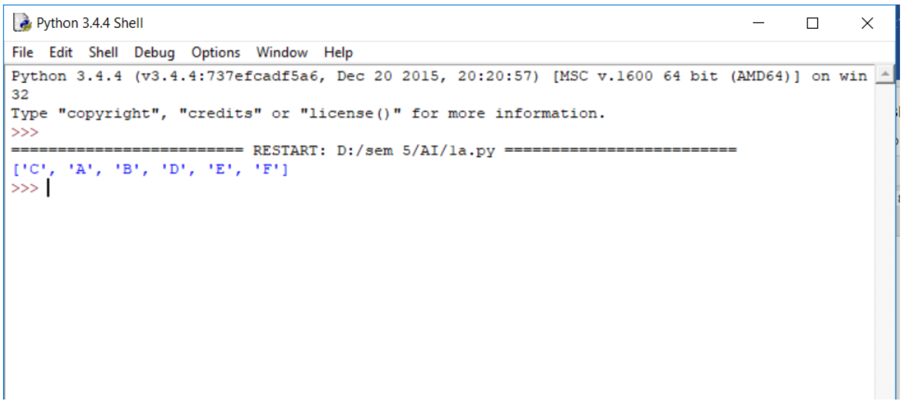
INDEX

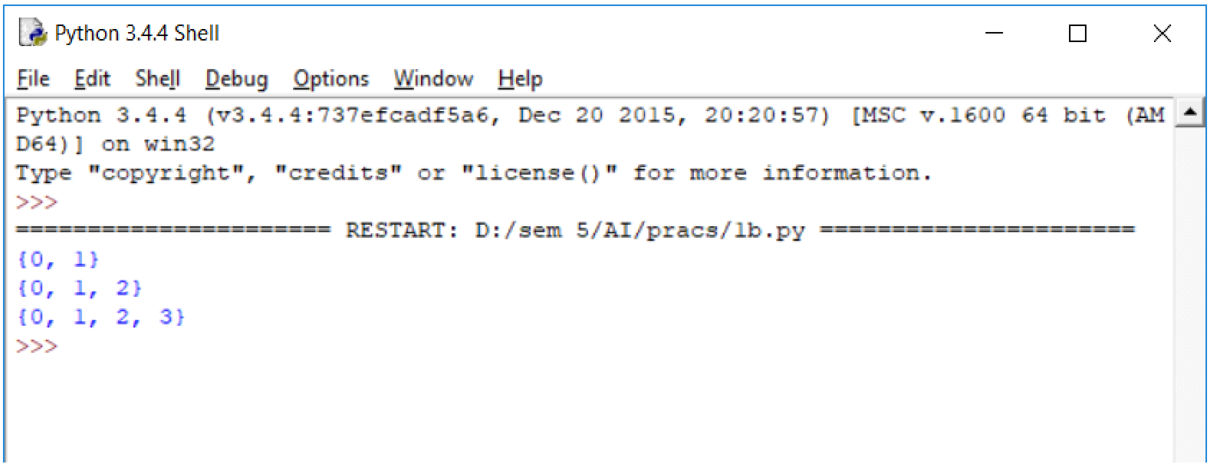
Sr. No		Practical Name	Date	Sign
1	a	Write a program to implement depth first search algorithm.		
	b	Write a program to implement breadth first search algorithm.		
2	a	Write a program to simulate 4-Queen / N-Queen problem.		
	b	Write a program to solve tower of Hanoi problem.		
3	a	Write a program to implement alpha beta search.		
	b	Write a program for Hill climbing problem.		
4	a	Write a program to implement A* algorithm.		
5	a	Write a program to solve water jug problem.		
	b	Design the simulation of tic – tac – toe game using min-max algorithm.		
6	a	Write a program to solve Missionaries and Cannibals problem.		
	b	Design an application to simulate number puzzle problem.		
7	a	Write a program to shuffle Deck of cards.		
	b	Solve traveling salesman problem using artificial intelligence technique.		
8	a	Solve constraint satisfaction problem		
9	a	Derive the expressions based on Associative law		
	b	Derive the expressions based on Distributive law		
10	a	Write a program to derive the predicate. (for e.g.: Sachin is batsman , batsman is cricketer) - > Sachin is Cricketer.		
	b	Write a program which contains three predicates: male, female, parent. Make rules for following family relations: father, mother, grandfather, grandmother, brother, sister, uncle, aunt, nephew and niece, cousin. Question: i. Draw Family Tree. ii. Define: Clauses, Facts, Predicates and Rules with conjunction and disjunction		

Output 1a



```
Python 3.4.4 (v3.4.4:737efcadf5a6, Dec 20 2015, 20:20:57) [MSC v.1600 64 bit (AMD64)] on win
32
Type "copyright", "credits" or "license()" for more information.
>>> 
========================= RESTART: D:/sem 5/AI/1a.py =========================
['C', 'A', 'B', 'D', 'E', 'F']
>>> |
```

Output 1b



```
Python 3.4.4 (v3.4.4:737efcadf5a6, Dec 20 2015, 20:20:57) [MSC v.1600 64 bit (AM
D64)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> 
====================== RESTART: D:/sem 5/AI/pracs/1b.py ======================
{0, 1}
{0, 1, 2}
{0, 1, 2, 3}
>>> 
```

Practical 1

a) Write a program to implement depth first search algorithm.

Code:

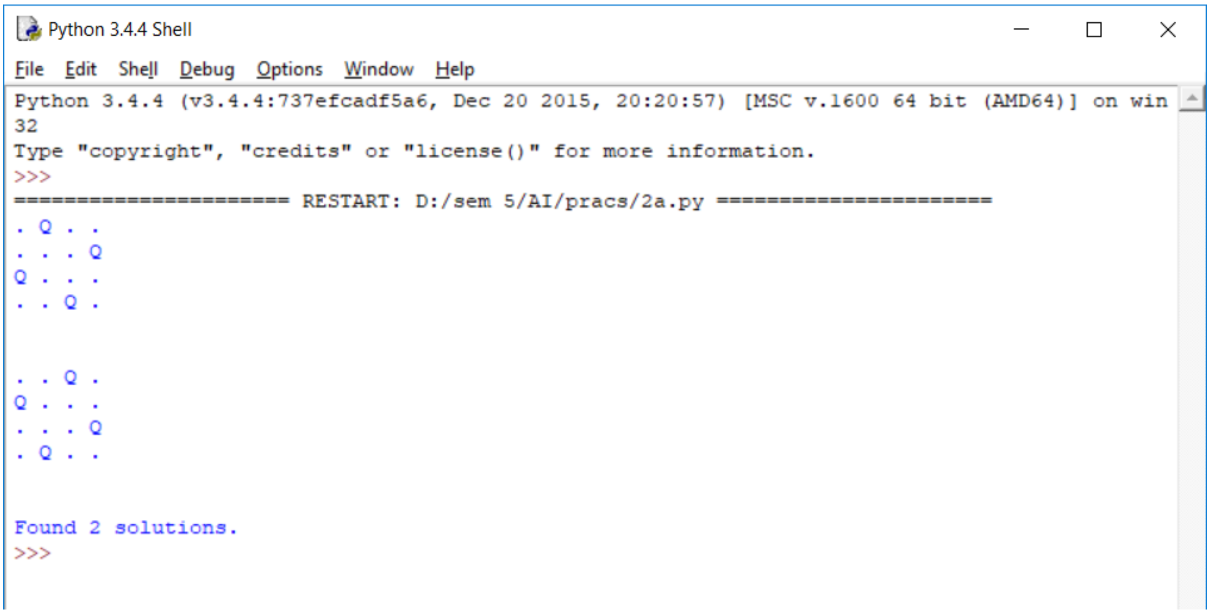
```
graph1 = {
   'A': set(['B', 'C']),
   'B': set(['A', 'D', 'E']),
   'C': set(['A', 'F']),
   'D': set(['B']),
   'E': set(['B', 'F']),
   'F': set(['C', 'E'])
   }
def dfs(graph, node, visited):
   if node not in visited:
      visited.append(node)
      for n in graph[node]:
         dfs(graph,n, visited)
   return visited
visited = dfs(graph1,'C', [])
print(visited)
```

b) Write a program to implement breadth first search algorithm.

Code:

```
#Implement BFS
visited=[]
import collections
def bfs(graph,root):
   visited,queue=set(),collections.deque([root])
   visited.add(root)
   while queue:
      vertex=queue.popleft()
      for neighbour in graph[vertex]:
         if neighbour not in visited:
            visited.add(neighbour)
            print(visited)
            queue.append(neighbour)
if __name__=='__main__':
   graph={0:[1,2],1:[2],2:[3],3:[1,2]}
   bfs(graph,0)
```

Output 2a



```
Python 3.4.4 (v3.4.4:737efcadf5a6, Dec 20 2015, 20:20:57) [MSC v.1600 64 bit (AMD64)] on win
32
Type "copyright", "credits" or "license()" for more information.
>>> 
====================== RESTART: D:/sem 5/AI/pracs/2a.py ======================
. Q . .
. . . Q
Q . . .
. . Q .


. . Q .
Q . . .
. . . Q
. Q . .


Found 2 solutions.
>>> 
```

Practical 2

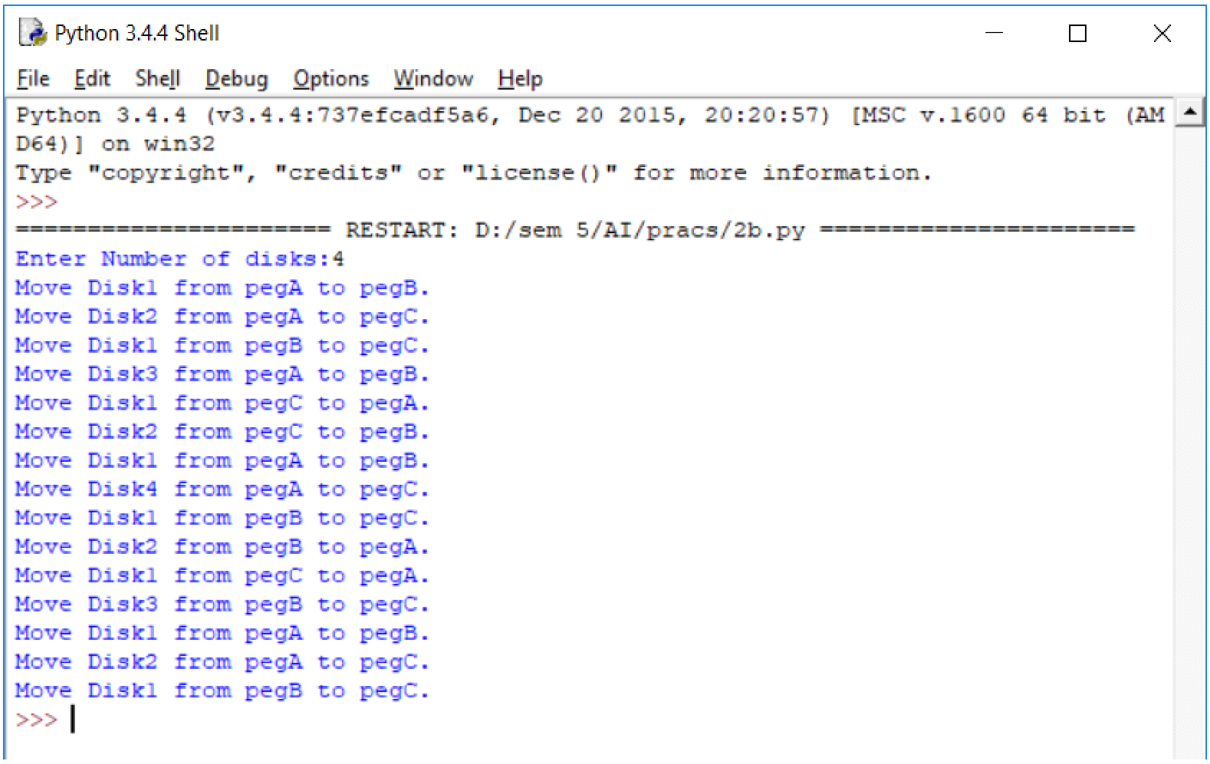
a) Write a program to simulate 4-Queen / N-Queen problem.

