INDEX

Supreme
Date
Page

NAME: Amallesh STD: SEC: ROLL NO.: SUB

S.No.	Date	Title	~~Page No~~ marks	Teacher's Sign / Remarks
1)	31/7/24	Sample programs in python using google colab Hotel price prediction.	9 10	
2)	7/8	Domain - Hotal price prediction	9	
3)	4/9	N queens	10	
4)	4/9	Depth first Search.	10	
5)	11/9	A* algorithm	10	
6)	18/9	AO* algorithm	10	
7)	25/9	Decision tree	10	
8)	9/10	k-means	10	
9)	16/10	Artificial neural network	10	
10)	23/10	MINIMAX	10	
11)	30/10	Introduction to prolog	10	
12)	6/11	prolog - family tree	10	

Completed

Exercise - 1

Ex: 1 31/7/24

1) Factorial

```
def factorial (n);
    if A = = 0;
    return;
    else
        return n* factorial(n-1)
num = int (input ("Enter a number: "))
print ("Factorial of " num "is" factorial
                                (num));
```

OUTPUT

Enter a number : 5

Factorial of 5 is 120

2) Sum of digits

```
def sum_digits(n);
    sum = 0
    while (n ! = 0)
    sum = sum + int(n %. 10)
        n = int (n/10)
    return sum;
num = int (input (" Enter a number "))
    print (sum : digits (num))
```

output:

Enter a number : 567
18

3) Palindrome

```
def isPalindrome(s):
    return s == s[::-1]
word = input("Enter a string")
ans = ispalindrome(word)
if ans:
    print("yes")
else
    print("No")
```

4) Amstrong Number

```
def isamstrong_number(numbers)
    sum = 0
    while (number > 0):
        digit = number % 10
        sum += digits ** 3
        number //= 10
    return sum;
number = int(input())
```

```
b=number
if Isarmstrong number(number)==b
    print("yes")
else
    print("No")
```

output:

23

5) Swapping of two Integers

```
def swap(a,b)
    a=a-b
    b=a+b
    a=b-a
    return a,b;
a=int(input())
b=int(input())
print(swap(a,b))
```

output : 10,20

20,10

6) add or even

```
num = int(input())
    if num %.2 == 0
        print ("Even")
    else
        print ("add")
```

output:

7

add

7) Area of rectangle

```
l = int(input())
b = int(input())
print (l * b)
```

output:

10

20

200

8) Prime or Not

```
def is prime (n);
  if n <= 1;
     print ("False"),
  for i in range (2, int (n ** 0.5) + 9);
     if n % i == 0;
        return False;
     return True;
  n = int (input())
  print (is prime (n))
```

output:

7

true:

9) Fibonacci series

```
$.def fibs(n);
  a, b = 0, 1
for (in range (n)
     print (a , end = " ");
     a, b = b, a+b
  n = int(input (n))
    print (fibo (n))
```

output

4.

0 1 1 2 3

10) Greatest common Divisor

```
def gcd (a, b)
  while b;
    a, b = b, a % b
  return a
n = int (input())
m = int(input())
print ( gcd (n, m))
```

31/07/2024

output:

16.

EX NO: 2

HOTEL PRICE PREDICTION

Domain: Tourism

In the tourism, Prediction hoted price is a key application of machine learning aimed at optimizing pricing strategies and enhancing competetiveness. Hotel face fluctating demand based on factors like location, seasonally and amenities machine learning algorithm analyze historical data and various forecate room rate, enabling hotel to set dynamic price that maximize revenue and occumpany.

PROJECT DESCRIPTION

Machine learning techenique can analyse various factor influencing hotel pricing, such a location.

Star rating, amenities and seasonal demands. By leveraging historical data. and sophisticated algorithms hotel and booking platforms.

platforms can forecast price more efficiently, adapt to market trends and provides personilized recommendation

End Users:

1. Hotel operators
2. Travel agencies
3. online travel platforms
4. consumers
5. Revenue Managers.

Linear Regression

* Linear regression is a type of supervised machine learning algorithm that computes that computes the linear relationship between the dependent variable and one or more independent features by fitting by linear operation to observed data.

* When there is only one independent feature. It is known as Simple linear Regression. While there are more than one feature. it is Multiple Linear Regression.

