

**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

Exam. Hours : 3

SEE: 50 Marks

L-T-P-C:0-0-1-1

Hours / Week : 02

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:

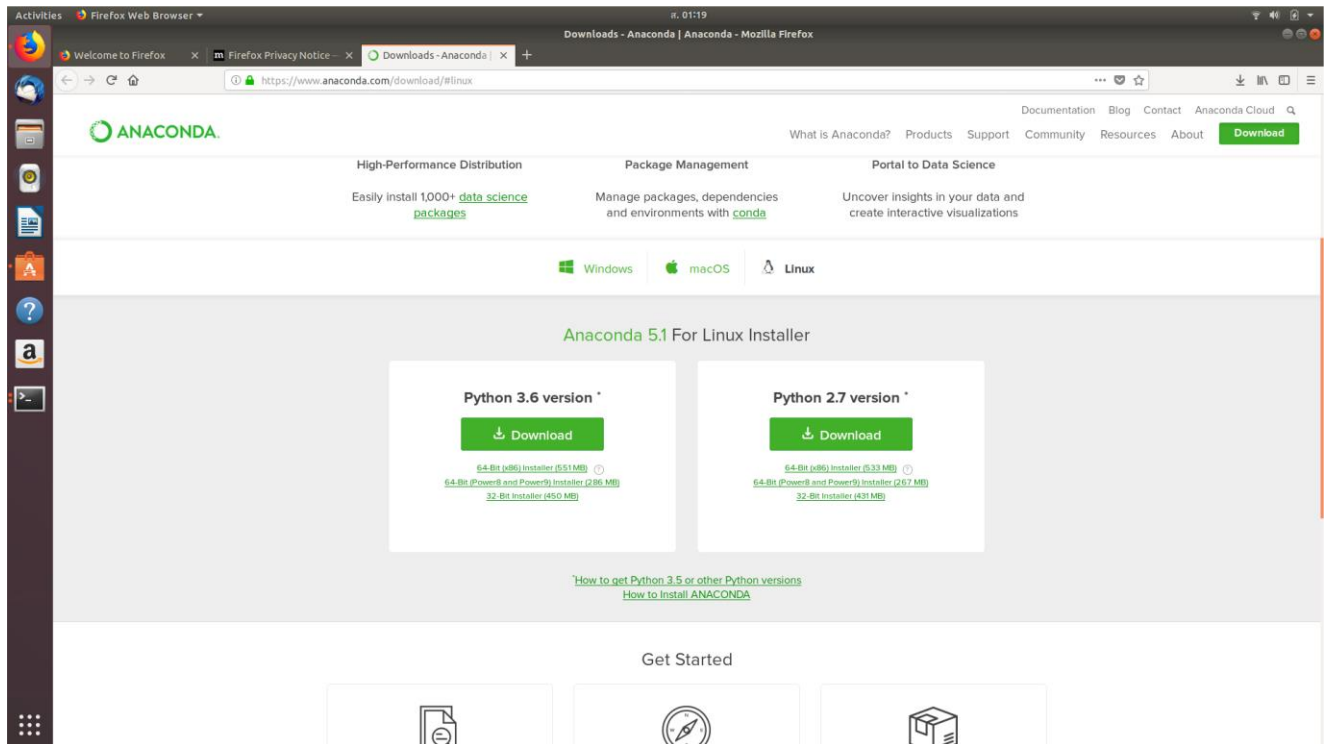
#### Implement the following using Python Language

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'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  $S=\{s_1,s_2,\dots,s_n\}$  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.

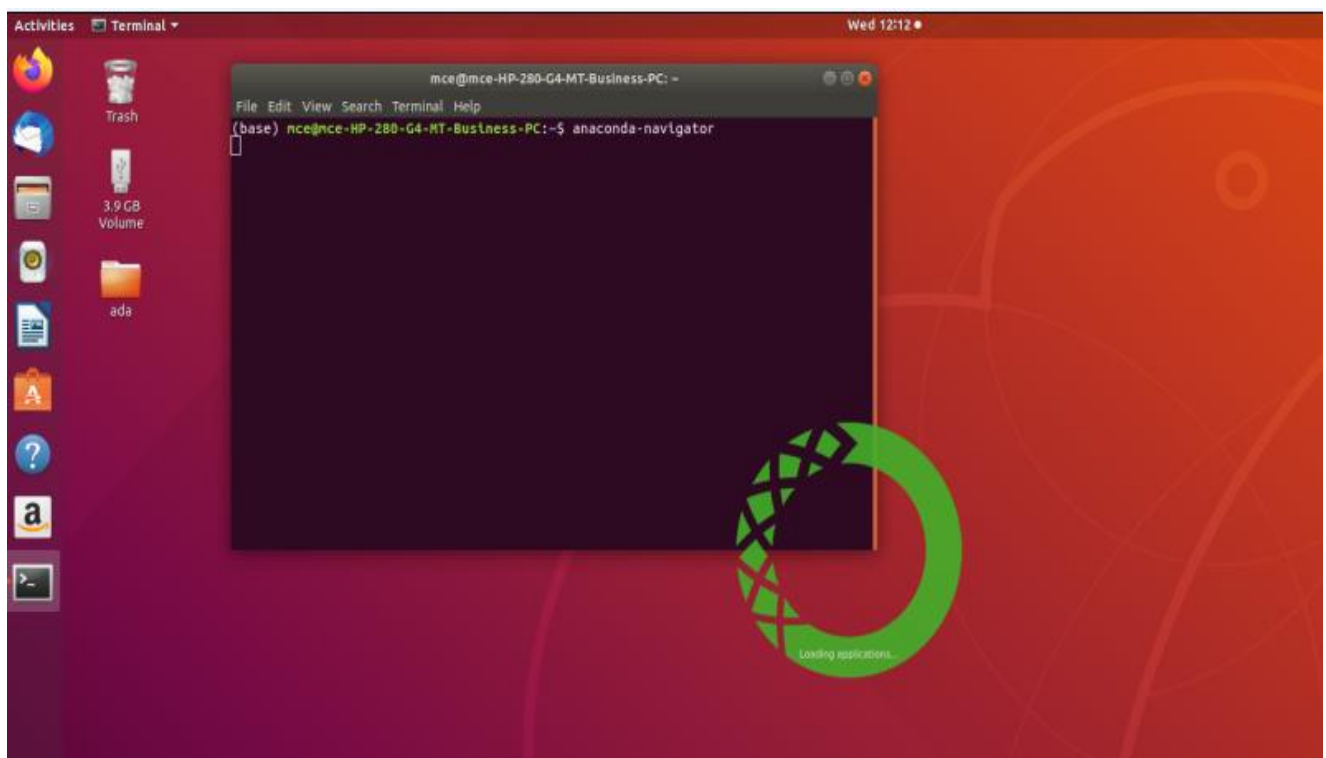
```
jitsejan@jjsvps:~/Downloads$ source ~/.bashrc
```

