Malnad College of Engineering, Hassan

Department of Computer Science and Engineering

ALGORITHMS LABORATORY WITH PYTHON (19CS407)

Lab Manual

ALGORITHMS LABORATORY WITH PYTHON

Course Code :19CS407 L-T-P-C:0-0-1-1
Exam. Hours : 3 Hours / Week : 02
SEE: 50 Marks Total hours: 30

Course Outcomes(COs): On the completion of this laboratory course, the students will be able to**:**

COs	Statement	POs
1	Design and Implement programs using java through suitable algorithm design methods like brute force greedy method, divide and conquer, dynamic programming, backtracking, searching, sorting	PO1,PO2, PO3,PO4
2	Documentation of various algorithms and deriving their complexity	PO10

Programs to Execute in the Laboratory:

Implement the following using Python Language

1. a. Consider the runs scored by batsmen in a cricket match. Implement Recursive Binary search to find whether the specified runs have been scored by any batsman or not. Determine the time required for search.
   b. Sort a given set of elements using the Heap sort method.

2. a. Employees in an organization need to be grouped for a tournament based on their ages. Sort the ages using Merge sort and find the time required to perform the sorting.
   b. Print all the nodes reachable from a given starting node in a graph using Depth First Search method and check whether a graph is connected.

3. a. Students in a department need to be selected for a high jump competition based on their height (integer values only). Sort the heights of students using Quick sort and find the time required for the sorting.
   b. Print all the nodes reachable from a given starting node in a digraph using BFS method.

4. a. Sort a given set of elements using Insertion sort method.
   b. Obtain the topological ordering of vertices in a given digraph.

5. a. Implement Horspool algorithm for String Matching.
   b. Write a program using Transform and Conquer technique for checking whether the digits of mobile number of a person are unique.

6. a. Consider n cities. The shortest path between every pair of cities needs to be determined. Implement Floyd's algorithm for the All-Pairs- Shortest-Paths problem.
   b. Find the Binomial Co-efficient using Dynamic Programming.

7. a. There are n different routes from hostel to college. Each route incurs some cost. Find the minimum cost route to reach the college from hostel using Prim's algorithm.
   b. Compute the transitive closure of a given directed graph using Warshall's algorithm.

8. a.Find Minimum Cost Spanning Tree of a given undirected graph using Kruskal's algorithm.
   b. Implement computing a mode using pre-sorting method.

9. Consider the distance between Hassan and n different cities. Every city can be reached from Hassan directly or by using intermediate cities whichever costs less. Find the shortest distance from Hassan to other cities using Dijkstra & #39;s algorithm.

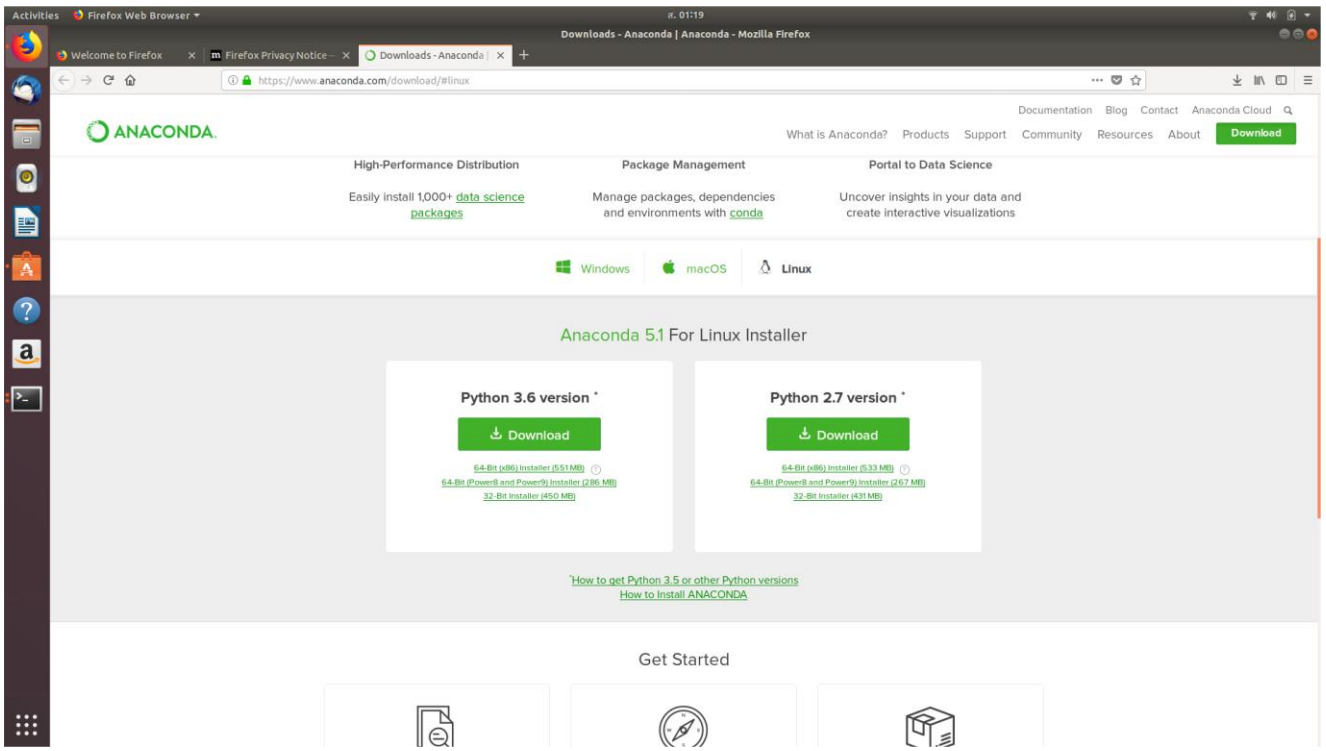
10. Implement 0/1 Knapsack problem using dynamic programming.

11. Implement N Queen's problem using Back Tracking.

12. Find the subset of given set S={s1,s2,….sn} of an positive integers whose sum is equal to a given positive integer d. A suitable message is to be displayed if the given problem instance doesn't have a solution.

Steps to install Anaconda on Ubuntu

Step 1: Go to https://www.anaconda.com/download/ and pick your package distributions (Windows, Linux, MacOS)



jitsejan@jjsvps:~$ **cd Downloads/**

jitsejan@jjsvps:~/Downloads$ **wget https://repo.continuum.io/archive/Anaconda2-4.1.1-Linux-x86_64.sh**

Step 2: Run the installer.

jitsejan@jjsvps:~/Downloads$ **bash Anaconda2-4.1.1-Linux-x86_64.sh**

Step 3: Update the terminal to include the Anaconda references.

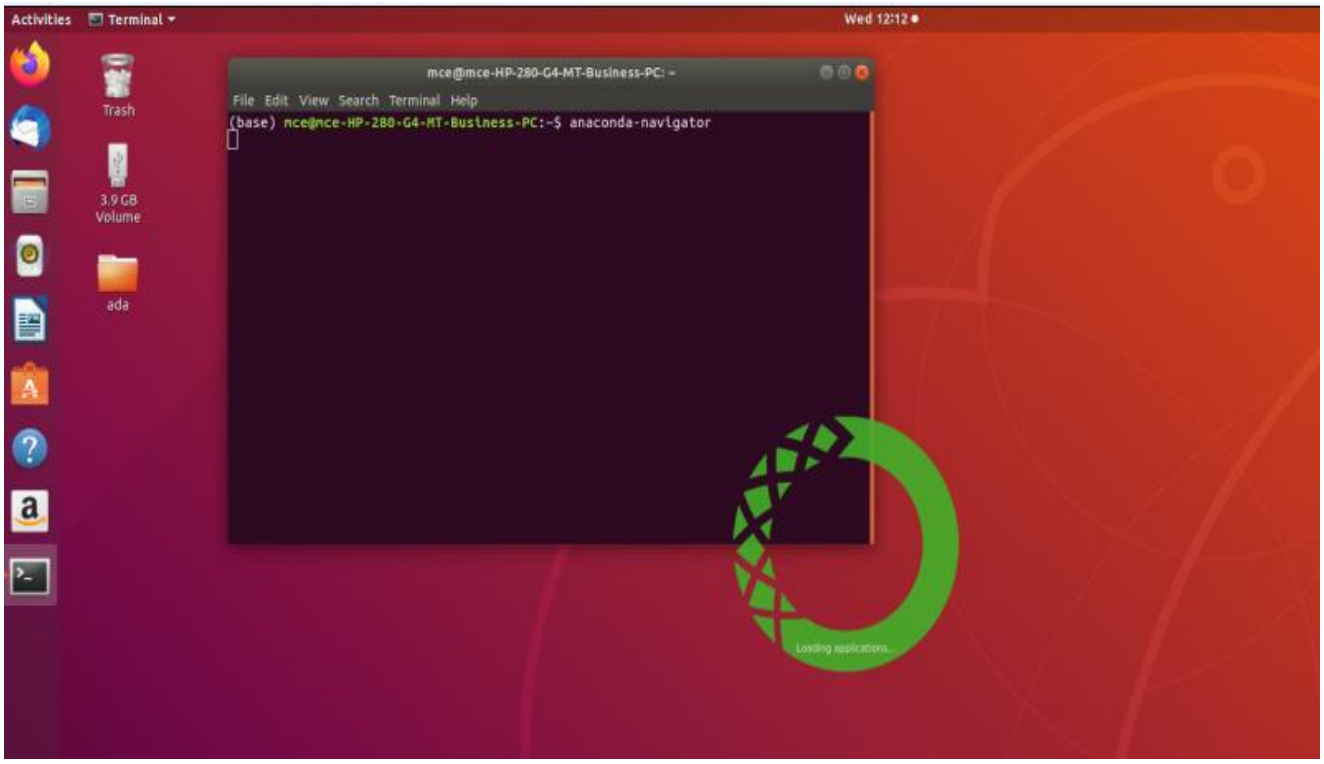
jitsejan@jjsvps:~/Downloads$ **source ~/.bashrc**

Step 4: Test if iPython is working now.