Depth First search:

4/9/24

Ex: 3

```
def dfs_recursive(graph, start, visited=None):
    if visited is None:
        visited = set()
    visited.add()
    print(start, end=' ')
    for neighor in graph[start]:
        if neighbor not in visited:
            dfs_recursive(graph, neighbor,
                          visited)
    return visited
def dfs_iterative(graph, start):
    visited = set()
    stack = [start]
    while stack:
        vetex = stack.pop()
        If vertex not in visited:
            print(vertex, end=' ')
            visited.add(vertex)
    stack.extended(neighor for neighor in
    graph[vertex] if neighor not in visited)
    return visited.
```

```
graph = {
    'A' = ['B', 'C'],
    'B' = ['D', 'E'],
    'C' = ['F'],
    'D' = [],
    'E' = ['F'],
    'F' : [],
}
print ("DFS using recursion:")
dfs_recursive (graph, 'A')
print ("\nDFS using iteration:")
dfs_iterative (graph, 'A')
```

OUTPUT:

DFS Using recursing:

A B D E F C

DFS Using iteration:

A C F B E D

{'A', 'B', 'C', 'D', 'F', 'F'}

RESULT:

Thus the DFS Algorithm has been executed successfully.

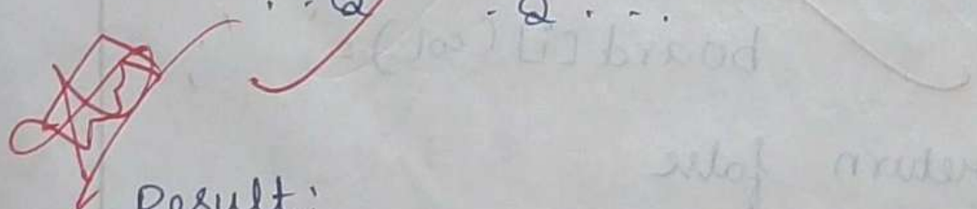
Ex: 4 N - Queen Problem Date: 4/9/24

```
def is_safe (bord, row, col):
    for i in rang (col):
        if bord [row] [i] == 1;
            return False
    for i, j in zip (round (row -1, -1),
                     range (col, -1, -1)):
        if bord [i] [j] == 1;
            return false
    for i, j in zip (range (row, n, 1))
                    range (cal, -1, -1)
        if board (i] (j] == 'Q';
            return false.
    for i in range (n);
        if is_safe (board, i, col)
            board [i] (col) = 'Q'
        if solve_rec (board, cal + 1)
            board [i] (col) = ' ';
    return false
def print_solution (board);
    for row in board;
        print (" ", join (row))
```

```
board = [['.' for _ in range(n)].
                for _ in range(n)]

if solve_rec(board, 0);
        print_solution(board)
else
    print("No solution exist")

n = 4

solve_n_queens(n)
```

OUTPUT

```
n = 4
..Q....Q
...Q.Q..
Q.....Q.
..Q..Q..
```



Result:

The above N Queen problem has been executed successfully

Ex:no: 5 A* PROBLEM Date: 11/9

```
import heapq

def heuristic (a ; b);
    return abs(a[0] - b [0]+ abs abs(a(1)
                                + b(1))
def a-s for search (grid, start, goal);

    row, col = len (grid) , len (grid (0))
    open_list ()
    heapq. heappush (open_list, col,
                                start ))

    came_From = {};
    g - score = {start : 0}
    f_score = {start: heuristic (start_goal)}
while open_list
        current = heapq . heappop (open-list)
                                    (1)
    if current = = goal;
            path = [ ]
        while current in came-from;
            path . append (current)
```

```
        current = came_from[current]
    path.append(start)
    return path[::-1]
for array in [(-1, 0), (1, 0) (0, -1)
                                (0, x);
    neighbour = ((current[0]+
                dx, current[1] + dy))
    if oc = neighbour[0] < rows
and ac = neighbour(1) < cols
    and grid[neighbor[0];
    tentative g_score = g_score
                [current]+1
    if neighbour not in
    g_score or tentative g_score.
                (g-score[neighbor])
    come_from[neighbar] = row
    f_score[neighbour] = tentative
                (g-score)
return None
```

Input

```
a = [
  [0,1,0,0,0],
  [0,1,0,1,0],
  [0,0,0,1,0],
  [1,1,0,0,0]
  ][0,0,0,1,0]
  ]
```

OUTPUT:

path found. [(0,0), (1,0), (2,0), (2,1),
(2,2), (3,2), (3,3), (3,4) (4,4)]

Result:

Thus the above A* program has been completed successfully.

