fil = open(filename, mode)

filename is the literal disk file name such as H:\myfolder\myfile.dat. This is a string that can be hardcoded as a constant (rarely) or more commonly obtained from the user via a input() statement.mode is a string that describes how the file is to be accessed. “r” is used for reading and “w” may be used for writing. There are other modes as well. fil is a file object that is returned to you. It willbe needed for subsequent read and write calls. Note that Python allows several files to be open at once, hence the need for file objects. So, a read mode access might look like this:

fn = input(“Please enter the file name: ”)fil = open(fn, “r”)

Once the file object is obtained, data may be read from the file. Data can be read character
by character or line by line. Line oriented files are easily read as follows:
str = fil.readline()
When you are done with the file, you need to close it. close() is another file object method:fil.close()

Object: To write a python program that takes in command line arguments as input and print the number of arguments.

Explanation: Python provides various ways of dealing with these types of arguments. The three most common are:

- Using sys.argv
- Using getopt module
- Using argparse module

Using sys.argv

One such variable is sys.argv which is a simple list structure. It’s main purpose are:

- It is a list of command line arguments.
- len(sys.argv) provides the number of command line arguments.
- sys.argv[0] is the name of the current Python script.

Code:

```
import sys


# total arguments

n = len(sys.argv)

print("Total arguments passed:", n)


# Arguments passed

print("\nName of Python script:", sys.argv[0])


print("\nArguments passed:", end = " ")

for i in range(1, n):

    print(sys.argv[i], end = " ")


# Addition of numbers

Sum = 0
```

```
# Using argparse module
for i in range(1, n):
    Sum += int(sys.argv[i])


print("\n\nResult:", Sum)
```

Output:

```
 python3 gfg.py 2 3 5 6
Total arguments passed: 5
Name of Python script: gfg.py
Arguments passed: 2 3 5 6
Result: 16
```

1.To Write a Python Program to find the square root of a number by Newton's Method

Explanation:

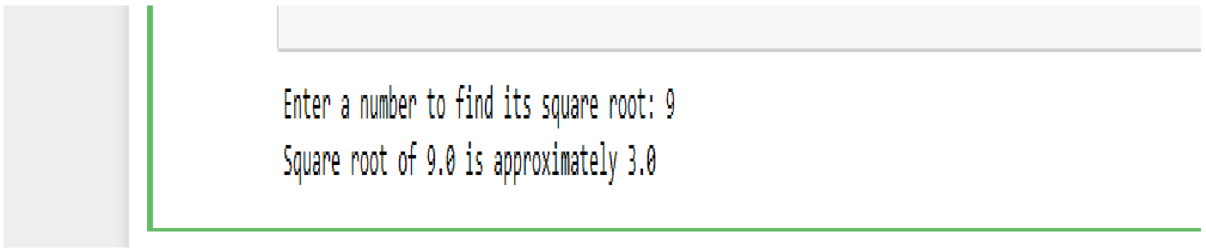
This code defines a function square_root(n, tolerance) which takes a number n as input and returns its square root using Newton's method. The tolerance parameter determines how close the approximation needs to be to consider the calculation accurate enough. The program then asks the user for input and prints out the square root of the provided number.

Code:

```
def square_root(n, tolerance=0.00001):

# Initial guess

    x = n / 2

    # Iterate until the difference between successive approximations is less than
tolerance

    while True:

        # Update the approximation using Newton's method

        root = 0.5 * (x + n / x)

        # Check if the difference between successive approximations is less than
tolerance

        if abs(root - x) < tolerance:

            break

        x = root

    return root

# Example usage:

if __name__ == "__main__":

    # Taking input from the user

    number = float(input("Enter a number to find its square root: "))
```

```
# Calculate square root using Newton's method

result = square_root(number)

print(f"Square root of {number} is approximately {result}")
```

Output:



```
Enter a number to find its square root: 9
Square root of 9.0 is approximately 3.0
```

3.To Write a Python program to find the exponentiation of a number.

