PDAT

OBSERVATIO

Mychoice

# INDEX

NAME: G. BALAJI STD: III SEC.: A ROLL NO.: 36

| S.No. | Date | Title | ~~Page No.~~ marks | Teacher's Sign / Remarks |
|---|---|---|---|---|
| 1) | 31/7/24 | BASIC PYTHON PROGRAMS | 9 | |
| 2) | 7/8/24 | HOTEL PRICE PREDICTION | 10 | |
| 3) | 4/9/24 | N-QUEEN PROBLEM | 10 | |
| 4) | 18/9/24 | A* PROBLEM | 10 | |
| 5) | 18/9/24 | AO* PROBLEM | 10 | |
| 6) | 27/9/24 | INTRODUCTION TO PROLOG | 10 | |
| 7) | 27/9/24 | PROLOG - FAMILY TREE | 10 | |
| 8) | 4/10/24 | IMPLEMENTATION OF DECISION TREE CLASSIFICATION TECHNIQUES | 10 | |
| 9) | 4/10/24 | IMPLEMENTATION OF CLUSTERING TECHNIQUES K-MEANS | 10 | |
| 10) | 18/10/24 | IMPLEMENTING ARTIFICIAL NEURAL NETWORKS FOR AN APPLICATION USING | 10 | |
| 11) | 25/10/24 | PYTHON REGRESSION WATER-JUG PROBLEM | 10 | |

completed

# BASIC PYTHON PROGRAMS

EXP. NO: 1 DATE:

1) Factorial

```
def factorial(n):
    if n==0:
        return 1
    else:
        return n * factorial(n-1)

num = int(input("Enter a number"))
print("Factorial of", num, "is", factorial(num))
```

INPUT: 2

OUTPUT: 2

2)

```
def sum_digits(n)
    sum=0
    while(n!=0)
        sum=sum+int(n%10)
        n=int(n/10)
    return sum

num= int(input("Enter a number"))
print(sum digits(num))
```

INPUT = 121

OUTPUT = 4

3)

```
def Palindrome(s)
    return s==s[::-1]

word = input("Enter a string"),
ans = Palindrome(word)
if ans:
    print("Yes")
else:
    print("No")
```

INPUT: HELLO
OUTPUT: OLLEH

4)

```
def armstrong(number)
    sum=0
    while(number>0)
        digit = number%10
        sum=sum+digit
        number//=10
    return sum

number = int(input())
b = number
```

INPUT: 153
OUTPUT: $1^3+5^3+3^3$
$=153$

5)

```
def swab(a,b):
    a=a-b
    b= b+a;
    a= b-a
```

INPUT: a=10
b=20
OUTPUT: a=20
b=10

return a,b

a=int(input())

b=int(input())

print(swap(a,b))

6)

```
num = int(input)
if num%.2 ==0
        print("Even")
else
        print("odd")
```

INPUT: 10

OUTPUT: EVEN

7)

```
l=int(input())
b= int(input())
print(l*b)
```

~~O/P l INPUT~~

10 b/a INPUT

20 b

200: OUTPUT

8)

```
def isPrime(n):
    if(n<=1)
        print False
    for i in range(2, int(n**0.5)+1):
        if n%.i==0:
            return False
```

return true

INPUT: 1

OUTPUT TRUE

```
n=int(input())
print(isPrime(n))
```

9)

```
def Fibonacci(n)
    a,b=0,1
    for i in range(n):
        print(a,end=" ")
        a,b=b,a+b
n=int(input(n))
print(fibonacci(n))
```

INPUT: 5

OUTPUT = 0,1,1
2,3

10)

```
def gcd(a,b)
    while b:
        a,b<b,a%.b
    return a
n=int(input())
m=int(input())
print(gcd(n,m))
```

INPUT: a=20
b=28

OUTPUT: 4

31/07/2024

# N-QUEEN PROBLEM

```
def solve_n_queens(n):
    def is_safe(board, row, col):
        for i in range(col):
            if board[row][i] == 'Q':
                return false

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

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

        return true

    def solve_rec(board, col):
        if col==n:
            return true
```

```
for i in range(n):
    if is_safe(board, i, col):
        board[i][col] = 'Q'
        if solve_rec(board, col+1):
            return True
        board[i][col] = '.'

return False

def print_solution(board):
    for row in board:
        print(" ".join(row))

board = [['.' for _ in range(n)] for _ in
         range(n)]

if solve_rec(board, 0):
    print_solution(board)
else:
    print("No solution exists")

n = 4
solve_n_queens(n)
```

RESULT

The above N queen problem has been executed successfully

OUTPUT:

n = 4

.Q.. ..Q.

