INDEX

NAME: AJAY KRISHNAN STD.: CLG SEC.: A ROLL NO.: 18 SUB.: POAI OBS

S. No.	Date	Title	~~Page No.~~ marks	Teacher's Sign / Remarks
1.	31/07/2024	SAMPLE PROGRAMS IN PYTHON USING GOOGLE COLLAB	~~1~~ 9	
2.	~~2~~ 10/8/24	PROJECT IDEA	9	
3.	4/9/24	n-Queens Problem	9	
4.	~~18~~ 11/9/24	DFS search	10	
5.	11/9/24	A* Algorithm	10	
6.	18/9/24	Ao* Algorithm	10	
~~7.~~	25/9/24	Decision tree	10	
8.	9/10/24	K-means	10	
9.	16/10/24	Artificial Neural Network	10	
10.	23/10/24	Minimax Algorithm	10	
11.	30/10/24	Intro to Prolog	10	
12.	6/11/24	Prolog - family tree	10	

completed

EX.1: SAMPLE PROGRAMS IN PYTHON USING GOOGLE COLLAB

10 PROGRAMS

1. Program to find factorial of a number

```
import math
n = int(input("Enter the number to find factorial : "))
print("Factorial of the number is : ")
print(math.factorial(n))
```

OUTPUT:

```
Enter the number to find factorial : 78
Factorial of the number is :
113242811788206297831457521158732046228731
```

2. Program to find the largest no.:

```
def maximum(x, y, z):
  print("enter three numbers :", x, y, z)
  if (x >= y and x >= z):
    largest = x
  elif (y >= z and y >= x):
    largest = y
  else:
    largest = z
  print("the largest no. is :", largest)
```

OUTPUT:

```
Enter three numbers : 21, 25, 48
The largest no. is : 48
```

3. Program to find the ~~square~~ quadratic root of the numbers

```
import cmath
print("Enter first number: ",x)
print("Enter second number: ",y)
print("Enter third number: ",z)
w=(y**2)-(4*x*z)
ans1=(-y-cmath.sqrt(w))/(2*x)
ans2=(-y+cmath.sqrt(w))/(2*x)
print("The solution is {0} and {1}".format(ans1,ans2))
```

OUTPUT:

```
Enter first number: 1
Enter second number: 5
Enter third number: 6
the solution is (-3+0j) and (-2+0j)
```

4. Program to check whether it is a leap year or not

```
year=int(input("Enter the year: "))
if(year%4)==0:
    if(year%100)==0:
        print("The given year is a leap year")
    else:
        print("It is not a leap year")
```

OUTPUT:

```
Enter the year: 2020
The given year is a leap yes
```

5.

6.

5. Program to swap two variables

```
var1 = input("Enter the first variable")
var2 = input("Enter the second variable")
temp = var1
var1 = var2
var2 = temp
print("The value of the first variable after swapping is:", var1)
print("The value of the second variable after swapping is:", var2)
```

OUTPUT:

```
Enter the first variable: 18
Enter the second variable: 27
The value of the first variable after swapping is: 27
The value of the second variable after swapping is: 18
```

6. PROGRAM TO FIND SQRT OF A NUMBER:

```
num = float(input('Enter a number: '))
sqrt = num ** 0.5
print("Square root of %0.3f is %0.3f" %(num, sqrt))
```

OUTPUT:

```
Enter a number: 25
Square root of 25.000 is 5.000
```

7. Sum of n numbers:

```
num = int(input("Enter a number: "))
if num<0:
    print("Enter a positive number")
else:
    sum=0
    while(num>0):
        sum+=num
        num-=1
    print("The sum is", sum)
```

OUTPUT:

Enter a number: 10
The sum is: 55

8. To find the LCM of the number:

```
def compute_lcm(x,y)
if x>y:
    greater=x
else
    greater=y
while(True):
    if ((greater%x==0) and (greater%y==0)
        lcm=greater
        break
    greater+=1
return lcm

num1= int(input("Enter a number 1:"))
num2= int(input("Enter a number 2:"));
print("The LCM is", compute_lcm(num1,num2))
```

OUTPUT:

Enter a number: 5
Enter a number 2: 10
The L.C.M is 10.

