*EX NO:11. Python program to implement Decision Tree*

*INPUT:*

*from collections import deque*

*class Graph:*

*def \_\_init\_\_(self, adjac\_lis):*

*self.adjac\_lis = adjac\_lis*

*def get\_neighbors(self, v):*

*return self.adjac\_lis[v]*

*# This is heuristic function which is having equal values for all nodes*

*def h(self, n):*

*H = {*

*'A': 1,*

*'B': 1,*

*'C': 1,*

*'D': 1*

*}*

*return H[n]*

*def a\_star\_algorithm(self, start, stop):*

*# In this open\_lst is a lisy of nodes which have been visited, but who's*

*# neighbours haven't all been always inspected, It starts off with the start*

*#node*

*# And closed\_lst is a list of nodes which have been visited*

*# and who's neighbors have been always inspected*

*open\_lst = set([start])*

*closed\_lst = set([])*

*# poo has present distances from start to all other nodes*

*# the default value is +infinity*

*poo = {}*

*poo[start] = 0*

*# par contains an adjac mapping of all nodes*

*par = {}*

*par[start] = start*

*while len(open\_lst) > 0:*

*n = None*

*# it will find a node with the lowest value of f() -*

*for v in open\_lst:*

*if n == None or poo[v] + self.h(v) < poo[n] + self.h(n):*

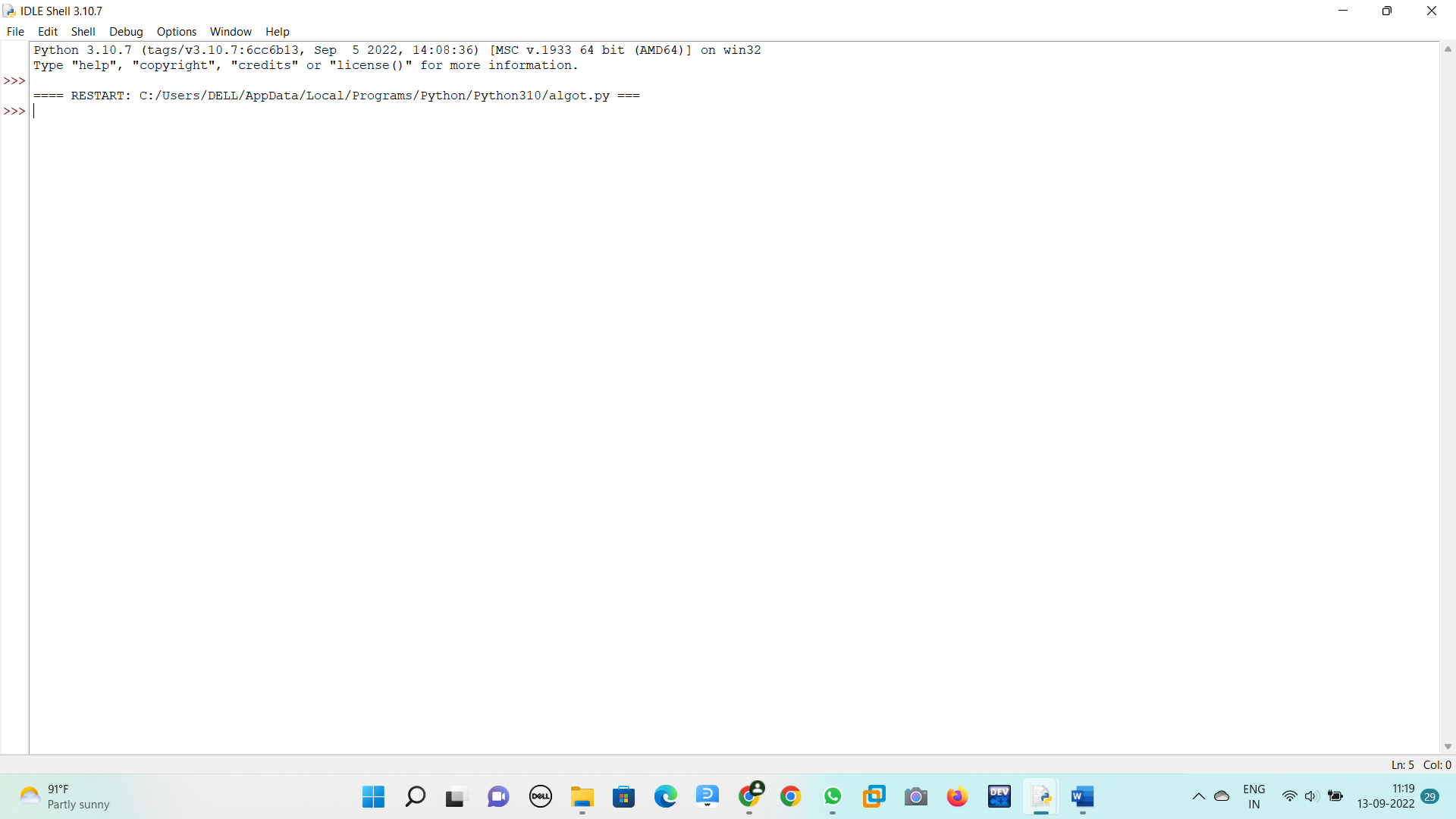
*n = v;*

*if n == None:*

*print('Path does not exist!')*

*return None*

*OUTPUT:*

**

*EX NO:12.Python program to solve the 8-Puzzle Problem*

*INPUT:*

*class Node:*

*def \_\_init\_\_(self,data,level,fval):*

*""" Initialize the node with the data, level of the node and the calculated fvalue """*

*self.data = data*

*self.level = level*

*self.fval = fval*

*def generate\_child(self):*

*""" Generate child nodes from the given node by moving the blank space*

*either in the four directions {up,down,left,right} """*

*x,y = self.find(self.data,'\_')*

*""" val\_list contains position values for moving the blank space in either of*

*the 4 directions [up,down,left,right] respectively. """*

*val\_list = [[x,y-1],[x,y+1],[x-1,y],[x+1,y]]*

*children = []*

*for i in val\_list:*

*child = self.shuffle(self.data,x,y,i[0],i[1])*

*if child is not None:*

*child\_node = Node(child,self.level+1,0)*

*children.append(child\_node)*

*return children*