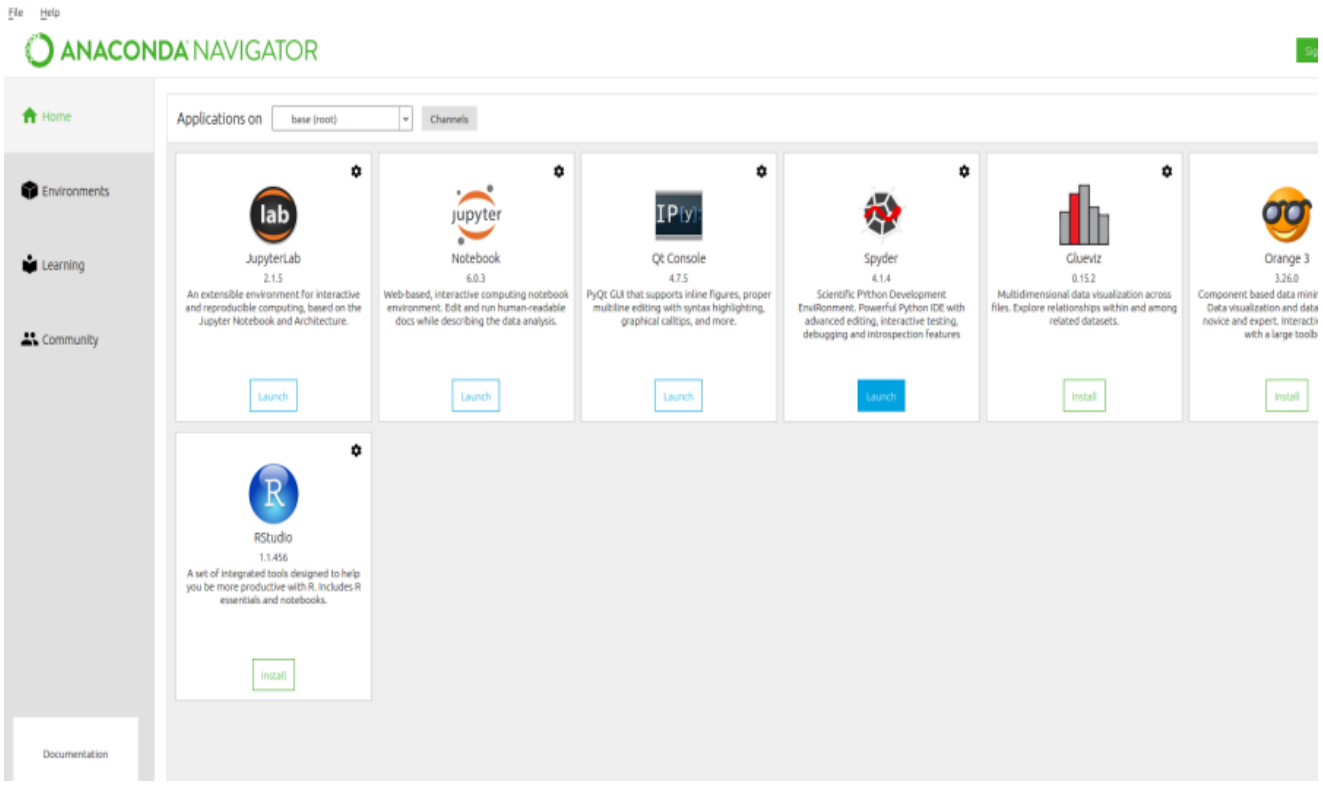
jitsejan@jjsvps:~$ **ipython -v**

All set.

After anaconda installation open terminal and run anaconda-navigator



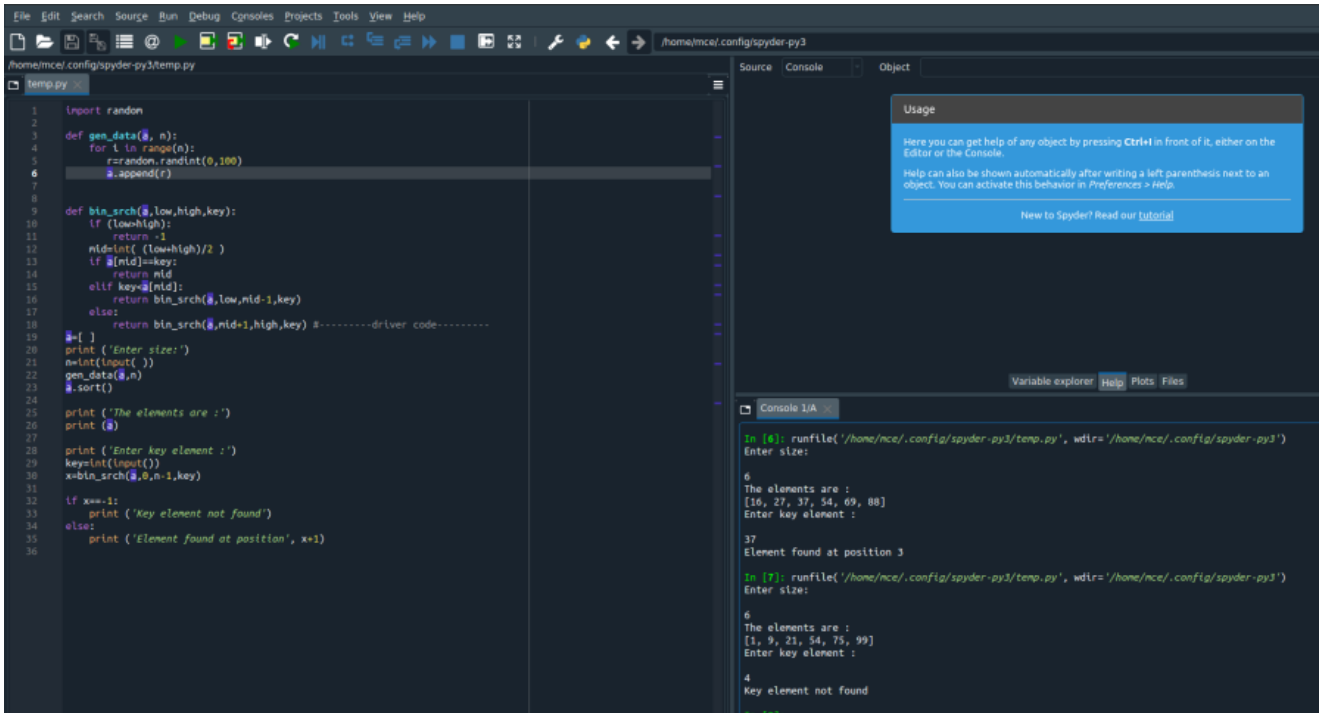
Launch spyder or jupyter to execute python program



1 a. Consider the runs scored by batsmen in a cricket match. Implement Recursive Binary search to find whether the specified runs have been scored by any batsman or not. Determine the time required for search.

```
# recursive binary search
import random

def gen_data(a, n):
    for i in range(n):
        x=randomization(0,100)
        append(x)

def bin_srch(a,low,high,key):
    if (low>high):
        return -1
    mid=int( (low+high)/2 )

    if a[mid]==key:
        return mid
    elif key<a[mid]:
        return bin_srch(a,low,mid-1,key)
    else:
        return bin_srch(a,mid+1,high,key)

#---------driver code---------

a=[ ]
print ('Enter size:')
n=int(input( ))
gen_data(a,n)
a.sort()

print ('The elements are :')
print (a)

print ('Enter key element :')
key=int(input())
x=bin_srch(a,0,n-1,key)

if x==-1:
    print ('Key element not found')
else:
    print ('Element found at position', x+1)
```

OUTPUT:



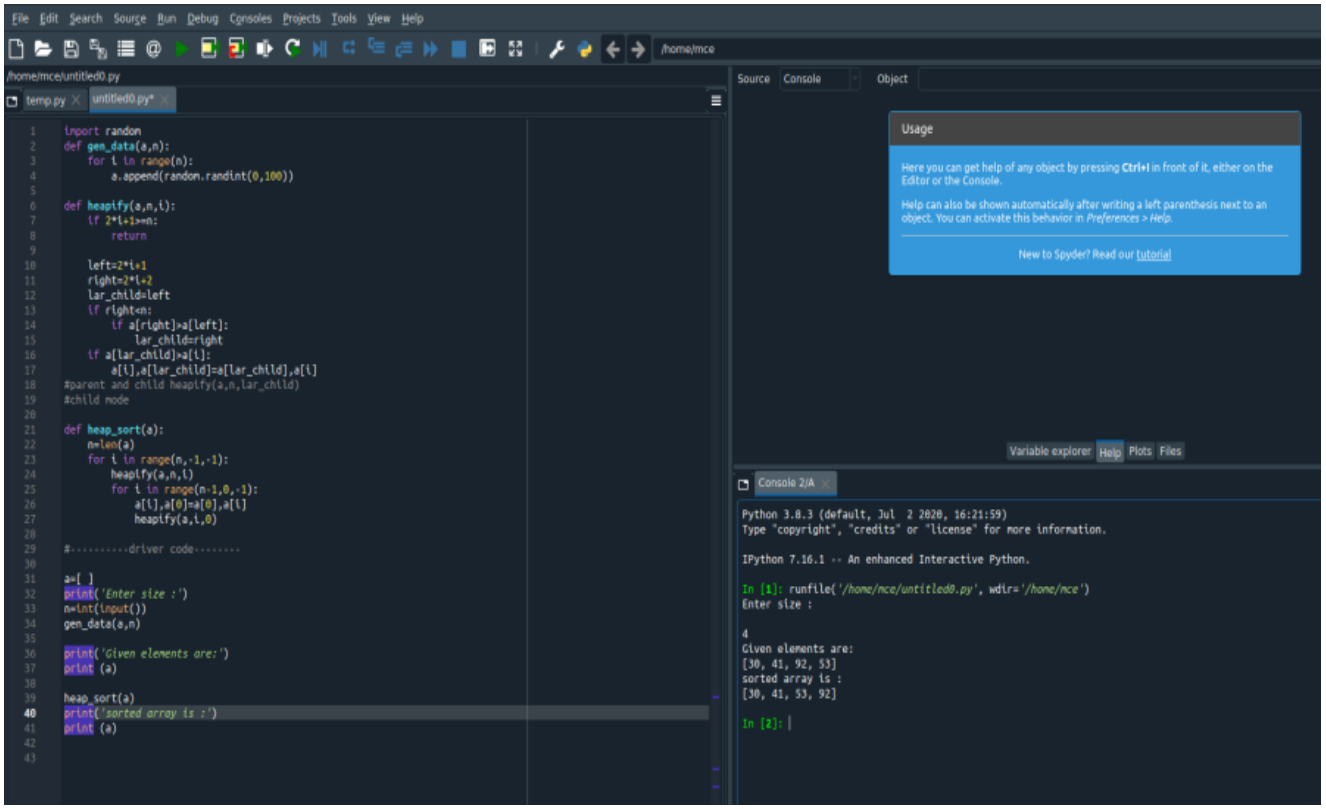
1.b

Sort a given set of elements using the Heap sort method.

```
#heap sort
import random
def gen_data(a,n):
    for i in range(n):
    a.append(random.randint(0,100))

def heapify(a,n,i):
    if 2*i+1>=n:
        return

    left=2*i+1
    right=2*i+2

    lar_child=left

    if right<n:
        if a[right]>a[left]:
            lar_child=right
    if a[lar_child]>a[i]:
        a[i],a[lar_child]=a[lar_child],a[i]
  #parent and child
        heapify(a,n,lar_child)
  #child mode

def heap_sort(a):
    n=len(a)

    for i in range(n,-1,-1):
        heapify(a,n,i)
    for i in range(n-1,0,-1):
```

```
        a[i],a[0]=a[0],a[i]
    #element
        heapify(a,i,0)

#----------driver code--------

a=[ ]
print('Enter size :')
n=int(input())
gen_data(a,n)

print('Given elements are:')
print (a)

heap_sort(a)
print('sorted array is :')
print (a)
```

