Exp.No:1

BASIC PYTHON PROGRAMS

1) Factorial

```
def factorial (n):
 if A = = 0
      return 1
 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 12
```

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

3) Palindrome

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

4) Armstrong number

```
def is_armstrong_number (number):
    sum = 0
    while (number > 0):
        digit = number % 10
        sum += digit * 3
        number = 10
    return sum
number = int (input ())
b = number
if is_armstrong.number (number) = b
print ("Yes")
```

```
else:
    print ("NO")
```

output

23

5) Swapping of two numbers

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

Output 10, 20

20, 10

6) Odd or even

```
num = int (input())
if num % 2 == 0:
        print ("Even")
else
    print ("odd")
```

output

7

Odd

7) Area of a 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)+1):
    if n % i <<0:
        return False
    return True
n= int (input ())
print (is_prime(n))
```

Output :

7

true

9) Fibonocci series

```
def fib(n):
a, b = 0, 1
for i in range(n);
        print(a, end=' ')
        a, b = b, a+b
n = int(input())
print(fib(n)
```

output :

4

0 1 1 2 3

10) Greatest common Division

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

output

16.

31/01/2024

Predictive analysis of student final Performance.

~~Domait~~ Domain Type:

Educational Data Mining and Learning Analytics

Problem Statement

Predictive Modeling of Student's final performance

Algorithm:-

~~Statistical~~ Models

* Decision trees:

Model decision making processes by splitting data based on feature value

* USE CASE: understanding which features (eg attendance, participation

most influence final grades

1) Criteria for splitting

* For Classification

  1) Gini impurity
  2) Entropy

* For Regression

2) Splitting Process:

1) Choose a Feature:

Select the feature that provides the best split according to the Criterion

2) Split Data

Divide the data into Subset based on the chosen features

3) Repeat.

Objective:

Develop and implement a decision tree model to predict Students' final performance in a specific course based on historical academic data, behavioral metrics and personal characteristics. The aim is to accurately forecast students' final grade or performance outcomes.

People benefit by this project

1) students
2) Educators
3) Academic Institutions.
4) Parents and Guardians.

Ex.no: 9

Date. 4/9/24

N QUEENS

Aim:

To implement a N Queens problem.

Program:

```
def solve nqueens(n):

    def is_safe(row, col, board):
        for i in range(row):
            if board[i] == col:
                return false
        for i, c in enumerate(board[:row]):
            if abs(row - i) == abs(col-c)
                return False
        return true

    def solve-recursive(row, board):
        if row == n:
            return [board[:]]
        solution = []
```

```
for col in range(n):
    if is_safe(row, col, board):
        board.append(col)
        solution.extend(solve_recursive
                           (row+1, board))
        board.pop()
    return solution

solution = solve_recursive(0, [])
return solution
n=8
solution = solve_nqueens(n)
if solution:
    print(f"Found {len(solution)}
              solution for the {n}-
                  Queens problem:")
    for solution in solution:
        print(solution)

else:
    print(f"No solution found
          for the {n}-Queens
                   problem.")
```

Output:

0 0 1 0
1 0 0 0
0 0 0 1
0 1 0 0

True

Result:

Thus the program was executed successfully & out is verified

Exno: Date.

Depth First Search

Ex

Aim:
To implement a depth-first Search problem using python

Program:

```
def dfs(graph, start_node):

    visited = set()
    stack = [start_node]
    traversal_order = []

  while stack:
      node = stack.pop()
    if node not in visited:
        visited.add(node)
        traversal_order.append(node)
        neighbors = graph.get(node, [])
      stack.extend([neighbor for
          in reversed(neighbors)
          if neighbor not in
            visited])
```

```
return traversal_order

graph = {
    'A' : ['B', 'C'],
    'B' : ['D', 'E']
    'C' : ['F']
    'D' : [],
    'E' : ['F'],
    'F' : []

start_node = 'A'
dfs_order = dfs (graph, start_node)
print ("DFS transversal order:",
                        dfs_order)
```

Output:

1 2 0 3 4

Result:

Thus the program was executed successfully & output was verified

Date
~~Exno~~: 11/9/24
Exno: 5

A* algorithm

Aim: To implement a A* algorithm using python

Program:

```
import heapq
def _init_(self, state, parent=none, cost=0,
                                  heuristic=0):
    self.state = state
    self.parent = parent
    self.cost = cost
    self.heuristic = heuristic
    self.f_value = cost + heuristic

def _lt_(self, other):
    return self.f_value < other.f_value

def a_star(graph, start_node, goal_node,
           heuristic_func):
```

```
open_list = [Node(start_node, cost=0,
                  heuristic = heuristic_func
                              (start_node))]

closed_list = Set()

while open_list:
    current_node = heapq.heappop
                       (open_list)

    if current_node.state == goal_node:
        path = []
        while current_node:
            path.insert(0, current_node.state)
            current_node = current_node.
                                parent
        return path

    closed_list.add(current_node.
                        state)

    for neighbor, cost in graph[current_
                              node.state]
                              .item():

        if neighbor not in closed_list

            new_cost = current_node.cost +
                               cost
```

```
new_heuristic = heuristic_func
                    (neighbor),
        current_node, new_cost, new_
                              heuristic)

in_open_list = False

for open_node in open_list:
    if open_node.state == neighbor
            and open_cost <= new_
                              cost

        in_open_list = True
        break

    if not in_open_list:
        heapq.heappush (open_list,
                           new_node)

return none

graph = {
    'A' : {'B' : 1, 'C' : 3},
    'B' : {'D' : 2 , 'E' : 4},
    'C' : {'F' : 5 },
    'D' : {'G' : 1}
```

```
'E' : {'G' : 5},
'F' : {'G' : 2},
'G' : {}
}

start_node = 'A'
goal_node = 'G'

shortest_path = a_star(graph,
                 start_node, goal_node,
                     heuristic)
if shortest_path
    print("shortest path",
                shortest_path)
else
    print("No path found")
```

output:

Path found: ['A', 'E', 'D', 'G']

Result:

Thus the program was executed successfully & output was verified

Ex no: 18/9/24 Date: 18/9/24

AO*

Aim: To implement a AO* algorithm using python

```
class node
    def-int- (self-name value)
        Self. name = name
        Self. value = value
        Self = child = end()
    def add-child (self node)
        Self. children (node)
    def- ao- sba (root, goal)
        queue = ((root, (root)))
    while queue:
    node. path = queue pop (0)
        if nade = = goal
    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 succesfully

1.

Ex no 7

Date 25/9/24

DECISION TREE Classification

Aim: To implement a Decision tree.

Program

```
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, 88]
  '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. Data Frame([[height, weight]],
                    columns = ["Height", "weight"])
predicted gender = classifier. predict (random values)
print( f"predicted gender for height {height} cm
                    and weight} kg: {predicted_gender[0]}")
```

Output

Enter height for prediction : 169

Enter weight for prediction : 61

Predicted gender for height 169.0 cm and weight 61.0 kg: Female.

Result:

The above program was executed and implement successfully.

Ex: no. 9

Date 16/10/24

Implementing Artificial Neural network for an application using python - classification.

Aim:

To implementing artificial networks for an application in classification using python.

Program.

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import standard scaler
from keras.models import sequential
from keras.layers import Dense
from keras.optimizers import Adam
import matplotlib.pyplot as plt

# Generating synthetic dataset for demonstration
# Replace this with your own dataset
np.random.seed(42)
x = np.random.rand(1000, 3) # 1000 Samples, 3 feature
y = 3 * x[:, 0] + 2 * x[:, 1] ** 2 + 1.5 * np.sin(x[:, 2] *
        np.pi) + np.random.normal
    (0, 0.1, 1000) # Non_linear relationship.
```

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 = Standard scaler()
x train = scale.fit transform (x train)
x test = scaler. transform (x_test)

Building the ANN Model
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 RoLu 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.flatten() ** 2)
print (f mean squard Error = {mse:.4f}")
```