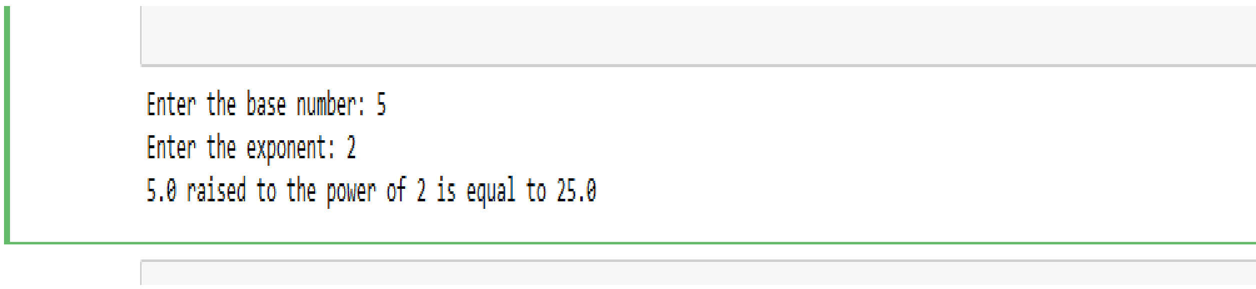
Algorithm:

1. Take the base and exponent as input from the user.
2. Initialize a variable to store the result.
3. Use a loop to multiply the base by itself for the number of times specified by the exponent.
4. Return the result.

Code:

```
def power(base, exponent):
    """
    Computes the power of a number.

    Args:
    - base: the base number
    - exponent: the exponent to raise the base to

    Returns:
    - result: the result of the exponentiation operation
    """
    result = 1
    for _ in range(exponent):
        result *= base
    return result

# Example usage:
if __name__ == "__main__":
    base = float(input("Enter the base number: "))
    exponent = int(input("Enter the exponent: "))
    result = power(base, exponent)
    print(f"{base} raised to the power of {exponent} is equal to {result}")
```

Output:



```
Enter the base number: 5
Enter the exponent: 2
5.0 raised to the power of 2 is equal to 25.0
```

4. To write a python program to compute the GCD of two numbers.

Algorithm:

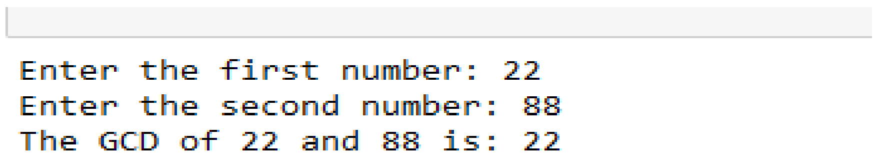
greatest common divisor (G.C.D) of two numbers is the largest positive integer that perfectly divides the two given numbers

1. Start iterating from 1 up to the minimum of the two numbers.
2. For each value i, check if both numbers are divisible by i.
3. If both numbers are divisible by i, update the GCD.
4. Repeat the process until reaching the minimum of the two numbers.
5. The last value of i that divides both numbers evenly will be the GCD.

Code:

```
def gcd(a, b):
    """
    Computes the greatest common divisor (GCD) of two numbers
    Args:
    - a: first integer
    - b: second integer

    Returns:
    - gcd: the greatest common divisor of a and b
    """
    gcd_result = 1
    # Iterate from 1 up to the minimum of a and b
    for i in range(1, min(a, b) + 1):
        # Check if both a and b are divisible by i
        if a % i == 0 and b % i == 0:
            gcd_result = i  # Update gcd_result
    return gcd_result

# Example usage:
if __name__ == "__main__":
    num1 = int(input("Enter the first number: "))
    num2 = int(input("Enter the second number: "))
    result = gcd(num1, num2)
    print(f"The GCD of {num1} and {num2} is: {result}")
```

Output:



```
Enter the first number: 22
Enter the second number: 88
The GCD of 22 and 88 is: 22
```