Code:

```
class NQueens:
    """Generate all valid solutions for the n queens puzzle"""
    def __init__(self, size):
        # Store the puzzle (problem) size and the number of valid solutions
        self.size = size
        self.solutions = 0
        self.solve()
    def solve(self):
        """Solve the n queens puzzle and print the number of solutions"""
        positions = [-1] * self.size
        self.put_queen(positions, 0)
        print("Found", self.solutions, "solutions.")
    def put_queen(self, positions, target_row):
        """Try to place a queen on target_row by checking all N possible cases.
         If a valid place is found the function calls itself trying to place a queen
         on the next row until all N queens are placed on the NxN board."""
       # Base (stop) case - all N rows are occupied
        if target_row == self.size:
            self.show_full_board(positions)
            self.solutions += 1
        else:
            # For all N columns positions try to place a queen
            for column in range(self.size):
                # Reject all invalid positions
                if self.check_place(positions, target_row, column):
                    positions[target_row] = column
                    self.put_queen(positions, target_row + 1)
    def check_place(self, positions, ocuppied_rows, column):
        """ Check if a given position is under attack from any of  the previously placed queens (check column
and diagonal positions)38     """
        for i in range(ocuppied_rows):
            if positions[i] == column or \
                positions[i] - i == column - ocuppied_rows or \
                positions[i] + i == column + ocuppied_rows:
```

```
 return False
        return True
    def show_full_board(self, positions):
        """Show the full NxN board"""
        for row in range(self.size):
            line = ""
            for column in range(self.size):
                if positions[row] == column:
                    line += "Q "
                else:
                    line += ". "
            print(line)
        print("\n")
def main():
    """Initialize and solve the n queens puzzle"""
    NQueens(2)
if __name__ == "__main__":
    # execute only if run as a script
    main()
```

Output 2b



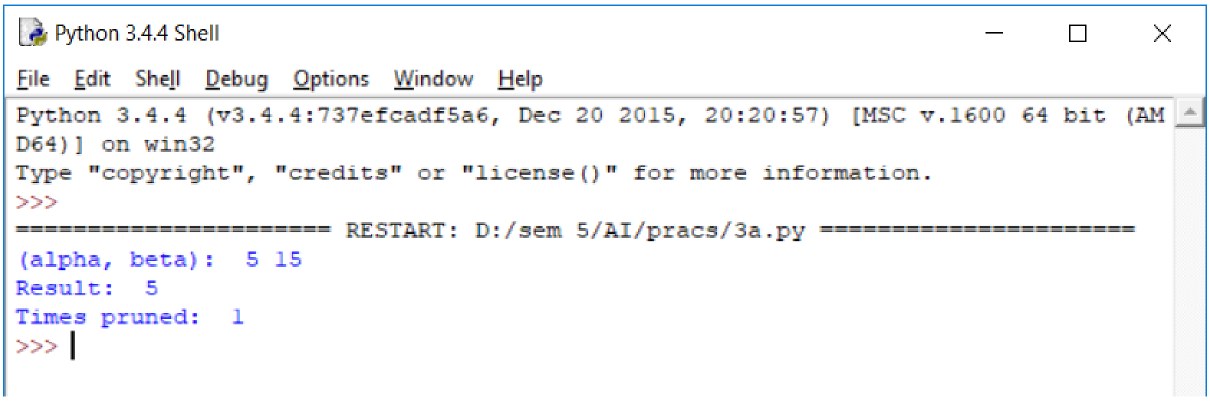
```
Python 3.4.4 (v3.4.4:737efcadf5a6, Dec 20 2015, 20:20:57) [MSC v.1600 64 bit (AM
D64)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> 
====================== RESTART: D:/sem 5/AI/pracs/2b.py ======================
Enter Number of disks:4
Move Disk1 from pegA to pegB.
Move Disk2 from pegA to pegC.
Move Disk1 from pegB to pegC.
Move Disk3 from pegA to pegB.
Move Disk1 from pegC to pegA.
Move Disk2 from pegC to pegB.
Move Disk1 from pegA to pegB.
Move Disk4 from pegA to pegC.
Move Disk1 from pegB to pegC.
Move Disk2 from pegB to pegA.
Move Disk1 from pegC to pegA.
Move Disk3 from pegB to pegC.
Move Disk1 from pegA to pegB.
Move Disk2 from pegA to pegC.
Move Disk1 from pegB to pegC.
>>> 
```

b) Write a program to solve tower of Hanoi problem.

Code:

```
#implement Tower of Hanoi

def hanoi(disks,source,auxillary,target):

    if disks ==1:

        print("Move Disk1 from peg{} to peg{}.".format(source,target))

        return

    hanoi(disks-1,source,target,auxillary)

    print("Move Disk{} from peg{} to peg{}.".format(disks,source,target))

    hanoi(disks-1,auxillary,source,target)

disks=(int(input("Enter Number of disks:")))

hanoi(disks,'A','B','C')
```

Output 3a



```
Python 3.4.4 (v3.4.4:737efcadf5a6, Dec 20 2015, 20:20:57) [MSC v.1600 64 bit (AM
D64)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> 
====================== RESTART: D:/sem 5/AI/pracs/3a.py ======================
(alpha, beta):  5 15
Result:  5
Times pruned:  1
>>> 
```

Practical 3

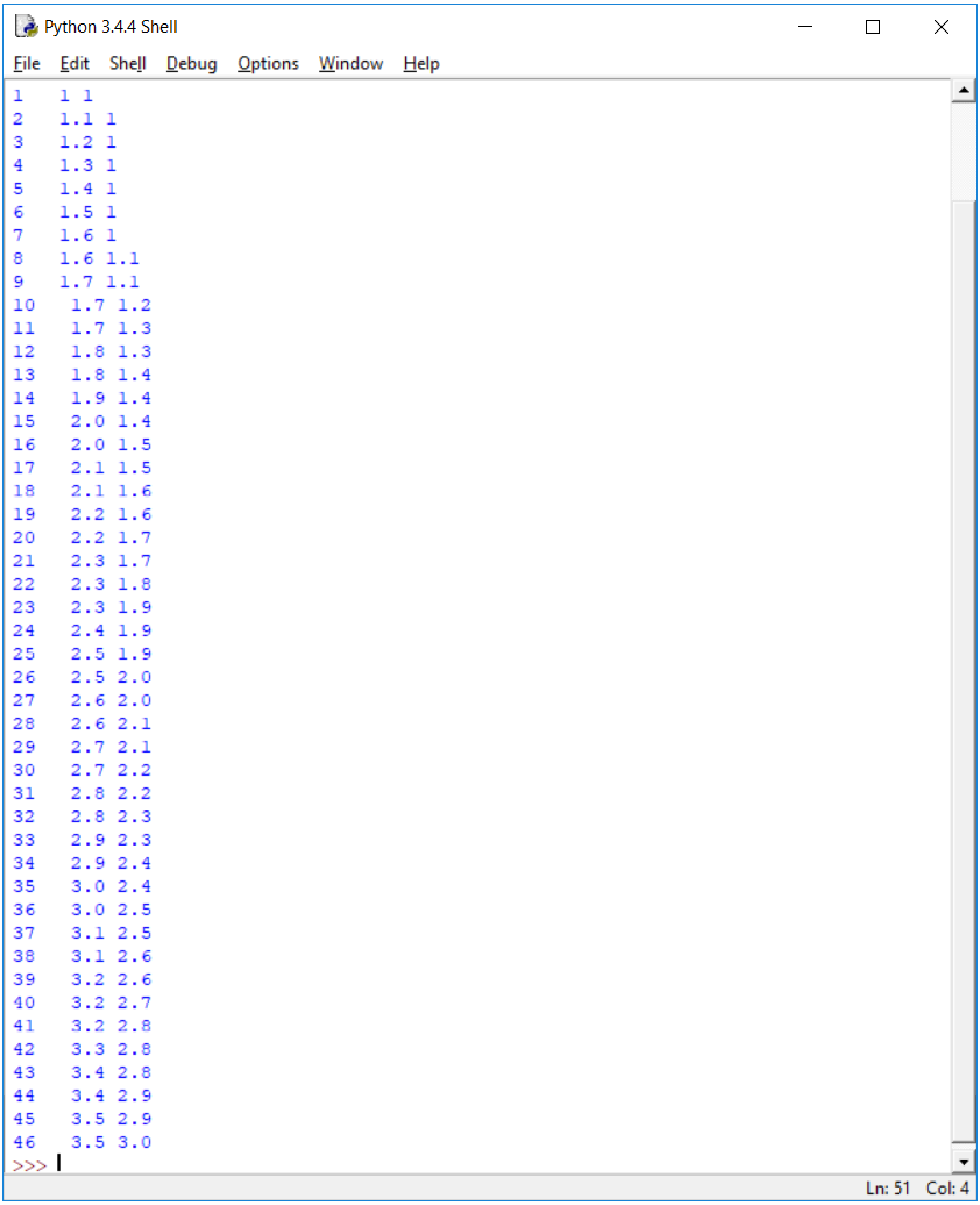
a) Write a program to implement alpha beta search.

Code:

```
tree = [[[5, 1, 2], [8, -8, -9]], [[9, 4, 5], [-3, 4, 3]]]

root = 0

pruned = 0

def children(branch, depth, alpha, beta):

   global tree

   global root

   global pruned

   i = 0

   for child in branch:

      if type(child) is list:

         (nalpha, nbeta) = children(child, depth + 1, alpha, beta)

         if depth % 2 == 1:

            beta = nalpha if nalpha < beta else beta

         else:

            alpha = nbeta if nbeta > alpha else alpha

         branch[i] = alpha if depth % 2 == 0 else beta

         i += 1

      else:

         if depth % 2 == 0 and alpha < child:

            alpha = child

         if depth % 2 == 1 and beta > child:

            beta = child

         if alpha >= beta:

            pruned += 1

            break

   if depth == root:

      tree = alpha if root == 0 else beta

   return (alpha, beta)

def alphabeta(in_tree=tree, start=root, upper=-15, lower=15):

   global tree

   global pruned

   global root

   (alpha, beta) = children(tree, start, upper, lower)

   if __name__ == "__main__":
```

```
 print ("(alpha, beta): ", alpha, beta)
        print ("Result: ", tree)
        print ("Times pruned: ", pruned)
    return (alpha, beta, tree, pruned)
if __name__ == "__main__":
    alphabeta(None)
```

Output 3b



```
1    1 1
2    1.1 1
3    1.2 1
4    1.3 1
5    1.4 1
6    1.5 1
7    1.6 1
8    1.6 1.1
9    1.7 1.1
10    1.7 1.2
11    1.7 1.3
12    1.8 1.3
13    1.8 1.4
14    1.9 1.4
15    2.0 1.4
16    2.0 1.5
17    2.1 1.5
18    2.1 1.6
19    2.2 1.6
20    2.2 1.7
21    2.3 1.7
22    2.3 1.8
23    2.3 1.9
24    2.4 1.9
25    2.5 1.9
26    2.5 2.0
27    2.6 2.0
28    2.6 2.1
29    2.7 2.1
30    2.7 2.2
31    2.8 2.2
32    2.8 2.3
33    2.9 2.3
34    2.9 2.4
35    3.0 2.4
36    3.0 2.5
37    3.1 2.5
38    3.1 2.6
39    3.2 2.6
40    3.2 2.7
41    3.2 2.8
42    3.3 2.8
43    3.4 2.8
44    3.4 2.9
45    3.5 2.9
46    3.5 3.0
>>> 
```

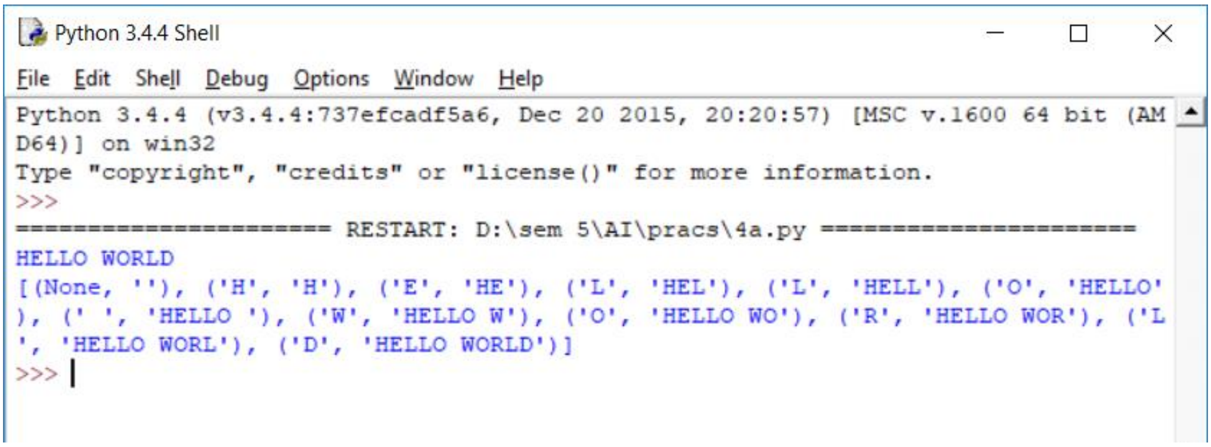
b) Write a program for Hill climbing problem.