Ex: 6 A* PROGRAM: Date: 18/9/24

```
class Node

    def-init-(self, name value)
        Self.name = name
        Self.value = value
        Self = child = ens()

    def add_child(self, node)
        Self.children(node)
    def-a0-star-(root, goal):
    queue = [(root, (root))]
    while queue:
    node.path = queue pop(0).
    if node == goal
        return path
    for child in node.children
        new_path = path + (node)
        queue.append((child))

return new_path
```

Input:

A= Node ("A", 0)

B = Node('B', 1)

C = Node('C', 2)

D = Node('D', 3)

E = Node('E', 4)

OUTPUT:

solution path .['A', 'B', 'D', 'E']

RESULT:

Thus the above program has been executed successfully.

DECISION TREE

Ex:7 Date: 25/9/24

AIM:

To implement a decision tree classification technique for gender classification using python.

```
import pandas as pd
from sklearn.tree import DecisionTreeClassifier

data = {
Height : [152,155,172,185,167,180,157,180,164,177]
Weight : [45,57,72,85,68,78,22,90,66,85]
Gender : ['Female','Female','Male','Male','Female',
'Male','Female','Male','Female','Male'] }

df = pd.DataFrame(data)
X = df[['Height','Weight']]
Y = df['Gender']
Classifier = DecisionTreeClassifier()
Classifier.fit(X,Y)
Height = float(input("Enter height (in cm) for
                     Prediction :"))
Weight = float(input("Enter weight (in kg) for
                     Prediction :"))
```

```
random_values = pd.DataFrame([[height, weight]], columns=['Height', 'Weight'])
predicted_gender = classifier.predict(random_values)
print("Predicted gender for height {height} cm and weight {weight} kg: {predicted_gender[0]}")
```

OUTPUT:

Enter height for prediction : 169
Enter weight for prediction : 61
Predicted gender for height 169.0 cm weight 61.0 : Female.

RESULT:

Thus the Decision tree classification has been implemented successfully.

Implementation ARTIFICIAL NEURAL NETWORKS FOR AN APPLICATION USING PYTHON - REGRESSION

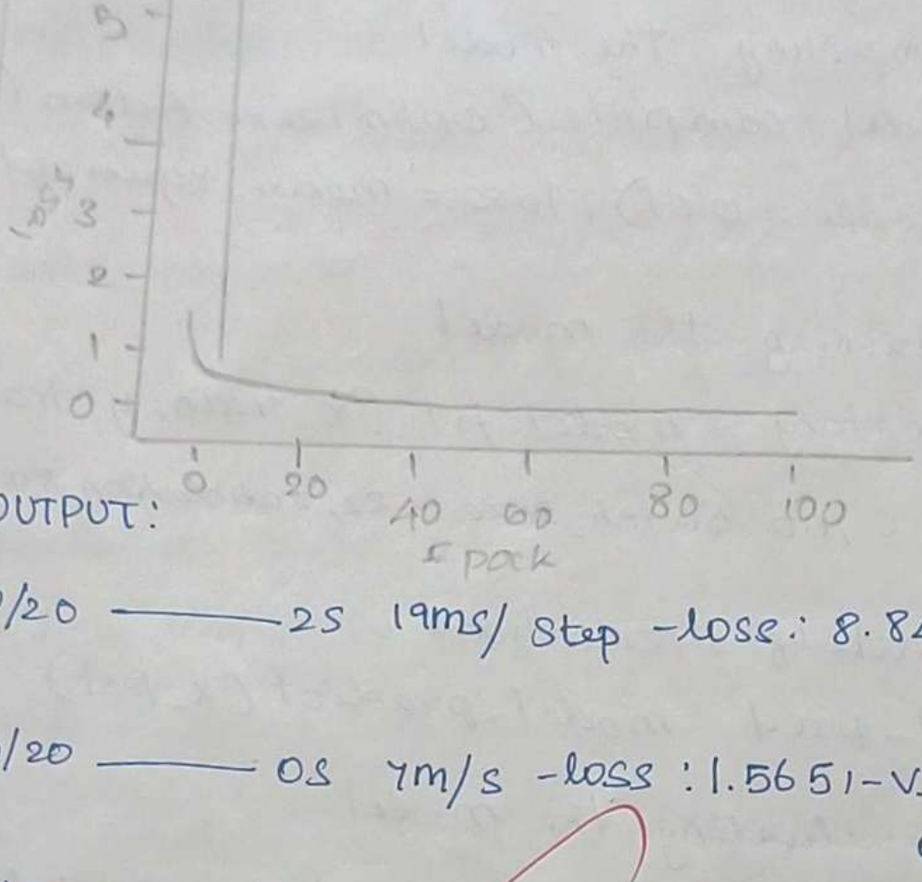
AIM:

To Implementation artificial neural for an application in classification using. python.