5. To write a python program to find first n prime numbers.

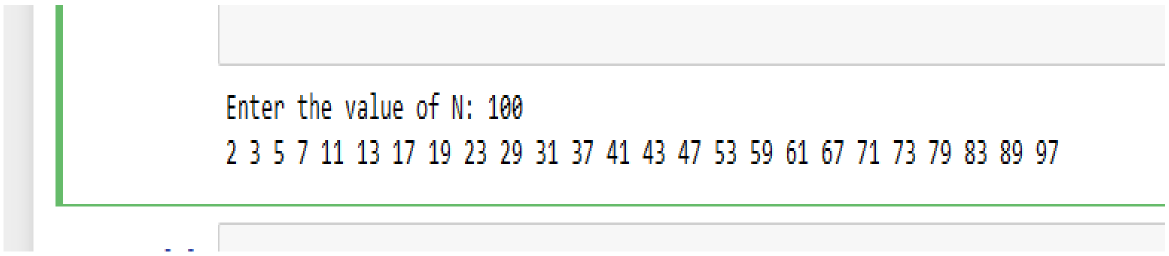
Algorithm:

1. First, take the number N as input.
2. Then use a for loop to iterate the numbers from 1 to N
3. Then check for each number to be a prime number. If it is a prime number, print it.

Code:

```
#function to check if a given number is prime

def isPrime(n):
#since 0 and 1 is not prime return false.

if(n==1 or n==0): return False

#Run a loop from 2 to n-1
    for i in range(2,n):

#if the number is divisible by i, then n is not a prime number.
          if(n%i==0):
                return False

#otherwise, n is prime number.
    return True

# Driver code
N=int(input("Enter the value of N: "))
#check for every number from 1 to N

for i in range(1,N+1):
#check if current number is prime

if(isPrime(i)):
          print(i,end=" ")
```

Output:



```
Enter the value of N: 100
2 3 5 7 11 13 17 19 23 29 31 37 41 43 47 53 59 61 67 71 73 79 83 89 97
```

6. To write a python program find the maximum of a list of numbers.

Algorithm:

1. Start with the first number in the list and assume it as the maximum.
2. Iterate through the list of numbers.
3. For each number in the list:

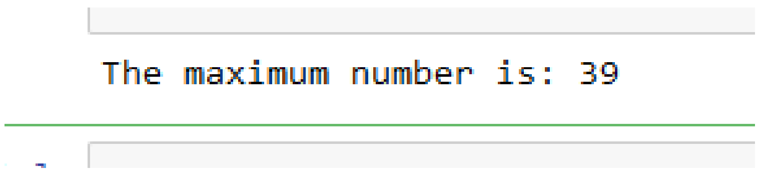
If the current number is greater than the assumed maximum, update the maximum to be the current number.

4. After iterating through all numbers in the list, the maximum will be the maximum found during the iteration.

Code:

```
def find_maximum(numbers):
    maximum = numbers[0]  # Assume the first number is the maximum
    for num in numbers:
        if num > maximum:
            maximum = num
    return maximum
numbers = [10, 5, 9, 12, 39]

maximum = find_maximum(numbers)
print(f"The maximum number is: {maximum}")
```

Output:



```
The maximum number is: 39
```

7. To write a python program to perform Matrix Multiplication.

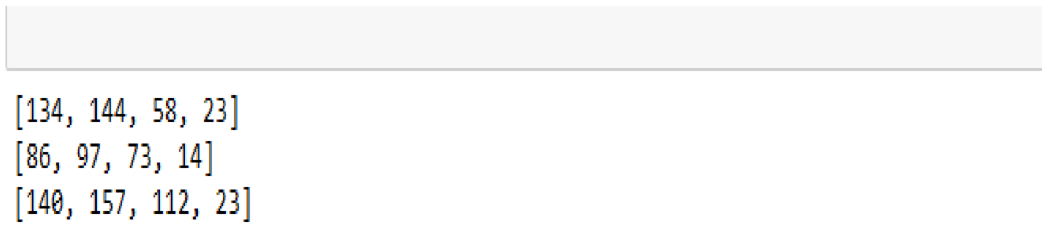
Algorithm:

```
1.  Define two matrices X and Y
2.  Create a resultant matrix named ‘result’
3.  for i in range(len(X)):
      for j in range(len(Y[0])):
      for k in range(len(Y))
      result[i][j] += X[i][k] * Y[k][j]
4.  for r in result, print the value of r
```

Code:

```
# 3x3 matrix
X = [[10,7,3],
    [4 ,5,6],
    [7 ,8,9]]

# 3x4 matrix
Y = [[8,8,1,2],
    [6,7,3,0],
    [4,5,9,1]]

# result is 3x4
result = [[0,0,0,0],
         [0,0,0,0],
         [0,0,0,0]]

# iterate through rows of X
for i in range(len(X)):
   # iterate through columns of Y
   for j in range(len(Y[0])):
       # iterate through rows of Y
       for k in range(len(Y)):
           result[i][j] += X[i][k] * Y[k][j]

for r in result:
   print(r)
```

