INDEX

மரம் வளர்ப்போம்
மண் வளம் காப்போம்.

NEW MAHADEV
TAKE EVERY OPPORTUNITY HANDED TO YOU

Name : CHANDRU

Subject : POAI

Std : Sec : Roll No 220701051

School :

S.No.	Date	Title	~~Page No.~~ Marks	Teacher's Sign / Remarks
1.	9/8/24	8 Queens problem	9	
2	16/8/24	DFS	9	
3	23/8/24	DFS- water Jug Problem	9	
4	30/8/24	A* Search Algorithm	10	
5	6/9/24	implementation of Decision tree Classification technique	10	
6.	13/9/24	implementation of ANN for using Python regression	10	
7.	4/10/24	implementation of clustering technique	10	
8.	18/10/24	Minax Algorithm	10	
9.	25/10/24	introduction to prolog	10	
10.	8/11/24	Prolog family tree	10	

completed

```
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):
 if y==0
    return "Error: Division by zero"
 else
   return x/y

Print ("Select operation:")
Print ("1. Add")
Print ("2. Subtract")
Print ("3. Multiply")
Print ("4. divide")

Choice = input ("enter choice (1/2/3/4):")
num = float (input ("Enter first number:"))
num2 = float (input ("Enter second number:"))

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))

else

Print ("invalid input")

OUTPUT

1. Add
2. Subtract
3. Multiply
4. Divide

Enter choice (1/2/3/4): 1

Enter 1st number: 30

Enter 2nd number: 50

30.0 + 50.0 = 80.0

```
def Multiply (Mylist):
    result = 1
    for x in mylist:
        result = result * x
    return result

list 1 = [2, 2, 3]
list 2 = [3, 3, 4]
Print (multiplylist (list 1))
Print (multiplylist (list 2))
```

OUTPUT

12

36

(3)

```
list = ['iris', 'orchids', 'rose', 'lavender', 'lily', 'carnations']
print("original list is:", list)
list.remove('orchids')
print("After deleting 2nd item:", list)
```

output

original list is: ['iris', 'orchids', 'rose', 'lavender', 'lily', 'carnations']

After deleting 2nd items: ['iris', 'rose', 'lavender', 'lily', 'carnations']

(4)

```
list_1 = [1, 4, 7, 5, 5]
list_4 = [3, 7, 5, 2, 5]
list_3 = list_1 + list_4
print("concatenated list using :" + str(list_3))
```

output

concatenated list using : [1, 4, 5, 5, 5, 3, 5, 5, 2, 5]

(5)

```
list 1 = [10, 20, 20, 4, 45, 45, 99, 99]
list2 = list (set (list 1))
list2.sort()
Print ("second largest element is :", list2[-2])
```

Output

second largest element is

(6)

