LAKSHMI NARAIN COLLEGE OF TECHNOLOGY & SCIENCE, BHOPAL

CS-506
Python

Vision of the Department

To be recognized for keeping innovation, research and excellence abreast of learning in the field of computer science & engineering to cater the global society.

Mission of the Department

M1: To provide an exceptional learning environment with academic excellence in the field of computer science and engineering.

M2: To facilitate the students for research and innovation in the field of software, hardware and computer applications and nurturing to cater the global society.

M3: To establish professional relationships with industrial and research organisations to enable the students to be updated of the recent technological advancements.

M4: To groom the learners for being the software professionals catering the needs of modern society with ethics, moral values and full of patriotism.

Program Educational Objectives (PEO's)

PEO1: The graduate will have the knowledge and skills of major domains of computer science and engineering in providing solution to real world problems most efficiently.

PEO2: The graduate will be able to create and use the modern tools and procedures followed in the software industry in the relevant domain.

PEO3: The graduate will be following the ethical practices of the software industry and contributing to the society as a responsible citizen.

PEO4: The graduate will have the innovative mindset of learning and implementing the latest developments and research outcomes in the computer hardware and software to keep pace with the fast changing socio economic world.

LAKSHMI NARAIN COLLEGE OF TECHNOLOGY & SCIENCE, BHOPAL

CS-506
Python

COURSE OUTCOMES

CO1: Demonstrate basic concept of programming with Python.

CO2: Use premitive data structure in Python and application.

CO3: Solve statistical problems using python library.

CO4: Analyze real world data using visualization methods.

CO5: Evaluate models for real world problems applicable in recent machine

LIST OF EXPERIMENTS

1. Write a python program to find GCD of two no.
2. Write Program to find the square root of a number by Newton's Method.
3. Write a Python program to find the exponentiation of a number.
4. Write a Python Program to find the maximum from a list of numbers.
5. Write a Python Program to perform Linear Search.
6. Write a Python Program to perform selection sort.
7. Write a Python Program to perform insertion sort.
8. Write a Python Program to perform Merge sort.
9. Write a Python program to find first n prime numbers.
10. Write a Python program for command line arguments.

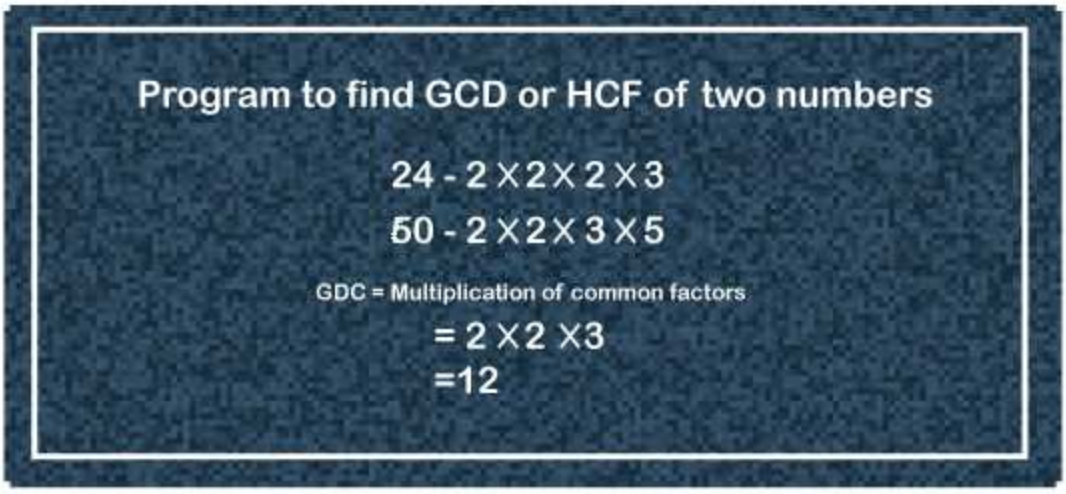
LAKSHMI NARAIN COLLEGE OF TECHNOLOGY & SCIENCE, BHOPAL

CS-506
Python

EXPERIMENT -1

AIM: Write a python program to find GCD of two no.