Code:

```
import math
increment = 0.1
startingPoint = [1, 1]
point1 = [1,5]
point2 = [6,4]
point3 = [5,2]
point4 = [2,1]
def distance(x1, y1, x2, y2):
   dist = math.pow(x2-x1, 2) + math.pow(y2-y1, 2)
   return dist
def sumOfDistances(x1, y1, px1, py1, px2, py2, px3, py3, px4, py4):
   d1 = distance(x1, y1, px1, py1)
   d2 = distance(x1, y1, px2, py2)
   d3 = distance(x1, y1, px3, py3)
   d4 = distance(x1, y1, px4, py4)
   return d1 + d2 + d3 + d4
def newDistance(x1, y1, point1, point2, point3, point4):
   d1 = [x1, y1]
   d1temp = sumOfDistances(x1, y1, point1[0],point1[1], point2[0],point2[1],point3[0],point3[1],
point4[0],point4[1] )
   d1.append(d1temp)
   return d1
minDistance = sumOfDistances(startingPoint[0], startingPoint[1], point1[0],point1[1],
point2[0],point2[1],point3[0],point3[1], point4[0],point4[1] )
flag = True
def newPoints(minimum, d1, d2, d3, d4):
   if d1[2] == minimum:
      return [d1[0], d1[1]]
   elif d2[2] == minimum:
      return [d2[0], d2[1]]
   elif d3[2] == minimum:
      return [d3[0], d3[1]]
   elif d4[2] == minimum:
      return [d4[0], d4[1]]
I = 1
```

```
while flag:
  d1 = newDistance(startingPoint[0]+increment, startingPoint[1], point1, point2, point3, point4)
  d2 = newDistance(startingPoint[0]-increment, startingPoint[1], point1, point2, point3, point4)
  d3 = newDistance(startingPoint[0], startingPoint[1]+increment, point1, point2, point3, point4)
  d4 = newDistance(startingPoint[0], startingPoint[1]-increment, point1, point2, point3, point4)
  print (I,’ ‘, round(startingPoint[0], 2), round(startingPoint[1], 2))
  minimum = min(d1[2], d2[2], d3[2], d4[2])
  if minimum < minDistance:
    startingPoint = newPoints(minimum, d1, d2, d3, d4)
    minDistance = minimum
    #print I,’ ‘, round(startingPoint[0], 2), round(startingPoint[1], 2)
    i+=1
  else:
    flag = False
```

Output 4a



```
Python 3.4.4 (v3.4.4:737efcadf5a6, Dec 20 2015, 20:20:57) [MSC v.1600 64 bit (AM
D64)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> 
====================== RESTART: D:\sem 5\AI\pracs\4a.py ======================
HELLO WORLD
[(None, ''), ('H', 'H'), ('E', 'HE'), ('L', 'HEL'), ('L', 'HELL'), ('O', 'HELLO'
), (' ', 'HELLO '), ('W', 'HELLO W'), ('O', 'HELLO WO'), ('R', 'HELLO WOR'), ('L
', 'HELLO WORL'), ('D', 'HELLO WORLD')]
>>> 
```

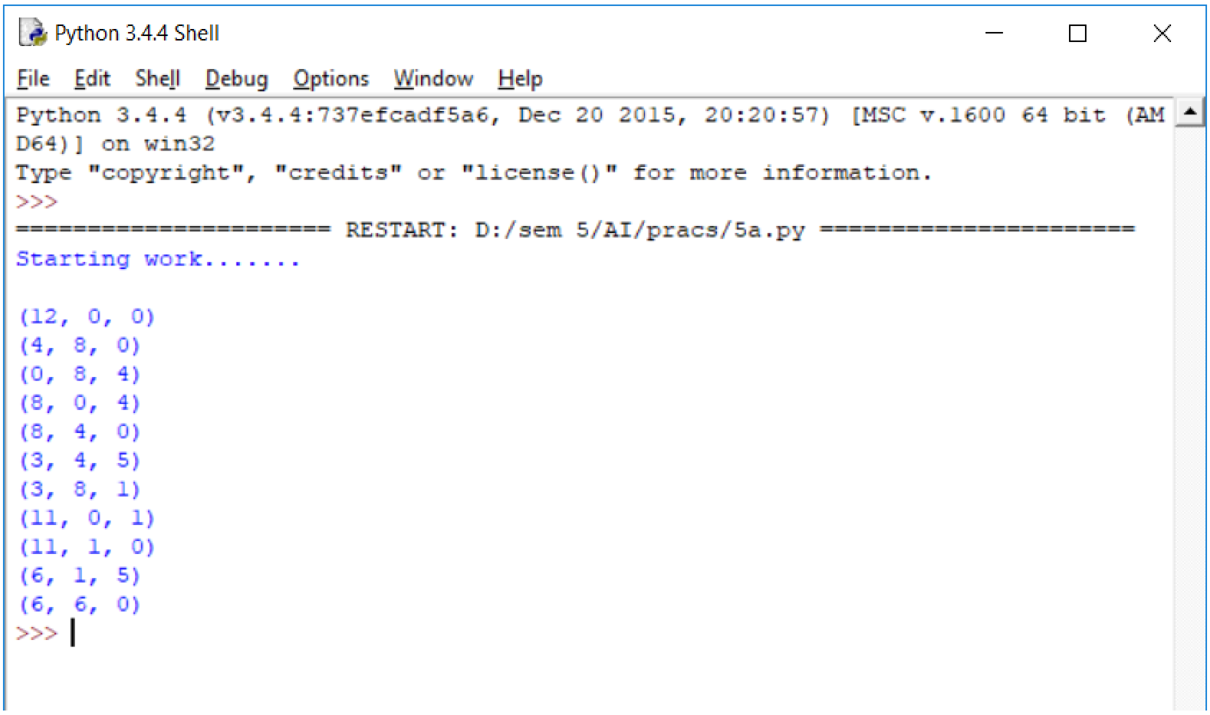
Practical 4

a) Write a program to implement A* algorithm.

Code:

```
from simpleai.search import SearchProblem, astar

GOAL = 'HELLO WORLD'
class HelloProblem(SearchProblem):
    def actions(self, state):
        if len(state) < len(GOAL):
            return list(' ABCDEFGHIJKLMNOPQRSTUVWXYZ')
        else:
            return []

    def result(self, state, action):
        return state + action
    def is_goal(self, state):
        return state == GOAL

    def heuristic(self, state):
        # how far are we from the goal?
        wrong = sum([1 if state[i] != GOAL[i] else 0
                     for i in range(len(state))])
        missing = len(GOAL) - len(state)
        return wrong + missing

problem = HelloProblem(initial_state='')
result = astar(problem)
print(result.state)
print(result.path())
```

Output 5a



```
Python 3.4.4 (v3.4.4:737efcadf5a6, Dec 20 2015, 20:20:57) [MSC v.1600 64 bit (AM
D64)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> 
====================== RESTART: D:/sem 5/AI/pracs/5a.py ======================
Starting work.......

(12, 0, 0)
(4, 8, 0)
(0, 8, 4)
(8, 0, 4)
(8, 4, 0)
(3, 4, 5)
(3, 8, 1)
(11, 0, 1)
(11, 1, 0)
(6, 1, 5)
(6, 6, 0)
>>> 
```

Practical 5

a) Write a program to solve water jug problem.

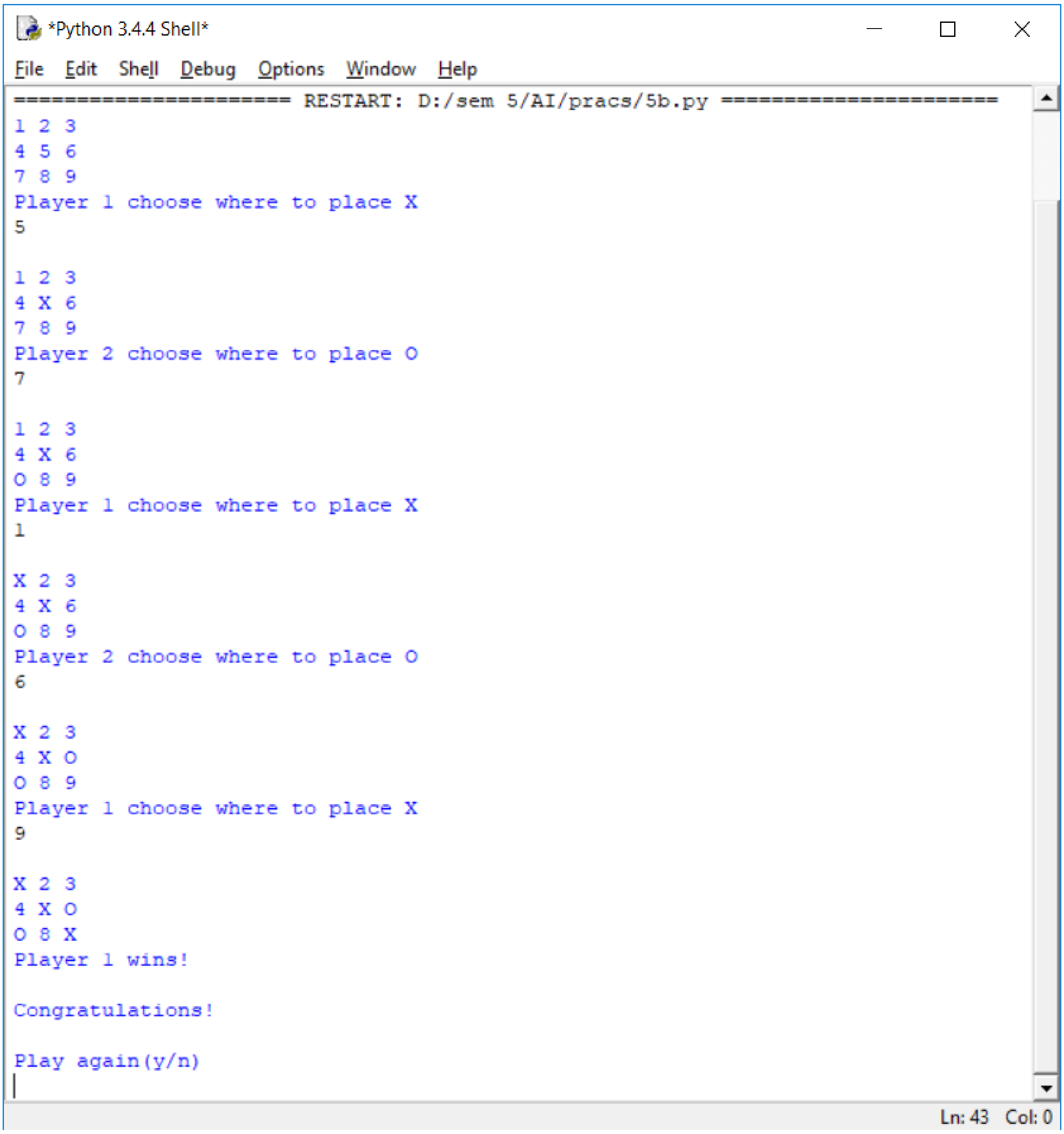
Code:

```
capacity=(12,8,5)
#max capacity of 3 jugs->x,y,z
x=capacity[0]
y=capacity[1]
z=capacity[2]
#to mark visisted states
memory={}
#store solution path
ans=[]
def get_all_states(state):
    #let 3 jugs be called a,b,c
    a=state[0]
    b=state[1]
    c=state[2]
    if(a==6 and b==6):
        ans.append(state)
        return True
    #if current state is already visted earlier
    if((a,b,c )in memory):
        return False
    memory[(a,b,c)]=1
    #empty jug a
    if(a>0):
        #empty a into b
        if((a+b)<=y):
            if(get_all_states((0,a+b,c))):
                ans.append(state)
                return True
        else:
            if(get_all_states((a-(y-b),y,c))):
                ans.append(state)
                return True
        #empty a into c
```

```
  if(a+c<=z):
        if(get_all_states((0,b,a+c))):
            ans.append(state)
            return True
    else:
        if(get_all_states((a-(z-c),b,z))):
            ans.append(state)
            return True
#empty b
if(b>0):
    #empty b into a
    if((a+b)<=x):
        if(get_all_states((a+b,0,c))):
            ans.append(state)
            return True
    else:
        if(get_all_states((x,b-(x-a),c))):
            ans.append(state)
            return True
    #empty b into c
    if(b+c<=z):
        if(get_all_states((a,0,b+c))):
            ans.append(state)
            return True
    else:
        if(get_all_states((a,b-(z-c),z))):
            ans.append(state)
            return True
#empty c
if(c>0):
    #empty c into a
    if((a+c)<=x):
        if(get_all_states((a+c,b,0))):
            ans.append(state)
            return True
```

```
else:
            if(get_all_states((x,b,c-(x-a)))):
                ans.append(state)
                return True
        #empty c into b
        if(b+c<=y):
            if(get_all_states((a,b+c,0))):
                ans.append(state)
                return True
        else:
            if(get_all_states((a,y,c-(y-b)))):
                ans.append(state)
                return True
    return False
initial_state=(12,0,0)
print('Starting work.......\n')
get_all_states(initial_state)
ans.reverse()
for i in ans:
    print (i)
```