9. PROGRAM TO FIND PALINDROME:

```
word = input()
if word = word [::-1]:
    print("word, "is a palindrome")
else:
    print(word, "is not a palindrome")
```

OUTPUT:

affle
affle is not a palindrome.

10. PROGRAM TO SORT LIST ALPHABETICALLY:

```
str_1 = input("Enter a string")
words = [word, lower() for word in str_1.split()]
words.sort()
print("The words sorted in alphabetical order are as follows")
for word in words:
    print(word)
```

OUTPUT:

Enter a string: hello
The words sorted in alphabetical order are as follows:
ehllo

17 Merge two sorted array:

```
def Merge(arr1, arr2, n1, n2, arr3)
    i=0
    j=0
    k=0
    while(i<n1):
        arr3[k]=arr1[i]
        k+=1
        i+=1
    while(j<n2):
        arr3[k]=arr2[j]
        k+=1
        j+=1
    arr3.sort()
if arr1=[1,3,5,7]
    n1=len(arr1)
arr2=[2,4,6,8]
n2=len(arr2)
arr3=[0 for i in range(n1+n2)]
Merge(arr1, arr2, n1, n2, arr3)
for i in range(n1+n2):
    print(arr3[i], end=" ")
```

Output:

Arr:
[1 2 3 4 5 6 7 8]

FOODZEE

ABSTRACT:

A city consists of n number of hotels out of which a person maximum chooses 2a top rated hotels in his/her bucket list. It gets complecated when a person uses google for a review of every restaurant one by one. So using this foodzee website leads to combination of every hotels reviews under a single domain. It uses a combination of fuzzy logic & bubble sort which rates the best restaurant out of the lot by arranging it points from high to low.

LOGIC BEHIND THE PROCESS:

→ The person's interest in food, location of the person & rating of the hotels. The 3 notes highlighted the

→ Bubble Sort

Base IEEE xplore papers:

Worthleat: A similar food review app.

EX:3

N-QUEENS PROBLEM

★ AIM:

To solve N-Queens problem and display the output.

★ PROGRAM:

```
def print_solution(board):
    """Function to print the chessboard"""
    N = len(board)
    for i in range(N):
        row = ""
        for j in range(N):
            if board[i][j] == 1:
                row += "Q "
            else:
                row += ". "
        print(row)
    print("\n")

def is_safe(board, row, col):
    """Check if it's safe to place a queen at board[row][col]"""
    N = len(board)

    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, N, 1), range(col-1,
    if board[i][j]==1:
        return False

return True

def solve_n_queens_util(board, col):
    """Utility backtracking to solve the N-Queens
    N = len(board)
    if col >= N:
        return True

    for i in range(N):
        if is_safe(board, i, col):
            board[i][col] = 1

            if solve_n_queens_util(board, col+1):
                return True

            board[i][col] = 0
    return False

def solve_n_queens(N):
    board = [[0] * N for _ in range(N)]

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

    print_solution(board)
    return True

N = 4
solve_n_queens(N)
```

☆ OUTPUT:

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

True

☆ RESULT:

Thus the above program is executed successfully.

EXP.NO:4

DFS

☆ AIM:

To Solve DFS and give the best possible shortest path

☆ PROGRAM:

```
def dfs(graph, start, visited=None):
    if visited is None:
        visited = set()

    visited.add(start)
    print(start)

    for neighbor in graph[start]:
        if neighbor not in visited:
            dfs(graph, neighbor, visited)

    return visited

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

print("DFS starting from node 'A':")
dfs(graph, 'A')
```

★ OUTPUT:

```
DFS starting from node 'A':
A
B
D
C
['A', 'B', 'C', 'D']
```

★ RESULT:

Thus the above program is executed successfully

11/9/24 EXP:5

A* Algorithm

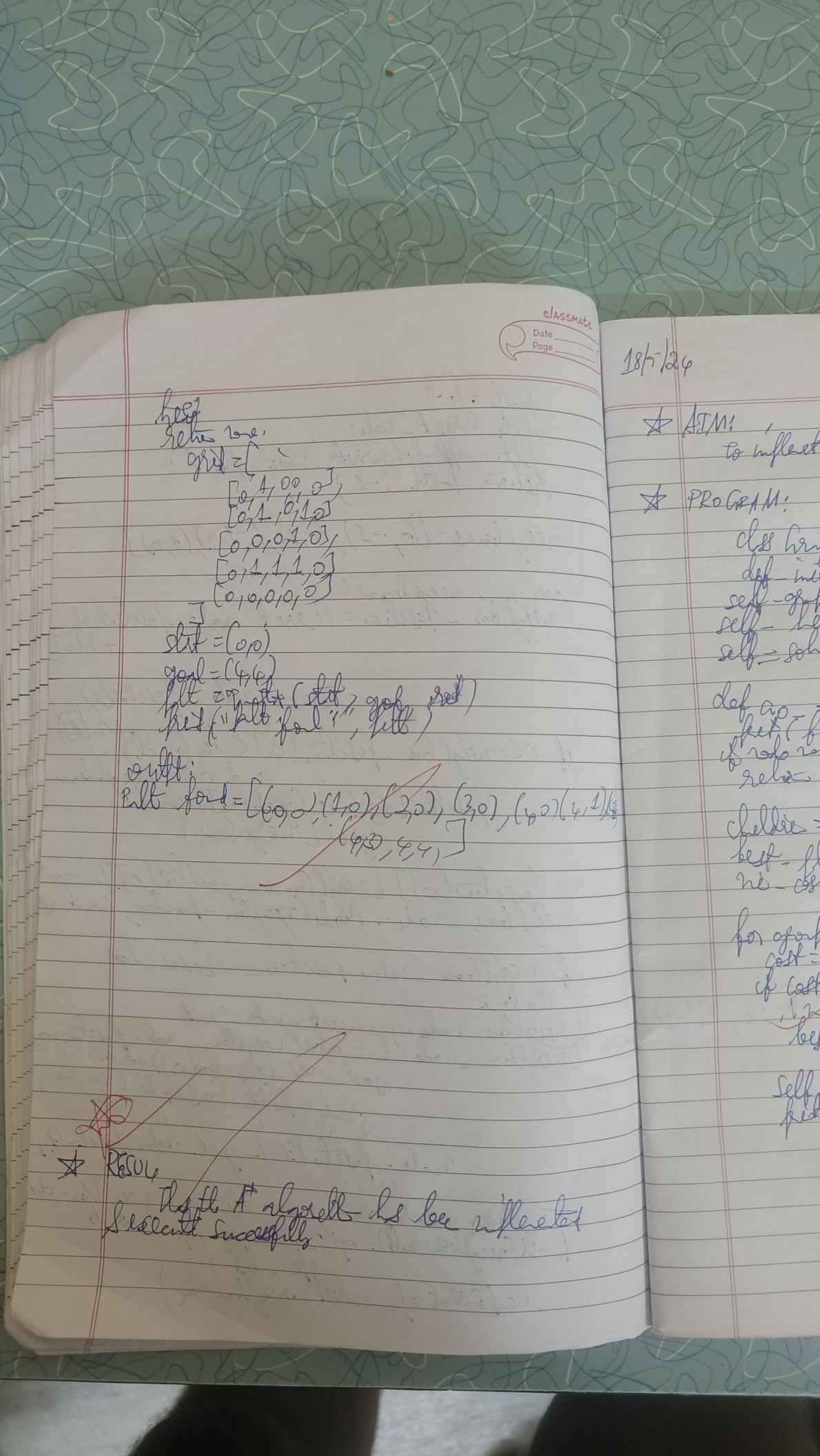
★ AIM:

To implement A* algorithm

★ PROGRAM:

```
from heapq import heappush, heappop
class Node:
    def __init__(self, position, parent=None):
        self.position = position
        self.parent = parent
        self.g = 0
        self.h = 0
        self.f = 0

    def __eq__(self, other):
        return self.position == other.position
    def __lt__(self, other):
        return self.f < other.f
def a_star(start, goal, grid):
    start_node = Node(start)
    goal_node = Node(goal)
    open_list = []
    closed_list = []
    closed_list = set()
    heappush(open_list, start_node)
while open_list:
    current_node = heappop(open_list)
    closed_list.add(current_node.position)
    if current_node == goal_node:
```

```
path = []
while current_node:
    path.append(current_node.position)
return path[::-1]

neighbors = [(0, -1), (0, 1), (-1, 0), (1, 0)]

for n in neighbors:
    neighbor_position = (current_node.position[0] + n[0],
                         current_node.position[1] + n[1])

if 0 <= neighbor_position[0] < len(grid) and 0 <= neighbor
    _position[1] < len(grid[0]) and
    grid[neighbor
    _position[0]][neighbor_position[1]] == 0:
neighbors_node = Node(neighbor_position, current_node)

if neighbor_node.position in closed_list:
    continue
neighbor_node.g = current_node.g + 1
neighbor_node.h = abs(neighbor_node.position[0] -
    goal_node.position[0]) + abs(neighbor_
    node.position[1] - goal_node.position[1])

neighbor_node.f = neighbor_node.g + neighbor_node.h
if all(neighbor_node != open_node for open_node in
    open_list):
    heappush(open_list, neighbor_node)
```



18/7/24

EXP-6

AO* algorithm

☆ AIM:
to implement AO* algorithm

☆ PROGRAM:

```
class Graph:
    def __init__(self, graph, heuristics):
        self.graph = graph
        self.heuristics = heuristics
        self.solution = {}

    def ao_star(self, node):
        print("Expanding node:", node)
        if node not in self.graph or not self.graph[node]:
            return

        children = self.graph[node]
        best_child = None
        min_cost = float('inf')

        for child in children:
            cost = sum([self.heuristics[child] for child in child])
            if cost < min_cost:
                min_cost = cost
                best_child = child

        self.solution[node] = best_child
        print("Best child for", node, "is", best_child, "with cost", min_cost)
```

```
for child in best_fit:
    self.go_to(child)
def get_solution(self):
    return self.solution
graph = {
'A' = [['B', 'C'], ['D']],
'B' = [['E']],
'C' = [['G']],
'D' = [['H']],
'F' = [],
'G' = [],
'H' = []
}
heuristic = {
'A': 0, 'B' = 1, 'C' = 2, 'D' = 6, 'E' = 1, 'G' = 4
}
graph_obj = Graph(graph, heuristic)
graph_obj.a_star(graph, heuristic)
solution = graph_obj.get_solution
```

Output:
Explored: A
Best left for A = ['B', 'C'] with cost 3
Explored: B
Best left for B: ['E'] with cost 1
Explored: F
Explored: C

Best left C = ['G']
Explored: G
Solution = {'A': ['B', 'C'], 'B': ['E'], 'C': ['G']}

★ RESULT:

Thus A0* algorithm implemented successfully.

Exp. No. 8

25/9/24 IMPLEMENTATION OF DECISION TREE CLASS[illegible]

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

☆ EXPLANATION:
☆ Import the tree
☆ Call the function for tree
☆ Assign values for x & y.
☆ Call the function on basis of given random values for and given feature
☆ Display the output.

```
import pandas as pd.
from sklearn tree import decision tree
data = {'Height':[152, 155, 172, 185, 167, 182, 164, 1[illegible]
'weight':[45, 57, 72, 75, 63, 73, 22, 23]
'Gender':['female', 'female', 'male', 'male',
'female', 'male', 'Female', 'male',
'female']
df = pd.DataFrame(data)
x = df['height', 'weight']
y = df['Gender']

classifier = Decision tree classifier()
classifier = fit(x, y)
height = float(input("enter height (in cm) for prediction")
random_value = pd.DataFrame([[height, weight]]
columns = ['height', 'weight'])
```

print(f'[illegible]
[illegible]

