**Medi-Caps University, Indore**

**Department of Computer Science & Engineering**

**Dhananjay Porwal EN19CS301110**

**Software Workshop-I (CS3ES12)**

**INDEX**

|  |  |  |  |
| --- | --- | --- | --- |
| **S. No.** | **Name of the Practical** | **Page No.** | **Remarks** |
| 1 | Case study of AI applications which are available in your phones. | 2 |  |
| 2 | Case study of any two implemented AI Projects. | 5 |  |
| 3 | Write a program to implement BFS. | 6 |  |
| 4 | Write a program to implement DFS. | 9 |  |
| 5 | Case study of AI software or tool (Prolog). | 11 |  |
| 6 | Explain variables, constants with the help of some facts and rules in Prolog and run some queries. | 14 |  |
| 7 | Program of weather (City, Season, Temperature) with how to print output? | 17 |  |
| 8 | Explain Structure with program and queries. | 18 |  |
| 9 | Explain Operators in Prolog | 20 |  |
| 10 | Write a program to convert Fahrenheit temperature to degree Celsius. | 21 |  |
| 11 | Explain List in Prolog. What is nested list? | 22 |  |
| 12 | Explain Member functions in Prolog | 24 |  |
| 13 | Propose one AI based start-up which you can implement as your project | 25 |  |

**Q. Case study of AI applications which are available in your phones.**

**1.Voice Assistant Tools:Amazon Alexa**

A mobile app for this famous virtual assistant created by Microsoft allows users to enjoy all aspects of infotainment, planning, and scheduling features. With voice recognition, the app helps users manipulate other Alexa-powered devices such as smart house utilities, tablets, TVs, wearables, cars, etc. from the phone. It converts speech into text and uses the Wolfram language to provide accurate answers to the questions asked. More and more people trust their routine tasks to this personal assistant, making the Amazon company gain record-setting profits every new quarter.

**Feature range:** list-making, alarm and reminder setting, product ordering, audio streaming, messaging and call making, information browsing, and many others.

**Key AI implementations:** voice recognition, Wolfram’s technology for collecting data and answering questions, etc.

### 2. Education: ELSA Speak

ELSA speak is a bot-aided English speaking coach mainly aimed at reducing accent. The app learns from speech data. It uses voice recognition technology to listen to learners’ accents and defines the “quality” of pronounced words with red, yellow, and green color according to the rules of American English pronunciation. When the speech fragment is analyzed, the app presents audio tips on mouth and tongue positioning for better articulation.

**Feature range:** progress tracking, language proficiency assessments, personalized practicing curriculum, lessons, vocabulary, etc.

**Key AI implementations:** speech recognition, machine learning.

### 3. Retail: eBay

The eBay system uses the powers of artificial intelligence to the fullest extent to provide users with a convenient shopping experience. This shopping app is programmed to understand the context of the customer’s search and offer a full list of relevant goods, which decreases the search time to seconds and increases customer satisfaction. Machine vision helps the app find the required item by the photo that a user added for a search request. Besides, machine learning lets sellers find out and compare current costs of specific goods to help them sell products faster.

**Feature range:** shopping lists, cart, ratings and reviews, wish list, notifications, messaging system, media files support, payment system, etc.

**Key AI implementations:** computer vision, translation, machine learning, natural language processing.

### 4. Media & Entertainment: Pandora

Although the Pandora music streaming service has been in business for 20 years already, it has taken the 3rd place among radio streaming AI apps for iPhone just recently. The mix of musicology and data science made it possible to produce the app that analyzes the listening habits of users and provides them with the songs they exactly want to hear at that moment.

Using the Voice Mode, music lovers can get the desired track even if their requests are vague, for instance: “Play something for me” or “Let’s listen to something new.” To fit any mood of a person, the AI system decomposes every musical piece into the smallest attributes to understand the mood of the song. The app creators named this feature “Music DNA.”

**Feature range:** customized Pandora modes for sorted music, podcasts, contextual recommendations, music collections, downloading service, playlists, etc.

**Key AI implementations:** voice and speech recognition, machine learning, a Voice Mode assistant.

### 5. Photo Editing: Gradient

Artificial intelligence [made Gradient a viral app](https://www.usatoday.com/story/tech/2019/10/18/who-your-celebrity-lookalike-gradient-app-try-tell-you/4020203002/) in the last months of 2019. It was the Gradient app that caused the burst of celebrity lookalike pictures posted on every social media platform by thousands of people, including celebrities themselves. Although the hype about this feature ceased, this photo editor remains popular for the effects that offer users to edit their face shots with AI-powered painting features.

**Feature range:** filters, editing instruments, special effects, etc.

**Key AI implementations:** face recognition, image recognition.

### Q. Case study of any two implemented AI Projects.

### ****1.**** [****Online Assignment Plagiarism Checker****](https://nevonprojects.com/online-assignment-plagiarism-checker-project-using-data-mining/)

This is one of the needed AI projects of the hour. Plagiarism is a serious issue that needs to be controlled and monitored. It refers to the act of blindly copying someone else’s work and presenting it as your unique work. Plagiarism is done by paraphrasing sentences, using similar keywords, changing the form of sentences, and so on. In this sense, plagiarism is like theft of intellectual property.

In this project, you will develop a plagiarism detector that can detect the similarities in copies of text and detect the percentage of plagiarism. This plagiarism detector used the text mining method. In this software, users can register by login by creating a valid login id and password.

