

## SAVEETHA INSTITUTE OF MEDICAL AND TECHNICAL SCIENCES

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Exp.No | Date | Title | Page no. | Marks | Sign |
| 1 | 12.07.21 | Python program to solve 8-Puzzle problem. |  |  |  |
| 2 | 12.07.21 | Python program to solve 8-Queen problem. |  |  |  |
| 3 | 13.07.21 | Python program to implement BFS. |  |  |  |
| 4 | 13.07.21 | Python program to implement DFS. |  |  |  |
| 5 | 14.07.21 | Python to implement Travelling Salesman Problem. |  |  |  |
| 6 | 14.07.21 | Python program to implement Minimax Algorithm. |  |  |  |
| 7 | 15.07.21 | Python program to implement Decision Tree. |  |  |  |
| 8 | 16.07.21 | Python program to implement Feed forward neural Network. |  |  |  |
| 9 | 17.07.21 | Prolog program to implement family tree. |  |  |  |
| 10 | 17.07.21 | Prolog program to implement factorial. |  |  |  |
| 11 | 17.07.21 | Prolog program to implement Finding GCD of two numbers. |  |  |  |
| 12 | 17.07.21 | Write the prolog program for Working with Lists:   1. Printing all elements of a list. 2. To append an integer into the list. 3. List Membership. |  |  |  |
| 13 | 19.07.21 | Prolog program for Medical Diagnosis. |  |  |  |
| 14 | 19.07.21 | Create a Web Blog using WordPress. |  |  |  |

INDEX

Date: 8-PUZZLE PROBLEM

Ex. No: 1

AIM:

To write a python program to implement 8(eight) puzzle problem.

ALGORITHM:

STEP 1: Start the program.

STEP 2: Get the current state of the scenario (refers to the board or game in real world).

STEP 3: Find the available moves and their cost.

STEP 4: Choose the move with the least cost and set it as the current state. STEP 5: Check if it matches the goal state, if yes terminate, if no move to step 2. STEP 6: Compile and execute the program.

STEP 7: Print the result.

STEP 8: End.

PROGRAM:

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]

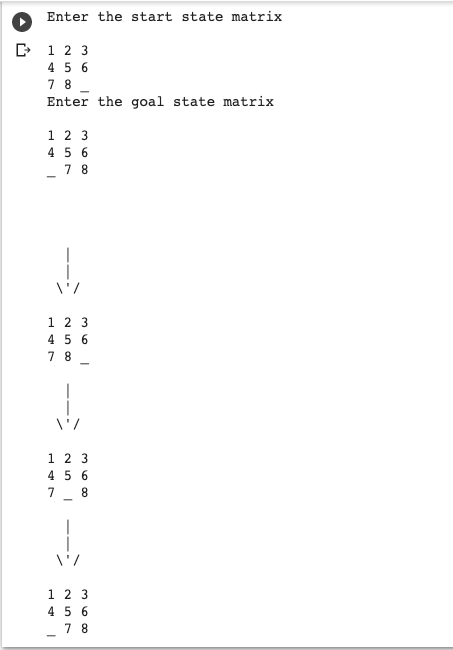
""" sort the opne list based on f value """

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

puz = Puzzle(3)

puz.process()

OUTPUT:



RESULT:

Therefore, the above program to implement 8 puzzle problem is compiled and executed successfully.

Date: 8-QUEEN PROBLEM

Ex. No: 2

AIM:

To write a python program to implement 8(eight) Queen problem.

ALGORITHM:

STEP 1: Start the program.

STEP 2: Start by placing the first queen on the top-left square of the chessboard.

STEP 3: If we have placed the 8 queens, we’re done. Otherwise, is it possible to place a another queen in a safe position?

STEP 4: If yes, then mark this [row, column] as part of the solution, and go back to point 3.

END CIENCES STEP 5: If no, then change the position of the previous queen (backtracking) and go back to point 3.

STEP 6: Compile and execute the program.

STEP 7: Print the result.

STEP 8: End

PROGRAM:

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.show\_short\_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)

"""

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 show\_short\_board(self, positions):

"""

Show the queens positions on the board in compressed form,

each number represent the occupied column position in the corresponding row.

"""

line = ""

for i in range(self.size):

line += str(positions[i]) + " "

print(line)

def main():

"""Initialize and solve the n queens puzzle"""

n=int(input("Enter the n value:"))

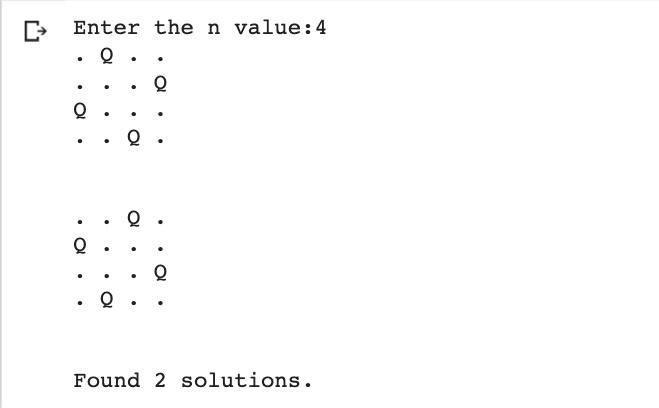
NQueens(n)

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

# execute only if run as a script

main()

OUTPUT:



RESULT: Therefore, the above program to implement 8 Queen problem is compiled and executed successfully.

Date: BREADTH FIRST SEARCH (BFS )

Ex. No: 3

AIM:

To write a python program to implement Breadth First Search Algorithm.

ALGORITHM:

STEP 1: Start the program.

STEP 2: Start by putting any one of the graph’s vertices at the back of the queue.

STEP 3: Now take the front item of the queue and add it to the visited list.

STEP 4: Create a list of that vertex's adjacent nodes.

STEP 5: Add those which are not within the visited list to the rear of the queue

STEP 6: Keep continuing steps two and three till the queue is empty.

STEP 7: Compile and execute the program.

STEP 8: Print the result.

STEP 9: End.

**PROGRAM:**

graph = {

'5' : ['3','7'],

'3' : ['2', '4'],

'7' : ['8'],

'2' : [],

'4' : ['8'],

'8' : []

}

visited = [] # List for visited nodes.

queue = [] #Initialize a queue

def bfs(visited, graph, node): #function for BFS

visited.append(node)

queue.append(node)

while queue: # Creating loop to visit each node

m = queue.pop(0)

print (m, end = " ")

for neighbour in graph[m]:

if neighbour not in visited:

visited.append(neighbour)

queue.append(neighbour)

# Driver Code

print("Following is the Breadth-First Search")

bfs(visited, graph, '5') # function calling

OUTPUT:



RESULT:

Therefore, the above program to implement Breadth First Search Algorithm is compiled and executed successful

Date: DEPTH FIRST SEARCH (DFS )

Ex. No: 4

AIM:

To write a python program to implement Depth First Search Algorithm.

ALGORITHM:

STEP 1: Start the program.

STEP 2: We will start by putting any one of the graph's vertex on top of the stack.

STEP 3: After that take, the top item of the stack and add it to the visited list of the vertex.

STEP 4: Next, create a list of that adjacent node of the vertex.

STEP 5: Add the ones which are not in the visited list of vertexes to the top of the stack.

STEP 6: Lastly, keep repeating steps 3, 4 and 5 until the stack is empty.

STEP 7: Compile and execute the program.

STEP 8: Print the result.

STEP 9: End.

PROGRAM:

# Using a Python dictionary to act as an adjacency list

graph = {

'A' : ['B','C'],

'B' : ['D', 'E'],

'C' : ['F'],

'D' : [],

'E' : ['F'],

'F' : []

}

visited = set() # Set to keep track of visited nodes.

def dfs(visited, graph, node):

if node not in visited:

print (node)

visited.add(node)

for neighbour in graph[node]:

dfs(visited, graph, neighbour)

# Driver Code

dfs(visited, graph, ‘A')

OUTPUT:

RESULT: Therefore, the above program to implement Depth First Search Algorithm is compiled and executed successfully.

Date: TRAVELLING SALESMAN PROBLEM

Ex. No: 5

AIM:

To write a python program to implement Travelling Salesman Problem.