...Q Q...

Q... ...Q

..Q. .Q..

# A* PROBLEM

```
import heapq

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

def a_star_search(grid, start, goal):
    rows, cols = len(grid), len(grid[0])
    open_list = []
    heapq.heappush(open_list, (0, start))
    came_from = {}
    g_score = {start: 0}
    f_score = {start: heuristic(start, goal)}
    while open_list:
        current = heapq.heappop(open_list)[1]

        if current == goal:
            path = []
            while current in came_from:
                path.append(current)
                current = came_from[current]
            path.append(start)
            return path[::-1]
```

```
for dxdy in [(-1,0), (1,0), (0,-1), (0,1)]:
    neighbour = [current[0] +
                 dx, current[1] + dy]
    if 0<=neighbour[0]<rows
     and 0<=neighbour[1]<cols
     and grid[neighbor[0]]:
    tentative_g_score = g_score
                 [current]+1
    if neighbour not in
 g_score or tentative_g_score <
                 g_score[neighbor]:
    came_from[neighbor] = current
    he
    f_score[neighbour] = tentative_
                 gscore

return None
```

INPUT: d = [

[0,1,0,0,0],

[0,1,0,1,0],

[0,0,0,1,0],

[1,1,0,0,0].

[0,0,0,0]

OUTPUT:

Path found: [(0,0),(1,0),(2,0),(2,1),
(2,2),(3,2),(3,3),(3,4),(4,4)]

RESULT:

Thus the above A* program has been completed successfully

# AO* PROGRAM

```
class Node:
    def __init__(self, name, value):
        self.name = name
        self.value = value
        self.children = []

    def add_child(self, node):
        self.children.append(node)

def ao_star(root, goal):
    queue = [[root, [root]]]
    while queue:
        node, path = queue.pop(0)
        if node == goal:
            return path
        for child in node.children:
            new_path = path + [child]
            queue.append([child, new_path])
    return None
```

INPUT:

A = Node('A', 0)

B = Node('B', 1)

C = Node('C', 2)

D = Node('D', 3)

E = Node('E', 4)

OUTPUT:

Solution path: ['A', 'B', 'D', 'E']

RESULT:

Thus the above program has been executed successfully.

EXP.NO: INTRODUCTION TO PROLOG

AIM:

To learn prolog terminologies & write basic programs

TERMINOLOGIES:

ATOMIC TERMS: Strings, made up of lower-case & uppercase letters.

VARIABLES: strings of letters, digits & underscore

COMPOUND TERMS: Made up of PROLOG atom

FACTS: Predicate followed by dot

RULES: Consists of a head & a body

SOURCE CODE

KB1:

```
woman(mia)
woman(jody)
woman(yolanda)
playsAirGuitar(jody)
party
```

OUTPUT:

```
? - woman(mia)
true
```

? - playsAirGuitar(mia)

false

? -party

true

? -concert

ERROR

KB2:

happy(yolanda)

listen2music(mia)

Listens2music(yolanda)

playsAirGuitar(mia)

OUTPUT:

? -playsAirGuitar(mia)

true

? -playsAirGuitar(yolanda)

true

KB3:

likes(dan, sally)

likes(sally, dan)

likes(john, britney)

OUTPUT:

? -likes(dan, x)

x=sally

? - married(dan, sally)

true

KB4:

food(burger)

food(sandwich)

food(pizza)

O/P:

? - food(pizza)

true

? - dinner(sandwich)

false

KB5:

owns(jack, car(bmw))

owns(john, car(chevy))

owns(olivia, car(civic))

sedan(car(bmw))

O/P:

? - owns(john)

true

? - owns(who, car(chevy))

who = john.

RESULT:

Thus the output has been verified successfully.

