```
                        #MODULE-8
#A)THIS IS A PROGRAM TO PRINT SORTED LIST
 A)Write a function called is_sorted that takes a list as a parameter
 and returnsTrue if the list is sorted in ascending order
 and False otherwise. You can assume (as a precondition) that the
 elements of the list can be compared with the comparison
 operators <,>, etc.
 For example, is_sorted([1,2,2]) should
 return True and is_sorted(['b','a']) should return False.

SOLUTION:-
#THIS IS SOLUTION OF SORTED LIST

print
"==========================================================================================================
print "This is a program to check whether elements in the list are sorted or not if they are in
ascending order they are sorted "
print
"==========================================================================================================

def is_sorted():
        list=[]
        number=input("How many elements do u expect in list==>")#This loop is to create a list
        list=[]
        for j in range(number):
                element=input("Enter elements==")
                list.append(element)#This is to arrange the elements in the list manner
        print list#This prints the list
        c=0
        for i in range(len(list)-1):
                if list[i]<list[i+1]:#This checks the first and second element in the list and so on
                        c=c+1
        if c==len(list)-1:
                print "List is sorted"
        else:
                print "List is not sorted"
is_sorted()


#PROGRAM 2
#THIS IS PROGRAM TO CHECK TWO WORDS ARE ANAGRAMS OR NOT
B)Two words are anagrams if you can rearrange the letters from one
to spell the other. Write a function called is_anagram that takes
two strings and returns True if they are anagrams.

SOLUTION:-
#THIS IS SOLUTION OF PROGRAM TO CHECK TWO WORDS ARE ANAGRAMS OR NOT
def is_anagram(str1,str2):
    count=0
    if(len(str1)!=len(str2)):
        return False

    i=0
    while(i<len(str1)):
        j=0
        while(j<len(str2)):
            if(str1[i]==str2[j]):
                count=count+1
                str2.remove(str2[j])
            j=j+1
        i=i+1

    if(count==len(str1)):
        return True
    else:
        return False

str1=raw_input("Enter your first String: ")
str2=raw_input("Enter your Second String: ")
str1=list(str1)
str2=list(str2)
print is_anagram(str1,str2)
```

```
#PROGRAM 3
#THIS IS BIRTHDAY PROBLEM
C)The (so-called) Birthday Paradox:
#THIS IS A PROGRAM OF DUPLICATES
1. Write a function called has_duplicates that takes a list and
   returns True if there is any element that appears more than
   once. It should not modify the original list.
#THIS IS A PROGRAM OF BIRTHDAY PROBLEM
2. If there are 23 students in your class, what are the chances
   that two of you have the same birthday? You can estimate this
   probability by generating random samples of 23 birthdays and
   checking for matches. Hint: you can generate random
   birthdays with therandint function in the random module.

SOLUTION:-
1.
#THIS IS A SOLUTION TO PROGRAM OF DUPLICATES
def has_duplicates():
        list=[]
        n=input("number of elements do u want in list==>")
        for i in range(1,n+1):
                b=input("Enter elements==>")
                list.append(b)
        print  list
        a=len(list)
        for j in range(a):
                c=0
                for k in range(a):
                        if list[j]==list[k]:
                                c=c+1
        if c>=2:
                print True
        else:
                print False
has_duplicates()

2.
#THIS IS A SOLUTION TO BIRTHDAY PROBLEM
import random
n=23
list=[]
for i in range(n):
        birthday=random.randint(1,365)
        list.append(birthday)
b=[]
for i in range(23):
        c=0
        if list[i] not in b:
                for j in range(i+1,23):
                        if list[i]==list[j]:
                                b.append(list[i])
                                c=c+1
                if c>1:
                        print list[i],c
                else:
                        print False


#PROGRAM 4
#THIS IS A PROGRAM OF BINARY SEARCH
F) To check whether a word is in the word list, you could use
 the in operator, but it would be slow because it searches through
 the words in order.
Because the words are in alphabetical order, we can speed things
 up with a bisection search, which is similar to what you do when
 you look a word up in the dictionary. You start in the middle and
 check to see whether the word you are looking for comes before
 the word in the middle of the list. If so, then you search the first
 half of the list the same way. Otherwise you search the second
 half.