Output 5b



```
====================== RESTART: D:/sem 5/AI/pracs/5b.py ======================
1 2 3
4 5 6
7 8 9
Player 1 choose where to place X
5

1 2 3
4 X 6
7 8 9
Player 2 choose where to place O
7

1 2 3
4 X 6
O 8 9
Player 1 choose where to place X
1

X 2 3
4 X 6
O 8 9
Player 2 choose where to place O
6

X 2 3
4 X O
O 8 9
Player 1 choose where to place X
9

X 2 3
4 X O
O 8 X
Player 1 wins!

Congratulations!

Play again(y/n)
```

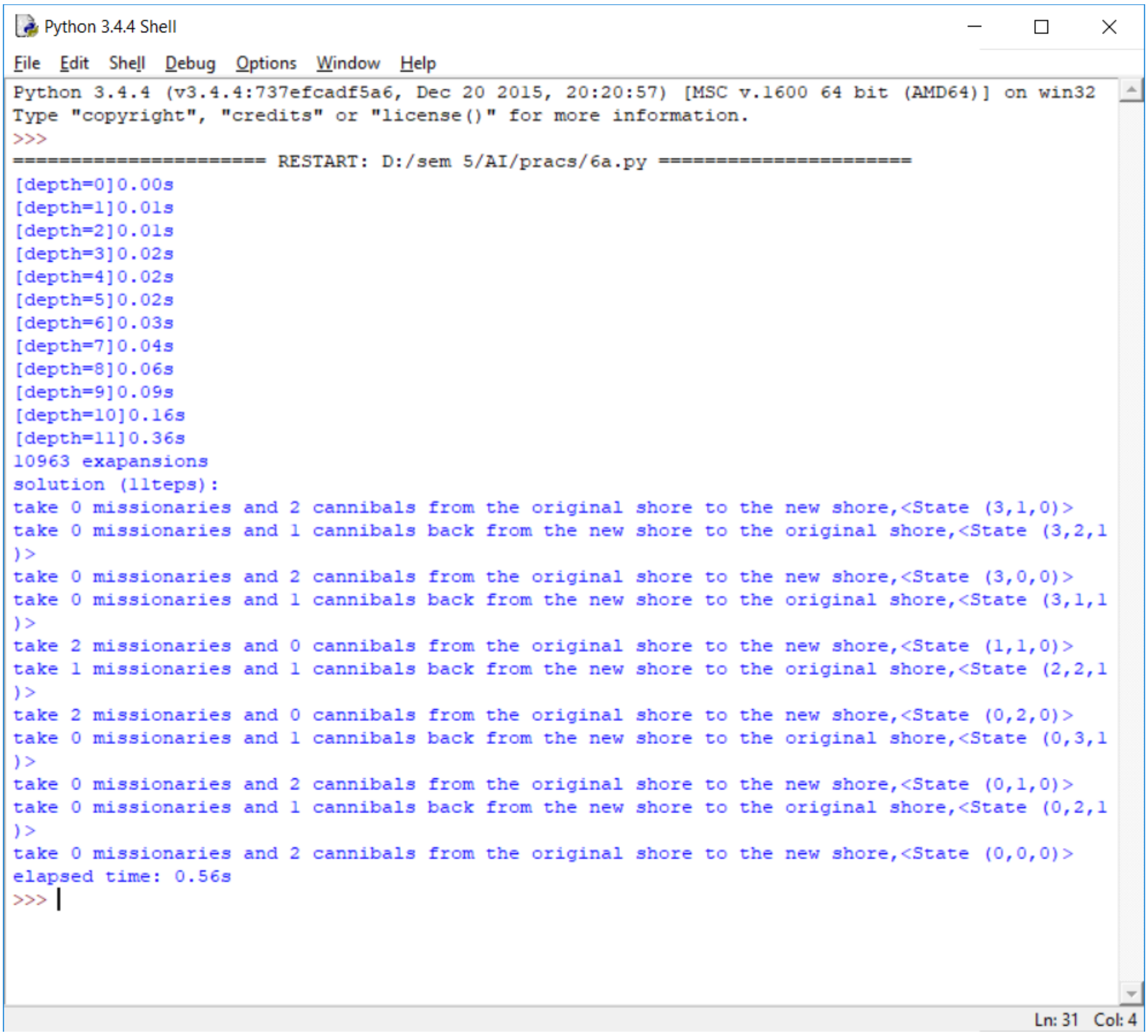
b) Design the simulation of tic – tac – toe game using min-max algorithm.

Code:

```
def tic_tac_toe():
    #board=b
    b=[1,2,3,4,5,6,7,8,9]
    end=False
    win_combinations=((0,1,2),(3,4,5),(6,7,8),(0,3,6),(1,4,7),(2,5,8),(0,4,8),(2,4,6))
    def draw():
        print(b[0],b[1],b[2])
        print(b[3],b[4],b[5])
        print(b[6],b[7],b[8])
    def p1():
        n=choose_number()
        if b[n]=="X" or b[n]=="O":
            print("\nYou cant go there.Try again")
            p1()
        else:
            b[n]="X"
    def p2():
        n=choose_number()
        if b[n]=="X" or b[n]=="O":
            print("\nYou cant go there.Try again")
            p2()
        else:
            b[n]="O"
    def choose_number():
        while True:
            while True:
                a=input()
                try: a=int( a)
                    a-=1
                    if a in range(0,9):
                        return a
                    else: print("\nThat is not on board. Try again")
                        continue
                    except ValueError:
```

```
                         print("\nThat is not a number.Try again")
                        continue
    def check_board():
          count=0
          for a in win_combinations:
                if b[a[0]]==b[a[1]]==b[a[2]]=="X":
                      print("Player 1 wins!")
                      print("\nCongratulations!\n")
                      return True
                if b[a[0]]==b[a[1]]==b[a[2]]=="O":
                      print("Player 2 wins!")
                      print("\nCongratulations!\n")
                      return True
          for a in range(9):
                if b[a]=="X" or b[a]=="O":
                      count+=1
                if count==9:
                      print("The game ends in a tie\n")
                      return True
    while not end:
          draw()
          end=check_board()
          if end==True:
                break
          print("Player 1 choose where to place X")
          p1()
          print()  draw()
          end=check_board()
          if end==True:
                break
          print("Player 2 choose where to place O")
          p2()
          print()
    if input("Play again(y/n) \n")=="y": print()
     tic_tac_toe()
tic_tac_toe()
```

Output 6a



```
Python 3.4.4 (v3.4.4:737efcadf5a6, Dec 20 2015, 20:20:57) [MSC v.1600 64 bit (AMD64)] on win32
Type "copyright", "credits" or "license()" for more information.
>>>
====================== RESTART: D:/sem 5/AI/pracs/6a.py ======================
[depth=0]0.00s
[depth=1]0.01s
[depth=2]0.01s
[depth=3]0.02s
[depth=4]0.02s
[depth=5]0.02s
[depth=6]0.03s
[depth=7]0.04s
[depth=8]0.06s
[depth=9]0.09s
[depth=10]0.16s
[depth=11]0.36s
10963 exapansions
solution (11teps):
take 0 missionaries and 2 cannibals from the original shore to the new shore,<State (3,1,0)>
take 0 missionaries and 1 cannibals back from the new shore to the original shore,<State (3,2,1
)>
take 0 missionaries and 2 cannibals from the original shore to the new shore,<State (3,0,0)>
take 0 missionaries and 1 cannibals back from the new shore to the original shore,<State (3,1,1
)>
take 2 missionaries and 0 cannibals from the original shore to the new shore,<State (1,1,0)>
take 1 missionaries and 1 cannibals back from the new shore to the original shore,<State (2,2,1
)>
take 2 missionaries and 0 cannibals from the original shore to the new shore,<State (0,2,0)>
take 0 missionaries and 1 cannibals back from the new shore to the original shore,<State (0,3,1
)>
take 0 missionaries and 2 cannibals from the original shore to the new shore,<State (0,1,0)>
take 0 missionaries and 1 cannibals back from the new shore to the original shore,<State (0,2,1
)>
take 0 missionaries and 2 cannibals from the original shore to the new shore,<State (0,0,0)>
elapsed time: 0.56s
>>>
```

Practical 6

a) Write a program to solve Missionaries and Cannibals problem.