Ex. No:

# PROLOG - FAMILY TREE

## AIM:

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

```
male(peter)
male(john)
male(chris)
male(kevin)

female(betty)
female(jeny)
female(lisa)
female(helen)

parentof(chris, peter)
parentof(chris, betty)
parentof(helen, peter)
parentof(helen, betty)
parentof(jeny, john)
parentof(jeny, helen)

father(X,Y): male(X) parentof(X,Y)
mother(X,Y): female(X), 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), father(X,Z), father
(Y,W), Z==W

sister(X,Y) = female(Y), father(X,Z), father
(Y,W), Z
= =W

OUTPUT:

| Y | X |
|---|---|
| Peter | chris |
| Peter | helen |
| john | jeny |
| chris | kevin |

female(X), parentof(X,Z), parentof(Z,Y)

| Y | X | Z |
|---|---|---|
| betty | kevin | chris |
| betty | jeny | helen |

female(X), father(X,Z), father(Y,W), Z==W

procedure 'father(A,B)' does not exist

OUTPUT:

Thus the program has been executed successfully.

EXP. NO:

# 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 predict for predicting on the basics of given random values for each given feature

PROGRAM

```
import pandas as pd.
from sklearn.tree import DecisionTree
data = {
height = [152, 155, 172, 187, 177]
weight = [45, 57, 72, 85, 60]
Gender: [Female, Female, Male, Male]
}
df = pd.DataFrame(data)
x = df[['Height', 'Weight']]
Y = df['Gender']
classifier = DecisionTreeClassifier()
```

```
classifier.fit(x, y)
height = float(input(" Enter the height "))
weight = float(input(" Enter the weight "))
random_values = pd.DataFrame([[height,
                weight]])

Predicted_gender = classifier.predict(random_
                                values)

print("predicted gender for height {
    height} cm & weight {weight} kg :
    {predicted_gender[0]}")
```

| OUTPUT | FEATURE 1 | FEATURE 2 |
|---|---|---|
| 0 | 0.836857 | 2.136354 |
| 1 | -1.413658 | |
| 2 | 1.155213 | |
| 3 | | |
| 4 | 1.271351 | |

EXP. NO:

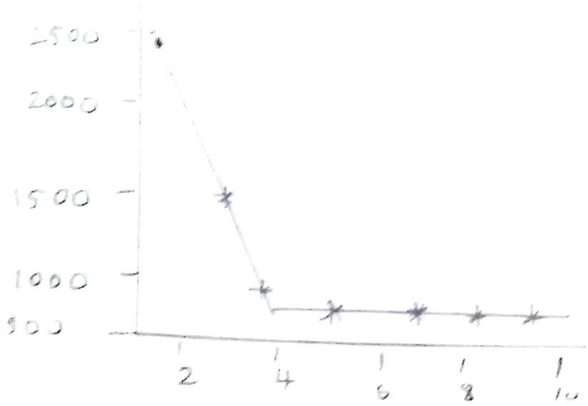
# IMPLEMENTATION OF CLUSTERING TECHNIQUES k-MEANS

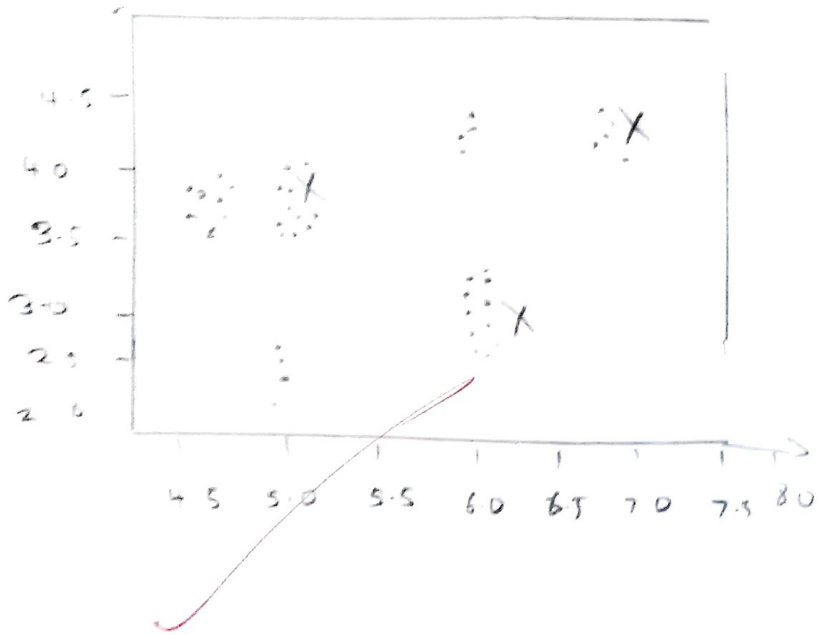
AIM:

To implement a k-means clustering technique using python language

