Laboratory Manual

Name of the School:	**School of Computing Sciences and Engineering**
Name of the Department:	**Computer Science and Engineering**
Name of the Programme:	**B.Tech CSE**
Course/ Course Code:	**Python Programming 17YCS512**
Academic Year:	**2022-23**

Guidelines

Laboratory rules

1. Attendance is required for all lab classes. Students who do not attend lab will not receive credit.
2. Ensure that you are aware of the test and its procedure before each lab class. **You will NOT be allowed to attend the class if you a report unprepared!**
3. Personal safety is top priority. Do not use equipment that is not assigned to you.
4. All accidents must be reported to your instructor or laboratory supervisor.
5. The surroundings near the equipment must be cleaned before leaving each lab class.
6. Ensure that readings are checked and marked by your TA for each lab period.

Laboratory report

1. Each student has to submit the report for each experiment.
2. Your report is to be written only in the space provided in the manual
3. Please write your number and, batch and group number.
4. Your report must be neat, well organized, and make a professional impact. Label all axes and include proper units.
5. Your reports should be submitted within 7 days and before the beginning of the lab class of the next batch.
6. Your reports will be as per rubrics provided at the end of each experiment
7. Anyone caught plagiarizing work in the laboratory report, from a current or past student's notebook, will receive 0 on the grading scale.

Name of Student .....

Academic Year .....

Programme .....

Class/ Roll No. .....

PRN No. .....

CERTIFICTAE

This is to certify that Mr./Miss……

Roll no…….. PRN No……

of class…… ……..has satisfactorily/unsatisfactorily completed the Term Work of Course …….

in this School during academic year …….

Course Teacher Head of Department Dean Academics

Table of Contents

Experiment No: 1

Title:

Implement a Python program to Calculate GCD of two numbers.

Theory:

Greatest Common Divisor (GCD) is a mathematical term to find the greatest common factor that can perfectly divide the two numbers. A GCD is also known as the **Highest Common Factor (HCF)**. For example, the HCF/ GCD of two numbers 54 and 24 is 6. Because 6 is the largest common divisor that completely divides 54 and 24.

GCD Using recursion Recursion is a memory consuming function defined in python that calls itself via self-referential expression. It means that the function will continuously call and repeat itself until the defined condition is met to return the greatest common divisor of the number

Procedure:

```
# Python code to demonstrate naive

# method to compute gcd ( recursion )

def hcf(a, b):

if(b == 0):

return a

else:

return hcf(b, a % b)


a = 60

b = 48

# prints 12

print("The gcd of 60 and 48 is : ", end="")

print(hcf(60, 48))
```

Results: The gcd of 60 and 48 is : 12

Experiment No: 2

Title: Implement a Python Program to calculate the square root of a number by Newton's Method.

Theory:

You have seen that the math library contains a function that computes the square root of numbers. In this exercise, you are to write your own algorithm for computing square roots. One way to solve this problem is to use a guess-and-check approach. You first guess what the square root might be and then see how close your guess is. You can use this information to make another guess and continue guessing until you have found the square root (or a close approximation to it). One particularly good way of making guesses is to use Newton's method. Suppose x is the number we want the root of and guess is the current guessed answer. The guess can be improved by using as the next guess.

Write a program that implements Newton's method. The program should prompt the user for the value to find the square root of (x) and the num ber of times to improve the guess. Starting with a guess value of 1/2, your program should loop the specified number of times applying New- ton's method and report the final value of guess. You should also subtract your estimate from the value of math.sqrt(x) to show how close it is.

Procedure:

```
def newton_method(number, number_iters = 100):
    a = float(number)
    for i in range(number_iters):
        number = 0.5 * (number + a / number)
    return number
a=int(input("Enter first number:"))
b=int(input("Enter second number:"))
```

```
print("Square root of first number:",newton_method(a))

print("Square root of second number:",newton_method(b))
```

Results:

```
Enter first number:81
Enter second number:5

Square root of first number: 9.0
Square root of second number: 2.23606797749979
```

Experiment No: 3

Title:

Implement a Python Program to find the largest number from a list of numbers

Theory:

largest number in a list. List is an ordered set of values enclosed in square brackets [ ]. List stores some values called elements in it, which can be accessed by their particular index.