*def shuffle(self,puz,x1,y1,x2,y2):*

*""" Move the blank space in the given direction and if the position value are out*

*of limits the return None """*

*if x2 >= 0 and x2 < len(self.data) and y2 >= 0 and y2 < len(self.data):*

*temp\_puz = []*

*temp\_puz = self.copy(puz)*

*temp = temp\_puz[x2][y2]*

*temp\_puz[x2][y2] = temp\_puz[x1][y1]*

*temp\_puz[x1][y1] = temp*

*return temp\_puz*

*else:*

*return None*

*def copy(self,root):*

*""" Copy function to create a similar matrix of the given node"""*

*temp = []*

*for i in root:*

*t = []*

*for j in i:*

*t.append(j)*

*temp.append(t)*

*return temp*

*def find(self,puz,x):*

*""" Specifically used to find the position of the blank space """*

*for i in range(0,len(self.data)):*

*for j in range(0,len(self.data)):*

*if puz[i][j] == x:*

*return i,j*

*class Puzzle:*

*def \_\_init\_\_(self,size):*

*""" Initialize the puzzle size by the specified size,open and closed lists to empty """*

*self.n = size*

*self.open = []*

*self.closed = []*

*def accept(self):*

*""" Accepts the puzzle from the user """*

*puz = []*

*for i in range(0,self.n):*

*temp = input().split(" ")*

*puz.append(temp)*

*return puz*

*def f(self,start,goal):*

*""" Heuristic Function to calculate hueristic value f(x) = h(x) + g(x) """*

*return self.h(start.data,goal)+start.level*

*def h(self,start,goal):*

*""" Calculates the different between the given puzzles """*

*temp = 0*

*for i in range(0,self.n):*

*for j in range(0,self.n):*

*if start[i][j] != goal[i][j] and start[i][j] != '\_':*

*temp += 1*

*return temp*

*def process(self):*

*""" Accept Start and Goal Puzzle state"""*

*print("Enter the start state matrix \n")*

*start = self.accept()*

*print("Enter the goal state matrix \n")*

*goal = self.accept()*

*start = Node(start,0,0)*

*start.fval = self.f(start,goal)*

*""" Put the start node in the open list"""*

*self.open.append(start)*

*print("\n\n")*

*while True:*

*cur = self.open[0]*

*print("")*

*print(" | ")*

*print(" | ")*

*print(" \\\'/ \n")*

*for i in cur.data:*

*for j in i:*

*print(j,end=" ")*

*print("")*

*""" If the difference between current and goal node is 0 we have reached the goal node"""*