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import standardscaler
from keras.models import sequential
from keras.layers import Dense
from keras.optimizers import Adam
import matplotlib.pyplot as plt
# Generating syntetic dataset for demonstration
# Replace this with your own dataset
np.random.seed(42)
X = np.random.rand(1000,3) # 1000 samples, 3 features
```

```
Y = 8 * X[:, 0] + 2 * X[:, 1] ** 2 + 1.5 * np.sin
                (X[:, 2] * np.pi) +
np.random.normal(0, 0.1, 1000)  # Non-linear relation

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split
                (X, Y, test_size = 0.2 random_state = 42)

# Feature scaling
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Build the Ann model
model = Sequential()
# input layer and first hidden layer
with 10 neurons and ReLU activation
model.add(Dense(10, input_dim = X_train.
        Shape[1], activation = 'relu'))

# Second hidden layer with 10 neurons
and ReLU activation model.add(Dense
        (10, activation = 'relu'))
```

```
Output layer with linear activation
for regression model.add(Dense(1, activation='linear'))

Compiling the model
model.compile(optimizer=Adam(learning_
rate=0.01), loss='mean_squared_error')

Training the model
history = model.fit(X_train, y_train, epochs
=100 batch_size=32, validation_split=0.2, verbose=1)

Making predictions
y_pred = model.predict(X_test)

Evaluating the model
mse = np.mean((y_test - y_pred.flattend()**2)
print(f'mean squared Error: {mse:.4f}')

plotting training history
plt.figure(figsize=(12,6))
plt.plot(history, history['loss'], label='Trainingloss')
plt.plot(history.history['val_loss'], label='validationloss')
plt.title('Training and validation Loss')
plt.title('Training')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()
plt.show()
```



OUTPUT:

20/20 ——— 2s 19ms/ step -lose: 8.843 - Val: 1.9373

20/20 ——— 0s 7ms/s -loss: 1.5651 - V_loss: 0.5565

20/20 ——— 0s 4ms/step loss: 0.4276.

Val_loss: 0.5148

RESULT:

Thus the Artificial Nural Network has been verified successfully

K-MEANS

Ex: 8 Date: 9/10/24

AIM:

To implement a k-means clustering technique using python language.

```
Import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import Kmeans.
from sklern.datasets import make_blobs
```

step 1: import kmeans from sklearn.cluster
(already done above)

Step 2: Generate synthetic data and assign X and y

Creating a dataset with 3 clusters