So, you can log in using your unique ID and password and upload your assignment file. After the upload is complete, the file will be divided into content and reference link. The checker will then process the full content, check grammatical errors, visit each reference link, and scan the content of all the links to find matches with your content. Users can also store their files and view them later.

### [****2. Personality Prediction System via CV Analysis****](https://nevonprojects.com/personality-prediction-system-through-cv-analysis/)

This is one of the interesting Artificial Intelligence project ideas. It is a challenging task to shortlisting deserving candidates from a massive pile of CVs. What if there’s a software that can interpret the personality of a candidate by analyzing their CV? This will make the selection process much more manageable. This project aims to create advanced software that can provide a legally justified and fair CV ranking system.

The system will work something like this – candidates will register in the system by entering all the relevant details and upload their CV. They will also take an online test that focuses on personality traits and a candidate’s aptitude. Candidates can also view their test results.

First, the system will rank candidates based on their skills and experience for a particular job profile. It will also consider all other crucial aspects, like soft skills, interests, professional certifications, etc. This will eliminate all the unsuitable candidates for a job role and create a list of the most suitable candidates for the same. Together with the online personality test and CV analysis, the system will create a comprehensive picture of the candidates, simplifying the HR department’s job.

**Q. Write a program to implement BFS.**

#include<iostream>

#include <list>

using namespace std;

// This class represents a directed graph using

// adjacency list representation

class Graph

{

int V; // No. of vertices

// Pointer to an array containing adjacency

// lists

list<int> \*adj;

public:

Graph(int V); // Constructor

// function to add an edge to graph

void addEdge(int v, int w);

// prints BFS traversal from a given source s

void BFS(int s);

};

Graph::Graph(int V)

{

this->V = V;

adj = new list<int>[V];

}

void Graph::addEdge(int v, int w)

{

adj[v].push\_back(w); // Add w to v’s list.

}

void Graph::BFS(int s)

{

// Mark all the vertices as not visited

bool \*visited = new bool[V];

for(int i = 0; i < V; i++)

visited[i] = false;

// Create a queue for BFS

list<int> queue;

// Mark the current node as visited and enqueue it

visited[s] = true;

queue.push\_back(s);

// 'i' will be used to get all adjacent

// vertices of a vertex

list<int>::iterator i;

while(!queue.empty())

{

// Dequeue a vertex from queue and print it

s = queue.front();

cout << s << " ";

queue.pop\_front();

// Get all adjacent vertices of the dequeued

// vertex s. If a adjacent has not been visited,

// then mark it visited and enqueue it

for (i = adj[s].begin(); i != adj[s].end(); ++i)

{

if (!visited[\*i])

{

visited[\*i] = true;

queue.push\_back(\*i);

}

}

}

}

// Driver program to test methods of graph class

int main()

{

// Create a graph given in the above diagram

Graph g(4);

g.addEdge(0, 1);

g.addEdge(0, 2);

g.addEdge(1, 2);

g.addEdge(2, 0);

g.addEdge(2, 3);

g.addEdge(3, 3);

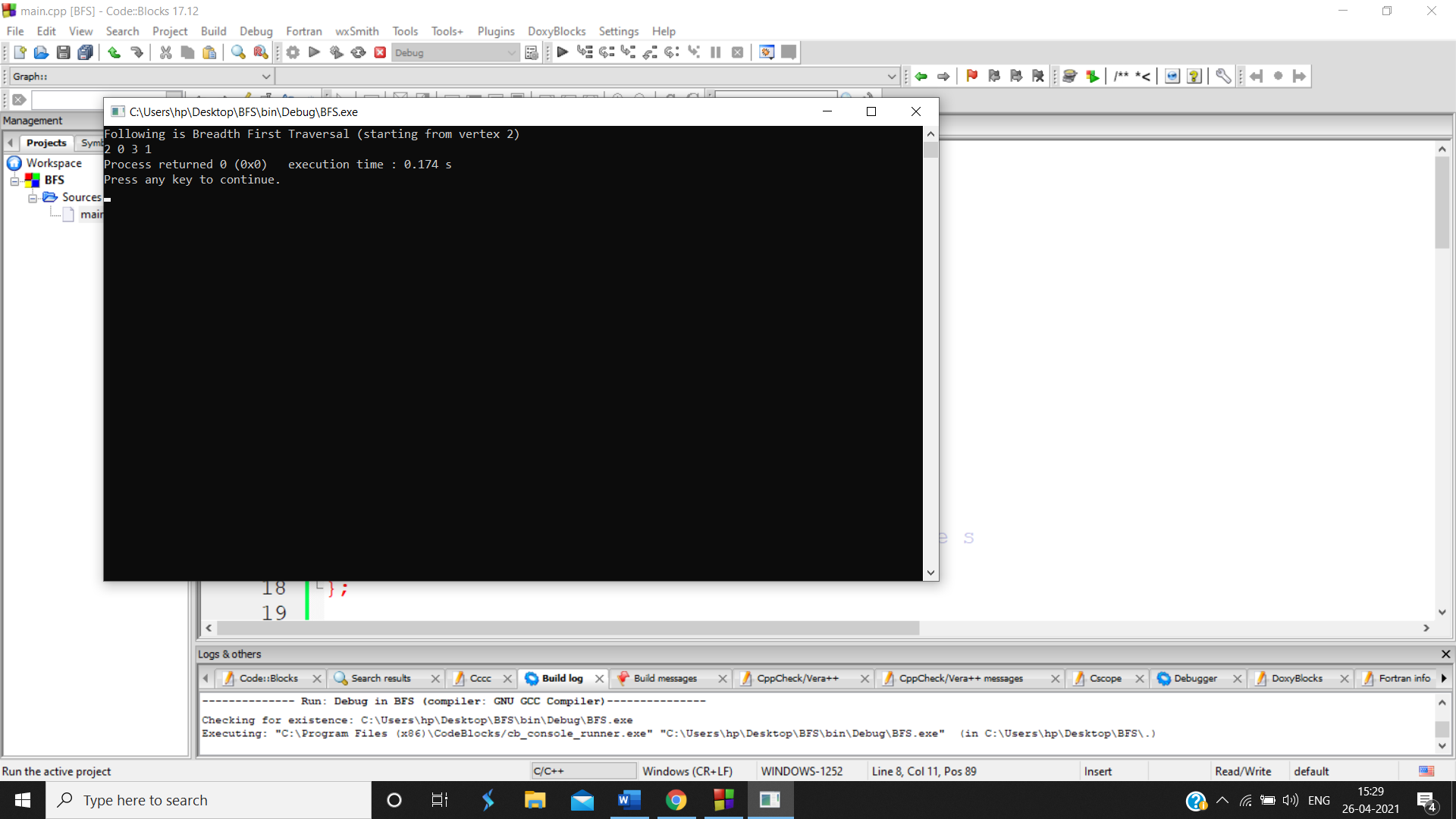
cout << "Following is Breadth First Traversal "

