GENERAL INSTRUCTIONS

1. **Attendance and Punctuality:**
    - Students should arrive on time for the lab session.
    - Regular attendance is essential for understanding and completing lab assignments.
2. **Preparation:**
    - Students should come prepared by reviewing relevant lecture material and pre-lab readings.
    - Bring any required materials, such as laptops, notebooks, or specific software.
3. **Lab Safety:**
    - Adhere to safety guidelines, especially if working with hardware components.
    - Keep the workspace tidy and organized.
4. **Collaboration:**
    - Encourage collaboration but avoid copying. Students should understand and do their own work.
    - Respect the academic integrity policies of the institution.
5. **Lab Reports:**
    - Follow specific guidelines for lab reports, including formatting and content.
    - Include observations, results, and conclusions in a clear and concise manner.
6. **Equipment Usage:**
    - Treat lab equipment with care and report any malfunctions promptly.
    - Log in and out of computers responsibly, and follow any specific instructions for software use.
7. **Problem Solving:**
    - Work independently on problem-solving tasks and programming assignments.
    - Seek help from teaching assistants or the instructor if encountering difficulties.
8. **Submission Deadlines:**
    - Meet deadlines for submitting lab assignments.
    - Late submissions may be subject to penalties, so it's crucial to plan and manage time effectively.

9. Code of Conduct:

- Adhere to a code of conduct that promotes a positive and respectful learning environment.
- Report any instances of academic misconduct.

10. Feedback:

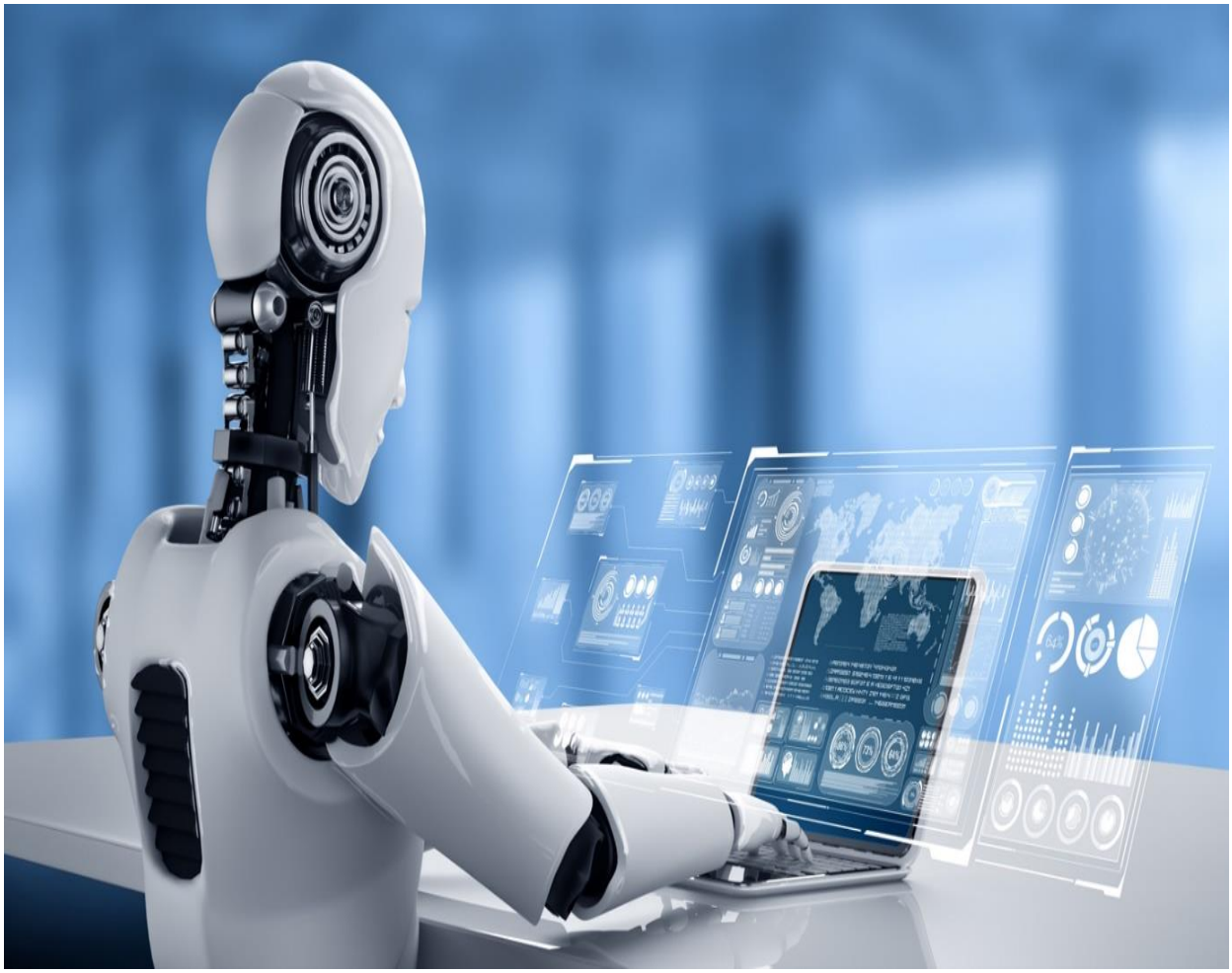
- Provide constructive feedback to the instructor about the lab content, structure, or any difficulties faced during the session.

