Palindrome

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

O/P

madam
madam is a palindrome

2)
```
num = int(input("Enter a number: "))
flag = False
if num == 1:
    print(num, "is not a prime number")
elif num > 1:
    for i in range(2, num):
        if (num % i) == 0:
            flag = True
            break
    if flag:
        print(num, "is not a prime number")
    else:
        print(num, "is not a valid number to
              check for prime")
```

O/P

Enter a number: 144
144 is not a prime number.

6) Armstrong number or not

```
num = int(input("Enter a number"))
sum = 0
temp = num
while temp > 0:
    digit = temp % 10
    sum += digit ** 3
    temp // = 10
if num == sum:
    print(num, "is an Armstrong number")
else:
    print(num, "is not an Armstrong number")
```

O/P

Enter a number: 121
121 is not an Armstrong Number.

7.

```
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)
```

O/P

Enter a number: 5
The sum is 15

8)
```
num1 = int(input())
num2 = int(input())
num3 = int(input())

if (num1 >= num2) and (num1 >= num3):
    largest = num1
elif (num2 >= num1) and (num2 >= num3):
    largest = num2
else:
    largest = num3

print("The largest number is", largest)
```

O/P
```
5
6
2
The largest number is 6
```

9) Swapping:

```
x = input("Enter value of x: ")
y = input("Enter value of y: ")

temp = x
x = y
y = temp

print('The value of x after swapping: {}'.format(x))
print('The value of y after swapping: {}'.format(y))
```

O/P
```
The value of x after swapping: 10
The value of y after swapping: 5
```
x = 5
y = 10

6) LCM.

```
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 = 54
num2 = 24
print("The L.C.M is", compute_lcm(num1, num2))
```

31-10-2024

The LCM is 216

DIET RECOMMENDATION USING ML

OBJECTIVE

The primary objective of the "Diet Recommendation Using ML" project is to develop personalised nutrition plan to Individual health. By identifying potential nutritional deficiencies, the platform can suggests food or supplements to address these gaps, ensuring user receive comprehensive dietary support. Special dietary needs such as vegetarian, vegan and gluten-free diets, will also, be supported, ensuring balanced nutrition within these contexts.

DOMAIN:

The Domain for diet recommendation system lies at the intersection of health and wellness, with a strong focus on personalized nutrition. Data science and Machine learning are crucial for analyzing user data and generating customized diet plans.

Target people:

* Health-conscious individuals:

people seeking to improve their overall health through personalized diet plans.

* Health professionals.
* Individual with specific Dietary Needs.

Methods used:

Data collection and Input Handling:

Collect user input such as age, gender, weight, height, dietary preferences from Kaggle Datasets

Language Model Integration:

OpenAI Language Model to generate Recommendation based on the user inputs.

Data collection:

Collect data from various sources such as user input, health records and dietary logs

Supervised learning:

Use regression models to predict nutrition needs and suggest appropriate foods.

4/9/24

N-Queens

```
def is_safe(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
    return True

def solve_n_queens(board, col):
    if col >= len(board):
        return True
    for i in range(len(board)):
        if is_safe(board, i, col):
            board[i][col] = 1
            if solve_n_queens(board, col+1):
                return True
            board[i][col] = 0
    return False

def print_board(board):
    for row in board:
        print(" ".join(str(x) for x in row))
```

```
def solve():
    n=8
    board=[[0 for _ in range(n)] for _ in range(n)]
    if not solve_n_queens(board, 0):
        print("solution does not exist")
        return False
    print_board(board)
    return True

solve()
```

O/P

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

True.

Result:

Thus the above 8 queen problem is executed successfully.

A* algorithm

Program :-

```
def astaralgo (start_node, stop_node):
    open_set = set(start_node)
    closed_set = set()
    g = {}
    Parents = {}
    g[start_node] = 0
    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
        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 exis")
    return None

if n== stop_node:
    path[]

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

    Print("path found:{}".format(path))
    return path

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,
        'B': 6,
        'C': 99,
        'D': 1,
        'E': 7,
        'G': 0, }
    return H.dist[n]

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

astaralgo('A', G)
```

O/P

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

Result:

Thus the program is verified and executed successfully.