```
str = 'char, hi'
Print("the original string")
x = set (str)
r = []
for in x:
    if (str.count(i)-1.2!=0)
        r.append
Print ("the odd frequency character
```

Output:-

The original string is: char, hi

The odd frequency characters are:

['i', 'r', 'a', 'h']

JOB RECOMMENDATION USING MACHINE LEARNING

DOMAIN:- CARRER TECHNOLOGY

ABSTRACT:-

A Job Recommendation System use machine Learning to Match Job Seekers with Suitable job opportunities By analysing profiles and job listing, the System provides Personalized job Suggestion. It employs Techniques like Content based and Collaborative filtering and use natural language processing to understand job description and resumes the System aim to enhance job Seeker's Experience and improve recruitment efficiency by delivering relevant job match and adapting to user feedback.

Problem Statement :-

Develope a machine learning- based job recommendation System to match job Seekers with suitable job opportunities. the System should analyse user Profiles and job listing to deliver personalized recommendations By Utilizing algorithms like Content-based and Collaborative filtering, and employing natural language processing, the System aim to enhance the job Search experience and improve recruitment efficiency. It should continuously adapt to user feedback and evolving data to provide relevant and timely job Suggestion

SOLUTION:-

Implement a machine learning job recommendation system that analyze job seeks profiles and job listing. Use content-based and collaborative filtering to Deliver Personalized job suggestions. Employ natural language processing for better understanding and matching. Continuously update recommendation based on user feedback and interactions.

TARGET AUDIENCE

1. JOB SEEKERS: People actively searching for job.
2. Employers / Recruiters: Company looking to hire the right candidate
3. Student / graduate:- New entrants to the job market seeking entry-level position.
4. Career changers:- professional transforming to new industries or role
5. Freelance / gig workers: Individuals seeking short term or freelance opportunities
6. Mid-Career professionals: Experienced professionals looking for advancement
7. Tech-savvy users: User who accept advanced, personalized recommendation

Objectives:-

+ Match:- Accurately match job seekers with job opportunities
+ Personalize:- Tailor recommendation based on individual skills, experience and preferences
+ Enhance Efficiency:- Improve the job search process for users
+ Increase Satisfaction:- Provide relevant job suggestions to improve user satisfaction

Algorithm used:-

Collaborative filtering:- Recommends jobs based on user similarity and past interactions.
Content-based filtering:- Suggest job seekers with user profile. Matrix factorization:- Decomposes user-job interaction data to identify latent factors and preferences. Natural Language:- Use deep learning to captures complex patterns in job and user data for accurate recommendation

Dataset:-

→ Yelp dataset Challenge

6/9/24

N Queens Problem

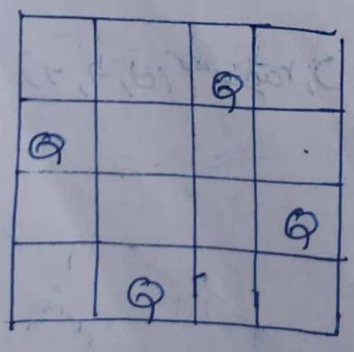
Aim: write a Python program for N Queen problem

Program:

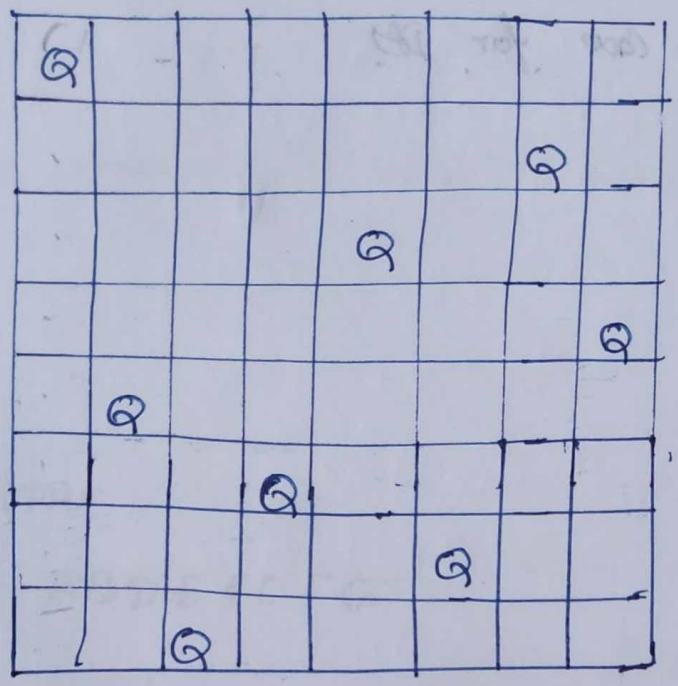
```
Def print_Solution (board):
N= len (board)
for i in range (N):
row = ['Q' if board [i][j] else '.' for j in range (N)]

Print (" ".join (row))
Print()
Def is_safe (board, row, col, N):
for i in range (col):
if board [row][i]:
return false

for i, j in zip (range (row, -1, -1) range (-1, -1)
range (col, -1, -1)):
if board [i][j]:
return false

for i, j in zip (range (row, N, 1), range (col, -1, -1)
if board [i][j]:
return false

return True
```

```
def Solve_nqueens_util(board, col, N):
    if col > N:
        Print_Solution(board)
        return True

    res = False
    for i in range(N):
        if is_Safe(board, i, col, N):
            board[i][col] = True
            res = True
    return res

def Solve_n_Queens(N):
    board = [[False] * N for _ in range(N)]
    Print(f"Solution for {N}-Queens problem:")
    Solve_nqueens_util(board, 0, N)