ALGORITHM:

STEP 1: Start the program.

STEP 2: Consider city 1 as the starting and ending point.

STEP 3: Since the route is cyclic, we can consider any point as a starting point.

STEP 4: Generate all (n-1)! permutations of cities.

STEP 5: Calculate the cost of every permutation and keep track of the minimum cost permutation.

STEP 6: Return the permutation with minimum cost.

STEP 7: Compile and execute the program.

STEP 8: Print the result.

STEP 9: End.

PROGRAM:

# Python3 program to implement traveling salesman

# problem using naive approach.

from sys import maxsize

from itertools import permutations

V = 4

# implementation of traveling Salesman Problem

def travellingSalesmanProblem(graph, s):

# store all vertex apart from source vertex

vertex = []

for i in range(V):

if i != s:

vertex.append(i)

# store minimum weight Hamiltonian Cycle

min\_path = maxsize

next\_permutation=permutations(vertex)

for i in next\_permutation:

# store current Path weight(cost)

current\_pathweight = 0

# compute current path weight

k = s

for j in i:

current\_pathweight += graph[k][j]

k = j

current\_pathweight += graph[k][s]

# update minimum

min\_path = min(min\_path, current\_pathweight)

return min\_path

# Driver Code

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

# matrix representation of graph

graph = [[0, 10, 15, 20], [10, 0, 35, 25],

[15, 35, 0, 30], [20, 25, 30, 0]]

s = 0

print(travellingSalesmanProblem(graph, s))

OUTPUT:



RESULT: Therefore, the above program to implement Travelling Salesman Problem is compiled and executed successfully.

Date: MINIMAX ALGORITHM

Ex. No: 6

AIM:

To write a python program to implement Min-Max Algorithm.

ALGORITHM:

STEP 1: Start the program

STEP 2: Minimax is a kind of [backtracking](https://www.geeksforgeeks.org/tag/backtracking/) algorithm that is used in decision making and game theory to find the optimal move for a player, assuming that your opponent also plays optimally.

STEP 3: It is widely used in two player turn-based games such as Tic-Tac-Toe, Backgammon, Mancala, Chess, etc.

STEP 4: In Minimax the two players are called maximizer and minimizer.

STEP 5: Maximizer goes LEFT: It is now the minimizers turn. The minimizer now has a choice between 3 and 5. Being the minimizer it will choose the least among both, that is 3

STEP 6: Maximizer goes RIGHT: It is now the minimizers turn. The minimizer now has a choice between 2 and 9. He will choose 2 as it is the least among the two values.

STEP 7: Being the maximizer, you would choose the larger value

STEP 8: Compile and execute the program.

STEP 9: Print the result.

STEP 10: End.

**PROGRAM:**

# A simple Python3 program to find

# maximum score that

# maximizing player can get

import math

def minimax (curDepth, nodeIndex,

maxTurn, scores,

targetDepth):

# base case : targetDepth reached

if (curDepth == targetDepth):

return scores[nodeIndex]

if (maxTurn):

return max(minimax(curDepth + 1, nodeIndex \* 2,

False, scores, targetDepth),

minimax(curDepth + 1, nodeIndex \* 2 + 1,

False, scores, targetDepth))

else:

return min(minimax(curDepth + 1, nodeIndex \* 2,

True, scores, targetDepth),

minimax(curDepth + 1, nodeIndex \* 2 + 1,

True, scores, targetDepth))

# Driver code

scores = [3, 5, 2, 9, 12, 5, 23, 23]

treeDepth = math.log(len(scores), 2)

print("The optimal value is : ", end = "")

print(minimax(0, 0, True, scores, treeDepth))

# This code is contributed

# by rootshadow

OUTPUT:

RESULT: Therefore, the above program to implement Min-Max Algorithm is compiled and executed successfully.

Date: DECISION TREE

Ex. No: 7

AIM:

To write a python program to implement Decision Tree.

ALGORITHM:

STEP 1: Start the program. STEP 2: Load required packages. STEP 3: Load the dataset.

STEP 4: Visualize the dataset using a graph.

STEP 5: Define the features and the target. STEP 6: Split the dataset and test sets.

STEP 7: Build the model with the decision tree.

STEP 8: Compile and execute the program.

STEP 9: Print the result.

STEP 10: End.

PROGRAM:

# Run this program on your local python

# interpreter, provided you have installed

# the required libraries.

# Importing the required packages

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

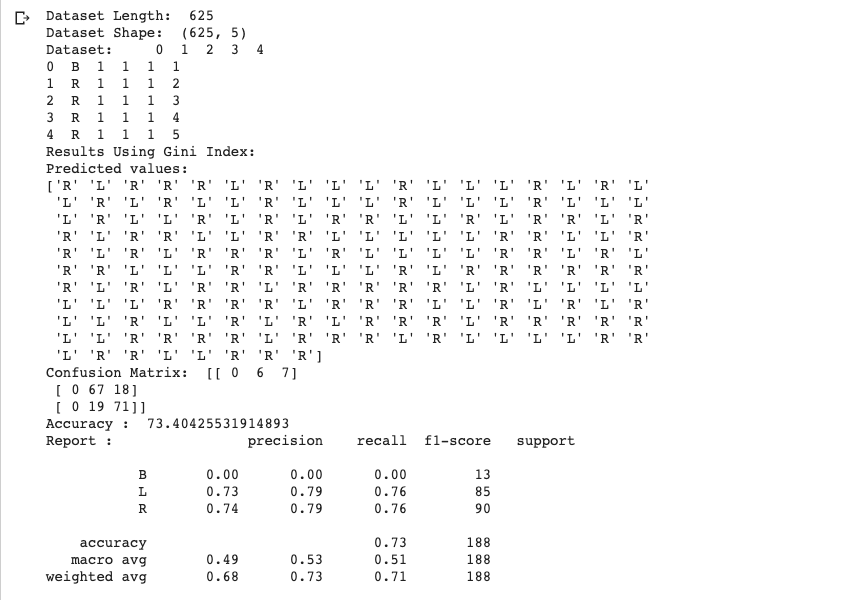
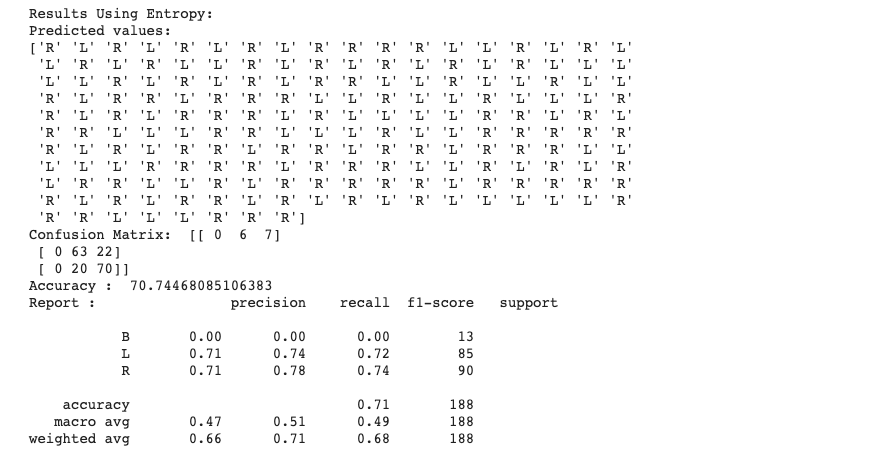
y\_pred\_entropy = prediction(X\_test, clf\_entropy)

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

# Calling main function

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

main()

OUTPUT:

RESULT: Therefore, the above program to implement Decision Tree is compiled and executed successfully

Date: NEURAL NETWORK

Ex. No: 8

AIM:

To write a python program to implement Feed forward and neural network.

ALGORITHM:

STEP 1: Start the program.

STEP 2: Create sample weights to be applied in the input layer, first hidden layer and the second hidden layer.

STEP 3: The weight for each layer is created as matrix of size M x N where M represents the number of neurons in the layer and N represents number of nodes / neurons in the next layer.

STEP 4: Weighted sum is calculated for neurons at every layer.