**Step 4:** Test if iPython is working now.

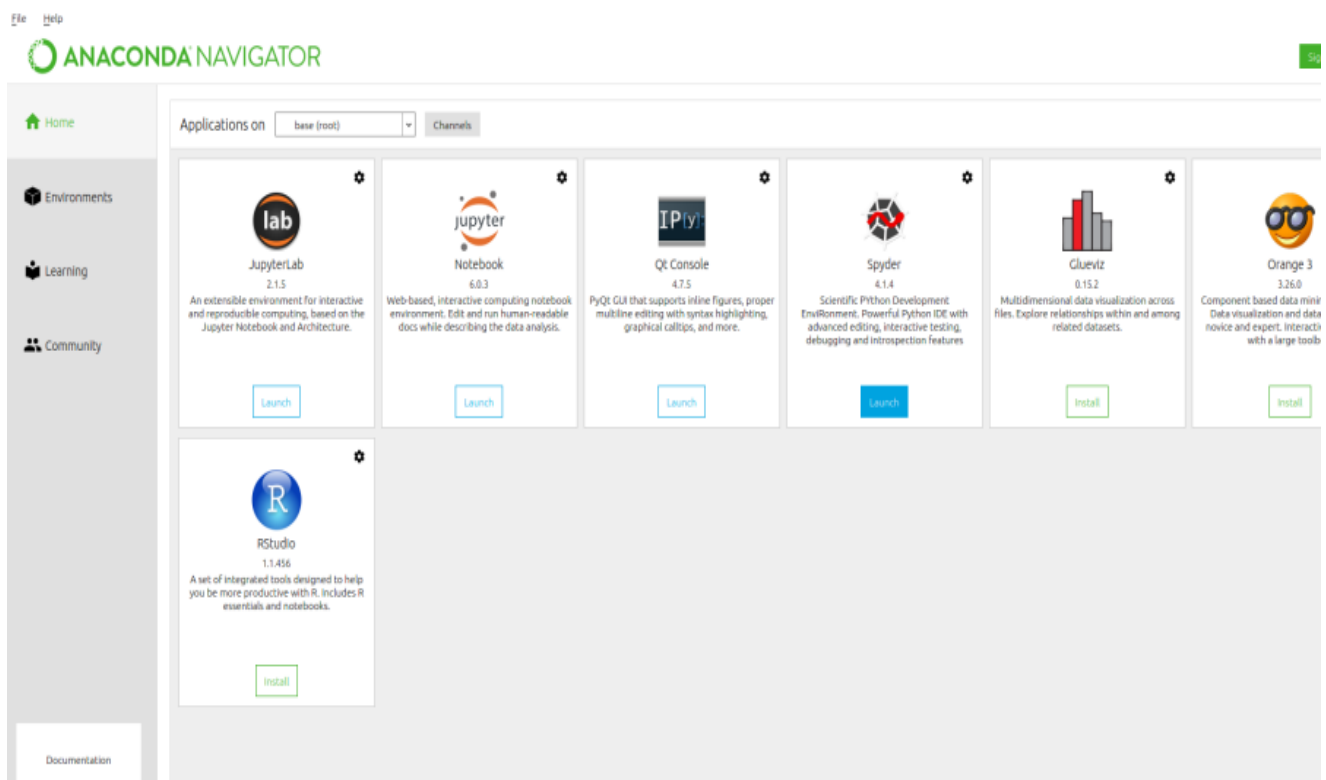
```
jitsejan@jjsvps:~$ ipython -v
```

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:

The screenshot shows the Spyder Python IDE interface. The main editor displays a Python script named `temp.py` implementing a binary search algorithm. The script includes functions for generating random data, performing a binary search, and a driver code to test the search. The console on the right shows the execution of the script, including user input for the size of the array and the key element to search for.

```
1 import random
2
3 def gen_data(a, n):
4     for i in range(n):
5         r=random.randint(0,100)
6         a.append(r)
7
8
9 def bin_srch(a, low, high, key):
10     if (low>high):
11         return -1
12     mid=int( (low+high)/2 )
13     if a[mid]==key:
14         return mid
15     elif key<a[mid]:
16         return bin_srch(a, low, mid-1, key)
17     else:
18         return bin_srch(a, mid+1, high, key) #-----driver code-----
19
20 a=[]
21 print ('Enter size:')
22 n=int(input( ))
23 gen_data(a,n)
24 a.sort()
25
26 print ('The elements are :')
27 print (a)
28
29 print ('Enter key element :')
30 key=int(input())
31 x=bin_srch(a,0,n-1,key)
32
33 if x==-1:
34     print ('Key element not found')
35 else:
36     print ('Element found at position', x+1)
```

Console 1/A:

```
In [6]: runfile('/home/mce/.config/spyder-py3/temp.py', wdir='/home/mce/.config/spyder-py3')
Enter size:
6
The elements are :
[16, 27, 37, 54, 69, 88]
Enter key element :
37
Element found at position 3

In [7]: runfile('/home/mce/.config/spyder-py3/temp.py', wdir='/home/mce/.config/spyder-py3')
Enter size:
6
The elements are :
[1, 9, 21, 54, 75, 99]
Enter key element :
4
Key element not found
```

### 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:

```

1  import random
2  def gen_data(a,n):
3      for i in range(n):
4          a.append(random.randint(0,100))
5
6  def heapify(a,n,i):
7      if 2*i+1>=n:
8          return
9
10     left=2*i+1
11     right=2*i+2
12     lar_child=left
13     if right<n:
14         if a[right]>a[left]:
15             lar_child=right
16     if a[lar_child]>a[i]:
17         a[i],a[lar_child]=a[lar_child],a[i]
18     #parent and child heapify(a,n,lar_child)
19     #child node
20
21 def heap_sort(a):
22     n=len(a)
23     for i in range(n-1,-1,-1):
24         heapify(a,n,i)
25     for i in range(n-1,0,-1):
26         a[i],a[0]=a[0],a[i]
27         heapify(a,i,0)
28
29 #-----driver code-----
30
31 a=[ ]
32 print('Enter size :')
33 n=int(input())
34 gen_data(a,n)
35
36 print('Given elements are:')
37 print (a)
38
39 heap_sort(a)
40 print('sorted array is :')
41 print (a)
42
43

```

Console 2/A

```

Python 3.8.3 (default, Jul 2 2020, 16:21:59)
Type "copyright", "credits" or "license" for more information.

IPython 7.16.1 -- An enhanced Interactive Python.

In [1]: runfile('/home/mce/untitled0.py', wdir='/home/mce')
Enter size :
4
Given elements are:
[30, 41, 92, 53]
sorted array is :
[30, 41, 53, 92]

In [2]:

```

## 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)

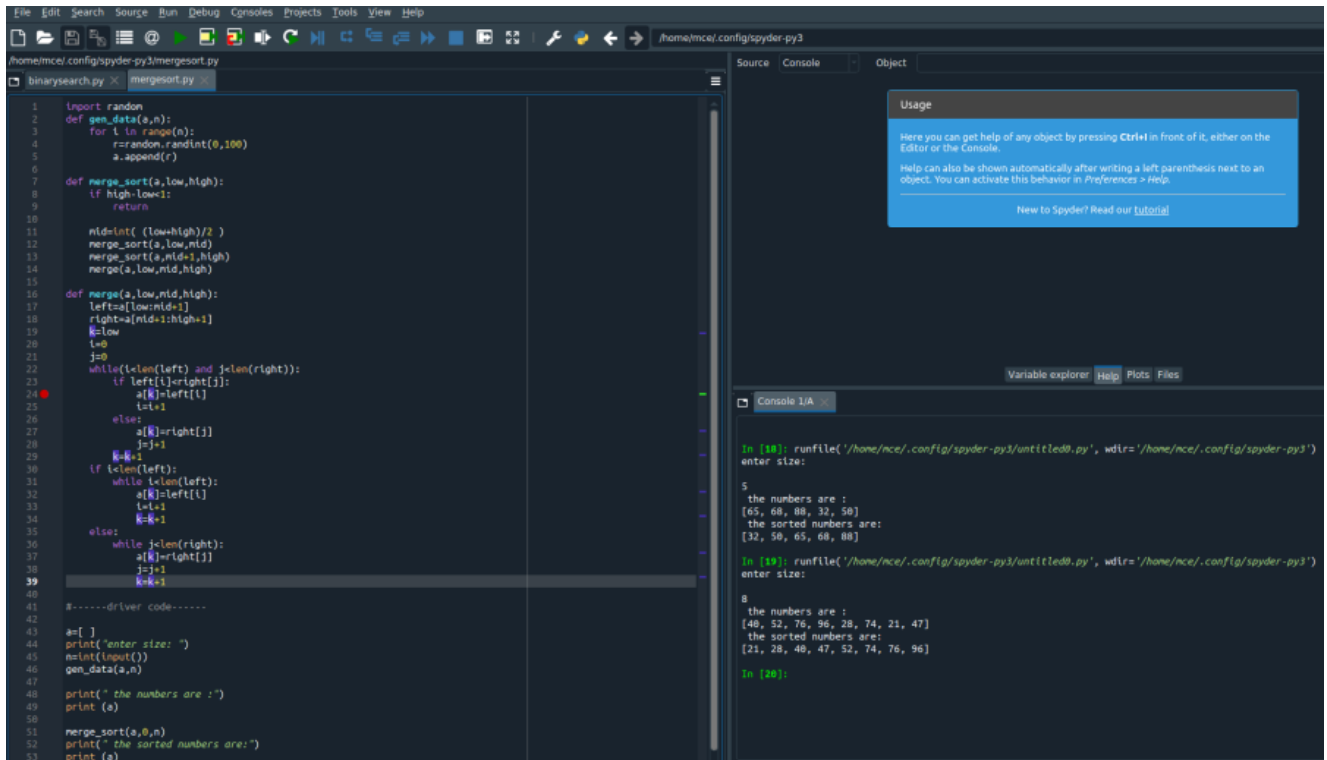
print(" the numbers are :")
print (a)

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

```



## OUTPUT:



The screenshot shows the Spyder Python IDE with a file named `mergesort.py` open. The code implements a merge sort algorithm. The console output shows the execution of the code, including the generation of random data and the sorting process.

```
1 import random
2 def gen_data(a,n):
3     for i in range(n):
4         r=random.randint(0,100)
5         a.append(r)
6
7 def merge_sort(a,low,high):
8     if high-low<1:
9         return
10
11     mid=int( (low+high)/2 )
12     merge_sort(a,low,mid)
13     merge_sort(a,mid+1,high)
14     merge(a,low,mid,high)
15
16 def merge(a,low,mid,high):
17     left=a[low:mid+1]
18     right=a[mid+1:high+1]
19     i=low
20     j=0
21     k=0
22     while(i<len(left) and j<len(right)):
23         if left[i]<right[j]:
24             a[k]=left[i]
25             i=i+1
26         else:
27             a[k]=right[j]
28             j=j+1
29         k=k+1
30     if i<len(left):
31         while i<len(left):
32             a[k]=left[i]
33             i=i+1
34             k=k+1
35     else:
36         while j<len(right):
37             a[k]=right[j]
38             j=j+1
39             k=k+1
40
41 #-----driver code-----
42
43 a=[ ]
44 print("enter size: ")
45 n=int(input())
46 gen_data(a,n)
47
48 print(" the numbers are :")
49 print (a)
50
51 merge_sort(a,0,n)
52 print(" the sorted numbers are:")
53 print (a)
```

Console Output:

```
In [10]: runfile('/home/mce/.config/spyder-py3/untitled0.py', wdir='/home/mce/.config/spyder-py3')
enter size:
5
the numbers are :
[65, 68, 88, 32, 58]
the sorted numbers are:
[32, 58, 65, 68, 88]

In [10]: runfile('/home/mce/.config/spyder-py3/untitled0.py', wdir='/home/mce/.config/spyder-py3')
enter size:
8
the numbers are :
[48, 52, 76, 96, 28, 74, 21, 47]
the sorted numbers are:
[21, 28, 48, 47, 52, 74, 76, 96]

In [20]:
```

## 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

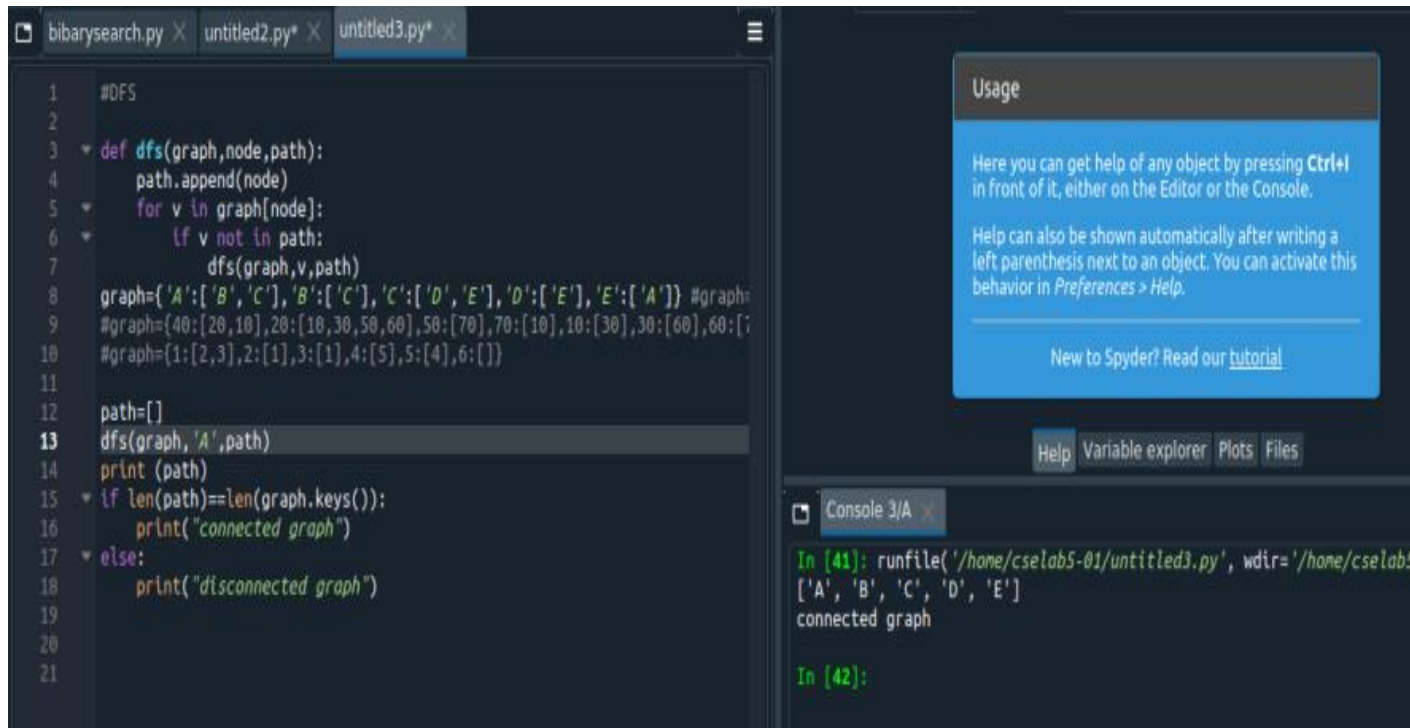
```
def dfs(graph,node,path):
    path.append(node)
    for v in graph[node]:
        if v not in path:
            dfs(graph,v,path)
```

#-----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:



The screenshot shows the Spyder Python IDE with three tabs: `bibarysearch.py`, `untitled2.py*`, and `untitled3.py*`. The active tab is `untitled3.py*`, which contains a Depth-First Search (DFS) implementation. The code defines a `dfs` function that takes a graph, a node, and a path as arguments. It appends the node to the path and recursively visits its neighbors if they are not already in the path. The graph is defined as a dictionary where keys are nodes and values are lists of adjacent nodes. The code then calls `dfs` on node 'A' and prints the resulting path. The console output shows the path `['A', 'B', 'C', 'D', 'E']` and the message `connected graph`.