Either way, you cut the remaining search space in half. If the word
 list has 113,809 words, it will take about 17 steps to find the word
 or conclude that it’s not there.
Write a function called bisect that takes a sorted list and a target
 value and returns the index of the value in the list, if it’s there,
 or None if it’s not.
Or you could read the documentation of the bisect module and use
 that!



SOLUTION:-
#THIS IS SOLUTION TO BINARY SEARCH PROGRAM
m=input("number of elements do u want in list=====>")
list=[]
for i in range(1,m+1):
        b=input("Enter elements==>")
        list.append(b)
print list
n=input("Enter search value===>")
i=0
c=0
for i in range(len(list)):
        if list[i]==n:
                c=c+1
if c>=1:
        print "yes,the element is found"
else:
        print "No,sorry the element is not in the list"


#PROGRAM 5
#THIS IS A PROGRAM OF REVERSE PAIR
G)
 Two words are a “reverse pair” if each is the reverse of the other.
 Write a program that finds all the reverse pairs in the word list.

SOLUTION:-
#THIS IS SOLUTION TO REVERSE PAIR PROBLEM
def reversepair():
        l=[]
        n=input("no.of words do u want in the words list:")
        word=[]
        for i in range(1,n+1):
                words=raw_input("enter a word===>")
                word.append(words)
        print word
        for j in word:
                a=j
                s=j[::-1]
                for k in range(len(word)):
                        if word[k]==s:
                                l.append(s)
        print l
reversepair()

#PROGRAM 6
#THIS IS A PROGRAM OF INTERLOCKING TWO STRINGS
H)
Two words “interlock” if taking alternating letters from each
forms a new word1. For example, “shoe” and “cold” interlock to
form “schooled.”


SOLUTION:-
#THIS IS A SOLUTION TO INTERLOCK PROBLEM
#THIS IS METHOD 1
#METHOD 1:-
#This is a program to interlock two strings
print "This is a program to Interlock two strings"
#This is a program to interlock two strings and give the result
```

```
def interlock():#Function
        str1=raw_input("Enter first string:")#To enter a string
        str2=raw_input("Enter second string:")
        word=""
        for k in range(len(str1)):
                for l in range(len(str2)):#Inner loop
                        if k==l:
                                word=word+str1[k]+str2[l]
                if k>len(str2)-1:
                        word=word+str1[k]
        if len(str1)<len(str2):
                word=word+str2[len(str1):]


        print "The Result interlocked word is ",word#Result
interlock()


#INTERLOCK PROBLEM IN METHOD 2
#METHOD 2:-
def has_InterlockPair(l,str):
    a=[]
    for i in range(len(l)):
        if str==l[i]:
                a.append(l[i])
    return a

list=["abc","cde","efg","acbdce","cedfeg"]
c=0
for i in range(len(list)-1):
        s1=list[i]
        for k in range(i+1,len(list)):
            s2=list[k]
            #checking lengths of two elements to avoid array out of bounds error
            if len(s1)<len(s2):
                length=len(s1)
            else:
                length=len(s2)
            str=""
            #creating interlock pairs
            for j in range(length):
                #interlock pair
                str=str+s1[j]+s2[j]

            #adding reamining elements of string
            if len(s1)<len(s2):
                str=str+s2[j+1:]
            else:
                str=str+s1[j+1:]
            #print final interlock pair
            #print str
            for i in range(len(list)):
                if str==list[i]:
                    print "Interlock pairs::",s1,"and",s2,"=",list[i]



                        #MODULE 9
#PROGRAM 7:-
#THIS IS A PROGRAM OF ROTATE PAIRS
A)
Two words are “rotate pairs” if you can rotate one of them and get
the other
Write a program that reads a wordlist and finds all the rotate
pairs.