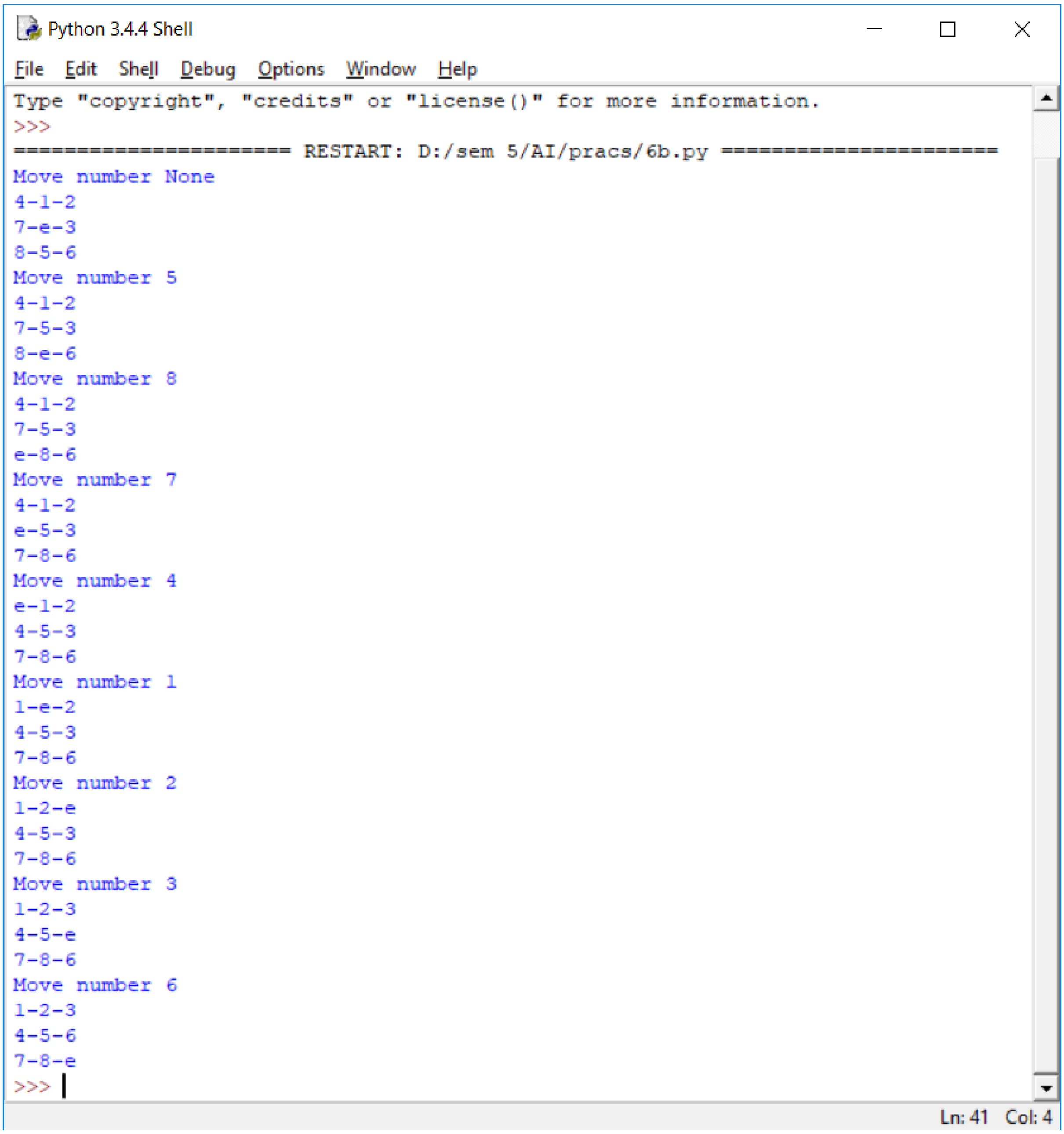
Code:

```
from copy import deepcopy
from collections import deque
import sys
import time
class State(object):
    def __init__(self,missionaries,cannibals,boats):
        self.missionaries=missionaries
        self.cannibals=cannibals
        self.boats=boats
    def successors(self):
        if self.boats==1:
            sgn=-1
            direction="from the original shore to the new shore"
        else:
            sgn=1
            direction="back from the new shore to the original shore"
        for m in range(3):
            for c in range(3):
                newState=State(self.missionaries+sgn*m,self.cannibals+sgn*c,self.boats+sgn*1)
                if m+c>=1 and m+c<=2 and newState.isValid():
                    action="take %d missionaries and %d cannibals %s,%r"%(m,c,direction,newState)
                    yield action,newState
    def isValid(self):
        if self.missionaries<0 or self.cannibals<0 or self.missionaries>3 or self.cannibals>3 or (self.boats!=0 and self.boats!=1):
            return False
        if self.cannibals>self.missionaries and self.missionaries>0:
            return False
        if self.cannibals<self.missionaries and self.missionaries<3:
            return False
        return True
    def is_goal_state(self):
        return self.cannibals==0 and self.missionaries==0 and self.boats==0
```

```
def __repr__(self):
        return "<State (%d,%d,%d)>"%(self.missionaries,self.cannibals,self.boats)
class Node(object):
    def __init__(self,parent_node,state,action,depth):
        self.parent_node=parent_node
        self.state=state
        self.action=action
        self.depth=depth
    def expand(self):
        for(action,succ_state) in self.state.successors():
            succ_node=Node(parent_node=self,state=succ_state,action=action,depth=self.depth+1)
            yield succ_node
    def extract_solution(self):
        solution=[]
        node=self
        while node.parent_node is not None:
            solution.append(node.action)
            node=node.parent_node
        solution.reverse()
        return solution
def breadth_first_tree_search(initial_state):
    initial_node=Node(parent_node=None,state=initial_state,action=None,depth=0)
    fifo=deque([initial_node])
    num_expansions=0
    max_depth=-1
    while True:
        if not fifo:
            print("%d expansions"%num_expansions)
            return None
        node=fifo.popleft()
        if node.depth>max_depth:
            max_depth=node.depth
            print("[depth=%d]%.2fs"%(max_depth,time.clock()))
        if node.state.is_goal_state():
            print("%d exapansions"%num_expansions)
            solution=node.extract_solution()
```

```
            return solution
         num_expansions+=1
         fifo.extend(node.expand())
def  usage():
    print >> sys.stderr,"usage:"
    print >> sys.stderr," %s"% sys.argv[0]
    raise SystemExit(2)
def main():
    initial_state=State(3,3,1)
    solution=breadth_first_tree_search(initial_state)
    if solution is None:
        print ("no solution")
    else:
        print ("solution (%steps):" % len(solution))
        for step in solution:
            print ("%s" % step)
        print ("elapsed time: %.2fs" % time.clock())
if __name__=="__main__":
    main()
```

Output 6b



```
Type "copyright", "credits" or "license()" for more information.
>>> 
====================== RESTART: D:/sem 5/AI/pracs/6b.py ======================
Move number None
4-1-2
7-e-3
8-5-6
Move number 5
4-1-2
7-5-3
8-e-6
Move number 8
4-1-2
7-5-3
e-8-6
Move number 7
4-1-2
e-5-3
7-8-6
Move number 4
e-1-2
4-5-3
7-8-6
Move number 1
1-e-2
4-5-3
7-8-6
Move number 2
1-2-e
4-5-3
7-8-6
Move number 3
1-2-3
4-5-e
7-8-6
Move number 6
1-2-3
4-5-6
7-8-e
>>> 
```

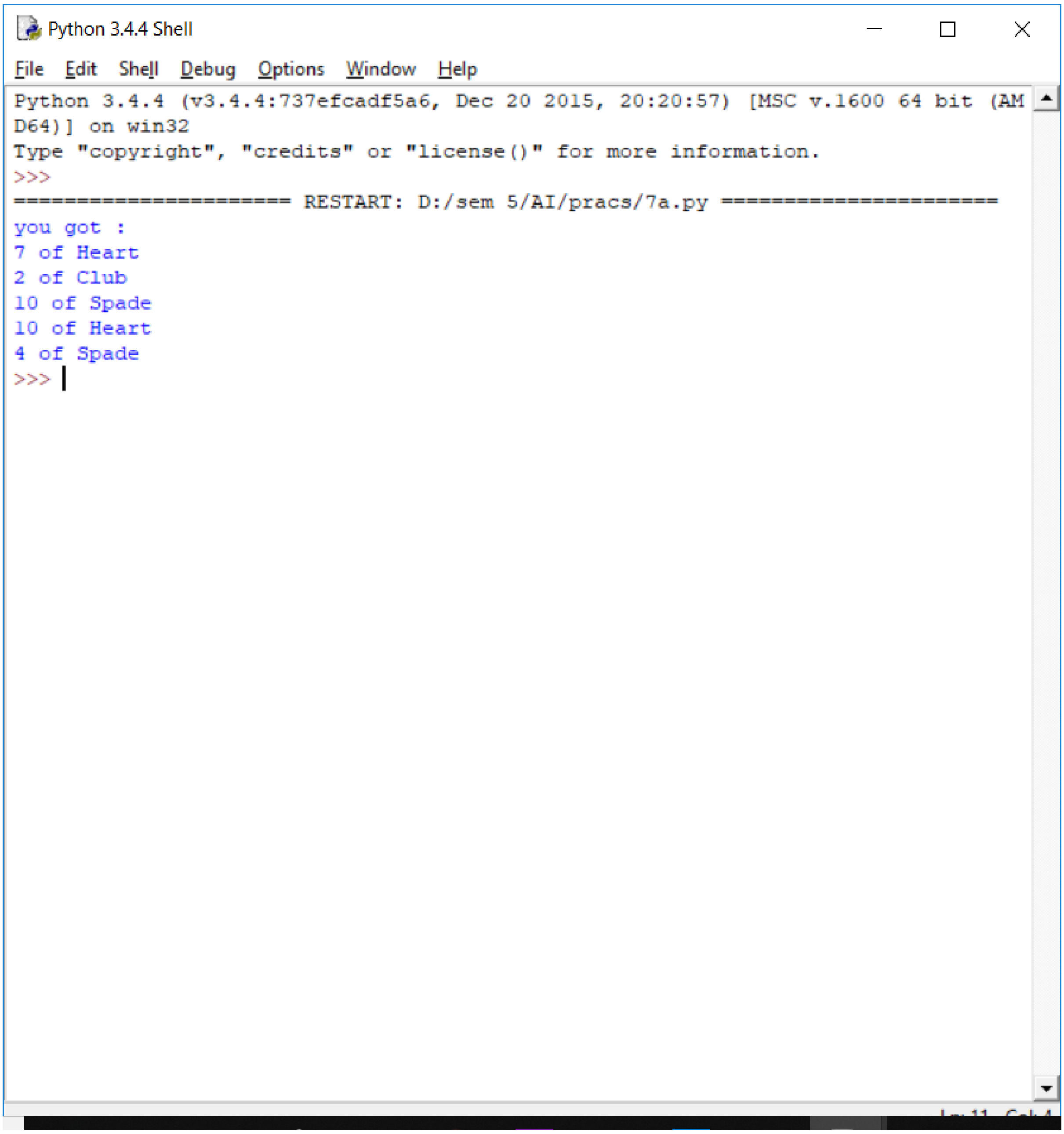
b) Design an application to simulate number puzzle problem.

Code:

```
from __future__ import print_function
from simpleai.search import astar,SearchProblem
from simpleai.search.viewers import WebViewer
GOAL = '''1-2-3
4-5-6
7-8-e'''
INITIAL='''4-1-2
7-e-3
8-5-6'''
def list_to_string(list_):
    return '\n'.join(['-'.join(row) for row in list_])
def string_to_list(string_):
    return [row.split('-') for row in string_.split('\n')]
def find_location(rows,element_to_find):
    '''Find the location of a piece in the puzzle
      Returns a tuple:row,column'''
    for ir,row in enumerate(rows):
        for ic,element in enumerate(row):
            if element==element_to_find:
                return ir,ic
#we create a cache for the goal position of each piece so we dont have to recalculate every time
goal_positions={}
rows_goal=string_to_list(GOAL)
for number in '12345678e':
    goal_positions[number]=find_location(rows_goal,number)
class EightPuzzleProblem(SearchProblem):
    def actions(self,state):
        '''Returns a list of pieces we can move to the empty space.'''
        rows=string_to_list(state)
        row_e,col_e=find_location(rows,'e')
        actions=[]
        if row_e>0:
            actions.append(rows[row_e-1][col_e])
        if row_e<2:
```

```
        actions.append(rows[row_e+1][col_e])
        if col_e>0:
            actions.append(rows[row_e][col_e-1])
        if col_e<2:
            actions.append(rows[row_e][col_e+1])
        return actions
        def result(self,state,action):
        '''Return the resulting state after moving a piece to the empty space(the "action" parameter contains the piece to move)'''
        rows=string_to_list(state)
        row_e,col_e=find_location(rows,'e')
        row_n,col_n=find_location(rows,action)
        rows[row_e][col_e],rows[row_n][col_n]=rows[row_n][col_n],rows[row_e][col_e]
        return list_to_string(rows)
    def is_goal(self,state):
        '''Returns true if a state is the goal state.'''
        return state==GOAL
    def cost(self,state1,action,state2):
        '''Returns the cost of performing an action. Not useful in this problem but needed.'''
        return 1
    def heuristic(self,state):
        '''Returns an *estimation* of the distance from a state to the goal.We are using the manhattan distance.'''
        rows=string_to_list(state)
        distance=0
        for number in '12345678e':
            row_n,col_n=find_location(rows,number)
            row_n_goal,col_n_goal=goal_positions[number]
            distance+=abs(row_n-row_n_goal)+abs(col_n-col_n_goal)
            return distance
result=astar(EightPuzzleProblem(INITIAL))
for action,state in result.path():
    print('Move number',action)
    print(state)
```

