CH. DEVI LAL STATE INSTITUTE
of Engineering & Technology
PANNIWALA MOTA, SIRSA

LABORATORY MANUAL
OF
ARTIFICIAL INTELLIGENCE USING PYTHON
(PC/CDS/63-P)

Prepared By:
Dr. Arushi Bansal
Assistant Professor
(Computer Science & Engg. Deptt.)

Artificial Intelligence using Python Lab Manual (PC/CDS/63-P)

1. Introduction to Artificial Intelligence

Artificial Intelligence (AI) is a branch of computer science that focuses on developing intelligent machines capable of performing tasks that normally require human intelligence such as learning, reasoning, problem-solving, perception, and decision making.

The term **Artificial Intelligence** was first introduced by John McCarthy in 1956 during the **Dartmouth Conference**, which marked the beginning of AI as an academic discipline.

Goals of AI

- Create expert systems
- Implement human intelligence in machines
- Enable machines to learn and adapt
- Solve complex problems efficiently

Applications of AI

- Natural Language Processing
- Robotics
- Expert Systems
- Computer Vision
- Game Playing
- Speech Recognition
- Autonomous Vehicles

2. Intelligent Agents

An **agent** is an entity that perceives its environment through sensors and acts upon the environment through actuators.

Example:

- A robot uses **cameras (sensors)** and **motors (actuators)**.

Types of Agents

1. **Simple Reflex Agent**
2. **Model-Based Agent**
3. **Goal-Based Agent**
4. **Utility-Based Agent**
5. **Learning Agent**

Structure of an Intelligent Agent

Agent = Architecture + Program

3. Problem Solving in AI

AI systems solve problems by searching through possible states.

Components of a Problem

- Initial state
- State space
- Goal state
- Operators
- Path cost

Example
In the **8-Puzzle problem**, tiles are moved until they reach the goal configuration.

4. Search Techniques
Search algorithms help in exploring possible solutions.

4.1 Uninformed Search (Blind Search)
These algorithms have **no additional knowledge about the goal**.

Examples:
- Breadth First Search (BFS)
- Depth First Search (DFS)
- Depth Limited Search
- Iterative Deepening Search
- Uniform Cost Search

Breadth First Search

Explores nodes level by level.

Advantages
- Finds shortest path

Disadvantages
- Requires large memory

4.2 Informed Search (Heuristic Search)
Uses **heuristic functions** to guide the search.
Examples:
- Greedy Best First Search
- A* Algorithm

A* Algorithm
Uses evaluation function:
$f(n) = g(n) + h(n)$
$f(n)=g(n)+h(n)$
Where:
- **g(n)** = cost from start node
- **h(n)** = heuristic estimate to goal

5. Constraint Satisfaction Problems (CSP)
A **Constraint Satisfaction Problem** consists of:

- Variables
- Domains
- Constraints

Example problems:

- Map Coloring
- Sudoku
- N-Queens Problem

Techniques for Solving CSP

- Backtracking Search
- Constraint Propagation
- Local Search

6. Game Playing in AI

Game playing involves decision making in competitive environments.

Important Concepts

- Game Tree
- Minimax Algorithm
- Alpha-Beta Pruning

Minimax Algorithm

Used in two-player games like chess or tic-tac-toe.

It assumes:

- One player tries to maximize score
- The other tries to minimize score

Alpha-Beta Pruning

Optimization technique that **reduces nodes explored in Minimax**.

7. Knowledge Representation

Knowledge representation is the method of encoding information so that AI systems can reason about it.

Types of Knowledge Representation

- Logical Representation
- Semantic Networks
- Frames
- Production Rules

8. Propositional Logic

A formal system used for reasoning.

Components

- Propositions
- Logical Connectives

Examples:

- AND ($\wedge$)
- OR ($\vee$)

- NOT (¬)
- IMPLIES (→)

Example Statement:
If it rains → the ground is wet.

9. First Order Logic (FOL)

An extension of propositional logic that allows reasoning about objects and relationships.

Elements of FOL

- Constants
- Variables
- Predicates
- Quantifiers

10. Machine Learning in AI

Machine Learning enables systems to **learn from data instead of explicit programming**.