N_values = [4, 8]
for n in N_values:
    Solve_n_Queens(n)
Print("\n" + "=" * 20 + "\n")
```

OUTPUT: Solution for 4 Queens



Solution for 8 Queen



Result :- Thus the above python code executed Successfully

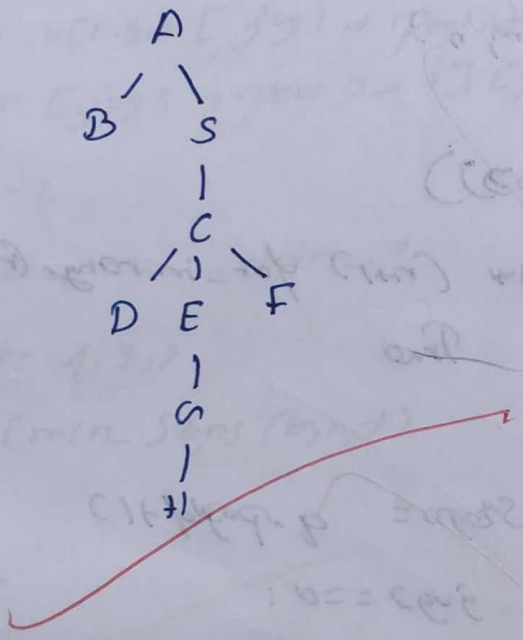
Depth first Search (graph)

Aim: write a Python code for DFS

```
graph = {
  'A' = ['B', 'C']
  'B' : ['A', 'D', 'F'],
  'C' : ['A', 'F', 'G'],
  'D' : ['B']
  'E' : ['B'].
  'F' : ['C'].
  'W' : ['J']
}

Def dfs (graph, start, visited = NULL):
  if visited is none:
  visited.add (start)
  Print (start, end = ' ')
  for (start, end = ' ')
  for neighbour in graph [start]:
  if neighbor not in visited:
  DFS (graph, neighbor, visited)
  dfs (graph, 'A')
```

OUTPUT

A B D E F C f G



Result :-

Thus the above Python code for DFS executed Successfully..

13/9/24 Water Jug Problem using BFS

- Aim:-

To write a Python code water Jug problem using BFS

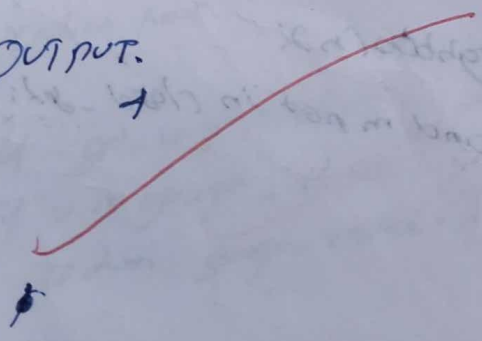
Program:-

```
from collection import Deque
Def. min_steps (m, n, d):
if d > max (m, n):
return -1

q = deque [[(0, 0, 0)]]

visited = [[False] * (n+1) for _ in range (m+1)]
visited [0][0] = True

while q:
    jug1, jug2, steps = q.popleft()
if jug1 == d or jug2 == d:
return steps

if not visited [m][jug2]:
visited[m][jug2] = True
q.append ((m, jug2, steps+1))

if not visited [jug1][n]:
visited [jug1][n] = True
q.append ((jug1, n, steps+1))
```

```
    if not visited[0][y_jug2]:
        visited[0][y_jug2] = True
        q.append((0, y_jug2, steps+1))

    if not visited[y_jug1][0]:
        visited[y_jug1][0] = True
        q.append((y_jug1, 0, steps+1))

    pour1to2 = min(y_jug1, n - y_jug2)
    if not visited[y_jug1 - pour1to2][y_jug2 + pour1to2]:
        visited[y_jug1 - pour1to2][y_jug2 + pour1to2] = True
        q.append((y_jug1 - pour1to2, y_jug2 + pour1to2, steps+1))

    pour2to1 = min(y_jug2, m - y_jug1)
    if not visited[y_jug1 + pour2to1][y_jug2 - pour2to1]:
        visited[y_jug1 + pour2to1][y_jug2 - pour2to1] = True
        q.append((y_jug1 + pour2to1, y_jug2 - pour2to1, steps+1))

return -1

if __name__ == "__main__":
    m, n, d = 4, 3, 2
    print(min_steps(m, n, d))
```

OUTPUT.

1



Result:-

Thus the above Python code for water Jug Problem was executed Successfully water Jug Successfully.

A* Algorithm

Aim:-

To write a python code for A* Algorithm.

Program:-

