Date

Expt. No. Experiment : 1 Page No.

- Study the available platform for AI programming.

Q: AIM: Study about python programming language:

Q.1 What was the problem in privious programming language so that python was made?

Ans The privious programming language way ABC which was not extensible, and Couldn't easily add features or adapt new features from other languages like C, C++ etc.

- Exception handeling was missing
- C, shell cscripting were harder to learn and less readable syntax.
- python combines ABC's clear syntax with powerful features like flexible data types & exception handeling.

Q.2 Explain the development of python over the years.

Ans (i) > late 1980's Guido van Rossum begin working on python as a hobby thing during the christmas holidays in Netherlands.

SAMAY

Teacher's Signature

Date

Expt. No. Page No.

(ii) 1991 :- Python version (0.9.0) first public release, featuring classes, functions, exception handeling and fundamental data types like lists & dictionaries.

(iii) 1994 : (python 1.0) : official public release, Concept such as lambda function, map | filter | reducer & improved exception handeling, solidified concepts.

(iv) 2000 (python 2.0) : feature like list comprehension memory management, garbage collector, unicode support

(v) 2008 (python 3.0) :- Backward incompatible release. focus on removing inconsistencies from 2.x, improved unicode support & simplifying syntax.

(vi) 2010 (python 2.7) final major release in python 2.x line, support legacy codebases until officially end of life in 2020.

(vii) python 3.5 (2015) :- async await keyword for asynchronous programming.

(viii) python 3.5 & 3.8 (2016-2019) :- Bring f-strings, the walrus operator (:=) & enhancements to types..

SAMAY

Teacher's Signature

Date

Expt. No. Page No.

(ix) python 3.9 - 3.12 (2020-2023) Adds dictionary merge/update operator, pattern matching, timezone management.

(x) python 3.13 (2024) Enhanced syntax, performance & library support.

Q: Usages of python:

(i) Data Science & AI: Python is top language for DS & AI & ML pandas, tensorflow, pytorch

(ii) Web development: Django, Flask, FastAPI, backend web dev & API dev, Python. & rapid prototyping & build scable apps.

(iii) Automation & scripting: Automating repetitive, tasks, system scripting process workflows in IT devops.

(iv) sdw & App development: Quick dev of desktop mobile, cross platform apps.

(v) IOT (internet of things): Python powers IOT devices due to flexibility & ease of interfacing with hardware & sensors.

(vi) Big Data & Scientific Computing: Used for statistics maths, simulation, visual, by

SAMAY

Teacher's Signature

Date

Expt. No. Page No.

Numpy & Matplotlib.

<viii> Cyber Security

<viii> First Choise Language.

Q:4 Problem in python:-

- Slower than C, C++
- Weak in Mobile & Browser based application.
- More memory consumption, not suitable for memory
- Not ideal for low level system (driver)
- Indentation based system lead to hidden bugs.
- Error detection at runtime.

Q:5 What are the versions of python?

Ans python (0.9.0)
python (1.0)
python (2.x)
python (3.x) - 3.6 (2016)
3.7 (2017)
3.8 (2019)
3.9 (2020)
3.10 (2021)
3.11 (2022)
3.12 (2023)
3.13 (2024)

SAMAY

Teacher's Signature

Date

Expt. No. Page No.

Q.6 Why python is used for AI programming?

Ans → Simple syntax like that focus of AI logic rather than complex code.

→ Runs smoothly on windows, Macos, Linux, etc.

→ large community supports

→ Easily integrates with C, C++ for performance critical tasks.

→ Numpy & Matplotlib and data manipulation

Q.7 Online & offline Compilers of python and environment specification required for running Python?

Ans • Operating system specification:-

1. windows (8 and above)
2. Mac OS
3. Linux (Mostly all versions)

• Offline & Editors & IDES:-

1. pycharm
2. Vscode
3. IDLE
4. Spyder
5. Jupyter Notebook
6. Sublime text
7. Atom

SAMAY

Teacher's Signature

Date

Expt. No. Page No.

Online platform:-

- Replit
- Python Anywhere
- Google colab
- Jupyter Notebook
- Vscode etc.

Q: 8 How many libraries are there in Python?

Ans As of 2025, there are 2,00,000 libraries

Q: 9 Is Python a Compiled or Interpreted language?

Ans Interpreted language:

Q: 10 Size of python:

Ans
- Install size:
  - Windows: 25 MB (64 bit)
  - MacOS - 43 MB
  - Linux: 20-26 MB
- On disk size: 150 - 350 MB
  - → Python - 3.12 on Linux - 270 - 350 MB
  - → Python 2.7 with additional package can be 3.5 GB or more.
  - → standered anaconda for Python 3 GB.

SAMAY

Teacher's Signature

Date

Expt. No. Experiment : 4 Page No.

=> Aim: Write a program to implement Tic-Tac-Toe game.

=> Software and Hardware requirments:-

- Software requirments:- An OS suitable Compiler or Interpiter for the python programming language.
- Hardware requirments:- Minimal Intel i3 or equivalent.
  - Ram - 4 GB
  - Storage: 2 MB free space
  - Input Devise - Keyboard (player input)
  - Output devise - Monitor / Display.

=> Algorithum's

1. Start the program
2. Initialize a 3x3 board with empty cells.
3. Assign a symbol to players (player 1 = X, player 2 = O)
4. Display the empty board
5. Repet until the game ends.

SAMAY

Teacher's Signature

Expt. No. Date

Page No.

- Ask the current player to enter the position.
- Check if the position is valid.
- Place the player symbol in the cell.
- Display the updated board
- Check for win condition
- if a player wins display the message
- if all the moves are filled without a winner, declear the game draw.

6: End the program.

