Practical No 1

a. Implement depth first search algorithm.

Code:

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

Output:

```
['A', 'C', 'F', 'E', 'B', 'D']
```

b. Implement breadth first search algorithm.

Code:

```
graph = {'A': set(['B', 'C']),
         'B': set(['A', 'D', 'E']),
         'C': set(['A', 'F']),
         'D': set(['B']),
         'E': set(['B', 'F']),
         'F': set(['C', 'E']) }
def bfs(start):
    queue = [start]
    levels={}
    levels[start]=0
    visited = set(start)
    while queue:
        node = queue.pop(0)
        neighbours=graph[node]
        for neighbor in neighbours:
            if neighbor not in visited:
                queue.append(neighbor)
                visited.add(neighbor)
                levels[neighbor]= levels[node]+1
    print(levels)
    return visited
```

```
print(str(bfs('A')))
def bfs_paths(graph, start, goal):
    queue = [(start, [start])]
    while queue:
        (vertex, path) = queue.pop(0)
        for next in graph[vertex] - set(path):
            if next == goal:
                yield path + [next]
            else:
                queue.append((next, path + [next]))
result=list(bfs_paths(graph, 'A', 'F'))
print(result)
def shortest_path(graph, start, goal):
    try:
        return next(bfs_paths(graph, start, goal))
    except StopIteration:
        return None
result1=shortest_path(graph, 'A', 'F')
print(result1)
```

Output:

```
=========
{'A': 0, 'C': 1, 'B': 1, 'F': 2, 'E': 2, 'D': 2}
{'A', 'E', 'C', 'F', 'B', 'D'}
[['A', 'C', 'F'], ['A', 'B', 'E', 'F']]
['A', 'C', 'F']
```

PRACTICAL – 2

A. Write a program to simulate 4-Queen / N-Queen problem. B. Write a program to solve tower of Hanoi problem.

A. Write a program to simulate 4-Queen / N-Queen problem. Code:

```
class QueenChessBoard:
    def __init__(self,
size):
        self.size =
size
        self.columns =
[]
```

```
    def
place_in_next_row(self,
column):

self.columns.append(col
umn)
    def
remove_in_current_row(s
elf):
        return
self.columns.pop()
    def
is_this_column_safe_in_
next_row(self, column):
        row =
len(self.columns)
        for
queen_column in
self.columns:
            if column
== queen_column:
                return
False
        for queen_row,
queen_column in
enumerate(self.columns)
:
            if
queen_column -
queen_row == column -
row:
                return
False
        for queen_row,
queen_column in
enumerate(self.columns)
:
            if
((self.size -
queen_column) -
queen_row
                ==
(self.size - column) -
row):
                return
False
        return True
```

```
    def display(self):
        for row in
range(self.size):
            for column
in range(self.size):
                if
column ==
self.columns[row]:

print('Q', end='  ')
                else:

print(' . ', end='  ')
            print()
def solve_queen(size):
    board =
QueenChessBoard(size)
    number_of_solutions
= 0
    row = 0
    column = 0
    while True:
        while column <
size:
            if
board.is_this_column_sa
fe_in_next_row(column):

board.place_in_next_row
(column)
                row +=
1
                column
= 0
                break
            else:
                column
+= 1
        if (column ==
size or row == size):
            if row ==
size:

board.display()
                print()
```

```
number_of_solutions +=
1

board.remove_in_current
_row()
                row -=
1
            try:

prev_column =
board.remove_in_current
_row()
            except
IndexError:
                break
            row -= 1
            column = 1
+ prev_column
    print('Number of
solutions:',
number_of_solutions)
n = int(input('Enter n:
'))
solve_queen(n)
```

output

```
Enter n: 4
  .   Q   .   .
  .   .   .   Q
Q   .   .   .
  .   .   Q   .

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

Number of solutions: 2
>
```

B. Write a program to solve tower of Hanoi problem.

```
def
moveTower(hieght,fromPo
le,toPole,withPole):
    if hieght>=1:

moveTower(hieght-
1,fromPole,withPole,toP
ole)

moveDisk(fromPole,toPol
e)

moveTower(hieght-
1,withPole,toPole,fromP
ole)
def moveDisk(fp,tp):
    print("moving disk
from",fp,"to",tp)
moveTower(3,'A','B','c'
)
```

output:

```
moving disk from A to B
moving disk from A to C
moving disk from B to C
moving disk from A to B
moving disk from C to A
moving disk from C to B
moving disk from A to B
>>>
```

PRACTICAL NO.-3

A. Write a program to implement alpha beta search.

```
tree=[[[5,1,2],[8,-8,-9]],[[9,4,5],[-3,4,3]]]
root=0
pruned=0
def children(branch,depth,alpha,beta):
```

```
    global tree
    global root
    global pruned
    i=0
    for child in branch:
        if type(child) is list:
            (nalpha,nbeta)=children(child,depth+1,alpha,beta)
            if depth%2==1:
                beta=nalpha if nalpha<beta else beta
            else:
                alpha=nbeta if nbeta>alpha else alpha
            branch[i]=alpha if depth%2==0 else beta
            i+=1
        else:
            if depth %2==0 and alpha<child:
                alpha= child
            if depth%2==1 and beta>child:
                beta=child
            if alpha>=beta:
                pruned+=1
                break
    if depth==root:
        tree =alpha if root ==0 else beta
    return (alpha,beta)
def alphabeta(in_tree=tree, start=root,upper=-15,lower=15):
    global tree
    global pruned
    global root
    (alpha,beta)=children(tree,start,upper,lower)
    print("(alpha,beta):",alpha,beta)
    print("Result:",tree)
    print("Times pruned:",pruned)
alphabeta(None)
```

Output:

```
= RESTART: C:/Users/Riddhi Shinde/AppData/Local/Programs/Python/Python312/beta.py
(alpha, beta):  5 15
Result:  5
Times pruned:  1
```

B. Write a program for Hill climbing problem.

```
import math
```

```
increment = 0.1

startingPoint = [1, 1]

point1 = [1,5]
point2 = [6,4]
point3 = [5,2]
point4 = [2,1]

def distance(x1, y1, x2, y2):
    dist = math.pow(x2 - x1, 2) + math.pow(y2 - y1, 2)
    return dist

def sumOfDistances(x1, y1, px1, py1, px2, py2, px3, py3, px4, py4):
    d1 = distance(x1, y1, px1, py1)
    d2 = distance(x1, y1, px2, py2)
    d3 = distance(x1, y1, px3, py3)
    d4 = distance(x1, y1, px4, py4)
    return d1 + d2 + d3 + d4

def newDistance(x1, y1, point1, point2, point3, point4):
    d1 = [x1, y1]
    d1temp = sumOfDistances(x1, y1, point1[0], point1[1],
                                    point2[0], point2[1],
                                    point3[0], point3[1],
                                    point4[0], point4[1])
    d1.append(d1temp)
    return d1

minDistance = sumOfDistances(startingPoint[0], startingPoint[1],
                             point1[0], point1[1],
                             point2[0], point2[1],
                             point3[0], point3[1],
                             point4[0], point4[1])

flag = True

def newPoints(minimum, d1, d2, d3, d4):
    if d1[2] == minimum:
        return [d1[0], d1[1]]
    elif d2[2] == minimum:
        return [d2[0], d2[1]]
    elif d3[2] == minimum:
        return [d3[0], d3[1]]
    elif d4[2] == minimum:
        return [d4[0], d4[1]]
```

```
i = 1
while flag:
    d1 = newDistance(startingPoint[0] + increment, startingPoint[1],
point1, point2, point3, point4)
    d2 = newDistance(startingPoint[0] - increment, startingPoint[1],
point1, point2, point3, point4)
    d3 = newDistance(startingPoint[0], startingPoint[1] + increment,
point1, point2, point3, point4)
    d4 = newDistance(startingPoint[0], startingPoint[1] - increment,
point1, point2, point3, point4)

    print(i, ' ', round(startingPoint[0], 2), round(startingPoint[1],
2))

    minimum = min(d1[2], d2[2], d3[2], d4[2])

    if minimum < minDistance:
        startingPoint = newPoints(minimum, d1, d2, d3, d4)
        minDistance = minimum
        i += 1
    else:
        flag = False
```

Output:

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

```
38    3.1 2.6
39    3.2 2.6
40    3.2 2.7
41    3.2 2.8
42    3.3 2.8
43    3.4 2.8
44    3.4 2.9
45    3.5 2.9
46    3.5 3.0
```

Practical no-4

A. Write a program to implement A* algorithm.

B. Solve Water Jug Problem

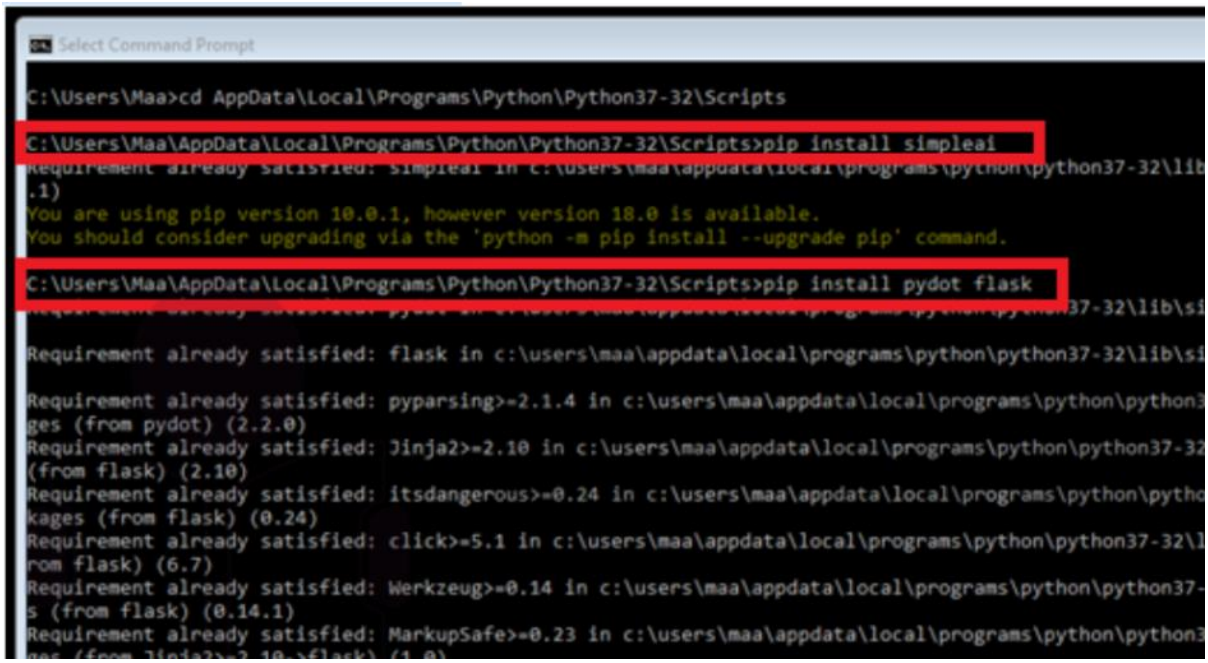
A.Write a program to implement A* algorithm.

Note: Install 2 package in python scripts directory using pip command.

1. pip install simpleai

2. pip install pydot flask

According to the version there might be needing some changes in the commands




```python
from simpleai.search import SearchProblem, astar

GOAL = 'HELLO WORLD'
class HelloProblem(SearchProblem):
    def actions(self, state):
        if len(state) < len(GOAL):
            return list(set(GOAL))  # Reduces action space for
efficiency
        else:
            return []
    def result(self, state, action):
        return state + action
    def is_goal(self, state):
        return state == GOAL
    def heuristic(self, state):
        wrong = sum([1 if state[i] != GOAL[i] else 0
                     for i in range(len(state))])
        missing = len(GOAL) - len(state)
        return wrong + missing
problem = HelloProblem(initial_state='')
result = astar(problem)
print(result.state)
print(result.path())
```

output:

```
t.py
HELLO WORLD
[(None, ''), ('H', 'H'), ('E', 'HE'), ('L', 'HEL'), ('L', 'HELL'), ('O', 'HELLO'
), (' ', 'HELLO '), ('W', 'HELLO W'), ('O', 'HELLO WO'), ('R', 'HELLO WOR'), ('L
', 'HELLO WORL'), ('D', 'HELLO WORLD')]
```

B. Solve Water Jug Problem.

Code:

```
capacity = (12, 8, 5)
x = capacity[0]
y = capacity[1]
z = capacity[2]

memory = {}
ans = []

def get_all_states(state):
    a = state[0]  # Jug A
    b = state[1]  # Jug B
    c = state[2]  # Jug C



    if a == 6 and b == 6:
        ans.append(state)
        return True



    if (a, b, c) in memory:
        return False
    memory[(a, b, c)] = 1

    if a > 0:

        if a + b <= y:
            if get_all_states((0, a + b, c)):
                ans.append(state)
                return True
        else:
            if get_all_states((a - (y - b), y, c)):
                ans.append(state)
                return True

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

```
                ans.append(state)
                return True

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

        if b + c <= z:
            if get_all_states((a, 0, b + c)):
                ans.append(state)
                return True
        else:
            if get_all_states((a, b - (z - c), z)):
                ans.append(state)
                return True

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

        if b + c <= y:
            if get_all_states((a, b + c, 0)):
                ans.append(state)
                return True
        else:
            if get_all_states((a, y, c - (y - b))):
                ans.append(state)
                return True

    return False

initial_state = (12, 0, 0)
print("Starting work...\n")

get_all_states(initial_state)
```

```
ans.reverse()

for i in ans:
    print(i)
```

output:

```
Starting work...

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

Practical No 5

A. Simulate tic-tac-toe game using min-max algorithm.

```
import os
import time

board = [' ' for _ in range(10)]
player = 1

Win = 1
Draw = -1
Running = 0

Game = Running

def DrawBoard():
    print(f"{board[1]} | {board[2]} | {board[3]}")
    print("--+---+--")
    print(f"{board[4]} | {board[5]} | {board[6]}")
    print("--+---+--")
    print(f"{board[7]} | {board[8]} | {board[9]}")
    print()
```

```
def CheckPosition(x):
    return board[x] == ' '

def CheckWin():
    global Game
    win_conditions = [
        (1, 2, 3), (4, 5, 6), (7, 8, 9),
        (1, 4, 7), (2, 5, 8), (3, 6, 9),
        (1, 5, 9), (3, 5, 7)
    ]
    for a, b, c in win_conditions:
        if board[a] == board[b] == board[c] and board[a] != ' ':
            Game = Win
            return

    if all(cell != ' ' for cell in board[1:]):
        Game = Draw
    else:
        Game = Running



while Game == Running:
    os.system('cls' if os.name == 'nt' else 'clear')
    print("Tic-Tac-Toe Game")
    print("Player 1 [X] --- Player 2 [O]\n")
    DrawBoard()

    if player % 2 != 0:
        print("Player 1's turn")
        mark = 'X'
    else:
        print("Player 2's turn")
        mark = 'O'

    try:
        choice = int(input("Enter the position (1-9) to mark: "))
        if 1 <= choice <= 9 and CheckPosition(choice):
            board[choice] = mark
            player += 1
            CheckWin()
        else:
            print("Invalid move. Try again.")
            time.sleep(1)
    except ValueError:
        print("Please enter a number between 1 and 9.")
        time.sleep(1)
```

```
os.system('cls' if os.name == 'nt' else 'clear')
DrawBoard()
if Game == Draw:
    print("Game Draw!")
elif Game == Win:
    winner = "Player 1" if player % 2 == 0 else "Player 2"
    print(f"{winner} wins!")
```

Output:

```
Tic-Tac-Toe Game
Player 1 [X] --- Player 2 [O]



Please Wait...
     |     |
_____|_____|_____
     |     |
_____|_____|_____
     |     |
     |     |
Player 1's chance
Enter the position between [1-9] where you want to mark : 4
     |     |
_____|_____|_____
  X  |     |
_____|_____|_____
     |     |
     |     |
Player 2's chance
Enter the position between [1-9] where you want to mark : 1
  O  |     |
_____|_____|_____
```

B. Write a program on Shuffle deck of cards

Code:

```
import random
cardFaces=[]
Suits=['Hearts','Diamond','Clubs','Spades']
Royals=['J','K','Q','A']
deck=[]
for i in range(2,11):
    cardFaces.append(str(i))
for j in range (4):
    cardFaces.append(Royals[j])
for k in range (4):
    for l in range(13):
        card= (cardFaces[l]+" of "+ Suits[k])
        deck.append(card)

random.shuffle(deck)
for m in range(52):
    print(deck[m])
```

Output:

```
7 Of Hearts
4 Of Spades
Q Of Clubs
3 Of Spades
2 Of Hearts
6 Of Diamond
J Of Spades
5 Of Clubs
J Of Clubs
Q Of Diamond
10 Of Clubs
4 Of Clubs
4 Of Hearts
5 Of Spades
5 Of Hearts
K Of Clubs
K Of Hearts
10 Of Diamond
K Of Spades
7 Of Clubs
7 Of Diamond
J Of Diamond
10 Of Hearts
A Of Hearts
K Of Diamond
Q Of Hearts
A Of Diamond
6 Of Spades
2 Of Clubs
8 Of Hearts
10 Of Spades
9 Of Clubs
2 Of Spades
8 Of Clubs
4 Of Diamond
6 Of Hearts
8 Of Spades
3 Of Diamond
```

Practical No 6

A. Design an application to simulate number puzzle problem.

```
from __future__ import print_function
from simpleai.search import astar, SearchProblem
from simpleai.search.viewers import WebViewer

GOAL = '''1-2-3
4-5-6
7-8-e'''

INITIAL = '''4-1-2
7-e-3
8-5-6'''

def list_to_string(list_):
    return '\n'.join(['-'.join(row) for row in list_])

def string_to_list(string_):
    return [row.split('-') for row in string_.split('\n')]

def find_location(rows, element_to_find):
    '''Find the location of a piece in the puzzle.
       Returns a tuple: (row, column)'''
    for ir, row in enumerate(rows):
        for ic, element in enumerate(row):
            if element == element_to_find:
                return ir, ic

goal_positions = {}
rows_goal = string_to_list(GOAL)
for number in '12345678e':
    goal_positions[number] = find_location(rows_goal, number)

class EigthPuzzleProblem(SearchProblem):

    def actions(self, state):
        '''Returns a list of the pieces we can move to the empty
space.'''
        rows = string_to_list(state)
        row_e, col_e = find_location(rows, 'e')
        actions = []
```

```
        if row_e > 0:
            actions.append(rows[row_e - 1][col_e])
        if row_e < 2:
            actions.append(rows[row_e + 1][col_e])
        if col_e > 0:
            actions.append(rows[row_e][col_e - 1])
        if col_e < 2:
            actions.append(rows[row_e][col_e + 1])

        return actions

    def result(self, state, action):
        '''Return the resulting state after moving a piece to the empty
space.'''
        rows = string_to_list(state)
        row_e, col_e = find_location(rows, 'e')
        row_n, col_n = find_location(rows, action)
        rows[row_e][col_e], rows[row_n][col_n] = rows[row_n][col_n],
rows[row_e][col_e]
        return list_to_string(rows)

    def is_goal(self, state):
        '''Returns true if a state is the goal state.'''
        return state == GOAL

    def cost(self, state1, action, state2):
        '''Returns the cost of performing an action.'''
        return 1

    def heuristic(self, state):
        '''Estimate distance to goal using Manhattan distance.'''
        rows = string_to_list(state)
        distance = 0
        for number in '12345678e':
            row_n, col_n = find_location(rows, number)
            row_goal, col_goal = goal_positions[number]
            distance += abs(row_n - row_goal) + abs(col_n - col_goal)
        return distance

result = astar(EigthPuzzleProblem(INITIAL))

for action, state in result.path():
    print('Move tile:', action)
    print(state)
```

Output:

```
Move number None
4-1-2
7-e-3
8-5-6
Move number 5
4-1-2
7-5-3
8-e-6
Move number 8
4-1-2
7-5-3
e-8-6
Move number 7
4-1-2
e-5-3
7-8-6
Move number 4
e-1-2
4-5-3
7-8-6
Move number 1
1-e-2
4-5-3
7-8-6
Move number 2
1-2-e
4-5-3
7-8-6
Move number 3
1-2-3
4-5-e
7-8-6
Move number 6
1-2-3
4-5-6
7-8-e
>
```

A. Write a program to solve constraint satisfaction problem.

Code:

```
from __future__ import print_function
from simpleai.search import CspProblem, backtrack, min_conflicts,
MOST_CONSTRAINED_VARIABLE, HIGHEST_DEGREE_VARIABLE,
LEAST_CONSTRAINING_VALUE

variables = ('WA', 'NT', 'SA', 'Q', 'NSW', 'V', 'T')

domains = dict((v, ['red', 'green', 'blue']) for v in variables)



def const_different(variables, values):
    return values[0] != values[1]

constraints = [
    (('WA', 'NT'), const_different),
    (('WA', 'SA'), const_different),
    (('SA', 'NT'), const_different),
    (('SA', 'Q'), const_different),
    (('NT', 'Q'), const_different),
    (('SA', 'NSW'), const_different),
    (('Q', 'NSW'), const_different),
    (('SA', 'V'), const_different),
    (('NSW', 'V'), const_different),
]



my_problem = CspProblem(variables, domains, constraints)

print(backtrack(my_problem))
print(backtrack(my_problem,
variable_heuristic=MOST_CONSTRAINED_VARIABLE))
print(backtrack(my_problem,
variable_heuristic=HIGHEST_DEGREE_VARIABLE))
print(backtrack(my_problem, value_heuristic=LEAST_CONSTRAINING_VALUE))
print(backtrack(my_problem,
variable_heuristic=MOST_CONSTRAINED_VARIABLE,
value_heuristic=LEAST_CONSTRAINING_VALUE))
print(backtrack(my_problem, variable_heuristic=HIGHEST_DEGREE_VARIABLE,
value_heuristic=LEAST_CONSTRAINING_VALUE))
print(min_conflicts(my_problem))
```

Output:

```
{'WA': 'red', 'NT': 'green', 'SA': 'blue', 'Q': 'red', 'NSW': 'green', 'V': 'red', 'T': 'red'}
{'WA': 'red', 'NT': 'green', 'SA': 'blue', 'Q': 'red', 'NSW': 'green', 'V': 'red', 'T': 'red'}
{'SA': 'red', 'NT': 'green', 'Q': 'blue', 'NSW': 'green', 'WA': 'blue', 'V': 'blue', 'T': 'red'}
{'WA': 'red', 'NT': 'green', 'SA': 'blue', 'Q': 'red', 'NSW': 'green', 'V': 'red', 'T': 'red'}
{'WA': 'red', 'NT': 'green', 'SA': 'blue', 'Q': 'red', 'NSW': 'green', 'V': 'red', 'T': 'red'}
{'SA': 'red', 'NT': 'green', 'Q': 'blue', 'NSW': 'green', 'WA': 'blue', 'V': 'blue', 'T': 'red'}
{'WA': 'blue', 'NT': 'red', 'SA': 'green', 'Q': 'blue', 'NSW': 'red', 'V': 'blue', 'T': 'green'}
```

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

```
um = 3
uc = 3
rm = 0
rc = 0
flag = 0
select = 0

def display(p1, p2):
    for i in range(rm):
        print('M', end=' ')
    for i in range(rc):
        print('C', end=' ')

    print('|', end='')

    if flag == 0:
        print("------water------\\%c , %c/--" % (p1, p2), end=' ')
    else:
        print("--\\%c , %c/------water------" % (p1, p2), end=' ')
    print("|", end='')
    for i in range(um):
        print('M', end=' ')
    for i in range(uc):
        print('C', end=' ')
```

```
   print('')

def isreached():
   if rm == 3 and rc == 3:
      return 0
   return 1

def solve():
   global um, uc, rm, rc, flag, select

   while isreached():
      if flag == 0:

         if select == 1:
            display('C', ' ')
            uc += 1

         if select == 2:
            display('C', 'M')
            um += 1
            uc += 1

         if ((um - 2) >= uc and (rm + 2) >= rc) or (um - 2) == 0:
            um -= 2
            select = 1
            display('M', 'M')
            flag = 1

         elif ((uc - 2) < um and (rm == 0 or (rc + 2) <= rm)) or um ==
0:
            uc -= 2
            select = 2
            display('C', 'C')
            flag = 1

         elif (uc - 1 <= um - 1) and (rm + 1 >= rc + 1):
            um -= 1
            uc -= 1
            select = 3
            display('M', 'C')
            flag = 1

      else:

         if select == 1:
            display('M', 'M')
            rm += 2
```

```
        if select == 2:
            display('C', 'C')
            rc += 2

        if select == 3:
            display('M', 'C')
            rc += 1
            rm += 1

        if isreached():
            if (rc > 1 and rm == 0) or um == 0:
                rc -= 1
                select = 1
                display('C', ' ')
                flag = 0
            elif (uc + 2) > um:
                rc -= 1
                rm -= 1
                select = 2
                display('C', 'M')
                flag = 0

def main():
    print("Missionaries And Cannibal Problem")
    display(' ', ' ')
    solve()
    display(' ', ' ')

main()
```

Output:

```
========== RESTART: C:/Users
Missionaries And Cannibal Problem
|------water------\  ,  /-- |M M M C C C
|------water------\C , C/-- |M M M C
|--\C , C/------water------ |M M M C
C |--\C ,  /------water------ |M M M C
C |------water------\C ,  /-- |M M M C
C |------water------\C , C/-- |M M M
C |--\C , C/------water------ |M M M
C C |--\C ,  /------water------ |M M M
C C |------water------\C ,  /-- |M M M
C C |------water------\M , M/-- |M C
C C |--\M , M/------water------ |M C
M C |--\C , M/------water------ |M C
M C |------water------\C , M/-- |M C
M C |------water------\M , M/-- |C C
M C |--\M , M/------water------ |C C
M M M |--\C ,  /------water------ |C C
M M M |------water------\C ,  /-- |C C
M M M |------water------\C , C/-- |C
M M M |--\C , C/------water------ |C
M M M C |--\C ,  /------water------ |C
M M M C |------water------\C ,  /-- |C
M M M C |------water------\C , C/-- |
M M M C |--\C , C/------water------ |
M M M C C C |--\  ,  /------water------ |
>
```

Practical No 8

A. Derive the expression based on Associative Law

Code:

```
A = int(input("Enter first number (A): "))
B = int(input("Enter second number (B): "))
C = int(input("Enter third number (C): "))

left_add = (A + B) + C
right_add = A + (B + C)

print("\nAssociative Law of Addition:")
print(f"(A + B) + C = {left_add}")
print(f"A + (B + C) = {right_add}")
print("Law holds:", left_add == right_add)
```

```
left_mul = (A * B) * C
right_mul = A * (B * C)

print("\nAssociative Law of Multiplication:")
print(f"(A * B) * C = {left_mul}")
print(f"A * (B * C) = {right_mul}")
print("Law holds:", left_mul == right_mul)
```

Output:

```
==================================== F
Enter first number (A): 4
Enter second number (B): 2
Enter third number (C): 4

Associative Law of Addition:
(A + B) + C = 10
A + (B + C) = 10
Law holds: True

Associative Law of Multiplication:
(A * B) * C = 32
A * (B * C) = 32
Law holds: True
```

B. Derive the expression based on Distributive Law

Code:

```
A = int(input("Enter first number (A): "))
B = int(input("Enter second number (B): "))
C = int(input("Enter third number (C): "))

left_side = A * (B + C)
right_side = (A * B) + (A * C)

print("\nDistributive Law:")
print(f"A * (B + C) = {left_side}")
print(f"(A * B) + (A * C) = {right_side}")
print("Law holds:", left_side == right_side)
```

Output:

```
Enter first number (A): 2
Enter second number (B): 2
Enter third number (C): 3

Distributive Law:
A * (B + C) = 10
(A * B) + (A * C) = 10
Law holds: True
```

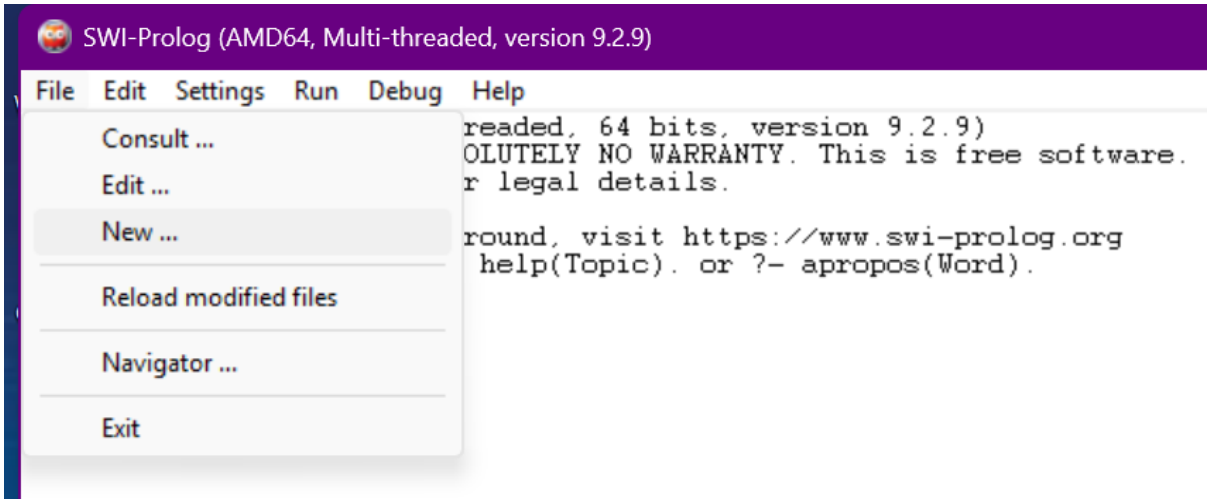
Practical No 9

Derive the predicate. (for e.g.: Sachin is batsman, batsman is cricketer -> Sachin is Cricketer)

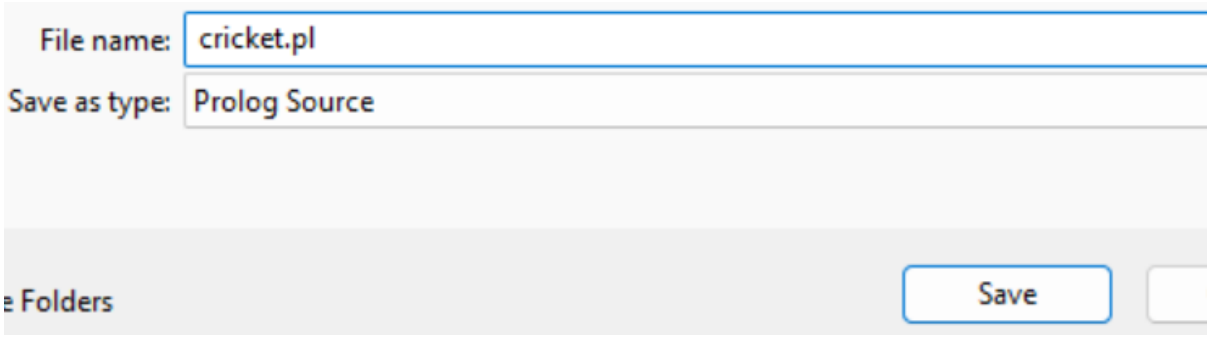
Save as .pl extension

Do this practical in SWI-Prolog

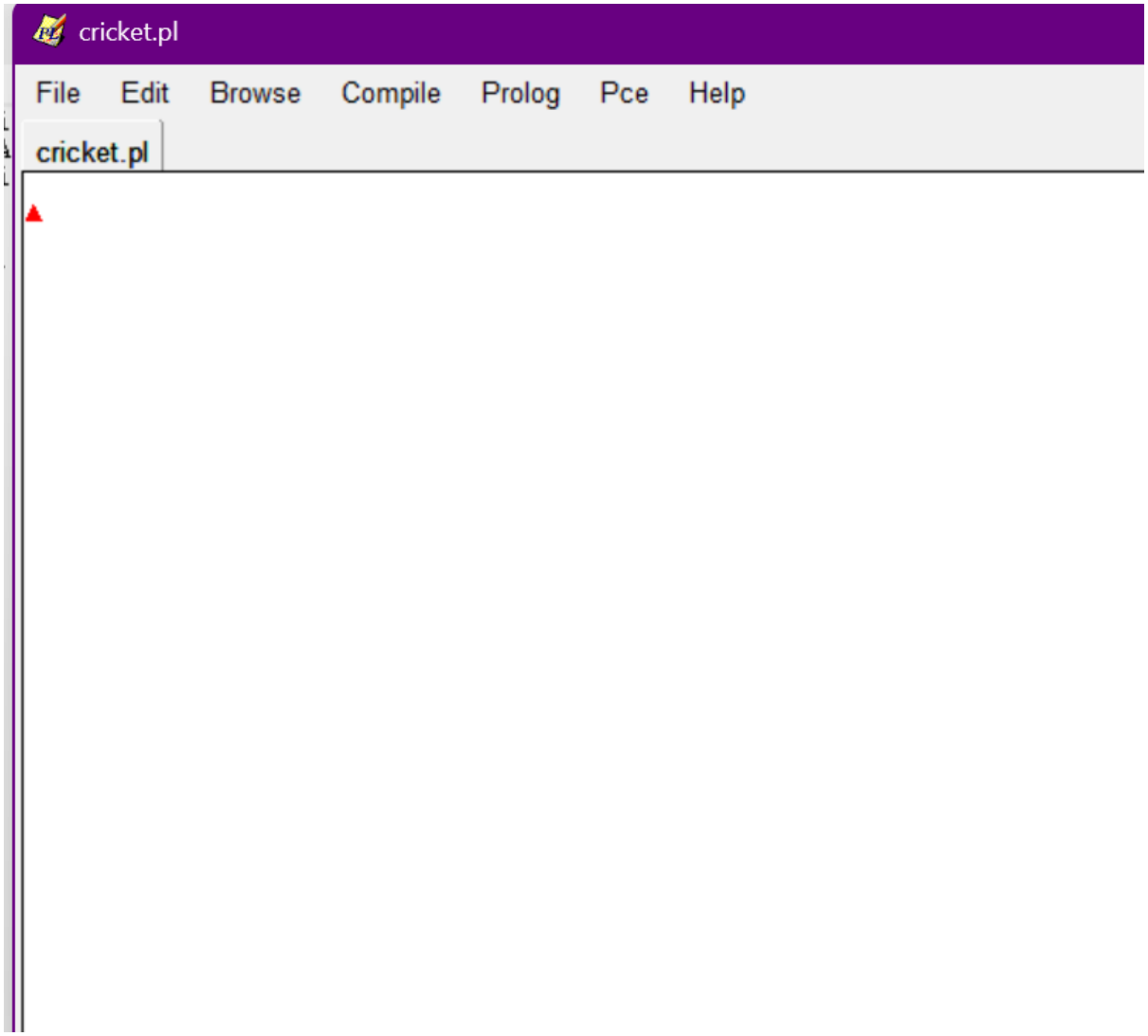
Open the software click on new



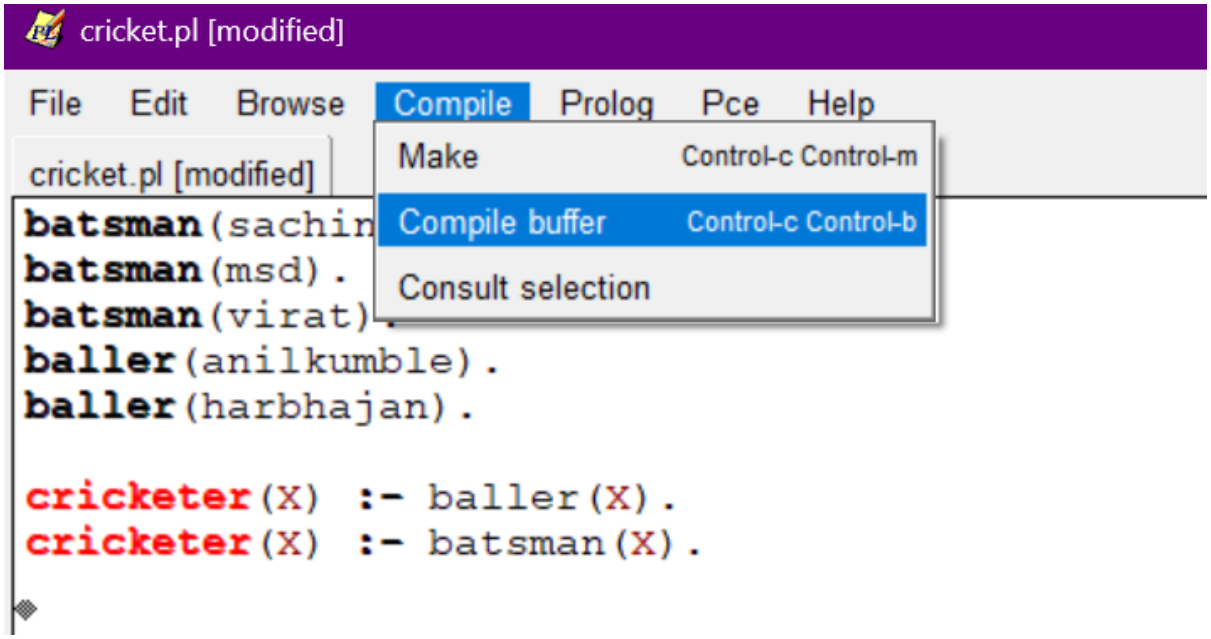
Create a file with .pl extension write your codename



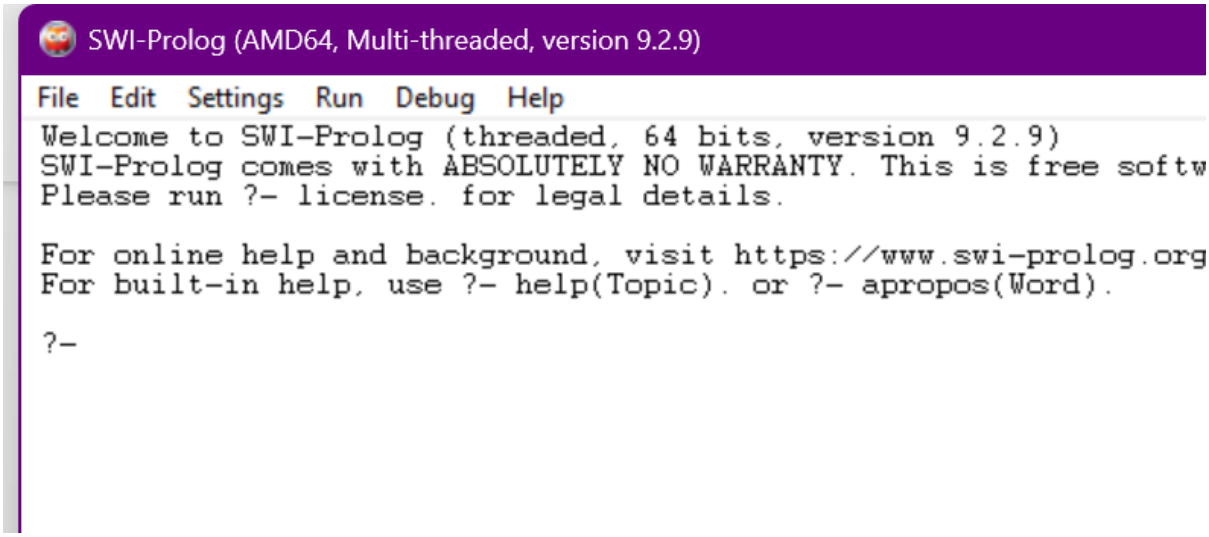
A window will be poped



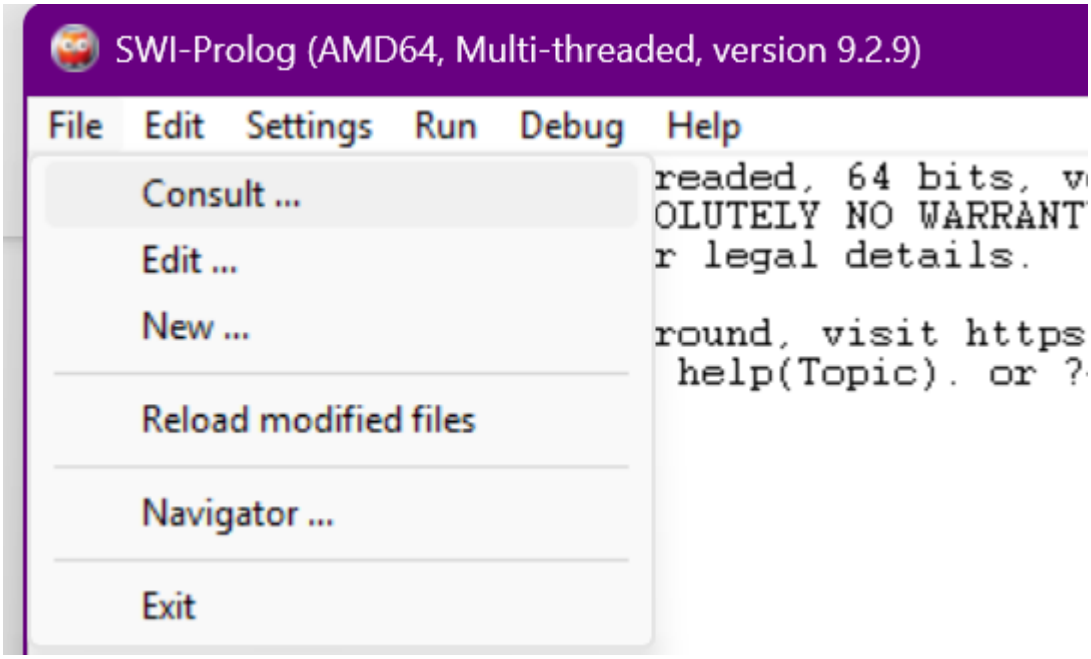
After completing your code ,click on “compile buffer” then click on “Make”



Then switch to the other window



Go to File and click on consult



After clicking on Consult a dialogue box will appear open the file that you saved with .pl extension and then run the command (Don’t forget to put the fullstop at the end)

Code:

```
batsman(sachin).
batsman(msd).
batsman(virat).
baller(anilkamble).
baller(harbajan).
cricketer(X):- baller(X).
cricketer(X):- batsman(X).
```

Output:

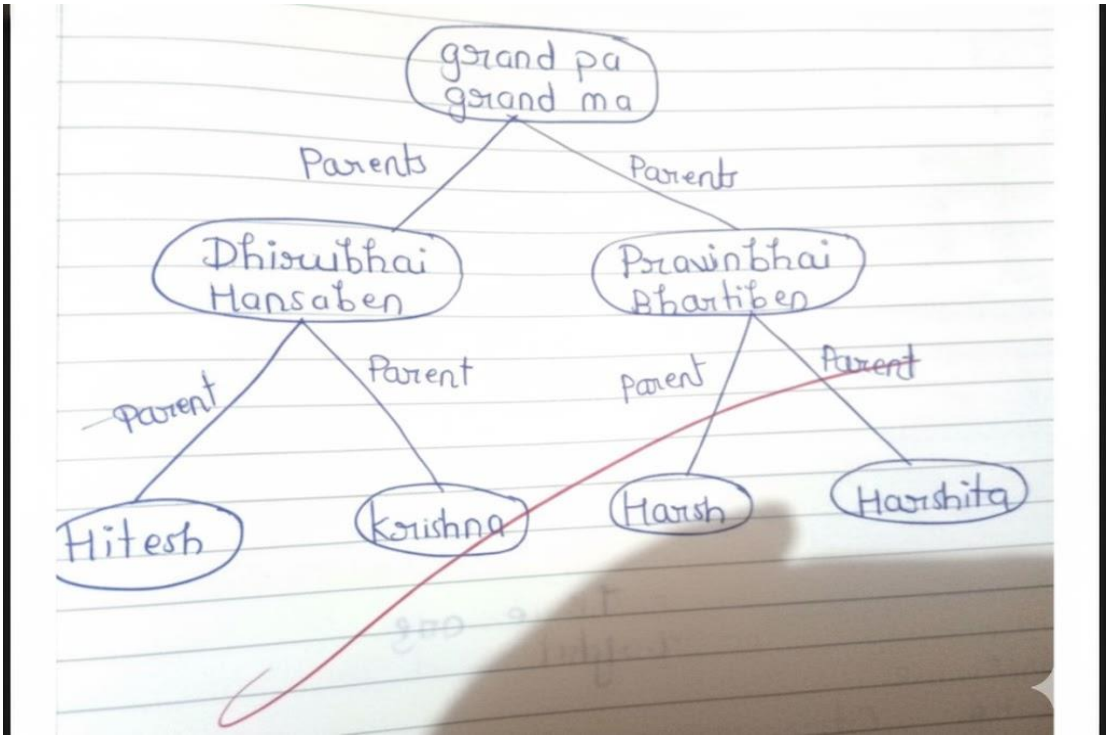
```
?- baller(X).
X = anilkamble .

?- batsman(virat).
true.

?- baller(msd).
false.

?-
```

Practical No 10

Write a program which contains three predicates: male, female, parent. Make rules for following family relations: father, mother, grandfather, grandmother, brother, sister, uncle, aunt, nephew and niece, cousin. Question:

i. Draw Family Tree.

ii. Define : Clauses, Facts, Predicates and Rules with conjunction and disjunction


grand pa
grand ma
Parents
Parents
Dhirubhai
Hansaben
Pravinbhai
Bhartiben
Parent
Parent
parent
Parent
Hitesh
krishna
Harsh
Harshita

Code:

fam.pl [modified]

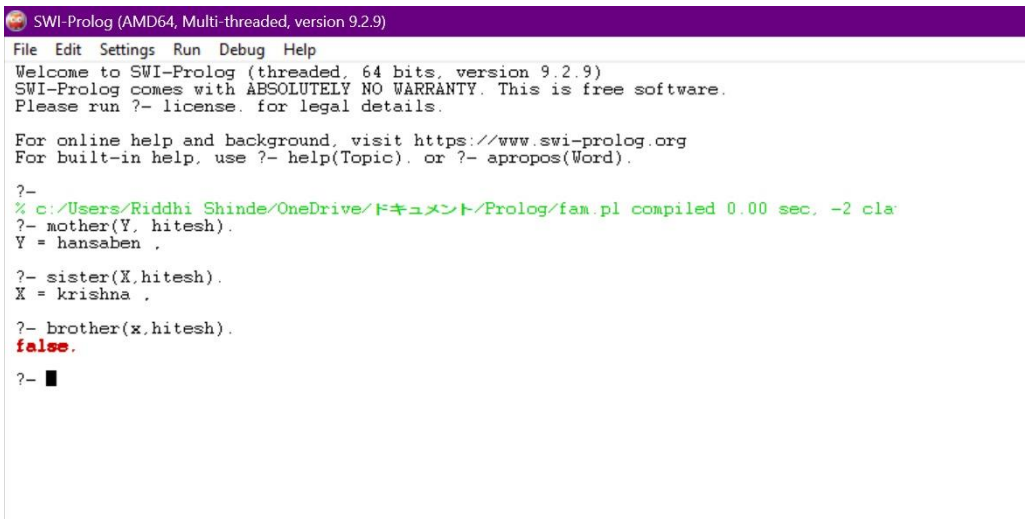
File Edit Browse Compile Prolog Pce Help

fam.pl [modified]

```
male(ramjibhai).
male(dhirubhai).
male(pravinbhai).
male(hitesh).
male(harsh).

female(shantiben).
female(hansaben).
female(bhartiben).
female(krishna).
female(harshita).

parent(shantiben, dhirubhai).
parent(ramjibhai, dhirubhai).
parent(ramjibhai, pravinbhai).
parent(shantiben, pravinbhai).
parent(dhirubhai, hitesh).
parent(hansaben, hitesh).
parent(dhirubhai, krishna).
parent(hansaben, krishna).
parent(bhartiben, harsh).
parent(bhartiben, harshita).


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

% siblings (share at least one parent)
sibling(X, Y) :- parent(P, X), parent(P, Y), X \= Y.

brother(X, Y) :- sibling(X, Y), male(X).
```

```
brother(X, Y) :- sibling(X, Y), male(X).
sister(X, Y)  :- sibling(X, Y), female(X).

% grandparents
grandparent(X, Y) :- parent(X, Z), parent(Z, Y).
grandfather(X, Y) :- grandparent(X, Y), male(X).
grandmother(X, Y) :- grandparent(X, Y), female(X).

% aunt / uncle (sibling of parent)
uncle(X, Y) :- parent(P, Y), brother(X, P).
aunt(X, Y)  :- parent(P, Y), sister(X, P).

% cousins (their parents are siblings)
cousin(X, Y) :- parent(PX, X), parent(PY, Y), sibling(PX, PY).

% nephew / niece (child of someone's sibling)
nephew(X, Y) :- male(X), parent(P, X), sibling(P, Y).
niece(X, Y)  :- female(X), parent(P, X), sibling(P, Y).
```

Output:


SWI-Prolog (AMD64, Multi-threaded, version 9.2.9)
File Edit Settings Run Debug Help
Welcome to SWI-Prolog (threaded, 64 bits, version 9.2.9)
SWI-Prolog comes with ABSOLUTELY NO WARRANTY. This is free software.
Please run ?- license. for legal details.
For online help and background, visit https://www.swi-prolog.org
For built-in help, use ?- help(Topic). or ?- apropos(Word).
?-
% c:/Users/Riddhi Shinde/OneDrive/ドキュメント/Prolog/fam.pl compiled 0.00 sec, -2 cla
?- mother(Y, hitesh).
Y = hansaben ,
?- sister(X,hitesh).
X = krishna ,
?- brother(x,hitesh).
false.
?-