```
import numpy as np
Import matplotlib.pylot as plt
from sklearn.cluster import k-means
from sklearn.datasets import male-blobs
X, y_true= male_blobs( n_samples = 300,
        centers=3, cluster_ctd = 0.60, random-
        state=0)
k=3
lc means = lcmeans(n_ clusters = k, random_state
                                =0)
y_lc means = kmeans.fit_predict(x)
plt.figure( fig.size=(8,6))
plt.scaHer (x[:0], x[:1], c=y_kmeans
centers= lcmeans.cluster_center
plt.scaHer (center[:0], center[:1], c='red',
        s=200, alpha=0.75)
```

```
plt.xlabel('Feature1')
plt.ylabel('Feature2')
plt.legend()
plt.show()
```




**RESULT:**

Thus the output of the above program has been verified successfully.

# IMPLEMENTING ARTIFICIAL NEURAL NETWORKS FOR AN APPLICATION USING PYTHON

## REGRESSION

EXP.NO:

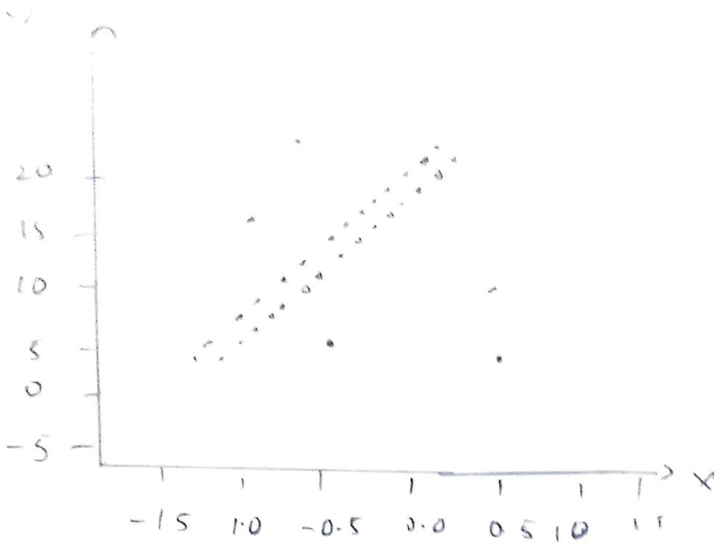
**AIM:**

To implement artifical neural networks for an application in regression using python.

```
import numpy as np
import pandas as pd
from sklearn.model_selection
from sklear.preprocessing import standascaler
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.random.normal(0,0.1000)
x_train, x_test, y_train, y_test= train_test_
                    split(x,y, test_size=0.2, random
                                    _state=42)
scales=standard scaler()
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'))
model.add(Dense(10, activation = 'linear'))
model.compile(optimizer = adam
        (learning_rate = 0.01)
history = model.fit(x_train, y_train,
epochs = 100, batch_size = 32
validation_split = 0.2, verbose = 1)
y_pred = model.predict(x_test)
print(tmean
plt.figure(figsize = (12,6))
plt.plot(history.history['loss'],
  label = 'Training loss')
  plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()
plt.show()
```

OUTPUT:




FEATURE

RESULT:

The output of the above program has been verified successfully.

DEPTH-FIRST SEARCH

Exp No.

WATER JUG PROBLEM

Aim:

To implement the water jug problem using depth first search in python

Program:

```
def water_jug_problem_dfs(jug1_cap, jug2_cap, target_amount):
    j1=0
    j2=0
    actions=[('fill',1),('fill',2),('empty',1),
    ('empty',2),('pour',1,2),('pour',2,1)]
    visited=set()
    stack=[(j1,j2,[])]
    while stack:
        j1,j2,seq=stack.pop()
        if (j1,j2) not in visited:
            visited.add((j1,j2))
            if j1==target_amount or
                j2==target_amount:
                return seq
```

```
for actions in actions:
    if action[0]=='fill':
        if action[1]==1:
            next_state=(jug1_cap, j2)
        else:
            next_state=(j1, jug2_cap)
    elif action[0]=="empty":
        if action[1]==1:
            next_state=(0,j2)
        else:
            next_state=(j1,0)
    else:
        if action[1]==1:
            amount= min(j1,jug2_cap-j2)
            next_state=(j1.-amount,j2+
                amount)
        else:
            amount= min(j2, jug_cap-j1)
            next_state=(j1+amount,j2-
                amount)
```

```
if next_state not in visited:
        next_seq = seq + [action]
        stack.append((next_state[0],
            next_state[1], next_seq))
return none
```

Eg:

```
result = water_jug_problem_dfs(3,2,1)
print(result)
```

OUTPUT:

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

RESULT:

Thus the above program has been executed successfully.