Output:



```
[134, 144, 58, 23]
[86, 97, 73, 14]
[140, 157, 112, 23]
```

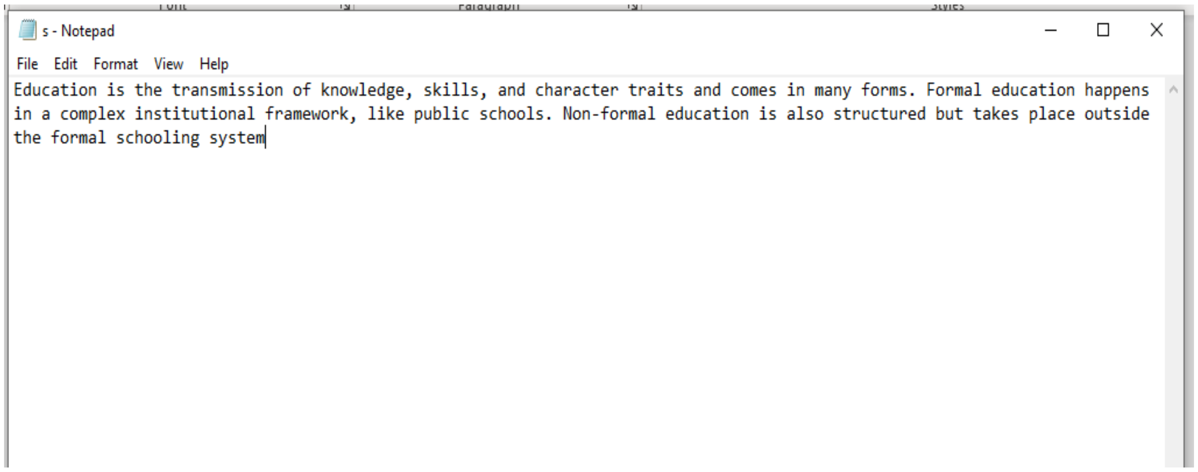
8. To write a python program to find the most frequent words in a text file

Algorithm:

1. Open the file in read mode and assign to fname variable.
2. Initialize count, maxcount to 0.
3. Spilt the file into no. of lines.
4. Spilt the lines into words.
5. Loop through each words and count the occurrences.

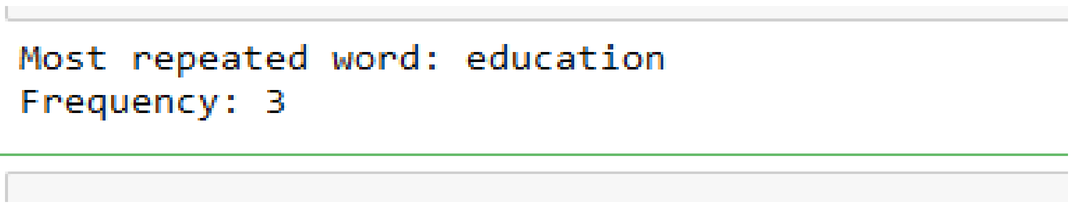
Code:

```
# A file named "s", will be opened with the
# reading mode.
file = open("s.txt","r")
frequent_word = ""
frequency = 0
words = []

# Traversing file line by line
for line in file:

    # splits each line into
    # words and removing spaces
    # and punctuations from the input
    line_word = line.lower().replace(',','').replace('.','').split(" ");

    # Adding them to list words
    for w in line_word:
        words.append(w);

# Finding the max occurred word
for i in range(0, len(words)):

    # Declaring count
    count = 1;

    # Count each word in the file
    for j in range(i+1, len(words)):
        if(words[i] == words[j]):
            count = count + 1;
        # If the count value is more
        # than highest frequency then
        if(count > frequency):
                frequency = count;
                frequent_word = words[i];

print("Most repeated word: " + frequent_word)
print("Frequency: " + str(frequency))
file.close();
```



```
Education is the transmission of knowledge, skills, and character traits and comes in many forms. Formal education happens
in a complex institutional framework, like public schools. Non-formal education is also structured but takes place outside
the formal schooling system
```