Plotting training history

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

Out put

Epoch 1/100

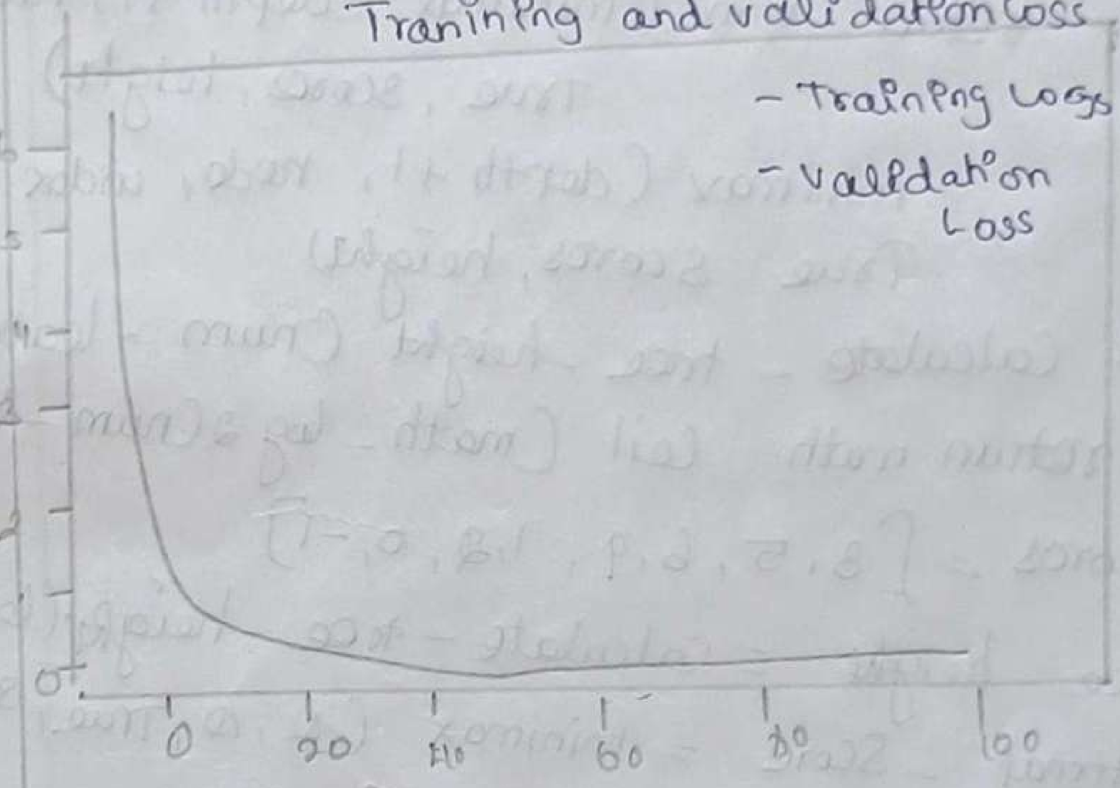
20/20 ———————— 2s 19ms/st - loss: 8.8470

- val_loss : 1.9373

Epoch 2/100

20/20 ———————— 0s 7ms/step - loss 1.5651

- val_loss : 0.6565



Output:

The above ANN program was Successfully executed and Implemented.

Exno: 10 Implementation of Adversarial Techniques Minmax 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 - log 2 (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:

The above program was Successfully implement and exeuted.

Ex no: 11

Date: 9/10

Implementation of Clustering Technique K-Means

Aim:

To implement a K-Means clustering technique using python language.

Import numpy as np
import mat plotlib.py plot as pit
from skiearn. cluster import k means
from sklern. data sets import make_blobs

Step 1: import K means from sk learn. cluster (already done above)

Step 2: Generate synthetic data and assign x and y

Creating a data set with 3 clusters

x, y true = make_blobs (n snaples = 300 centre 3 cluster_std = 0.60 random_state = 0)

Step3 : Call the function k-means()

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

k=3
k means = k means (n_cluster = k, random_state
y-k means = k means - fit predict (x)

Step 4 : perform scatter operation and display
the output

plt_figure (fig size = (8,6)]
ptt - scatter (x[:0], x[;], c=y _ k means, s= 30
c=map = 'virdis', label =('clusters')

Centers = k means. cluster - Centers

pts- Scatter. (Centers[;0], Centers[; 1]
c= 'red', s = 200, alpha = 0.75, marker = 'x'
Label =("centroids')