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 gcd() Function

In python, a gcd() is an inbuilt function offered by the math module to find the greatest common divisor of two numbers.

Syntax

gcd(a, b)

Where a and b are the two integer number passes as an argument to the function gcd().

 # create a program to print the gcd of two number in python using the math.gcd() function.

```
import math
        print(" GCD of two number 0 and 0 is ", math.gcd(0, 0)) #math.gcd(a, b), a and b are the
two integer number
print(" GCD of two number 0 and 48 is ", math.gcd(0, 48))
```

```
a = 60 # assign the number to variable a
b = 48 # assign the number to variable b
print(" GCD of two number 60 and 48 is ", math.gcd(a, b)) # pass the variable a and b to math.gc
d() function.
print(" GCD of two number 48 and -12 is ", math.gcd(48, -12)) # pass the integer number
print(" GCD of two number -24 and -18 is ", math.gcd(-24, -18))
print(" GCD of two number -60 and 48 is ", math.gcd(-60, 48))
```

In the above example, math.gcd() function generates the GCD of two given numbers. In the gcd() function, a and b pass as an argument that returns the greatest common divisor of two integer numbers, completely dividing the numbers.

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.

Pseudo Code of the Algorithm

Step 1: Take two inputs, x and y, from the user.

Step 2: Pass the input number as an argument to a recursive function.

Step 3: If the second number is equal to zero (0), it returns the first number.

Step 4: Else it recursively calls the function with the second number as an argument until it gets the remainder, which divides the second number by the first number.

Step 5: Call or assign the gcd_fun() to a variable.

Step 6: Display the GCD of two numbers.

Step 7: Exit from the program.

 # write a program to understand the GCD of two number in python using the recursion.

```
1.  def gcd_fun (x, y):
2.    if (y == 0): # it divide every number
3.      return x  # return x
4.    else:
5.      return gcd_fun (y, x % y)
6.  x =int (input ("Enter the first number: ")) # take first no.
7.  y =int (input ("Enter the second number: ")) # take second no.
8.  num = gcd_fun(x, y) # call the gcd_fun() to find the result
9.  print("GCD of two number is: ")
10. print(num) # call num
```

Output:

```
C:\Users\AMIT YADAV\C Program>python -u "c:\Users\AMIT YADAV\C Program\gcdRecur.py"
Enter the first number: 54
Enter the second number: 24
GCD of two number is:
6
```

As we can see in the above program, we take two values as input and pass these numbers to the GCD_Loop () function to return a GCD.

GCD Using Euclid's algorithm or Euclidean Algorithm

Euclid's algorithm is an efficient method to find the greatest common divisor of two numbers. It is the oldest algorithm that divides the greater number into smaller ones and takes the remainder. Again, it divides the smaller number from the remainder, and this algorithm continuously divides the number until the remainder becomes 0.

For example, suppose we want to calculate the H.C.F of two numbers, 60 and 48. Then we divide the 60 by 48; it returns the remainder 12. Now we again divide the number 24 by 12, and then it returns the remainder 0. So, in this way, we get the H.C.F is 12.

Pseudo Code of the Euclid Algorithm

Step 1: There are two integer numbers, such as a and b.

Step 2: if a = 0, then the GCD(a, b) is b.

Step 3: if b = 0, the GCD(a, b) is a.

Step 4: a mod b find the

Step 5: Suppose a = b and b = R

Step 6: Repeat steps 4 and 3 until a mod b is equal or greater than 0.

Step 7: GCD = b and then print the result.

Step 8: Stop the program.

Let's find the H.C.F or GCD of two numbers using Euclid's algorithm in python.