<< "(starting from vertex 2) \n";

g.BFS(2);

return 0;

}



**Q. Write a program to implement DFS.**

#include <bits/stdc++.h>

using namespace std;

class Graph

{

public:

map<int, bool> visited;

map<int, list<int>> adj;

void DFS(int v);

};

void Graph::addEdge(int v, int w)

{

adj[v].push\_back(w); // Add w to v’s list.

}

void Graph::DFS(int v)

{

// Mark the current node as visited and

// print it

visited[v] = true;

cout << v << " ";

// Recur for all the vertices adjacent

// to this vertex

list<int>::iterator i;

for (i = adj[v].begin(); i != adj[v].end(); ++i)

if (!visited[\*i])

DFS(\*i);

}

// Driver code

int main()

{

// Create a graph given in the above diagram

Graph g;

g.addEdge(0, 1);

g.addEdge(0, 9);

g.addEdge(1, 2);

g.addEdge(2, 0);

g.addEdge(2, 3);

g.addEdge(9, 3);

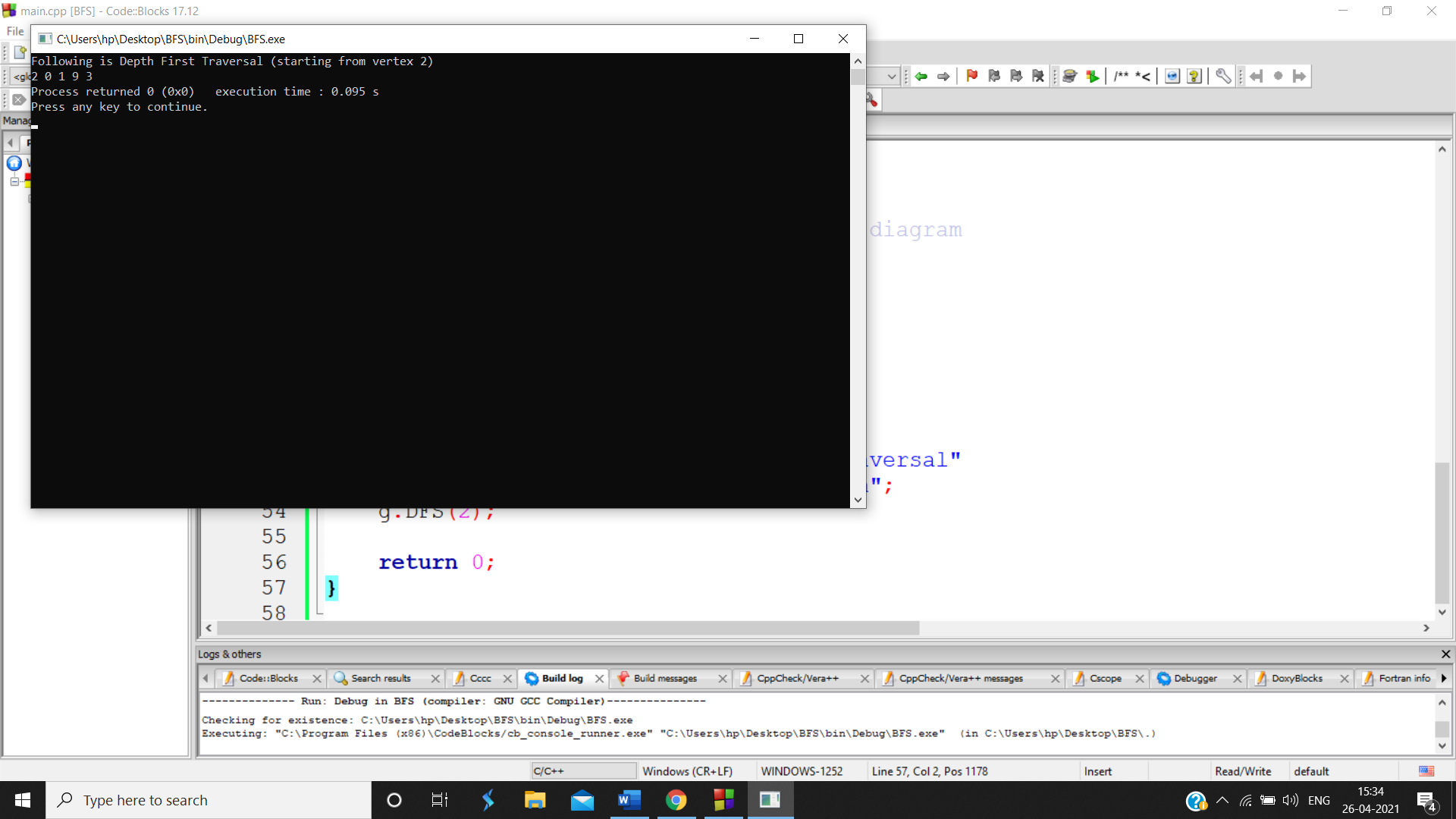
cout << "Following is Depth First Traversal"

