INDEX

3rd year CSE-A 220701030

NAME: Arindh.V STD.: _____ SEC.: _____ ROLL NO.: _____ SUB.: POAI Observation.

S. No.	Date	Title	~~Page No.~~ marks	Teacher's Sign / Remarks
1.	31/7/24	Sample python program Using Google Colab.	9	
2.	7/8/24	Domain: Sports Analytics. Cricket player performance Performance Prediction	10	
3.	4/9/24.	N Queens Problem	10	
4.	18/9/24.	Depth First search	10	
5.	~~18~~/9/24	A* Algorithm	10	
6.	18/9/24	AO* Algorithm	10	
7.	25/9/24	Decesion tree	10	
8	8/10/24	Kmeans	10	
9.	16/10/24	Artificial neutral network	10	
10	23/10/24	Minimax	10	
11.	30/10/24	Introduction to Prolog	10	
12.	6/11/24	Prolog family tree	10	

completed

GK00322915

Ex:-1 Solving Python Programs Using Google Colab.

1. Simple calculator.

```
a = int(input("enter a number1: "))
b = int(input("enter a number2: "))
print("1.add 2.Subtract 3. multiply 4.divide")
n = int(input("enter choice:"))
if n==1:
    print(a+b)
elif n==2:
    print(a-b)
elif n==3:
    print(a*b)
elif n==4:
    print(a/b)
else: print("the given option is invalid").
```

2. Number of digits

```
a = int(input("enter the number"))
count=0
i=a
while(i>0):
    count=count+1
    i=i//10
print("The num of digits in a:",count).
```

3. Armstrong Number

```
num = int(input("Enter a number: "))
sum = 0
n1 = len(str(num))
temp = num
while temp > 0:
    digit = temp % 10
    sum += digit ** n1
    temp //= 10
if num == sum:
    print(num, "is an Armstrong number")
else:
    print(num, "is not an Armstrong number")
```

4. Factorial

```
def recur_factorial(n):
    if n == 1:
        return n
    else:
        return n * recur_factorial(n-1)
num = int(input("Enter a number: "))
if num < 0:
    print("Factorial does not exist for negative numbers")
elif num == 0:
    print("The Factorial of 0 is 1")
else:
    print("The factorial of", num, "is",
          recur_factorial(num))
```

5. Multiplication table

```
def Multiplication_table():
    num = int(input("Enter a number: "))
    print(f"Multiplication Table for {num}:")
    for i in range(1,11):
        print("{num x {i} ={num* i}")
Multiplication_table()
```

6.

```
def count_vowels()
    S=input("Enter a string")
    vowels = "aeiouAEIOU"
    count=0
    for char in S:
        if char in vowels:
            count+=1
    print("Number of vowels in the string:{count}")
count_vowels()
```

7. Remove duplicates:

```
def remove_duplicates():
    n = int(input("Enter number of elements in the list:"))
    list = []
    for i in range(n):
        element = input(f"Enter element{i+1}:")
        list.append(element)
    unique_list = list(set(list))
    print(f"List after removing duplicates: {unique_list}")
remove_duplicates()
```

8.

```
def square_elements():
    n = int(input("Enter the number of elements in list:"))
    list = []
    for i in range(n):
        element = float(input(f"Enter element{i+1}:"))
        list.append(element)
    SquaredList = [x**2 for x in lst]
    print(f"Squared elements: {squared_list}")
square_elements()
```

9. Even

```
def C
```

10.

9. Even or odd.

```
def check_even_odd():
    print("Check if a number is Even or odd")
    num = int(input("Enter a number: "))
    if num % 2 == 0:
        print(f"{num} is Even")
    else:
        print(f"{num} is odd")
check_even_odd()
```

10. Palindrome

```
def palindrome_check():
    s = input("Enter a string: ")
    reversed_s = s[::-1]
    if s == reversed_s:
        print("{s} is a palindrome")
    else:
        print("{s} is not a palindrome")
palindrome_check()
```

Output :-

1.) enter a number1 : 5
enter a number 2 : 3
1. add 2. Subtract 3. Multiply 4. Divide
enter a choice : 1
8

2.) enter the number: 9753909.
The number of digits in 9753909 : 7

3.) Enter a number: 153
153 is an Armstrong number.

4.) Enter a number: 4
The Factorial of 4 is 24

Enter a number: 0
The factorial of 0 is 1

5. Enter
Mul

6.

7.