Euclid.py

```
# Create a program to find the GCD of two number in python using the Euclid's Algorithm.
def find_hcf(a,b):
 while(b):
 a, a = b, a % b
 return a
 a = int(input (" Enter the first number: ") ) # take first no.
 b = int(input (" Enter the second number: ")) # take second no.
 num = find_hcf(a, b) # call the find_hcf() to get the result
 print("  The HCF of two number a and b is ")
 print(num) # call num
```

```
C:\Users\AMIT YADAV\C Program>python -u "c:\Users\AMIT YADAV\C Program\Euclid.py"
 Enter the first number: 144
 Enter the second number: 96
  The HCF of two number a and b is
48
```

EXPERIMENT -2

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

Loop:

While loop:

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

Syntax:

The syntax of a while loop in Python programming language is −

while(condition)

{

statement(s)

}while expression:

statements(s)

Here, statement(s) may be a single statement or a block of statements. The condition may be any expression, and true is any non-zero value. The loop iterates while the condition is true.

When the condition becomes false, program control passes to the line immediately following the loop.

Keywords used

Return:

- The python return statement is used in a function to return something to the caller program.
- We can use the return statement inside a function only.
- In Python, every function returns something. If there are no return statements, then it returns None.
- If the return statement contains an expression, it is evaluated first and then the value is returned.
- The return statement terminates the function execution.
- A function can have multiple return statements. When any of them is executed, the function terminates.
- A function can return multiple types of values.
- Python function can return multiple values in a single return statement.

Newton's Method:

0.5*(approx+n/approx) is the Newton method to find the square root of the number.

CODE:

```
def newtonsqrt(n):
    approx=0.5*n
    better=0.5*(approx+n/approx)
    while better!=approx:
        approx=better
        better=0.5*(approx+n/approx)
    return approx
print(newtonsqrt(64))
```

EXPERIMENT -3

AIM: Write a Python program to find the exponentiation of a number.

Method 1:

Code:

```
num=int(input("Enter number: "))
exp=int(input("Enter exponential value: "))
result=1
for i in range(1,exp+1):
result=result*num
print("Result is:",result)
```

Output:

Enter number: 2
Enter exponential value: 10
Result is: 1024

Method 2 using exponential operator (**)

```
num=int(input("Enter number: "))

exp=int(input("Enter exponential value: "))

result=num**exp

print("Result is:",result)
```

Output:

Enter number: 8
Enter exponential value: 3
Result is: 512

Method 3 using in-built method

```
import math

num=int(input("Enter number: "))

exp=int(input("Enter exponential value: "))

result=math.pow(num,exp)

print("Result is:",result)
```

Output:

Enter number: 3
Enter exponential value: 5
Result is: 243.0

EXPERIMENT NO.4

AIM: Write a Python Program to find the maximum from a list of numbers.

List is one of the most commonly used data structures provided by python. List is a data structure in python that is mutable and has an ordered sequence of elements. Following is a list of integer values.

Example

Following is a list of integer values

```
lis= [1,2,3,4,5]
print(lis)
```

Output

If you execute the above snippet, it produces the following output.

[1, 2, 3, 4, 5]

Using the max() function

In this method, we use the **max()** function to find the maximum valued element. The **max()** function returns an element from the list with the maximum value.

Example

The following is an example code for finding the maximum element of a list using the **max()** function.

```
List 1 = [1, 0, 4, 5, 100]
print("Largest element of the list is:", max(list1))
```

Output

```
Largest element of the list is: 100
```

Using the sort() method

Here we will find the maximum element of a list using the sort() method. The sort() method which is provided by python is used to sort the elements of the list in ascending or descending order. Default, it will sort in ascending order. After sorting the largest element is at the last position of the list, so we print that element.

Example 1

```
list1 = [100, 1, 5, 80, 20]
list1.sort()
print("Largest element of the list is:", list1[ 1])
```

Output

If you execute the above snippet, it produces the following output

```
Largest element of the list is: 5
```

Example 2

Following is another example to find the element from a Python list with a maximum value −

```
list = [54, 57, 827, 74, 91, 74, 62]
the_list.sort()
maximum_element = the_list [-1]
print("The Maximum element of the given list is: ", maximum_element)
```

Output

Following is an output of the above code −

The Maximum element of the given list is: 827

Using for loop or without using the built in function

In this method, we will not use built-in functions, rather we use a loop to find the largest element in a list.