*if(self.h(cur.data,goal) == 0):*

*break*

*for i in cur.generate\_child():*

*i.fval = self.f(i,goal)*

*self.open.append(i)*

*self.closed.append(cur)*

*del self.open[0]*

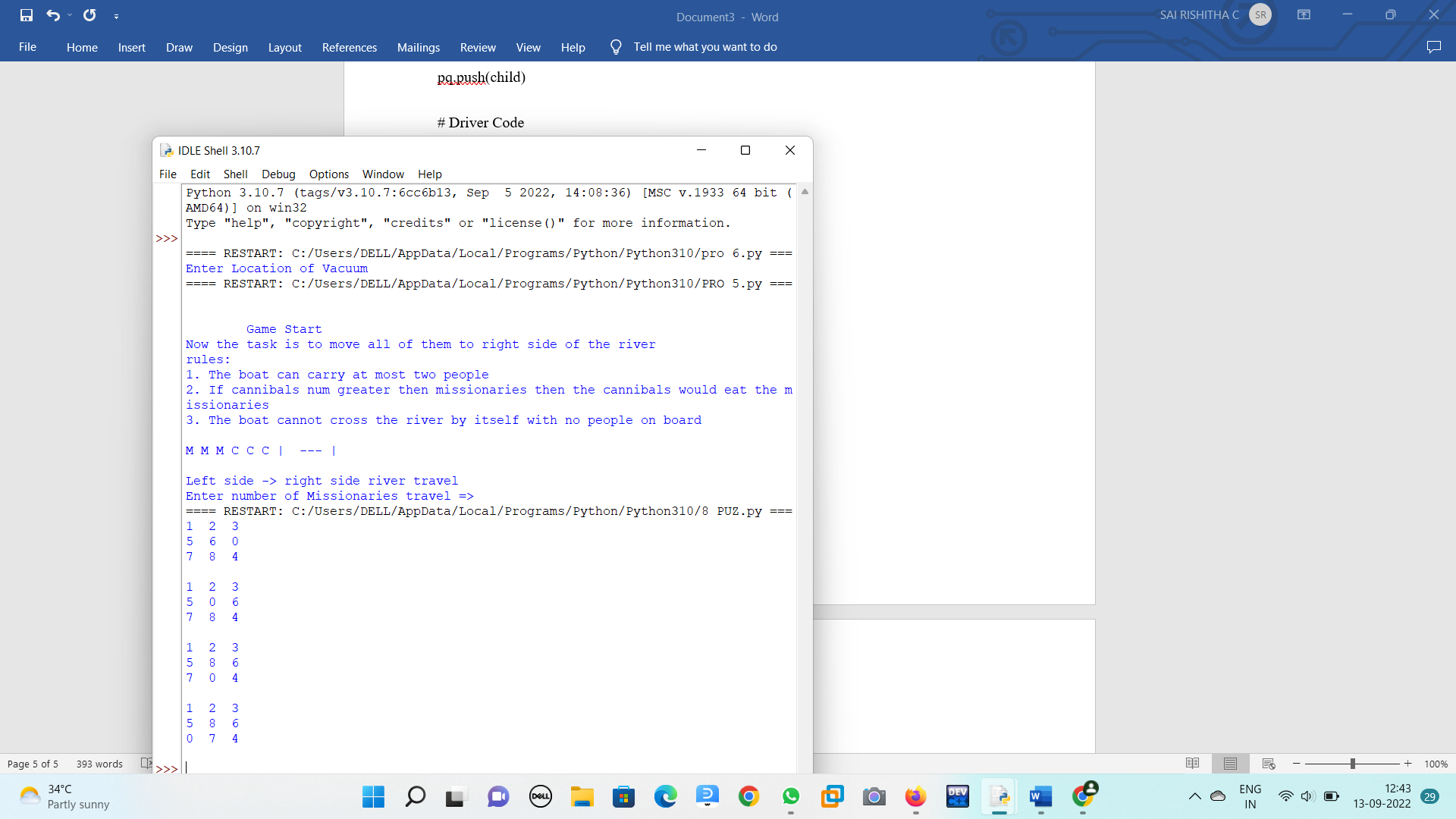
*""" sort the opne list based on f value """*