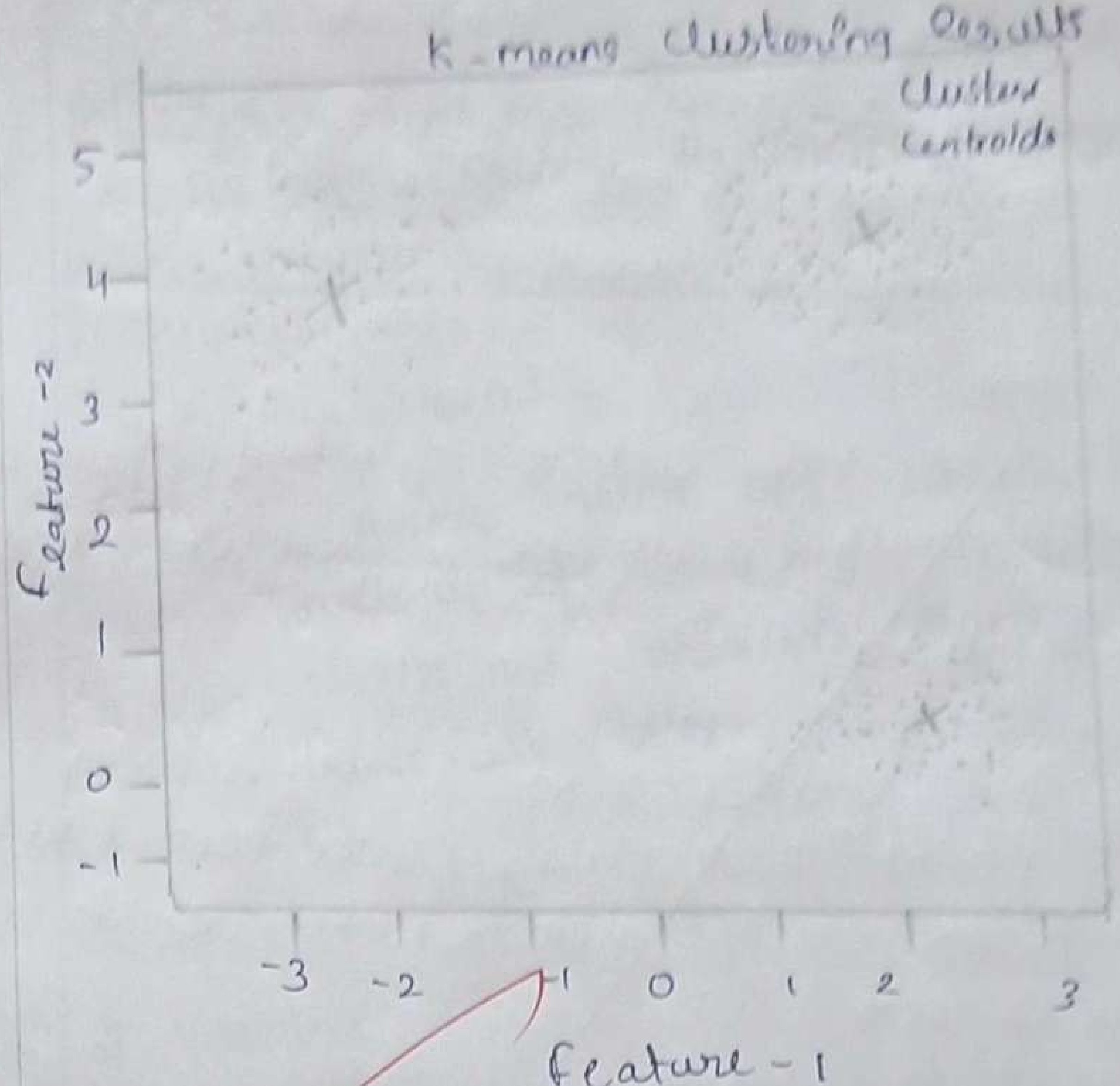
plt - title (k - means clustering Results)

plt . xlabel ('feature')

plt. y label ('feature 2')

plt - legend ()

plt - Show ()



Result:

The above program was executed and implement successfully.

Ex: 11 INTRODUCTION TO PROLOG Date: [illegible]

AIM:

To learn PROLOG terminologies and write basic programs.

TERMINOLOGIES:

1. Atomic Terms:-

Atomic terms are usually strings made up of lower and uppercase letters, digits and underscore, 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 madeup of PROLOG atom and a number of arguments (PROLOG terms, ie atoms, numbers, variables, or other compound terms) enclosed in parenthese and separated by commas.

Ex. is bigger (elephant, x)

f(g(x,_),7)

4. Facts :-

A fact is a predicate followed by a d

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 commas)

Ex

is smaller (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 (DWIM could not 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.
?-

KB3:

likes (dan, sally)
likes (sally, dan)
likes (John, brittney)
married (x, y) :- likes (x, y), likes (y, x)
friends (x, y) :- likes (x, y) : likes (y, x)

OUTPUT :-

?- likes (dan . x)

x = sally

?- married (dan, sally)

true

?- married (john, britthay)

false

KB4

food (burger)
food (sandwich)
food (pizza)
lunch (sandwich)
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 :

?-
| 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 :

To learn prolog terminologies and write basic programs.

Ex no: 12 Date: 6/11/24.

PROLOG - FAMILY TREE

AIM:

To develop a family tree program using PROLOG with all possib 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 of (x,y)

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

grand mother (x,y):- female (y), 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),
=w

Result:

The program was executed and output was verified.

INDEX

Supreme

Date
Page

NAME: Akilesh. B STD: CSE SEC: A ROLL NO: 021 SUB: POAI

completed