OUTPUT:



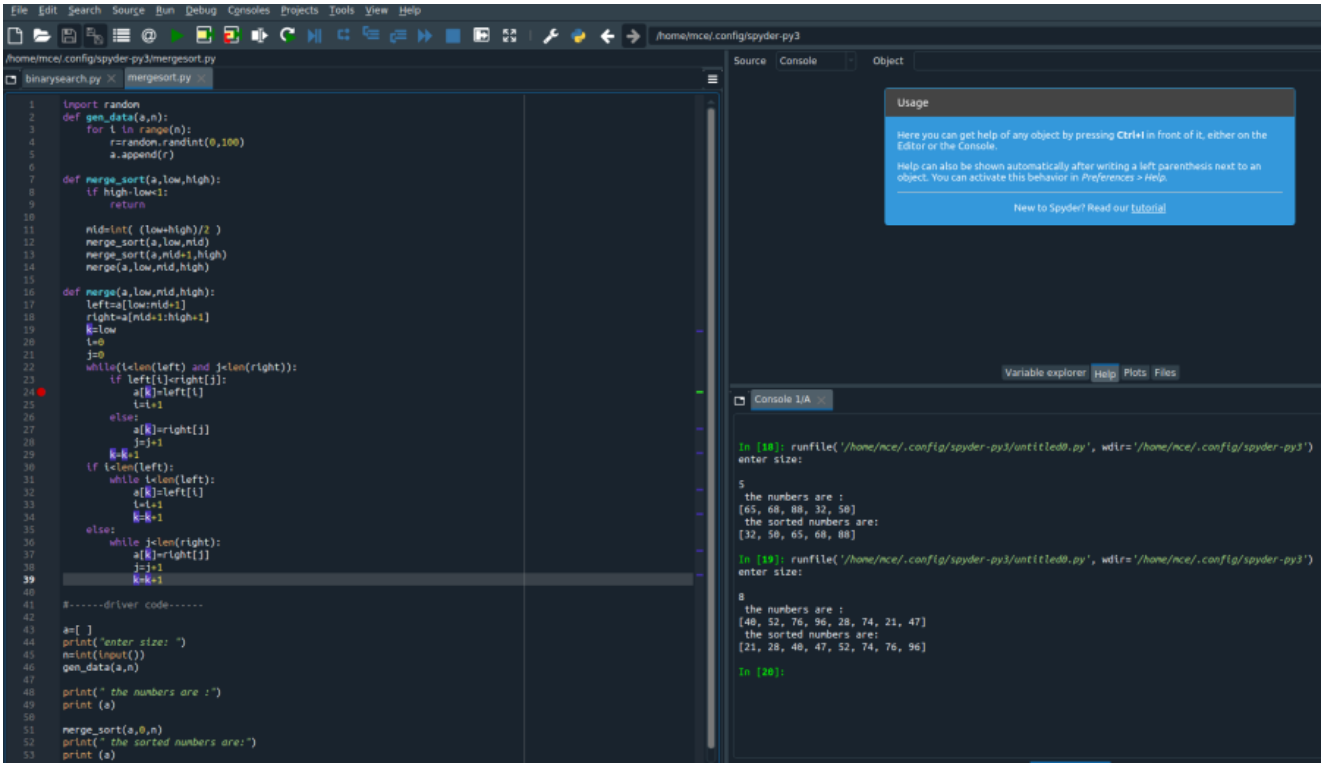
2.a

Employees in an organization need to be grouped for a tournament based on their ages. Sort the ages using Merge sort and find the time required to perform the sorting.

```
#merge sort
import random
def gen_data(a,n):
        for i in range(n):
        r=random.randint(0,100)
        a.append(r)
```

```
def merge_sort(a,low,high):
    if high-low<1:
        return

    mid=int( (low+high)/2 )
    merge_sort(a,low,mid)
    merge_sort(a,mid+1,high)
    merge(a,low,mid,high)

def merge(a,low,mid,high):
    left=a[low:mid+1]
    right=a[mid+1:high+1]
    k=low
    i=0
    j=0

    while(i<len(left) and j<len(right)):
        if left[i]<right[j]:
            a[k]=left[i]
            i=i+1
        else:
            a[k]=right[j]
            j=j+1
        k=k+1
    if i<len(left):
        while i<len(left):
            a[k]=left[i]
            i=i+1
            k=k+1
    else:
        while j<len(right):
            a[k]=right[j]
            j=j+1
            k=k+1

#------driver code------

a=[ ]
print("enter size: ")
n=int(input())
gen_data(a,n)

print(" the numbers are :")
print (a)

merge_sort(a,0,n)
print(" the sorted numbers are:")
print (a)
```

OUTPUT:

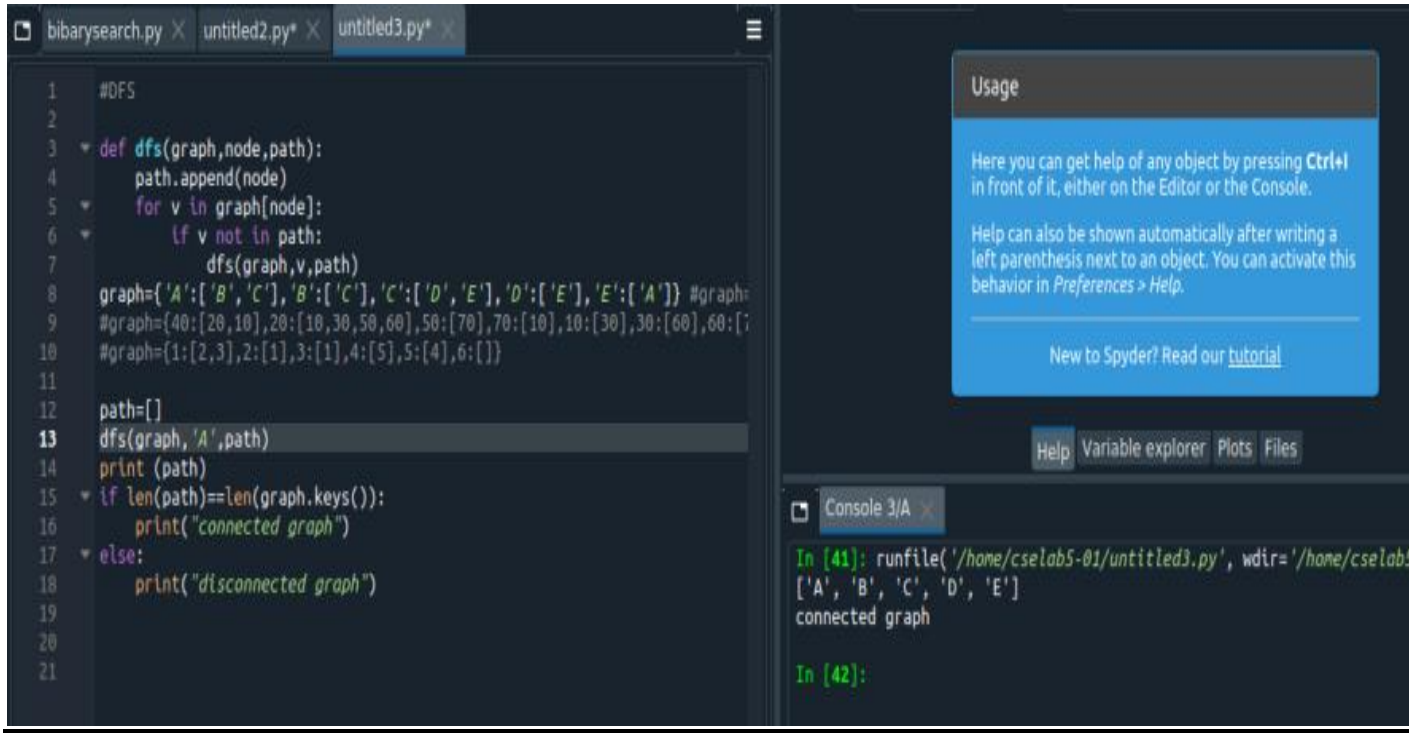


2.b

Print all the nodes reachable from the given starting node in a graph using DFS method and check whether a graph is connected or not

```
#DFS

def dfs(graph,node,path):
    path.append(node)
    for v in graph[node]:
        if v not in path:
            dfs(graph,v,path)

#-------driver code---------

graph={'A':['B','C'],'B':['C'],'C':['D','E'],'D':['E'],'E':['A']}
#graph={'A':['B','C'],'B':['E','G'],'C':['F'],'D':['A','B','C','E'],'E':['F','D'],'F':[],'G':['E','F']}
#graph={1:[2,3,4],2:[6,3,1],3:[1,2,6,5,4],4:[1,3,5],5:[3,4],6:[2,3]}
#graph={40:[20,10],20:[10,30,50,60],50:[70],70:[10],10:[30],30:[60],60:[70]}
#graph={1:[2,3],2:[1],3:[1],4:[5],5:[4],6:[]}

path=[]
dfs(graph,'A',path)
print (path)
if len(path)==len(graph.keys()):
   print("connected graph")
else:
    print("disconnected graph")
```

OUTPUT:



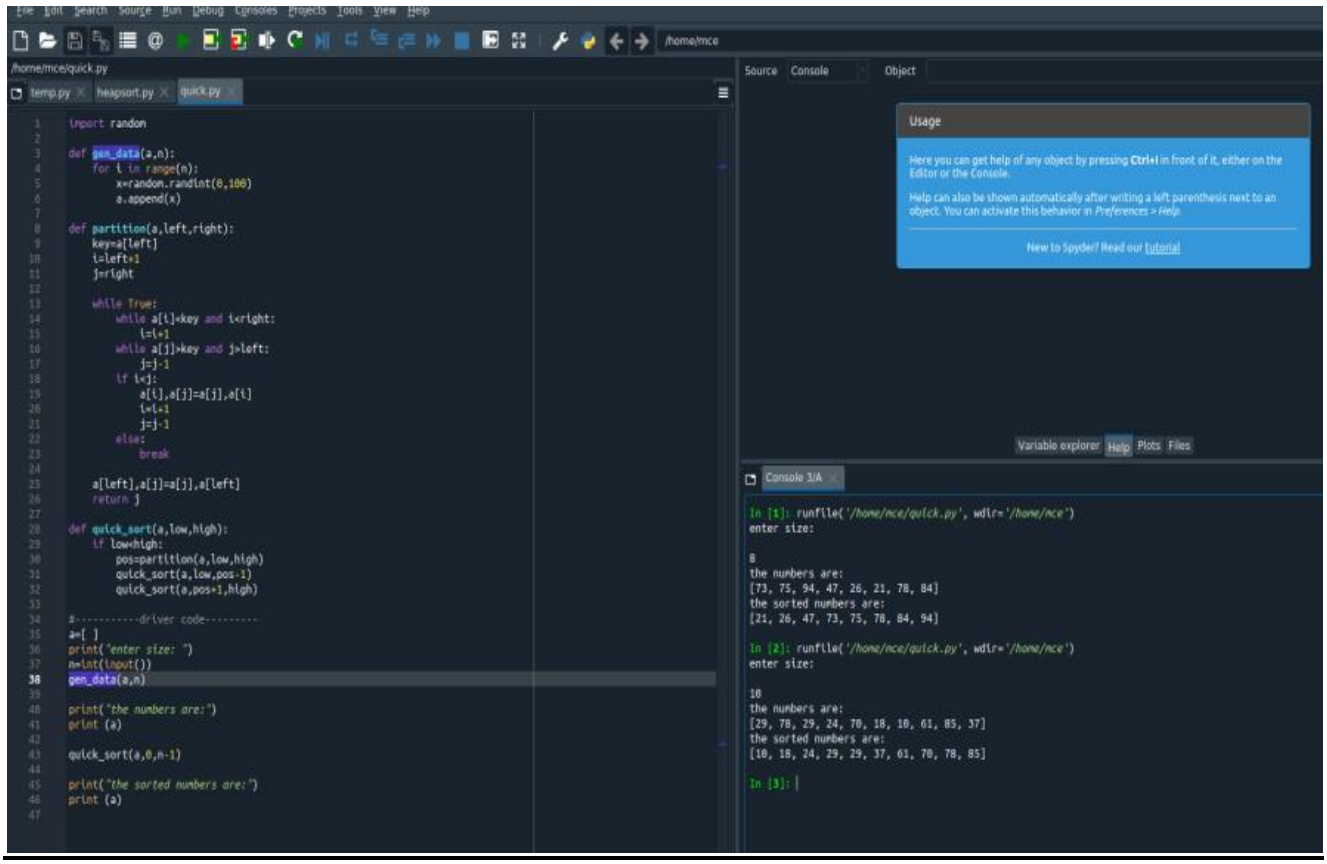
3.a

Students in a department need to be selected for a high jump competition based on their height (integer values only). Sort the heights of students using Quick sort and find the time required for the sorting.

```
#Quick sort
import random

def gen_data(a,n):
    for i in range(n):
        x=random.randint(0,100)
        a.append(x)

def partition(a,left,right):
    key=a[left]
    i=left+1
    j=right

    while True:
            while a[i]<key and i<right: i=i+1
        while a[j]>key and j>left:
            j=j-1
        if i<j:
            a[i],a[j]=a[j],a[i]
            i=i+1
            j=j-1
        else:
            break

    a[left],a[j]=a[j],a[left]
    return j
```

```
def quick_sort(a,low,high):
    if low<high:
        pos=partition(a,low,high)
        quick_sort(a,low,pos-1)
        quick_sort(a,pos+1,high)

#----------driver code---------
a=[ ]
print("enter size: ")
n=int(input())
gen_data(a,n)

print("the numbers are:")
print (a)

quick_sort(a,0,n-1)

print("the sorted numbers are:")
print (a)
```

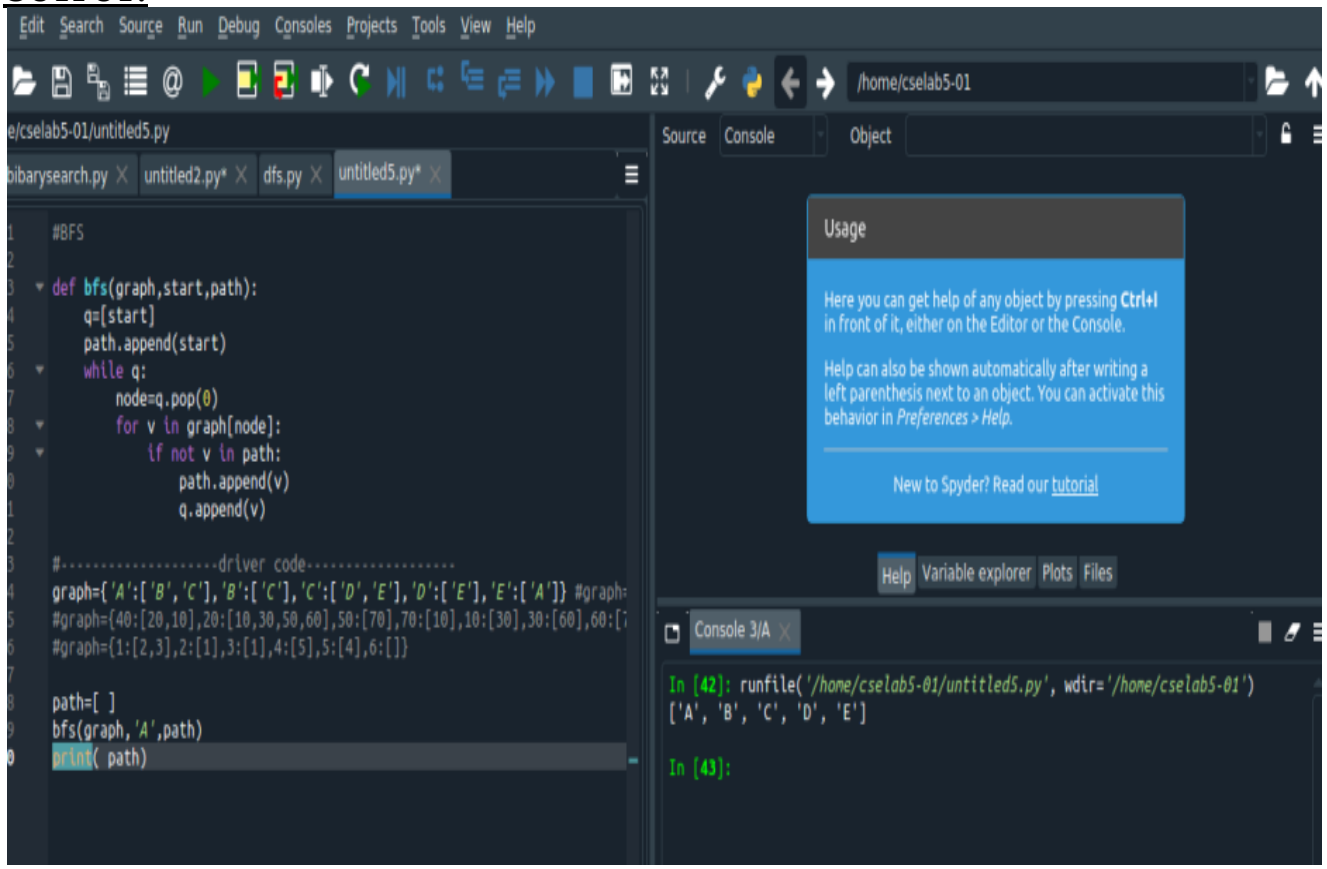
OUTPUT:



3 b. Print all the nodes reachable from the given starting node in a digraph using BFS method.

```
#BFS

def bfs(graph,start,path):
    q=[start]
    path.append(start)
    while q:
        node=q.pop(0)
        for v in graph[node]:
            if not v in path:
                path.append(v)
                q.append(v)

#--------------------driver code--------------------
graph={'A':['B','C'],'B':['C'],'C':['D','E'],'D':['E'],'E':['A']}
#graph={'A':['B','C'],'B':['E','G'],'C':['F'],'D':['A','B','C','E'],'E':['F','D'],'F':[],'G':['E','F']}
#graph={1:[2,3,4],2:[6,3,1],3:[1,2,6,5,4],4:[1,3,5],5:[3,4],6:[2,3]}
#graph={40:[20,10],20:[10,30,50,60],50:[70],70:[10],10:[30],30:[60],60:[70]}
#graph={1:[2,3],2:[1],3:[1],4:[5],5:[4],6:[]}

path=[ ]
bfs(graph,'A',path)
print( path)
```

OUTPUT:

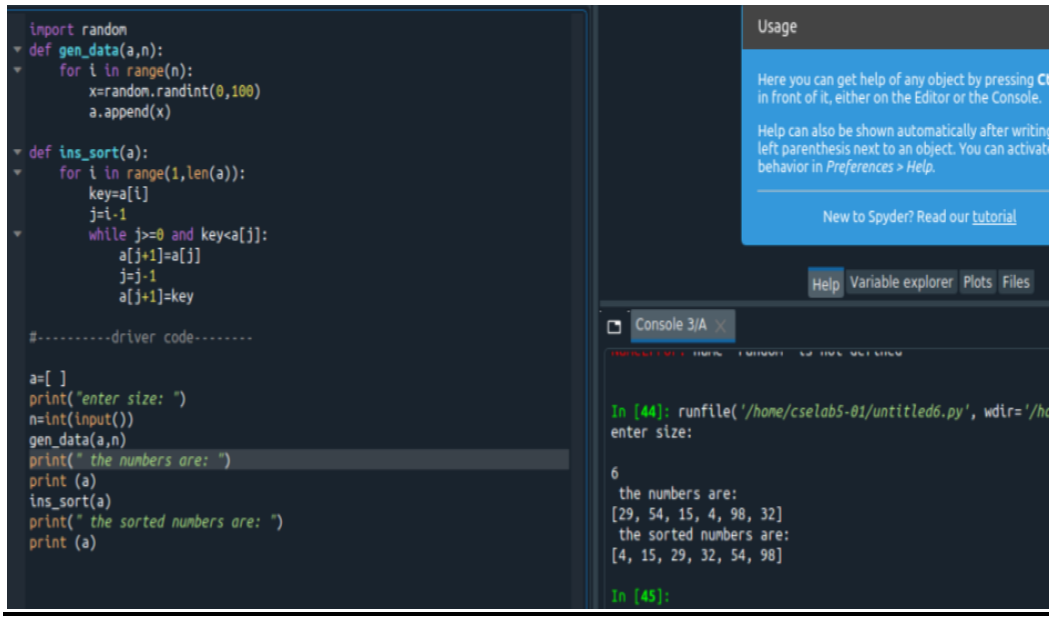


4a.