☆ OUTPUT:
Enter [illegible]
Enter we[illegible]
Predicted [illegible]
height [illegible]

☆ RESULT:
[illegible]

```
print(f"Predicted gender for height {height} and weight {weight}: {predicted_gender[0]}")
```

☆ OUTPUT:

Enter height (in cm) for prediction = 169
Enter weight (in kg) for prediction = 61
Predicted gender for height 169.0 cm & weight 61.0 kg: Female

☆ RESULT:

Thus the decision tree classification has been executed successfully.

16/10/24

EXP-9

Implementation of ANN for an application using Python Keras

☆ AIM:
To implement artificial neural network for application in regression using python

☆ PROGRAM CODE:

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

# Generate synthetic dataset for demo
X = np.random.rand(1000, 3)
y = 5 * X[:, 0] + 2 * X[:, 1] ** 2 + 1.5 * np.sin(X[:, 2]
[:, 2] * np.pi) + np.random.randn(1000) * 0.1

X_train, X_test, y_train, y_test = train_test_split(
plt.scatter(X, y, color='blue', s=2)

X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

model = Sequential()
model.add(Dense(64, activation='relu'))
model.add(Dense(1, activation='linear'))
```

☆ OUTPUT
Epoch
20/2
Epoch
20/2

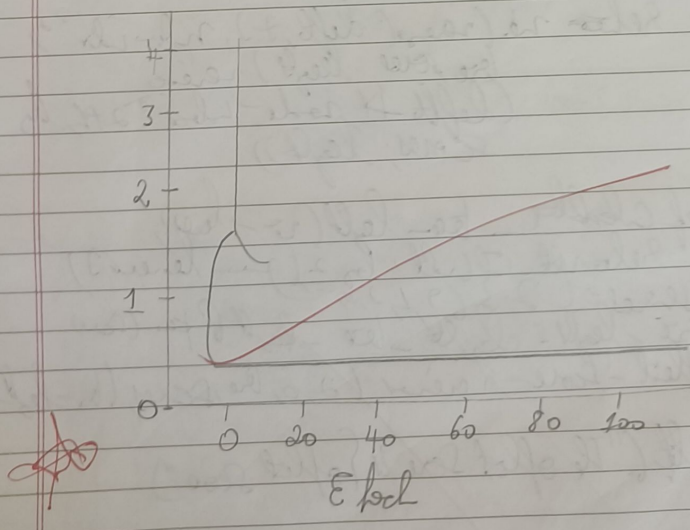
☆

model.compile(optimizer='adam', loss='mse')
y_pred = model.predict(X_test)
plt.figure(figsize=(12,6))
plt.plot(history.history['loss'], label=
plt.plot(history.history['val_loss'], label=
'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 ANN for regression problem successfully.

23/10/24

EXP-10

MINIMAX ALGORITHM

☆ AIM:
To implement minimax algorithm

☆ PROGRAM:

```
import math
def minimax(depth, node_index, is_maximizing, scores, height):
    if depth == height:
        return scores[node_index]
    if is_maximizing:
        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.log2(num_leaves)
scores = [3, 5, 2, 9, 12, 5, 23, 23]
tree_height = calculate_tree_height(len(scores))
optimal_score = minimax(0, 0, True, scores, tree_height)
print("The optimal score is", optimal_score)
```

☆ OUTPUT:

The optimal score is: 5

9/10/24

EXP. NO: 8

K-means clustering

☆ AIM:

To implement a K-means clustering technique using Python lang.

☆ Explanation:

☆ Import the required libraries for clustering.
☆ Assign x & y.
☆ Split the data K-means.
☆ Perform scatter operation & display the output.

☆ PROGRAM:

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
plt - x)
y_pred = KMeans(n_clusters=...).fit_predict(x)
plt.figure(figsize=(5,6))
plt.scatter(x[:,0], x[:,1], c=y_kmeans, s=30, cmap='...',
            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()
```