```
X,y true = make_blobs(n_samples = 300, centers3,
                      Cluster_std = 0.60 random_state = 0)
```

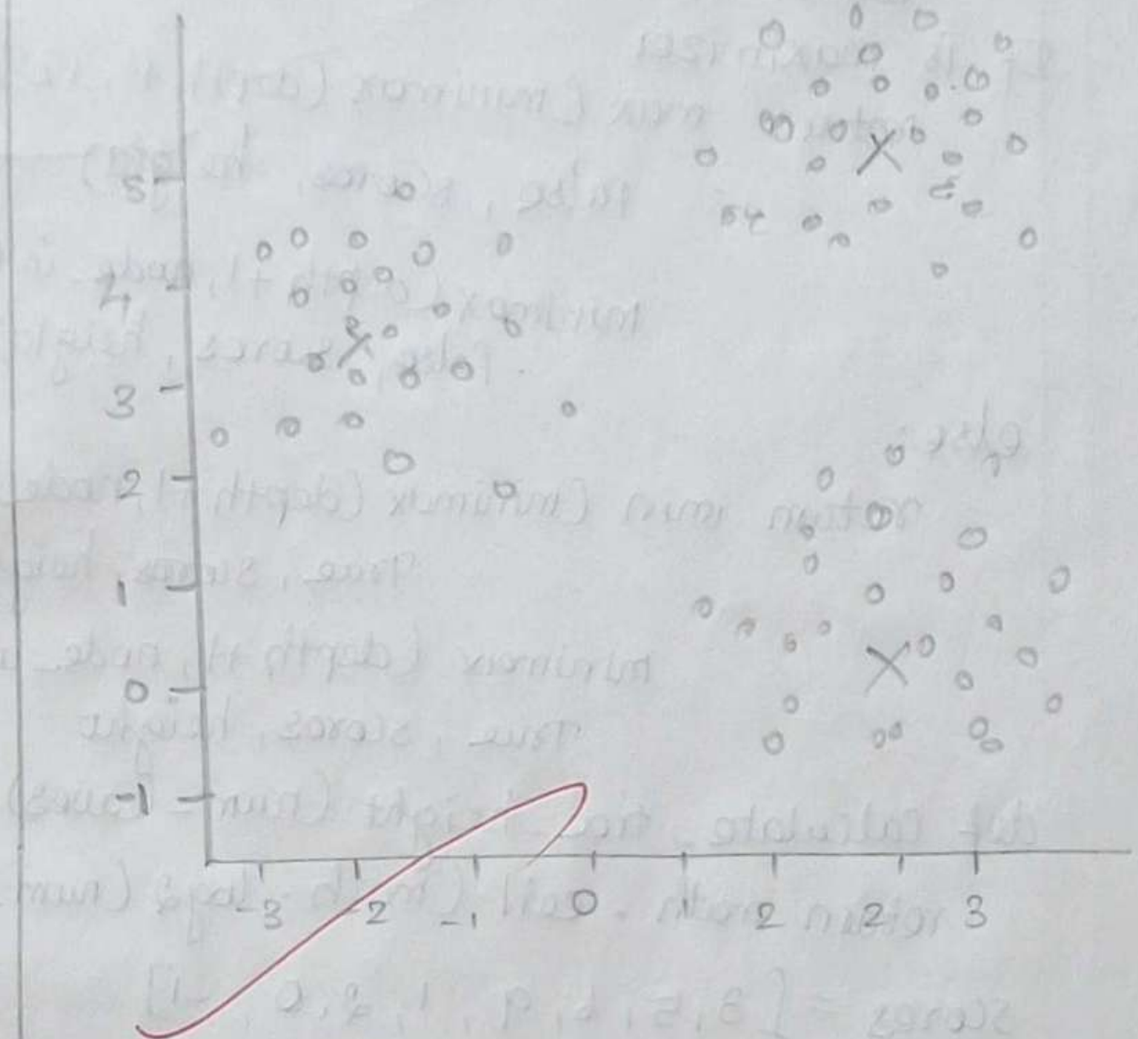
Step 3: Call the function kmeans()

Fit k-means with the chosen number of cluster (k = 3))

```
k = 3

kmeans = kmeans(n_cluster = k, random_state =
y_kmeans = kmeans.fit_predict(x)
```

Step 4: perform scatter operation and display the output

```
plt.figure(fig size = (8,6))
plt.scatter(x[:,0], x[:,1], c=y_kmeans, s=30
    cmap = 'viridis', label = 'clusters')

Centers = k means.cluster_centers
pts.scatter(centers[:,0], centers[:,1],
    c='red', s=200, alpha = 0.75, maker = 'x'
label = 'centroids')
plt.title('k-means clustering Results')
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
plt.legend()
plt.Show()
```



RESULT:

Thus the k-means clustering has been implemented successfully.

MINIMAX ALGORITHM