Types of Learning

1. Supervised Learning
2. Unsupervised Learning
3. Reinforcement Learning

Applications include:

- Spam detection
- Recommendation systems
- Image recognition

11. Applications of Artificial Intelligence

AI is widely used in many industries:

- Healthcare
- Finance
- Education
- Autonomous vehicles
- Smart assistants

Examples of AI systems include **Siri**, **Google Assistant**, and **Alexa**.

Practical 1: Study & List Tuple, Set, Dictionary, Classes and Inheritance in Python

Aim

To study and understand tuple, set, dictionary, classes and inheritance in Python.

Theory

- **Tuple**: Ordered, immutable collection of elements.
- **Set**: Unordered collection with unique elements.
- **Dictionary**: Stores data in key–value pairs.
- **Class**: Blueprint for creating objects.
- **Inheritance**: Mechanism where one class inherits properties of another.

Tuple: A tuple is an ordered and immutable collection of elements.
Example: (10,20,30)
Set: A set is an unordered collection of unique elements.
Example: {1,2,3}
Dictionary: A dictionary stores data in key-value pairs.
Example: {“Name":"Arushi","Course":"AI"}
Classes: A class is a blueprint for creating objects.
Inheritance: Inheritance allows one class to acquire properties of another class.

Flowchart

Start
|
Create Tuple
|
Create Set
|
Create Dictionary
|
Define Class
|
Apply Inheritance
|
Display Output
|
Stop

Program

```
# Tuple
t = (10, 20, 30)
print("Tuple:", t)

# Set
s = {1, 2, 3, 3, 4}
print("Set:", s)

# Dictionary
d = {"Name": "Arushi", "Age": 20}
print("Dictionary:", d)

# Class
class Student:
    def __init__(self, name):
        self.name = name

    def display(self):
        print("Student Name:", self.name)

# Inheritance
class Result(Student):
    def __init__(self, name, marks):
        super().__init__(name)
        self.marks = marks

    def show(self):
        print("Marks:", self.marks)

s1 = Result("Arushi", 90)
s1.display()
s1.show()
```

Result

```
Tuple: (10, 20, 30)
Set: {1, 2, 3, 4}
Dictionary: {'Name': 'Arushi', 'Age': 20}
Student Name: Arushi
Marks: 90
```

Practical 2: Study Simple Reflex Agent and Model Based Agent

Aim

To understand Simple Reflex Agent and Model Based Agent.

Theory

- **Simple Reflex Agent**: Acts only on current percept using condition-action rules.
- **Model Based Agent**: Maintains internal state to track aspects of the environment.

Example Program

```
def simple_reflex_agent(percept):
    if percept == "Dirty":
        return "Clean"
    else:
        return "Move"

print(simple_reflex_agent("Dirty"))
```

Result

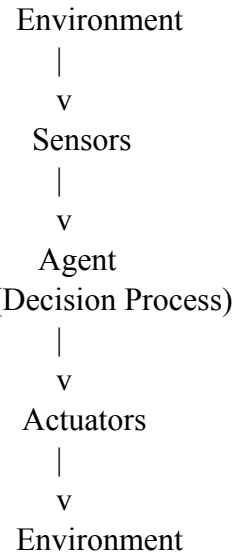
Clean

Practical 3: Graph for Profit or Loss in Banking Application

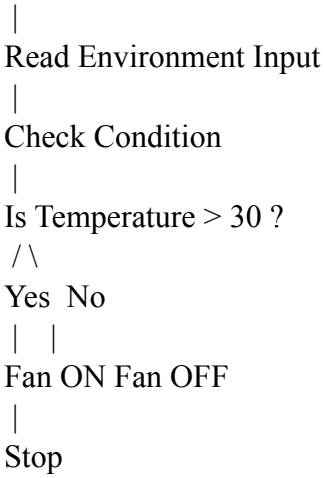
Aim
To represent profit or loss using graphs in Python.