Here, initially assume the first element as the maximum and then we iterate over a for loop where we compare with other elements in the list. While comparing with elements in the list we change the max variable if the element is greater than the compared maximum element. Finally, after getting terminated from the loop we get the maximum element.

Example

The following is an example code to get the maximum element from a list.

```
def MaxElement(lst):
   max = lst[0]
   for ele in lst:
      if ele > max :
         max = ele
      return max
lst = [100, 1, 5, 80, 20]
print("Largest element of the list is:", MaxElement(lst))
```

Output:

If you execute the above snippet, it produces the following output.

Largest element of the list is: 100

EXPERIMENT NO.5

AIM: Write a Python Program to perform Linear Search.

Linear Search in Python:

Python is one of the most popular and powerful languages. It takes a few lines to execute the code, which makes it much user-friendly language. In this tutorial, we will learn the linear search in Python. Searching is a technique to find the particular element is present or not in the given list.

There are two types of searching -

- Linear Search
- Binary Search

Both techniques are widely used to search an element in the given list.

Linear Search

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.

Linear Search Algorithm

There is list of n elements and key value to be searched.

Below is the linear search algorithm.

```
LinearSearch(list, key)
  for each item in the list
    if item == value
      return its index position
   return -1
```

Python Program

```
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)
```

Output:

Element found at index 4

Explanation:

In the above code, we have created a function **linear Search (),** which takes three arguments - list1, length of the list, and number to search. We defined for loop and iterate each element and compare to the key value. If element is found, return the index else return -1 which means element is not present in the list.

EXPERIMENT NO.6

AIM: Write a Python Program to perform selection sort.

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.

Working of Selection Sort

The following are the steps to explain the working of the Selection sort in Python.

Let's take an unsorted array to apply the selection sort algorithm.

[30, 10, 12, 8, 15, 1]

Step - 1: Get the length of the array.

length = len(array) → 6

Step - 2: First, we set the first element as minimum element.

Step - 3: Now compare the minimum with the second element. If the second element is smaller than the first, we assign it as a minimum.

Again we compare the second element to the third and if the third element is smaller than second, assign it as minimum. This process goes on until we find the last element.

Step - 4: After each iteration, minimum element is swapped in front of the unsorted array.

Step - 5: The second to third steps are repeated until we get the sorted array.

Selection Sort Algorithm

The selection sort algorithm as follows.

```
selection_sort(array)
  repeat (0, length - 1) times
  set the first unsorted element as the minimum
  for each of the unsorted elements
    if element < currentMinimum
      set element as new minimum
  swap minimum with first unsorted position
end selection_sort
```

Selection Sort Program using Python

The following code snippet shows the selection sort algorithm implementation using Python.

Code :

```
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))
```

Output:

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

EXPERIMENT NO.7

AIM: Write a Python Program to perform insertion sort.

Insertion Sort:

Insertion sort operates in a similar way to how we sort cards in a card game. We choose an unsorted card since we think the first card is already sorted. If the unsorted card is larger than the one in hand, it goes to the right; otherwise, it goes to the left. Similarly, other unsorted cards are taken and placed in their proper order. Insertion sort takes a similar method.

1. The array's initial element is considered to be sorted. Take the second ingredient and put it in a different key.
2. Compare the first element to the key. If the first element is bigger than the key, the first element is put in front of the key.
3. If the element isn't discovered, the return value isn't present.
4. Examine the third element in relation to the ones to its left.
5. It was placed just beneath the element that was smaller than it. Place it at the start of the array if there are no elements smaller than it.
6. Similarly, put each unsorted element in its proper location.

Algorithm of Insertion Sort:

Algorithm followed by code for better understanding:

Create a function insetion_sort()

Declare a list.

Mark the first element as sorted

Initialize for loop

Iterate each element and extract the locations.