" (starting from vertex 2) \n";

g.DFS(2);

return 0;

}



**Q. Case Study of AI software or tool (Prolog).**

**Prolog** is a [logic programming](https://en.wikipedia.org/wiki/Logic_programming) language associated with [artificial intelligence](https://en.wikipedia.org/wiki/Artificial_intelligence) and computational statistics.

Prolog has its roots in [first-order logic](https://en.wikipedia.org/wiki/First-order_logic), a [formal logic](https://en.wikipedia.org/wiki/Formal_logic), and unlike many other [programming languages](https://en.wikipedia.org/wiki/Programming_language), Prolog is intended primarily as a [declarative programming](https://en.wikipedia.org/wiki/Declarative_programming) language: the program logic is expressed in terms of [relations](https://en.wikipedia.org/wiki/Finitary_relation), represented as facts and [rules](https://en.wikipedia.org/wiki/Rule_of_inference). A [computation](https://en.wikipedia.org/wiki/Computation) is initiated by running a query over these relations.

The language was developed and implemented in Marseille, France, in 1972 by [Alain Colmerauer](https://en.wikipedia.org/wiki/Alain_Colmerauer) with Philippe Roussel, based on [Robert Kowalski](https://en.wikipedia.org/wiki/Robert_Kowalski)'s procedural interpretation of [Horn clauses](https://en.wikipedia.org/wiki/Horn_clause).

Prolog was one of the first logic programming languages and remains the most popular such language today, with several free and commercial implementations available. The language has been used for [theorem proving](https://en.wikipedia.org/wiki/Automated_theorem_proving), [expert systems](https://en.wikipedia.org/wiki/Expert_system), [term rewriting](https://en.wikipedia.org/wiki/Term_rewriting), [type systems](https://en.wikipedia.org/wiki/Type_system), and [automated planning](https://en.wikipedia.org/wiki/Automated_planning), as well as its original intended field of use, [natural language processing](https://en.wikipedia.org/wiki/Natural_language_processing). Modern Prolog environments support the creation of [graphical user interfaces](https://en.wikipedia.org/wiki/Graphical_user_interface), as well as administrative and networked applications.

Prolog is well-suited for specific tasks that benefit from rule-based logical queries such as searching [databases](https://en.wikipedia.org/wiki/Database), [voice control](https://en.wikipedia.org/wiki/Voice_control) systems, and filling templates.

In Prolog, program logic is expressed in terms of relations, and a computation is initiated by running a query over these relations. Relations and queries are constructed using Prolog's single data type, the term.[[4]](https://en.wikipedia.org/wiki/Prolog" \l "cite_note-lloyd84-4) Relations are defined by clauses. Given a query, the Prolog engine attempts to find a [resolution](https://en.wikipedia.org/wiki/Resolution_(logic)) [refutation](https://en.wikipedia.org/wiki/Refutation) of the negated query. If the negated query can be refuted, i.e., an instantiation for all free variables is found that makes the union of clauses and the singleton set consisting of the negated query false, it follows that the original query, with the found instantiation applied, is a [logical consequence](https://en.wikipedia.org/wiki/Logical_consequence) of the program. This makes Prolog (and other logic programming languages) particularly useful for database, [symbolic mathematics](https://en.wikipedia.org/wiki/Symbolic_mathematics), and language parsing applications. Because Prolog allows impure [predicates](https://en.wikipedia.org/wiki/Predicate_(mathematical_logic)), checking the [truth value](https://en.wikipedia.org/wiki/Truth_value) of certain special predicates may have some deliberate [side effect](https://en.wikipedia.org/wiki/Side_effect_(computer_science)), such as printing a value to the screen. Because of this, the programmer is permitted to use some amount of conventional [imperative programming](https://en.wikipedia.org/wiki/Imperative_programming) when the logical paradigm is inconvenient. It has a purely logical subset, called "pure Prolog", as well as a number of extralogical features.

In prolog, We declare some facts. These facts constitute the Knowledge Base of the system. We can query against the Knowledge Base. We get output as affirmative if our query is already in the knowledge Base or it is implied by Knowledge Base, otherwise we get output as negative. So, Knowledge Base can be considered similar to database, against which we can query. Prolog facts are expressed in definite pattern. Facts contain entities and their relation. Entities are written within the parenthesis separated by comma (, ). Their relation is expressed at the start and outside the parenthesis. Every fact/rule ends with a dot (.). So, a typical prolog fact goes as follows :

Format : relation(entity1, entity2, ....k'th entity).

Example :

friends(raju, mahesh).

singer(sonu).

odd\_number(5).

Explanation :

These facts can be interpreted as :

raju and mahesh are friends.

sonu is a singer.

5 is an odd number.

**Key Features :**  
**1. Unification :** The basic idea is, can the given terms be made to represent the same structure.