18/9/24

DEPTH FIRST SEARCH

PROGRAM:

```
def add_edge(adj, s, t):
    adj[s].append(t)
    adj[t].append(s)

def dfs_rec(adj, visited, s):
    visited[s] = True
    print(s, end=" ")
    for i in adj[s]:
        if not visited[i]:
            dfs_rec(adj, visited, i)

def dfs(adj, s):
    visited = [False] * len(adj)
    dfs_rec(adj, visited, s)

if __name__ == "__main__":
    V = 5
    adj = [[] for _ in range(V)]
    edges = [[1, 2], [1, 0], [2, 0], [2, 3], [2, 4]]
    for e in edges:
        add_edge(adj, e[0], e[1])
    source = 1
    print("DFS from source:", source)
    dfs(adj, source)
```

O/P

DFS from source: 1

1 2 0 3 4

Result:

Thus the DFS program is verified and executed successfully

DECISION TREE CLASSIFICATION

AIM:

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

PROGRAM:

```
import pandas as pd
from sklearn.tree import DecisionTreeClassifier
data={"height": [152, 155, 172, 185, 167, 180,
                 151, 180, 164, 177],
      "weight": [45, 57, 72, 85, 65, 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 your height in cm:"))
weight = float(input("Enter weight in kg: "))
```

```
Prediction_data = pd.DataFrame([[height, weight]],
                  columns=['Height', 'Weight'])
Predicted_gender = classifier.predict(prediction_data)
Print(f"Predicted gender: {Predicted_gender[0]}")
```

O/P

```
Enter height for prediction: 160
Enter weight for prediction: 46
Predicted gender for height 160.0 cm
    and weight: 46.0 kg : Female
```

Result:

Thus the Decision tree classifier was executed successfully

K-MEANS

AIM:

TO Implement a K-Means clustering technique using python.

PROGRAM:

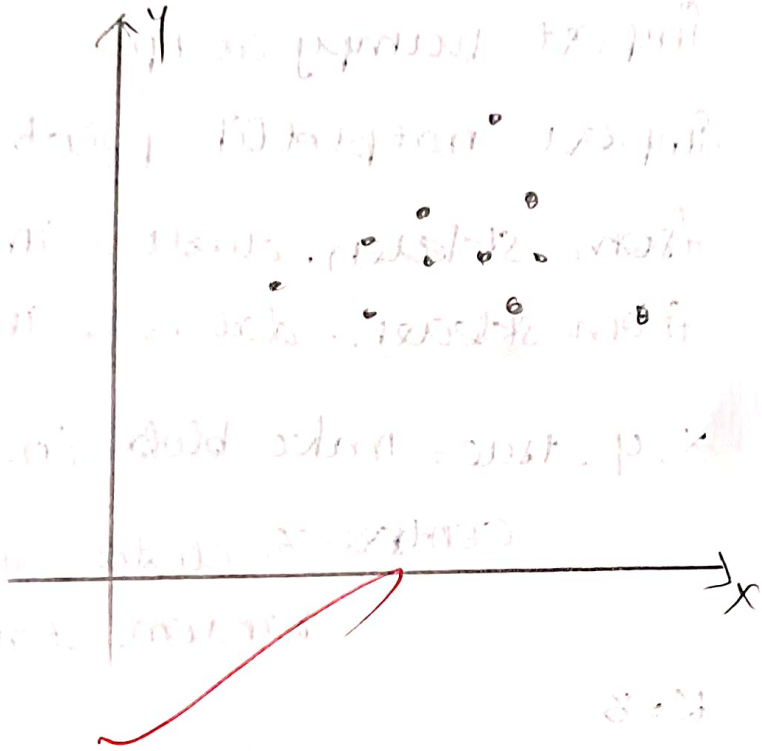
```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import kMeans
from sklearn.datasets import make_blobs

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()
```



Result:

Thus the K-means clustering was executed successfully

ANN

AIM:

TO implement artificial neural networks for an application in Regression using python.

PROGRAM:

```
from sklearn.neural_network import
                        MLPRegressor
from sklearn.model_selection import
                        train_test_split
from sklearn.datasets import make_regression
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
% matplotlib inline
X,y = make_regression(n_samples=1000,
                      noise=0.05, n_features=100)
X_train, X_test, y_train, y_test =
        train_test_split(X, y, test_size=0.2,
                         shuffle=True, random_state=42)
clf = MLPRegressor(max_iter=1000)
clf.fit(X_train, y_train)
```

AO* Algorithm.

AIM.

TO Implement AO* algorithm using Python programming Language.