5. Enter a number: 3.

Multiplication Table for 3:

3 x 1 = 3
3 x 2 = 6
3 x 3 = 9
3 x 4 = 12
3 x 5 = 15
3 x 6 = 18
3 x 7 = 21
3 x 8 = 24
3 x 9 = 27
3 x 10 = 30

6. Enter a string: alphabet

Number of vowels in the string: 3

7. Enter number of elements in the list: 5

Enter element 1: a

Enter element 2: s

Enter element 3: a.

Enter element 4: d

Enter element 5: e.

List after removing duplicates.

8)
Enter number of elements in the list: 3.
Enter element 1: 4
Enter element 2: 6.
Enter element 3: 7
Squared elements: [16.0, 36.0, 49.0]

9.)
Check if a number is Even or Odd.
Enter a number: 8
8 is Even.

10.)
Enter a string: malayalam
Malayam is a palindrome.

31/07/2024

Domain: Sports Analytics - Cricket Player Performance Prediction.

~~Problem Statement~~

Overview:-

Sports analytics is a rapidly growing field that uses data analysis techniques to gain insights and inform decision making in sports. In the context of Cricket prediction is a critical area where data analytics can be applied to forecast the future performance of players. This involves analyzing historical data to identify patterns and trends that can be used to make informed predictions about a player's future performance.

Key Components:

1. Data Collection: Gathering historical data on cricket players, including match performance metrics, physiological data and situational factors.
2. Data Preprocessing: Cleaning and preparing the data for analysis. This includes handling missing values, normalizing values and encoding categorical variables.
3. Feature Selection:- Identifying the most relevent metrics that influence player performance. These can include runs scored, wickets taken, strike rate, batting average, fitness level and more.

4. Modeling and Prediction: Applying machine learning algorithm to predict player performance. Common algorithm include k means clustering for player segmentation, linear regression for predicting performance metrics and Random Forest for robust, non-linear predictions.

Problem statement:

Develop a machine learning model to predict the future performance of cricket players based on historical data and various influencing factors. The goal is to provide actionable insights that assist teams, coaches and analysts in strategic planning and player management.

Target Audience:

Sports Teams and Coaches:

. Professional cricket Teams: They can utilize the performance prediction tool to make data-driven decision about player selection, match strategies and training programs.

Coaches and Support Staffs: Coaches and support staffs can benefit from detailed insights into player performance trends.

Talent Scouts
young
data an

Talent Scouts:- Scouts can identify promising young players by analyzing performance data and predicting future potential.

NQueens Problem

Aim:- To implement NQueens Problem using Python.

Code:-

```
def printSolution(board):
    for i in range(len(board)):
        for j in range(len(board)
            print(board[i][j], end=' ')
        print().

def isSafe(board, row, col):
    for i in range(col):
        if board[row][i] == 1:
            return False

    for i,j in zip(range(row,-1,-1), range(col,-1,-1)):
        if board[i][j] == 1:
            return False
    for i,j in zip(range(row,len(board),1), range(col,-1,-1)):
        if board[i][j] = 1
            return False
    return True

def solveNQutil(board, col):
    if col >= len(board):
        return True.

    for i in range(len(board)):
        if isSafe(board, i, col):
            board[i][col] = 1
```

using Python.

:)

,+1):

ard, 1), range (col, -1, -1)):

```
if SolveNQUtil(board, col+1):
    return True
    board[i][col] = 0

return False

def SolveNQ():
    N = int(input("Enter the size of the board(N): "))

    board = [[0 for _ in range(N)] for _ in range(N)]

if not SolveNQUtil(board, 0):
    print("Solution does not exist")
    return False

print Solution(board)
return True

SolveNQ()
```

Output: Enter the size of the board(N): 6

```
0 0 0 1 0 0
1 0 0 0 0 0
0 0 0 0 1 0
0 1 0 0 0 0
0 0 0 0 0 1
0 0 1 0 0 0
```

