# INDEX

NAME SARAN RAJ . A SUBJECT PDAI Observation

STD. _____ DIV. _____ ROLL NO. 220701510 SCHOOL _____

| SR. NO. | PAGE NO. | TITLE | DATE | TEACHER'S SIGN / REMARKS |
|---|---|---|---|---|
| 01. | | Basic python program | 10 | |
| 02. | | N queen problem | 10 | |
| 03. | | DFS | 10 | |
| 04. | | A* algorithm | 10 | |
| 06. | | water jug problem using DFS. | 10 | |
| 06. | | Implementation of decision tree classification technique. | 10 | |
| 07. | | Implementation of artificial neural network for an application using python - keras | 10 | |
| 08. | | K-means clustering techniques using python | 10 | |
| 09. | | Introduction to PROLOG | 10 | |
| 10. | | Family tree program using prolog. | 10 | |

completed

Exno: 1

# Basic Python Program

1. Program to print the number of occurence of any element in list.

```
def count x (list x)
    count = 0
    for ele in list:
        if (ele == x)
            count = count + 1
        return count
list = {8, 6, 8, 10, 8, 20, 20, 30, 8, 8}
x = 8
Print ({} has occured {} times format (x, count x (list, x)))
```

Output:

8 has occured 5 times.

2. Program to print the index of element in list.

```
my-list = (10, 20, 30, 40, 50)
element-to-find = 30
if element-to-find in list:
    index = my-list index (element-to-find)
    Print (f "The index of {element-to-find} is {index}")
else
Print ("Not in list");
```

Output:

The index of 30 is 2.

3. Program to find the string is palindrome or not.

```
def ifpalin (text)
text = text.lower().replace(" ","")
reverse_text = text[::-1]
text = "hoh"
if ifpalin (text)
    Print ("It is palindrome");
else
    Print ("It is not palindrome");
```

Output:

It is palindrome.

4. Remove the element in the list:

```
my-list = (10, 20, 30, 40, 50)
remove = 30
if remove in my-list
    my-list.remove (remove)
    Print (f"The element {remove} has been removed")
    Print ("updated list", my-list)
else:
    Print (f"The element {remove} is not in list")
```

Output:

The element 30 has removed.

Updated list = (10, 20, 40, 50).

5. check the greater number:

```
a = 10
b = 20
if (a>b)
    Print (f "{a} is greater than {b}")
elif (b>a):
    Print (f "{b} is greater than {a}")
else:
    Print ("Both are equal")
```

Output:

20 is greater than 10.

6. Program to add the two list and return count.

```
list 1 = [1,2,3]
list 2 = [4,5,6,7]
combined list = list 1 + list 2
count = len (combined list)
Print ("combined list:" combined list)
Print ("count of element:" count)
```

Output:

combined list [1, 2, 3, 4, 5, 6, 7]

count of element : 6

7. Program to add the list value and return.

```
list 1 = [1, 2, 3]
list 2 = [4, 5, 6]
result = []
for i in range (len (list 1))
    result. append (list 1 [i] + list 2 [i])
Print (" Sum of element of corresponding position", result)
```

Output:

Sum of element corresponding position = (5, 7, 9).

8. swapping the element between two list:

```
list 1 = [1, 2, 3]
list 2 = [4, 5, 6]
list 1, list 2 = list 2, list 1
Print ("list 1:" list 1)
Print (" list 2:", list 2)
```

Output:

```
list 1 = [4, 5, 6]
list 2 = [1, 2, 3]
```

9. Reverse the array:

```
my - list = [10, 20, 30, 40, 50]
my - list reverse ()
Print (" Reverse list", my - list)
```

Output:

Reverse list [50, 40, 30, 20, 10].

10. Vowel count:

```
text = "This is a sample text"
vowel = "aeiou AEIOU"
vowel_count = 0
for char in text
    if char in vowels:
        vowel_count += 1
Print ("Number of vowels", vowels, count)
```

Output:

Number of vowels: 6

# Architecture recognization and design recommendation

Project overview:

AI based system to recognize architecture styles from image and provide recommendation design and estimating the cost of the design and how to implement.

Data collection:

→ Architecture style dataset
→ Design element
→ estimation value for implement design.

Model training:

→ Recognization
→ Design recommendation.

Target.

→ Architecture engineer
→ CAD designer
→ estimater for design
→ Educational use.

Algorithm:

convolutional neural network (CNN)

Domain: Architecture.

Architecture design Recommendation and recognization using ML algorithm.

Objective:

To develop an AI based system that accurately recognizes architectural styles from image and provide design recommendation based on user preference and recognized style.

Problem:

Architecture styles are defined by distinctive visual and structure element that can be challenging to identify manually. especially given the variety and evolution of styles. Further more translating style recognization into optimal design recommendation requires an intelligent system capable of understanding user preference and providing design solution.