⇒ Code:

```
board = [' ']*9
player = 'X'

def print_board():
    print(board[0], "|", board[1], "|", board[2])
    print("--+---+--")
    print(board[3], "|", board[4], "|", board[5])
    print("--+---+--")
    print(board[6], "|", board[7], "|", board[8])

def check_winner(p):
```

SAMAY

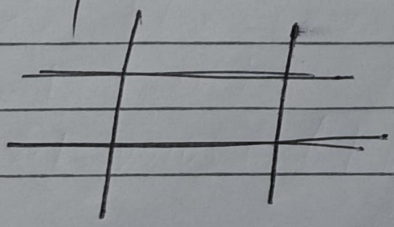
Teacher's Signature

Date

Expt. No. Page No.

```
wins = [(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] !=
           for a,b,c in wins)

for turn in range(9):
    print_board()

    pos = int(input(f"player {player}, enter
                    (1-9):"))

    if check_winner(player):
        print_board()
        print("player", player, "wins!")
        break
    player = 'O' if player == 'X' else 'X'
    else:
        print("Invalid! try again")

else:
    print_board()
    print("It's a draw")
```

⇒ Output / Result :



player's X's turn
Enter row : 0
Enter col : 0

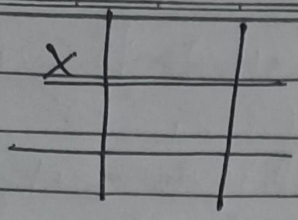
SAMAY

Teacher's Signature

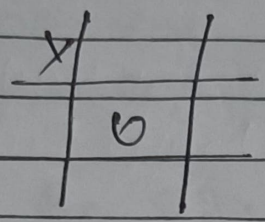
Date

Expt. No.

Page No.

⇒ 

player O's turn
Enter row : 1
Enter Col : 1

⇒ 

⇒ player X wins!

Date

Expt. No. Ex: 2

Page No.

Aim: Implement BFS using Python

<2a> Implement BFS using fixed inputs.
<2b> Implement BFS for user inputs
<2c> Implement BFS using fringes
<2d> Implements BFS for given query.

Ex: <2.a>

Aim: Implement BFS using fixed input.

(2a)-1 H/w & S/w Requirments -

- Computer
- Notebook
- Operating system
- Python 3.x

(2a)-2 Algorithm:

(i) Begin from the starting node.

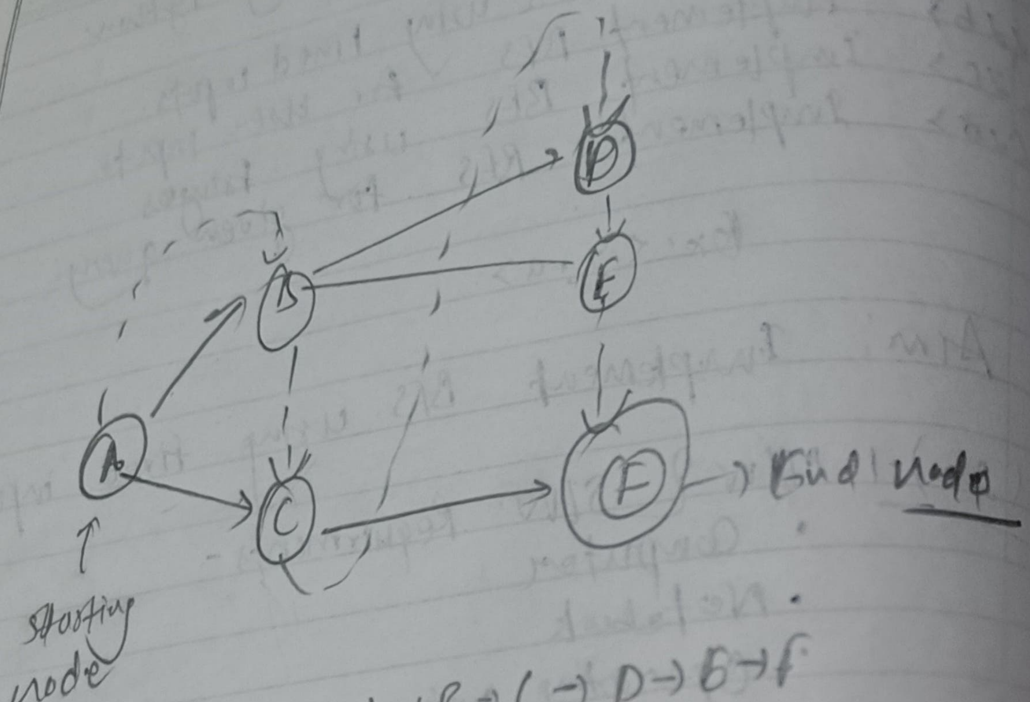
(ii) Find all nodes connected to starting node.

(iii) Chose one dir either left or right.

(iv) For each node in the current layer identify all of its neighbours that you haven't visited yet from the

SAMAY

Teacher's Signature



Path :- A → B → C → D → E → F

Fig: BFS using fixed Input!

Expt. No. Date

Page No.

Choose directions

(v) Once all nodes in the current layer have been fully explored, move to the next layer.

(vi) Continue this process until you either run out of nodes to explore

(2a)-3 Source Code-

```
from collection import deque:

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

def bfs (graph start):
    visited = set()
    queue = deque ([start])
    visited.add (start)
    print ("BFS traversal : ", end).
    while queue
        vertex = queue.popleft()
        print (vertex, end = " ")
        for neighbour not in visited
```

Output: A → B → C → D → E → F

SAMAY

Teacher's Signature

Date

Expt. No. (Experiment-26:) Page No.

Aim: Implement BFS for user input.

26.1 S/w & H/w Requirments:-
- Computer
- OS
- Python 3.x

26.2 Algorithm:

(i) start ib
(ii) Input the no. of vertexes in the graph.
(iii) Input adjacenty matrix of graph.
(iv) Input the starting vertex.
(v) Initalize queue & mark all as unvisited
(vi) Repeat until the queue is empty.

(a) Dequen the insert vertex
(b) If u == goal
(c) Use :i) for goal each adjacent vertex (v) fo. o'
- -) If visited 'is' t
- -) mark v as visited
- -) Enqueue(v) into queue.

(vii) stop

26.3 Source Code:-

```
from collection import deque
```

SAMAY

Teacher's Signature

Date

Expt. No.