```
1 #DFS
2
3 def dfs(graph,node,path):
4     path.append(node)
5     for v in graph[node]:
6         if v not in path:
7             dfs(graph,v,path)
8 graph={'A':['B','C'],'B':['C'],'C':['D','E'],'D':['E'],'E':['A']} #graph=
9 #graph={40:[20,10],20:[10,30,50,60],50:[70],70:[10],10:[30],30:[60],60:[
10 #graph=[1:[2,3],2:[1],3:[1],4:[5],5:[4],6:[]]
11
12 path=[]
13 dfs(graph,'A',path)
14 print (path)
15 if len(path)==len(graph.keys()):
16     print("connected graph")
17 else:
18     print("disconnected graph")
19
20
21
```

Console 3/A

```
In [41]: runfile('/home/cselab5-01/untitled3.py', wdir='/home/cselab5-01/untitled3.py')
['A', 'B', 'C', 'D', 'E']
connected graph

In [42]:
```

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:

The screenshot shows a Python IDE with the following components:

- Source Editor:** Contains the Python code for a quick sort algorithm and its driver code. The code is as follows:
 

```

1 import random
2
3 def gen_data(a,n):
4     for i in range(n):
5         x=random.randint(0,100)
6         a.append(x)
7
8 def partition(a,left,right):
9     key=a[left]
10    i=left+1
11    j=right
12
13    while True:
14        while a[i]<key and i<right:
15            i=i+1
16        while a[j]>key and j>left:
17            j=j-1
18        if i<j:
19            a[i],a[j]=a[j],a[i]
20            i=i+1
21            j=j-1
22        else:
23            break
24
25    a[left],a[j]=a[j],a[left]
26    return j
27
28 def quick_sort(a,low,high):
29     if low<high:
30         pos=partition(a,low,high)
31         quick_sort(a,low,pos-1)
32         quick_sort(a,pos+1,high)
33
34 #-----driver code-----
35 a=[ ]
36 print("enter size: ")
37 n=int(input())
38 gen_data(a,n)
39
40 print("the numbers are:")
41 print (a)
42
43 quick_sort(a,0,n-1)
44
45 print("the sorted numbers are:")
46 print (a)
47

```
- Console:** Shows the output of the program for two different input sizes.
 

```

In [1]: runfile('/home/mce/quick.py', wdir='/home/mce')
enter size:
8
the numbers are:
[73, 75, 94, 47, 26, 21, 78, 84]
the sorted numbers are:
[21, 26, 47, 73, 75, 78, 84, 94]

In [2]: runfile('/home/mce/quick.py', wdir='/home/mce')
enter size:
10
the numbers are:
[29, 78, 29, 24, 78, 18, 18, 61, 85, 37]
the sorted numbers are:
[18, 18, 24, 29, 29, 37, 61, 78, 78, 85]

In [3]:

```

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

#BFS

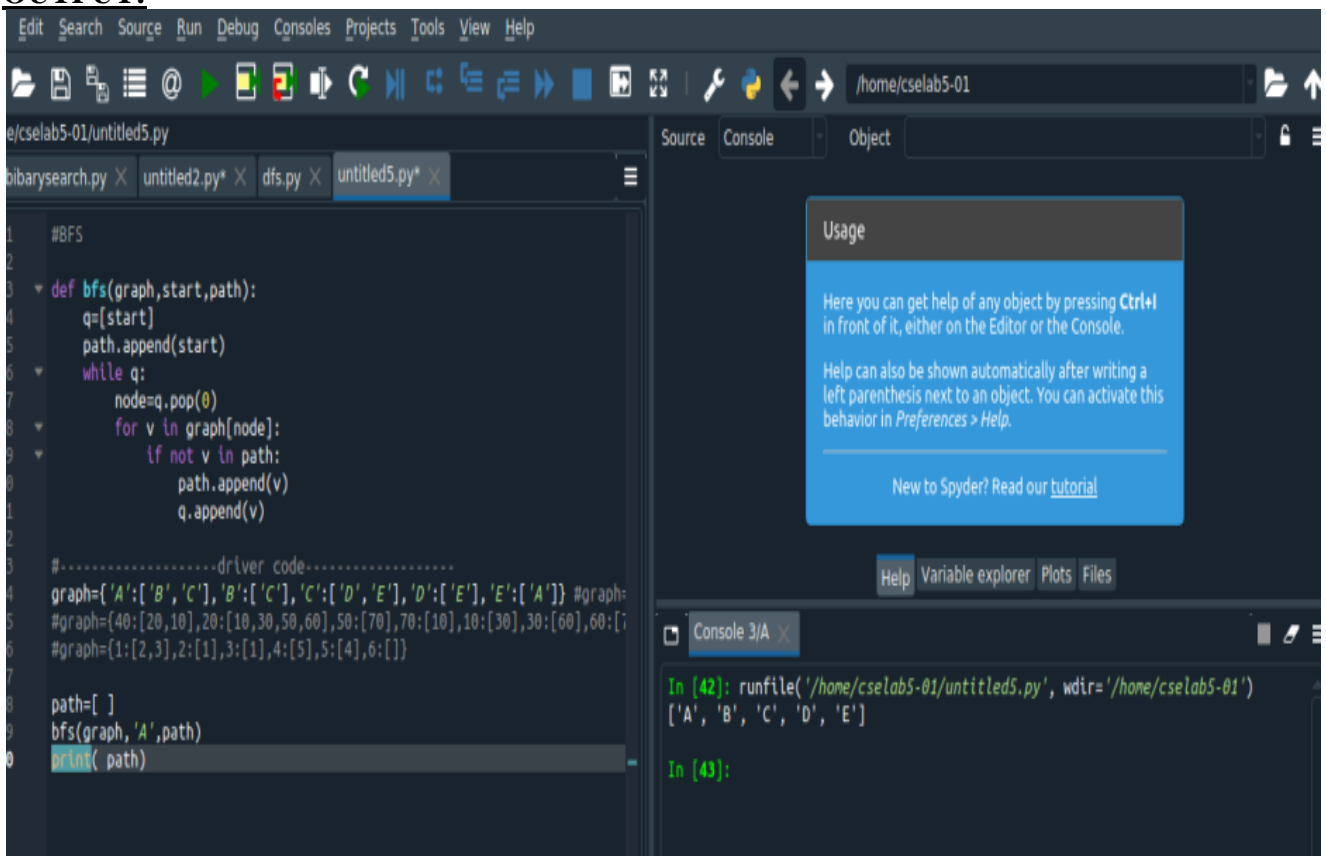
```
def bfs(graph,start,path):
    q=[start]
    path.append(start)
    while q:
        node=q.pop(0)
        for v in graph[node]:
            if not v in path:
                path.append(v)
                q.append(v)