Output:



```
Most repeated word: education
Frequency: 3
```

9. Write a Python Program to perform Linear Search

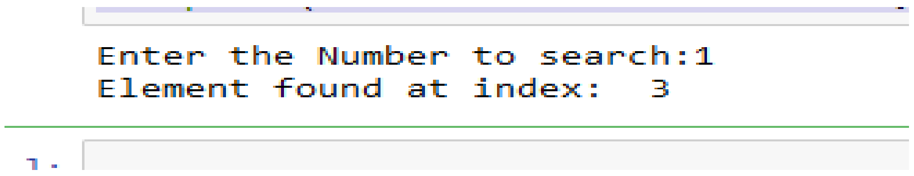
Algorithm:

1. Read n elements into the list
2. Read the element to be searched
3. If alist[pos]==item, then print the position of the item
4. else increment the position and repeat step 3 until pos reaches the length of the list

Code:

```
def linearSearch(array, n, x):

    # Going through array sequencially
    for i in range(0, n):
        if (array[i] == x):
            return i
    return -1

array = [2, 4, 0, 1, 9]
x =int(input("Enter the Number to search:"))
n = len(array)
result = linearSearch(array, n, x)
if(result == -1):
    print("Element not found")
else:
    print("Element found at index: ", result)
```

Output:



```
Enter the Number to search:1
Element found at index:  3
```

10. To write a python program to perform Binary search.

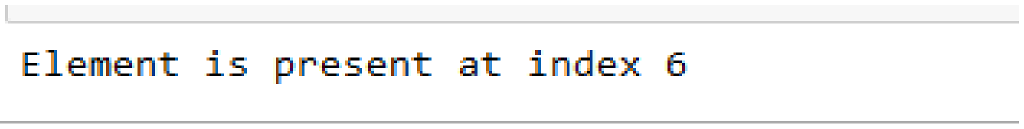
Algorithm:

1. Read n elements into the list
2. Read the element x to be searched
3. Compare x with the middle element.
4. If x matches with middle element, we return the mid index.
5. Else If x is greater than the mid element, then x can only lie in right half subarray
6. a. After the mid element. So we recur for right half.
7. Else (x is smaller) recur for the left half.

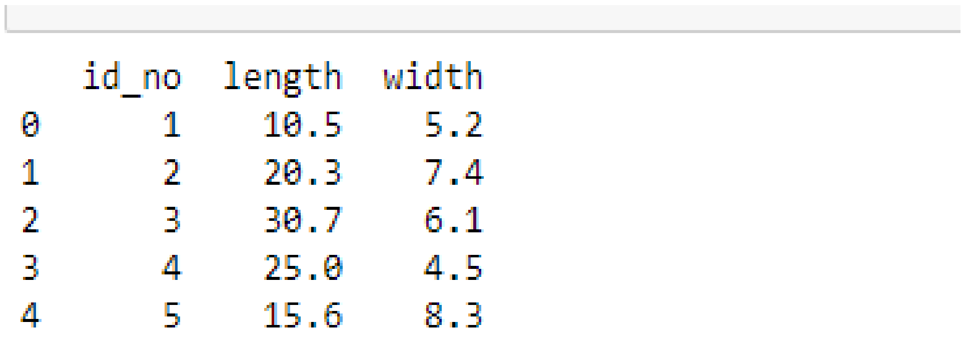
Code:

```
def binarySearch(array, x, low, high):

    # Repeat until the pointers low and high meet each other

    while low <= high:

     mid = low + (high - low)//2

        if array[mid] == x:

            return mid

        elif array[mid] < x:

            low = mid + 1

        else:

            high = mid - 1

    return -1

array = [3, 4, 5, 6, 7, 8, 9]

x = 9

result = binarySearch(array, x, 0, len(array)-1)

if result != -1:

    print("Element is present at index " + str(result))

else:
    print("Not found")
```

Output:



```
Element is present at index 6
```



```
   id_no  length  width
0      1    10.5    5.2
1      2    20.3    7.4
2      3    30.7    6.1
3      4    25.0    4.5
4      5    15.6    8.3
```