STEP 5: weighted sum is sum of weights and input signal combined with the bias element.

STEP 6: SoftMax function is applied to the output in the last layer.

STEP 7: Compile and execute the program.

STEP 8: Print the result.

STEP 9: End.

PROGRAM:

import numpy as np

class NeuralNetwork():

def \_\_init\_\_(self):

# seeding for random number generation

np.random.seed(1)

#converting weights to a 3 by 1 matrix with values from -1 to 1 and mean of 0

self.synaptic\_weights = 2 \* np.random.random((3, 1)) - 1

def sigmoid(self, x):

#applying the sigmoid function

return 1 / (1 + np.exp(-x))

def sigmoid\_derivative(self, x):

#computing derivative to the Sigmoid function

return x \* (1 - x)

def train(self, training\_inputs, training\_outputs, training\_iterations):

#training the model to make accurate predictions while adjusting weights continually

for iteration in range(training\_iterations):

#siphon the training data via the neuron

output = self.think(training\_inputs)

#computing error rate for back-propagation

error = training\_outputs - output

#performing weight adjustments

adjustments = np.dot(training\_inputs.T, error \* self.sigmoid\_derivative(output))

self.synaptic\_weights += adjustments

def think(self, inputs):

#passing the inputs via the neuron to get output

#converting values to floats

inputs = inputs.astype(float)

output = self.sigmoid(np.dot(inputs, self.synaptic\_weights))

return output

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

#initializing the neuron class

neural\_network = NeuralNetwork()

print("Beginning Randomly Generated Weights: ")

print(neural\_network.synaptic\_weights)

#training data consisting of 4 examples--3 input values and 1 output

training\_inputs = np.array([[0,0,1],

[1,1,1],

[1,0,1],

[0,1,1]])

training\_outputs = np.array([[0,1,1,0]]).T

#training taking place

neural\_network.train(training\_inputs, training\_outputs, 15000)

print("Ending Weights After Training: ")

print(neural\_network.synaptic\_weights)

user\_input\_one = str(input("User Input One: "))

user\_input\_two = str(input("User Input Two: "))

user\_input\_three = str(input("User Input Three: "))

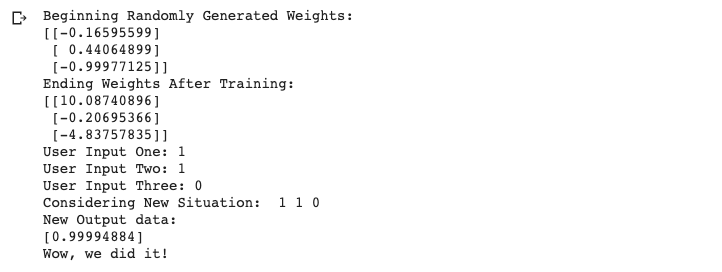
print("Considering New Situation: ", user\_input\_one, user\_input\_two, user\_input\_three)

print("New Output data: ")

print(neural\_network.think(np.array([user\_input\_one, user\_input\_two, user\_input\_three])))

print("Wow, we did it!")

OUTPUT :

 RESULT:Therefore, the above program to implement Feed forward and neural network is compiled and executed successfully

Date: PROLOG PROGRAM FOR FAMILY TREE

Ex. No: 9

AIM:

To write a prolog program to implement Family Tree.

ALGORITHM:

STEP 1: Start the program.

STEP 2: Declare the facts and rules for family tree.

STEP 3: Perform suitable operations.

STEP 4: Post Queries related to the given problem in the Query box.

STEP 5: Compile and execute the program.

STEP 6: Get the answers for the queries asked and print the result.

STEP 7: End.

PROGRAM:

female(vasundhara).

female(padmavathamma).

female(nikitha).

female(aarthi).

male(narayana).

male(prakash).

male(raju).

parent(narayana,prakash).

parent(padmavathamma,prakash).

parent(prakash,nikitha).

parent(vasundhara,nikitha).

parent(prakash,aarthi).

parent(vasundhara,aarthi).

parent(prakash,raju).

parent(vasundhara,raju).

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

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

sister(X,Y):-parent(Z,X),parent(Z,Y),female(X),X\==Y.

brother(X,Y):-parent(Z,X),parent(Z,Y),male(X),X\==Y.

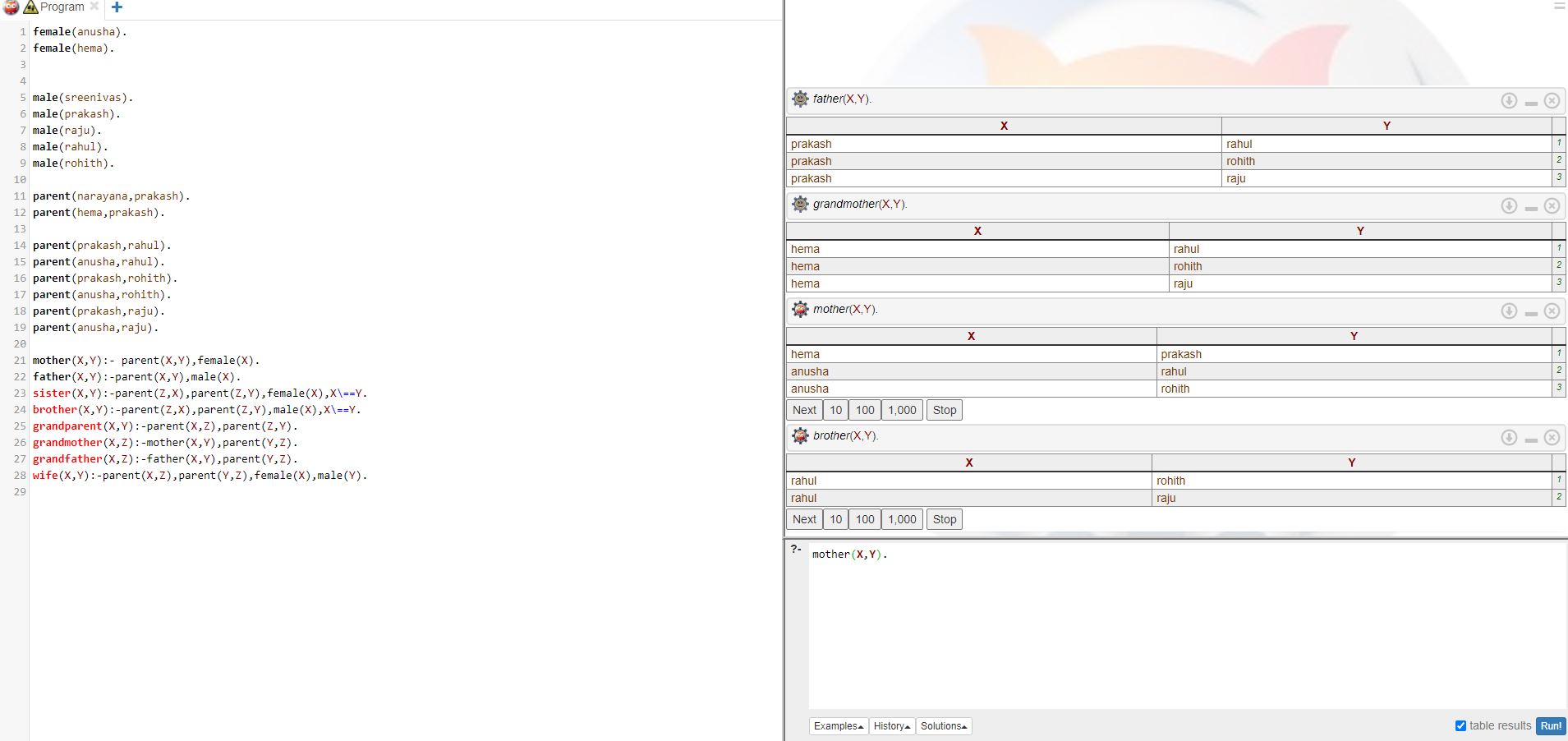
grandparent(X,Y):-parent(X,Z),parent(Z,Y).

