Experiment - 1

Write a Python program to implement Simple Calculator Program.

AIM:

To write a python program to implement simple calculator.

ALGORITHM:

1. Take 2 no as inputs.
2. Take operation (+, -, /) as input
3. Perform operation:
   * if "+", add the numbers.
   * if "-", subtract second from first.
   * if "/", divide first by second (check for zero).
4. Display the result.
5. Ask if user wants another calculation; if not, exit.

Program:

```
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):
    return x/y
print ("select oke")
print ("1. Add")
```

```
print ("2. subtract")
print ("3. Multiply")
print ("4. Divide")
choice = input ("Enter choice (1/2/3/4):")
num1 = 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))
else:
    print ("Invalid input").
```

Output:

```
Select operation.
1. Add
2. Subtract
3. Multiply
4. Divide
Enter choice (1/2/3/4): 3
Enter first num : 15
Enter 2nd number : 14
15.0 * 14.0 = 210.0
Let's do next calculation ? (yes/no): no.
```

Result:

Hence, the simple python program for Multiplication table is done.

Experiment : 2

Write a python program to Add 2 matrices.

AIM:

To write a python Program to add 2 matrices.

ALGORITHM:

1. Initialize two matrices, A and B, of size m x n.
2. Create an empty matrix C of same size
3. use nested loops to iterate through each element:
   * Add corresponding elements of A and B.
   * Store the sum in matrix C.
4. Display matrix C.

PROGRAM:

```
matrix 1 = [[1,2,3],
            [4,5,6],
            [7,8,9]]

matrix 2 = [[10,11,12],
            [13,14,15],
            [16,17,18]]

result = [[0,0,0],
          [0,0,0],
          [0,0,0]]

for i in range (len (matrix 1)):
    for j in range (len (matrix 1[0])):
        result [i] [j] = matrix [i] [j] + matrix 2[i][j]

for row in result:
    print(row)
```

Output:

```
x = [[1,2,3],
     [4,5,6],
     [7,8,9]]

y = [[10,11,12],
     [13,14,15],
     [16,17,18]]

Result = [[0,0,0],
          [0,0,0],
          [0,0,0]]
```

Result:

Hence, the simple python program for matrix table is done.

Experiment : 3

Write a python Program to Transpose a Matrix.

AIM:

To write a python program to transpose a matrix.

ALGORITHM:

1. Initialize matrix A of size m x n.
2. Create an empty matrix B of size n x m.
3. use nested loops to iterate through A:
   * Store A[i][j] in B[j][i].
4. Display matrix B.

PROGRAM:

```
matrix = [[1,2,3],
          [4,5,6],
          [7,8,9]]
row = len(matrix)
col = len(matrix[0])
transpose_matrix = [[0 for x in range(row)]
for y in range(col)]
for i in range(row):
    for j in range(col):
        transpose_matrix[j][i] = matrix[i][j]
print("row")
print("Transposed matrix:")
for row in transpose_matrix:
    print(row)
```

OUTPUT:

```
[
    [1, 2],
    [2, 3],
    [3, 4]
]

[
    [1, 2, 3],
    [2, 3, 4]
]
```

RESULT:

Hence, the simple python program for transpose a matrix is done.

Experiment : 4

Write a python program to sort the sentence in alphabetical order.

AIM:

To write a python program to sort the sentence in alphabetical order.

ALGORITHM:

1. Start by taking input.
2. split sentence into individual words & store it
3. use sort() method to sort in alphabetical order.
4. Join the sorted list of words back into a sentence.
5. Display the resulting sorted sentence.

PROGRAM:

```
sentence = input ("Enter a sentence: ")
words = sorted (sentence.split())
sorted_sentence = " ".join (words)
print ("sorted sentence:", sorted_sentence)
```

OUTPUT:

The sorted words are:

```
an
cased
example
hello
is
letters
this
with.
```

Experiment:5

Write a python program to implement list operations (Nested list, Length, Concatenation, Membership, Iteration, Indexing and slicing)

AIM:

To write a python program to implement the given list operations.