Simple Reflex Agent: Acts only on the current percept without considering past states.
Example rule:
IF temperature > 30
THEN turn fan ON

Model Based Agent: Maintains an internal state of the environment and uses it for decision making.
Agent Architecture Diagram



Flowchart



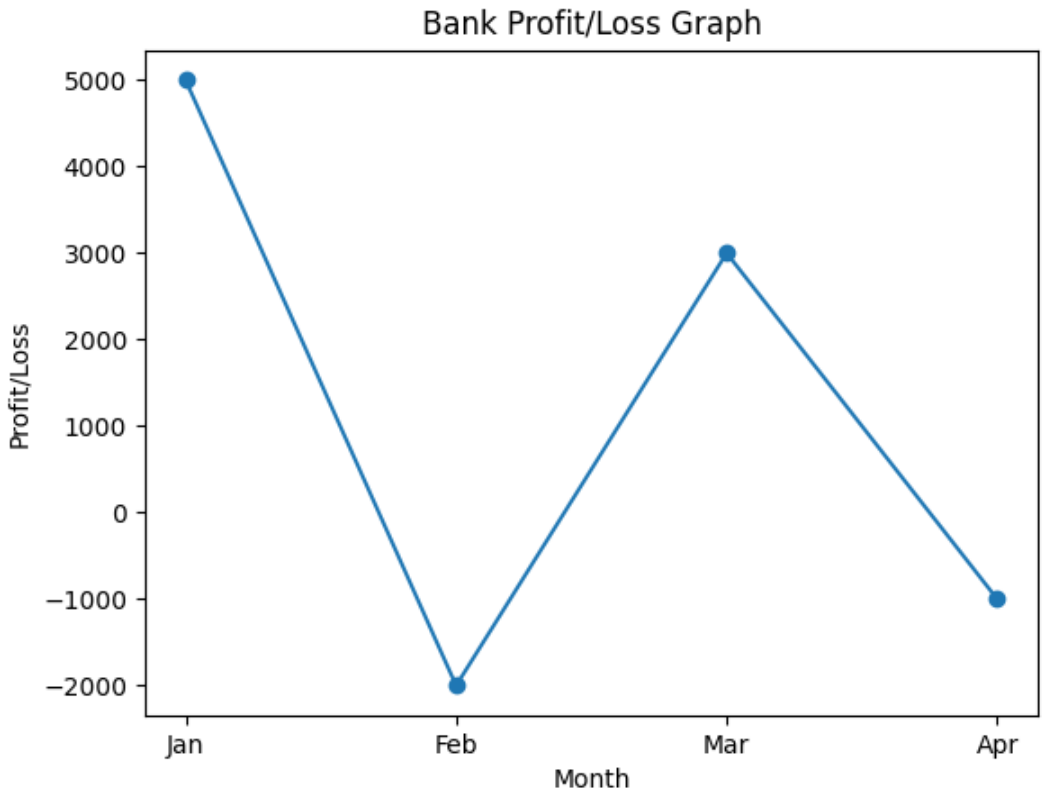
Program

```
import matplotlib.pyplot as plt

months = ["Jan", "Feb", "Mar", "Apr"]
profit_loss = [5000, -2000, 3000, -1000]

plt.plot(months, profit_loss, marker='o')
plt.title("Bank Profit/Loss Graph")
plt.xlabel("Month")
plt.ylabel("Profit/Loss")
plt.show()
```

Result



Practical 4: PEAS Description

Aim

To describe a problem using PEAS.

Example: Self Driving Car

Component	Description
Performance	Safety, speed, comfort
Environment	Roads, traffic, pedestrians
Actuators	Steering, brakes, accelerator
Sensors	Camera, GPS, radar

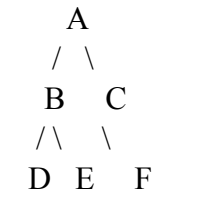
Result

Problem environment was successfully described using PEAS.

Practical 5: Basic Searching Algorithm

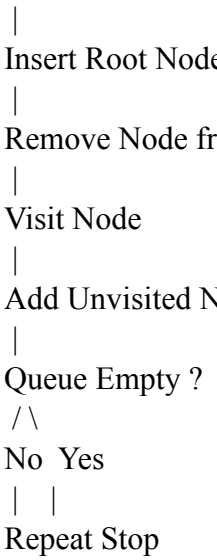
Aim
To implement searching algorithm.