**2. Backtracking :** When a task fails, prolog traces backwards and tries to satisfy previous task.

**3. Recursion :** Recursion is the basis for any search in program.

**Running queries :**  
A typical prolog query can be asked as :

Query 1 : ?- singer(sonu).

Output : Yes.

Explanation : As our knowledge base contains   
the above fact, so output was 'Yes', otherwise   
it would have been 'No'.

Query 2 : ?- odd\_number(7).

Output : No.

Explanation : As our knowledge base does not   
contain the above fact, so output was 'No'.

**Advantages :**  
**1.** Easy to build database. Doesn’t need a lot of programming effort.  
**2.** Pattern matching is easy. Search is recursion based.  
**3.** It has built in list handling. Makes it easier to play with any algorithm involving lists.

**Disadvantages :**  
**1.** LISP (another logic programming language) dominates over prolog with respect to I/O features.  
**2.** Sometimes input and output is not easy.

**Applications :**

Prolog is highly used in artificial intelligence(AI). Prolog is also used for pattern matching over natural language parse trees.

**Q. Explain variables, constants with the help of some facts and rules in Prolog and run some queries.**

A Prolog program consists of constants, variables, predicates, facts, and rules.

**CONSTANT** A constant can be numeric, like 1, 3.14159, etc. or symbolic, like sd, sf, etc. The symbolic constants start with lower case alphabets.

**VARIABLE** If we want to ask, "What course does Jeff teach"? This could be written as:

Is there a subject, X, which Jeff teaches?

Here, X, is a variable which stands for an object which the questioner does not know about yet.

To answer the question, Prolog to find out the value of X, if it exists. As long as we do not know the value of a variable it is said to be unbound or uninstantiated. When a value is found, the variable is said to instantiated or bound to the value. The name of a variable must begin with a capital letter or an underscore character, "\_".

To ask Prolog to find the course which Jeff teaches, the following query is entered:

lectures(jeff, X)?

X = 611 output from Prolog

To ask which course(s) Ken teaches, a similar question may be asked,

lectures(ken , X)?

X = 621

X = 643

When there is more than one possible answer, Prolog will try to find all of them.

**FACT** A fact such as, "Jeff lectures in course 611", is written as:

lectures(jeff, 611).

The names of relationships are in lower case letters. The name of the relationship appears as the first term and the objects appears as arguments to a function. A period "." must end a fact. lectures(jeff, 611) is also called a predicate.

Here is a collection of facts about a hypothetical computer science department:

lectures(jeff, 611).

lectures(ken, 621).

lectures(claude, 641).

lectures(graham, 642).

lectures(ken, 643).

studies(fred, 611).

studies(jack, 621).

studies(jill, 641).

studies(jll, 642).

studies(henry, 642).

studies(henry, 643).

year(fred, 1).

year(jack, 2).

year(jill, 2).

year(henry, 3).

Together, these facts for Prolog's database.

**RULES** The previous question can be restated as a general rule:

One person, Teacher, teaches another person, Student if

X lectures in a subject, Subject and

Student studies Subject.

In Prolog this is written as:

teaches(Teacher, Student) :-

lectures(Teacher, Student),

studies(Student, Subject).

This is also called a clause. Facts are unit clauses and rules are non-unit clauses. ":-" means "if" or "is implied by". This symbol is often called the neck symbol. The left hand side of the neck is called the head. The right hand side of the neck is called the body. The comma, ",", separating the goals is stands for and.

Try this rule out:

more\_advanced(Student1, Student2) :-

year(Student1, Year1),

year(Student2, Year2),

Year1 > Year2.

Note the use of the predefined predicate ">".

**Q. Program of weather (City, Season, Temperature) with how to print output?**

import requests

from pprint import pprint

def weather\_data(query):

res=requests.get('http://api.openweathermap.org/data/2.5/weather?'+query+'&APPID=\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*8&units=metric');

return res.json();

def print\_weather(result,city):

print("{}'s temperature: {}°C ".format(city,result['main']['temp']))

print("Wind speed: {} m/s".format(result['wind']['speed']))

print("Description: {}".format(result['weather'][0]['description']))

print("Weather: {}".format(result['weather'][0]['main']))

def main():

city=input('Enter the city:')

print()

try:

query='q='+city;

w\_data=weather\_data(query);

print\_weather(w\_data, city)

print()

except:

print('City name not found...')

if \_name=='main\_':

main()

**Q. Explain Structure with program and queries.**

Functional terms can be used to construct complex datastructures. For example, if we want to say that John owns the book The Hitchhicker's Guide to the Galaxy, this may be expressed as:

owns(john, "The Hitchhicker's Guide to the Galaxy").

This statement tells us very little about the book. Often we ascribe to objects a number of attributes. In this case, we would like to represent a book as having a title and an author. A structured object such a book can be represented by a functional term as follows:

owns(john, book("The Hitchhicker's Guide to the Galaxy", adams)).

Now the second argument to the predicate shows us more detail. Adams is avery common name, so to be more accurate we should give the author's family and given names.

owns(john, book("The Hitchhicker's Guide to the Galaxy",

author(adams,douglas))).

How do we ask, "What books does John own which were written by someone called "Adams"?

owns(john, book(Title, author(adams, GivenName))?

Title = " The Hitchhicker's Guide to the Galaxy "

GivenName = douglas

Notice that in order to find a fact in the database which would answer the question, Prolog performed a quite complex matching operation between the structures in the query and those in the clause head.