```
Def astar (start_node, stop_node):
open_set = set (start_node)
closed_set = set()
g = {}
parents [start_node] = start_node
while len (open_set) > 0:
n = none
for v in open_set:
if n == none or g[v] + heuristic(v) < g[n] +
heuristic(n):
n = v
if n == stop_node or graph_nodes [n] == none:
pass
for (m, weight) in get_neighbors(n):
if m not in open_set and m not in closed_set:
open_set.add(m)
parents [m] = n
g[m] = g[n] + weight
else:
if g[m] > g[n] + weight:
g[m] = g[n] + weight
parents [m] = n
```

```
If m in closed_set:
    closed_set.remove(m)
    open_set.add(m)

if n = None:
    print("path does not exist!")
    return None

if n == stop_node:
    path=[]

while parents[n] != n:
    path.append(n)
    n = parents[n]

path.append(stop_node)
path.reverse()
print("path found:{}".format(path))
return

    open_set.remove(n)
    closed_set.add(n)

print("path does not exist!")
return None

def get_neighbors(v):
if v in graph_nodes:
    return graph_nodes[v]

else:
    return None

def heuristic(n)
    H_dist={
        'A': 11,    'G': 0,
        'B': 6,
        'C': 99,
        'D': 1,
        'E': 7,
```

```
"3
Kodim Hadiya(n)

graph_nodes={
  'A': [('B',2),('E',3)],
  'B': [('C',1),('G',9)],
  'C': None,
  'E': [('D',6)],
  'D': [('G',1)],
}
astar('A','G')
```

Path found:

Output:

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

Result:-

Thus the Python program for A* Algorithm was executed Successfully

IMPLEMENTATION of DECISION TREE CLASSIFICATION TECHNIQUES

Aim:

To implement a Decision tree classification technique for gender classification using Python

Explanation:-

- Import tree from sklearn
- Call the function DecisionTreeClassifier from tree
- assign value for x and y
- Call the function Predict for Predicting on the basis of given random values for each given features
- Display the output

CODE:-

```
import pandas as pd
from sklearn.tree import DecisionTreeClassifier
Data={
'Height': [150, 160, 170, 180, 165, 175, 155, 185, 62, 172]
'weight': [50, 60, 70, 80, 65, 75, 55, 85, 68, 78]
'gender': ['female', 'female', 'male', 'male']
['female', 'male', 'female', 'male', 'female', 'male']
3
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}
in cm and weight {weight} kg: {Predicted_
gender[0]}")
```

Input + Output

Enter height (in cm): 150
Enter weight (in kg): 50
Predicted gender for height 150.0 cm and weight
50.0 kg: female

Result:-

Thus the implementation of Decision Tree classification technique is Done and executed successfully.

7. Implementation of clustering Techniques

K-means

Aim:-

To implement a K-means clustering techniques using Python language

Explanation

- Import K-means from Sklearn.cluster
- Assign x and y
- Call the function KM(means)
- Perform Scatter operation and Display the output

CODE:-

```
import numpy as np
import matplotlib.pyplot as plt
num_points = int(input("Enter the number of Data points:"))
X = np.zeros((num_points, 2))
for i in range(num_points):
    x = float(input("Enter x-coordinate for Data point {i+1}:"))
    y = float(input("Enter y-coordinate for Data point {i+1}:"))
    X[i] = [x, y]
num_clusters = int(input("Enter the number of clusters:"))
```