PROGRAM:

```
nested_list = [[1,2,3], [4,5,6], [7,8,9]]
print ("Nested List:", nested_list)
my_list = [1,2,3,4,5]
print ("Length of list:", len(my_list))
list1 = [1,2,3]
list2 = [4,5,6]
Concatenated_list = list1 + list2
print ("Concatenated list:", concatenated_list)
my_list = [1,2,3,4,5]
if 3 in my_list:
    print ("3 is in list")
else:
    print ("3 is not in the list")
my_list = [1,2,3,4,5]
if 3 in my_list:
    print ("3 is in list")
else:
    print ("3 is not in list")
```

RESULT:

Hence, the simple python program for sort the sentence in alphabetical order is done.

```
my_list = [1,2,3,4,5]
for item in my_list:
print (item)
my_list = [1,2,3,4,5]
print ("First element:", my_list [0])
print ("Last element:", my_list [-1])
print ("Sliced List:", my_list [1:4])
```

OUTPUT:

Before Append: [21, 34, 54, 12]

After Append: [21, 34, 54, 12, 32].

RESULT:

Hence, the simple python program for implement the operations is done.

21/2/24

Experiment : 6

Write a python program to implement List methods (Add, Append, Extend & Delete).

AIM:

To write a python program to implement the given list methods.

ALGORITHAM:

1. Initialize a list : my_list = []
2. Append an element : my_list.append (10)
3. Extend with multiple elements :

   my_list.extend ([20, 30, 40])
4. Insert at a specific index : my_list.insert (1,15)
5. Remove a specific element : my_list.remove (30)
6. pop the last or specific element : my_list.pop()

   or my_list.pop (2)
7. Delete an element or slice : del my_list [1] or

   del my_list [1:3].

PROGRAM:

```
my_list = []
my_list.add (1)
print ("List after adding an item using add method:",
my_list)
my_list.append (2)
print ("List after appending an item using append
method:", my_list)
my_list.remove (3)
print ("List after removing an item using remove
                method:", my_list).
```

OUTPUT:

List elements after deleting are : 2 1 3 8

List elements after popping are : 2 1 8

RESULT:

Hence, the simple python program for implement the list of elements is done.

Experiment : 7

Write a Python Program to Illustrate Different set operations.

AIM:

To write a python to illustrate different set operations.

ALGORITHM:

1. Create two sets, set A and set B.
2. Find intersection : intersection_set = set
3. union_Set = setA.union(setB)
4. Difference : difference_set = setA.difference(setB)
5. check subset : print(setA.issubset(setB)).

OUTPUT:

union of E and N is {0,1,2,3,4,5,6,8}

Intersection of E and N is {2,4}

Difference of E and N is {8,0,6}

Symmetric difference of E and N is {0,1,3,5,6,8}.

RESULT:

Hence, the simple python program for different set of operations is done.

Experiment: 8

Write a python program to generate calendar for the given month and year.

AIM:

To write a python program to generate calendar for the given month and year.

ALGORITHM:

1. Input month and year from the user.
2. Calculate the total days in the month, considering leap years.
3. Determine the first day of the month using Zeller's congruence or another method.
4. Print the calendar header (month and year).
5. Print days of week as column headers.
6. Print calendar layout, aligning dates correctly.

PROGRAM:

```
import calendar
year = int(input("Enter the year: "))
month = int(input("Enter the month: "))
print(calendar.month(year, month))
```

OUTPUT:

```
    November 2014
Mo Tu We Th Fr Sa Su
                1  2
 3  4  5  6  7  8  9
10 11 12 13 14 15 16
17 18 19 20 21 22 23
24 25 26 27 28 29 30
```

RESULT:

Hence, the simple python program for calendar is done.

Experiment : 9

Write a Python Program to Remove punctuations from the given string.

AIM:

To write a python program to remove punctuations from the given string.