Here is a more complicated example of the use of structures in Prolog. It is also out first moderately useful set of prolog programs. A database of books in a library contains facts of the form

book(CatalogNumber, Title, author(FamilyName, GivenName)).

member(MemberNumber, name(FamilyName, GivenName), Address).

loan(CatalogNumber, MemberNumber, BorrowDate, DueDate).

A member of the library may borrow a book. When this is done, a "loan" is entered into the database recording the catalogue number of the book which was borrowed and the number of the member who borrowed it. The date at which the book was borrowed and the due date are also recorded. Dates are stored as structures of the form date(Year, Month, Day). For example date(86, 6, 16) represents 16 June 1986. Names and address are all stored as character strings (i.e. atoms).

The first program we write tells us which books a member has borrowed:

has\_borrowed(MemberFamilyName, Title, CatalogNumber) : -

member(MemberNumber, name(MemberFamilyName, \_), \_),

loan(CatalogNumber, MemberNumber, \_, \_),

book(CatalogNumber, Title, \_).

Next we would like to know which books are overdue but before we can get started on this program we first have to work out how to compare dates. The following predicate tells us when the first date comes after the second.

later(date(Year, Month, Day1), date(Year, Month, Day2)) :- !,

Day1 > Day2.

later(date(Year, Month1, \_), date(Year, Month2, \_)) :- !,

Month1 > Month2.

later(date(Year1, \_, \_), date(Year2, \_, \_)) :-

Year1 > Year2.

Note that the cuts indicate that if Prolog has found a clause head which matches the goal then there is no need to consider the following clauses. Also notice that the program uses comparison operators. These operators can be thought of predicates that have been predefined in the Prolog system.

overdue(DateToday, Title, CatalogNumber, MemberFamilyName) :-

loan(CatalogNumber, MemberNumber, \_, DueDate),

later(DueDate, DateToday),

book(CatalogNumber, Title, \_),

member(MemberNumber, name(MemberFamilyName, \_), \_).

For the final program in this example, let's write a rule which will help us to determine that date on which a book is due.

due\_date(date(Year, Month1, Day), date(Year, Month2, Day)) :-

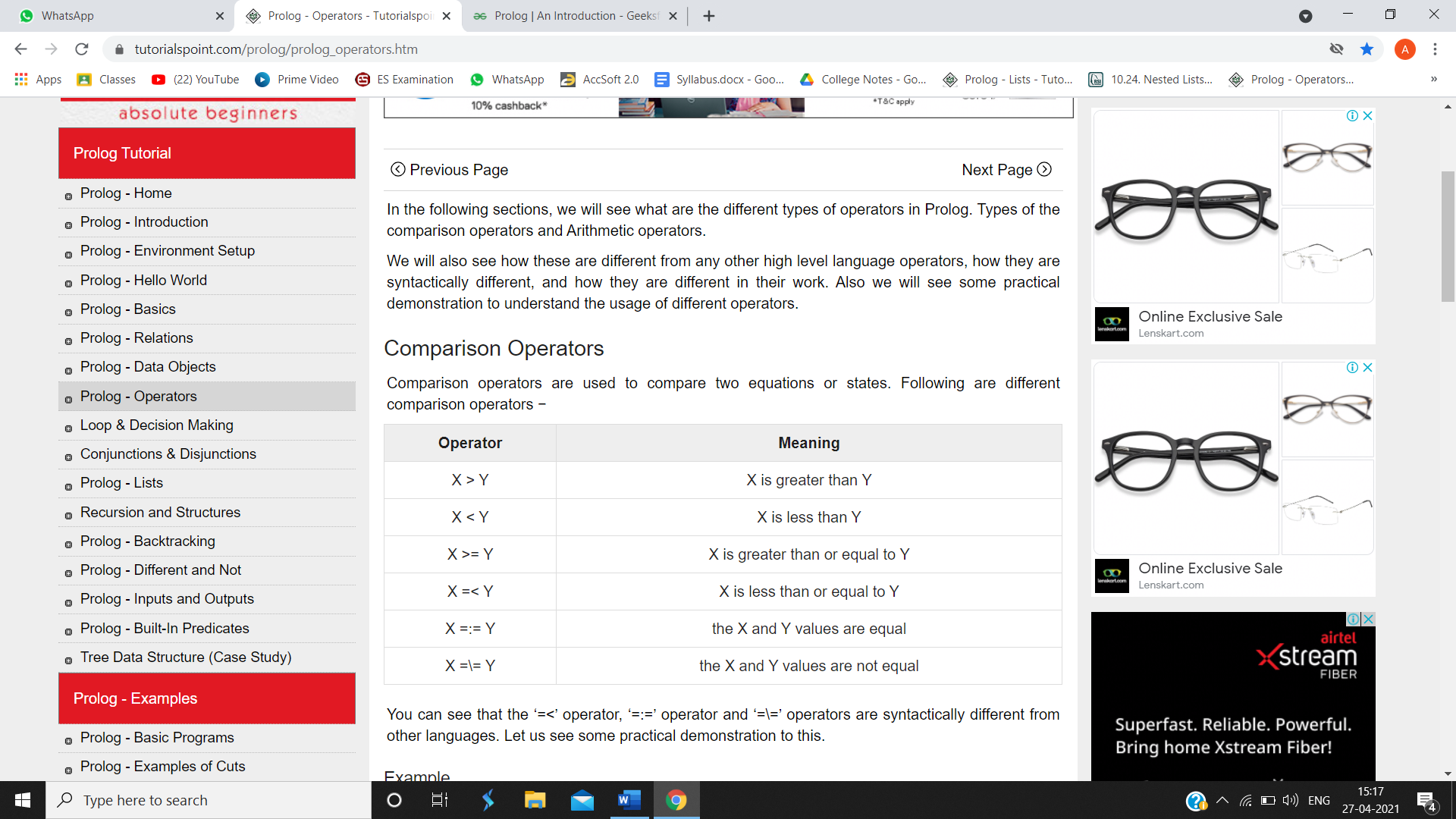
Month2 is Month1 + 1.