For executing this program in Python, there are multiple approaches we can follow-

1. By comparing each element for finding the largest number
2. By using max() function
3. By sorting the list using sort() function and printing the last element in the list

Procedure:

```
def max_num_in_list( list ):
    max = list[ 0 ]
    for a in list:
        if a > max:
            max = a
    return max
print(max_num_in_list([1, 2, -8, 0]))
```

Results:

```
2
```

Experiment No: 4

Title:

Implement a Python Program to perform Liner search

Theory:

Linear search is a method of finding elements within a list. It is also called a sequential search. It is the simplest searching algorithm because it searches the desired element in a sequential manner It compares each and every element with the value that we are searching for. If both are matched, the element is found, and the algorithm returns the key's index position.

Procedure:

```
def linear_Search(list1, n, key):

# Searching list1 sequentially
for i in range(0, n):
if (list1[i] == key):
return i
return -1

 list1 = [1 ,3, 5, 4, 7, 9]
key = 7

n = len(list1)
res = linear_Search(list1, n, key)
if(res == -1):
print("Element not found")
else:
print("Element found at index: ", res)
```

Results:

Element found at index: 4

Experiment No: 5

Title:

Implement a Python Program to perform Binary search

Theory:

A binary search is an algorithm to find a particular element in the list. Suppose we have a list of thousand elements, and we need to get an index position of a particular element. We can find the element's index position very fast using the binary search algorithm.

There are many searching algorithms but the binary search is most popular among them.

The elements in the list must be sorted to apply the binary search algorithm. If elements are not sorted then sort them first.

Procedure:

```
    def binary_search(list1, n):
  low = 0
  high = len(list1) - 1
  mid = 0

      while low <= high:
          # for get integer result
          mid = (high + low) // 2

          # Check if n is present at mid
          if list1[mid] < n:
      low = mid + 1

          # If n is greater, compare to the right of mid
    elif list1[mid] > n:
              high = mid - 1
```

```
        # If n is smaller, compared to the left of mid
    else:
      return mid

            # element was not present in the list, return -1
        return -1


    # Initial list1
    list1 = [12, 24, 32, 39, 45, 50, 54]
    n = 45

    # Function call
    result = binary_search(list1, n)

    if result != -1:
        print("Element is present at index", str(result))
    else:
        print("Element is not present in list1")
```

Results:

Element in present at index 4

Experiment No: 6

Title:

Implement a Python Program to perform insertion sort.

Theory:

The Insertion sort is a straightforward and more efficient algorithm than the previous bubble sort algorithm. The insertion sort algorithm concept is based on the deck of the card where we sort the playing card according to a particular card. It has many advantages, but there are many efficient algorithms available in the data structure.

The insertion sort implementation is easy and simple because it's generally taught in the beginning programming lesson. It is an **in-place** and **stable algorithm** that is more beneficial for nearly-sorted or fewer elements.

Procedure:

```
def insertionSort(nlist):
  for index in range(1,len(nlist)):

    currentvalue = nlist[index]
    position = index

    while position>0 and nlist[position-1]>currentvalue:
        nlist[position]=nlist[position-1]
        position = position-1

    nlist[position]=currentvalue

nlist = [14,46,43,27,57,41,45,21,70]
insertionSort(nlist)
print(nlist)
```

Results:

```
[14, 21, 27, 41, 43, 45, 46, 57, 70]
```

Experiment No: 7

Title:

Implement a Python Program to perform selection sort.

Theory:

In this tutorial, we will implement the selection sort algorithm in Python. It is quite straightforward algorithm using the less swapping.

In this algorithm, we select the smallest element from an unsorted array in each pass and swap with the beginning of the unsorted array. This process will continue until all the elements are placed at right place. It is simple and an in-place comparison sorting algorithm.

Procedure:

```
def selection_sort(array):
        length = len(array)

        for i in range(length-1):
            minIndex = i

            for j in range(i+1, length):
                if array[j]<array[minIndex]:
                    minIndex = j

            array[i], array[minIndex] = array[minIndex], array[i]
        return array
        array = [21,6,9,33,3]
print("The sorted array is: ", selection_sort(array))
```

Results:

The sorted array is:[3,6,9,21,33]

Experiment No: 8

Title:

Implement a Python program to multiply matrices.