Sort the given set of elements using Insertion sort method.

```
# Insertion sort
import random

def gen_data(a,n):
    for i in range(n):
        x=random.randint(0,100)
        a.append(x)

def ins_sort(a):
    for i in range(1,len(a)):
        key=a[i]
        j=i-1
        while j>=0 and key<a[j]:
            a[j+1]=a[j]
            j=j-1
        a[j+1]=key

#----------driver code--------

a=[ ]
print("enter size: ")
n=int(input())
gen_data(a,n)

print(" the numbers are: ")
print (a)

ins_sort(a)

print(" the sorted numbers are: ")
print (a)
```

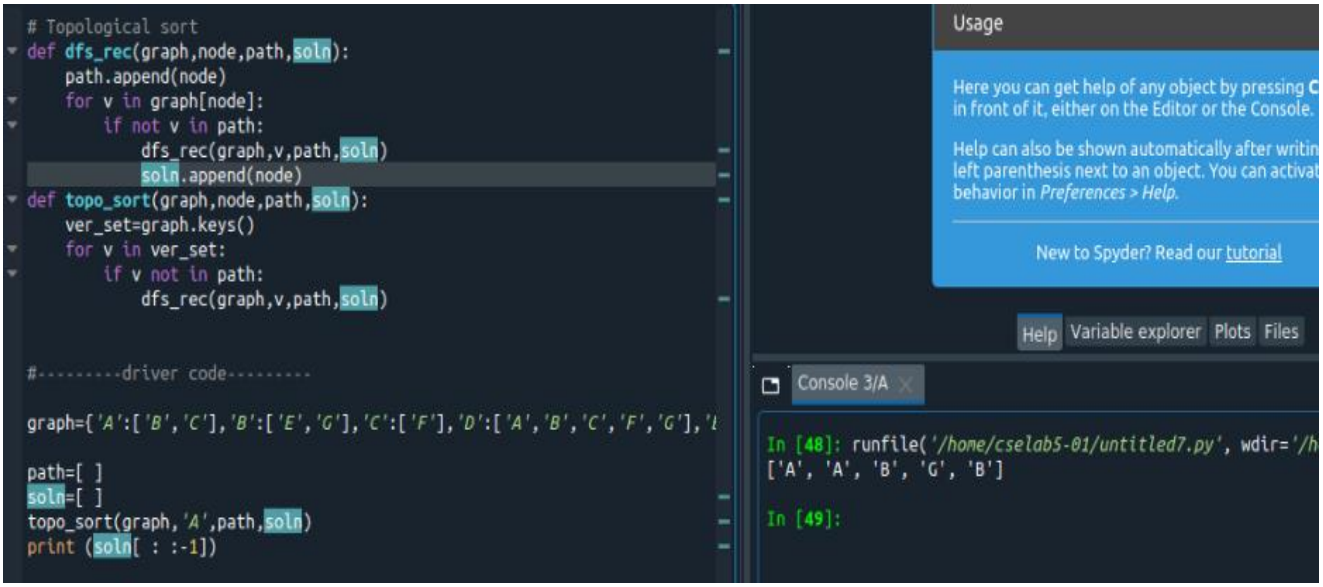
OUTPUT:



4 b. Obtain the topological ordering of vertices in a given digraph

```
# Topological sort
def dfs_rec(graph,node,path,soln):
    path.append(node)
    for v in graph[node]:
        if not v in path:
            dfs_rec(graph,v,path,soln)
    soln.append(node)

def topo_sort(graph,node,path,soln):
    ver_set=graph.keys()

    for v in ver_set:
        if v not in path:
            dfs_rec(graph,v,path,soln)


#---------driver code---------

graph={'A':['B','C'],'B':['E','G'],'C':['F'],'D':['A','B','C','F','G'],'E':[ ],'F':[ ],'G':['E','F']}

path=[ ]
soln=[ ]
topo_sort(graph,'A',path,soln)
print (soln[ : :-1])
```

OUTPUT:



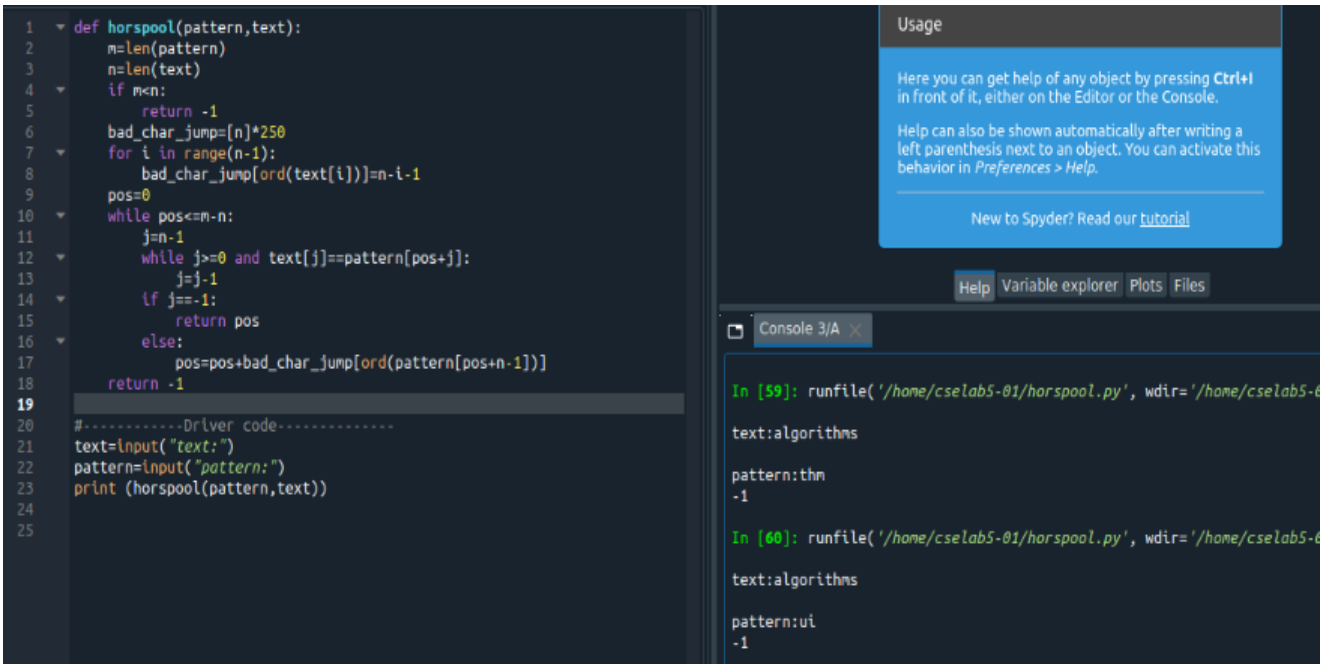
5 a.

Implement Horspool algorithm of string matching.

```
def horspool(pattern,text):
    m=len(pattern)
    n=len(text)
```

```
    if m<n:
        return -1
    bad_char_jump=[n]*250
    for i in range(n-1):
        bad_char_jump[ord(text[i])]=n-i-1
    pos=0
    while pos<=m-n:
        j=n-1
        while j>=0 and text[j]==pattern[pos+j]:
            j=j-1
        if j==-1:
            return pos
        else:
            pos=pos+bad_char_jump[ord(pattern[pos+n-1])]
    return -1

#------------Driver code--------------

pattern=raw_input("pattern:")
text=raw_input("text:")
print (horspool(pattern,text))
```

OUTPUT:

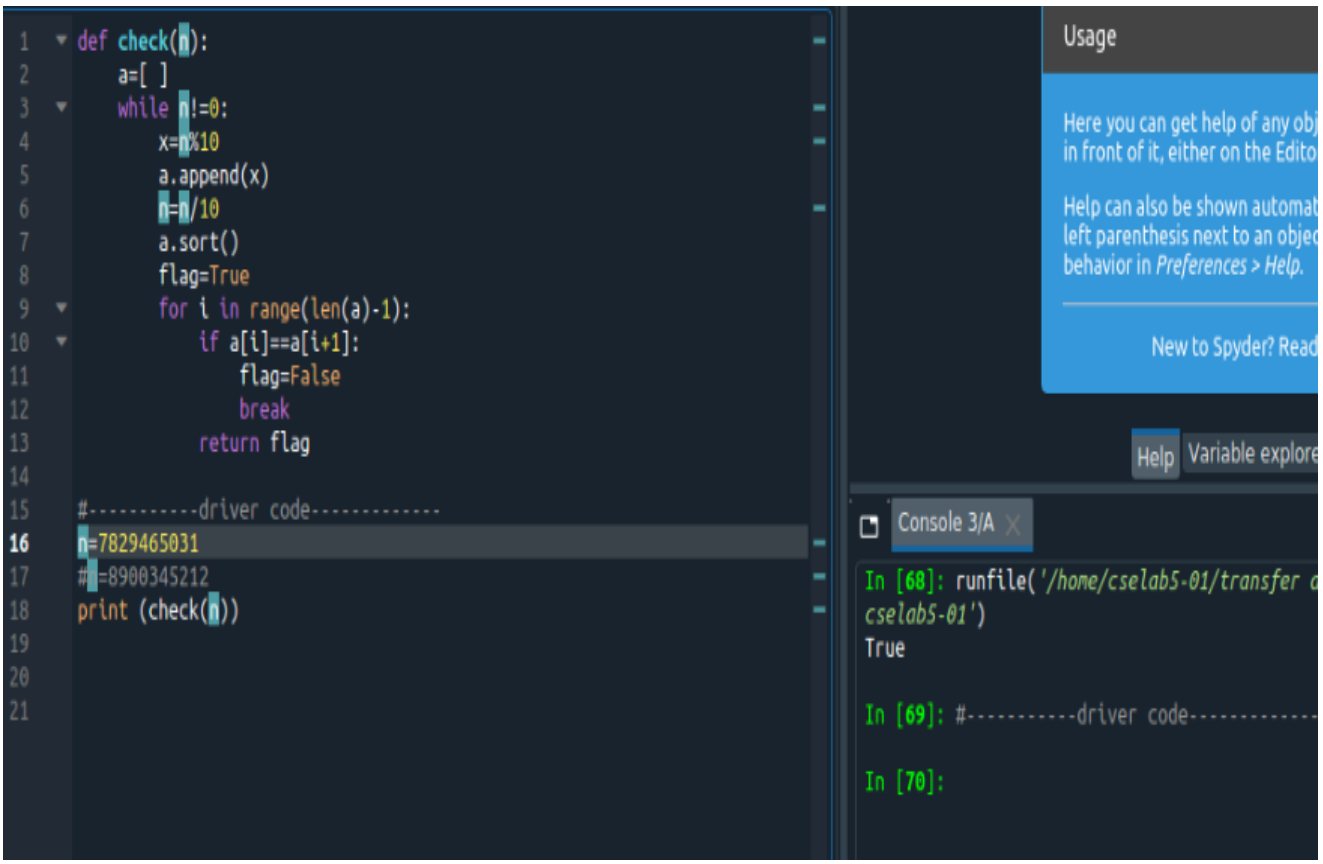


5 b. Write a programme using transfer and conquer technique for checking whether the digits of mobile number of a person are unique.

```
def check(n):
    a=[ ]
    while n!=0:
        x=n%10
        a.append(x)
        n=n/10
    a.sort()
    flag=True
    for i in range(len(a)-1):
```

```
        if a[i]==a[i+1]:
            flag=False
            break
    return flag

#----------driver code----------
n=7829465031
#n=8900345212
print (check(n))
```

OUTPUT:



6 a.

Consider n cities. The shortest path between every pair of cities needs to be determined. Implement Floyd's algorithm for the All-Pairs- Shortest-Paths problem.

```
# floyds algorithm
def all_pair_sort(dist,n):
    for k in range(n):
        for i in range(n):
            for j in range(n):
                dist[i][j]=min(dist[i][j],(dist[i][k] + dist[k][j]))

#-------------------driver code---------------------
graph=[[0,100,3,100],[2,0,100,100],[100,7,0,1],[6,100,100,0]]
n=4
all_pair_sort(graph,n)

for i in range(n):
```

print (graph[i])

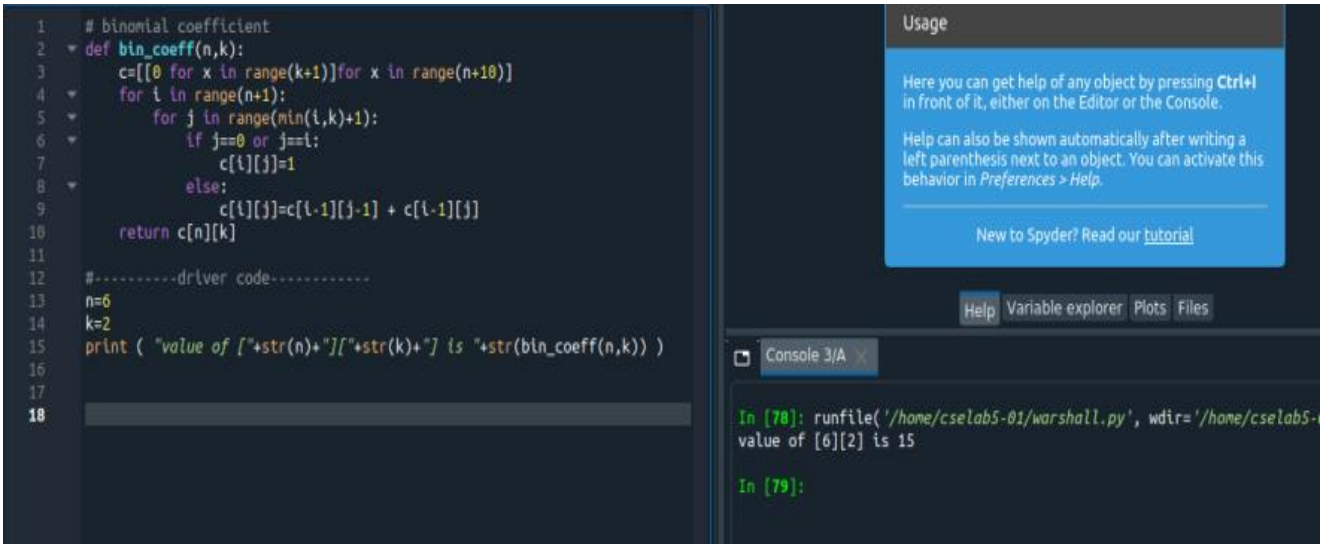
OUTPUT:

6 b.

Find Binomial co-efficient using Dynamic programing.

```
# binomial coefficient
def bin_coeff(n,k):
    c=[[0 for x in range(k+1)]for x in range(n+10)]
    for i in range(n+1):
        for j in range(min(i,k)+1):
            if j==0 or j==i:
                c[i][j]=1
            else:
                c[i][j]=c[i-1][j-1] + c[i-1][j]
    return c[n][k]

#----------driver code------------
n=6
k=2
print ( "value of ["+str(n)+"]["+str(k)+"] is "+str(bin_coeff(n,k)) )
```

OUTPUT:



7 a.

There are n different routes from hostel to college. Each route incurs some cost. Find the minimum cost route to reach the college from hostel using Prim's algorithm

```
#Prims
def min_edge(edge,v,vt):
    min=999
    for i in range(len(edge)):
        x=edge[i][0]
        y=edge[i][1]
        w=edge[i][2]

        if(x in v and y in vt)or(x in vt and y in v):
            if w<min:
                min=w
                pos=i
    return pos

def prims(edge,v):
    vt=[]
    et=[]
    vert=v.pop(0)
    vt.append(vert)
    n=len(v)
    for i in range(n):
        pos=min_edge(edge,v,vt)
        x=edge[pos][0]
        y=edge[pos][1]
        b=edge[pos]
        del edge[pos]
        et.append(b)
        if x in vt:
            vt.append(y)
            v.remove(y)
        else:
            vt.append(x)
            v.remove(x)
    return et

#------------driver code--------------

#edge=[['a','b',3],['b','c',1],['c','d',6],['e','d',8],['a','e',6],['a','f',5],
      #['b','f',4],['c','f',4],['d','f',5],['e','f',2]]
#vert=['a','b','c','d','e','f']

edge=[['a','b',5],['b','d',6],['c','d',4],['a','c',7],['a','e',2],['b','e',3],
      ['d','e',5],['c','e',4]]
vert=['a','b','c','d','e']
st_edge=[ ]
st_edge=prims(edge,vert)

print (st_edge)
```

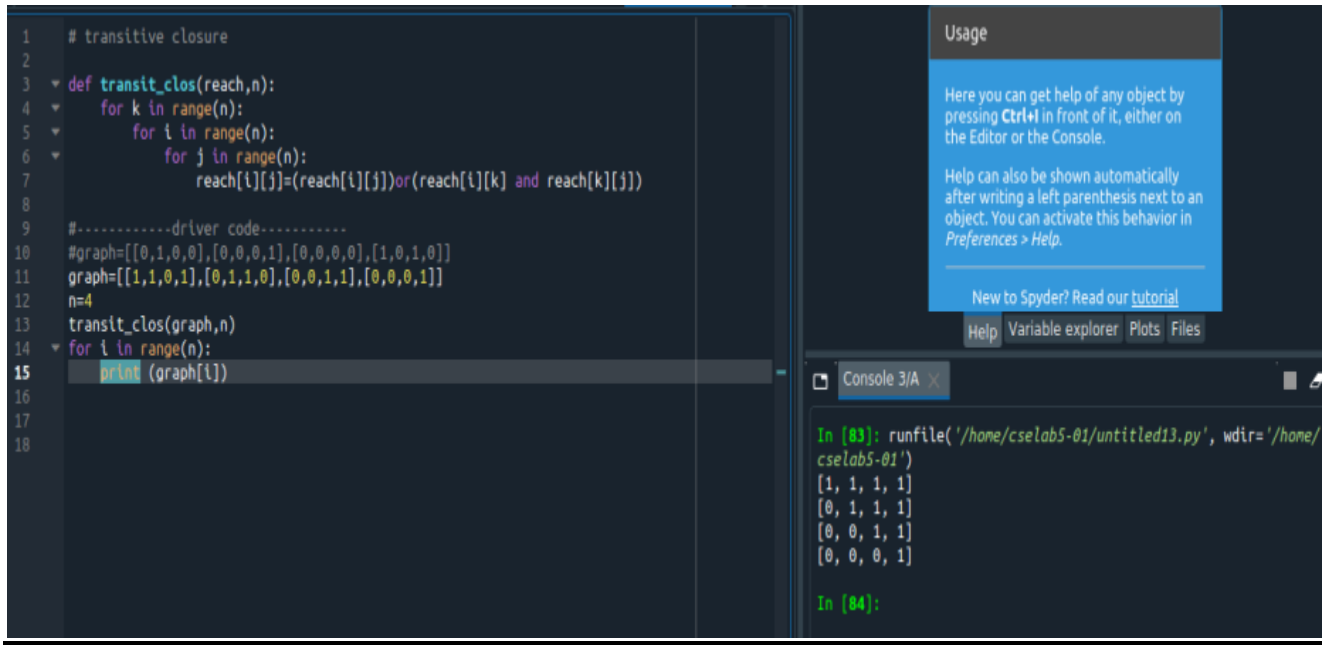
OUTPUT:

7 b.