Output 7a



```
Python 3.4.4 (v3.4.4:737efcadf5a6, Dec 20 2015, 20:20:57) [MSC v.1600 64 bit (AM
D64)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> 
====================== RESTART: D:/sem 5/AI/pracs/7a.py ======================
you got :
7 of Heart
2 of Club
10 of Spade
10 of Heart
4 of Spade
>>> 
```

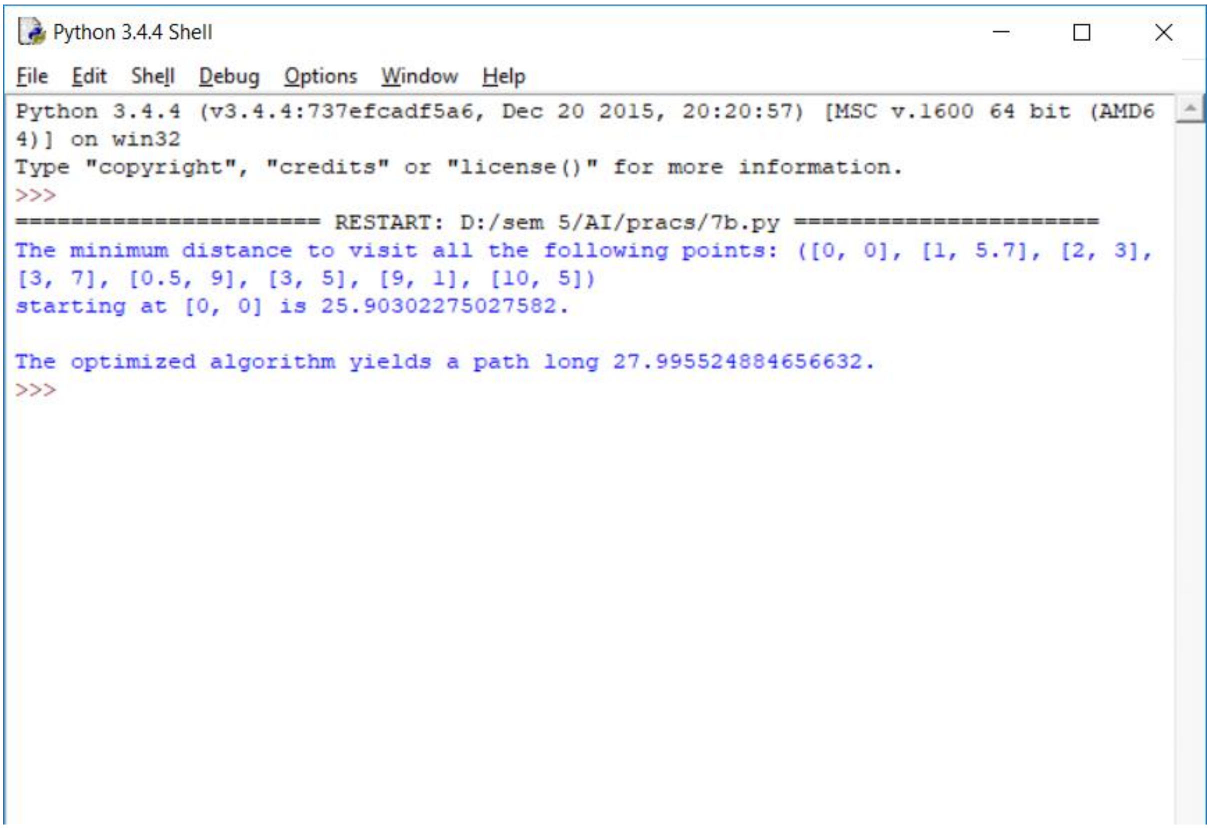
Practical 7

a) Write a program to shuffle Deck of cards.

Code:

```
#write program to shuffle deck of cards
#import modules
import itertools,random
#make a deck of cards
deck=list(itertools.product(range(1,14),['Spade','Heart','Diamond','Club']))
#shuffle cards
random.shuffle(deck)
#draw 5 cards
print("you got : ")
for i in range(5):
    print(deck[i][0],"of",deck[i][1])
```

Output 7b



```
Python 3.4.4 (v3.4.4:737efcadf5a6, Dec 20 2015, 20:20:57) [MSC v.1600 64 bit (AMD6
4)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> 
====================== RESTART: D:/sem 5/AI/pracs/7b.py ======================
The minimum distance to visit all the following points: ([0, 0], [1, 5.7], [2, 3],
[3, 7], [0.5, 9], [3, 5], [9, 1], [10, 5])
starting at [0, 0] is 25.90302275027582.

The optimized algorithm yields a path long 27.995524884656632.
>>> 
```

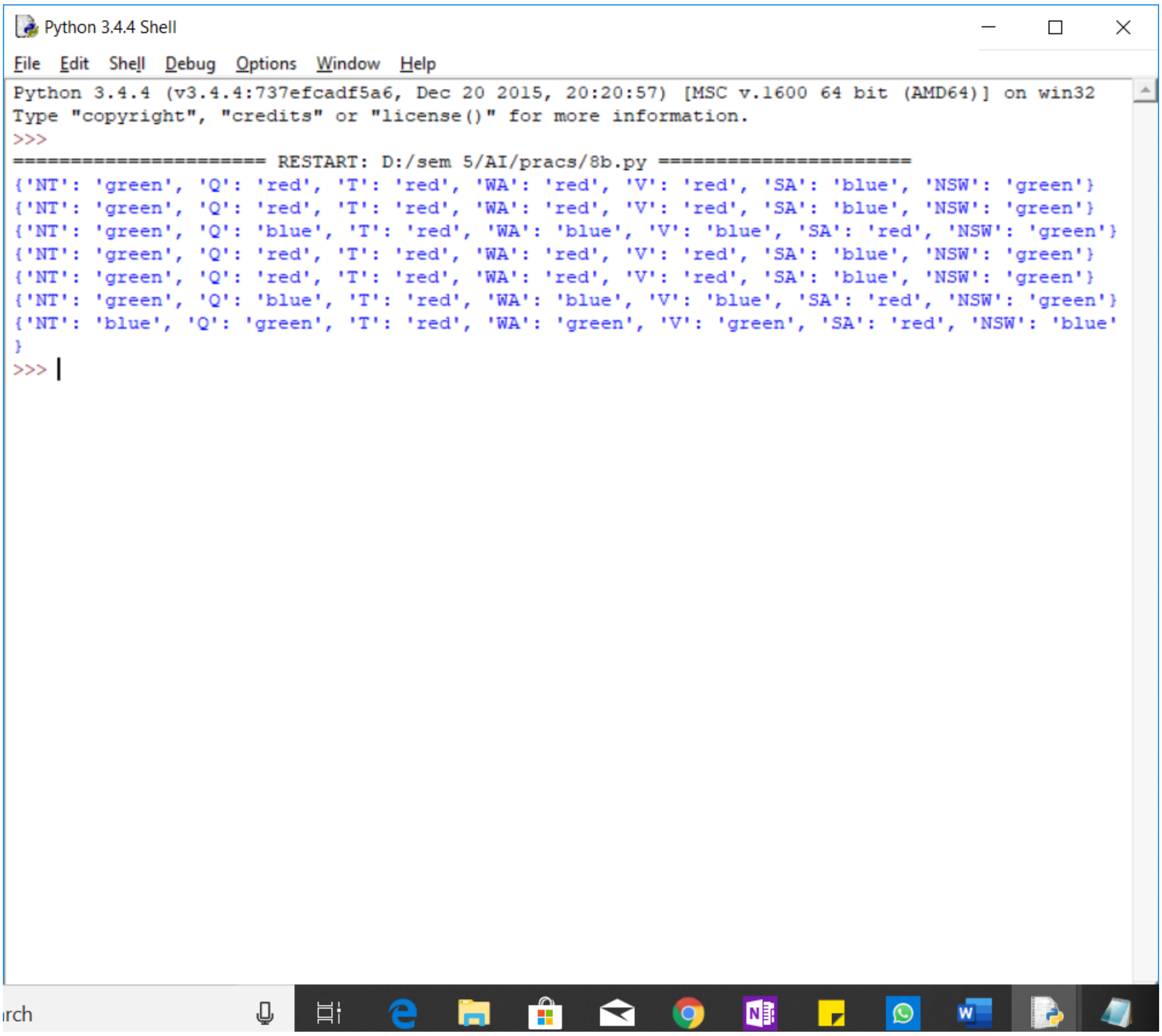
b) Solve traveling salesman problem using artificial intelligence technique.

Code:

```
import doctest
from itertools import permutations
def distance(point1, point2):
    """ Returns the Euclidean distance of two points in the Cartesian Plane.
    >>> distance([3,4],[0,0])
    5.0
    >>> distance([3,6],[10,6])
    7.0
    """    return ((point1[0] - point2[0])**2 + (point1[1] - point2[1])**2) ** 0.5
def total_distance(points):
    """ Returns the length of the path passing throught all the points in the given order.
    >>> total_distance([[1,2],[4,6]])
    5.0
    >>> total_distance([[3,6],[7,6],[12,6]])
    9.0
    """    return sum([distance(point, points[index + 1]) for index, point in enumerate(points[:-1])])
def travelling_salesman(points, start=None):
    """    Finds the shortest route to visit all the cities by bruteforce. Time complexity is O(N!), so never use on long lists.

    >>> travelling_salesman([[0,0],[10,0],[6,0]])
    ([0, 0], [6, 0], [10, 0])
    >>> travelling_salesman([[0,0],[6,0],[2,3],[3,7],[0.5,9],[3,5],[9,1]])
    ([0, 0], [6, 0], [9, 1], [2, 3], [3, 5], [3, 7], [0.5, 9])
    """
    if start is None:
        start = points[0]
    return min([perm for perm in permutations(points) if perm[0] == start], key=total_distance)
def optimized_travelling_salesman(points, start=None):
    """
    As solving the problem in the brute force way is too slow,
    this function implements a simple heuristic: always
    go to the nearest city.

    Even if this algoritmh is extremely simple, it works pretty well
```

```
giving a solution only about 25% longer than the optimal one (cit. Wikipedia),
    and runs very fast in O(N^2) time complexity.

    >>> optimized_travelling_salesman([[i,j] for i in range(5) for j in range(5)])
    [[0, 0], [0, 1], [0, 2], [0, 3], [0, 4], [1, 4], [1, 3], [1, 2], [1, 1], [1, 0], [2, 0], [2, 1], [2, 2], [2, 3], [2, 4], [3, 4], [3, 3], [3, 2], [3, 1], [3, 0], [4, 0], [4, 1], [4, 2], [4, 3], [4, 4]]
    >>> optimized_travelling_salesman([[0,0],[10,0],[6,0]])
    [[0, 0], [6, 0], [10, 0]]
    """
    if start is None:
        start = points[0]
    must_visit = points
    path = [start]
    must_visit.remove(start)
    while must_visit:
        nearest = min(must_visit, key=lambda x: distance(path[-1], x))
        path.append(nearest)
        must_visit.remove(nearest)
    return path
def main():
    doctest.testmod()
    points = [[0, 0], [1, 5.7], [2, 3], [3, 7],
              [0.5, 9], [3, 5], [9, 1], [10, 5]]
    print("""The minimum distance to visit all the following points: {}
starting at {} is {}.

The optimized algorithm yields a path long {}.""".format(
        tuple(points),
        points[0],
        total_distance(travelling_salesman(points)),
        total_distance(optimized_travelling_salesman(points))))

if __name__ == "__main__":
    main()
```