#-----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:**

```
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)
```

Usage

Here you can get help of any object by pressing **Ctrl+H** in front of it, either on the Editor or the Console.

Help can also be shown automatically after writing a left parenthesis next to an object. You can activate this behavior in *Preferences > Help*.

[New to Spyder? Read our tutorial](#)

Help Variable explorer Plots Files

Console 3/A

```
In [44]: runfile('/home/cselabs-01/untitled6.py', wdir='/home/cselabs-01')
enter size:
6
 the numbers are:
[29, 54, 15, 4, 98, 32]
 the sorted numbers are:
[4, 15, 29, 32, 54, 98]
In [45]:
```

#### 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()

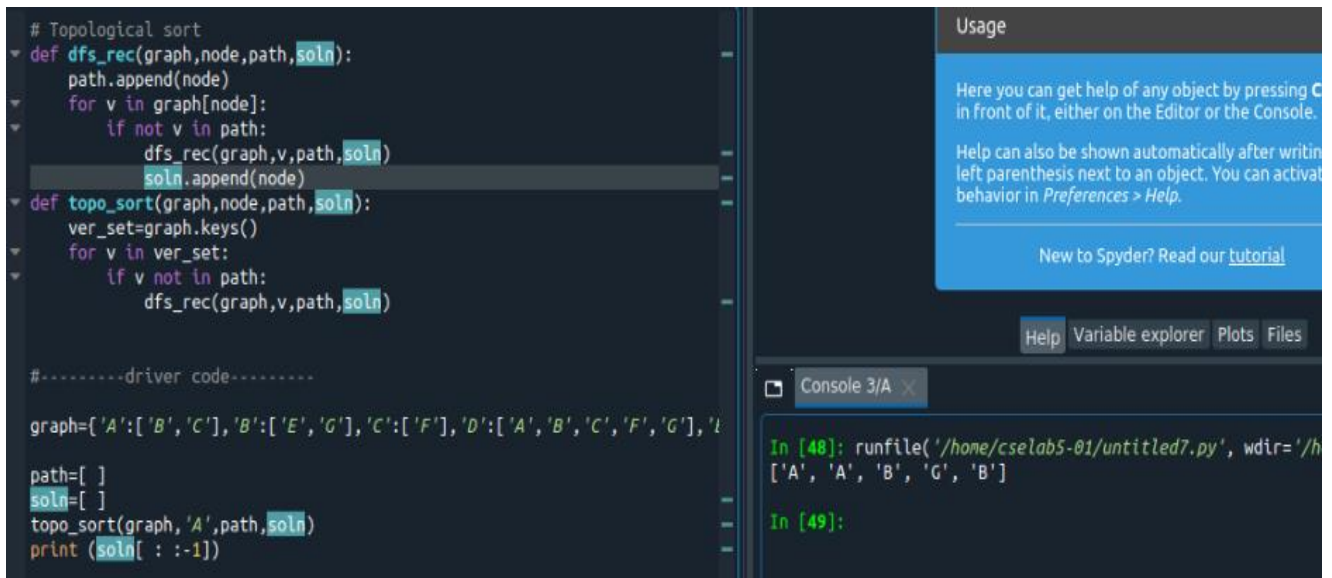
    for v in ver_set:
        if v not in path:
            dfs_rec(graph,v,path,soln)

#-----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:



The screenshot displays the Spyder Python IDE interface. The left pane shows the code editor with the following Python code:

```
# 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])
```

The right pane shows the console output. A 'Usage' help window is open at the top. Below it, the console shows the execution of the code:

```
In [48]: runfile('/home/cselab5-01/untitled7.py', wdir='/h
['A', 'A', 'B', 'G', 'B']

In [49]:
```

#### 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:

```

1  def horspool(pattern,text):
2      m=len(pattern)
3      n=len(text)
4      if m<n:
5          return -1
6      bad_char_jump=[n]*250
7      for i in range(n-1):
8          bad_char_jump[ord(text[i])]=n-i-1
9      pos=0
10     while pos<=m-n:
11         j=n-1
12         while j>=0 and text[j]==pattern[pos+j]:
13             j=j-1
14         if j==-1:
15             return pos
16         else:
17             pos=pos+bad_char_jump[ord(pattern[pos+n-1])]
18     return -1
19
20 #-----Driver code-----
21 text=input("text:")
22 pattern=input("pattern:")
23 print (horspool(pattern,text))
24
25

```

Usage

Here you can get help of any object by pressing **Ctrl+I** in front of it, either on the Editor or the Console.

Help can also be shown automatically after writing a left parenthesis next to an object. You can activate this behavior in *Preferences > Help*.

New to Spyder? Read our [tutorial](#)

Help Variable explorer Plots Files

Console 3/A

```

In [59]: runfile('/home/cselab5-01/horspool.py', wdir='/home/cselab5-01')
text:algorithms
pattern:thm
-1

In [60]: runfile('/home/cselab5-01/horspool.py', wdir='/home/cselab5-01')
text:algorithms
pattern:ui
-1

```

**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:**

The screenshot shows the Spyder IDE interface. The left pane contains the following Python code:

```

1  def check(n):
2      a=[ ]
3      while n!=0:
4          x=n%10
5          a.append(x)
6          n=n//10
7          a.sort()
8          flag=True
9          for i in range(len(a)-1):
10             if a[i]==a[i+1]:
11                 flag=False
12                 break
13             return flag
14
15 #-----driver code-----
16 n=7829465031
17 #n=8900345212
18 print (check(n))
19
20
21

```

The right pane shows the 'Console 3/A' with the following output:

```

In [68]: runfile('/home/cselabs-01/transfer a
cselabs-01')
True

In [69]: #-----driver code-----

In [70]:

```

There is also a 'Usage' panel on the right with text about getting help and a 'New to Spyder? Read' link.