Page No.

```
def bfs (graph, start)
    visited = set()
    queue = deque ([start])
    visited.add (start)
    print ("BFS Traversal : ", end = " ")

    while queue:
        vertex = queue.popleft()
        print (vertex, end = " ")
        for neighbour in graph [vertex]:
            if neighbour not in visited:
                visited.add (neighbour)
                queue.append (neighbour)

if main - program:
    graph = {}
    n = int (input ("Enter no. of vertex: "))
    for i in range (n):
        vertex = input (f"Enter name of vertex [{i}]: ")
        graph [vertex] = []
    e = int (input ("Enter no. of edges"))
    for i in range (e):
        u = input (f"Enter edge {i} start vertex: ")
        v = input (f"Enter edge {i} end vertex): ")
```

SAMAY

Teacher's Signature

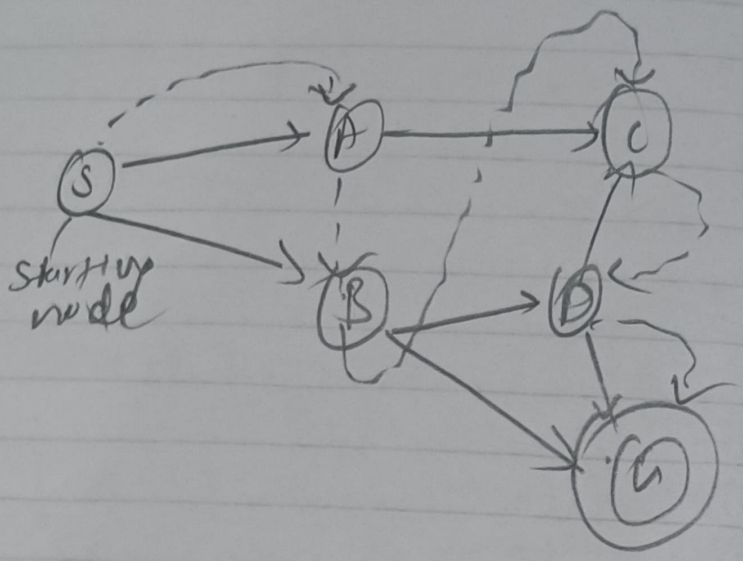


fig 2b

BFS for user Input

path: S → A → B → C → D → G

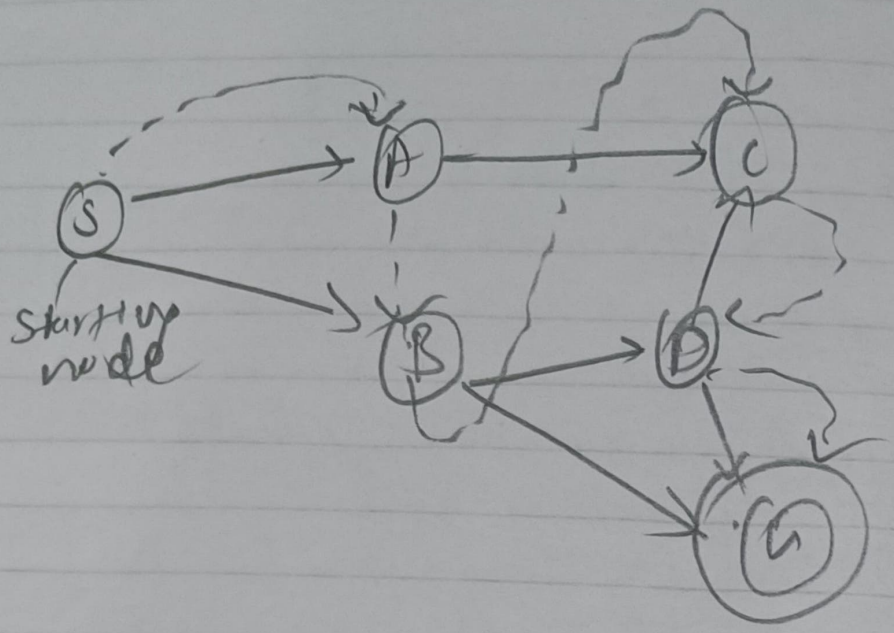


fig 2b

BFS for User Input

path: S → A → B → C → D → G

Date

Expt. No. Page No.

```
graph[u].append(v)
graph[v].append(u)
start_vertex = input("Enter starting vertex:")
bfs(graph, start_vertex)
```

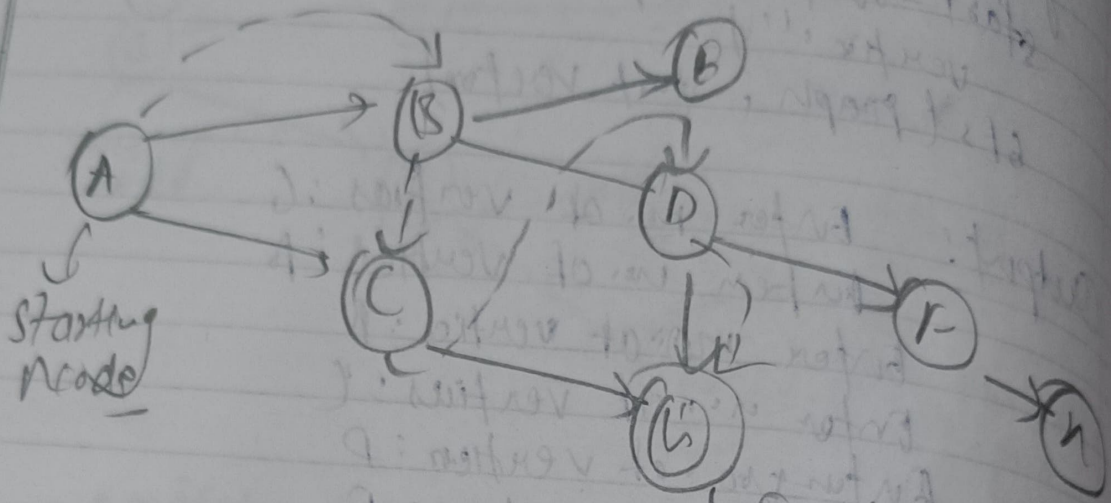
Output:

```
Enter no. of vertices: 6
Enter no. of vertices: A
Enter no. of vertices: B
Enter no. of vertices: C
Enter no. of vertices: D
Enter no of vertices: E
Enter no. of verte = F
Enter :: :: Edges 6
:: :: :: start vertex: A
:: :: end :: 5 B
:: :: start :: 5 A
:: :: end :: : C
:: :: start :: : B
:: :: end :: : B
:: :: stat : : E
:: :: end : : C
:: :: start : : A
:: :: end : f
:: :: starting vertex : A
```

BFS traversal) ABCDEF

SAMAY

Teacher's Signature



path: A → B → C → D → G

Fig: BFS using open & closed strings in python

Date

Expt. No. "Experiment-2C" Page No.