Output 8a



```
Python 3.4.4 (v3.4.4:737efcadf5a6, Dec 20 2015, 20:20:57) [MSC v.1600 64 bit (AMD64)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> 
====================== RESTART: D:/sem 5/AI/pracs/8b.py ======================
{'NT': 'green', 'Q': 'red', 'T': 'red', 'WA': 'red', 'V': 'red', 'SA': 'blue', 'NSW': 'green'}
{'NT': 'green', 'Q': 'red', 'T': 'red', 'WA': 'red', 'V': 'red', 'SA': 'blue', 'NSW': 'green'}
{'NT': 'green', 'Q': 'blue', 'T': 'red', 'WA': 'blue', 'V': 'blue', 'SA': 'red', 'NSW': 'green'}
{'NT': 'green', 'Q': 'red', 'T': 'red', 'WA': 'red', 'V': 'red', 'SA': 'blue', 'NSW': 'green'}
{'NT': 'green', 'Q': 'red', 'T': 'red', 'WA': 'red', 'V': 'red', 'SA': 'blue', 'NSW': 'green'}
{'NT': 'green', 'Q': 'blue', 'T': 'red', 'WA': 'blue', 'V': 'blue', 'SA': 'red', 'NSW': 'green'}
{'NT': 'blue', 'Q': 'green', 'T': 'red', 'WA': 'green', 'V': 'green', 'SA': 'red', 'NSW': 'blue'
}
>>> |
```

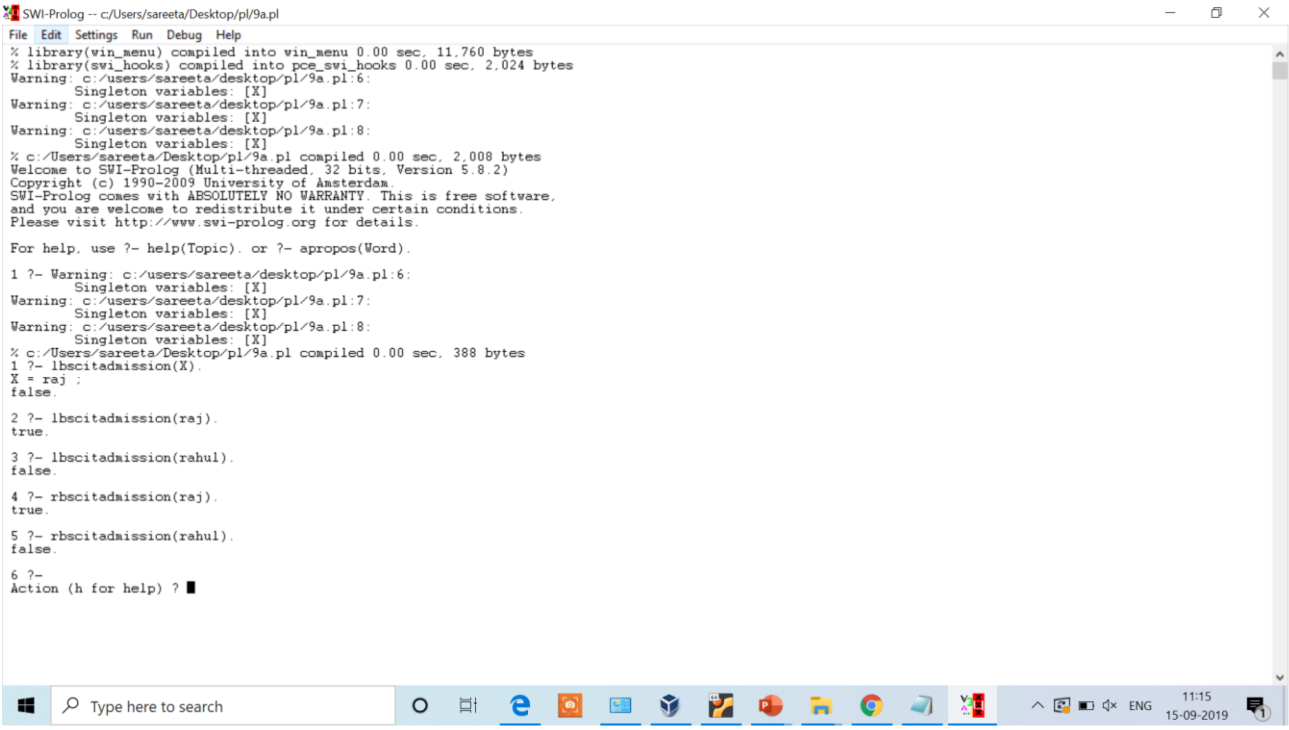
Practical 8

a) Solve constraint satisfaction problem

Code:

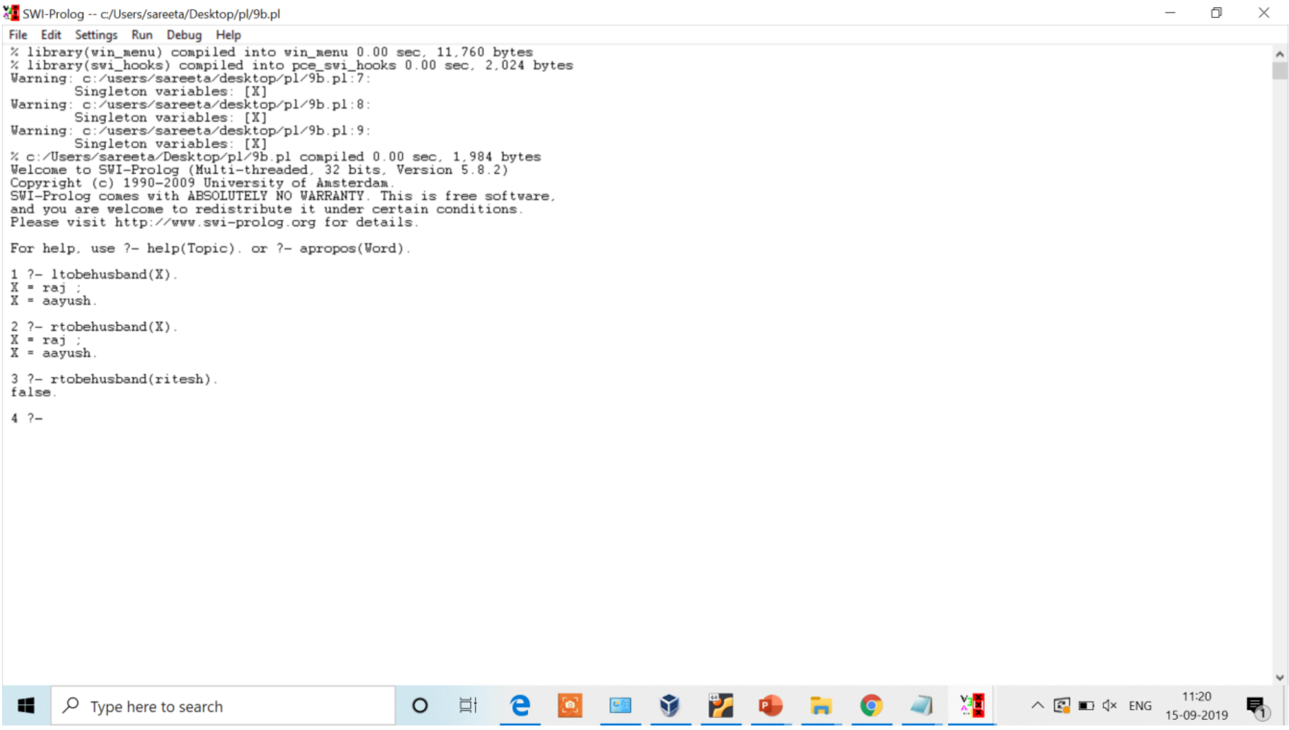
```
from __future__ import print_function
from simpleai.search import (CspProblem, backtrack,
                        min_conflicts,MOST_CONSTRAINED_VARIABLE,
                        HIGHEST_DEGREE_VARIABLE,LEAST_CONSTRAINING_VALUE)
variables=('WA','NT','SA','Q','NSW', 'V','T')
domains=dict((v,['red','green','blue']) for v in variables)
def const_different(variables,values):
    return values[0]!=values[1] #expect the value of neighbors to be different
constraints=[
    (('WA','NT'),const_different),
    (('WA','SA'),const_different),
    (('SA','NT'),const_different),
    (('SA','Q'),const_different),
    (('NT','Q'),const_different),
    (('SA','NSW'),const_different),
    (('Q','NSW'),const_different),
    (('SA','V'),const_different),
    (('NSW','V'),const_different),]
my_problem=CspProblem(variables,domains,constraints)
print(backtrack(my_problem))
print(backtrack(my_problem,variable_heuristic=MOST_CONSTRAINED_VARIABLE))
print(backtrack(my_problem,variable_heuristic=HIGHEST_DEGREE_VARIABLE))
print(backtrack(my_problem,value_heuristic=LEAST_CONSTRAINING_VALUE))
print(backtrack(my_problem,variable_heuristic=MOST_CONSTRAINED_VARIABLE,value_heuristic=LEAST_CONSTRAINING_VALUE))
print(backtrack(my_problem,variable_heuristic=HIGHEST_DEGREE_VARIABLE,value_heuristic=LEAST_CONSTRAINING_VALUE))
print(min_conflicts(my_problem))
```

Output 9a



```
% library(win_menu) compiled into win_menu 0.00 sec, 11,760 bytes
% library(swi_hooks) compiled into pce_swi_hooks 0.00 sec, 2,024 bytes
Warning: c:/users/sareeta/desktop/pl/9a.pl:6:
        Singleton variables: [X]
Warning: c:/users/sareeta/desktop/pl/9a.pl:7:
        Singleton variables: [X]
Warning: c:/users/sareeta/desktop/pl/9a.pl:8:
        Singleton variables: [X]
% c:/Users/sareeta/Desktop/pl/9a.pl compiled 0.00 sec, 2,008 bytes
Welcome to SWI-Prolog (Multi-threaded, 32 bits, Version 5.8.2)
Copyright (c) 1990-2009 University of Amsterdam.
SWI-Prolog comes with ABSOLUTELY NO WARRANTY. This is free software,
and you are welcome to redistribute it under certain conditions.
Please visit http://www.swi-prolog.org for details.

For help, use ?- help(Topic). or ?- apropos(Word).

1 ?- Warning: c:/users/sareeta/desktop/pl/9a.pl:6:
        Singleton variables: [X]
Warning: c:/users/sareeta/desktop/pl/9a.pl:7:
        Singleton variables: [X]
Warning: c:/users/sareeta/desktop/pl/9a.pl:8:
        Singleton variables: [X]
% c:/Users/sareeta/Desktop/pl/9a.pl compiled 0.00 sec, 388 bytes
1 ?- lbscitadmission(X).
X = raj ;
false.

2 ?- lbscitadmission(raj).
true.

3 ?- lbscitadmission(rahul).
false.

4 ?- rbscitadmission(raj).
true.

5 ?- rbscitadmission(rahul).
false.

6 ?-
Action (h for help) ? █
```

Output 9b



```
% library(win_menu) compiled into win_menu 0.00 sec, 11,760 bytes
% library(swi_hooks) compiled into pce_swi_hooks 0.00 sec, 2,024 bytes
Warning: c:/users/sareeta/desktop/pl/9b.pl:7:
        Singleton variables: [X]
Warning: c:/users/sareeta/desktop/pl/9b.pl:8:
        Singleton variables: [X]
Warning: c:/users/sareeta/desktop/pl/9b.pl:9:
        Singleton variables: [X]
% c:/Users/sareeta/Desktop/pl/9b.pl compiled 0.00 sec, 1,984 bytes
Welcome to SWI-Prolog (Multi-threaded, 32 bits, Version 5.8.2)
Copyright (c) 1990-2009 University of Amsterdam.
SWI-Prolog comes with ABSOLUTELY NO WARRANTY. This is free software,
and you are welcome to redistribute it under certain conditions.
Please visit http://www.swi-prolog.org for details.

For help, use ?- help(Topic). or ?- apropos(Word).

1 ?- ltobehusband(X).
X = raj ;
X = aayush.

2 ?- rtobehusband(X).
X = raj ;
X = aayush.

3 ?- rtobehusband(ritesh).
false.

4 ?-
```