## **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)

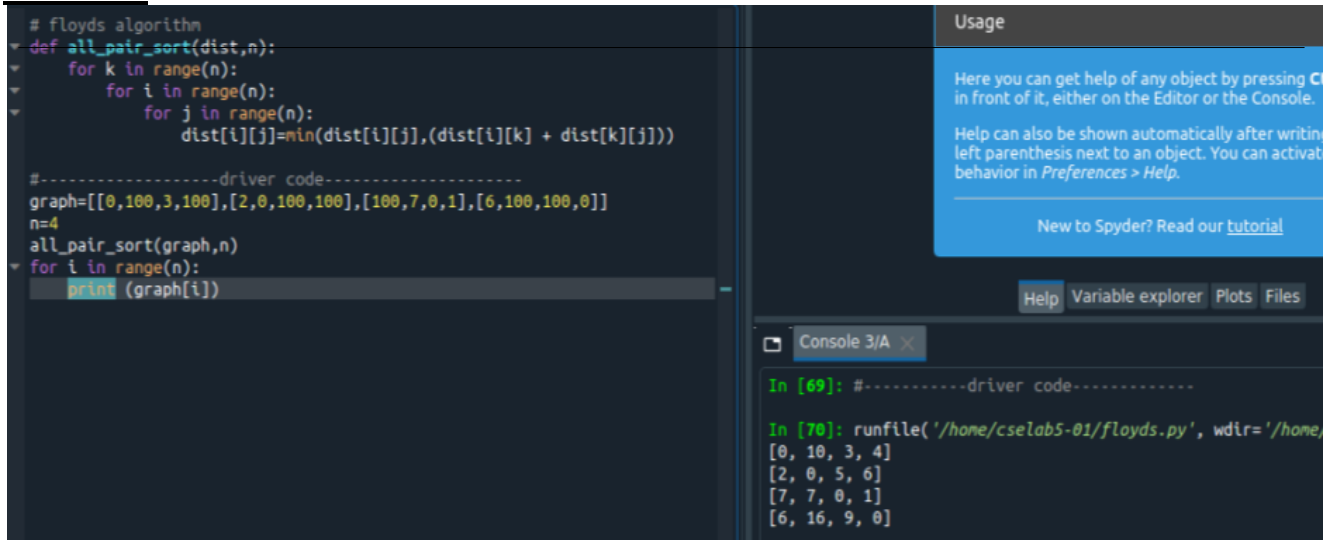
for i in range(n):

```



```
print (graph[i])
```

### OUTPUT:



The screenshot shows the Spyder IDE interface. The left pane contains the code for Floyd's algorithm, which calculates the shortest paths between all pairs of nodes in a weighted graph. The code defines a function `all_pair_sort` and uses it on a specific graph. The right pane shows the console output, which displays the resulting shortest path matrix.

```
# 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])
```

Console 3/A

```
In [69]: #-----driver code-----
In [70]: runfile('/home/cselab5-01/floyds.py', wdir='/home/cselab5-01')
[0, 10, 3, 4]
[2, 0, 5, 6]
[7, 7, 0, 1]
[6, 16, 9, 0]
```

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]
```

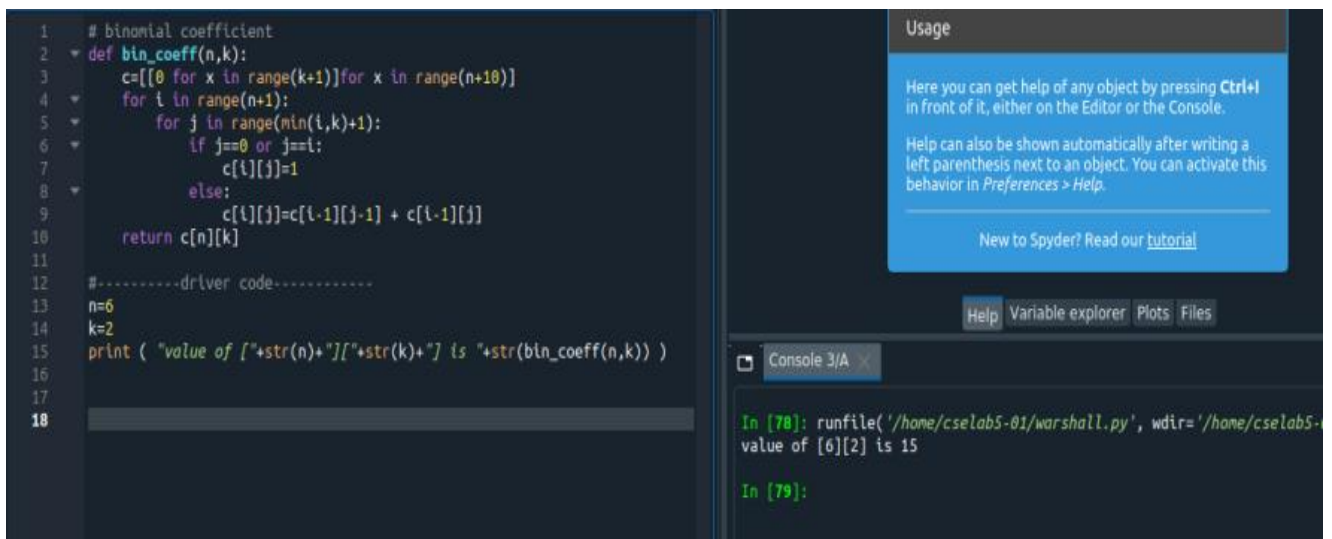
#-----driver code-----

n=6

k=2

```
print ( "value of [" +str(n)+"][" +str(k)+"] is "+str(bin_coeff(n,k)) )
```

### OUTPUT:



The screenshot shows the Spyder IDE interface. The left pane contains the code for calculating the binomial coefficient using dynamic programming. The code defines a function `bin_coeff` and uses it to calculate the value of  $C(6, 2)$ . The right pane shows the console output, which displays the result 15.

```
1 # binomial coefficient
2 def bin_coeff(n,k):
3     c=[[0 for x in range(k+1)]for x in range(n+10)]
4     for i in range(n+1):
5         for j in range(min(i,k)+1):
6             if j==0 or j==i:
7                 c[i][j]=1
8             else:
9                 c[i][j]=c[i-1][j-1] + c[i-1][j]
10    return c[n][k]
11
12 #-----driver code-----
13 n=6
14 k=2
15 print ( "value of [" +str(n)+"][" +str(k)+"] is "+str(bin_coeff(n,k)) )
16
17
18
```

Console 3/A

```
In [70]: runfile('/home/cselab5-01/warshall.py', wdir='/home/cselab5-01')
value of [6][2] is 15
In [79]:
```