Aim: Implementing BFS using open & closed fringe.

<2C>1 H/w & s/w requirements:
- Computer
- Python 3.x
- Operating system

<2C>2

step	open fringe (BND)	Closed fringe (Expand)
1.	S	S ≠ Goal
2.	AS, BS	S, A, ≠ Goal
3.	BAS, CAS, DAS	A, B ≠ Goal
4.	CAS, DAS, DBS, EBS	B, C ≠ Goal
5.	DAS, DBS, EAS	C, D ≠ Goal
6.	EBS	D, E ≠ Goal

Soln path: S → B → G

b = 2

T.C = $O(b^d)$ = S.C = $O(b^d)$

BFS optimal & complete

total node = $\left(\frac{b^{(d+1)} - 1}{b-1}\right)$

SAMAY

Teacher's Signature

Date

Expt. No. Experiment :- 2D Page No.

Aim: Implement BFS for given query:

2.01 S/W & H/W requirement:
- Computer
- Operating system
- Network
- python 3.x

2.02

Step	open fringe [GNO]	closed fringe [expend]
1.	A	
2.	BA, CA	A,
3.	CA, EBA, DBA	B,
4	EBA, DBA, DCA, GBA	C,
5.	FCA	D,

Solution: path: A → C → F

b = 02

d = 02

T.C = O(b^d) = O(2^2) = O(4)

S.C = O(bd) = O(4)

Date

Expt. No. Experiment 5 Page No.

→ Aim: Write a program to solve 5-puzzle in python.

→ Software & Hardware Requirement:-

Hardware:- - Basic Dual Core processor
- RAM - 2GB or Higher
- Minimal storage (16GB)
- Input / output devices

Software:- - Operating system windows or mac.
- Python 3.x installed
- IDE / text Editor.

Logic:-

→ First we need to decided whether we are solving it using Informed search technique or Uninformed Search.

→ If we solve it using Uniformed Search then:- it solve with no domain specific knowledge about the problem.

→ We use BFS technique here to find the path to our goal exploring all the paths - LEFT, RIGHT, UP, DOWN & remembering all the paths to goal.

SAMAY

Teacher's Signature

Expt. No. Date

Page No.

→ Here we use informed search technique using A* algorithm which incorporates domain specific knowledge to move better decision about which path to explore next.

→ Logic for Implementation:

1. The puzzle is represented as a 3x3 grid with numbers 1-8 and one blank or (0)

2. The objective is to slide tiles to reach the goal configuration.

3 A* Search is used:-
- → f(n) = g(n) + h(n)
- → g(n) = cost to reach the current state.
- → h(n) heuristic = Manhatten distance of each tile from it's goal position.

4 At each step, expand the state with lowest f(n) until the goal is find.

→ Algorithm:
- → start with Initial puzzle configuration
- → Compute f s g th for the start state
- → Insert it into a priority queue (minheap)
- → while the queue is not empty.

SAMAY

Teacher's Signature

Expected Output

Solution Found in 14 moves:-

(1,2,3)
(4,0,5)
(6,7,8)

↓ RIGHT

(1,2,3)
(4,5,0)
(6,7,8)

↓ DOWN

(1,2,3)
(4,5,8)
(6,7,0)

↓ LEFT

(1,2,3)
(4,5,8)
(6,0,7)

↓ LEFT

(1,2,3)
(4,5,8)
(6,7,0)

(1,2,3)
(0,5,8)
(4,6,7)

↓ Right

(1,2,3)
(5,0,8)
(4,6,7)

↓ DOWN

(1,2,3)
(5,6,8)
(4,0,7)

↓ Right

(1,2,3)
(5,6,8)
(4,7,0)

↓ UP

(1,2,3)
(5,6,0)
(4,7,8)

↓ LEFT

(1,2,3)
(5,0,6)
(4,7,8)

(1,2,3)
(0,5,6)
(4,7,8)

↓ DOWN

(1,2,3)
(4,5,6)
(0,7,8)

↓ Right

(1,2,3)
(4,5,6)
(7,0,8)

↓ Right

(1,2,3)
(4,5,6)
(7,8,0)

↓

Goal State REACHED

Date

Expt. No. Page No.

→ Remove the state with the lowest f(n).
→ If it matches the goal, stop and return the path.

→ Otherwise, generate all valid moves of blank tile (0)
→ for each child state, compute g, h and f. Then add it to queue.
→ track visited states to avoid repetitive
→ print the solution path when goal is reached.