Compute the transitive closure of a given digraph using Warshall's algorithm.

```
# transitive closure

def transit_clos(reach,n):
    for k in range(n):
        for i in range(n):
            for j in range(n):
                reach[i][j]=(reach[i][j])or(reach[i][k] and reach[k][j])

#------------driver code-----------
#graph=[[0,1,0,0],[0,0,0,1],[0,0,0,0],[1,0,1,0]]
graph=[[1,1,0,1],[0,1,1,0],[0,0,1,1],[0,0,0,1]]

n=4

transit_clos(graph,n)

for i in range(n):
    print (graph[i])
```

OUTPUT:




```
# transitive closure

def transit_clos(reach,n):
    for k in range(n):
        for i in range(n):
            for j in range(n):
                reach[i][j]=(reach[i][j])or(reach[i][k] and reach[k][j])

#-----------driver code----------
#graph=[[0,1,0,0],[0,0,0,1],[0,0,0,0],[1,0,1,0]]
graph=[[1,1,0,1],[0,1,1,0],[0,0,1,1],[0,0,0,1]]
n=4
transit_clos(graph,n)
for i in range(n):
    print (graph[i])
```

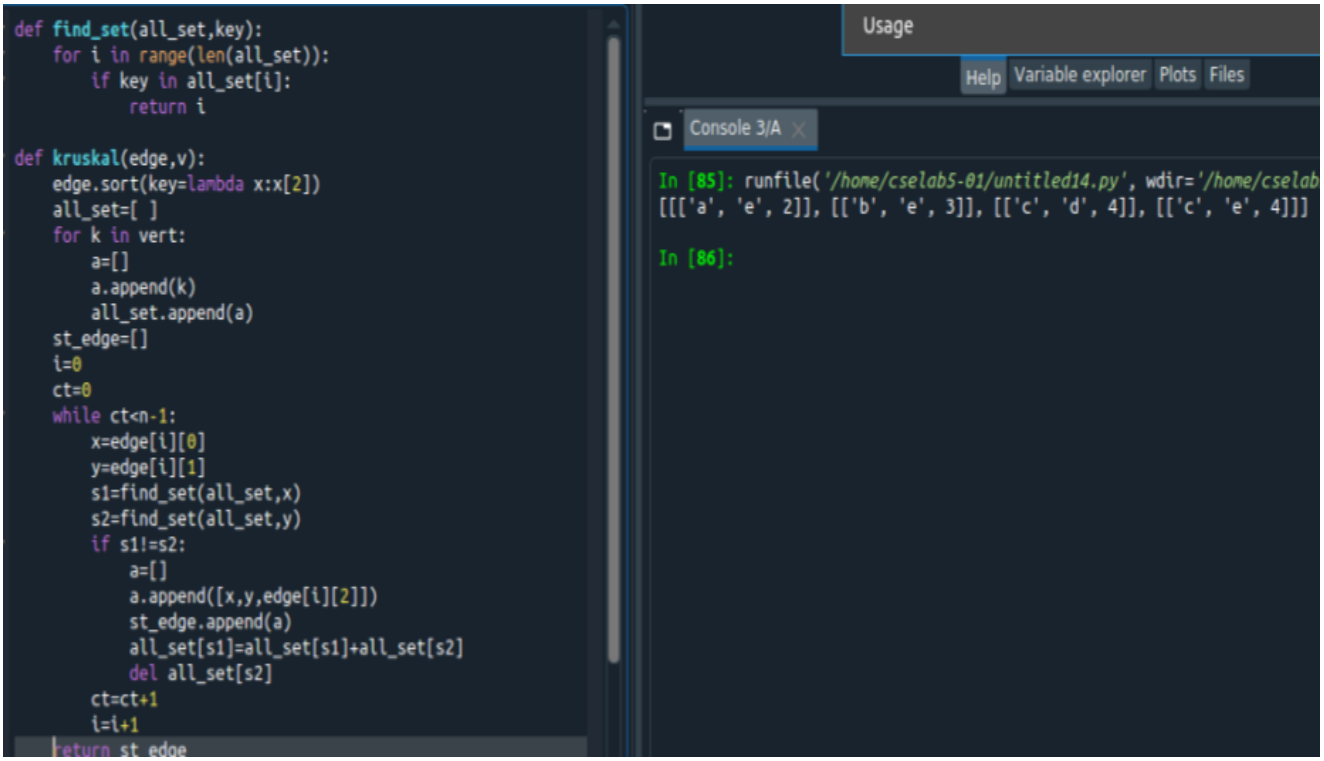
8 a.

Find minimum cost spanning tree of a given undirected graph using Kruskal's algorithm

```
def find_set(all_set,key):
    for i in range(len(all_set)):
        if key in all_set[i]:
            return i

def kruskal(edge,v):
    edge.sort(key=lambda x:x[2])
    all_set=[ ]
    for k in vert:
        a=[]
        a.append(k)
        all_set.append(a)
    st_edge=[]
    i=0
    ct=0
    while ct<n-1:
        x=edge[i][0]
        y=edge[i][1]
        s1=find_set(all_set,x)
        s2=find_set(all_set,y)
        if s1!=s2:
            a=[]
            a.append([x,y,edge[i][2]])
            st_edge.append(a)
            all_set[s1]=all_set[s1]+all_set[s2]
            del all_set[s2]
            ct=ct+1
        i=i+1
    return st_edge
```

```
#--------driver code------
#edge=[['a','b',3],['b','c',1],['c','d',6],['e','d',8],['a','e',6],['a','f',5], ['b','f',4],['c','f',4],['d','f',5],['e','f',2]]
#vert=['a','b','c','d','e','f']

edge=[['a','b',5],['b','d',6],['c','d',4],['a','c',7],['a','e',2],['b','e',3],
      ['d','e',5],['c','e',4]]
vert=['a','b','c','d','e']
n=len(vert)

st_edge=kruskal(edge,vert)
print (st_edge)
```

OUTPUT:



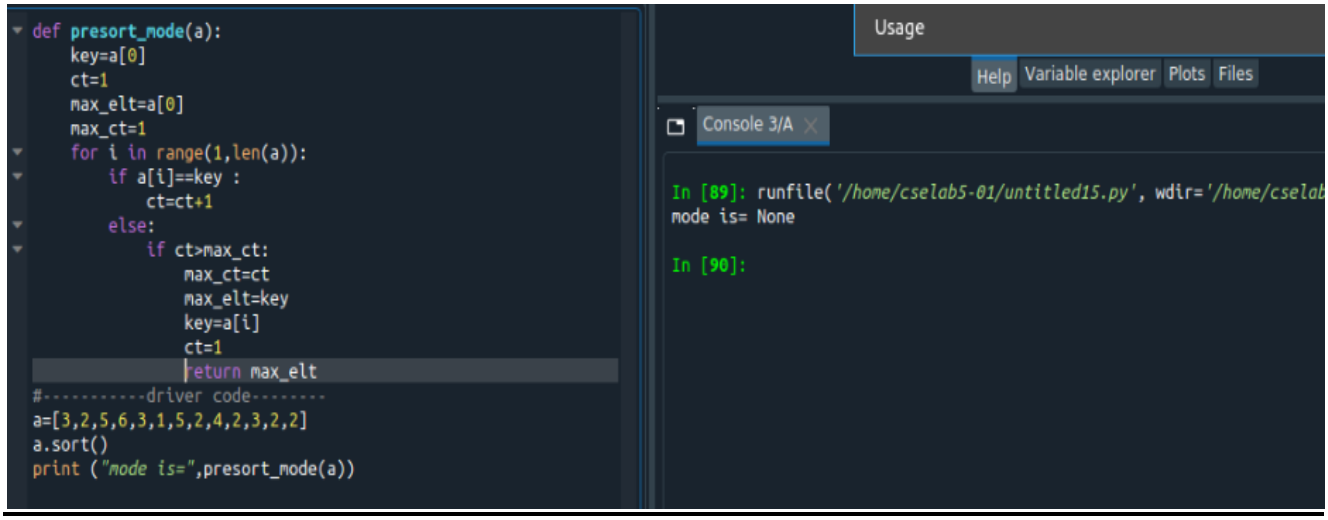
8 b.

Implement computing a mode using presorting method.

```
def presort_mode(a):
    key=a[0]
    ct=1
    max_elt=a[0]
    max_ct=1

    for i in range(1,len(a)):
        if a[i]==key :
            ct=ct+1
        else:
            if ct>max_ct:
                max_ct=ct
                max_elt=key
            key=a[i]
            ct=1
              return max_elt
#-----------driver code--------
```

```
a=[3,2,5,6,3,1,5,2,4,2,3,2,2]
a.sort()
print ("mode is=",presort_mode(a))
```

OUTPUT:



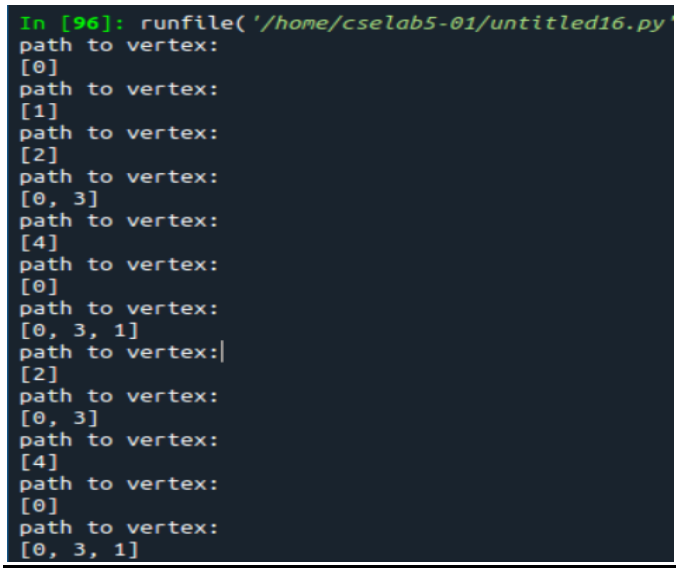
9.