11. Ask Questions:

- Don't hesitate to ask questions when uncertain about instructions or concepts.
- Participate actively in discussions and engage with the material.

12. Backup Work:

- Regularly back up code and project work to avoid data loss.



ARTIFICIAL INTELLIGENCE LABORATORY MANUAL
[PCS-601]

Department of Computer Science and Engineering

Prolog

What is Prolog?

Prolog is a logic programming language that is used in artificial intelligence. It is a declarative programming language expressing logic as relations, called facts and rules. A Prolog program consists of a collection of facts and rules; a query is a theorem to be proved. Here are some basic elements of Prolog:

Facts: These are statements that are true. They are written as a predicate followed by a period. For example, `likes(john, mango).` is a fact that states that John likes mango.

Rules: These are statements that define relationships between facts. They are written as a predicate followed by a body, which is a list of one or more predicates separated by commas and enclosed in parentheses, followed by a period.

For example,

Grandfather(X, Y) :-father(X,Z),parent(Z,Y)

This implies that for X to be the grandfather of Y, Z should be a parent of Y and X should be the father of Z.

Queries: These are questions that we ask Prolog. They are written as a predicate followed by a question mark. For example, `?- likes(john, mango).` is a query that asks if John likes mango.

Key features of Prolog:

The key features of Prolog are as follows:

1. Declarative Language: Prolog is a declarative programming language, meaning that programs are written as sets of logical statements. This makes Prolog programs concise and easy to read.

2. Predicate Calculus: Prolog uses the language of predicate calculus, which allows for the representation of relationships between facts and rules.

3. Handling Lists and Recursion: Prolog naturally handles lists and recursion, making it well-suited for tasks that involve these concepts.

4. Unification: Prolog uses unification, which is the process of determining if given terms can represent the same structure. Unification is a fundamental concept in Prolog.

5. Backtracking: When a task fails, Prolog traces backward and tries to satisfy the previous task. This backtracking feature allows for flexible problem-solving.

6. Efficiency: Prolog is known for its efficiency in solving problems that would be difficult in other programming languages.

Symbols in Prolog

Using the following truth-functional symbols, the Prolog expressions are comprised.

English	Predicate Calculus	Prolog
If	-->	:-
Not	~	Not
Or	V	;
And	^	,

These symbols have the same interpretation as in the predicate calculus.

OBJECTIVE 1:

WRITE A PROLOG PROGRAM TO FIND THE SUM OF ELEMENTS IN GIVEN LIST.

we will define a clause, list_sum(List, Sum), this will return the sum of the elements of the list.

- If the list is empty, then sum will be 0.
- Represent list as [Head|Tail], find sum of tail recursively and store them into SumTemp, then set Sum = Head + SumTemp.

Program

```
list_sum([],0).
list_sum([Head|Tail], Sum) :-
   list_sum(Tail,SumTemp),
   Sum is Head + SumTemp.
```

Output

```
yes
| ?- list_sum([5,12,69,112,48,4],Sum).

Sum = 250

yes
| ?- list_sum([8,5,3,4,7,9,6,1],Sum).

Sum = 43

yes
| ?-
```

OBJECTIVE 2:

WRITE A PROLOG PROGRAM TO DELETE AN ELEMENT FROM GIVEN LIST.

Suppose we have a list L and an element X, we have to delete X from L. So there are three cases −

- If X is the only element, then after deleting it, it will return empty list.
- If X is head of L, the resultant list will be the Tail part.
- If X is present in the Tail part, then delete from there recursively.

Program

```
list_delete(X, [X], []).
list_delete(X,[X|L1], L1).
list_delete(X, [Y|L2], [Y|L1]) :- list_delete(X,L2,L1).
```

Output

```
| ?- list_delete(a,[a,e,i,o,u],NewList).

NewList = [e,i,o,u] ?

yes
| ?- list_delete(a,[a],NewList).

NewList = [] ?

yes
| ?- list_delete(X,[a,e,i,o,u],[a,e,o,u]).
```

```
X = i ? ;

no
| ?-
```

OBJECTIVE 3:

WRITE A PROLOG CODE TO FIND THE LAST ELEMENT OF A GIVEN INPUT LIST.

Define a predicate last_element(X, L) that is true if X is the last element of list L

```
last_element(X, [X]).          % Base case: X is the last element of a singleton list
last_element(X, [_|T]) :- last_element(X, T).          % Recursive case: X is the last element of the tail of the list
```

Output:

```
last_element(X, [1, 2, 3, 4, 5]).

which should return X = 5.
```

OBJECTIVE 4:

WRITE A PROLOG PROGRAM TO CHECK IF GIVEN ELEMENT IS A MEMBER OF A GIVEN LIST.

During this operation, we can check whether a member X is present in list L or not? So how to check this? Well, we have to define one predicate to do so. Suppose the predicate name is **list_member(X,L)**. The goal of this predicate is to check whether X is present in L or not.

To design this predicate, we can follow these observations. X is a member of L if either −

- X is head of L, or
- X is a member of the tail of L

Program

```
list_member(X,[X|_]).
list_member(X,[_|TAIL]) :- list_member(X,TAIL).
```

Output

```
| ?- list_member(b,[a,b,c]).

true ?

yes
| ?- list_member(b,[a,[b,c]]).

no
| ?- list_member([b,c],[a,[b,c]]).
```

```
true ?

yes
| ?- list_member(d,[a,b,c]).

no
| ?- list_member(d,[a,b,c]).
```

OBJECTIVE 5:

WRITE A PROLOG PROGRAM TO APPEND LIST L1 TO LIST L2 AND BIND THE RESULT TO LIST L3.

Appending two lists means adding two lists together, or adding one list as an item. Now if the item is present in the list, then the append function will not work. So we will create one predicate namely, list_append(L1, L2, L3). The following are some observations −

- Let A is an element, L1 is a list, the output will be L1 also, when L1 has A already.
- Otherwise new list will be L2 = [A|L1].

Program

```
list_member(X,[X|_]).
list_member(X,[_|TAIL]) :- list_member(X,TAIL).

list_append(A,T,T) :- list_member(A,T),!.
list_append(A,T,[A|T]).
```

In this case, we have used (!) symbol, that is known as cut. So when the first line is executed successfully, then we cut it, so it will not execute the next operation.

Output

```
| ?- list_append(a,[e,i,o,u],NewList).

NewList = [a,e,i,o,u]
```

```
yes
| ?- list_append(e,[e,i,o,u],NewList).

NewList = [e,i,o,u]

yes
| ?- list_append([a,b],[e,i,o,u],NewList).

NewList = [[a,b],e,i,o,u]

yes
| ?-
```

OBJECTIVE 6:

WRITE A PROLOG CODE TO FIND THE MAXIMUM ELEMENT OF A LIST.

This operation is used to find the maximum element from a list. We will define a predicate, **list_max_elem(List, Max)**, then this will find Max element from the list and return.

- If there is only one element, then it will be the max element.
- Divide the list as [X,Y|Tail]. Now recursively find max of [Y|Tail] and store it into MaxRest, and store maximum of X and MaxRest, then store it to Max.

Program

```
max_of_two(X,Y,X) :- X >= Y.
max_of_two(X,Y,Y) :- X < Y.
list_max_elem([X],X).
list_max_elem([X,Y|Rest],Max)  :-
   list_max_elem([Y|Rest],MaxRest),
   max_of_two(X,MaxRest,Max).
```

Output

```
| ?- list_max_elem([8,5,3,4,7,9,6,1],Max).

Max = 9 ?

yes
| ?- list_max_elem([5,12,69,112,48,4],Max).
```

```
Max = 112 ?

yes
| ?-
```

OBJECTIVE 7:

WRITE A PROLOG PREDICATE TO REVERSE THE ORDER OF THE ELEMENTS OF A GIVEN INPUT LIST.

Suppose we have a list L = [a,b,c,d,e], and we want to reverse the elements, so the output will be [e,d,c,b,a]. To do this, we will create a clause, list_reverse(List, ReversedList). Following are some observations −

- If the list is empty, then the resultant list will also be empty.
- Otherwise put the list items namely, [Head|Tail], and reverse the Tail items recursively, and concatenate with the Head.
- Otherwise put the list items namely, [Head|Tail], and reverse the Tail items recursively, and concatenate with the Head.

Program

```
list_concat([],L,L).
list_concat([X1|L1],L2,[X1|L3]) :- list_concat(L1,L2,L3).


list_rev([],[]).
list_rev([Head|Tail],Reversed) :-
   list_rev(Tail, RevTail),list_concat(RevTail, [Head],Reversed).
```

Output

```
| ?- list_rev([a,b,c,d,e],NewList).

NewList = [e,d,c,b,a]

yes
| ?- list_rev([a,b,c,d,e],[e,d,c,b,a]).

yes
| ?-
```

VIVA VOCE QUESTIONS - (10-15)

1. What do you understand by Artificial Intelligence? List some real-life applications of AI.
2. What are the types of Artificial Intelligence?
3. What is the difference between Strong AI and Weak AI?
4. What are the areas and domains where AI is used?
5. What are the tools of AI?
6. What is an agent in AI?
7. Name some popular programming languages in AI.
8. What is the difference between inductive, deductive, and abductive learning?
9. What is the difference between Statistical AI and Classical AI?
10. Are AI and ML the same? If yes, how, and if not then why so?