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

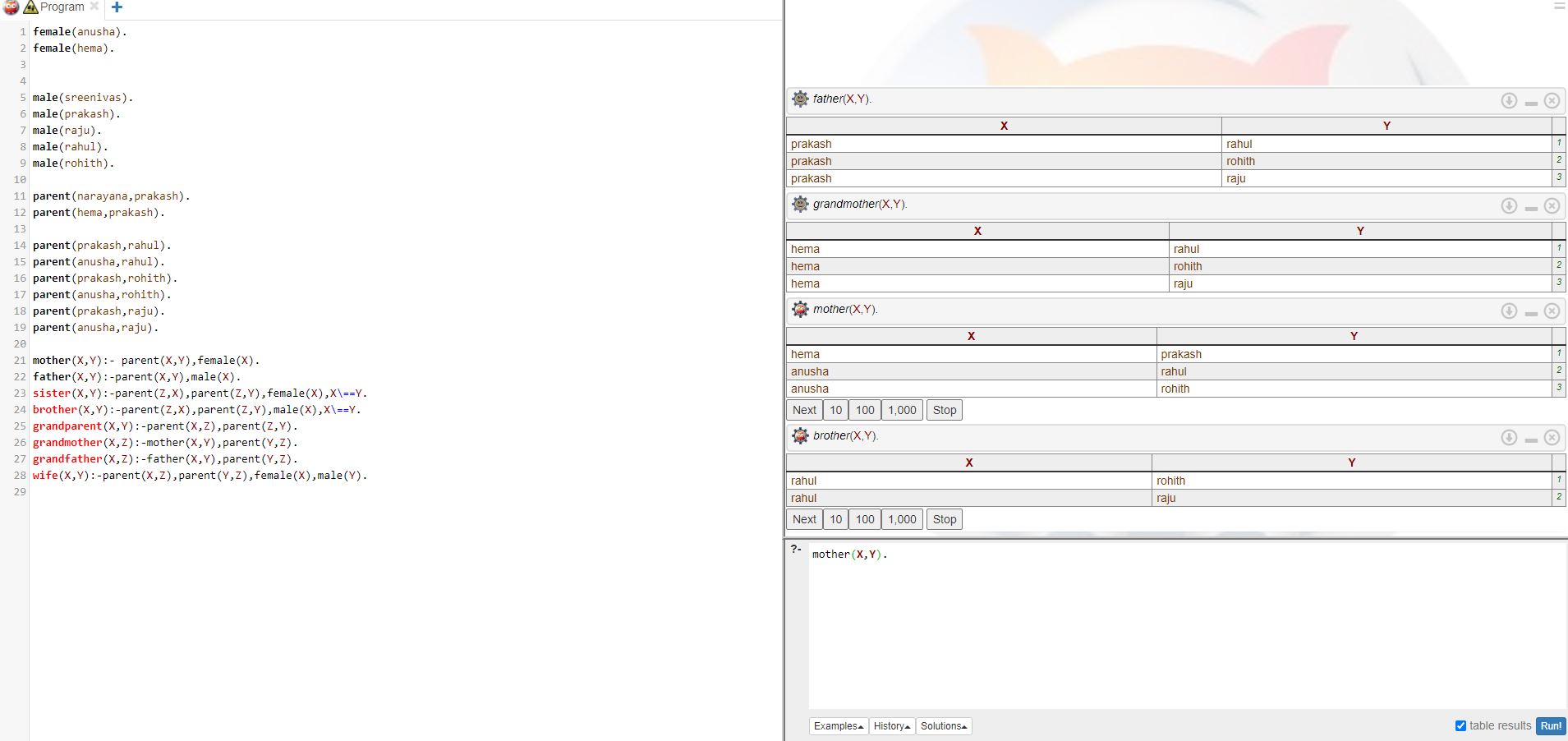
grandfather(X,Z):-father(X,Y),parent(Y,Z).

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

OUTPUT:



RESULT:



Date: PROLOG PROGRAM FOR GETTING INPUT

AND PROCESSING OUTPUT

Ex. No: 10(A)

A).PROCESSING NUMBERS: CALCULATING FACTORIAL

AIM:

To write a prolog program to implement Factorial.

ALGORITHM:

STEP 1: Start the program

STEP 2: Enter the integer as X.

STEP 3: Initialize A=1

STEP 4: If X=0, then print Factorial of A.

STEP 5: If X! =0, (ie) =B, then perform Factorial of B and print the result.

STEP 6: End.

PROGRAM:

fact(0,1).

fact(N,F) :-

( % the below is for +ve factorial N > 0->

( N1 is N - 1,

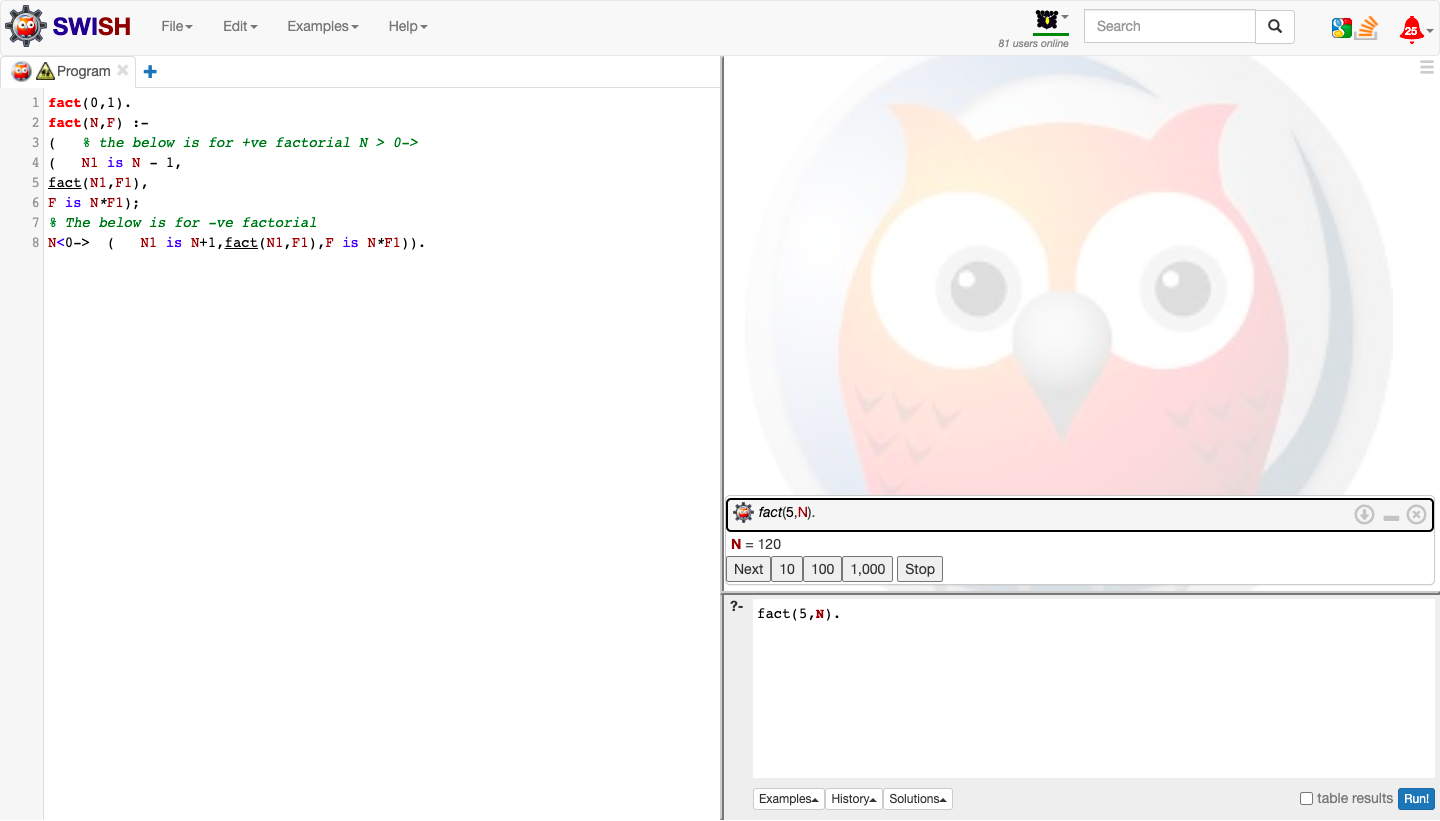
fact(N1,F1),

F is N\*F1);

% The below is for -ve factorial

N<0-> ( N1 is N+1,fact(N1,F1),F is N\*F1)).

OUTPUT:



Date: PROLOG PROGRAM FOR GETTING INPUT

AND PROCESSING OUTPUT

Ex. No: 10(B)

B).GCD OF TWO NUMBERS

AIM:

To write a prolog program to implement GCD of Two numbers.

ALGORITHM:

STEP 1: Start the program. Get input for the two numbers

STEP 2: Declare the facts and rules for GCD of two numbers.

STEP 3: Perform suitable operations.

STEP 4: Post Queries related to the given problem in the Query box.

STEP 5: Compile and execute the program.

STEP 6: Get the answers for the queries asked and print the result.

STEP 7: End

PROGRAM:

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

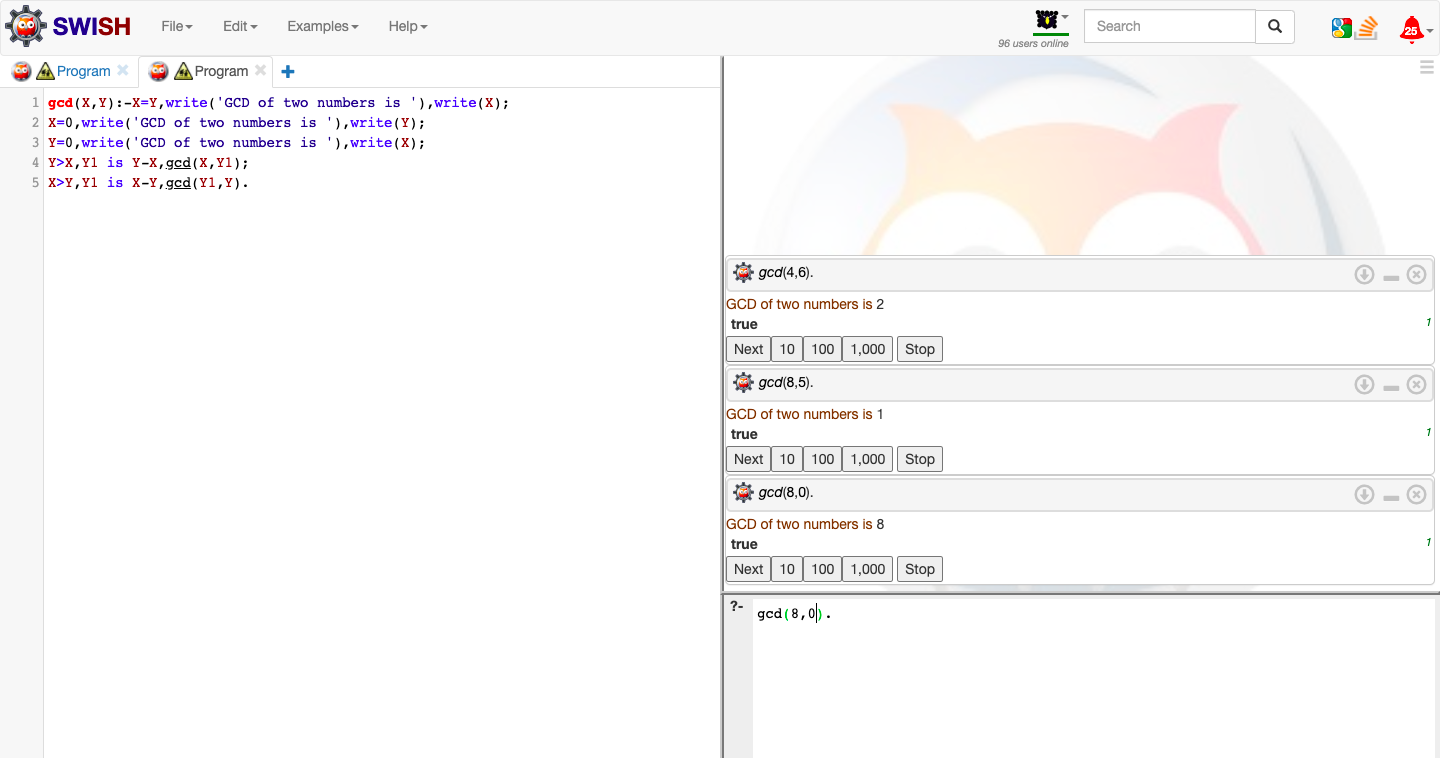
X=0,write('GCD of two numbers is '),write(Y);

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:



Date: PROLOG PROGRAM-WORKING WITH LISTS

Ex. No: 11(A)

A)PRINTING ALL ELEMENTS OF A LIST

AIM :

To write a prolog program to implement List and perform given operations to print list, append list and to check whether a member is present or not in the list.

ALGORITHM:

STEP 1: Start the program. Get inputs.

STEP 2: Declare the facts and rules for List.

STEP 3: Perform suitable operations.

STEP 4: Post Queries related to the given problem in the Query box.

STEP 5: Compile and execute the program.

STEP 6: Get the answers for the queries asked and print the result.

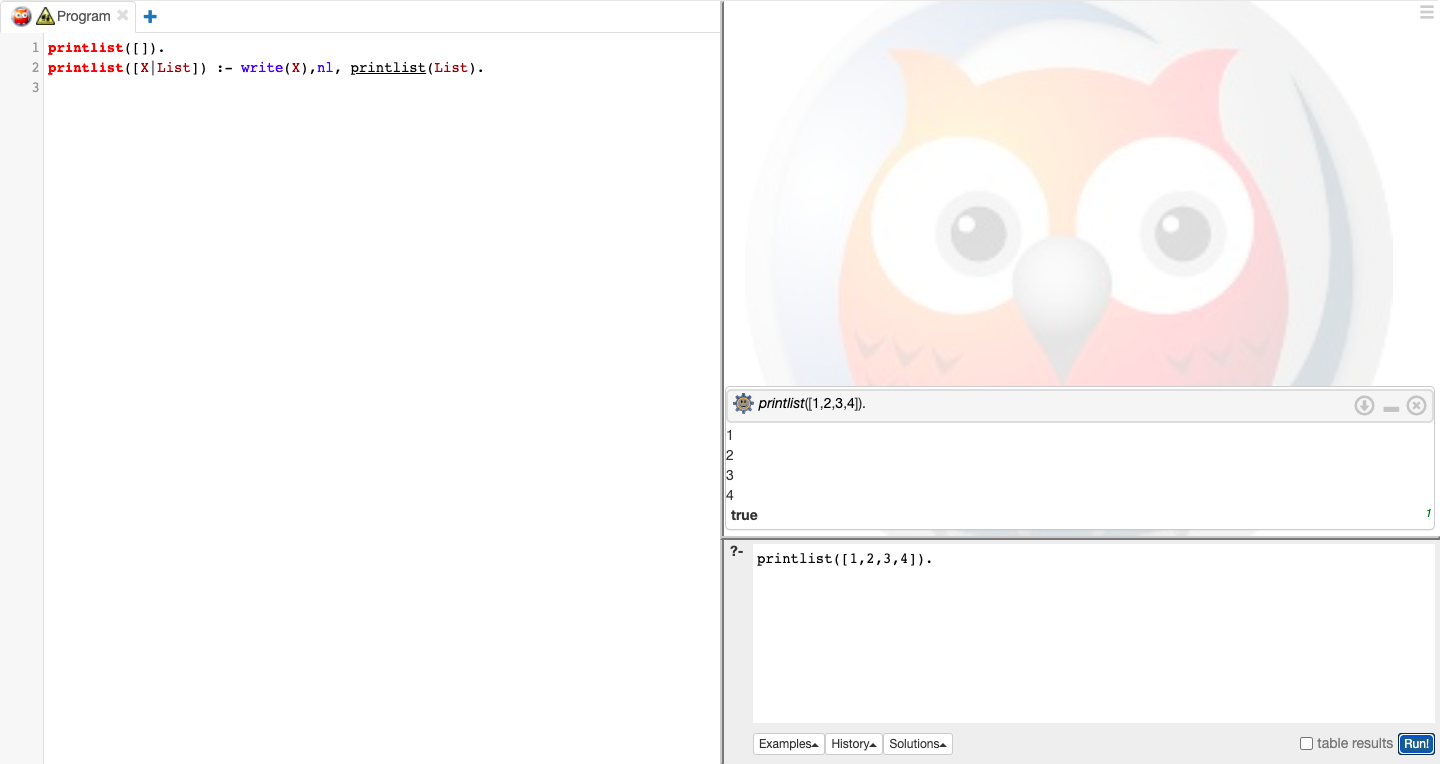
STEP 7: End

PROGRAM:

printlist([]).

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

OUTPUT:



Date: PROLOG PROGRAM-WORKING WITH LISTS

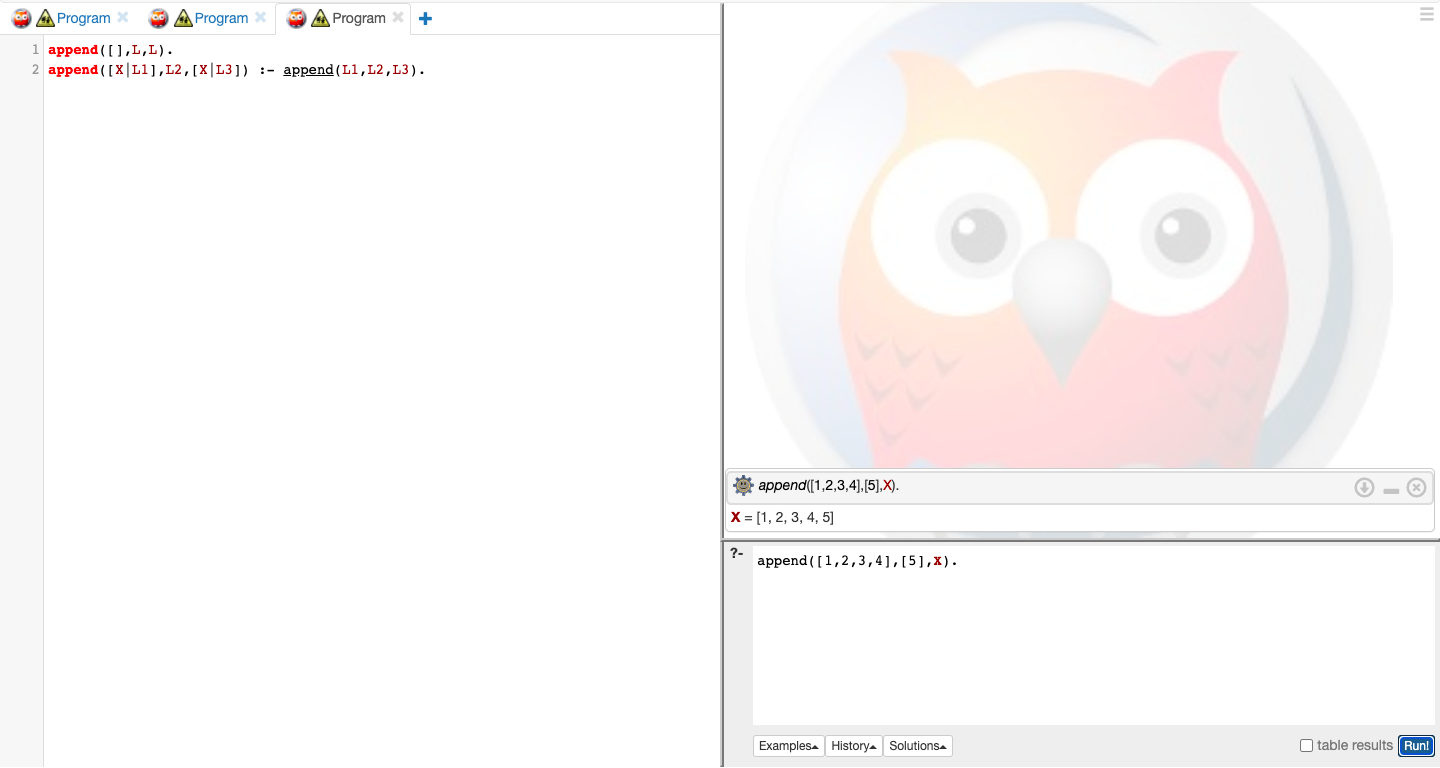
Ex. No: 11(B)

B)TO APPEND AN INTEGER INTO THE LIST

PROGRAM:

append([],L,L).

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

OUTPUT:

Date: PROLOG PROGRAM-WORKING WITH LISTS

Ex. No: 11(C)

C) LIST MEMBERSHIP

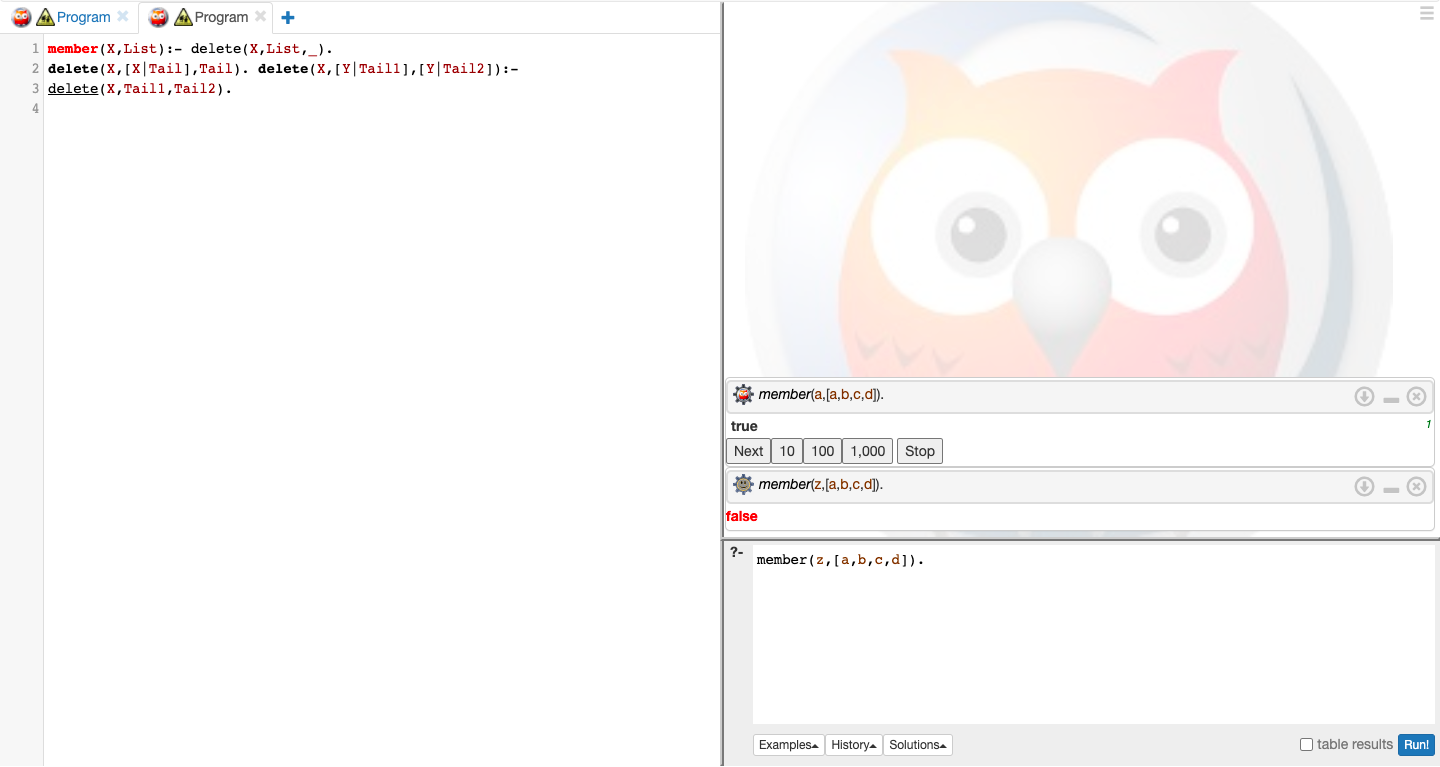
PROGRAM:

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

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

delete(X,Tail1,Tail2).

OUTPUT:



RESULT : Therefore, the above prolog program to implement List is compiled and executed successfully.

Date: PROLOG PROGRAM FOR MEDICAL DIAGNOSIS

Ex. No: 12

AIM:

To write a prolog program to implement Medical Diagnosis System.

ALGORITHM:

STEP 1: Start the program. Get inputs.

STEP 2: Declare the facts and rules for Medical Diagnosis System.

STEP 3: Perform suitable operations.

STEP 4: Post Queries related to the given problem in the Query box.

STEP 5: Compile and execute the program.

STEP 6: Get the answers for the queries asked and print the result.

STEP 7: End

PROGRAM:

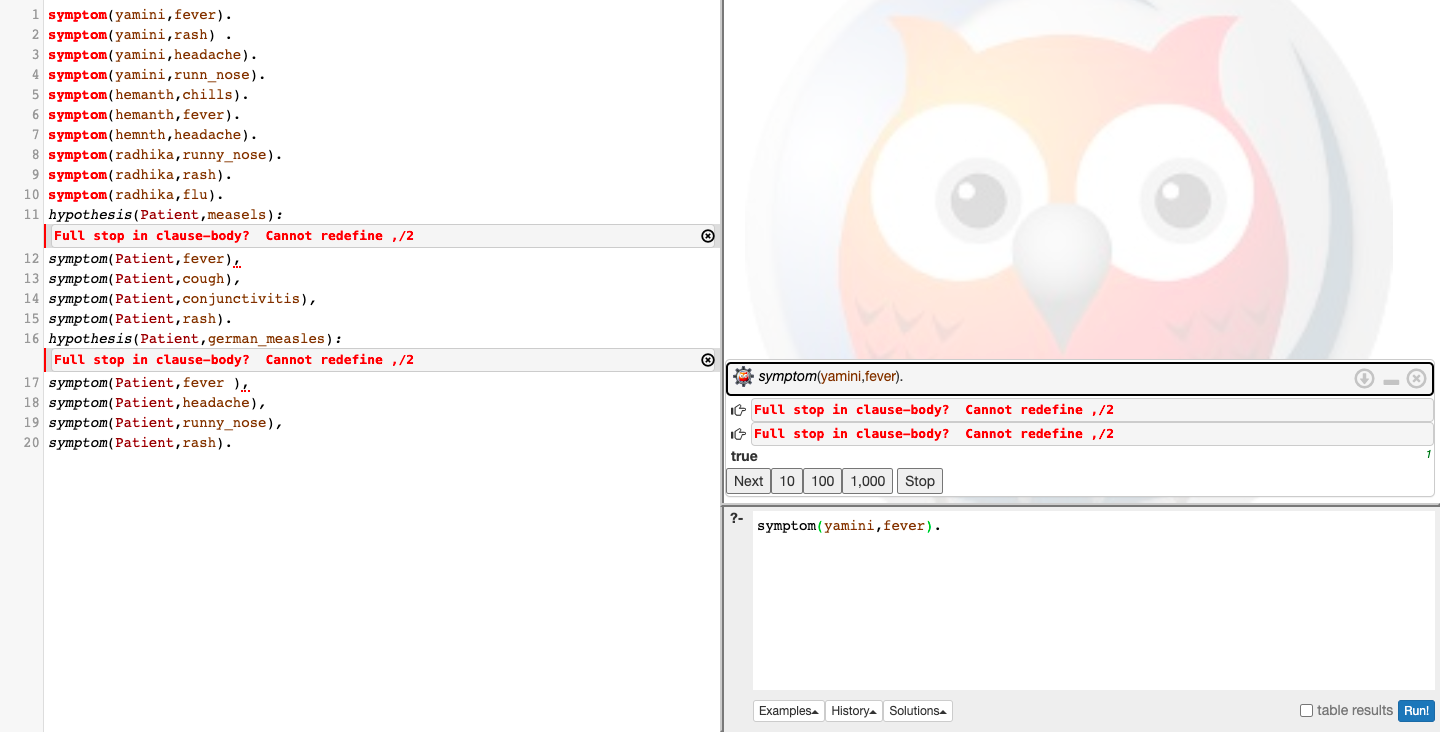
domains disease,indication,name=symbol predicates

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:



RESULT:

Therefore, the above prolog program to implement Medical Diagnosis System is compiled and executed successfully.

Date: WEB BLOG USING WORDPRESS

Ex. No: 13

AIM:

TO CREATE A WEB BLOG USING WORDPRESS

ALGORITHM:

STEP I:- CHOOSE WORDPRESS AS YOUR WEBSITE PLATFORM

There are many website platforms that you can use when building a new site–Content Management Systems (CMS) is what they’re usually called.

The idea of a CMS is to give you some easy-to-use tools so that you’re able to edit your site’s content without any knowledge of coding. For the most part – from the user’s point of view–those CMS look much like the familiar interfaces at Facebook or Google Docs. You basically create new pages or documents ,and then have them published to the web.

key details aboutWordPress:

• it’s open source

• it’s free

• it’s the ultimate DIY solution for website building

• it’s extra versatile – can run any type of website

• it’s fast, optimized, and secure

* it’s SEO-ready – makes promotion easier



STEP 2: PICK A NAME FOR YOUR WEBSITE

There are nearly 2 billion websites online on the web. Meaning, staying original can be quite challenging. It’s a really good idea to construct our website’s name (and thus your domain name) around either the name of your organization (the most obvious approach) or a phrase that’s associated with the niche you’re in, but with some added words for better brand ability.

In short, a good domain name should be:

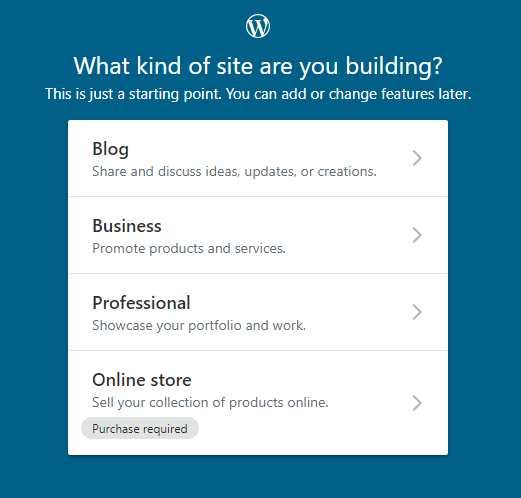
❖ brandable– uniquesounding, like nothing else that’s out there in the market

❖ easy to memorize

❖ short – those are also easier to memorize

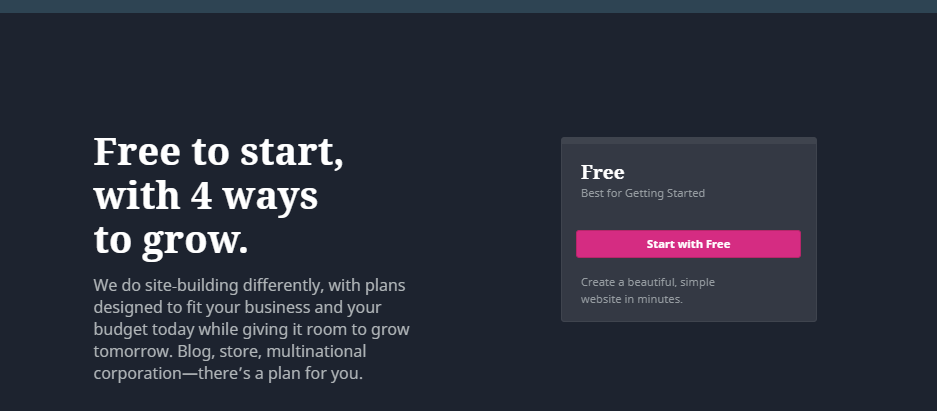
❖ easy to type and hard to mix up – you don’t want people to be wondering how to spell your site’s name

❖ including niche-related keywords – for instance, if you do anything with , it would be cool to have “pizza” somewhere in the name of the site; it works the same in non-pizza industries as well.

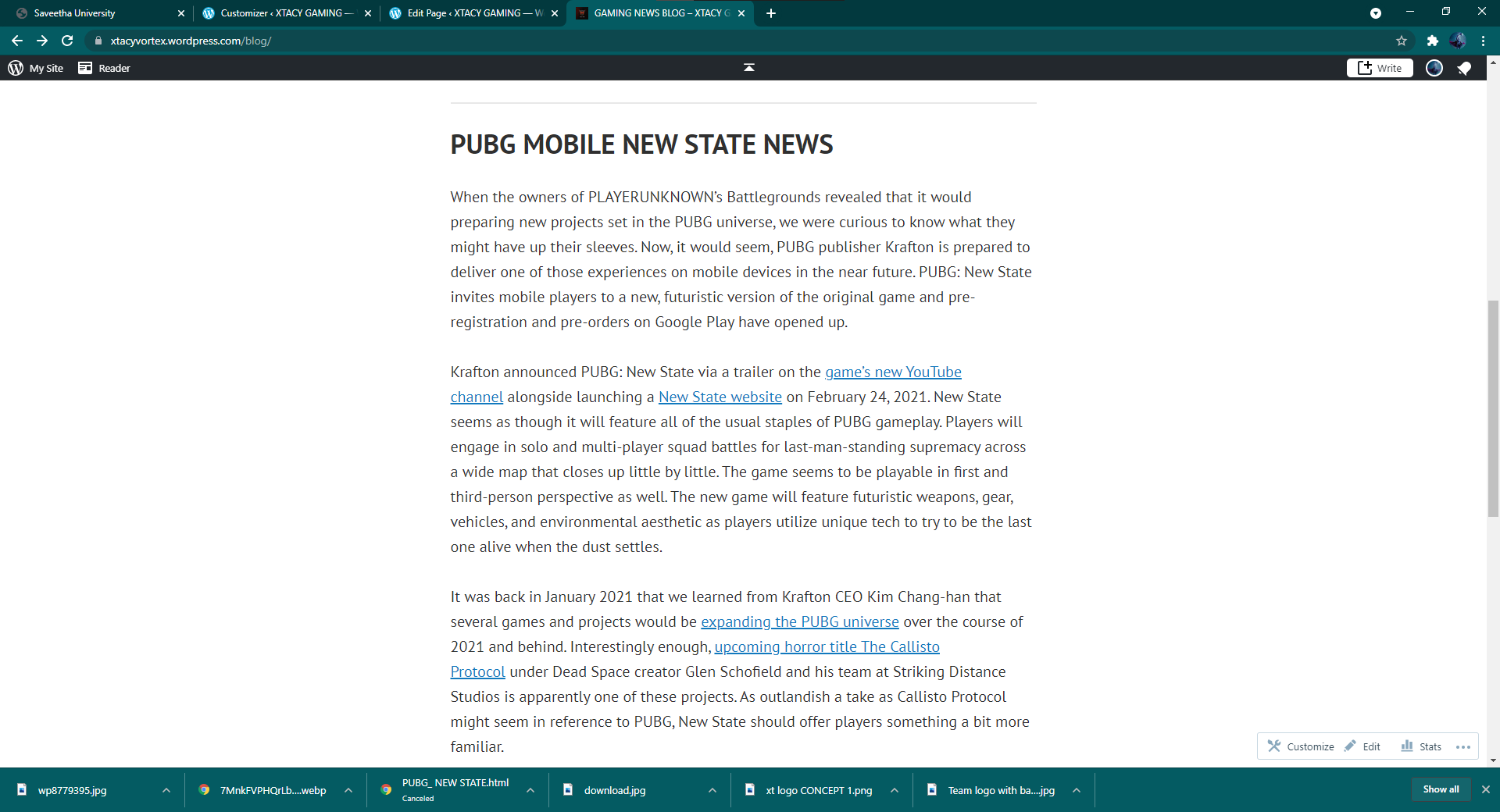




STEP 3:-CREATE A BLOG

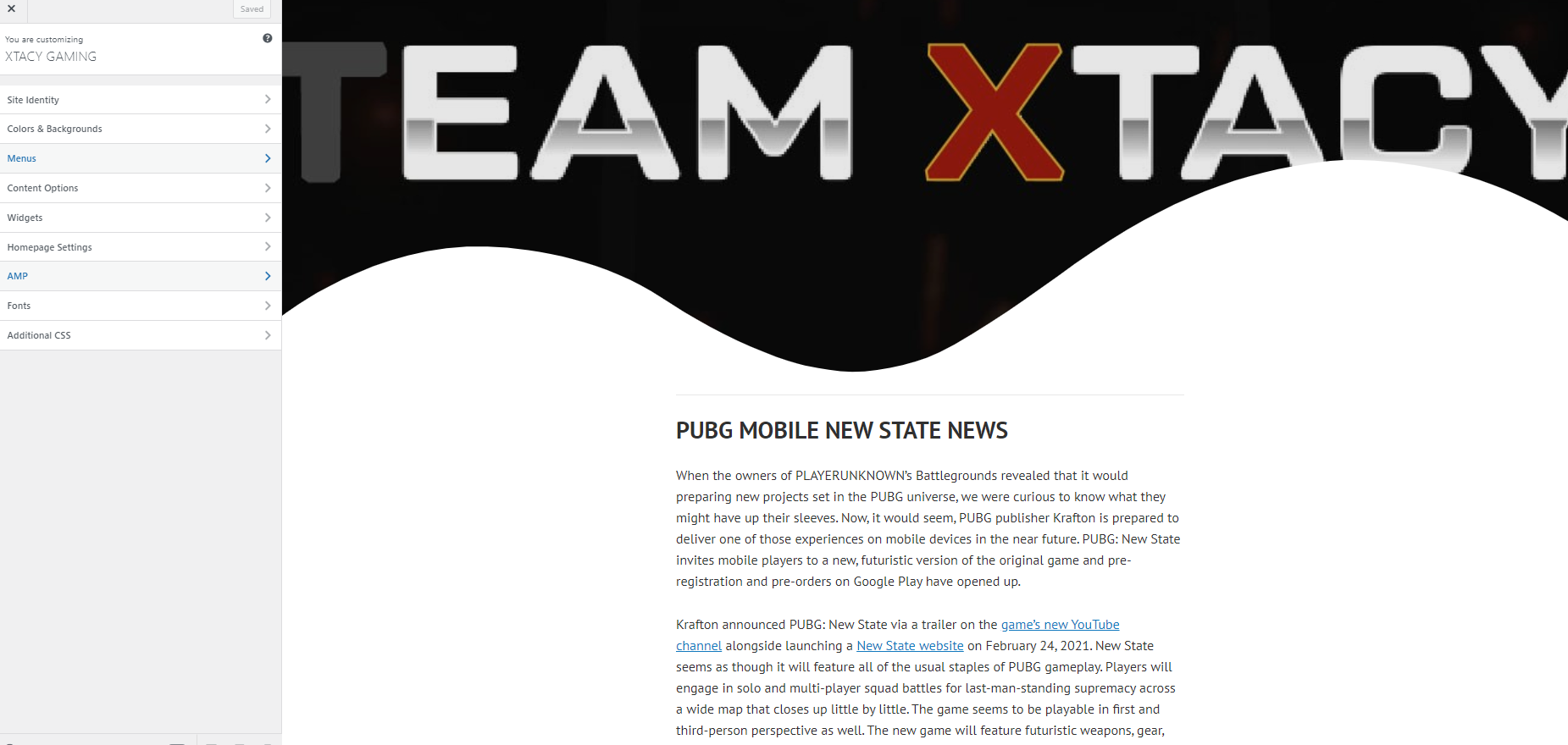


STEP 4:-UPDATE BLOG



STEP 5: UPDATE BLOG

A blog(as well as marketing through content–aka “GAMING” in general)is among the most effective ways to promote not only your website but also any products that you might want to sell through that website.



OUTPUT:

Blog site: <https://xtacyvortex.wordpress.com/>

RESULT : Therefore, the blog is successfully created.