SOLUTION:-
#THIS IS SOLUTION TO ROTATE PAIRS
#This is a program to print rotate pairs
def rotate_word(s,a):
    newstr = ""
    for i in s:
        value = ord(i)+a
        if value<ord('a')and i.islower()or value<ord('A')and i.isupper():
            newstr = newstr+chr(value+26)
        elif value>ord('z')and i.islower() or value>ord('Z')and i.isupper():
            newstr = newstr+chr(value-26)
        else:
            newstr = newstr+chr(value)

    return newstr

list=['sid','dis','ukf']
p=[]
a=input("Enter the place do u want to rotate :")
for j in list:
        u=rotate_word(j,a)
        p.append(u)
print "Givenlist",list
print "The rotated list:",p
e=[]
for k in range(len(list)):
        for r in range(len(p)):
                if list[k]==p[r]:
                        e.append(list[k])
print "The result which contains rotate pairs:",e


#PROGRAM 8:-
#THIS IS A PROGRAM OF CAR TALK PUZZLER
B)
Here’s another Puzzler from Car Talk:
This was sent in by a fellow named Dan O’Leary. He came
upon a common one-syllable, five-letter word recently that
has the following unique property. When you remove the first
letter, the remaining letters form a homophone of the original
word, that is a word that sounds exactly the same. Replace
the first letter, that is, put it back and remove the second
letter and the result is yet another homophone of the original
word. And the question is, what’s the word?
Now I’m going to give you an example that doesn’t work. Let’s
look at the five-letter word, ‘wrack.’ W-R-A-C-K, you know like
to ‘wrack with pain.’ If I remove the first letter, I am left with a
four-letter word, ’R-A-C-K.’ As in, ‘Holy cow, did you see the
rack on that buck! It must have been a nine-pointer!’ It’s a
perfect homophone. If you put the ‘w’ back, and remove the
‘r,’ instead, you’re left with the word, ‘wack,’ which is a real
word, it’s just not a homophone of the other two words.
But there is, however, at least one word that Dan and we
know of, which will yield two homophones if you remove
either of the first two letters to make two, new four-letter
words. The question is, what’s the word?
You can use the dictionary from Exercise to check whether a string
is in the word list.

function namedread_dictionary that reads the pronouncing
dictionary and returns a Python dictionary that maps from each
word to a string that describes its primary pronunciation.
Write a program that lists all the words that solve the Puzzler.


SOLUTION:-
#THIS IS SOLUTION TO CARTALK PUZZLER
#This function is to store values inot dictionary from words.txt file
def read_file():
    fin=open("words.txt","r")
    for line in fin:
        #split string and store into word(which is of type list)
        word=line.strip().split()
        #Divide key from list
        key=word[0]
        #add rest of values to pron(pronuncation) and join as string
        pron=' '.join(word[1:])
        #store values into dictioanry
        dic[key]=pron
    fin.close()


 #The following function checks whether the given key is existed in the dictionary
def check_key(getkey):
    #if key exists returns TRUE
    if getkey in dic:
        return True


#the program execution starts form here
dic={} # or dict()
#function
read_file()
#it is just to count final values not important
count=1

#here is the logic to solve puzzle
print "list of all words that satisfies given puzzler"
#iterate over dict items
for key,value in  dic.items():
    #consider if key of dictionary length exceeds more than 3 characters
    if len(key)>3:
        #puzzle:remove first letter from original word and store into word1
        word1=key[1:]
        #puzzle: put it back the first letter then remove second letter
        word2=key[0]+key[2:]
        #key is the original word

        # compare whether word1 and word2 existed in the dictionary by calling check_key
function
        if check_key(word1) and check_key(word2):
            #value of word1 or pronunciation of word1
            pron_word1=dic[word1]
            #value of word2 or pronunciation of word2
            pron_word2=dic[word2]

            # compare original word, word1, word2 pronunciation and print final values
            if dic[key]==pron_word1 and dic[key]==pron_word2:
                print "<---------->",count,"<------------->"
                print key,":",value
                print word1,":",dic[word1]
                print word2,":",dic[word2]
                print ""
                count=count+1

print "finsihed"


                         #MODULE 10
#PROGRAM 9:-
A)
#THIS IS A PROGRAM TO PRINT FREQUENCY
Write a function called most_frequent that takes a string and
prints the letters in decreasing order of frequency. Find text
samples from several different languages and see how letter
frequency varies between languages

SOLUTION:-
#THIS IS SOLUTION OF FREQUENCY PROBLEM
def hist(s):
        d= dict()
        for c in s:
            x = d.get(c, 0)
            print x
            d[c] = x+1
        items=[(v,k)for k,v in d.items()]
        items.sort()
        items=[(v,k)for k,v in items]
        items.reverse()
        print items

hist("pppppaaassd")


#PROGRAM 10
B)
#THIS IS A PROGRAM OF ANAGRAMS IN DICTIONARY
More anagrams!