Move sorted element right by 1

break the loop and insert the key here

Print the sorted Array

Python Program of Insertion Sort

```
def insertion_sort(arr):

for i in range(1, len(arr)):

key = arr[i]

 j = i - 1

while j >= 0 and key < a

 arr[j + 1] = arr[j]

 j = j – 1

# Place key at after the element just smaller than it.

 arr[j + 1] = key

arr = [9, 8, 6, 7, 1]

print("Unsorted Array:", arr)

insertion_sort(arr)

print('Sorted Array: ', arr)
```

OutPut

```
Unsorted Array: [9, 8,    6,    7,    1]

Sorted Array:  [1,  6,    7,    8,    9]
```

EXPERIMENT NO.8

CS-506

Python

AIM: Write a Python Program to perform Merge sort.

Merge Sort in Python is a popular and efficient sorting algorithm that works on the concept of divide and conquer. This technique involves dividing a problem into multiple sub-problems. Each sub-problem is then solved individually. Finally, sub-problems are combined to form the final solution.

In merge sort, the given array is divided into roughly two equal sub-arrays. These sub-arrays are divided over and over again into halves until we end up with arrays having only one element each. At last, we merge the sorted pairs of sub-arrays into a final sorted array.

Python Programs for Merge Sort

In this section, we will implement a merge sort algorithm in python on two types of collections - integer element's array and custom objects.

Sorting Array The merge sort algorithm divides the given array into roughly two equal halves and sorts them recursively. This process is continued until arrays have only one element left.

The First step is to create copies of arrays. The first array contains the elements from ``[start_index,middle]and the second from[middle+1,end_index]`.

In second step, we traverse both copies of the array with the help of pointers and select the smaller value of the two values and add them to the sorted array.

The Last step is, if we run out of elements in any array, we select the remaining elements and place them in the sorted array.

Implementation merge sort algorithm in Python.

```
def mergeSort(arr):

    if len(arr) > 1:



        # Create sub_array2 ← A[start..mid] and sub_array2 ← A[mid+1..end]

        mid = len(arr)//2
```

```
    sub_array1 = arr[:mid]

    sub_array2 = arr[mid:]


    # Sort the two halves

    mergeSort(sub_array1)

    mergeSort(sub_array2)


    # Initial values for pointers that we use to keep track of where we are in each array

    i = j = k = 0


# Until we reach the end of either start or end, pick larger among

# elements start and end and place them in the correct position in the sorted array

    while i < len(sub_array1) and j < len(sub_array2):

        if sub_array1[i] < sub_array2[j]:

            arr[k] = sub_array1[i]

            i += 1

        else:

            arr[k] = sub_array2[j]

            j += 1

        k += 1
```

```
    # When all elements are traversed in either arr1 or arr2,

    # pick up the remaining elements and put in sorted array

        while i < len(sub_array1):

            arr[k] = sub_array1[i]

            i += 1

            k += 1



        while j < len(sub_array2):

            arr[k] = sub_array2[j]

            j += 1

            k += 1


arr = [10, 9, 2, 4, 6, 13]

mergeSort(arr)

print(arr)
```

Output

[2, 4, 6, 9, 10, 13]

EXPERIMENT NO.9

AIM: Write a Python program to find first n prime numbers.

Code:

```
# Python Program to find first n prime numbers

from math import sqrt


# To take input from the user and initialize the variables

num = int(input("Enter a number: "))

count = 0

n = 2


print("First", num, "prime numbers are: ")

while count < num:

    # define a flag variable

    prime_flag = True


    for i in range(2, int(sqrt(n)) + 1):

        if (n % i) == 0:

            prime_flag = False

            break
```

```
# check if flag is True

if prime_flag:

    print(n, end =" ")

    count = count + 1

n = n + 1
```

EXPERIMENT NO.10

AIM: Write a Python program for command line arguments.

Python Command line arguments

The Python supports the programs that can be run on the command line, complete with command line arguments. It is the input parameter that needs to be passed to the script when executing them.

It means to interact with a command-line interface for the scripts.

It provides a **getopt** module, in which command line arguments and options can be parsed.