```
Def kmeans(x, num_clusters, max_iters=100):
    centroids = x[np.random.choice(x.shape[0],
                num_clusters, replace=False)]
    for i in range(max_iters):
        Distance = np.linalg.norm(x[:,np.newaxis]
                   - centroids, axis=2)
        labels = np.argmin(Distance, axis=1)
        new_centroids = np.array([x[labels==k]
        .mean(axis=0) for k in range
                                (num_clusters)])
        if np.all(centroids == new_centroids):
            break
        centroids = new_centroids
    return labels, centroids
labels, centroids = kmeans(X, num_clusters)
print("Cluster labels:\n", labels)
print("Centroids:\n", centroids)
plt.figure(figsize=(8,6))
plt.scatter(X[:,0], X[:,1], c=labels, cmap='viridis',
            marker='o', label='Data points')
plt.scatter(centroids[:,0], centroids[:,1],
plt.title('K-means clustering from scratch')
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
plt.legend()
plt.grid(True)
plt.show()
```

7.

Aim:-

To implement artificial neural network for an application in Regression using Python

Algorithm:-

-> generate Data: Create Simple Data with a linear relationship

-> Prepare Data: - Scale the input and output values

-> Split Data: Divide the Data into training and testing sets

-> Build model: create an ANN with input, hidden and output layers

-> Train model: fit the model to the training Data over several epochs

-> Evaluate model: Test the Model using the testing set

-> Visual Result: Plot actual vs predicted values to visualize performance

Code:-

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from tensorflow import keras
from tensorflow.keras import layers
np.random.seed(0)
```

```
y = 4 * X + np.random.rand(1000,1)
X_scaler = StandardScaler()
y_scaler = StandardScaler()
X_scaled = X_scaler.fit_transform(X)
y_scaled = y_scaler.fit_transform(y)
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y_scaled, test_size=0.2, random_state=42)

model = keras.Sequential([
    layers.Dense(64, activation='relu', input_shape=(X_train.shape[1],)),
    layers.Dense(64, activation='relu'),
    layers.Dense(1)
])
model.fit(X_train, y_train, epochs=100, batch_size=32, verbose=0)
loss = model.evaluate(X_test, y_test)
print(f'Test loss: {loss}')
y_pred_scaled = model.predict(X_test)
y_pred = y_scaler.inverse_transform(y_pred_scaled)

plt.figure(figsize=(10,6))
plt.scatter(X_test, y_scaler.inverse_transform(y_test), label='Actual Data', color='blue', alpha=0.6)
plt.scatter(X_test, y_pred, label='Predictions', color='red', alpha=0.6)
```

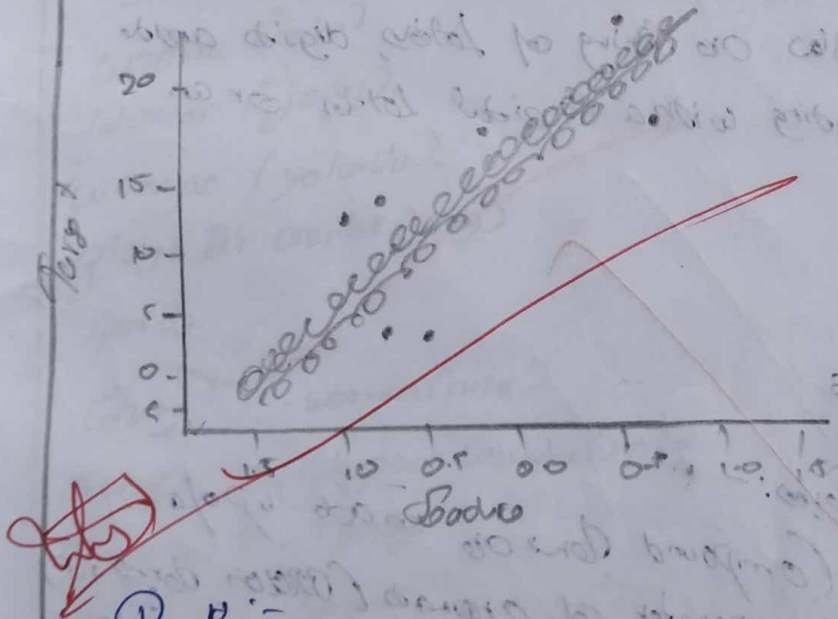
plt.title("Actual vs Predicted")
plt.xlabel('x')
plt.ylabel('y')
plt.legend()
plt.show()

	Feature	Target
0	3.740401	7.806204
1	9.501143	16.343597
2	7.319939	15.400258
3	5.986555	13.194391
4	1.560186	4.239934

Epoch 1/100
20/20 ——— 2s 10ms/step loss: 4.9368

⋮

Epoch 100/100
20/20 ——— 0s 4ms/step – loss: 3.4·2248
Test loss (mse): 3.54[illegible]2305297855



Result:-
Thus the implementation of neural networks for an application in Regression using Pytorch is done and executed successfully