This rule illustrates the use of another arithmetic predefined predicate, namely, the is operator. Is accepts two arguments. The right hand argument must be a term which can be treated as an arithmetic expression. This term is evaluated and then unified with the left hand argument.

**Q. Explain Operators in Prolog.**

## **Comparison Operators**

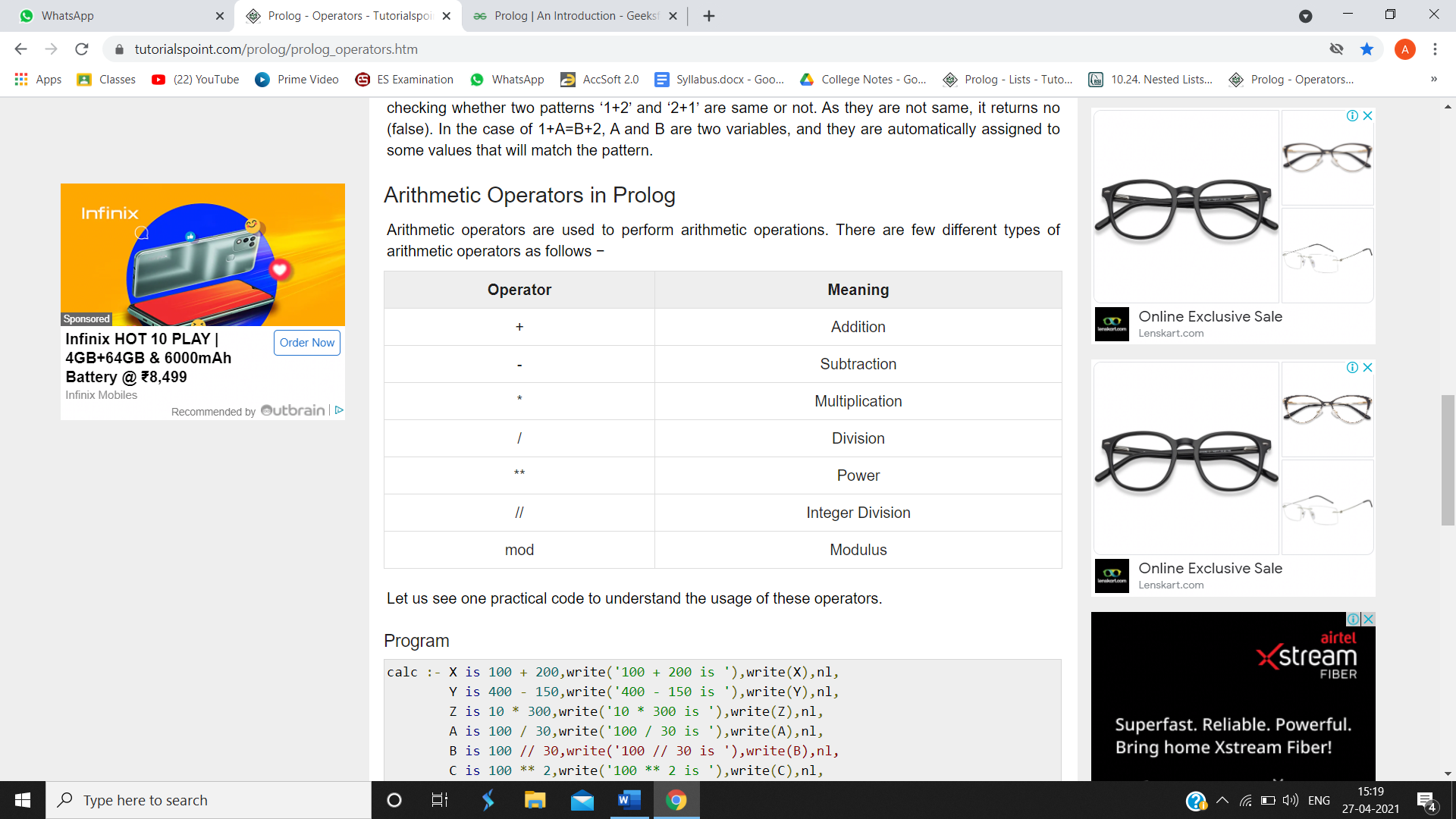
Comparison operators are used to compare two equations or states. Following are different comparison operators −



You can see that the ‘=<’ operator, ‘=:=’ operator and ‘=\=’ operators are syntactically different from other languages. Let us see some practical demonstration to this.

## **Arithmetic Operators in Prolog**

Arithmetic operators are used to perform arithmetic operations. There are few different types of arithmetic operators as follows −



**Q. Write a program to convert Fahrenheit temperature to degree Celsius.**

DegreesF = (9 / 5 \* DegreesC) + 32

DegreesC = 5 / 9 \* (DegreesF - 32)

Write two Prolog rules: one called c2f(C, F) to convert from Celsius to Fahrenheit and one called f2c(F, C) to convert from Fahrenheit to Celsius.

Make sure your rules work exactly as shown for the following queries:

?- c2f(20, F).

F = 68

?- f2c(98.6, C).