Practical 9

a) Derive the expressions based on Associative law

Code:

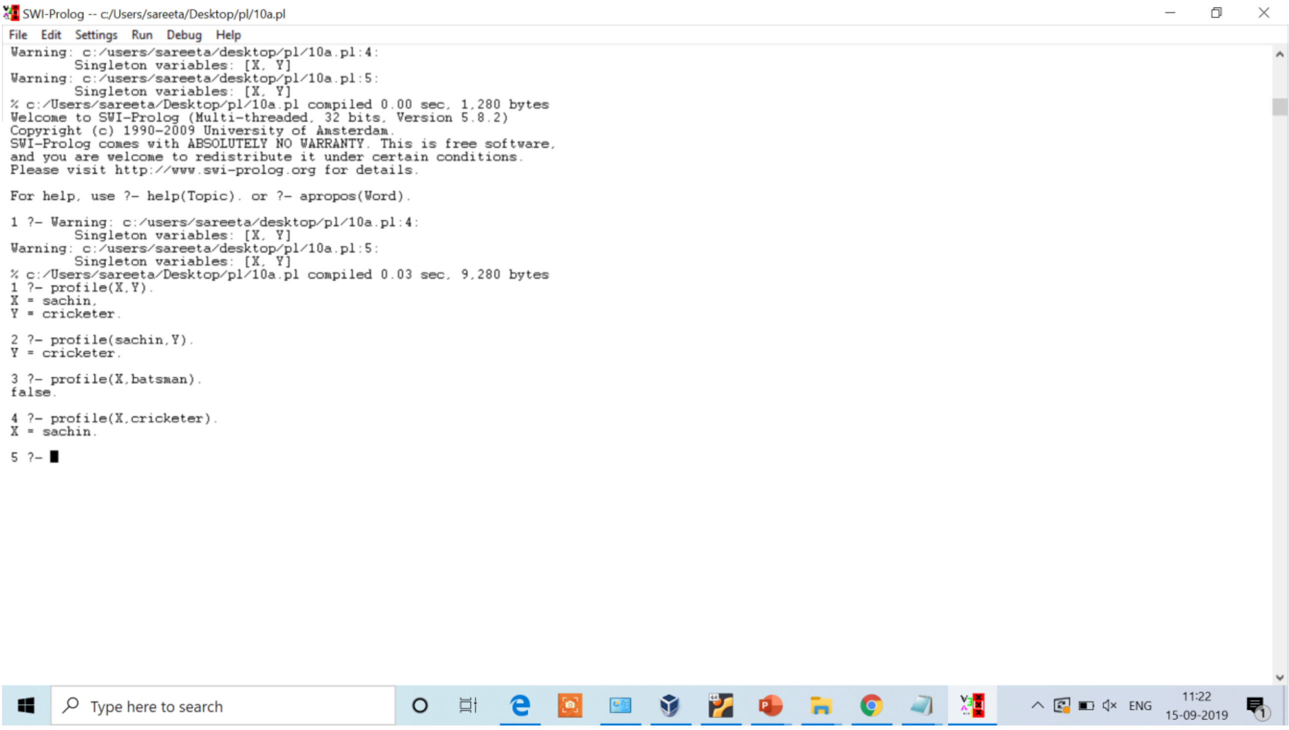
```
student(raj).
student(rahul).
hscstudent(raj).
sscstudent(raj).
sscstudent(rahul).
nondetermhscstudent(X).
nondetermsscstudent(X).
nondetermstudent(X).
lbscitadmission(X) :-((student(X),hscstudent(X)),sscstudent(X)).
rbscitadmission(X) :-(student(X),(hscstudent(X),sscstudent(X))).
```

b) Derive the expressions based on Distributive law

Code:

```
male(raj).
male(aayush).

doctor(raj).
engineer(aayush).

nondetermmale(X).
nondetermengineer(X).
nondetermdoctor(X).

ltobehusband(X) :-male(X),(doctor(X);engineer(X)).
rtobehusband(X) :- ( (male(X),doctor(X));(male(X),engineer(X))).
```

Output 10a



```
Warning: c:/users/sareeta/desktop/pl/10a.pl:4:
        Singleton variables: [X, Y]
Warning: c:/users/sareeta/desktop/pl/10a.pl:5:
        Singleton variables: [X, Y]
% c:/Users/sareeta/Desktop/pl/10a.pl compiled 0.00 sec, 1,280 bytes
Welcome to SWI-Prolog (Multi-threaded, 32 bits, Version 5.8.2)
Copyright (c) 1990-2009 University of Amsterdam.
SWI-Prolog comes with ABSOLUTELY NO WARRANTY. This is free software,
and you are welcome to redistribute it under certain conditions.
Please visit http://www.swi-prolog.org for details.

For help, use ?- help(Topic). or ?- apropos(Word).

1 ?- Warning: c:/users/sareeta/desktop/pl/10a.pl:4:
        Singleton variables: [X, Y]
Warning: c:/users/sareeta/desktop/pl/10a.pl:5:
        Singleton variables: [X, Y]
% c:/Users/sareeta/Desktop/pl/10a.pl compiled 0.03 sec, 9,280 bytes
1 ?- profile(X,Y).
X = sachin,
Y = cricketer.

2 ?- profile(sachin,Y).
Y = cricketer.

3 ?- profile(X,batsman).
false.

4 ?- profile(X,cricketer).
X = sachin.

5 ?- █
```

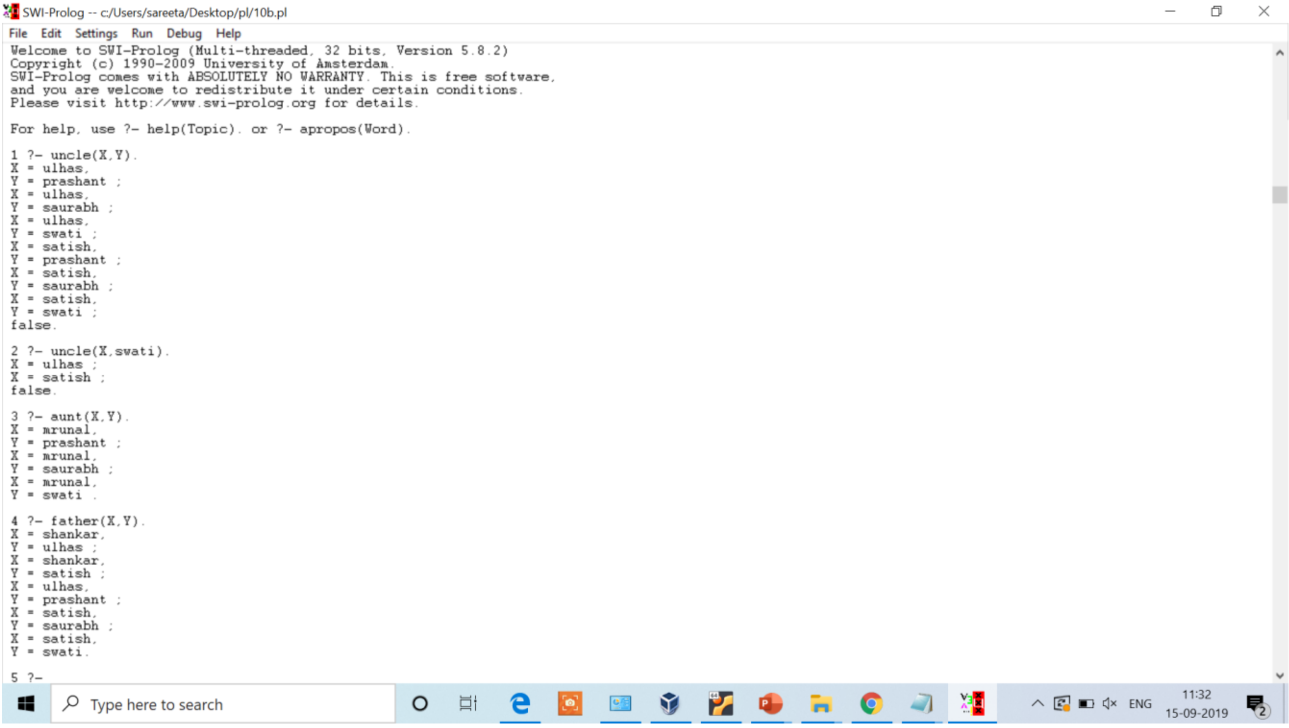
Practical 10

a) Write a program to derive the predicate.
(for e.g.: Sachin is batsman , batsman is cricketer) - > Sachin is Cricketer.

Code:

```
batsman(sachin,batsman).

cricketer(batsman,cricketer).

nondetermbatsman(X,Y).

nondetermcricketer(X,Y).

profile(X,Y) :-batsman(X,Z),cricketer(Z,Y).
```

Output 10b


SWI-Prolog -- c:/Users/sareeta/Desktop/pl/10b.pl
File Edit Settings Run Debug Help
Type here to search
ENG
11:32
15-09-2019

```
Welcome to SWI-Prolog (Multi-threaded, 32 bits, Version 5.8.2)
Copyright (c) 1990-2009 University of Amsterdam.
SWI-Prolog comes with ABSOLUTELY NO WARRANTY. This is free software,
and you are welcome to redistribute it under certain conditions.
Please visit http://www.swi-prolog.org for details.

For help, use ?- help(Topic). or ?- apropos(Word).

1 ?- uncle(X,Y).
X = ulhas,
Y = prashant ;
X = ulhas,
Y = saurabh ;
X = ulhas,
Y = swati ;
X = satish,
Y = prashant ;
X = satish,
Y = saurabh ;
X = satish,
Y = swati ;
false.

2 ?- uncle(X,swati).
X = ulhas ;
X = satish ;
false.

3 ?- aunt(X,Y).
X = mrunal,
Y = prashant ;
X = mrunal,
Y = saurabh ;
X = mrunal,
Y = swati .

4 ?- father(X,Y).
X = shankar,
Y = ulhas ;
X = shankar,
Y = satish ;
X = ulhas,
Y = prashant ;
X = satish,
Y = saurabh ;
X = satish,
Y = swati.

5 ?-
```

b) Write a program which contains three predicates: male, female, parent. Make rules for following family relations: father, mother, grandfather, grandmother, brother, sister, uncle, aunt, nephew and niece, cousin.
Question:
i. Draw Family Tree.
ii. Define: Clauses, Facts, Predicates and Rules with conjunction and disjunction

Code:

```
male(shankar).

male(ulhas).

male(satish).

male(saurabh).

male(prashant).

female(umabai).

female(mrunal).

female(sadhana).

female(swati).

parent(shankar,umabai,ulhas).

parent(shankar,umabai,satish).

parent(ulhas,mrunal,prashant).

parent(satish,sadhana,saurabh).

parent(satish,sadhana,swati).

brother(ulhas,satish).

brother(satish,ulhas).

brother(prashant,saurabh).

brother(saurabh,prashant).

sister(swati,saurabh).

sister(swati,prashant).

father(X,Y) :- parent(X,Z,Y).

mother(X,Y) :- parent(Z,X,Y).

son(X,Y,Z) :- male(X),father(Y,X),mother(Z,X).

daughter(X,Y,Z) :- female(X),father(Y,X),mother(Z,X).

wife(X,Y) :- female(X),parent(Y,X,Z).

grandfather(X,Y) :- male(X),father(X,Z),father(Z,Y).

grandmother(X,Y):-female(X),mother(X,Z),father(Z,Y).

uncle(X,Y):-
male(X),((father(Z,Y),father(A,Z),father(A,X));(mother(Z,Y),father(A,Z),father(A,X))).

aunt(X,Y) :- wife(X,Z),uncle(Z,Y).

brother(X,Y):-male(X),father(Z,X),father(Z,Y).
```

```
cousin(X,Y) :- father(Z,X),brother(Z,W),father(W,Y).
ancestor(X,Y,Z) :- parent(X,Y,Z).
ancestor(X,Y,Z) :- parent(X,Y,W),ancestor(W,U,Z).
```