Source Code:

```
import heapq

def print_puzzle(state):
    for i in range(0, 9, 3):
        print(state[i:i+3])
    print()

def manhattan(state, goal):
    distance = 0
    for i in range(9):
        if state[i] != 0:
            goal_index = goal.index(state[i])
            x1, y1 = divmod(i, 3)
            x2, y2 = divmod(goal_index, 3)
            distance += abs(x1-x2) + abs(y1-y2)
    return distance
```

SAMAY

Teacher's Signature

Date

Expt. No. Page No.

```
def get_moves(state):
    moves = []
    zero_index = state.index(0)
    row, col = divmod(zero_index, 3)

    direction = [(-1,0), (1,0), (0,+1), (0,-1)] # down
                                                # up, left, right

for dr, dc in direction:
    new_row, new_col = row + dr, col + dc
    if 0 <= new_row < 3 and 0 <= new_col < 3:
    new_index = new_row * 3 + new_col
    new_state = list(state)
    moves.append(tuple(new_state))
return moves.

def astar(start, goal):
    pq = []
    heapq.heappush(pq, (manhattan(start, goal),
        0, start, []))
    visited = set()

while pq:
    f, g, state, path = heapq.heappop(pq)
    if state in visited:
        continue.
    visited.add(state)
        if state == goal:
```

SAMAY

Teacher's Signature

Date

Expt. No. Page No.

```
return path + [state]

for move in get_moves(state):
    if move not in visited:
        new_g = g + 1
        new_f = new_g + manhattan(move, goal)
        heapq.heappush(pq, (new_f, new_g, move,
            path + [state]))

if solution:
    print("Solution found in", (len(solution)
        - 1, "moves"))

    for step in solution:
        print_puzzle(step)
else:
    print("NO solution found")
```

SAMAY

Teacher's Signature

Date

Expt. No. Experiment: 6 Page No.

Aim: Write a program to implement travelling salesman problem (TSP) in python.

H/W & S/W Requirements:-
- Windows operating system (RAM 4GB, intel i3)
- python 3.12 version and new.

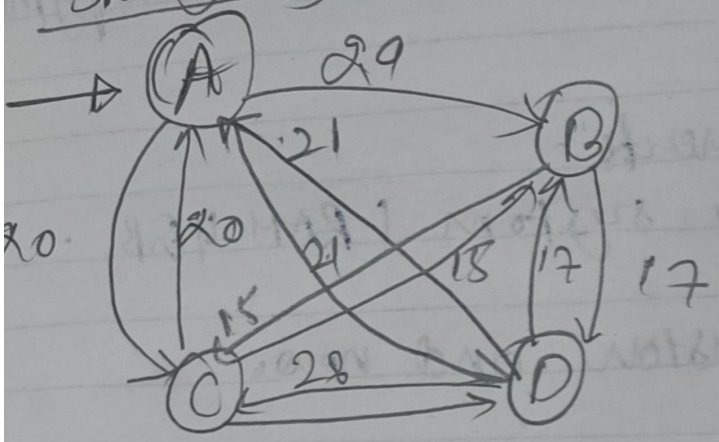
Logic:
i> Assumptions:-
- Each city is connected to every other city.
- the salesman starts from a particular distance (city) visits all exactly one and returns back.
- the goal is to find the route with minimum distance.

ii> Working:
- we represents the cities and distance b/w them in 2D Matrix.
- the problem checks all possible routes the can visit all cities.
- for each route it, calculates the total distance travelled
- the program then compares all distance

SAMAY The Power of Writing

Teacher's Signature

Starting edge



A → B = 29	B → A = 29
A → C = 20	C → A = 20
A → D = 21	D → A = 21
B → C = 15	C → B = 15
B → D = 17	D → B = 17
C → D = 28	D → C = 28

Ry. ——————

Date

Expt. No. Page No.

and find the shortest path

Algorithm:

- Start with a list of all the cities.
- Chose our city as the starting point.
- Make all possible routes that visit every city exactly once and come back to the start.
- for each route, calculate the total distance travelled.
- Distance the shortest route and its total distance.

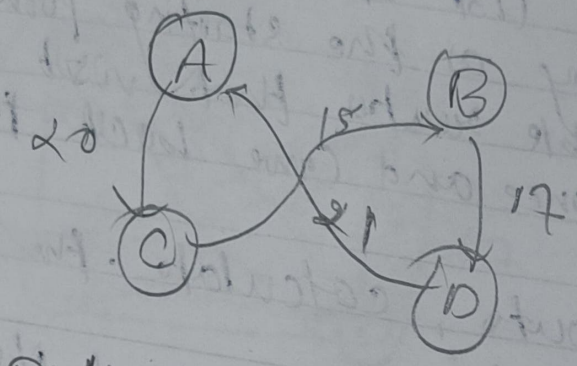
Source Code:-

```
from itertools import permutations:
  def total_distance (graph, path):
      distance = 0
      for i in range (len(path)-1):
          distance += graph [path [i]] [path [i+1]]
      distance += graph [path [-1]] [path [0]]
      return distance

def travelling salesman (graph)
      n = len(graph)
      cities = list (range (n))
      min_distance = float ('inf')
```

SAMAY

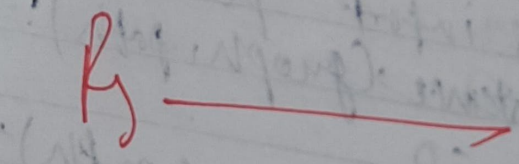
Teacher's Signature

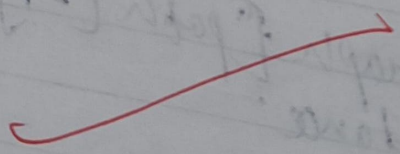
Solution path:



path: A → C → B → D → A

total distance = 73





Date

Expt. No. Page No.

```
for perm in permutation (cities)
    dist = total distance (graph, perm)

if dist < min distance:
    min_distance = dist
    min_path = perm
return min_path, min_distance

graph = [
    [0, 29, 20, 21],
    [29, 0, 15, 17],
    [20, 15, 0, 28],
    [21, 17, 28, 0]
]

path, distance = travelling_salesman(graph)
print("Optimal path:", path)
print("Minimum Distance:", distance)
```

Expected outcome

optimal path : (A, C, B, D, A)

minimum distance : 73

=

Expt. No. Date

Experiment - 7

Page No.

Aim: Write a program to implement water Jug problem using Python

H/W & S/W Requirements:
- operating system (windows)
- python 3.12 programming language
- personal Computer (RAM: 4GB, Storage: 100MB, & More)

Logic:

1> Assumption:
- There are two jugs Jug A and Jug B with capacity m and n liters.
- The goal is to measure exactly d. liters of water using these two jugs.
- the following operations can be performed:
  - Fill a jug completely
  - pour water from one jug is empty or the second jug is full

2> Working:
- The produce can be modele as a state space se produces.
- Each state is represented by a pair(x,y):

SAMAY

Teacher's Signature

Date

Expt. No. Page No.