Solution:

The solution involves creating a two parts system. one for architecture style recognization and another for design recommendation. The recognization use machine learning to classify architecture style from image the recommendation system a design based on user input.

Abstract:

In contemporary architecture, the fusion of artificial intelligence (AI) with design praxis hold the potential to revolutional how architecture

styles are identified and how design recommendation are generate. This paper proposes a it prepare a novel AI-based system designed to recognize architecture styles from images and provide design recommendation it integrated advanced machine learning algorithm and computer vision techniques.

Data set:

Building data set for architecture style. Architectural style recognization algorithm.

(i) Convolutional neural network (CNN)

It is used in training a large data set and to classify the image.

(ii) Vision transformer (VITs)

used to process the image which divide the image into patches which helps to predict accurately.

Exno:- 2

# N queen problem:

Aim:

To write a python program to place the q
on safe safe (board, row, col-n)

side from all the direction.

Program:

```
def is-safe (board, row, col, n)
    for i in range (col):
        if board [row] [i] == 1;
        return false
for i, j in zip (range (row, -1, -1); range (col,-
    if board [i][j] == 1:
        return false
    return true
def solve -n (board, col, n):
    if col >= n:
    return true.
for i in range (n)
    if is-safe (board, i, col, n)
    board (i) (col) = 1
    if solve n (board, col+1, n):
        return true
    board (i) (col) = 0
    return false.
def print board ( board, n)
```

```
Print ("\n Solution"):
    for row in board
        for col in row
            if col == 1
                Print ("Q", end = " ")
            where
                Print ('.' end ' ')
        Print ()
def true-n-queen (n)
board = [[0 for_in range(n)) for_in - range(n)]
    if not solve - n - queen - util (board, 0, n):
        Print ("no solution")
            return false
        Print board (board, n)
            return true
n = int (input ("Enter the value of N"))
    Solve - n - queen (n)
```

Result:

Thus N queen problem is executed successfully.

Output:

Enter the value : 4

" " Q "

Q " " "

" " " Q

" Q " "

True

Exno: 3

# DFS

Aim:

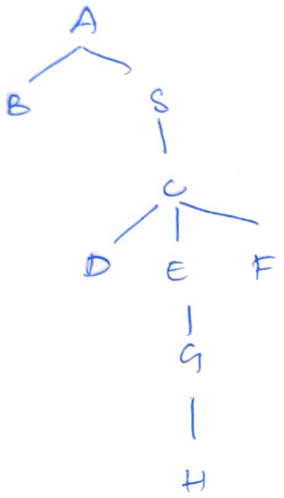
To write a python program for DFS

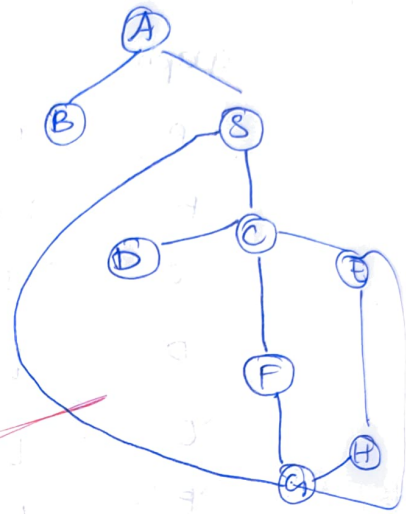
Program:

```
graph = {
'A' : ['B','S']
'B' : ['A']
'C' : ['D','E','F','S']
'D' : ['C']
'E' : ['C','H']
'F' : ['C','G']
'G' : ['F','S']
'H' : ['E','G']
'S' : ['A','C','G']
}
def dfs(graph, node, visited = None):
    if visited is None:
        visited = []
    if node not in visited
        visited.append[node]
    for neighbor in graph[node]:
        if neighbor not in visited:
        dfs(graph, neighbor, visited)
    return visited
visited, node, dfs(graph,'A')
Print(visit)
```

Output:

('A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I')




Result:

Thus the python program for DFS is executed successfully.

Ex no: 4

# A* algorithm

Aim:

To write a program code for A* algorithm

Code:

```
def - astar (start-node, stop-node)
    open-set = set (start-node)
        closed, cost = set()
            g = {}
            Parent = {}
            g[start-node] = 0
        Parent (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)
        if n = = stop-node or graph-node (n)
    else
        for (m, weight) in get-neighbor (n)
            not in close-set
                open-set. add (m)
            parent (m) = n
                g(m) = g(n) + weight
    else
            if g(m) > g(n) + weight
                g(m) = g(n) + weight
                    parent (m) = n
            if n = = none
        Print (" path does not exist")
```

```
        return none
    if n == stop_node:
        path = []
        while parent(n) != n:
            path.append(n)
            n = parent(n)
        path.append(start_node)
        path.reverse()
        print("path found:", path)
        return
    open_set.add(n)
    return none
else:
    return none
def heuristic(n):
    H = dist = {
        'A': 11,
        'B': 6,
        'C': 9,
        '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)],
}
astar('A', 'G')
```

Output:

Path found = ('A', 'B', 'G')

Result:

Thus the program for A* is executed successfully.