PROGRAM:

```
import string
input_string = input("Enter a string: ")
no_punctuations = input_string.translate(str.maketrans
                  ("","", string.punctuation))
print("string with no punctuations:", no_punctuation)
```

OUTPUT:

```
original text:
string ! with . punctuation?
After removing punctuations from said string:
string with punctuation.
```

RESULT:

Hence, the simple python program for punctuations is done.

Experiment : 10

Write the Python Program to solve 8-Puzzle Problem.

AIM:

To write a python Program to solve the 8-puzzle problem.

PROGRAM:

```
from queue import PriorityQueue
goal_state = (1,2,3,4,5,6,7,8,0)
def heuristic (state):
    in range (3):
        for j in range (3):
            tile = state [i*3+j]
            if tile != 0:
                x, y = (tile-1) // 3, (tile-1) % 3
                distance += abs (x-i) + abs (y-j)
    return distance.
def solve (initial_state):
    frontier = PriorityQueue()
    frontier.put ((heuristic (initial_state), initial_state)
    explored = set()
    while not frontier.empty():
        _, state = frontier.get()
        if state == goal_state:
            return True
        explored.add (state)
        for successor not in explored:
```

```
        frontier.put((priority, successor))
    return False
def successors(state):
    successors = []
    i = state.index(0)
    if i % 3 != 0:
        new_state = list(state)
        new_state[i], new_state[i-1] = new_state[i-1],
                                       new_state[i]
    if i % 3 != 2:
        new_state = list(state)
        successors.append(tuple(new_state))
    if i // 3 != 0:
        new_state = list(state)
        successors.append(tuple(new_state))
    if solve(initial_state):
        print("The puzzle is solvable!")
    else:
        print("The puzzle is unsolvable.")
initial_state = (2,8,3,1,6,4,7,0,5)
        new_state[i], new_state[i+3] = new_state[i+3],
                                       new_state[i]
    return successors.
```

OUTPUT:

```
1 2 3
5 6 0
7 8 4

1 2 3
5 0 6
7 8 4

1 2 3
5 8 6
7 0 4

1 2 3
5 8 6
0 7 4
```

RESULT:

Hence, the simple python program for 8-Puzzle is done.

Experiment : 11

Write the Python Program to Solve 8-Queen Problem.

Program :

```
def solve_queens(n):
    board = [-1]*n
    result = []
    def is_valid(row, col):
        for i in range(row):
            if board[i] == col or \
                abs(row-i) == abs(col-board[i]):
                return False
        return True
    def backtrack(row):
        if row == n:
            result.append(list(board))
            return
        for col in range(n):
            if is_valid(row, col):
                board[row] = col
                backtrack(row+1)
                board[row] = -1
    backtrack(0)
    return result
solutions = solve_queens(8)
for solution in solutions:
    for row in range(8):
        line = " "
        for col in range(8):
            if solution[row] == col:
                line += "Q"
            else:
                line += "-"
        print(line)
    print("\n")
```

Output :

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

RESULT :

Hence, the simple python program for 8-Queen problem is done.

Experiment - 12

Write the Program for Water Jug Problem

Program:

```
from collections import deque

def water_jug_problem(x, y, z):
    """
    Solves the water jug problem using BFS.
    x: size of jug 1
    y: size of jug 2
    z: target amount of water
    Returns a sequence of steps to reach target amount
    of water.
    """
    queue = deque([[(0, 0, [])]])
    visited = set()
    while queue:
        a, b, steps = queue.popleft()
        if (a, b) in visited:
            continue
        visited.add((a, b))
        if a == z or b == z:
            return steps
        queue.append((x, b, steps + ['fill jug 1']))
        queue.append((a, y, steps + ['fill jug 2']))
        queue.append((0, b, steps + ['empty jug 1']))
        queue.append((a, 0, steps + ['empty jug 2']))
        amount = min(a, y - b)
        queue.append((a - amount, b + amount, steps +
                      ['pour jug 1 into jug 2']))
        amount = min(x - a, b)
        queue.append((a + amount, b - amount, steps +
                      ['pour jug 2 into jug 1']))
    return None

steps = water_jug_problem(4, 3, 2)
if steps:
    print('\n'.join(steps))
else:
    print('No solution found.')
```

Output:

Path of states by jugs followed is:

```
0, 0
0, 3
3, 0
3, 3
4, 2
0, 2
```

Result:

Hence, the simple python program for water jug problem is done.