- x = Current amount of value in Jug A.
- y = Current amount of water in Jug B.
- Start from initial state (0,0)
- The algorithm explores all possible state reachable by performing valid operations
- Do until it reaches a jug to desired amount

Algorithm

- There are two jugs - A and B with capacity m and n liters.
- The goal is to measure exactly d liters of water using these two jugs.

- The following operations can be performed:
  - Fill a jug completely
  - Empty a jug completely
  - pour water from one jug in empty until the ~~first~~ jug is to second jug is full.

(1) working:

Start with both the jugs empty (Jug A=0, Jug B=0)
- Decide your goal - how many liters of water you want to measure.
- Repeat the following steps until one jug

SAMAY

Teacher's Signature

Date

Expt. No. Page No.

has required amount.
- fill one of the jugs completely from water.
- pour water from one into another jug.
  - if the second jug becomes full, stop pouring.
  - if any jug becomes full, you can empty it needed.
- keep repeating these steps, in different ways taught amount.

Source Code:

```
from Collection import deque
def water_jug_problem(m, n, d):
    # m → capacity of jug A

visited = set()
queue = deque()
queue.append((0, 0))

while queue:
    a, b = queue.popleft()
    print(f"({a}, {b})")

    if a == d or b == d:
        print("In Goal unreached")
        return True
```


Teacher's Signature

Date

Expt. No. Page No.

visited.add((a,b))

possible_states = [
(m,b),
(a,n),
(0,b),
(a,0),
(a-min(a,n-b), b+min(a,n-b)),
(a+min(b,m-a), b-min(b,m-a))
]

for state in possible_states:
if state not in visited:
queue.append(state)

print("\n No solution possible")
return false

m = 4
n = 3
d = 2

water_jug_problem(m,n,d)

Expected Outcome:

(0,0) (0,1)
(4,0) (4,1)
(0,3) (2,3)
(4,3) Goal reached
(1,3)
(1,0)

SAMAY

Teacher's Signature

Expt. No. Experiment-9 Page No. Date

Aim: To implement the alpha-Beta pruning algo in Python to find the optimal values in a Minimax game tree by reducing node evaluations

Soft & Hard requirements:
- Python 3.x
- Basic text editor
- Any standered Computer.

Algorithm:
1. Use Minimum to evaluate a game tree where MAX tries to maximize and MIN tries to minimize the value.

2. Maintain two variable:
Alpha - best value MAX can guarantee
Beta - best value Min can guarantee

3. At each node:
- if it's MAX's turn:
  - Evaluate both children
  - Update Alpha
  - Stop if Alpha ≥ Beta
- if it's MIN's turn:
  - evaluate both children
  - update Beta

SAMAY

Teacher's Signature

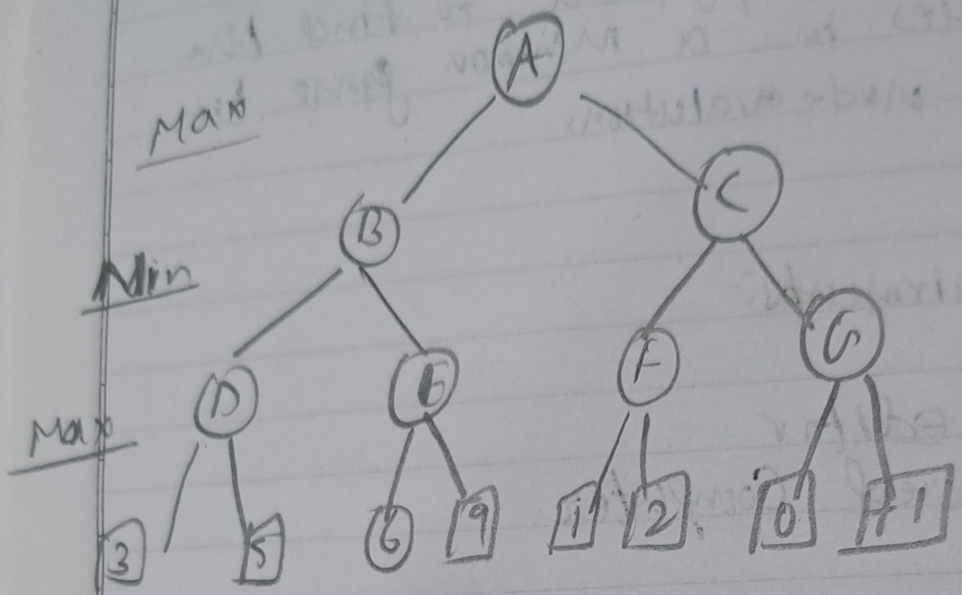


fig: optimal start from A

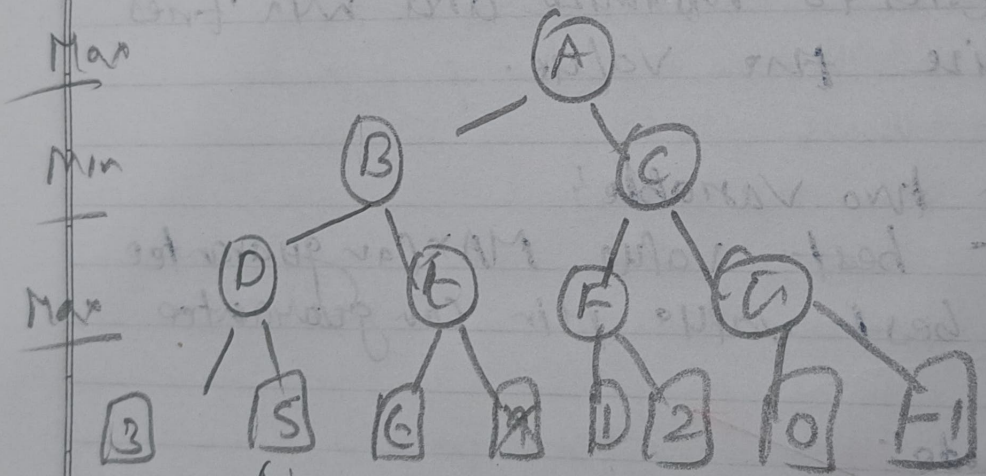


fig: 9 crossed out because was never computed.