Qu 4. Water jug problem using DFS.

Aim:-

To write a python code for water jug problem using DFS.

Code :-

```
def water_jug (jug1, jug2, target)
    def dfs (j1, j2, jug, visited)
        if j1 = target & j2 == target
            return jug
        visited add ((j1, j2))
    action = ((fill,1) (fill,2) (empty,1) (empty,2))
    (pour 1,2) (pour 2,1)

    for location in action
        if action(0) == "fill"
            if action(1) == 1
            next state = (jug1, jug2)
        else
            next state (j1, jug2)
        else
        next state (j1, jug2)
            if action(1) == 1

                next state (0, j2)
    else
        if action (1) == 1
        amount = min (j1, jug, j2 + amount))
else
    amount - min (j2, jug2, j2)
```

next state ( j1 + amount , j2 , amount)

if next state not in visited :

next_seq = seq + (action)

next seq. visited

if result

return result

return none

visited = set()

return dfs (0, 0, []), visited )

result = water_jug_dfs (4, 3, 2)

Print (result)

Output:

(('fill', 1) , ('fill', 2) , ('empty', 1) ('pour', 2,

('fill', 2) ('pour', 2, 1))

Result:

Thus the program of water jug problem using DFS is executed successfully.

exno:6

# Implementation of decision tree classification technique.

Aim:

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

Explanation:

- Import tree from sklearn.
- Call the function decision tree classifier() from tree
- Assign value for x and y.
- Call the function predict for predicting on basis of given random for each given features.
- 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, 55, 68, 78, 82, 90, 66, 88]
'gender' ['Female', 'Female', 'male', 'Female',
'Male', 'Female', 'Male', 'Female' 'male']
}
df = pd.DataFrame(data)
x = df[['height', 'weight']]
x = df['gender']
```

```
classifier = DecisionTreeClassifier()
classifier = fit(x, y)
height = float(input("enter height (in cm) for prediction:"))
height = float(input("enter height in kg"))
Random variable = pd.DataFrame([[height, weight]])
column = ['height', 'weight']
predict_gender = classifier.predict(random_values)
Print predict gender for height (height y in cm and height & weight y kg = & predict_gender[0&1])
```

Output:

```
enter the height : 152
enter the age : 45
Predicted gender : Female
```

Result:

Thus the program for the decision tree was executed successfully.

Ex no:-1 Implementation of Artificial neural network for an application using python - regression.

Aim:

To implement of Artifical neural network for an application using python regression.

Code:

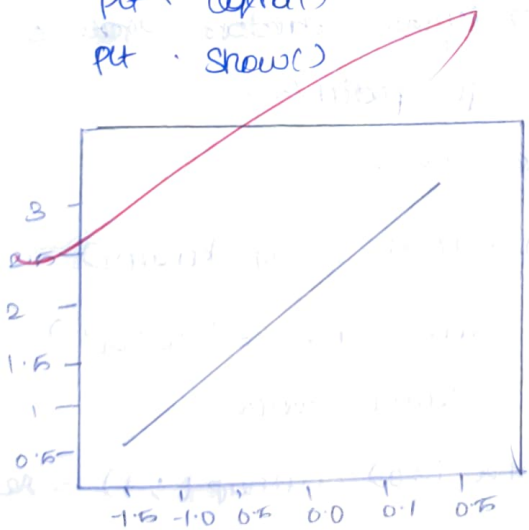
```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import standard scale
from keras.layers import Dense
from keras.optimizers import adam
import matplotlib.pyplot as plt
np.random.seed(42)
x = np.random.rand(1000,3)
y = 3*x[:,0]+2*x[:,1:]**2+1.5*
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, test_size=0.2, random_state=
scaler = standardscaler()
x_train = scaler.fit_transform(x_train)
y_test = scaler.transform(x_test)
model = sequential()
```

```
model.add(Dense(10, input_dim=x_train.shape[1],
                activation="relu"));

# Building the ANN model
model.add(Dense(1, activation='relu'))
model.add(Dense(1, activation='linear'))
model.compile(optimizer=adam(learning_rate=0.01),
loss='mean_squared_error')

history=model.fit(x_train, x_train, epochs=50,
                  batch_size=32, validation=0.2)

y_pred = model.predict(x_test)
mse = np.mean((y_test - y_pred.flatten())**2)
Print('mean squared error', mse.%(y)
plt.figure(fig size=(12, 6))
plt.plot(history, history['loss'], label="Training loss")
plt.plot(history history['val_loss'], label='validation
                                              loss')
plt.title('Training & validation loss')
plt.xlabel('epoch')
plt.ylabel('Loss')
plt.legend()
plt.show()
```