CODE:

```
def __init__(self, graph, heuristic):
    self.graph = graph
    self.heuristic = heuristic
    self.solution = {}

def compute_cost(self, node):
    print(f"Expanding {node}")
    if node not in self.graph or not
        self.graph[node]
        return
    children = self.graph[node]
    best_path = None
    min_cost = float("inf")
    for group in children:
        cost = sum(self.heuristic[child] for
            child in group)
        if cost < min_cost:
            min_cost = cost
            best_path = group
```

```
self.solution[node] = best_path
print(f"Best path for {node} = {best_path}
        with cost {min_cost}")

for child in best_path:
    self.ao_star(child)

def get_solution(self):
    return self.solution

graph = {
    'A' = [['B', 'C'], ['D']],
    'B' = [['E']],
    'C' = [['G']],
    'D' = [['H']],
    'E' = [],
    'G' = [],
    'H' = [],
}

heuristic = {'A': 0, 'B': 1, 'C': 2, 'D': 4, 'E': 1,
             'G': 3, 'H': 5}

graph_obj = Graph(Graph, heuristic)
graph_obj.ao_star('A')
solution = graph_obj.get_solution
print('solution': solution)
```

OUTPUT:

Expanding : A

Best path for A : ['B', 'C'] with cost 3

Expanding B

Best path for B : ['E'] with cost 1

Expanding E

Expanding : C

Best path for C : ['G'] with cost 3

Expanding : G

Solution: {'A': ['B', 'C'], 'B': ['E'], 'C': ['G']}

Result:

Thus the program to implement AO* algorithm was verified and executed successfully

MIN-MAX ALGORITHM.

AIM:

To implement Min Max algorithm using python language.

CODE:

```
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_treeheight(len(scores))
optimal_score = minimax(0,0,True,
            scores, tree_height)
print(f"the optimal score is
            {optimal_score}")
```

Output:

The Optimal score is 5

Result:

Thus the minimax algorithm was executed successfully and output was verified

INTRODUCTION TO PROLOG

AIM

To learn prolog terminologies and write basic programs.

TECHNOLOGIES:

1) Atomic terms:

Atomic term is usually string made up of lower & upper case, digits, underscore starting with a lowercase letter

eg: dog, abc_321

2) Variables:

Variables are string of letters, digits and the underscore starting with captial letter. o, underscore.

eg: dog Apple-420

3) Compound terms:

compound terms are used to make up of a prolog atom. & a no. of arguments (prolog terms) and enclosed in parantheses and separated by commas.

eg. is_bigger (elephant, X)

4. Facts :

A fact is a pseudocode followed by a dot

eg: bigger_animal (whale).

5. Rules :

A rule of a head (a predicate) and a body (a sequence)

eg: is_smaller(x, y) :- is_bigger(y, x).

Source_code :

KB1 :

```
women(mia).
women(jody).
women(yolanda).
plays_air_guitar(mia).
party.
```

O/P

```
?- women(mia).
true

?- plays_air_guitar(mia).
false

?- party.
true

?- concert.
procedure concert
doesn't exist
```

Source_code:

KB:

happy(yosida)

listen2music(mia)

listen2music(yolandea) :- happy(yolandea)

playsAirGuitar(mia) : listen2music(mia)

playsAirGuitar(yolandea) : listen2music(yolandea)

o/p

?- playsAirGuitar(mia)

true

?- playsAirGuitar(yolandea)

true

KB 3

likes(dan, sally)

likes(sally, dan)

likes(jon, britney)

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, britteney)

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.

KB5:

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

O/P

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

Result:

Thus the prolog programs are 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 :

```
male (pelee)
male (john)
male (chris)
male (kevin)
female (betty)
female (jeny)
female (lisa)
femal (helen)
Parent of (chris. pelee)
parent of (chris. betty)
parentOf (helen. pelee)
parentOf (helen, betty)
parentOf ( kevin. chris)
parentOf (steven. lisa)
parent of ( jey. john)
parentOf (jeny. helen)
```

RULES:-

father(x, y) :- male(y), parentOf(x, y)

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

grandfather(x, y) :- male(y), parentOf(x, z),
parentOf(z, y)

grandmother(x, y) :- female(y), parentOf(x, z),
parentOf(z, y)

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

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

male(y), parentOf(x, y)

x: chris, y: peter

female(y), parentOf(x, y)

x: chris y: betty

male(y), parentOf(x, z), parentOf(z, y)

x: kevin y: peter z: chris

female(y), parentOf(x, z), parentOf(x, y)

x: kevin x2: betty z: chris.

Result:

Thus the prolog family tree was executed successfully