Exno:

Date:-

Introduction To Prolog

Aim:-

To Learn Prolog terminologies and write basic Progs

Terminologies

1. Atomic Terms:-

Atomic terms are usually strings made up of lower- and uppercase letters, Digits, and the underscore, starting with a lowercase letter

Ex:-

dog
ab_c_321

2. Variables:-

Variables are strings of letters, digits and the underscore, starting with a capital letter or an underscore

Ex:-

Dog
Apple-426

3. Compound Terms:-

Compound terms are made up of a Prolog atom and a number of arguments (Prolog terms i.e., atoms, numbers, variables, or other compound terms) enclosed in parentheses and separated by commas

Ex:-
is bigger (elephant, X)
f(g(X,-7),7)

4. Facts:-
A facts is a predicate followed by a Dot

Ex:-
bigger - animal (whale)
life - is beautiful.

5. Rules:-
A rule consists of a head (a predicate) and a body (a sequence of predicate separated by commas).

Ex:-
is - Smaller (X,Y) is - bigger (X,Y)
and (Edward, child) :- Sister (Edward, Bobby), parent (grandma, child)

Source Code:-

```
woman (mia).
woman (jody).
woman (yolando).
playsAirGuitar (jody)
party.
```

Query 1: ? - woman (mia)
Query 2: ? - playsairGuitar (mia).
Query 3: ? - party
Query 4: ? - concert

Output :-

?- woman(mia).

true

?- playsAirGuitar(mia).

false

?- concat.

ERROR: unknown procedure: concat/0 (D.Wim could not form goal)

?-

KB1 :

happy(yolanda).

listens2music(mia).

listens2music(yolanda) :- happy(yolanda).

playsAirGuitar(mia) :- listens2music(mia).

playsAirGuitar(yolanda) :- listens2music(yolanda).

Output

Query1 :- ?- playsAirGuitar(mia)

true

Query2 :- playsAirGuitar(yolanda)

true.

KB3

married(mia, sally).

likes(sally, dan).

likes(john, britney).

married(X,Y) :- likes(X,Y) ; likes(Y,X).

friends(X,Y) :- likes(X,Y) ; likes(Y,X).

Output

Query1:- ?- like(dan,x)
X = sally
Query2:- married(dan, Sally).
true
Query2: ?- married(john, bindaas)

KB4:

food(burger)
food(sandwich)
food(pizza)
lunch(sandwich)
dinner(pizza).
meal(X): food(X).

Output

Query1. ?- food(pizza)
true
Query2: ?- meal(X), lunch(X)
X = sandwich
Query3: ?- 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

Query: -? - owns (John, x)

X = car (Clay)

Query 2 ?. owns (John, _)

true

Query 3 : ? owns (who, car (Clay))

who = John

Query 4 : owns (jane, x), sedan (x)

false

Query 5. ? owns (jane, x), drive (x)

X = car (Clay)

Result :-

Thus to learn Prolog terminologies & write a Program is done & executed Successfully

PROLOG family tree

Exno: 10 Date 08-11-2024

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(jenny).
female(user).
female(ellen)

Parent of (chris, peter)
Parent of (chris, betty)
Parent of (ellen, peter)
Parent of (ellen, betty)
Parent of (kevin, chris)
Parent of (kevin, lisa)
Parent of (jeny, John)

/*RULES.*/
/* Son, parent.
   Son. grandparent.*/
father(x,y):- male(y), parent of (x,y)
```

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

OUTPUT

?- male (y), parent of (x,y)

y	x
Peter	Chris
Peter	Helen
john	Jeny
Chris	Kevin

? female (y), parent of (x,y)

y	x
Betty	Chris
Betty	Helen
lisa	Kevin
Helen	jeny

? male (y), parent of (x,z) parent of (z,y)

y	x	z
Peter	Kevin	Chris
Peter	jeny	Helen

? female (y), parent (y,z) parent of (z,y)

Y	X	Z
daddy	Kevin	Chris
daddy	Jeny	Robin

?- male (y), father (x,z), father (y,w), z == w
Procedure father (A,B) Does not exist

? -female (y), father (x,z), father (y,w), z==w
Procedure 'father (A,B) does not exist

Result:

Result:-
Thus we have Developed a family tree program using PROLOG with all possible facts, rules and queries