*self.open.sort(key = lambda x:x.fval,reverse=False)*

*puz = Puzzle(3)*

*puz.process()*

*OUTPUT:*

**

*EX NO:13 Python program to implement Feed forward neural Network*

*import numpy as np*

*import pandas as pd*

*from sklearn.metrics import confusion\_matrix*

*from sklearn.model\_selection import train\_test\_split*

*from sklearn.tree import DecisionTreeClassifier*

*from sklearn.metrics import accuracy\_score*

*from sklearn.metrics import classification\_report*

*# Function importing Dataset*

*def importdata():*

*balance\_data = pd.read\_csv(*

*'https://archive.ics.uci.edu/ml/machine-learning-'+*

*'databases/balance-scale/balance-scale.data',*

*sep= ',', header = None)*

*# Printing the dataswet shape*

*print ("Dataset Length: ", len(balance\_data))*

*print ("Dataset Shape: ", balance\_data.shape)*

*# Printing the dataset obseravtions*

*print ("Dataset: ",balance\_data.head())*

*return balance\_data*

*# Function to split the dataset*

*def splitdataset(balance\_data):*

*# Separating the target variable*

*X = balance\_data.values[:, 1:5]*

*Y = balance\_data.values[:, 0]*

*# Splitting the dataset into train and test*

*X\_train, X\_test, y\_train, y\_test = train\_test\_split(*

*X, Y, test\_size = 0.3, random\_state = 100)*

*return X, Y, X\_train, X\_test, y\_train, y\_test*

*# Function to perform training with giniIndex.*

*def train\_using\_gini(X\_train, X\_test, y\_train):*

*# Creating the classifier object*

*clf\_gini = DecisionTreeClassifier(criterion = "gini",*

*random\_state = 100,max\_depth=3, min\_samples\_leaf=5)*

*# Performing training*

*clf\_gini.fit(X\_train, y\_train)*

*return clf\_gini*

*# Function to perform training with entropy.*

*def tarin\_using\_entropy(X\_train, X\_test, y\_train):*

*# Decision tree with entropy*

*clf\_entropy = DecisionTreeClassifier(*

*criterion = "entropy", random\_state = 100,*

*max\_depth = 3, min\_samples\_leaf = 5)*

*# Performing training*

*clf\_entropy.fit(X\_train, y\_train)*

*return clf\_entropy*

*# Function to make predictions*

*def prediction(X\_test, clf\_object):*

*# Predicton on test with giniIndex*

*y\_pred = clf\_object.predict(X\_test)*

*print("Predicted values:")*

*print(y\_pred)*

*return y\_pred*

*# Function to calculate accuracy*

*def cal\_accuracy(y\_test, y\_pred):*

*print("Confusion Matrix: ",*

*confusion\_matrix(y\_test, y\_pred))*

*print ("Accuracy : ",*

*accuracy\_score(y\_test,y\_pred)\*100)*

*print("Report : ",*

*classification\_report(y\_test, y\_pred))*

*# Driver code*

*def main():*

*# Building Phase*

*data = importdata()*

*X, Y, X\_train, X\_test, y\_train, y\_test = splitdataset(data)*

*clf\_gini = train\_using\_gini(X\_train, X\_test, y\_train)*

*clf\_entropy = tarin\_using\_entropy(X\_train, X\_test, y\_train)*

*# Operational Phase*

*print("Results Using Gini Index:")*

*# Prediction using gini*

*y\_pred\_gini = prediction(X\_test, clf\_gini)*

*cal\_accuracy(y\_test, y\_pred\_gini)*

*print("Results Using Entropy:")*

*# Prediction using entropy*

*y\_pred\_entropy = prediction(X\_test, clf\_entropy)*

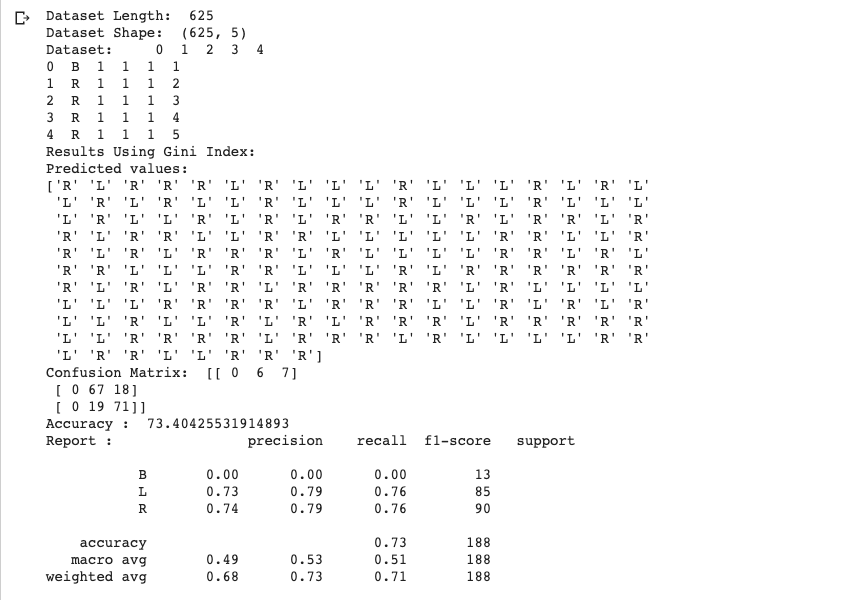
*cal\_accuracy(y\_test, y\_pred\_entropy)*