**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

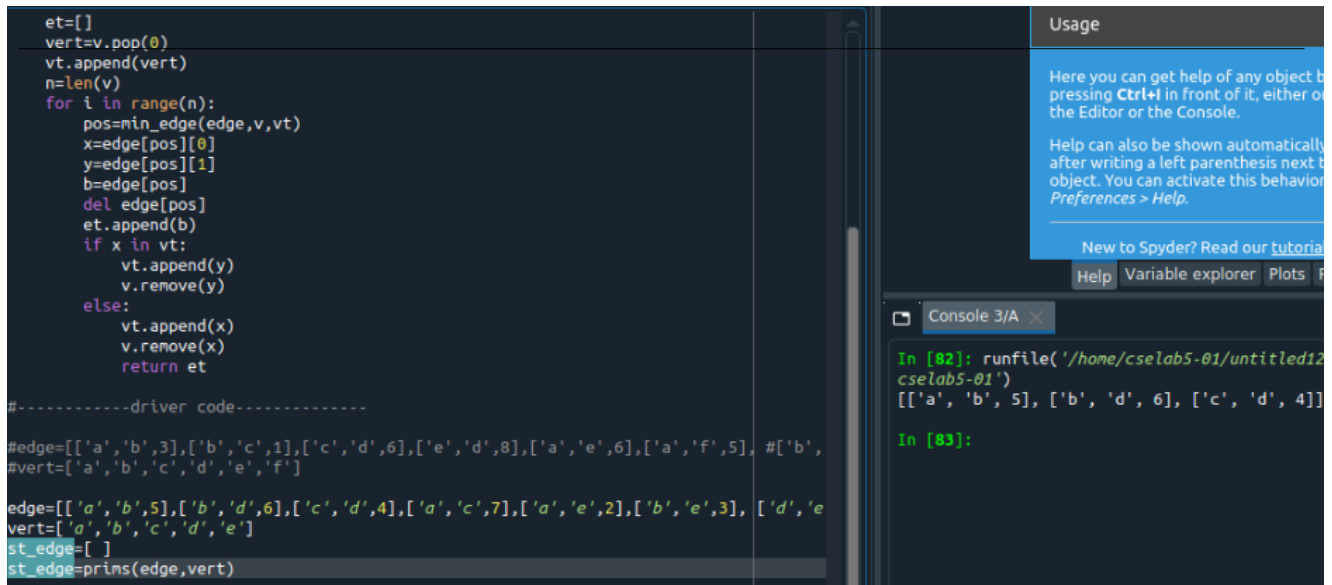
#-----driver code-----

#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:



```
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',
#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
vert=['a','b','c','d','e']
st_edge=[]
st_edge=prins(edge,vert)
```

Usage

Here you can get help of any object by pressing **Ctrl+I** in front of it, either on the Editor or the Console.

Help can also be shown automatically after writing a left parenthesis next to an object. You can activate this behavior in [Preferences > Help](#).

New to Spyder? Read our [tutorial](#)

Help Variable explorer Plots

Console 3/A

```
In [82]: runfile('/home/cselab5-01/untitled12',
               cselab5-01')
[['a', 'b', 5], ['b', 'd', 6], ['c', 'd', 4]]

In [83]:
```

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-----

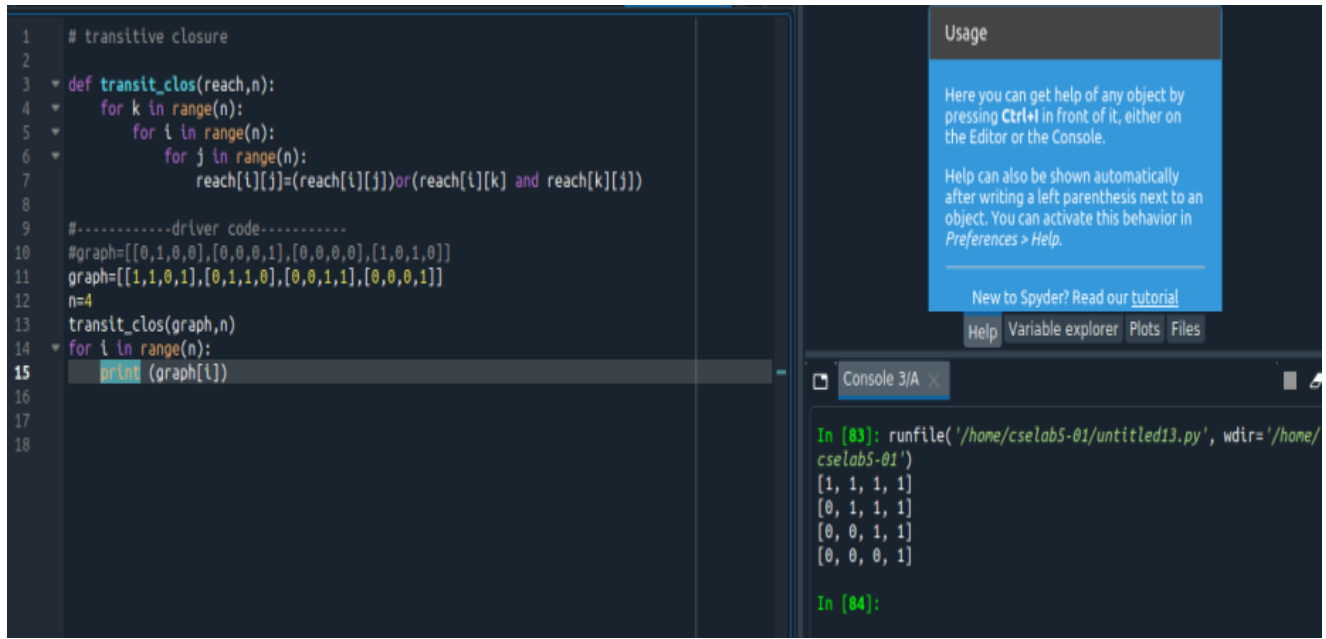
```
#graph=[[0,1,0,0],[0,0,0,1],[0,0,0,0],[1,0,1,0]]
graph=[[1,1,0,1],[0,1,1,0],[0,0,1,1],[0,0,0,1]]
```

n=4

transit\_clos(graph,n)

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

## OUTPUT:



The screenshot shows a code editor with a Python script and its output in the console. The script defines a function `transit_clos` to calculate the transitive closure of a graph. The graph is represented as a 4x4 adjacency matrix. The console output shows the result of the function call, which is a 4x4 matrix of 1s and 0s.

```
1 # transitive closure
2
3 def transit_clos(reach,n):
4     for k in range(n):
5         for i in range(n):
6             for j in range(n):
7                 reach[i][j]=(reach[i][j])or(reach[i][k] and reach[k][j])
8
9 #-----driver code-----
10 #graph=[[0,1,0,0],[0,0,0,1],[0,0,0,0],[1,0,1,0]]
11 graph=[[1,1,0,1],[0,1,1,0],[0,0,1,1],[0,0,0,1]]
12 n=4
13 transit_clos(graph,n)
14 for i in range(n):
15     print (graph[i])
16
17
18
```

Usage

Here you can get help of any object by pressing **Ctrl+I** in front of it, either on the Editor or the Console.

Help can also be shown automatically after writing a left parenthesis next to an object. You can activate this behavior in *Preferences > Help*.