  1. Write a program that reads a word list from a file and
     prints all the sets of words that are anagrams.
      Here is an example of what the output might look like:
      ['deltas', 'desalt', 'lasted', 'salted', 'slated', 'staled']
      ['retainers', 'ternaries']
      ['generating', 'greatening']
      ['resmelts', 'smelters', 'termless']
      Hint: you might want to build a dictionary that maps from
      a set of letters to a list of words that can be spelled with
      those letters


SOLUTION:-
#THIS IS SOLUTION OF ANAGRAMS IN DICTIONARY
l=[]
f=open("words.txt","r")
for line in f:
    word = line.strip().lower()
    l.append(word)
f.close()

empty_list=[]
d={}
for i in range(len(l)):
    temp=sorted(l[i])
    temp=''.join(temp)
    t=[]
    if l[i] not in empty_list:
        for j in range(i+1,len(l)):
            key=''.join(sorted(l[j]))
            if temp==key:
                empty_list.append(l[j])
                t.append(l[j])
        d[l[i]]=t


for k,v in d.items():
    print k,":",v

#THIS IS A PROGRAM TO PRINT LARGEST SET OF ANAGRAMS
2. Modify the previous program so that it prints the largest set
   of anagrams first, followed by the second largest set, and so
   on.

SOLUTION:-
#THIS IS SOLUTION OF PROGRAM OF LARGEST SET OF ANAGRAMS
list=[]
f=open("words.txt")
for i in f:
        u=i.strip().lower()
        list.append(u)
f.close()
siddhu=[]
d={}
for j in range(len(list)):
        temp=' '.join(sorted(list[j]))
        vikas=[]
        if list[j] not in siddhu:
                for z in range(j+1,len(list)):
                        key=' '.join(sorted(list[z]))
                        if temp==key:
                                siddhu.append(list[z])
                                vikas.append(list[z])
                d[list[j]]=vikas

items=[(len(v),k) for k,v in d.items()]
items.sort()
items=[(v,k) for k,v in items]
items.reverse()
for k,v in items:
        print k,":",d[k]





#PROGRAM 11
C)
#THIS IS CARTALK PUZZLER TO PRINT LONGEST WORD IN ENGLISH BY REMOVING LETTERS
Here’s another Car Talk Puzzler:
What is the longest English word, that remains a valid English word, as
you remove its letters one at a time?
Now, letters can be removed from either end, or the middle, but you
can’t rearrange any of the letters. Every time you drop a letter, you wind
up with another English word. If you do that, you’re eventually going to
wind up with one letter and that too is going to be an English word—one
that’s found in the dictionary. I want to know what’s the longest word
and how many letters does it have?
I’m going to give you a little modest example: Sprite. Ok? You start off
with sprite, you take a letter off, one from the interior of the word, take
the r away, and we’re left with the word spite, then we take the e off the
end, we’re left with spit, we take the s off, we’re left with pit, it, and I.
Write a program to find all words that can be reduced in this way, and
then find the longest one.
This exercise is a little more challenging than most, so here are some
suggestions:
   1. You might want to write a function that takes a word and
       computes a list of all the words that can be formed by removing
       one letter. These are the “children” of the word.
   2. Recursively, a word is reducible if any of its children are reducible.
       As a base case, you can consider the empty string reducible.
   3. The wordlist I provided, words.txt, doesn’t contain single letter
       words. So you might want to add “I”, “a”, and the empty string.
   4. To improve the performance of your program, you might want to
       memoize the words that are known to be reducible.


SOLUTION:-
#THIS IS SOLUTION TO CARTALK PUZZLER TO PRINT LONGEST WORD IN ENGLISH BY REMOVING LETTERS
d={}
f=open("words.txt")
def read():
        for i in f:
                w=i.strip().split()
                key=w[0]
                p=' '.join(w[1:])
                d[key]=p

read()
max=0
for k,v in d.items():
        list=[]
        for j in range(1,len(k)):
                s=k[j:]
                list.append(s)
        c=0
        for r in list:
                if r in d:
                        c=c+1
                else:
                        break
        if c>max:
                max=c
                print k,list,":",max
```

```
#THIS IS CREATED BY SIDDHU
```