True

Depth-First Search

Aim: To implement DFS using Python

Code:

```
From collections import defaultdict

class Graph:
    def __init__(self):
        self.graph = defaultdict(list)
        self.vertices = set()
    def add_edge(self, u, v):
        self.graph[u].append(v)

    def dfs_util(self, v, visited):
        visited[v] = True
        print(v, end=' ')

    for neighbor in self.graph[v]:
        if not visited[neighbor]:
            self.dfs_util(~~start~~ neighbor, visited)

def dfs(self, start_vertex):
    visited = ~~defaultdict(bool)~~ {vertex: False for vertex in self.vertices}
    self.dfs_util(start_vertex, visted)
```

```
if __name__ == "__main__":
    g = Graph()
    num_vertices = int(input("Enter the number of vertices: "))
    print("Enter the vertices:")
    for _ in range(num_vertices):
        vertex = input()
        g.vertices.add(vertex)

    num_edges = int(input("Enter the number of edges: "))
    print("Enter each edge in the format 'u v':")
    for _ in range(num_edges):
        u, v = input().split()
        g.add_edge(u, v)

    start_vertex = input("Enter the starting vertex for DFS:")

    if start_vertex not in g.vertices:
        print(f"Error: The vertex {start_vertex} is not in the list of vertices.")
    else:
        print(f"Depth First Traversal starting from vertex {start_vertex}:")
        g.DFS(start_vertex)
```

vertex in
f.vertices}

Output:-

```
Enter the number of vertices: 4
Enter the vertices:
1
2
3
4
Enter the number of edges: 5
Enter each edge in the format 'u v':
1 2
1 3
2 3
3 4
2 4
Enter the starting vertex for DFS: 1
Depth First Traversal starting from vertex 1:
1 2 3 4
```

Result:-

Thus the above program for Depth First Search is completed successfully.

A* Algorithm

~~Program~~ :- Aim :- ~~import heapq~~ To implement A* Algorithm

Prog :-

```
import heapq to
def heuristic (a, b)
        return abs (a[0] - b[0]) + abs (a[1] - b[1])
def astar (grid, start, end):
    open_list = []
    heapq.heappush (open_list, (heuristic (start, end),
                                0, start))
    g_cost = {start: 0}
    came_from = {}
    directions = [(0, 1), (1, 0), (0, -1), (-1, 0)]
    while open_list:
        current_g, current = heapq.heappop (open_list)
        if current == end:
            path = []
            while current in came_from:
                path.append (current)
                current = came_from [current]
            path.append (start)
            return path [::-1]
        for dx, dy in directions:
            neighbor = (current [0] + dx, current [1]
                                          + dy)
```

```
if (0<=neighbor[0] < len(grid) and 0<=neighbor[1]
    < len(grid[0]) and grid[neighbor[0]][neighbor
    [1]]
    ==0)
    tentative_g = current_g + 1
    if neighbor not in g_cost or tentative_g <
    g_cost[neighbor]
    g_cost[neighbor] = tentative_g
    f_cost = tentative_g + heuristic(neighbor, end)
    heapq.heappush(open_list, (f_cost, tentative_g,
                               neighbor))
    came_from[neighbor] = current
return none.

def main():
    grid = [[0,0,0,0,0], [0,1,1,1,0], [0,0,0,1,0], [0,1,0,0,0],
            [0,0,0,0,0]]
Start = tuple(map(int, input("Enter start node: ").split()))
end = tuple(map(int, input("Enter end node: ").split()))
    Path = astar(grid, start, end)
    if path:
        print("Path found:", path)
    else:
        print("No path found")
if __name__ == "__main__"
    main()
```

output :

Ent

Ent

Path

P

output :

Enter start node: 1 2

Enter End node: 3 4

Path found: [(1,2), (2,2), (3,2), (3,3), (3,4)]

Result:

Thus the above code has been executed successfully

AO* Algorithm

Aim:

To implement AO* Algorithm.

Code:

```
import heapq
def heuristic(a,b):
    return abs(a[0]-b[0])+abs(a[1]-b[1])

def ao_star(grid, start, goals):
    directions=[(0,1),(1,0),(0,-1),(-1,0)]
    open_list=[(0+min(heuristic(start,goal) for goal in
                goals),0,start)]
    g_cost,came_from={start:0},{}
    while open_list:
        _,current_g,current=heapq.heappop(open_list)
    if any(current==goal for goal in goals):
        path=[]
    while current in came_from:
        path.append(current)
        current=came_from[current]
    return path+[start][::-1]

for dx,dy in directions:
    neighbour=(current[0]+dx,current[1]+dy)
    if 0<=neighbour[0]<len(grid) and
       0<=neighbour[1]<len(grid[0]) and
    grid[neighbour[0]][neighbour[1]]==0
```

```
        tentative_g = current_g + 1
        if neighbour not in g_cost or tentative_g < g_cost[neighbour]:
            g_cost[neighbour] = tentative_g
            f_cost = tentative_g + min(heuristic(neighbour, g_cost)
            heapq.heappush(open_list, (f_cost, tentative_g, neighbour))
            came_from[neighbour] = current
    return none

def get_position(prompt):
    while True:
        try:
            x, y = map(int, input(prompt).split())
            return (x, y)
        except ValueError:
            print("Invalid")

def main():
    grid = [[0,0,0,0,0], [0,1,1,1,0], [0,0,0,1,0], [0,1,0,0,0],
            [0,0,0,0,0]]
    start = get_position("Enter the starting position
                          (x,y): ")
    goal = get_position("Enter the goal position (x,y):")
```

```
path = a0_star(grid, start[grid])
if path:
    print("path found:", path)
else: print("No path found")
if __name__ == "__main__"
    main().
```