30/10/20

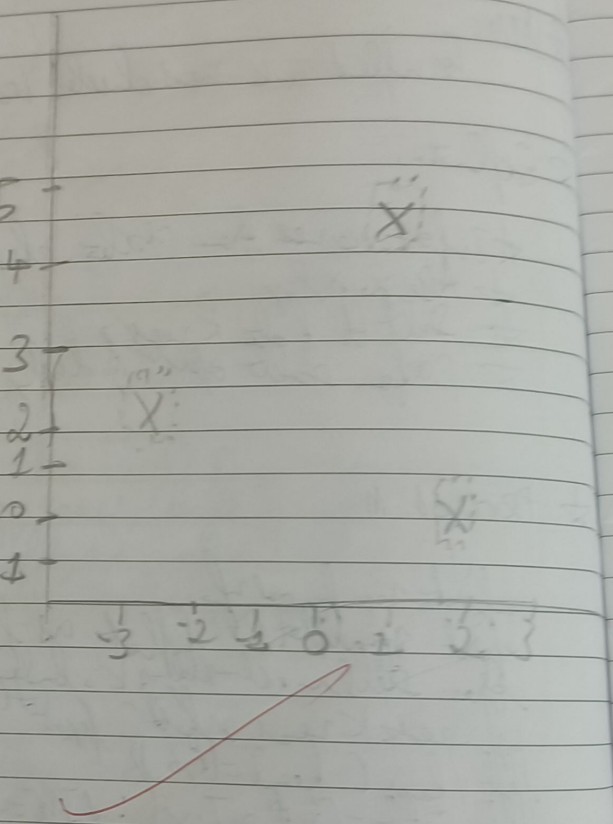
☆ AIM:

☆ TERM

1)

2)

3)



☆ RESULT

EXP-11

30/10/24

IMPLEMENTATION TO PROLOG

☆ AIM:

To learn PROLOG terminologies of write basic programs

☆ TERMINOLOGIES

1) Atomic terms

They are usually strings made up of lower & uppercase letters, digits & the underscore, starting with a lowercase

Eg: dog, ab-c-321

2) Variables

They are usually strings made up of lower & upper letters, digits & underscore, starting with a lowercase letter.

Eg: dog, apple-c-321

3) Compound terms:

Compound term:

Compound terms are the main block of PROLOG also of a name of argument.

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

4) Fact:

A fact is a predicate followed by a full stop.

Eg: bigger-animal(whale) life-is-beautiful

5) Rules:

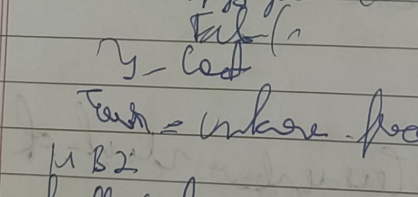
A rule consists of head & body. Eg:

is-smaller(X, Y), is-bigger(Y, X) & (aunt, die);

sister(Aunt, friend; friend, dad)

SOURCE CODE:

OUTPUT:



☆ OUTPUT;

```
?- owns(jack,X).
X = a(clay) ;
Y = owns(jack,T)
true.
```

☆ RESULT;

Thus the basic prolog program has been executed successfully.

EXP-12

6/11/24

PROLOG - FAMILY TREE

☆ AIM:
To develop a family tree program using PROLOG with all possible facts, rules & 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, betty)
parent of(kevin, chris)
parent of(kevin, lisa)
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)
grandfather(X, Y) :- male(Y), parent of(X, Z), parent of(Z, Y)
grandmother(X, Y) :- female(Y), parent of(X, Z), parent of(Z, Y)
brother(X, Y) :- male(Y) father(X, Z) father(X, W)
```

☆ OUTPUT:

☆ RE

☆ OUTPUT:

```
male(father)
true
[illegible](child_father)
true
[illegible](child-[illegible])
false
[illegible](daisy, x)
x = [illegible]
[illegible](child, [illegible])
false
```

☆ RESULT: This program for family tree program has been executed successfully.