*# Calling main function*

*if \_\_name\_\_=="\_\_main\_\_":*

*main()*

*OUTPUT:*

**

*EX NO:14. PROLOG PROGRAM TO IMPLEMENT FAMILY TREE.*

*male(jack).*

*male(oliver).*

*male(ali).*

*male(james).*

*male(simon).*

*male(harry).*

*female(helen).*

*female(sophie).*

*female(jess).*

*female(lily).*

*parent\_of(jack,jess).*

*parent\_of(jack,lily).*

*parent\_of(helen, jess).*

*parent\_of(helen, lily).*

*parent\_of(oliver,james).*

*parent\_of(sophie, james).*

*parent\_of(jess, simon).*

*parent\_of(ali, simon).*

*parent\_of(lily, harry).*

*parent\_of(james, harry).*

*/\* Rules \*/*

*father\_of(X,Y):- male(X),*

*parent\_of(X,Y).*

*mother\_of(X,Y):- female(X),*

*parent\_of(X,Y).*

*grandfather\_of(X,Y):- male(X),*

*parent\_of(X,Z),*

*parent\_of(Z,Y).*

*grandmother\_of(X,Y):- female(X),*

*parent\_of(X,Z),*

*parent\_of(Z,Y).*

*sister\_of(X,Y):- %(X,Y or Y,X)%*

*female(X),*

*father\_of(F, Y), father\_of(F,X),X \= Y.*

*sister\_of(X,Y):- female(X),*

*mother\_of(M, Y), mother\_of(M,X),X \= Y.*

*aunt\_of(X,Y):- female(X),*

*parent\_of(Z,Y), sister\_of(Z,X),!.*

*brother\_of(X,Y):- %(X,Y or Y,X)%*

*male(X),*

*father\_of(F, Y), father\_of(F,X),X \= Y.*

*brother\_of(X,Y):- male(X),*

*mother\_of(M, Y), mother\_of(M,X),X \= Y.*

*uncle\_of(X,Y):-*

*parent\_of(Z,Y), brother\_of(Z,X).*

*ancestor\_of(X,Y):- parent\_of(X,Y).*

*ancestor\_of(X,Y):- parent\_of(X,Z),*

*ancestor\_of(Z,X)*

*OUTPUT :*

*?-mother\_of(X,jess).*

*?-parent\_of(X,simon).*

*?-sister\_of(X,lily).*

*?-ancestor\_of(X,lily).*

*EX NO:15 PROLOG PROGRAM TO IMPLEMENT FIBONACCI SERIES*

*fib(0, 1) :-*

*!.*

*fib(n, f) :-*

*fib(1, n, 1, 1, f).*

*fib(n, n, \_, f, f) :-*

*!.*

*fib(n0, n, f0, f1, f) :-*

*n1 is n0 + 1,*

*f2 is f0 + f1,*

*fib(n1, n, f1, f2, f).*

*OUTPUT:*

*Fib = 1346269*

*?- time(fib(30, Fib)).*

*EX NO:16. PROLOG PROGRAM TO FIND FACTORIAL NUMBER*

*INPUT:*

*predicates*

*factorial(integer, real)*

*go*

*clauses*

*go if*

*write("Enter a positive integer number:"),*

*readint(N),*

*factorial(N,Result),*

*write("Factorial of", N, "is=", Result).*

*factorial(0, 1)*

*factorial(N, Result) if N>0,*

*N1=N-1,*

*factorial(N1, Res),*

*Result=N\*Res.*

*OUTPUT:*

*goals:*

*factorial(5, Answer)*

*Answer=120*

*EX NO: 17. GCD OF TWO NUMBERS*

*INPUT:*

*gcd(X,Y):-X=Y,write('GCD of two numbers is '),write(X);*

*X=0,write('GCD of two numbers is '),write(Y);*

*Y=0,write('GCD of two numbers is '),write(X);*

*Y>X,Y1 is Y-X,gcd(X,Y1);*

*X>Y,Y1 is X-Y,gcd(Y1,Y).*

*OUTPUT:*

**

*EX NO:18. PROLOG PROGRAM to PRINTING ALL ELEMENTS OF A LIST*

*INPUT:*

*printlist([]).*

*printlist([X|List]) :- write(X),nl, printlist(List).*

*OUTPUT:*

**

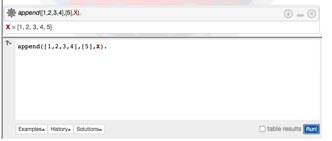
*EX NO:19. PROLOG PROGRAM TO APPEND AN INTEGER INTO THE LIST*

*INPUT:*

*append([],L,L).*

*append([X|L1],L2,[X|L3]) :- append(L1,L2,L3).*