Output:-

Enter starting position: 1 1

Enter the goal position: 3 4

Path found: [(3, 4), (3, 3), (3, 2) (2, 2), (2, 1), (1, 1)]

Result:- Thus the above code has been ~~veri~~ executed successfully and output is verified

Implementation of Decision Tree Classification Technique

Aim:-
To implement a decision tree classification technique for gender classification using python

Algorithm

- Import tree from sklearn.
- Call the function DecisionTreeClassifier() from tree.
- Assign values for x and y.
- Call the function predict for the Predicting on the basis of given random values for each given feature.
- Display the output.

Code:-

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

data={
    'Height':[152,155,172,185,167,180,157,180,164,177],
    'weight':[45,57,72,85,88,78,22,90,66,88],
    'Gender':['Female','Female','Male','Male','Female',
              'male','Female','male','Female',Male]
    }
```

```
df = pd.DataFrame(data)
X = df[['Height', 'Weight']]
y = df['Gender']

classifier = DecesionTreeClassifier()
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(F"Predicted gender for height {height} cm and
      weight {weight} kg: {predicted_gender[0]}")
```

Output:

Enter height (in cm) for prediction: 165

Enter weight (in kg) for prediction: 63

predicted gender for height 165.0 cm and weight 63.0 kg : Female.

Result: Thus the above experiments executed successfully.

Aim: To

using

Algorithm:

Code:

Implementation of clustering Techniques using kMeans.

Aim:- To implement a kMeans clustering technique using python language.

Algorithm:-

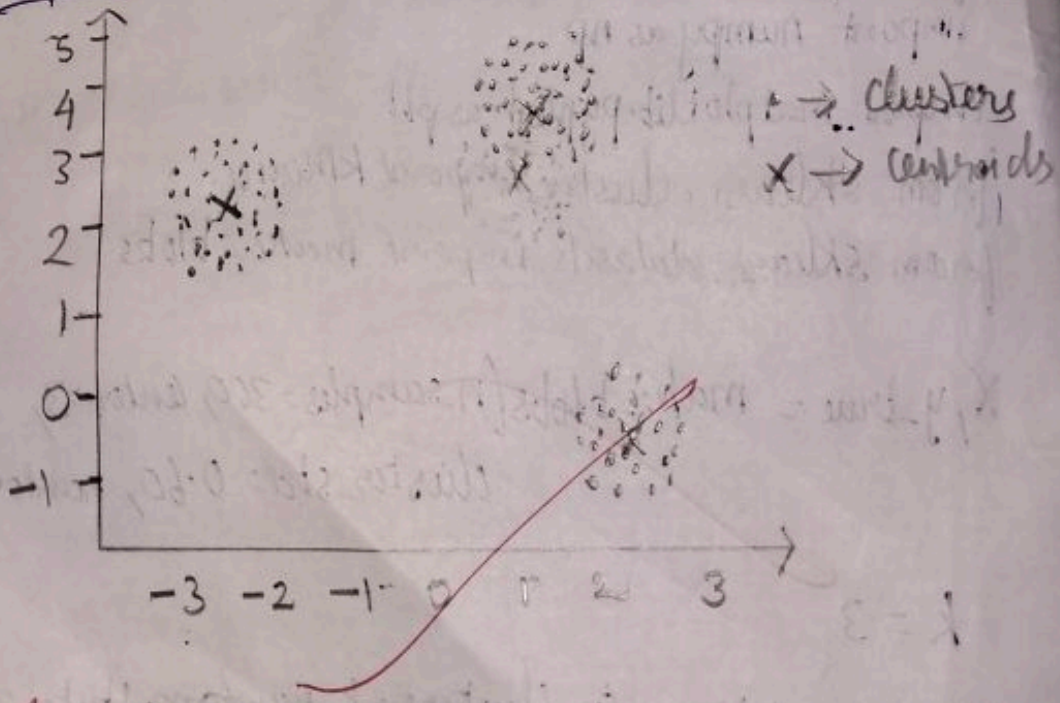
- Import kMeans from Sklearn.cluster
- Assign X and y.
- Call the function kMeans().
- Perform scatter operation and display the output

Code:-

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import kMeans.
from sklearn.datasets import make_blobs.

X, y_true = make_blobs(n_samples=300, centers=3,
                       cluster_std=0.60, random_state=0)

k=3
kmeans = kMeans(n_clusters=k, random_state=0)
y_kmeans= kMeans.fit_predict(X)

plt.figure(figsize=(8,6))
```

```
plt.scatter(X[:,0], X[:,1], c=y_kmeans, s=30,
            cmap='viridis', label='Clusters')
centers = kmeans.cluster_centers_
plt.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 1')
plt.ylabel('Feature 2')
plt.legend()
plt.show()
```