```
epoch 0/100
epoch 1/100
```

Result:

Thus the ANN using Python has been executed successfully.

Exno:8

# K means clustering technique using Python

Aim:

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

Explanation:

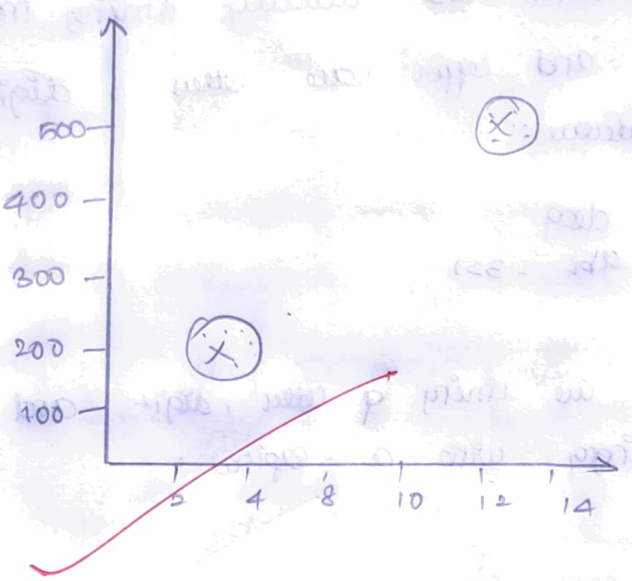
- Import K means from sklearn cluster
- Assign x and y
- Call the function K means ()
- Perform scatter operation and display the output.

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 = 3, random_state = 0)
y_kmeans = kmeans.fit_predict (x)
plt.figure (figure = (5,6))
plt.scatter (X[:,0] X [0.0] c = y_kmeans)
kso.cmap = 'viridis' label = 'cluster')
centers = kmeans.cluster_centers
plt.scatter (centers[:,0], centers[:,1] = red
s = 200, alpha = 0.75 marker = 'x',
```

```
label = 'Centroids'
plt.title('K-means clustering result')
plt.xlabel('Feature1')
plt.ylabel('Feature2')
plt.legend()
plt.show()
```

Output:




Result:

Thus the program for K-means clustering technique was executed successfully.

Exno: 9

# INTRODUCTION TO PROLOG

Aim:

To learn prolog terminologies and write the basic programs.

Terminologies:

Atomic term:

Atomic term are usually string make up of lower and upper case letters, digit and the underscore.

Ex: dog
abc_321

Variable:

Variable are string of letter, digit and the underscore starting with a capital.

Ex: Dog
Apple_420

Component terms:

Component terms are made up of a prolog atom and a number of argend and seperate by commas.

Ex: is_bigger(elephant, X)
:- big(X,-1), !

Fact:

A fact is a predicate followed by a dot.

Ex: bigger_animal(whale).
life_is_beautiful.

Rule:

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

Ex: p - smaller (x,y) _ p1 - bigger (y,x)

Source code:

women (min)

women (jody)

women (yolanda)

player guitar (jody)

Output:

Query 1: women (min)

true

Query 2: play air guitar (min)

false

happy (yolanda)

likes 2 music (mia)

likes 2 music yolanda : happy (yolanda)

Output:

Query 1 happy (yolanda)

true

Query 2 likes 2 music (mia)

false

KB3:

likes (don, sally)

likes (sally, don)

likes (john, betting)

means (x,y) :- likes (x,y), likes (y,x)

```
married (x, y) :- likes (x, y), likes (y, x)
friend (x, y) :- likes (x, y) ; likes (y, x)
```

Output:

```
likes (dl, x)
x = Sally
true
likes (sally, sally)
false
```

KB 4

```
food (burger)
food (sandwich)
food (pizza)
lunch (sandwich)
dinner (pizza)
```

Output:

```
food (pizza)
true
dinner (sandwich)
false.
```

Result:

Thus the program for prolog has been executed successfully.

ex 10:

# PROLOG

Aim:

To develop a family tree program using PROLOG with all possible facts, rules and queries.

Code:

```
male (peter)
male (John)
male (chris)
male (Kevin)

female (betty)
female (jenny)
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)

/* RULE */

Kron, parent
of son, paid 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)
```

mother (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:

? parent of (kevin, x)
x = chris

? father (x, chris)
x = kevin

? sister (x, chris)
false.

Result:

Thus the program for family tree using prolog has been verified successfully.