Theory:

Matrix multiplication is a binary operation that uses a pair of matrices to produce another matrix. The elements within the matrix are multiplied according to elementary arithmetic.

In the multiplication of two matrices, the row elements of the first matrix are multiplied to the column elements of the second matrix.

Procedure:

```
# Define two matrix A and B in program
    A = [[5, 4, 3],
         [2, 4, 6],
         [4, 7, 9]]
    B = [[3, 2, 4],
         [4, 3, 6],
         [2, 7, 5]]
    # Define an empty matrix to store multiplication result
    multiResult = [[0, 0, 0],
                   [0, 0, 0],
                   [0, 0, 0]]
    # Using nested for loop method on A & B matrix
    for m in range(len(A)):
      for n in range(len(B[0])):
          for o in range(len(B)):
```

```
    multiResult[m][n] += A[m][o] * B[o][n] # Storing multiplication result
 in empty matrix
# Printing multiplication result in the output
print("The multiplication result of matrix A and B is: ")
for res in multiResult:
  print(res)
```

Results:

The multiplication result of matrix A and B is:

[37, 43, 59]

[34, 58, 62]

[58, 92, 103]

Experiment No: 9

Title:

Implement a Python program to calculate the most frequent words in a text from a file.

Theory:

In this program, we need to find the most frequent word present in given text file. This can be done by opening a file in read mode using file pointer. Read the file line by line. Split a line at a time and store in an array. Iterate through the array and find the frequency of each word and compare the frequency with maxcount. If frequency is greater than maxcount then store the frequency in maxcount and corresponding word that in variable word. The content of data.txt file used in the program is shown below.

Procedure:

```
count = 0;
word = "";
maxCount = 0;
words = [];

#Opens a file in read mode
file = open("data.txt", "r")

#Gets each line till end of file is reached
for line in file:
   #Splits each line into words
   string = line.lower().replace(',',").replace('.',").split(" ");
   #Adding all words generated in previous step into words
   for s in string:
      words.append(s);

#Determine the most frequent word in a file
for i in range(0, len(words)):
   count = 1;
   #Count each word in the file and store it in variable count
```

```
        for j in range(i+1, len(words)):
            if(words[i] == words[j]):
                count = count + 1;

    #If maxCount is less than count then store value of count in maxCount
    #and corresponding word to variable word
    if(count > maxCount):
        maxCount = count;
        word = words[i];

print("Most frequent word: " + word);
file.close();
```

Results:

Most frequent word: computer

Experiment No: 10

Title: Implement function overloading with different function signatures

Procedure:

```
# Function to take multiple arguments
def add(datatype, *args):
    # if datatype is int
    # initialize answer as 0
    if datatype == 'int':
        answer = 0

    # if datatype is str
    # initialize answer as ''
    if datatype == 'str':
        answer = ''

    # Traverse through the arguments
    for x in args:
        # This will do addition if the
        # arguments are int. Or concatenation
        # if the arguments are str
        answer = answer + x

    print(answer)

# Integer
add('int', 5, 6)

# String
add('str', 'Hi ', 'how are you?!')
```

Results:

```
11
Hi how are you?!
```

Experiment No: 11

Title: Implement concept of class, instances and inheritance

Theory:

Inheritance enables us to define a class that takes all the functionality from a parent class and allows us to add more. In this tutorial, you will learn to use inheritance in Python.

Inheritance is a powerful feature in object oriented programming.

It refers to defining a new class with little or no modification to an existing class. The new class is called derived (or child) class and the one from which it inherits is called the base (or parent) class.

Procedure:

```
# (Generally, object is made ancestor of all classes)
# In Python 3.x "class Person" is
# equivalent to "class Person(object)"

class Person(object):
    # C
    # constructor
    def __init__(self, name):
        self.name = name

    # To get name
    def getName(self):
        return self.name

    # To check if this person is an employee
    def isEmployee(self):
        return False


# Inherited or Subclass (Note Person in bracket)
class Employee(Person):
    # Here we return true
    def isEmployee(self):
```

```
        return True


# Driver code
emp = Person("1")  # An Object of Person
print(emp.getName(), emp.isEmployee())

emp = Employee("2")  # An Object of Employee
print(emp.getName(), emp.isEmployee())
```

Results:

1 False

2 True

Experiment No: 12

Title: Search content using regular expression library in python

Theory:

A RegEx, or Regular Expression, is a sequence of characters that forms a search pattern.