New to Spyder? Read our [tutorial](#)

Help Variable explorer Plots Files

Console 3/A

```
In [03]: runfile('/home/cselab5-01/untitled13.py', wdir='/home/cselab5-01')
[1, 1, 1, 1]
[0, 1, 1, 1]
[0, 0, 1, 1]
[0, 0, 0, 1]

In [04]:
```

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:**

```
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
```

Usage

Help Variable explorer Plots Files

Console 3/A

```
In [85]: runfile('/home/cselab5-01/untitled14.py', wdir='/home/cselab5-01')
[[['a', 'e', 2]], [['b', 'e', 3]], [['c', 'd', 4]], [['c', 'e', 4]]]

In [86]:
```

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:

```

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))

```

Usage

Help Variable explorer Plots Files

Console 3/A

```

In [89]: runfile('/home/cselab5-01/untitled15.py', wdir='/home/cselab5-01')
mode is= None

In [90]:

```

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,v):
    min=999
    for i in range(len(d)):
        if i not in v:
            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:**

```

In [96]: runfile('/home/cselab5-01/untitled16.py')
path to vertex:
[0]
path to vertex:
[1]
path to vertex:
[2]
path to vertex:
[0, 3]
path to vertex:
[4]
path to vertex:
[0]
path to vertex:
[0, 3, 1]
path to vertex:
[2]
path to vertex:
[0, 3]
path to vertex:
[4]
path to vertex:
[0]
path to vertex:
[0, 3, 1]

```

```

path to vertex:
[0, 3, 2]
path to vertex:
[0, 3]
path to vertex:
[4]
path to vertex:
[0]
path to vertex:
[0, 3, 1]
path to vertex:
[0, 3, 2]
path to vertex:
[0, 3]
path to vertex:
[0, 3, 2, 4]

```

```
In [97]:
```

**10.**

**Implement knapsack problem using Dynamic programming.**

```
#knapsack
```

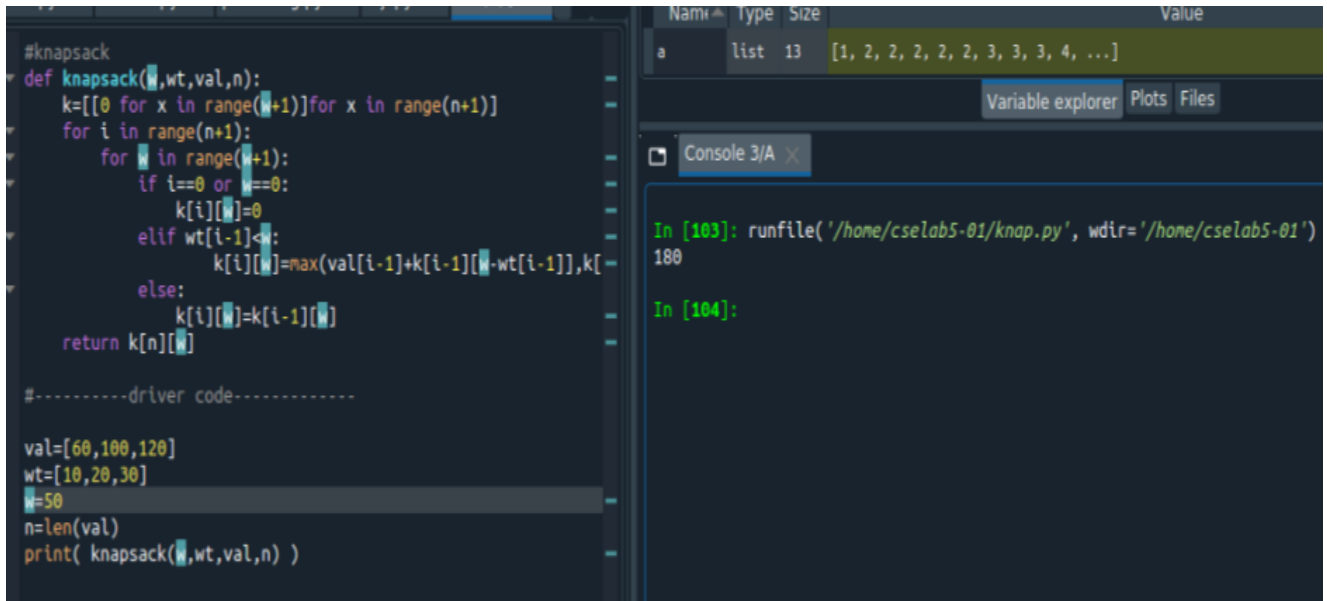
```
def knapsack(w,wt,val,n):
    k=[[0 for x in range(w+1)]for x in range(n+1)]
    for i in range(n+1):
        for w in range(w+1):
            if i==0 or w==0:
                k[i][w]=0
            elif wt[i-1]<w:
                k[i][w]=max(val[i-1]+k[i-1][w-wt[i-1]],k[i-1][w])
            else: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



The screenshot shows a Jupyter Notebook interface. On the left, a code editor displays a Python function `knapsack` and driver code. The function takes `wt`, `val`, and `n` as arguments and returns a list `k`. The driver code sets `val=[60,100,120]`, `wt=[10,20,30]`, `w=50`, and `n=len(val)`, then prints the result of `knapsack(w,wt,val,n)`. On the right, the 'Variable explorer' shows a variable `a` of type `list` with size 13 and value `[1, 2, 2, 2, 2, 2, 2, 3, 3, 3, 4, ...]`. Below it, the 'Console' shows the output of running the code: `In [103]: runfile('/home/cselab5-01/knap.py', wdir='/home/cselab5-01')` followed by the value `180`, and `In [104]:` on the next line.

## 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:

```
1 from math import *
2 import sys
3 x={ }
4 n=4
5
6 def place(k,l):
7     if(l in x.values()):
8         return False
9     j=1
10    while j<k:
11        a=x[j]
12        if abs(a-l)==abs(j-k):
13            return False
14        j+=1
15    return True
16
17 def clear_future_blocks(k):
18     for i in range(k,n+1):
19         x[i]=None
20
21 def Nqueens(k):
22     for i in range(1,n+1):
23         clear_future_blocks(k)
24         if place(k,i):
25             x[k]=i
26             if k==n:
27                 for j in x:
28                     print (x[j])
29                     print(" ")
30             else:
31                 Nqueens(k+1)
```

list 13 [1, 2, 2, 2, 2, 2, 3, 3, 3, 4, ...]

Variable explorer Plots Files

Console 3/A

In [109]: runfile('/home/cselab5-01/untitled18.py', wdir='/home/cselab5-01')

2

4

1

3

3

1

4

2

In [110]:

**12. Find the subset of given set  $S=\{s_1, s_2, \dots, s_n\}$  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:**

The screenshot shows the Spyder Python IDE interface. The source editor on the left contains the following code:

```

1  def isSubsetSum(set, n, sum):
2
3
4
5      subset = [[False for i in range(sum + 1)]
6                for i in range(n + 1)]
7
8
9      for i in range(n + 1):
10         subset[i][0] = True
11
12
13         for l in range(1, sum + 1):
14             subset[i][l] = False
15
16
17         for l in range(1, n + 1):
18             for j in range(1, sum + 1):
19                 if j < set[i-1]:
20                     subset[l][j] = subset[l-1][j]
21                 if j >= set[i-1]:
22                     subset[l][j] = (subset[l-1][j] or
23                                   subset[l-1][j-set[i-1]])
24
25     return subset[n][sum]
26
27 if __name__ == '__main__':
28     set = [3, 34, 4, 12, 5, 2]
29     sum = 9
30     n = len(set)
31     if (isSubsetSum(set, n, sum) == True):
32         print("Found a subset with given sum")
33     else:
34         print("No subset with given sum")
35
36

```

The console on the right shows the output of the program:

```

In [4]: runfile('/home/nce/subset.py', wdir='/home/nce')
Found a subset with given sum

In [5]:

```