```
import math
def minimax (depth, node_index is maximizer,
             scores, height):
    if depth == height
        return scores [node_index]
if is maximizer
    return max (minimax (depth+1, node_index*2,
                         false, scores, height)
               minimax (depth+1, node_index*2+1,
                        false, scores, height)
else:
    return min (minimax (depth+1, node_index*2,
                         True, scores, height)
               minimax (depth+1, node_index*2+1,
                        True, scores, height
def calculate_tree_height (num_leaves)
    return math.ceil (math.log2 (num_leaves)
scores = [3, 5, 6, 9, 1, 2, 0, -1]
tree height = calculate_tree_height (len scores)
optimal_score = minimax (0, 0, True, scores,
                         tree height
Print (f" The optimal score is : {optimal score}
```

→ The optimal score is : 5

OUTPUT:

The optimal score is: 5

RESULT:

Thus the minimax Algorithm has been implemented Successfully.

INTRODUCTION TO PROLOG

Ex. 11 Date: 30/10/24

AIM:

To learn PROLOG terminologies and write basic programs.

TERMINOLOGIES:

1. Atomic Terms :-

Atomic terms are usually strings made up of lower and upper case letters digits and under score, starting with a lower case letter.

Ex

dog

ab_c_321

2. variables :

variables are strings of letters digits and under score. Starting with a capital letter or an underscore

Ex

Dog

Apple_420

3. Compound Terms:

Compound terms are made up of PROLOG atom and a number of arguments

(PROLOG terms, ie - atoms, numbers, variables, or other compound terms) enclosed in parantheso and separated Commas

Ex is bigger (elephant, x)
f(g)(x).

4. Facts:

A fact is a predicate followed by a

Ex

bigger - animal (whale)
life is beautiful

5 - Rules:

A rule consists of a head (a predicate) and a body (a sequence of predicates separated by commas)

Ex

is smalled (x, y) :- is - bigger (y, x)
aunt (Aunt, child) :- sister (Aunt, Parent)
parent (parent, child)

SOURCE CODE:

KB1

woman (mia)
woman (jody)
woman (yolanda)

plays Air Guitar (jody)

Party

Query 1: ? - woman (mia)

Query 2: ? - plays Air Guitar (mia)

Query 3: ? - party

Query 4: ? - concert

OUTPUT

? - woman (mia)

true

? - plays Air Guitar (mia)

false

? - party

true

? - concert

ERROR - Unknown procedure : Concert/0 (DUM correct goal)

? -

KB2:

happy (yolanda)

listen 2 music (mia)

listens 2 music (yolanda) - happy (yolanda)

plays Air Guitar (mia) :- listens 2 music (mia)

plays Air Guitar (yolanda) :- listens 2 music (yolanda)

OUTPUT

```
? - plays Air Guitar (mia)
true
? - plays Air Guitar (yolanda)
true
?
```

KB8:

```
likes (dan, sally)
likes (sally, dan)
likes (John, birthday)
married (X, X) :- likes (X, Y) likes (Y, X)
friends (X, X) :- likes (X, Y) : likes (Y, X)
```

OUTPUT:

```
? - likes (dan. X)
X = sally
? - married (dan, sally)
true
? _ married (John, birthday)
false
```

KB4

```
food (burger)
food (sandwich)
food (pizza)
lunch (sand wich)
dinner (pizza)
meal (x) :- food (x)
```

OUTPUT

```
?-
1 food (Pizza)
true
?- real(x) lunch(x)
x - sandwich
?- dinner (sandwich)
false
```

KB5

```
owns (jack, car (bmw)
owns (john, car (chevy))
owns (olivia, car 'civic)
owns (jane, car (chevy)
sedan (car (bmw))
sedan (car (civic))
truck (car (chevy)
```

OUTPUT:

```
?-
1        owns (john, x)
x = car (chevy)
?- owns (john-)
true
?- owns (who, car (chevy)
who - john
?- owns (jane, x) sedan (x)
false
?- owns (jane, x), truck (x)
```

x = car (chevy)

RESULT:

RESULT:

The above program is executed and verified successfully.

PROLOG - FAMILY TREE

EX: 12 Date: 6/11/24

AIM:

To develop a family tree program using PROLOG with all possible facts, rules, and queries

SOURCE CODE:

KNOWLEDGE BASE:

```
/* FACTS : */
male (peter)
male (john)
male (chris)
male (kevin)

female (betty)
female (jeny)
female (lisa)
female (helen)

Parent of (chris, peter)
Parent of (chris, betty)
Parent of (helen, peter)
parent of (helen, betty)
Parent of (kevin, chris)
Parent of (kevin, lisa)
Parent of (jeny, john)
Parent of (jeny, helen)
```

* RULES :: *

/ son - parent

*son - grand parent *)

father (x, y) :- male (y), parent of (x, y)

mother (x, y) :- female (y), parent (x, y)

grand father (x, y) :- male y, parent of (x, z)
parent of (z, y)

grand mother (x, y) :- female (x), parent of (x, z)
parent of (z, y)

brother (x, y) :- male (y), father (x, z) father
(y, w) z = w

sister (x, y) :- female (y,) father (x, z), father (y, w)
z
= w

RESULT:

Thus the above program has been executed and verified successfully.