Output:-



Result:- Thus the above experiment for K means Clustering has executed successfully.

Implementing Artificial Neural Networks for an Application using python - Regression

Aim:- To implementing artificial neural networks for an application in Regression using python.

Algorithm:-

1. Generate and split synthetic data, scale features.
2. Build a sequential ANN with 2 hiddenlayers.
3. Compile with adam optimizer and MSE loss
4. Train the model and evaluate on test data
5. Plot training and validation loss curves

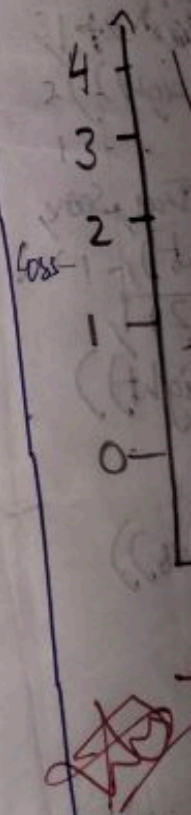
Code:-

```
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
np.random.seed(42)
X = np.random.rand(1000, 3)
y = 3*X[:,0] + 2*X[:,1]**2 + 1.5*np.sin(X[:,2]*np.pi)
    + np.random.normal(0, 0.1, 1000)
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, t
                                                    random
                                                    =2
Scales = StandardScales()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
model = Sequential()
model.add(Dense(10, input_dim=X_train.shape[
                activation='relu'))
model.add(Dense(10, activation='linear'))
model.compile(optimizer=Adam(learning_rate=0.01
              loss='mean_squared_er
history = model.fit(X_train, y_train, epochs=1
                    batch_size=32,
                    validation_split=0.2, verbose=
y_pred = model.predict(X_test)
mse = np.mean(y_test - y_pred.flatten())**2
print(f'Mean Squared Error: {mse:.4f}')
plt.figure(figsize=(12,6))
plt.plot(history.history['loss'], label='Training
plt.plot(history.history['val_loss'], label
                                    ='Valida
                                    Loss')
```

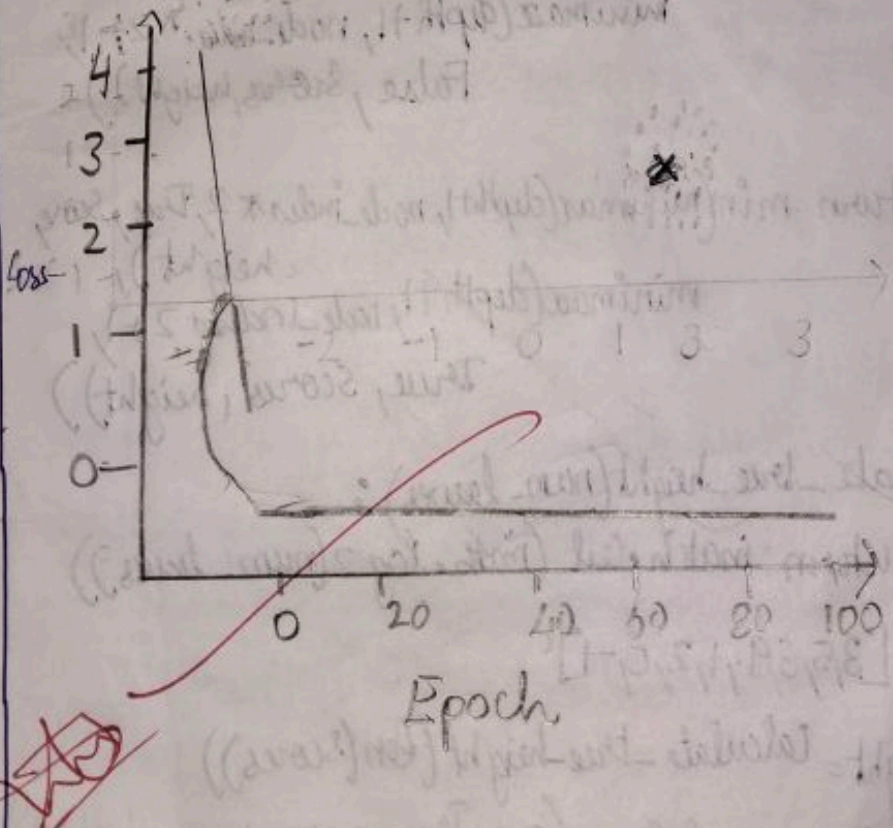
```
plt.title('T
plt.xlabel(
plt.ylabel
plt.legen
plt.show
```