Experiment - 13

Write the Python Program for Cryptarithmetic Problem.

Program:

```
def solve_cryptarithmetic(puzzle):
words = puzzle.split()
letters = set(''.join(words))
if len(letters) > 10:
    return None
for perm in permutations(range(10), len(letters)):
    mapping = dict(zip(letters, perm))
    if all(mapping[word[0]] != 0 for word in words):
a = sum(mapping[c] * (10**(len(word)-i-1))
for word in words for i,c in enumerate(word[::-1]))
    b = a // 10
    a % = 10
    c = sum(mapping[c] * (10**(len(words[2])-i-1)
for i,c in enumerate(word in words[:-1]):
    return mapping
return None.
puzzle = "SEND + MORE = MONEY"
mapping = solve_cryptarithmetic(puzzle)
if mapping:
    print(mapping)
    print(puzzle.translate(str.maketrans(mapp
else:
    print("no solution found.")
```

Output:

After defining the solve_cryptarithmetic function, the code provides an example usage using a cryptarithmetic puzzle: "SEND + MORE = MONEY." It calls the solve_cryptarithmetic function with the puzzle string and stores results in mapping variable.

If solution is found, it prints the mapping dictionary and puzzle string with letters replaced by their corresponding digits using the translate method.

If no solution is found, it prints "No solution found".

RESULT:

The output was verified.

Experiment - 14

Write the python program for Missionaries Cannibal problem.

Program:

```
from collections import deque

def is_valid(state):
    m1, c1, _, m2, c2, _ = state
    return all(x >= 0 for x in (m1, c1, m2, c2)) and
        (m1 == 0 or m1 >= c1) and (m2 == 0 or m2 >= c2)

def get_successors(state):
    m1, c1, b, m2, c2, d = state
    moves = [(1,0), (2,0), (0,1), (0,2), (1,1)]
    successors = []
    for m, c in moves:
        if b:
            new_state = (m1-m, c1-c, 0, m2+m, c2+c, 1)
        else:
            new_state = (m1+m, c1+c, 1, m2-m, c2-c, 0)
        if is_valid(new_state):
            successors.append(new_state)
    return successors

def bfs():
    start, goal = (3,3,1,0,0,0), (0,0,0,3,3,1)
    queue = deque([[start, []]])
    visited = set()
    while queue:
        state, path = queue.popleft()
        if state == goal:
            return path + [state]
        if state not in visited:
            visited.add(state)
            queue.extend((s, path + [state]) for s in
get_successors(state))
    return []
solution = bfs()
for s in solution:
    print(s)
```

Output:

```
(3, 3, 1, 0, 0, 0)
(2, 2, 0, 1, 1, 1)
(3, 2, 1, 0, 1, 0)
(1, 1, 0, 2, 2, 1)
(2, 1, 1, 1, 2, 0)
(0, 0, 0, 3, 3, 1)
```

RESULT:

The program was executed and output was found to be correct.

Experiment : 15

Write the Python Program for Vacuum cleaner Problem.

Program:

```
from typing import List, Tuple

def get_successors(state: Tuple[Tuple[int, int], List[List[int]]]) -> List[Tuple[Tuple[int, int], List[List[int]]]]:
    position, grid = state
    successors = []
    for dx, dy in [(0,0), (0,-1), (1,0), (-1,0)]:
        x,y = position
        new_x, new_y = x+dx, y+dy
        if 0<=new_x<len(grid) and 0<=new_y<len(grid[0]):
            new_grid = [row[:] for row in grid]
            if new_grid[new_x][new_y]==1:
                new_grid[new_x][new_y]=0
                successors.append(((new_x,new_y),new_grid))
    return successors

def dfs(state: Tuple[int,int], List[List[int]],
        visited: set() -> bool:
    if all(all(cell==0 for cell in row) for row in
        return True.
    visited.add(state)
    for successor in get_successor(state):
        if successor not in visited:
            if dfs(successor, visited):
                return True.
    return False
if __name__=='__main__':
    grid = [[0,1,1,1],
            [0,0,1,0],
            [1,0,0,1],
            [1,1,1,0]]
    initial_state = ((0,0), grid)
    if dfs(initial_state, set()):
        print("The room can be cleaned.")
    else:
        print("The room cannot be cleaned.")
```

Output:

```
vaccum in location new, 0,0
cleaned 0 0
vaccum in location new, 0 2
cleaned 0 2
vaccum in location new, 2 0
cleaned 2 0
room is clean now, Thanks for using : A.SAFARI1CLEANER
Performance = 68.75%.
```

RESULT: Hence, simple python program for vaccum cleaner problem was done.

Experiment : 16

Write the Python Program to implement BFS.

AIM:

To write a python program to implement BFS.

ALGORITHM:

1. Initialing queue & visited set.
2. Enqueue start node, mark visited.
3. While queue : dequeue, enqueue unvisited
4. Repeat.

PROGRAM:

```
from collections import deque
def bfs(adj_list, start, end):
queue = deque([[start, [start]]])
visited = set()
while queue:
    node, path = queue.popleft()
    if node == end:
        return path
    if node in visited:
        continue
    visited.add(node)
    for neighbor in adj_list[node]:
        if neighbor not in visited:
            queue.append((neighbor, path+[neighbor]))
return none.
if __name__ == '__main__':
    graph = {0:[1,2],
             1:[0,2,3],
             2:[0,1,3],
             3:[1,2,4],
             4:[3]}
    start_node = 0
    end_node = 4
    shortest_path = bfs(graph, start_node, end_node)
    if shortest_path is not none:
        print(f"The SP b/w {start_node} and {end_node} is {shortest_path}")
    else:
        print(f"There is no path b/w {start_node} and {end_node}")
```

OUTPUT:

The BFS Traversal for graph is as follow

3 1

RESULT:

Thus, the program was executed and found output successfully.

Experiment -17

Write the Python Program to implement DFS.

AIM:

To write a Python program to implement DFS.

ALGORITHM:

1. Initialize a visited set.
2. call DFS on start node.
3. Mark node visited, recurse on unvisited neighbors.
4. Repeat until all nodes are visited.

PROGRAM:

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

def dfs(graph, start, visited = None):
    if visited is None:
        visited = set()
    visited.add(start)
    for neighbor in graph[start]:
        if neighbor not in visited:
            dfs(graph, neighbor, visited)
    return visited
print(dfs(graph,'A')).
```

Output:

A D H C F J G K B E I

RESULT:

Thus the program was executed successfully using DFS.

Experiment - 18

Write the Python to Implement Travelling Salesman Problem.

AIM:

To write a python program to implement travelling salesman problem.

ALGORITHM:

1. Set best distance = ∞ and best route = [].
2. For each permutation of cities:
   - → compute total path distance.
   - → update best distance & route if current is short
3. Return best distance & route.

PROGRAM:

```
import numpy as np def tsp(cities):
    n = len(cities)
    dist_matrix = np.zeros((n,n))
    for i in range(n):
        for j in range(n):
            if i != j:
                dist_matrix[i][j] = np.linalg.norm
                    (cities[i] - cities[j])
    tour = [0]
    unvisited - cities = set(range(1,n))
    total - distance = 0
```

```
while unvisited - cities:
    current - city = tour[-1]
    nearest - city = min(unvisited - cities, key = lambda
    city: dist_matrix[curr - city][city])
    tour.append(near - city)
    total - dist += dist - matrix[curr - city][near - city]
tour.append(0)
total - distance += dist - matrix[tour][-2][0]
return tour, total - distance
```

Output:

Path found : ['A', 'F', 'G', 'I', 'J']

RESULT:

Thus, the program was executed and output was found successfully.

Experiment - 19

Write the python program to implement A* algorithm.

PROGRAM:

```
import heapq

def astar(start, goal, neighbors_fn, heuristic_fn):
    frontier = [(0, start)]
    came_from = {start: None}
    cost_so_far = {start: 0}

    while frontier:
        current = heapq.heappop(frontier)
        if current == goal:
            break
        for neighbor in neighbors_fn(current):
            new_cost = cost_so_far[current] + 1
            if neighbor not in cost_so_far or new_cost <
cost_so_far[neighbor]:
                cost_so_far[neighbor] = new_cost
                heapq.heappush(frontier, (new_cost +
heuristic_fn(neighbor, goal), neighbor))
                came_from[neighbor] = current

path = []
while current:
    path.append(current)
    current = came_from[current]
return path[::-1], cost_so_far[goal]
```

OUTPUT:

Assuming empty 5x5 grid with only cardinal moves allowed,

Shortest path: [(0,0), (1,0), (2,0), (3,0), (4,0), (4,1), (4,2), (4,3), (4,4)]

cost: 8

RESULT:

Thus, the program was executed successfully.

Experiment -20

Write the python program for map coloring to implement CSP.

AIM:

To write a python program for map coloring to implement CSP.

PROGRAM:

```
class CSP:
    def __init__(self, vars, domains, cons):
        self.vars, self.domains, self.cons = vars, domains, cons
    def solve(self, assign = {}):
        if len(assign) == len(self.vars):
            return assign
        var = next(v for v in self.vars if v not in assign)
        for val in sorted(self.domains[var], key=lambda:
            self.conflicts(var, v, assign)):
                new = assign.copy(); new[var] = val
                if self.consistent(new):
                    sol = self.solve(new)
                    if sol: return sol
        return None
    b in self.cons if var in (a, b))
    def consistent(self, assign):
        return all(assign[a] != assign[b] for a, b in
        self.cons if a in assign and b in assign)
```

```
def map-coloring():
    vars = ['WA', 'NT', 'SA', 'Q', 'NSW', 'V', 'T']
    domains = {v: ['r', 'g', 'b'] for v in vars}
    cons = [('WA', 'NT'), ('WA', 'SA'), ('NT', 'SA'),
            ('NT', 'Q'), ('SA', 'Q'), ('SA', 'NSW'), ('SA', 'V'),
            ('Q', 'NSW'), ('NSW', 'V')]
    sol = CSP(vars, domains, cons).solve()
    print(sol if sol else "No solution found")
map-coloring().
```

OUTPUT:

Solution Exists: Following are assigned colors.

1 2 3 2

RESULT:

Hence, simple python program for map-coloring to implement CSP is done.

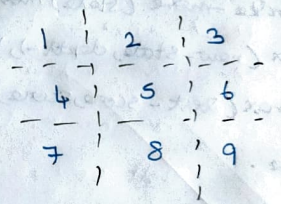
Experiment - 21

Write the Python Program for Tic Tac Toe game.

PROGRAM:

```
board = [' '] * 9
def print_board():
    for i in range(0,9,3):
        print('|'.join(board[i:i+3]))
        if i < 6: print('-' * 5)
def player_move(icon):
    player = 1 if icon == 'X' else 2
    print(f"Player {player}'s turn")
    choice = int(input("Enter your move (1-9): "))-1
    if board[choice] == ' ':
        board[choice] = icon
    else:
        print("That space is taken!")
def is_victory(icon):
    win_patterns = [(0,1,2),(3,4,5),(6,7,8),(0,3,6),
(1,4,7),(2,5,8),(0,4,8),(2,4,6)]
    return any(board[a] == board[b] == board[c] ==
icon for a,b,c in win_patterns)
def is_draw():
    return ' ' not in board
while True:
    print_board()
    player_move('X')
    if is_victory('X'):
        print("X wins! Congratulations!")
        break
    print_board()
    player_move('O')
    if is_victory('O'):
        print("O wins! Congratulations!")
        break
    if is_draw():
        print("It's a draw!")
        break
```

OUTPUT:



RESULT:

Hence, the simple Python Program for Tic Tac Toe game was executed successfully.

Experiment -22

Write the python Program to implement Minimax algorithm for gaming.

PROGRAM:

```
def evaluate (state):
def minimax (state, depth, player):
if depth == 0 or game-over (state):
    return evaluate (state)
if player == "x":
    best-score = float ('-inf')
    for move in get-possible-moves (state):
        new-state = make-move (state, move, player)
        score = minimax (new-state, depth-1, "o")
        best-score = max (best-score, score)
    return best-score.
else:
    best-score = float ("inf")
    for move in get-possible-moves (state):
        new-state = make-move (state, move, player)
        score = minimax (new-state, depth-1, "x")
        best-score = min (best-score, score)
    return best-score.
```

OUTPUT:

```
root = TreeNode (0)
root.left = TreeNode (3)
root.right = TreeNode (0)
root.right.left = TreeNode (0)
root.right.right = TreeNode (0)
root.right.left.left = TreeNode (-3)
root.right.right.right = TreeNode (4).
```

RESULT:

Thus, the program was executed successfully.

Experiment - 23

Write Python Program to implement Alpha & Beta Pruning algorithm for gaming.

PROGRAM:

```
def evaluate(state):
def alpha_beta_Pruning(state, depth, a, b, player):
    if depth == 0 or game_over(state):
        return evaluate(state)
    if player == "x":
        best_score = float('-inf')
        for move in get_possible_moves(state):
            new_state = make_move(state, move, player)
            score = alpha_beta_Pruning(new_st, depth-1, a, b, 'o')
            best_score = max(best_score, score)
            a = max(a, score)
            if b <= a:
                break
        return best_score.
else:
    best_score = float('inf')
    for move in get_possible_moves(state):
        new_state = make_move(state, move, player)
        score = alpha_beta_Pruning(new_st, depth-1, a, b, 'x')
        best_score = min(best_score, score)
        beta = min(beta, score)
        if beta <= alpha:
            break
    return best_score.
```

OUTPUT:

X	O	X
O	X	X
O	X	O

x → maximizing player

o → minimizing player

Best Score : 10

RESULT:

Thus, the program was executed successfully.

Experiment - 24

Write the python program to implement Decision Tree.

PROGRAM:

```
# import necessary libraries
from sklearn.tree import DecisionTreeClassifier
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score.

iris = load_iris()
X_train, X_test, y_train, y_test = train_test_split
    (iris.data, iris.target, test_size=0.3, random_state=42)

clf = DecisionTreeClassifier()
clf.fit(X_train, y_train)
y_pred = clf.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy).
```

output:

```
arr(['yes', 'yes', 'no', 'yes', 'no', yes', yes',
     'no', 'no', 'yes', 'no', 'no', yes', 'no',
     'yes', 'no', 'yes', 'no', 'no', 'yes', 'yes',
     'no', 'yes', 'yes', 'no', 'yes', 'no', 'no',
     'yes', 'no', 'no', 'yes', 'no', 'no', 'yes', 'no',
```

RESULT:

Thus the program was executed successfully.

Experiment - 25

Write the Python Program to implement Feed forward neural Network.

PROGRAM:

```
#import necessary libraries
from keras.models import sequential
from keras.layers import Dense
from keras.optimizers import SGD
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split.

iris = load_iris()
x_train, x_test, y_train, y_test = train_test_split
    (iris.data, iris.target, test_size=0.3, random_state=42)
model = sequential()
model.add(Dense(10, input_dim=4, act='relu'))
sgd = SGD(lr=0.01)
model.compile(loss='categorical_crossentropy',
    optimizer=sgd, metrics=['accuracy'])
model.fit(x_train, y_train, epochs=50, batch_size=10, verbose=
scores = model.evaluate(x_test, y_test, verbose=0)
print("Accuracy: %.2f%%" % (scores[1]*100))
```

OUTPUT:

The model is trained for 50 epochs using a batch size of 10. The training data (X_train and y_train) is used for training.

RESULT:

Thus, the program was executed successfully.