Date

Expt. No.

Page No.

'Stop (prune) if Beta ≤ Alpha.

4. When depth reaches leaf, return the leaf value.
5. Continue until root returns the optimal value.

Source Code:

```
MAX = 1000
MIN = -1000

def minmax( depth, nodeIndex, maximizingPlayer,
    values, alpha, beta):

    if depth == 3:
    return values[nodeIndex]

    # Maximizer
        if maximizingPlayer:
            best = MIN

            for i in range(2):
                val = minmax(depth+1, nodeIndex*2+i,
                false, values, alpha, beta)
            best = max( best, val)
            alpha = max (alpha, best)
        if beta <= alpha:
```

SAMAY

Teacher's Signature

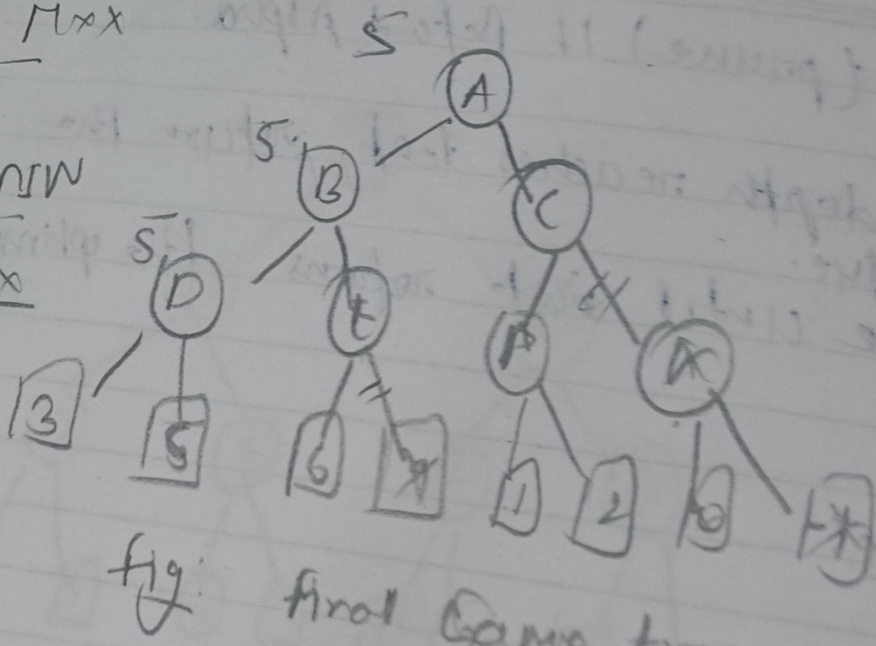


fig: final Game tree

Date

Expt. No.

Page No.

```
            break
        return best

    else:
        best = MAX
        for i in range (2):
            val = minimax (depth+1, nodeIndex*2+i,
            true, values, alpha, beta)

        if beta <= alpha:
            break
        return best

values = [3, 8, 6, 9, 1, 2, 0, -1]
print ("The Optimal values is: ", minimax(0, 0, True,
                values, MIN, MAX))
```

Output:

The optimal value is: 8

SAMAY

Teacher's Signature

Date

Expt. No. Experiment-8 Page No.

Aim: Write a program to implement Monkey Banana problem using python.

S/W Requirements:
- Any Computer with Python 3.x
- Text editor of your Choise.

Algorithm:
1. Read the location of Monkey banana and box.
2. Move the monkey to the box.
3. Move the box to the banana
4. Monkey climbs on the box
5. Monkey picks the banana
6. Display each step clearly.

Sourse Code:

```
i=0
def Monkey_go_box(monkey, box):
    global i
    i+=1
    print("step", i, ": Monkey goes from", monkey,
    "to", box)

def Monkey_move_box(box, banana):
    global i
    i+=1
```

SAMAY

Teacher's Signature

Expt. No. Date

Page No.

```
    print("Step", i, "-Monkey moves the
    box from", box, "to" banana)

def Monek-on-box():
    global i
    i+=1
    print("Step:", i, "-Monkey climbs
                on the box")

def Monkey-get-banana():
    global i
    i+=1
    print("Step", i, "Monkey picks the
    banana")

# Main program:

Monkey = input("Enter monkey location")
banan = input("Enter banana location")
box = input("Enter box location")

print("\n The steps are as follows:\n")
Monkey-go-box(monkey, box)
Monkey-move-box(box, banana)
Monkey-on-box()
Monkey-get-banana()
```

Output: Enter Monkey location: A

SAMAY

Teacher's Signature

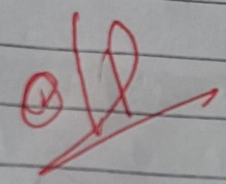
Expt. No. _____ Date _____

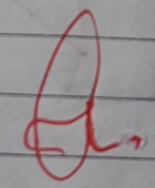
Page No. _____

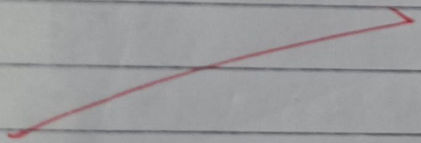
Enter banana location : C
Enter box location : B

The steps are as follows :-

Step : 1 - Monkey goes from A to B
Step : 2 - Monkey moves the box from B to C
Step : 3 - Monkey climbes on the box
Step : 4 - Monkey picks the banana







Expt. No. _____ Date _____

Page No. _____

Experiment-10

Aim: To study the firebird Robot.

features of FireBird Robot:-

→ Based on a two-microcontroller design to handle computer take & real-time operations

→ Features a highly scalable design that allows users to start with a bare platform & integrate various add-ons & sensor pods.