Output :



```
plt.title('Training and validation Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()
plt.show()
```

Output :- Epoch 1/100
20/20
⋮
Epoch 100/100
20/20



Result :- Thus the above Experiment for ANN has been sucessfully executed.

MINIMAX ALGORITHM

Aim:- To implement minimax algorithm.

Program:-

```
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}")
```

Output:-

The opti

Resu

Output:-

The optimal score is : 5

Result :-

Thus the program to implement minimax algorithm is successfully executed.

Aim:-
To learn Prolog terminologies and write basic programs.

Terminologies:-

1.) Atomic terms:
They are usually strings made up of lower and uppercase letters, digits and the underscore, starting with a lowercase letters.
eg:- dog, abc_321

2.) variables:-
They are strings of letters, digits and the underscore, starting with a capital letters or an underscore.
eg:- Dog, Apple_420

3.) Compound terms:
Compound terms are made up of a Prolog atom and a number of arguements enclosed in parathesis and separated by commas.
eg:- is_bigger(elephant, X). f(g(X, _), 7)

4.) Fact:-
A fact is a predicate followed by a dot.
eg:- bigger_animal(whale). life_is_beautiful.

5.) Rules:- A rule consists of a head and body
eg:- is_smaller(X, Y) :- bigger(Y, X); (aunt(aunt, child)

Source code:-

KB1

```
woman(mia).
woman(jody)
woman(yolanda)
playsairguitar(jody)
party.
Query 1 : ?- woman(mia)
Query 2 : ?- playsAirguitar(mia)
Query 3 : ?- party.
Query 4 : ?- concert.
```

O-utput :-

```
?- woman(mia)
   true.
?- playsairguitar(mia)
   false
?- concert
error = unknown procedure concert/0
```

KB2 :-

```
happy(yolanda).
listens2music(mia).
Listens2music(yolanda) :- happy(yolanda)
playsAirGuitar(mia) :- listens2music(mia).
playsAirGuitar(yolanda) :- listens2music(yolanda).
```

Output:-

```
?- playsAirGuitar(mia).
true
?- playsAirGuitar(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)
```

O/P:-

```
?- likes(dan, X).
X = sally
?- married(dan, sally)
true
?- married(john, brittney)
false
```

KB4:-

```
food(burger)
food(sandwich)
food(pizza)
lunch(sandwich)
dinner(pizza)
meal(X) - food(X).
```

O/P:-

?- food(pizza)

true.

?- meal(X), lunch(X)

X = sandwich

?- dinner(sandwich)

false.

?-

KB-5:-

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)).

O/P:-

?- 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).

false X = car(chevy)

Result:- Thus the experiment of prolog implementation is executed successfully

Prolog family tree

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, john).
Parent of (jeny, helen)
```

/* RULES */

/* Son, Parent */

/* Son, grandparent */

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

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

grandfather(X,Y) :- male(Y), parent of(X,Z),
parent(Z,Y).

grandmother(X,Y) :- female(Y), parent(X,Z), parent(Z,Y)

brother(X,Y) :- male(X), father(X,Z),
father(Y,W), Z == W.

sister(X,Y) :- female(X), father(X,Z), father(Y,W),
Z == W.

Output :-

male (Peter)
true
father (Chris, peter)
true
father (chris, betty)
false
mother(Chris, X)
X = betty
brother(Chris, helen)
false

Result :- Thus prolog for family tree program has been executed successfully.