C = 37

Write a Prolog rule called convert(C, F) that first checks which temperature (C or F) is given, and then does the appropriate conversion. That is, when C is given, convert(C, F) uses c2f(C, F). When F is given it uses f2c(F, C).

Make sure your rule works exactly as shown for the following queries:

?- convert(20, F).

F = 68

?- convert(C, 98.6).

C = 37

**Q. Explain List in Prolog. What is nested list?**

A very important type of recursive term is the list. The recursive definition of a list is: a list may be nil or it may be a term which has a head, which can be any term, and a tail which is another list. Using standard Prolog notation, we could define the list as:

is\_list(nil).

is\_list(list(Head, Tail)) :-

list(Tail).

A list of numbers [1, 2, 3] would look like:

list(1, list(2, list(3, nil)))

Although this notation is consistent with the way Prolog treats all other data structures, it can be rather clumsy at times. Because lists are used so often, most Prolog implementations use the alternative, more convenient notation, [1, 2, 3]. Internally, Prolog still stores the list as if it were entered in the prefix form.

To get some idea of how the compact list notation works look at the following queries to Prolog and the answers the system returns:

[X, Y, Z] = [1, 2, 3]?

X = 1

Y = 2

Z = 3

This query asks Prolog to match (or unify) the two terms on either side of the equals sign. If a variable appears in a position corresponding to an element in the second list then that variable is unified with the element.

[X | Y] = [1, 2, 3]?

X = 1

Y = [2, 3]

The most common procedure for processing a list is to find the first element, that is the head of the list, perform an operation on it and then repeat the process for all the elements remaining in the tail. The head and tail may be separated by using the vertical bar '|' to indicate that the term following the bar should unify with the tail of the list. Remember that the tail is usually also a list.

[X | Y] = [1]?

X = 1

Y = []

An empty list is written as '[]'. The end of a list is indicated by an empty tail. That is, the tail is []. If necessary, several elements from the front of the list can be selected before matching the tail.

[X, Y | Z] = [fred, jim, jill, mary]?

X = fred

Y = jim

Z = [jill, mary]

In this example there must be at least two elements in the list on the right in order for the goal to succeed. List elements can be as complex as desired.

[X | Y] = [[a, f(e)], [n, m, [2]]]?

X = [ a, f(e)]

Y = [[n, m, [2]]]

A nested list is a list that appears as an element in another list. In this list, the element with index 3 is a nested list. If we print(nested[3]), we get [10, 20]. To extract an element from the nested list, we can proceed in two steps. First, extract the nested list, then extract the item of interest. It is also possible to combine those steps using bracket operators that evaluate from left to right.

nested = ["hello", 2.0, 5, [10, 20]]

innerlist = nested[3]

print(innerlist)

item = innerlist[1]

print(item)

print(nested[3][1])

**Q. Explain Member functions in Prolog.**

To perform mathematical operations on numbers, we will need functions. To store the result of a mathematical operation in a variable, we will need to look more closely at equality.

Functions Edit

Until now, predicates have always represented a simple true or false. Predicate a(A, B) is true or false, depending on the values of A and B. Functions are predicates that represent a value. The sin() predicate, for instance, is a function. sin(0) represents the value 0 and sin(1) represents the value 0.841471. Functions can be used anywhere a number or constant can be used, in queries, predicates and rules. For instance, if the fact p(0). is in your program, the query ?- p(sin(0)). will unify with it.

The following common mathematical functions are built in to most Prolog implementations:

function example result

+ 2 + 3 5

- 36 - 5 31

\* 4 \* 3 12

/ 36/5 7.2

^ 4 ^ 2 16

sin sin(3) 0.14112

(table to be completed)

Note that functions themselves cannot be evaluated. the query ?- sin(3). will fail because sin() is implemented as function and not as a predicate.

One difference between functions and predicates is that the meaning (or definition) of a predicate is usually defined by you, in your program. When you use functions like sin(), they've already been defined in your prolog implementation. In other words, prolog will not find the definition in your program, but in it's library of built-in predicates. It is possible to create your own functions, but that's something you will usually not need.

**Q. Propose one AI based start-up which you can implement as your project.**

The mobile attendance system has been built to eliminate the time and effort wasted in taking attendances in schools and colleges. It also greatly reduces the amount of paper resources needed in attendance data management. This is an android mobile app. It’s built to be used for school/college faculty so that they may take student attendance on their phones.  
The system is divided into following modules:

* **Student Attendance List Creation:** Once this App is installed on a phone, a it allows user to create a student attendance sheet consisting of name, roll number, date, Absent/Present mark and subject. He has to fill student names along with associated roll numbers.
* **Attendance Marking:** The faculty has the list on his phone now. He may see the list call roll numbers and select absent id the student is absent or select present if student is present.
* **Attendance Storage:** This data is now stored in the faculty mobile phone. Faculty may also view it anytime on their phone.
* **Attendance sheet transfer:** The faculty can transfer the file to a server (normal computer) via a bluetooth connection where this data can be stored and maintained by the school or college.

Thus this system automates attendance system and eliminates the use of paperwork needed for attendance marking and monitoring student attendance.

##### **Advantages**

* The system eliminates the use of paperwork needed for attendance marking and monitoring.
* The file can be transferred from mobile to computer or server via Bluetooth.
* This gives the overall performance of class in attendance.
* There is no need for laptop or computer in every class to run the system as the system is run on mobile so no need of extra efforts and resources.
* The app is easy to install and use.

##### **Disadvantages**

* The system can be run on android platform only. Though most of the mobiles now are android version and available in reasonable rate so it won’t be a big issue.