RegEx can be used to check if a string contains the specified search pattern.

Steps of Regular Expression Matching:

1. Import the regex module with import re.
2. Create a Regex object with the re. compile() function. ...
3. Pass the string you want to search into the Regex object's search() method. ...
4. Call the Match object's group() method to return a string of the actual matched text.

Procedure:

```
import re

#Check if the string starts with "The" and ends with "Spain":

txt = "The rain in Spain"

x = re.search("^The.*Spain$", txt)

if x:

  print("YES! We have a match!")

else:

  print("No match")
```

Results:

YES! We have a match!

Experiment No: 13

Title: Implement Matrix multiplication using multi-threading in python

Theory:

Multithreading refers to concurrently executing multiple threads by rapidly switching the control of the CPU between threads (called context switching). The Python Global Interpreter Lock limits one thread to run at a time even if the machine contains multiple processors.

Benefits of Multithreading in Python:

Following are the benefits to create a multithreaded application in Python, as follows:

1. It ensures effective utilization of computer system resources.
2. Multithreaded applications are more responsive.
3. It shares resources and its state with sub-threads (child) which makes it more economical.
4. It makes the multiprocessor architecture more effective due to similarity.
5. It saves time by executing multiple threads at the same time.
6. The system does not require too much memory to store multiple threads.

Procedure:

```
from threading import Thread
import random
import math
import numpy as np
import time

Matrix_A = []
Matrix_B = []
Matrix_C = []

size_of_vectors_n = [int(math.pow(10, 2)), int(math.pow(10, 3)),
int(math.pow(10, 4))]
```

```
dimension_N = 2  # Default to a 2x2 matrix

num_of_threads = 1


def Input_for_matrix_dimensions():
    global dimension_N
    global num_of_threads

    dimension_N = int(input("Enter the number N to generate NxN matrix : "))
    num_of_threads = int(input("Enter the number of threads : "))


def Initialize_Matrix():
    global Matrix_A
    global Matrix_B
    global Matrix_C
    # for i in range(0,dimension_N):
    #     Matrix_A.append([random.randint(1,5) for i in range(0,dimension_N)])
    # print(Matrix_A)

    # Using numpy to generate matrix
    Matrix_A = np.random.random((dimension_N, dimension_N))
    Matrix_A = Matrix_A * 10
    Matrix_A = Matrix_A.astype(int)

    Matrix_B = np.random.random((dimension_N, dimension_N))
    Matrix_B = Matrix_B * 10
    Matrix_B = Matrix_B.astype(int)

    Matrix_C = np.zeros((dimension_N, dimension_N))
    Matrix_C = Matrix_C.astype(int)


def Matrix_multiply_parallel(start, end):
    for i in range(start, end):
        for j in range(dimension_N):
            for k in range(dimension_N):
                Matrix_C[i][j] += int(Matrix_A[i][k] * Matrix_B[k][j])
    print(Matrix_C)

def Thread_function():
    global num_of_threads
```

```
    thread_handle = []

    for j in range(0, num_of_threads):
        t = Thread(target=Matrix_multiply_parallel,
                   args=(int((dimension_N / num_of_threads) * j), int((dimension_N /
num_of_threads) * (j + 1))))
        thread_handle.append(t)
        t.start()

    for j in range(0, num_of_threads):
        thread_handle[j].join()


if __name__ == "__main__":
    Input_for_matrix_dimensions()
    Initialize_Matrix()

    start_time = time.time()
    Thread_function()
    # end_time = time.time()

    # print("Time taken to multiply two matrices in parallel comes out to be : " +
str(end_time - start_time))
```

Results:

```
Enter the number N to generate NxN matrix : 5

Enter the number of threads : 2

[[167 119  97 141 105]

 [186 138 110 188 180]

 [  0   0   0   0   0]

 [  0   0   0   0   0]

 [  0   0   0   0   0]]

[[167 119  97 141 105]

 [186 138 110 188 180]

 [171 132  86 171 137]
```

```
[114 107  38 131 138]
[167 118  99 161 144]]
```