BFS Search Tree Diagram



Traversal Order
A → B → C → D → E → F

Flowchart



Python Program

```
from collections import deque

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

```
visited=set()
queue=deque(['A'])

while queue:
    node=queue.popleft()

    if node not in visited:
        print(node,end=" ")
        visited.add(node)
        queue.extend(graph[node])
```

Program (Linear Search)

```
def linear_search(arr, key):
    for i in range(len(arr)):
        if arr[i] == key:
            return i
    return -1

arr = [10, 20, 30, 40]
key = 30

print("Element found at index:", linear_search(arr, key))
```

Result

Element found at index: 2

Practical 6: 8 Queens Problem

Aim
Place 8 queens on chessboard such that no queen attacks another.

Diagram
Q . . . . . .
. . Q . . . . .
. . . . Q . . .
. Q . . . . . .
. . . . . Q . .
. . . Q . . . .
. . . . . Q .
. . . . . Q

Flowchart
Start
|
Place Queen in Row
|
Check Column & Diagonal Safety
|
Safe ?
/ \
Yes No
| |
Next Row Try Next Column
|
All Queens Placed ?
/ \
Yes No
| |
Solution Backtrack
|
Stop

Program

```
N = 8
def solve(board, col):
    if col >= N:
        return True
```

```
    for i in range(N):
        if is_safe(board, i, col):
            board[i][col] = 1
            if solve(board, col + 1):
                return True
            board[i][col] = 0
    return False

def is_safe(board, row, col):
    for i in range(col):
        if board[row][i] == 1:
            return False
    return True
board = [[0]*N for i in range(N)]
solve(board, 0)

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

Result

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

Practical 7: A* Algorithm

Aim

To implement A* search algorithm.

A* Evaluation Function
f(n) = g(n) + h(n)
Where
g(n) = cost from start
h(n) = heuristic estimate

A* Search Diagram
Start A
 / \
B C
 \ /
  D (Goal)

Program

```
import heapq

graph = {
'A': [('B',1), ('C',3)],
'B': [('D',3)],
'C': [('D',1)],
'D': []
}

def astar(start, goal):
    queue = [(0,start)]
    visited = set()

    while queue:
        cost,node = heapq.heappop(queue)

        if node == goal:
            print("Goal reached")
            return

        if node not in visited:
            visited.add(node)
```

```
            for neighbour, weight in graph[node]:
                heapq.heappush(queue,(cost+weight,neighbour))

astar('A','D')
```

Result

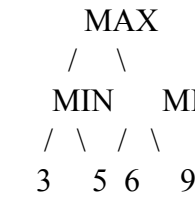
Goal reached

Practical 8: Alpha Beta Tree Search

Aim

To implement Alpha Beta pruning.

Game Tree Diagram



Alpha-Beta prunes unnecessary nodes.

Flowchart

Start
|
Generate Game Tree
|
Apply Minimax
|
Apply Alpha-Beta Pruning
|
Select Best Move
|
Stop

Program

```
import math

def alphabeta(depth, nodeIndex, maximizingPlayer, values, alpha, beta):
    if depth == 3:
        return values[nodeIndex]

    if maximizingPlayer:
        best = -math.inf
        for i in range(2):
            val = alphabeta(depth+1, nodeIndex*2+i, False, values, alpha, beta)
            best = max(best,val)
            alpha = max(alpha,best)
            if beta <= alpha:
                break
```

```
        return best
    else:
        best = math.inf
        for i in range(2):
            val = alphabeta(depth+1, nodeIndex*2+i, True, values, alpha, beta)
            best = min(best,val)
            beta = min(beta,best)
            if beta <= alpha:
                break
        return best

values = [3,5,6,9,1,2,0,-1]

print("Optimal Value:", alphabeta(0,0,True,values,-math.inf,math.inf))
```

Result

Optimal Value: 5

Practical 9: Classical Planning Algorithm

Aim

To implement classical planning.

Example

Goal: Move from Room A → Room B.

Program

```
state = "A"

if state == "A":
    action = "Move to B"
    state = "B"

print("Current State:", state)
```

Result

Current State: B

Practical 10: Robot Obstacle Traversal Problem

Aim
Move robot from start to goal while avoiding obstacles.

Grid Diagram
S → Start
G → Goal
X → Obstacle

S 0 0
X X 0
0 0 G

Flowchart
Start
|
Current Position
|
Compare with Goal
|
Obstacle Present ?
/ \
Yes No
| |
Change Path Move Forward
|
Goal Reached ?
/ \
No Yes
| |
Repeat Stop

Program

```
grid = [
[0,0,0],
[0,1,0],
[0,0,0]
]

def is_safe(x,y):
    return x>=0 and y>=0 and x<3 and y<3 and grid[x][y]==0
```

```
print("Robot can traverse avoiding obstacle.")
```

Result

```
Robot can traverse avoiding obstacle.
```