→ uses a different driver configurations for prices movement & turning.

→ Supports programming using industry standard tools like AVR studio & Atmel Studio allowing C/C++

→ The basic platform can be structurally upgrad to grow like tank drive or Hexapod.

Technical specifications & parts:-

Component types:-	part
- Microcontrollers	Master: Atmel ATMEGA2560
- Actuators	slave: ATMEGA8
- Locomotion	Two DC geared Motors
- Power supply	Differential driver with

SAMAY

Teacher's Signature _____

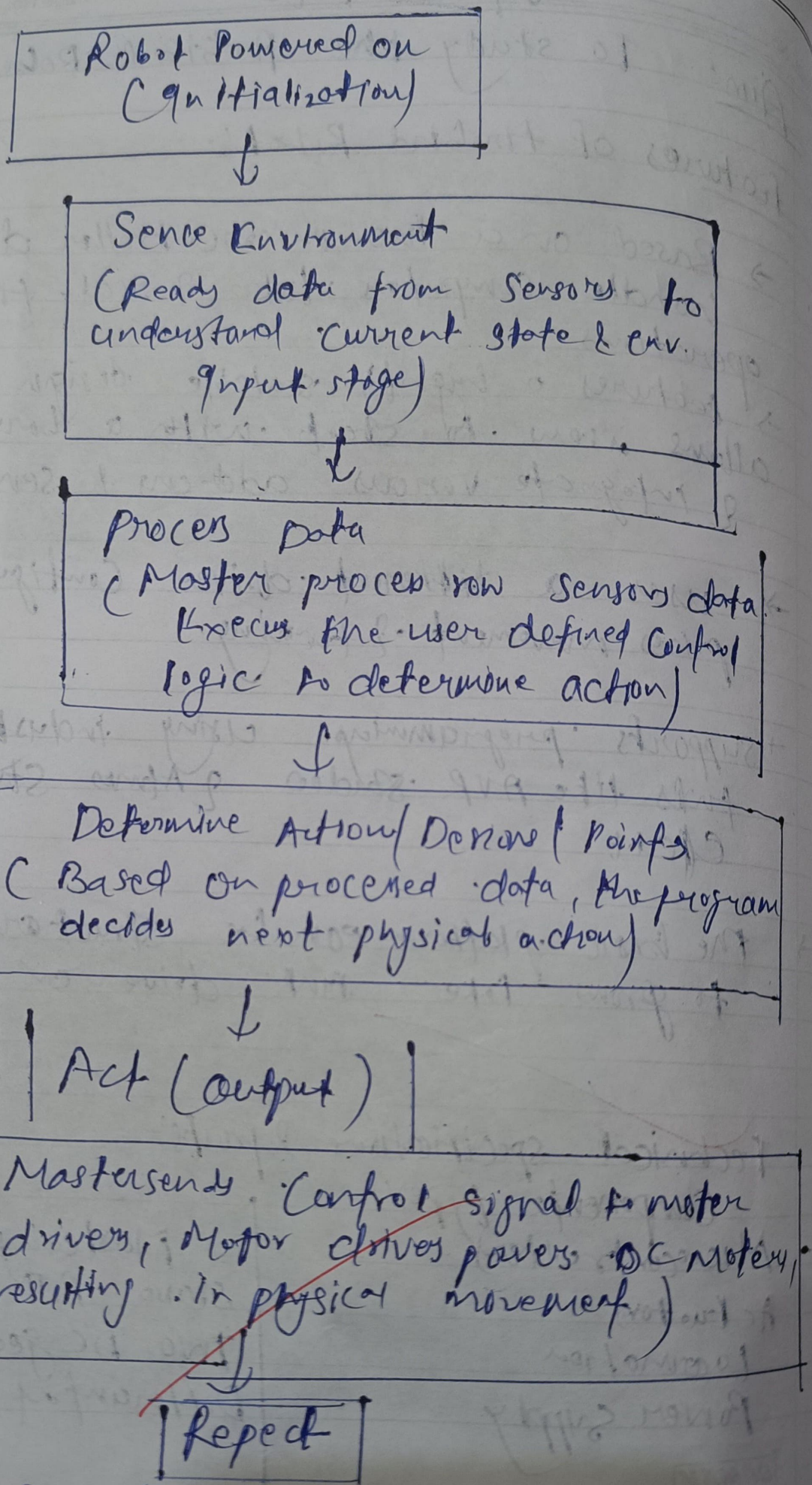


Fig: flow chart for basic working of robot

Date

Expt. No. Page No.

- Castor wheel, 8.6V; 2 wMPU
- Nickel Metal Hydride (NiMH) rechargable Endurance ~2hrs
- 2kg on flat surface

Communications	Wireless : 2.4 GHz ZigBee Module Wired : USB Communication; Wired RS232 Simplex 4w communication
Indicators	-2×16 Character LCD, Indicator LED, Bar
Dimentions	Diameter: 16cm Height: 10cm Weight: 1300 gm

(iv) Sensor System:

Sensor type	Default Quantity
→ White Line Sensors	7
→ Sharp IR Range Sensors	5
→ Analog IR Proximity Sensors	8 (Extendable to)
→ Position Encoders	2 (Extendable to 4)
→ Directional light sensors	8
- In built Sensors	1 (per type)

SAMAY

Teacher's Signature

Expt. No. Date

Page No.

(v) History & Context:

(a) Origin:- Designed in NEX Robotics in collaboration with ERTS lab at CSE Dept, IIT Bombay.

(b) Role in Education: It is primary robot used in the e-yantra lab setup curriculum & the e-yantra Robotics Competition; a project based learning.

(vi) Use Cases:

→ Educational Robotics → Perfect for lab experiments.

→ IOT Projects - it's modularity & wireless communication capabilities allows it to be integrated as Mobile node in WSNs.

→ Autonomous Line following Robot

→ Obstacle avoiding Robot

→ Maze Solving Robot.

SAMAY

Teacher's Signature