Argument passing

The command ls is often used to get a summary of files and folders present in a particular directory.

Argparse:
It means to communicate between the writer of a program and user which does not require going into the code and making changes to the script. It provides the ability to a user to enter into the command-line arguments.

Access command line arguments:

The Python sys module provides access to command-line arguments via sys.argv. It solves the two purposes:

Python sys module

It is a basic module that was shipped with Python distribution from the early days on. It is a similar approach as C library using argc/argv to access the arguments. The sys module implements command-line arguments in a simple list structure named sys.argv.

Each list element represents a single argument. The first one -- sys.argv[0] -- is the name of Python script. The other list elements are sys.argv[1] to sys.argv[n]- are the command line arguments 2 to n. As a delimiter between arguments, space is used. Argument values that contain space in it have to be quoted, accordingly.

It stores command-line arguments into a list; we can access it using **sys.argv**. This is very useful and a simple way to read command-line arguments as String

```
import sys
print(type(sys.argv))
print('The command line arguments are:')
for i in sys.argv:
print(i)
```

Python getopt module

The Python getopt module extends the separation of the input string by parameter validation. Based on getopt C function, it allows both short and long options, including a value assignment.

It is very similar to C getopt() function for parsing command line parameters.

It is useful in parsing command line arguments where we want the user to enter some options.

Code:

```
import getopt
import sys
argv = sys.argv[1:]
try:
    opts, args = getopt.getopt(argv, 'hm:d', ['help', 'my_file='])
    print(opts)
    print(args)
except getopt.GetoptError:
    # Print a message or do something useful
    print('Something went wrong!')
    sys.exit(2)
```

Python argparse module

It offers a command-line interface with standardized output, whereas the former two solutions leave most of the work in your hands. argparse allows verification of fixed and optional arguments with a name checking as either UNIX or GNU style. It is the preferred way to parse command-line arguments. It provides a lot of

option such as positional arguments, the default value for arguments, helps message, specifying the data type of argument etc.

It makes it easy to write the user-friendly command-line interfaces. It automatically generates help and usage messages and issues errors when a user gives invalid arguments to the program.

getopt.getopt method

This method is used for parsing the command line options and parameter list.

Syntax:

getopt.getopt(args, options, [long_options])

args- It is an argument list that needs to be parsed.

options- A string of option letters that the script wants to recognize, with options that require an argument which should be followed by a colon(:).

long_options(optional)- It must be a string with names of the long options, which should be supported.

- This method returns a value consisting of two elements, i.e. list of (**option, value**) pairs, list of program arguments left after option list was stripped.
- Each option-and-value pair are returned as an option as its first element, prefixed with a hyphen for short options (e.g.,'-x') or two hyphens for long options (e.g., '--long-option').

Exception getopt.GetoptError

This exception arises when an unrecognized option is found in the argument list or when any option requiring an argument is given none.

The argument to the exception is a string that indicates the cause of the error. The attributes **msg** and **opt** to give the error message and related option

Code

```
#!/usr/bin/python
import sys, getopt
```

```
def main(argv):
   inputfile = ''
   outputfile = ''
   try:
      opts, args = getopt.getopt(argv,"hi:o:",["ifile=","ofile="])
   except getopt.GetoptError:
      print 'test.py -i <inputfile> -o <outputfile>'
      sys.exit(2)
   for opt, arg in opts:
      if opt == '-h':
         print 'test.py -i <inputfile> -o <outputfile>'
         sys.exit()
      elif opt in ("-i", "--ifile"):
         inputfile = arg
      elif opt in ("-o", "--ofile"):
         outputfile = arg
   print 'Input file is "', inputfile
   print 'Output file is "', outputfile

if __name__ == "__main__":
   main(sys.argv[1:])
```

Output:

$ test.py –h

Usage: test.py-i<input file>- o<output file>

$ test.py –I BMP –o

Usage: test.py –i<inputfile> -o<output file>

$ test.py –i inputfile

Input file is “inputfile

Output file is ”