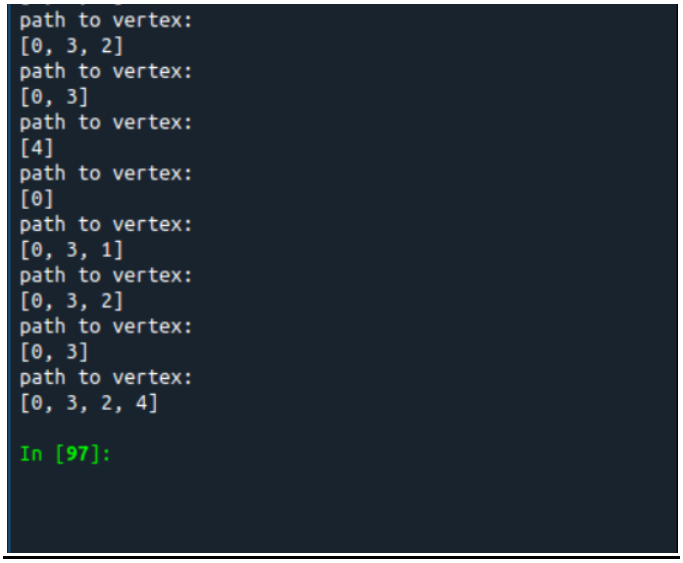
Consider the distance between Hassan and n different cities. Every city can be reached from Hassan directly or by using intermediate cities whichever costs less. Find the shortest distance from Hassan to other cities using Dijkstra's algorithm.

```
def near_vertex(d,vt):
    min=999
    for i in range(len(d)):
        if i not in vt:
            if d[i]<min:
                min=d[i]
                v=i
    return v

def path(v,pv,a):
    if pv[v]!=-1:
        a.append(pv[v])
        path(pv[v],pv,a)
def print_paths(d,pv):
    for i in range(len(d)):
        print
        print("path to vertex:")
        a=[]
        path(i,pv,a)
        a.reverse()
        a.append(i)
        print (a)

def dijkstra(graph):
    n=len(graph)
    d=[]
```

```
    pv=[]
    vt=[]
    for i in range(n):
        d.append(999)
        pv.append(-1)
    s=0
    d[s]=0
    for i in range(n):
        v=near_vertex(d,vt)
        vt.append(v)
        adj_vertex=graph[v]
        for val in adj_vertex:
            if val[0] not in vt:
                if d[v]+val[1]<d[val[0]]:
                    d[val[0]]=d[v]+val[1]
                    pv[val[0]]=v
            print_paths(d,pv)
#---------Driver code----------------

graph={0:[ [3,7]],1:[[2,4] ],2:[[4,6]],3:[[1,2],[2,5] ],4:[[3,4] ]}
#graph={0:[[1,3],[3,7] ],1:[[0,3],[2,4],[3,2] ],2:[[1,4],[3,5],[4,6] ],3:[[0,7],[1,2],[2,5],[4,4] ],4:[[2,6],
[3,4]]}

dijkstra(graph)
```

OUTPUT:



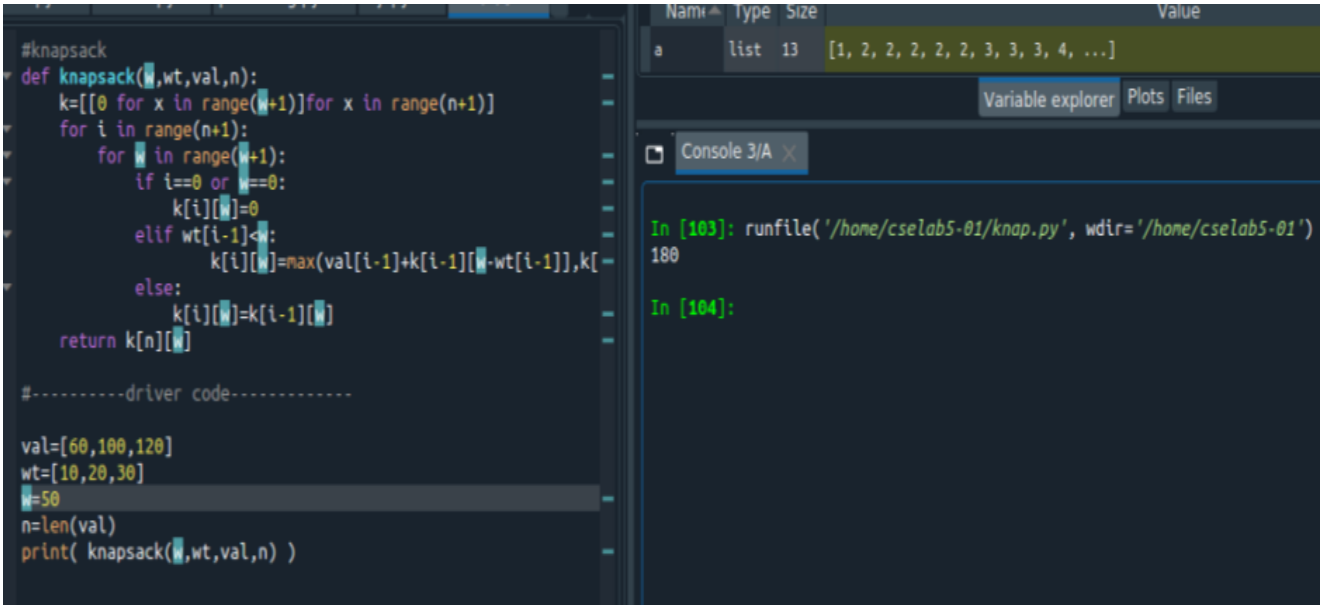


10.

Implement knapsack problem using Dynamic programming.

```
#knapsack
def knapsack(w,wt,val,n):
    k=[[0 for x in range(w+1)]for x in range(n+1)]
    for i in range(n+1):
        for w in range(w+1):
            if i==0 or w==0:
                k[i][w]=0
            elif wt[i-1]<w:
                k[i][w]=max(val[i-1]+k[i-1][w-wt[i-1]],k[i-1][w])
            else:
                k[i][w]=k[i-1][w]
    return k[n][w]

#----------driver code----------

val=[60,100,120]
wt=[10,20,30]
w=50
n=len(val)
print( knapsack(w,wt,val,n) )
```

OUTPUT:



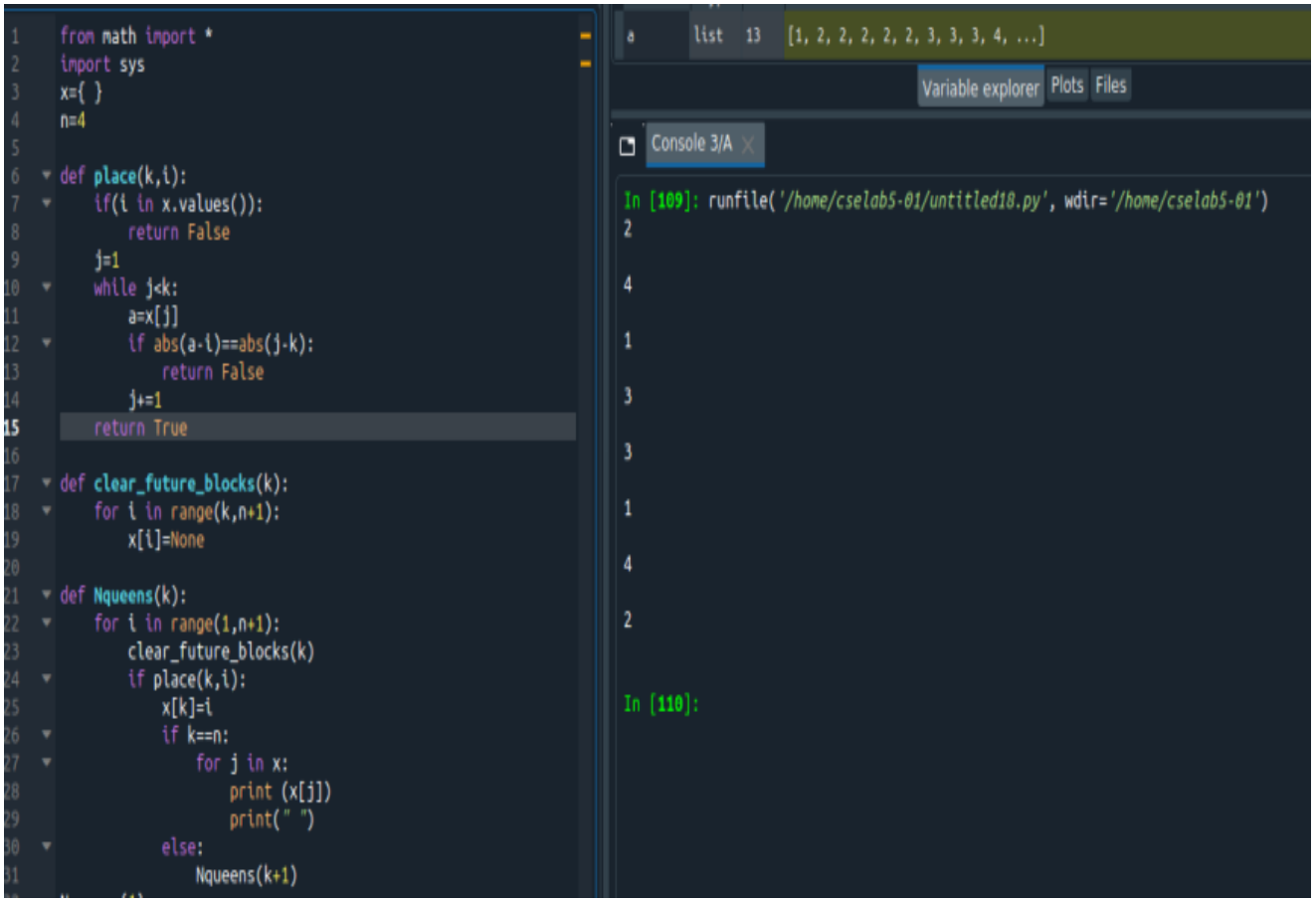
11.

Implement N Queen's problem using back tracking.

```
from math import *
import sys

x={ }
n=4

def place(k,i):
    if(i in x.values()):
        return False
    j=1
    while j<k:
        a=x[j]
        if abs(a-i)==abs(j-k):
            return False
        j+=1
    return True

def clear_future_blocks(k):
    for i in range(k,n+1):
        x[i]=None

def Nqueens(k):
    for i in range(1,n+1):
        clear_future_blocks(k)
        if place(k,i):
            x[k]=i
            if k==n:
                for j in x:
                    print (x[j])
                print(" ..... ")
            else:
                Nqueens(k+1)
```

```
#--------------Driver code--------------
Nqueens(1)
```

OUTPUT:



12. Find the subset of given set S={s1,s2,….sn} of an positive integers whose sum is equal to a given positive integer d. A suitable message is to be displayed if the given problem instance doesn't have a solution.

```
def isSubsetSum(set, n, sum):
        subset =([[False for i in range(sum + 1)]
                        for i in range(n + 1)])
                for i in range(n + 1):
                subset[i][0] = True
                        for i in range(1, sum + 1):
                subset[0][i]= False
                                for i in range(1, n + 1):
                for j in range(1, sum + 1):
                        if j<set[i-1]:
```

```
                    subset[i][j] = subset[i-1][j]
                if j>= set[i-1]:
                    subset[i][j] = (subset[i-1][j] or subset[i - 1][j-set[i-1]])
        return subset[n][sum]
        if __name__=='__main__':
set = [3, 34, 4, 12, 5, 2]
sum = 9
n = len(set)
if (isSubsetSum(set, n, sum) == True):
        print("Found a subset with given sum")
else:
        print("No subset with given sum")
```

OUTPUT: