Simple calculator

```
def add(x, y):
    return x + y
def subtract(x, y):
    return x - y
def multiply(x, y):
    return x * y
def divide(x, y):
    return x / y
print("Select operation.")
print("1.Add")
print("2.Subtract")
print("3.Multiply")
print("4.Divide")
while True:
    choice = input("Enter choice(1/2/3/4): ")
    if choice in ('1', '2', '3', '4'):
        try:
            num1 = float(input("Enter first number: "))
            num2 = float(input("Enter second number: "))
        except ValueError:
            print("Invalid input. Please enter a number.")
            continue
        if choice == '1':
            print(num1, "+", num2, "=", add(num1, num2))
        elif choice == '2':
            print(num1, "-", num2, "=", subtract(num1, num2))
        elif choice == '3':
            print(num1, "*", num2, "=", multiply(num1, num2))
        elif choice == '4':
            print(num1, "/", num2, "=", divide(num1, num2))
        next_calculation = input("Let's do next calculation?
(yes/no): ")
        if next_calculation == "no":
          break
    else:
        print("Invalid Input")
```

ADD TWO MATRICES

```
X = [[12,7,3],
    [4 ,5,6],
    [7 ,8,9]]

Y = [[5,8,1],
    [6,7,3],
    [4,5,9]]

result = [[0,0,0],
          [0,0,0],
          [0,0,0]]
for i in range(len(X)):
   for j in range(len(X[0])):
```

```
        result[i][j] = X[i][j] + Y[i][j]

for r in result:
   print(r)
```

Transpose Of a Matrix

```
X = [[12,7],
    [4 ,5],
    [3 ,8]]
result = [[0,0,0],
          [0,0,0]]
for i in range(len(X)):
   for j in range(len(X[0])):
       result[j][i] = X[i][j]
for r in result:
   print(r)
```

Sort The Sentence in Alphabetical order

```
a=input("enter string:")
my_str = a
words = [word.lower() for word in my_str.split()]
words.sort()
print("The sorted words are:")
for word in words:
    print(word)
```

List Operations& methods

```
iList = [10, 20, 30, 40, 50, 60, 70, 80, 90, 100]


print('iList: ',iList)

print('first element: ',iList[0])

print('fourth element: ',iList[3])

print('iList elements from 0 to 4 index:',iList[0: 5])

print('3rd or -7th element:',iList[-7])



iList.append(111)
print('iList after append():',iList)
```

```
print('index of \'80\': ',iList.index(80))


iList.sort()
print('after sorting: ', iList);


print('Popped elements is: ',iList.pop())
print('after pop(): ', iList);


iList.remove(80)
print('after removing \'80\': ',iList)



iList.insert(2, 100)
print('after insert: ', iList)


print('number of occurences of \'100\': ', iList.count(100))


iList.extend([11, 22, 33])
print('after extending:', iList)

iList.reverse()
print('after reversing:',iList)
```

SET OPERATIONS

```
# sets are define
A = {0, 2, 4, 6, 8};
B = {1, 2, 3, 4, 5};

# union
print("Union :", A | B)

# intersection
print("Intersection :", A & B)

# difference
print("Difference :", A - B)

# symmetric difference
print("Symmetric difference :", A ^ B)
```

Generate Calender

```
import calendar
yy = int(input("enter year:"))
mm = int(input("enter month:"))
```

```
print(calendar.month(yy, mm))
```

REMOVE PUNCTUATIONS IN A GIVEN STRING

```
punctuations = '''!()-[]{};:'"\,<>./?@#$%^&*_~'''
a=input("enter string:")
my_str = a
no_punct = ""
for char in my_str:
   if char not in punctuations:
       no_punct = no_punct + char
print(no_punct)
```

8-Puzzle problem

```
import copy

# Importing the heap methods from the python
# library for the Priority Queue
from heapq import heappush, heappop

# This particular var can be changed to transform
# the program from 8 puzzle(n=3) into 15
# puzzle(n=4) and so on ...
n = 3

# bottom, left, top, right
rows = [ 1, 0, -1, 0 ]
cols = [ 0, -1, 0, 1 ]

# creating a class for the Priority Queue
class priorityQueue:

    # Constructor for initializing a
    # Priority Queue
    def __init__(self):
        self.heap = []

    # Inserting a new key 'key'
    def push(self, key):
        heappush(self.heap, key)

    # funct to remove the element that is minimum,
    # from the Priority Queue
    def pop(self):
        return heappop(self.heap)

    # funct to check if the Queue is empty or not
    def empty(self):
        if not self.heap:
            return True
```

```
        else:
            return False

# structure of the node
class nodes:

    def __init__(self, parent, mats, empty_tile_posi,
                 costs, levels):

        # This will store the parent node to the
        # current node And helps in tracing the
        # path when the solution is visible
        self.parent = parent

        # Useful for Storing the matrix
        self.mats = mats

        # useful for Storing the position where the
        # empty space tile is already existing in the matrix
        self.empty_tile_posi = empty_tile_posi

        # Store no. of misplaced tiles
        self.costs = costs

        # Store no. of moves so far
        self.levels = levels

    # This func is used in order to form the
    # priority queue based on
    # the costs var of objects
    def __lt__(self, nxt):
        return self.costs < nxt.costs

# method to calc. the no. of
# misplaced tiles, that is the no. of non-blank
# tiles not in their final posi
def calculateCosts(mats, final) -> int:

    count = 0
    for i in range(n):
        for j in range(n):
            if ((mats[i][j]) and
                (mats[i][j] != final[i][j])):
                count += 1

    return count

def newNodes(mats, empty_tile_posi, new_empty_tile_posi,
             levels, parent, final) -> nodes:

    # Copying data from the parent matrixes to the present matrixes
    new_mats = copy.deepcopy(mats)

    # Moving the tile by 1 position
    x1 = empty_tile_posi[0]
    y1 = empty_tile_posi[1]
```

```
    x2 = new_empty_tile_posi[0]
    y2 = new_empty_tile_posi[1]
    new_mats[x1][y1], new_mats[x2][y2] = new_mats[x2][y2],
new_mats[x1][y1]

    # Setting the no. of misplaced tiles
    costs = calculateCosts(new_mats, final)

    new_nodes = nodes(parent, new_mats, new_empty_tile_posi,
                      costs, levels)
    return new_nodes

# func to print the N by N matrix
def printMatsrix(mats):

    for i in range(n):
        for j in range(n):
            print("%d " % (mats[i][j]), end = " ")

        print()

# func to know if (x, y) is a valid or invalid
# matrix coordinates
def isSafe(x, y):

    return x >= 0 and x < n and y >= 0 and y < n

# Printing the path from the root node to the final node
def printPath(root):

    if root == None:
        return

    printPath(root.parent)
    printMatsrix(root.mats)
    print()

# method for solving N*N - 1 puzzle algo
# by utilizing the Branch and Bound technique. empty_tile_posi is
# the blank tile position initially.
def solve(initial, empty_tile_posi, final):

    # Creating a priority queue for storing the live
    # nodes of the search tree
    pq = priorityQueue()

    # Creating the root node
    costs = calculateCosts(initial, final)
    root = nodes(None, initial,
                 empty_tile_posi, costs, 0)

    # Adding root to the list of live nodes
    pq.push(root)

    # Discovering a live node with min. costs,
    # and adding its children to the list of live
```

```
    # nodes and finally deleting it from
    # the list.
    while not pq.empty():

        # Finding a live node with min. estimatsed
        # costs and deleting it form the list of the
        # live nodes
        minimum = pq.pop()

        # If the min. is ans node
        if minimum.costs == 0:

            # Printing the path from the root to
            # destination;
            printPath(minimum)
            return

        # Generating all feasible children
        for i in range(n):
            new_tile_posi = [
                minimum.empty_tile_posi[0] + rows[i],
                minimum.empty_tile_posi[1] + cols[i], ]

            if isSafe(new_tile_posi[0], new_tile_posi[1]):

                # Creating a child node
                child = newNodes(minimum.mats,
                                 minimum.empty_tile_posi,
                                 new_tile_posi,
                                 minimum.levels + 1,
                                 minimum, final,)

                # Adding the child to the list of live nodes
                pq.push(child)

# Main Code

# Initial configuration
# Value 0 is taken here as an empty space
initial = [ [ 1, 2, 3 ],
            [ 5, 6, 0 ],
            [ 7, 8, 4 ] ]

# Final configuration that can be solved
# Value 0 is taken as an empty space
final = [ [ 1, 2, 3 ],
        [ 5, 8, 6 ],
        [ 0, 7, 4 ] ]

# Blank tile coordinates in the
# initial configuration
empty_tile_posi = [ 1, 2 ]

# Method call for solving the puzzle
solve(initial, empty_tile_posi, final)
```

8-QUEEN PROBLEM

```
print ("Enter the number of queens")
N = int(input())
board = [[0]*N for _ in range(N)]
def attack(i, j):
    for k in range(0,N):
        if board[i][k]==1 or board[k][j]==1:
            return True
    for k in range(0,N):
        for l in range(0,N):
            if (k+l==i+j) or (k-l==i-j):
                if board[k][l]==1:
                    return True
    return False
def N_queens(n):
    if n==0:
        return True
    for i in range(0,N):
        for j in range(0,N):
            if (not(attack(i,j))) and (board[i][j]!=1):
                board[i][j] = 1
                if N_queens(n-1)==True:
                    return True
                board[i][j] = 0
    return False
N_queens(N)
for i in board:
    print (i)
```

WATER JUG PROBLEM

```
from collections import deque
def Solution(a, b, target):
     m = {}
     isSolvable = False
     path = []

     q = deque()
     q.append((0, 0))

     while (len(q) > 0):
          u = q.popleft()
          if ((u[0], u[1]) in m):
               continue
          if ((u[0] > a or u[1] > b or
               u[0] < 0 or u[1] < 0)):
               continue
          path.append([u[0], u[1]])
```

```
            m[(u[0], u[1])] = 1

            if (u[0] == target or u[1] == target):
                  isSolvable = True

                  if (u[0] == target):
                        if (u[1] != 0):
                              path.append([u[0], 0])
                  else:
                        if (u[0] != 0):

                              path.append([0, u[1]])

                  sz = len(path)
                  for i in range(sz):
                        print("(", path[i][0], ",",
                              path[i][1], ")")
                  break

            q.append([u[0], b])
            q.append([a, u[1]])

            for ap in range(max(a, b) + 1):
                  c = u[0] + ap
                  d = u[1] - ap

                  if (c == a or (d == 0 and d >= 0)):
                        q.append([c, d])

                  c = u[0] - ap
                  d = u[1] + ap

                  if ((c == 0 and c >= 0) or d == b):
                        q.append([c, d])

            q.append([a, 0])

            q.append([0, b])

      if (not isSolvable):
            print("Solution not possible")

if __name__ == '__main__':

      Jug1, Jug2, target = 4, 3, 2
      print("Path from initial state "
            "to solution state ::")

      Solution(Jug1, Jug2, target)
```

Cript-arithmetic problem

```
import itertools
```

```
def solve2():
    letters = ('s', 'e', 'n', 'd', 'm', 'o', 'r', 'y')
    digits = range(10)
    for perm in itertools.permutations(digits, len(letters)):
        sol = dict(zip(letters, perm))
        if sol['s'] == 0 or sol['m'] == 0:
            continue
        send = 1000 * sol['s'] + 100 * sol['e'] + 10 * sol['n'] +
sol['d']
        more = 1000 * sol['m'] + 100 * sol['o'] + 10 * sol['r'] +
sol['e']
        money = 10000 * sol['m'] + 1000 * sol['o'] + 100 * sol['n']
+ 10 * sol['e'] + sol['y']
        if send + more == money:
          print(" send"," more"," money")
          return send, more, money
print(solve2())
```

missionaries cannibal

```
#Python program to illustrate Missionaries & cannibals Problem

#This code is contributed by Sunit Mal

print("\n")

print("\tGame Start\nNow the task is to move all of them to right side of the river")

print("rules:\n1. The boat can carry at most two people\n2. If cannibals num greater than missionaries then the cannibals would eat the missionaries\n3. The boat cannot cross the river by itself with no people on board")

lM = 3          #lM = Left side Missionaries number

lC = 3          #lC = Laft side Cannibals number

rM=0            #rM = Right side Missionaries number

rC=0            #rC = Right side cannibals number

userM = 0       #userM = User input for number of missionaries for right to left side travel

userC = 0       #userC = User input for number of cannibals for right to left travel

k = 0

print("\nM M M C C C |  --- | \n")

try:

        while(True):

                while(True):

                        print("Left side -> right side river travel")
```

```
            #uM = user input for number of missionaries for left to right travel
            #uC = user input for number of cannibals for left to right travel
            uM = int(input("Enter number of Missionaries travel => "))
            uC = int(input("Enter number of Cannibals travel => "))

            if((uM==0)and(uC==0)):
                        print("Empty travel not possible")
                        print("Re-enter : ")
            elif(((uM+uC) <= 2)and((lM-uM)>=0)and((lC-uC)>=0)):
                        break
            else:
                        print("Wrong input re-enter : ")
lM = (lM-uM)
lC = (lC-uC)
rM += uM
rC += uC

print("\n")
for i in range(0,lM):
            print("M ",end="")
for i in range(0,lC):
            print("C ",end="")
print("| --> | ",end="")
for i in range(0,rM):
            print("M ",end="")
for i in range(0,rC):
            print("C ",end="")
print("\n")

k +=1
```

```
        if(((lC==3)and (lM == 1))or((lC==3)and(lM==2))or((lC==2)and(lM==1))or((rC==3)and
(rM == 1))or((rC==3)and(rM==2))or((rC==2)and(rM==1))):
                print("Cannibals eat missionaries:\nYou lost the game")


                break


        if((rM+rC) == 6):
                print("You won the game : \n\tCongrats")
                print("Total attempt")
                print(k)
                break
        while(True):
                print("Right side -> Left side river travel")
                userM = int(input("Enter number of Missionaries travel => "))
                userC = int(input("Enter number of Cannibals travel => "))


                if((userM==0)and(userC==0)):
                                print("Empty travel not possible")
                                print("Re-enter : ")
                elif(((userM+userC) <= 2)and((rM-userM)>=0)and((rC-userC)>=0)):
                        break
                else:
                        print("Wrong input re-enter : ")
        lM += userM
        lC += userC
        rM -= userM
        rC -= userC


        k +=1
        print("\n")
        for i in range(0,lM):
```

```
                print("M ",end="")
        for i in range(0,lC):
                print("C ",end="")
        print("| <-- | ",end="")
        for i in range(0,rM):
                print("M ",end="")
        for i in range(0,rC):
                print("C ",end="")
        print("\n")



        if(((lC==3)and (lM == 1))or((lC==3)and(lM==2))or((lC==2)and(lM==1))or((rC==3)and
(rM == 1))or((rC==3)and(rM==2))or((rC==2)and(rM==1))):
                print("Cannibals eat missionaries:\nYou lost the game")
                break
except EOFError as e:
        print("\nInvalid input please retry !!")
```

BFS

```
# Python3 Program to print BFS traversal
# from a given source vertex. BFS(int s)
# traverses vertices reachable from s.
from collections import defaultdict

# This class represents a directed graph
# using adjacency list representation
class Graph:

      # Constructor
      def __init__(self):

            # default dictionary to store graph
            self.graph = defaultdict(list)

      # function to add an edge to graph
      def addEdge(self,u,v):
            self.graph[u].append(v)
```

```
      # Function to print a BFS of graph
      def BFS(self, s):

            # Mark all the vertices as not visited
            visited = [False] * (len(self.graph))

            # Create a queue for BFS
            queue = []

            # Mark the source node as
            # visited and enqueue it
            queue.append(s)
            visited[s] = True

            while queue:

                  # Dequeue a vertex from
                  # queue and print it
                  s = queue.pop(0)
                  print (s, end = " ")

                  # Get all adjacent vertices of the
                  # dequeued vertex s. If a adjacent
                  # has not been visited, then mark it
                  # visited and enqueue it
                  for i in self.graph[s]:
                        if visited[i] == False:
                              queue.append(i)
                              visited[i] = True

# Driver code

# Create a graph given in
# the above diagram
g = Graph()
g.addEdge(0, 1)
g.addEdge(0, 2)
g.addEdge(1, 2)
g.addEdge(2, 0)
g.addEdge(2, 3)
g.addEdge(3, 3)

print ("Following is Breadth First Traversal"
                   " (starting from vertex 2)")
g.BFS(2)

# This code is contributed by Neelam Yadav
```

DFS

```
# 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')
```

Travelling salesman problem

```
from sys import maxsize
from itertools import permutations
V = 4

def travellingSalesmanProblem(graph, s):


      vertex = []
      for i in range(V):
            if i != s:
                  vertex.append(i)


      min_path = maxsize
      next_permutation=permutations(vertex)
      for i in next_permutation:


            current_pathweight = 0


            k = s
            for j in i:
                  current_pathweight += graph[k][j]
                  k = j
            current_pathweight += graph[k][s]


            min_path = min(min_path, current_pathweight)
```

```
    return min_path


if __name__ == "__main__":

    graph = [[0, 10, 15, 20], [10, 0, 35, 25],
             [15, 35, 0, 30], [20, 25, 30, 0]]
    s = 0
    print(travellingSalesmanProblem(graph, s))
```

A* ALGORITHM

```
class Node():
    def __init__(self, parent=None, position=None):
        self.parent = parent
        self.position = position
        self.g = 0
        self.h = 0
        self.f = 0
    def __eq__(self, other):
        return self.position == other.position
def astar(maze, start, end):
    start_node = Node(None, start)
    start_node.g = start_node.h = start_node.f = 0
    end_node = Node(None, end)
    end_node.g = end_node.h = end_node.f = 0
    open_list = []
    closed_list = []
    open_list.append(start_node)
    while len(open_list) > 0:
        current_node = open_list[0]
        current_index = 0
        for index, item in enumerate(open_list):
            if item.f < current_node.f:
                current_node = item
                current_index = index
        open_list.pop(current_index)
        closed_list.append(current_node)
        if current_node == end_node:
            path = []
            current = current_node
            while current is not None:
                path.append(current.position)
                current = current.parent
            return path[::-1]
        children = []
```

```
        for new_position in [(0, -1), (0, 1), (-1, 0), (1, 0), (-1,
-1), (-1, 1), (1, -1), (1, 1)]:
            node_position = (current_node.position[0] +
new_position[0], current_node.position[1] + new_position[1])
            if node_position[0] > (len(maze) - 1) or
node_position[0] < 0 or node_position[1] > (len(maze[len(maze)-1]) -
1) or node_position[1] < 0:
                continue
            if maze[node_position[0]][node_position[1]] != 0:
                continue
            new_node = Node(current_node, node_position)
            children.append(new_node)
        for child in children:
            for closed_child in closed_list:
                if child == closed_child:
                    continue
            child.g = current_node.g + 1
            child.h = ((child.position[0] - end_node.position[0]) **
2) + ((child.position[1] - end_node.position[1]) ** 2)
            child.f = child.g + child.h
            for open_node in open_list:
                if child == open_node and child.g > open_node.g:
                    continue
            open_list.append(child)
def main():
    maze = [[0, 0, 0, 0, 1, 0, 0, 0, 0, 0],
            [0, 0, 0, 0, 1, 0, 0, 0, 0, 0],
            [0, 0, 0, 0, 1, 0, 0, 0, 0, 0],
            [0, 0, 0, 0, 1, 0, 0, 0, 0, 0],
            [0, 0, 0, 0, 1, 0, 0, 0, 0, 0],
            [0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
            [0, 0, 0, 0, 1, 0, 0, 0, 0, 0],
            [0, 0, 0, 0, 1, 0, 0, 0, 0, 0],
            [0, 0, 0, 0, 1, 0, 0, 0, 0, 0],
            [0, 0, 0, 0, 0, 0, 0, 0, 0, 0]]
    start = (0, 0)
    end = (7, 6)
    path = astar(maze, start, end)
    print(path)
if __name__ == '__main__':
    main()
```

Map coloring to implement CSP

```
def addColor(R, province, color):
    ans = []
    for rr in R:
        res = checkRestriction(rr, province, color)
        if res == False:
            return False
```

```
        elif res == None:
            continue
        else:
            ans.append(res)
    return ans

# checks if the restrition rr allows the given province to have the
given color
# returns false if not possible, otherwise returns the new
restriction
def checkRestriction(rr, province, color):
    #finding the index of the province (saved to index)
    index = -1
    other = -1
    if rr[0] == province:
        index = 0
        other = 1
    elif rr[1] == province:
        index = 1
        other = 0
    else:
        return rr

    if isinstance(rr[other], int):
        # other component is a color
        if (color != rr[other]):
            return None
        else:
            return False
    else:
        return [rr[other], color]


# solving the CSP by variable elimination
# recursive structure: ci is the province index to be colored (0 =
bc, 1 = ab, etc)
# n is the number of colors
# provinces is a list of provinces
# if coloring is possible returns the province-> color map,
otherwise False
def solveCSP(provinces, n, R, ci):
    if (ci == 0):
        # in the beginning any color can be assigned to the first
province, lets say 1
        newR = addColor(R, provinces[0], 1)
        if (newR == False):
            return False
        ans = {provinces[0]:1}
        res = solveCSP(provinces, n, newR, 1)
        if (res == False):
            return False
        ans.update(res)
        return ans
    elif (ci == len(provinces)):
        return {}
```

```
    # branching over all possible colors for provinces[ci]
    for color in range (1,n+1):
        ans = {provinces[ci]:color}
        newR = addColor(R, provinces[ci], color)
        if (newR == False):
            continue
        res = solveCSP(provinces, n, newR, ci+1)
        if (res == False):
            continue
        #print(ans)
        #print(res)
        #print("============")
        ans.update(res)
        return ans

    # no choice for the current province
    return False


# main program starts
# ====================================================

n=5 #int(input("Enter the number of color"))
colors=[]
for i in range(1,n+1):
    colors.append(i)
#print(colors)

# creating map of canada
# cmap[x] gives the neighbors of the province x
cmap = {}
cmap["ab"] = ["bc","nt","sk"]
cmap["bc"] = ["yt", "nt", "ab"]
cmap["mb"] = ["sk","nu","on"]
cmap["nb"] = ["qc", "ns", "pe"]
cmap["ns"] = ["nb", "pe"]
cmap["nl"] = ["qc"]
cmap["nt"] = ["bc", "yt", "ab", "sk", "nu"]
cmap["nu"] = ["nt", "mb"]
cmap["on"] = ["mb", "qc"]
cmap["pe"] = ["nb", "ns"]
cmap["qc"] = ["on", "nb", "nl"]
cmap["sk"] = ["ab", "mb", "nt"]
cmap["yt"] = ["bc", "nt"]

# CSP restrictions
# each restriction is modeled as a pair [a,b] which means the
province a's
# color is not equal to b, where b is either a color (a number 1 to
n) or
# another province. Examples ['bc', 'ab'] means the color of bc
should
# not be equal to ab -- ["bc",4] means the color of bc should not be
4
# R is the list of restrictions
```

```
R = []

# initiaitiong restrictions based on the province neighborhood

for x in cmap:
    for y in cmap[x]:
        R.append([x,y])

# initiating a list of provinces
provinces = []
for p in cmap:
    provinces.append(p)

#print(solveCSP(provinces, 3, R, 0))

while(1):
    num=int(input("Enter number of the color? "))
    print(solveCSP(provinces, num, R, 0))




#print(R)
#print(" ================= ")
#print(checkRestriction(["ab",4],"ab",4))
#R = addColor(R, 'bc', 4)
#print(R)

#print(" ================= ")
#print(checkRestriction(["ab",4],"ab",4))
#R = addColor(R, 'ab', 4)
#print(R)
```

TIC TAC TOE

```
import os
import time

board = [' ',' ',' ',' ',' ',' ',' ',' ',' ',' ']
player = 1

########win Flags##########
Win = 1
Draw = -1
Running = 0
Stop = 1
###########################
Game = Running
Mark = 'X'
def DrawBoard():
```

```
    print(" %c | %c | %c " % (board[1],board[2],board[3]))
    print("_|_|_")
    print(" %c | %c | %c " % (board[4],board[5],board[6]))
    print("_|_|_")
    print(" %c | %c | %c " % (board[7],board[8],board[9]))
    print("   |   |   ")
def CheckPosition(x):
    if(board[x] == ' '):
        return True
    else:
        return False
def CheckWin():
    global Game
    if(board[1] == board[2] and board[2] == board[3] and board[1] !=
' '):
        Game = Win
    elif(board[4] == board[5] and board[5] == board[6] and board[4]
!= ' '):
        Game = Win
    elif(board[7] == board[8] and board[8] == board[9] and board[7]
!= ' '):
        Game = Win
    elif(board[1] == board[4] and board[4] == board[7] and board[1]
!= ' '):
        Game = Win
    elif(board[2] == board[5] and board[5] == board[8] and board[2]
!= ' '):
        Game = Win
    elif(board[3] == board[6] and board[6] == board[9] and board[3]
!= ' '):
        Game=Win
    elif(board[1] == board[5] and board[5] == board[9] and board[5]
!= ' '):
        Game = Win
    elif(board[3] == board[5] and board[5] == board[7] and board[5]
!= ' '):
        Game=Win
    elif(board[1]!=' ' and board[2]!=' ' and board[3]!=' ' and
board[4]!=' ' and board[5]!=' ' and board[6]!=' ' and board[7]!=' '
and board[8]!=' ' and board[9]!=' '):
        Game=Draw
    else:
        Game=Running

print("Tic-Tac-Toe Game ")
print("Player 1 [X] --- Player 2 [O]\n")
print()
print()
print("Please Wait...")
time.sleep(3)
while(Game == Running):
    os.system('cls')
    DrawBoard()
    if(player % 2 != 0):
        print("Player 1's chance")
        Mark = 'X'
```

```
    else:
        print("Player 2's chance")
        Mark = 'O'
    choice = int(input("Enter the position between [1-9] where you
want to mark : "))
    if(CheckPosition(choice)):
        board[choice] = Mark
        player+=1
        CheckWin()

os.system('cls')
DrawBoard()
if(Game==Draw):
    print("Game Draw")
elif(Game==Win):
    player-=1
    if(player%2!=0):
        print("Player 1 Won")
    else:
        print("Player 2 Won")
```

MINIMAX

```
# 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
```

ALPHA BETA PRUNING

```
MAX, MIN = 1000, -1000
def minimax(depth, nodeIndex, maximizingPlayer,values, alpha, beta):
      if depth == 3:
            return values[nodeIndex]

      if maximizingPlayer:

            best = MIN

            for i in range(0, 2):

                  val = minimax(depth + 1, nodeIndex * 2 + i,
                                    False, values, alpha, beta)
                  best = max(best, val)
                  alpha = max(alpha, best)

                  if beta <= alpha:
                        break

            return best

      else:
            best = MAX
            for i in range(0, 2):

                  val = minimax(depth + 1, nodeIndex * 2 + i,
                                          True, values, alpha, beta)
                  best = min(best, val)
                  beta = min(beta, best)

                  if beta <= alpha:
                        break

            return best

if __name__ == "__main__":

      values = [3, 5, 6, 9, 1, 2, 0, -1]
```

```
     print("The optimal value is :", minimax(0, 0, True, values,
MIN, MAX))
```

DECISION TREE FOR PLAYING GAME

```
def introduction():
        print '''Welcome to Kevin's European Geography Quizzes.
        Test your knowledge of European geography. \n'''
        difficulty = raw_input('''Do you want to play an easy,
medium, or hard game?
        Please type the number 1 for easy, 2 for medium, or 3 for
hard.\n''' )
        game_chooser(difficulty)

    def game_chooser(difficulty):
        cursor = 0
        difficulty_choice = [easy_game(), medium_game(),
hard_game()]
 #each element of the above list links to a procedure and starts one
of the
 #mini-games.
        while cursor < len(difficulty_choice):
            if difficulty != cursor:
                cursor += 1
            else:
                difficulty_choice[cursor]
                break
```

DECISION TREE FOR RESTAURANT

```
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

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()
```

FEED FWD NEURAL NETWORK

```
import numpy as np
def relu(n):
    if n<0:
        return 0
    else:
        return n
inp=np.array([[-1,2],[2,2],[3,3]])
weights=[np.array([3,3]),np.array([1,5]),np.array([3,3]),np.array([1
,5]),np.array([2,-1])]
for x in inp :
    node0=relu((x*weights[0]).sum())
    node1=relu((x*weights[1]).sum())
    node2=relu(([node0,node1]*weights[2]).sum())
    node3=relu(([node0,node1]*weights[3]).sum())
```

```
op=relu(([node2,node3]*weights[4]).sum())
print(x,op)
```

PROLOG PROGRAMS

1 WPP FOR A DB WITH FNAME,SNAME,SEX,AGE,OCCUOATION.

WRITE THE QUERIES TO RETRIVE THE REQUIRED INFORMATION.

person(frances,wilson,female,28,architect).

person(fred,jones,male,62,doctor).

person(paul,smith,male,45,plumber).

person(martin,williams,male,23,chemist).

person(mary,jones,female,24,programmer).

person(martin,johnson,male,47,solicitor).

man(A):-person(A,B,male,C,D).

INPUT/OUTPUT

?- person(A,B,C,D,E).

A = frances,

B = wilson,

C = female,

D = 28,

E = architect .

2 WPP FOR A DB WITH COUNTRY, WEIGHT.

WRITE THE QUERIES TO RETRIVE THE REQUIRED INFORMATION.

countries([belgium, france, germany, italy, luxembourg, netherlands]).

weight(france, 4).

weight(germany, 4).

weight(italy, 4).

weight(belgium, 2).

weight(netherlands, 2).

weight(luxembourg, 1).

threshold(12).

INPUT/OUTPUT

?- countries(X).

X = [belgium, france, germany, italy, luxembourg, netherlands].

3. WPP FOR A DB WITH NAME, DOB.

WRITE THE QUERIES TO RETRIVE THE REQUIRED INFORMATION.

born(jan, 20,3,1977).

born(jeroen, 2,2,1992).

born(joris, 17,3,1995).

born(jelle, 1,1,2004).

born(jesus, 24,12,2000).

born(joop, 30,4,1989).

born(jannecke, 17,3,1993).

born(jaap, 16,11,1995).

INPUT

? born(A,B,C,D).

?- born(N,D,M,Y).

N = jan,

D = 20,

M = 3,

Y = 1977 ;

N = jeroen,

D = M, M = 2,

Y = 1992 ;

N = joris,

D = 17,

M = 3,

Y = 1995 ;

N = jelle,

D = M, M = 1,

Y = 2004 ;

N = jesus,

D = 24,

M = 12,

Y = 2000 ;

N = joop,

D = 30,

M = 4,

Y = 1989 ;

N = jannecke,

D = 17,

M = 3,

Y = 1993 ;

N = jaap,

D = 16,

M = 11,

Y = 1995.

4. 1 WPP FOR FIBONACCI SERIES

```
fib(0,1).
fib(1,1).
fib(N,F) :- N > 1, fib(N,1,1,F).
fib(2,F1,F2,F) :- F is F1 + F2.
fib(N,F1,F2,F) :- N > 2, N1 is N - 1, NF1 is F1 + F2,
   fib(N1,NF1,F1,F).
```

INPUT/OUTPUT

```
?- fib(5,F).
F = 8
```

5 WPP FOR A DB WITH DOG-NAME, SIZE-SMALL, MEDIUM, BIG.

WRITE THE QUERIES TO RETRIVE THE REQUIRED INFORMATION.

```
dog(fido).
dog(rover).
dog(jane).
```

```
dog(tom).
dog(fred).
dog(henry).
dog(fido).
cat(mary).
cat(harry).
cat(bill).
cat(steve).

small(henry).

medium(harry).
medium(fred).

large(fido).
large(mary).
large(tom).
large(fred).
large(steve).
large(jim).
large(mike).

small_animal(X):- dog(X),small(X).
small_animal(Z):- cat(Z),small(Z).

medium_animal(X):- dog(X),medium(X).
medium_animal(Z):- cat(Z),medium(Z).
```

large_animal(X):- dog(X),large(X).

large_animal(Z):- cat(Z),large(Z).

INPUT/OUTPUT

?- medium(X).

X = harry ;

X = fred.

6 WPP FOR A ANIMAL DB WITH MAMMAL, ANIMAL-NAME, TYPE-SKIN-COLOR.

WRITE THE QUERIES TO RETRIVE THE REQUIRED INFORMATION.

/* Animals Database */

animal(mammal,tiger,carnivore,stripes).

animal(mammal,hyena,carnivore,ugly).

animal(mammal,lion,carnivore,mane).

animal(mammal,zebra,herbivore,stripes).

animal(bird,eagle,carnivore,large).

animal(bird,sparrow,scavenger,small).

animal(reptile,snake,carnivore,long).

animal(reptile,lizard,scavenger,small).

INPUT/OUTPUT

?- animal(A,B,C,D).

A = mammal,

B = tiger,

C = carnivore,

D = stripes

7 WPP FOR A FAMILY-TREE.

WRITE THE QUERIES TO RETRIVE THE REQUIRED INFORMATION.

male(harry).

female(liz).

parent(phil, chas).

parent(liz, chas).

parent(chas,harry).

parent(chas,wills).

grandmother(GM, C):- mother(GM, P), parent(P, C).

mother(M,C):- female(M), parent(M, C).

INPUT/OUTPUT

?- parent(X,Y).

X = phil,

Y = chas ;

X = liz,

Y = chas ;

X = chas,

Y = harry ;

X = chas,

Y = wills.

8 WPP sum the integers from 1 to N

```
/* sum the integers from 1 to N (the first argument)inclusive */
sumto(1,1).
sumto(N,S):-N>1,N1 is N-1,sumto(N1,S1),S is S1+N.
```

INPUT/OUTPUT

```
?- sumto(10,N).
N = 55
```

9 WPP to print Hello World

```
goal:-write('Hello World'),nl,write('Welcome to Prolog'),nl.
```

INPUT/OUTPUT

```
?- goal.
Hello World
Welcome to Prolog
true.
```

10 WPP FOR ANIMAL DB. IDENTIFY ITS SIZE AS SMALL,MEDIUM, BIG

```
big(bear).
big(elephant).
small(cat).
brown(bear).
black(cat).
gray(elephant).
```

dark(Z) :- black(Z).

dark(Z) :-brown(Z).

INPUT/OUTPUT

?- big(X).

X = bear ;

X = elephant.

11WPP FOR STUDENT-TEACHER-SUBJECT-COURSE-CODE.
INCORPORATE REQUIRED QUERIES

takes(jane_doe, his201).

takes(jane_doe, cs245).

takes(ajit_chandra, art302).

takes(ajit_chandra, cs254).

classmates(X, Y) :- takes(X, Z), takes(Y, Z).

INPUT/OUTPUT

?- takes(X,Y).

X = jane_doe,

Y = his201 ;

X = jane_doe,

Y = cs245 ;

X = ajit_chandra,

Y = art302 ;

X = ajit_chandra,

Y = cs254.

12 WPP FOR DFS.

```
edge(a,b).
edge(b, c).
edge(c, d).
edge(d,e).
edge(b, e).
edge(d, f).
path(X, X).
path(X, Y) :- edge(Z, Y), path(X, Z).
```

INPUT/OUTPUT

```
?- path(X,Y).
X = Y ;
X = a,
Y = b ;
X = b,
Y = c ;
X = a,
Y = c ;
X = c,
Y = d ;
X = b,
Y = d ;
X = a,
Y = d ;
X = d,
```

Y = e ;

X = c,

Y = e ;

X = b,

Y = e ;

X = a,

Y = e ;

X = b,

Y = e;

X = a,

Y = e ;

X = d,

Y = f ;

X = c,

Y = f ;

X = b,

Y = f ;

X = a,

Y = f ;

13 WPP TO FIND GCD OF 2 NOS.

gcd(X,X,X).

gcd(X,Y,Z) :- X<Y, Y1 is Y-X, gcd(X,Y1,Z).

gcd(X,Y,Z) :- X>Y, X1 is X-Y, gcd(X1,Y,Z).

INPUT/OUTPUT

?- gcd(6,4,A).

A = 2

?- gcd(5,3,A).

A = 1

?- gcd(5,5,A).

A = 5 .

14 WPP TO FIND FACTORIAL OF A GIVEN NO.

fact(0,1).

fact(X,F) :- X>0, X1 is X-1, fact(X1,F1), F is X*F1.

INPUT/OUTPUT

?- fact(5,N).

N = 120 .

?- fact(3,N).

N = 6

?- fact(0,N).

N = 1

?- fact(1,N).

N = 1 .

15 WPP FOR PLANETS DB.

orbits(mercury, sun).

orbits(venus, sun).

orbits(earth, sun).

orbits(mars, sun).

orbits(moon, earth).

orbits(phobos, mars).

orbits(deimos, mars).

planet(P) :- orbits(P,sun).

satellite(S) :- orbits(S,P), planet(P).

INPUT/OUTPUT

?- orbits(A,B).

A = mercury,

B = sun ;

A = venus,

B = sun ;

A = earth,

B = sun ;

A = mars,

B = sun ;

A = moon,

B = earth ;

A = phobos,

B = mars ;

A = deimos,

B = mars.

16 WPP FOR FORWARD CHAINING.

rainy(chennai).

rainy(coimbatore).

rainy(ooty).

cold(ooty).

snowy(X):-rainy(X),cold(X).

INPUT/OUTPUT

?- rainy(X).

X = chennai ;

X = coimbatore ;

X = ooty.

?- cold(X).

X = ooty.

17. WPP FOR FRUIT AND ITS COLOR USING back tracking

```
colour(cherry, red).
colour(banana, yellow).
colour(apple, red).
colour(apple, green).
colour(orange, orange).
colour(X, unknown).
```

INPUT/OUTPUT

?- color(X,Y).

Correct to: "colour(X,Y)"? yes

X = cherry,

Y = red ;

X = banana,

Y = yellow ;

X = apple,

Y = red ;

X = apple,

Y = green ;

X = Y, Y = orange ;

Y = unknown.

18. WPP TO IMPLEMENT A Cut !

```
max(X,Y,Y) :- Y>X, !.
max(X,Y,X).
```

INPUT/OUTPUT

```
?- max(6,5,N).
N = 6.
```

19. WPP TO IMPLEMENT pattern matching

```
president(X) :- first_name(X, georgedubya), second_name(X, bush).
prime_minister(X) :- first_name(X, maggie), second_name(X, thatcher).
prime_minister(X) :- first_name(X, tony), second_name(X, blair).
first_name(tonyblair, tony).
first_name(georgebush, georgedubya).
second_name(tonyblair, blair).
second_name(georgebush, bush).
```

INPUT/OUTPUT

```
?- prime_mininster(X).
Correct to: "prime_minister(X)"? yes
```

X = tonyblair.

20. WPP TO IMPLEMENT SUM OF 1 TO N

```
/* sum the integers from 1 to N (the first argument)
inclusive */
sumto(1,1).
sumto(N,S):-N>1,N1 is N-1,sumto(N1,S1),S is S1+N.
```

INPUT/OUTPUT

```
?- sumto(10,N).
N = 55 .
```

21 WPP TO IMPLEMENT TOWERS OF HANOI

```
% move(N,X,Y,Z) - move N disks from peg X to peg Y, with peg Z being
the
%                auxilliary peg
%
% Strategy:
% Base Case: One disc - To transfer a stack consisting of 1 disc from
%    peg X to peg Y, simply move that disc from X to Y
% Recursive Case: To transfer n discs from X to Y, do the following:
% Transfer the first n-1 discs to some other peg X
%        Move the last disc on X to Y
%         Transfer the n-1 discs from X to peg Y

    move(1,X,Y,_) :-
        write('Move top disk from '),
        write(X),
        write(' to '),
```

```
        write(Y),
nl.
    move(N,X,Y,Z) :-
        N>1,
        M is N-1,
        move(M,X,Z,Y),
        move(1,X,Y,_),
        move(M,Z,Y,X).
```

INPUT/OUTPUT

```
?- move(2,X,Y,Z).
Move top disk from _1184 to _1188
Move top disk from _1184 to _1186
Move top disk from _1188 to _1186
?- move(3,A,B,C).
Move top disk from _1184 to _1186
Move top disk from _1184 to _1188
Move top disk from _1186 to _1188
Move top disk from _1184 to _1186
Move top disk from _1188 to _1184
Move top disk from _1188 to _1186
Move top disk from _1184 to _1186
True
?- move(4,A,B,C).
Move top disk from _1184 to _1188
Move top disk from _1184 to _1186
Move top disk from _1188 to _1186
```

Move top disk from _1184 to _1188

Move top disk from _1186 to _1184

Move top disk from _1186 to _1188

Move top disk from _1184 to _1188

Move top disk from _1184 to _1186

Move top disk from _1188 to _1186

Move top disk from _1188 to _1184

Move top disk from _1186 to _1184

Move top disk from _1188 to _1186

Move top disk from _1184 to _1188

Move top disk from _1184 to _1186

Move top disk from _1188 to _1186

True

22 WPP TO IMPLEMENT DFS

```
% solve(goal, solution Path)
% s(state, successor-state)
dfs(N,[N]) :- goal(N).
dfs(N,[N|Sol1]):-  s(N,N1),  dfs(N1,Sol1).

s(a,b). s(a,c). s(b,d). s(b,e). s(c,f).
s(c,g). s(d,h). s(e,i). s(e,j). s(f,k).
goal(i). goal(f).
```

INPUT/OUTPUT

```
?- dfs(A,B).
A = i,
```

B = [i] ;

A = f,

B = [f] ;

A = a,

B = [a, b, e, i] ;

A = a,

B = [a, c, f] ;

A = b,

B = [b, e, i] ;

A = c,

B = [c, f] ;

A = e,

B = [e, i] ;

false.

23 WPP FOR STUDENT-TEACHER-SUB-CODE.

WRITE PROPER QUERIES TO RETRIEVE REQUIRED INFORMATION

instructor(perkowski,ee271).

instructor(perkowski,ee171).

instructor(perkowski,ee478).

enrolled(jeske,ee171).

enrolled(greenwood,ee171).

enrolled(alan-chen,ee171).

enrolled(alan-chen,ee271).

enrolled(chris-clark,ee271).

enrolled(edison-tsai,ee171).

enrolled(chris-clark,ee171).

instructor(bebis,cs365).

instructor(looney,cs311).

instructor(yuksel,cs446).

instructor(helfand,cs493).

instructor(quint,math486).

enrolled(ben,cs365).

enrolled(bill,cs365).

enrolled(bill,cs446).

enrolled(brian,cs311).

teaches(professor,Student):-
instructor,professor,Class),enrolled(Student,Class).

INPUT/OUTPUT

?- enrolled(X,Y).

X = jeske,

Y = ee171 ;

X = greenwood,

Y = ee171 ;

X = alan-chen,

Y = ee171 ;

X = alan-chen,

Y = ee271 ;

X = chris-clark,

Y = ee271 ;

X = edison-tsai,

Y = ee171 ;

X = chris-clark,

Y = ee171 ;

X = ben,

Y = cs365 ;

X = bill,

Y = cs365 ;

X = bill,

Y = cs446 ;

X = brian,

Y = cs311.

?- insructor(X,Y).

Correct to: "instructor(X,Y)"? yes

X = perkowski,

Y = ee271 ;

X = perkowski,

Y = ee171 ;

X = perkowski,

Y = ee478 ;

X = bebis,

Y = cs365 ;

X = looney,

Y = cs311 ;

X = yuksel,

Y = cs446 ;

X = helfand,

Y = cs493 ;

X = quint,

Y = math486.

24 WPP TO ESTABLISH FAMILY RELATION.

female(sarah).

female(rebekah).

female(hagar_concubine).

female(milcah).

female(bashemath).

female(mahalath).

female(first_daughter).

female(second_daughter).

female(terahs_first_wife).

female(terahs_second_wife).

female(harans_wife).

female(lots_first_wife).

female(ismaels_wife).

female(leah).

female(kemuels_wife).

female(rachel).

female(labans_wife).

male(terah).

male(abraham).

male(nahor).

male(haran).

male(isaac).

male(ismael).

male(uz).

male(kemuel).

male(bethuel).

male(lot).

male(iscah).

male(esau).

male(jacob).

male(massa).

male(hadad).

male(laban).

male(reuel).

male(levi3rd).

male(judah4th).

male(aliah).

male(elak).

male(moab).

male(ben-ammi).

father(terah, sarah).

father(terah, abraham).

father(terah, nahor).

father(terah, haran).

father(abraham, isaac).

father(abraham, ismael).

father(nahor, uz).

father(nahor, kemuel).

father(nahor, bethuel).

father(haran, milcah).

father(haran, lot).

father(haran, iscah).

father(isaac, esau).

father(isaac, jacob).

father(ismael, massa).

father(ismael, mahalath).

father(ismael, hadad).

father(ismael, bashemath).

father(esau, reuel).

father(jacob, levi3rd).

father(jacob, judah4th).

father(esau, aliah).

father(esau, elak).

father(kemuel, aram).

father(bethuel, laban).

father(bethuel, rebekah).

father(lot, first_daughter).

father(lot, second_daughter).

father(lot, moab).

father(lot, ben_ammi).

father(laban, rachel).

father(laban, leah).

mother(terahs_second_wife, sarah).

mother(terahs_first_wife, abraham).

mother(terahs_first_wife, nahor).

mother(terahs_first_wife, haran).

mother(sarah, isaac).

mother(hagar_concubine, ismael).

mother(milcah, uz).

mother(milcah, kemuel).

mother(milcah, bethuel).

mother(harans_wife, milcha).

mother(harans_wife, lot).

mother(harans_wife, iscah).

mother(rebekah, esau).

mother(rebekah, jacob).

mother(ismaels_wife, massa).

mother(ismaels_wife, mahalath).

mother(ismaels_wife, hadad).

mother(ismaels_wife, bashemath).

mother(bethuels_wife, laban).

mother(bethuels_wife, rebekah).

mother(lots_first_wife, first_daughter).

mother(lots_first_wife, second_daughter).

mother(first_daughter, moab).

mother(second_daughter, ben_ammi).

mother(bashemath, reuel).

mother(leah, levi3rd).

mother(leah, judas4th).

mother(mahalath, aliah).

mother(mahalath, elak).

mother(lebans_wife, rachel).

mother(lebans_wife, leah).

husband(terah, terahs_first_wife).

husband(terah, terahs_second_wife).

```
husband(abraham, sarah).
husband(abraham, hagar_concubine).
husband(nahor, milcah).
husband(haran, harans_wife).
husband(isaac, rebekah).
husband(ismael, ismaels_wife).
husband(kemuel, kemuels_wife).
husband(bethuel, bethuels_wife).
husband(lot, lots_first_wife).
husband(lot, first_daughter).
husband(lot, second_daughter).
husband(esau, bashemath).
husband(jacob, leah).
husband(jacob, rachel).
husband(esau, mahalath).
husband(laban, labans_wife).

wife(X, Y):- husband(Y, X).
married(X, Y):- wife(X, Y).
married(X, Y):- husband(X, Y).
parent(X, Y):- mother(X, Y).
parent(X, Y):- father(X, Y).

grandmother(X, Y):- mother(X, Z), parent(Z, Y).
grandfather(X, Y):- father(X, Z), parent(Z, Y).
grandparent(X, Y):- grandfather(X, Y).
grandparent(X, Y):- grandmother(X, Y).
```

```
anc(0,X,X).
anc(N,X,Y) :- N > 0, M is N-1, parent(X,Z), anc(M,Z,Y).

great_grandparent(X,Y)        :- anc(3,X,Y).

half_sibling(X,Y) :- parent(Z,X), parent(Z,Y), X \= Y.
sibling(X,Y) :- mother(Z,X), mother(Z,Y), father(W,X), father(W,Y), X
\=Y.

aunt_or_uncle(X,Y) :- sibling(X,Z), parent(Z,Y).
cousin(X,Y):- parent(Z,X), sibling(Z,W), parent(W,Y).
deepcousin(X,Y):- sibling(X,Y).    % siblings are 0th cousins
deepcousin(X,Y):- parent(Z,X), deepcousin(Z,W), parent(W,Y).

deepcousin(X,Y):- sibling(X,Y).    % siblings are 0th cousins
deepcousin(X,Y):- parent(Z,X), deepcousin(Z,W), parent(W,Y).
```

25 WPP TO ESTABLISH FAMILY RELATION

```
family
% Figure 1.8   The family program.

parent( pam, bob).
parent( tom, bob).
parent( tom, liz).
parent( bob, ann).
```

```
parent( bob, pat).
parent( pat, jim).

female( pam).
female( liz).
female( ann).
female( pat).
male( tom).
male( bob).
male( jim).

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

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

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

```
predecessor( X, Z)  :-   % Rule pr1
  parent( X, Z).

predecessor( X, Z)  :-   % Rule pr2
  parent( X, Y),
  predecessor( Y, Z).
```

INPUT/OUTPUT

```
?- parent(X,Y).
X = pam,
Y = bob ;
X = tom,
Y = bob ;
X = tom,
Y = liz ;
X = bob,
Y = ann ;
X = bob,
Y = pat ;
X = pat,
Y = jim.
```

26 WPP TO IMPLEMENT PERSON-LIK-TOYS

```
Likes(ann,X) :- toy(X), plays(ann,X).
toy(doll).
toy(train).
plays(ann,train).
```

likes(john,Y) :- likes(ann,Y).

INPUT/OUTPUT

?- plays(X,Y).

X = ann,

Y = train.

27 WPP TO IMPLEMET ITEM-ITS LOCATION

location(desk, office).

location(apple, kitchen).

location(flashlight, desk).

location('washing machine', cellar).

location(nani, 'washing machine').

location(broccoli, kitchen).

location(crackers, kitchen).

location(computer, office).

INPUT/OUTPUT

?- loaction(apple,X).

Correct to: "location(apple,X)"? yes

X = kitchen.

28 WPP TO IMPLEMET CAUSES-DISEASE

/*Simple disease*/

domains

disease,indication,name = symbol

```
predicates
    hypothesis(name,disease)
    symptom(name,indication)

clauses
    symptom(amit,fever).
    symptom(amit,rash).
    symptom(amit,headache).
    symptom(amit,runn_nose).

    symptom(kaushal,chills).
    symptom(kaushal,fever).
    symptom(kaushal,hedache).

    symptom(dipen,runny_nose).
    symptom(dipen,rash).
    symptom(dipen,flu).


    hypothesis(Patient,measels):-
        symptom(Patient,fever),
        symptom(Patient,cough),
        symptom(Patient,conjunctivitis),
        symptom(Patient,rash).

    hypothesis(Patient,german_measles) :-
        symptom(Patient,fever),
```

```
    symptom(Patient,headache),
    symptom(Patient,runny_nose),
    symptom(Patient,rash).

hypothesis(Patient,flu) :-
    symptom(Patient,fever),
    symptom(Patient,headache),
    symptom(Patient,body_ache),
    symptom(Patient,chills).

hypothesis(Patient,common_cold) :-
    symptom(Patient,headache),
    symptom(Patient,sneezing),
    symptom(Patient,sore_throat),
    symptom(Patient,chills),
```

INPUT/OUTPUT

```
?- hypothesis(amit,X).
false.

?- symptom(X,Y).
X = amit,
Y = rash ;
X = amit,
Y = headache ;
X = amit,
Y = runn_nose ;
```

X = kaushal,

Y = chills ;

X = kaushal,

Y = fever ;

X = kaushal,

Y = hedache ;

X = dipen,

Y = runny_nose ;

X = dipen,

Y = rash ;

X = dipen,

Y = flu.

29 WPP TO FIND THAT THE GIVEN NO IS EVEN/ODD

checkeven(N):-M is N//2,N=:=2*M.

INPUT/OUTPUT

?- checkeven(100).

true.

?- checkeven(99).

false.

30 WPP TO PRINT MAN IS MORTAL/NOT

mortal(X) :- man(X).

man(socrates).

INPUT/OUTPUT

```
?- man(X).
X = socrates.
```

31. WPP TO IMPLEMENT INFERENCE ENGINE

```
% File INTERP.PL
% Meta-interpreter for Prolog
% interpret(+Goal)
% Executes Goal.
interpret(true) :- !.
interpret((GoalA,GoalB)) :- !,
interpret(GoalA),
interpret(GoalB).
interpret(Goal) :- clause(Goal,Body),
interpret(Body).
% Test knowledge base (note the dynamic declarations!)
:- dynamic(parent/2).
parent(michael,cathy).
parent(melody,cathy).
parent(charles_gordon,michael).
parent(hazel,michael).
parent(jim,melody).
parent(eleanor,melody).
:- dynamic(grandparent/2).
grandparent(X,Y) :- parent(Z,Y), parent(X,Z).
test :- interpret(grandparent(A,B)), write([A,B]), nl, fail.
% prints out all solutions
```

INPUT/OUTPUT

?- parents(X,Y).

Correct to: "parent(X,Y)"? yes

X = michael,

Y = cathy ;

X = melody,

Y = cathy ;

X = charles_gordon,

Y = michael ;

X = hazel,

Y = michael ;

X = jim,

Y = melody ;

X = eleanor,

Y = melody.

32 WPP TO PRINT PARTICULAR BIRD CAN FLY OR NOT

bird(sparrow).

bird(eagle).

bird(duck).

bird(crow).

~bird(ostrich).

bird(puffin).

bird(swan).

bird(albatross).

bird(starling).

bird(owl).

bird(kingfisher).

bird(thrush).

can_fly(X):-bird(X).

can_fly(ostrich):-fail.

INPUT/OUTPUT

?- bird(X).

X = sparrow ;

X = eagle ;

X = duck ;

X = crow ;

X = puffin ;

X = swan ;

X = albatross ;

X = starling ;

X = owl ;

X = kingfisher ;

X = thrush.

33 WPP TO COUNT NO OF VOWELS

go(Vowels):-count(0,Vowels).

count(Oldvowels,Totvowels):-

get0(X),process(X,Oldvowels,Totvowels).

process(42,Oldvowels,Oldvowels).

process(X,Oldvowels,Totalvowels):-

X=\=42,processChar(X,Oldvowels,New),

```
count(New,Totalvowels).
processChar(X,Oldvowels,New):-vowel(X),
New is Oldvowels+1.
processChar(X,Oldvowels,Oldvowels).
vowel(65). /* A */
vowel(69). /* E */
vowel(73). /* I */
vowel(79). /* O */
vowel(85). /* U */
vowel(97). /* a */
vowel(101). /* e */
vowel(105). /* i */
vowel(111). /* o */
vowel(117). /* u */
```

INPUT/OUTPUT

```
?- vowels(X).
Correct to: "vowel(X)"? yes
X = 65 ;
X = 69 ;
X = 73 ;
X = 79 ;
X = 85 ;
X = 97 ;
X = 101 ;
X = 105 ;
X = 111 ;
```

X = 117.