*OUTPUT:*

**

*EX NO:20. PROLOG PROGRAM TO LIST MEMBERSHIP*

*INPUT:*

*member(X,List):- delete(X,List,\_).*

*delete(X,[X|Tail],Tail). delete(X,[Y|Tail1],[Y|Tail2]):-*

*delete(X,Tail1,Tail2).*

*OUTPUT:*

**

*EX NO21:. PROLOG PROGRAM FOR Healthcare Data*

*INPUT:*

*domains disease,indication,name=symbol predicates*

*hypothesis(name,disease) symptom(name,indication)*

*clauses*

*symptom(yamini,fever). symptom(yamini,rash) . symptom(yamini,headache). symptom(yamini,runn\_nose). symptom(hemanth,chills). symptom(hemanth,fever). symptom(hemnth,headache). symptom(radhika,runny\_nose). symptom(radhika,rash). symptom(radhika,flu). hypothesis(Patient,measels): symptom(Patient,fever), symptom(Patient,cough), symptom(Patient,conjunctivitis), symptom(Patient,r ash). hypothesis(Patient,german\_measl es): symptom(Patient,f ev er ), symptom(Patient,headache), symptom(Patient,runny\_nose), symptom(Patient,rash).*

*OUTPUT:*

**

*EX NO:22 Eliminate consecutive duplicates of list elements.*

*If a list contains repeated elements they should be replaced with a single copy of the element. The order of the elements should not be changed.  
  
Example:  
?- compress([a,a,a,a,b,c,c,a,a,d,e,e,e,e],X).  
X = [a,b,c,a,d,e]*

*EX NO:23  Run-length encoding of a list.*

*Use the result of problem P09 to implement the so-called run-length encoding data compression method. Consecutive duplicates of elements are encoded as terms [N,E] where N is the number of duplicates of the element E.  
  
Example:  
?- encode([a,a,a,a,b,c,c,a,a,d,e,e,e,e],X).  
X = [[4,a],[1,b],[2,c],[2,a],[1,d][4,e]]*

*EX NO:24 Truth tables for logical expressions.*

*Define predicates and/2, or/2, nand/2, nor/2, xor/2, impl/2 and equ/2 (for logical equivalence) which succeed or fail according to the result of their respective operations; e.g. and(A,B) will succeed, if and only if both A and B succeed. Note that A and B can be Prolog goals (not only the constants true and fail).*

*A logical expression in two variables can then be written in prefix notation, as in the following example: and(or(A,B),nand(A,B)).*

*Now, write a predicate table/3 which prints the truth table of a given logical expression in two variables.*

*Example:  
?- table(A,B,and(A,or(A,B))).  
true true true  
true fail true  
fail true fail  
fail fail fail*

*EX NO:25 Prolog - Monkey and Banana Problem*

*on(floor,monkey).*

*on(floor,chair).*

*in(room,monkey).*

*in(room,chair).*

*in(room,banana).*

*at(ceiling,banana).*

*strong(monkey).*

*grasp(monkey).*

*climb(monkey,chair).*

*push(monkey,chair):-*

*strong(monkey).*

*under(banana,chair):-*

*push(monkey,chair).*

*canreach(banana,monkey):-*

*at(floor,banana);*

*at(ceiling,banana),*

*under(banana,chair),*

*climb(monkey,chair).*

*canget(banana,monkey):-*

*canreach(banana,monkey),grasp(monkey).*