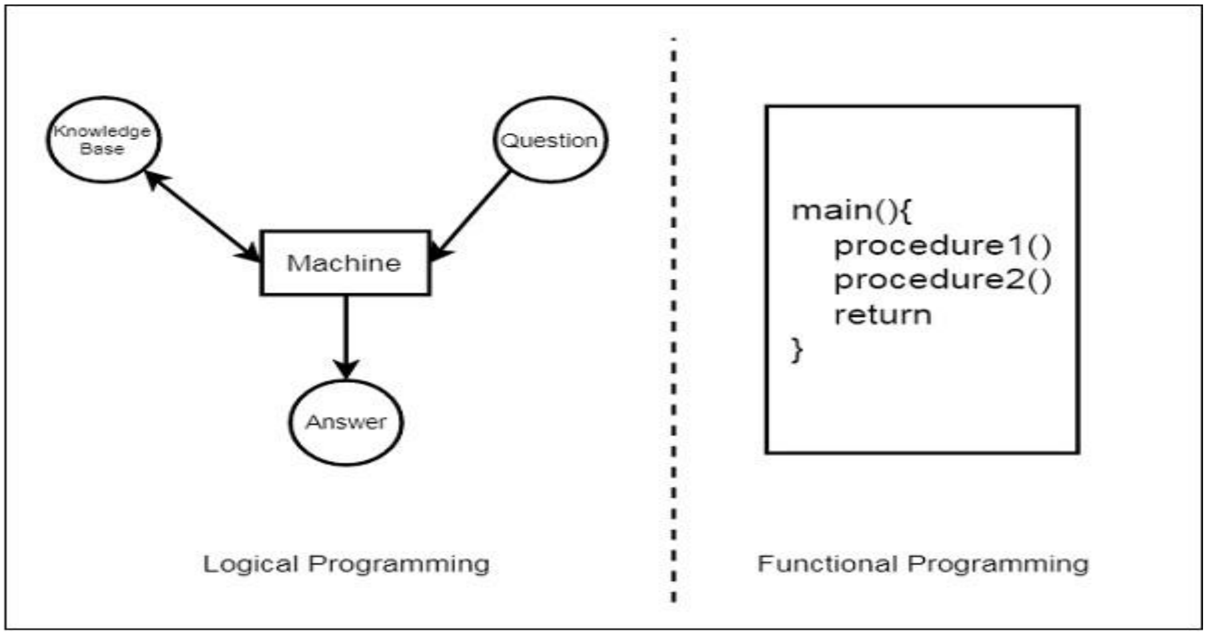
Logic vs. Functional Programming

We will discuss about the differences between Logic programming and the traditional functional programming languages. We can illustrate these two using the below diagram −



```
main(){
   procedure1()
   procedure2()
   return
}
```

From this illustration, we can see that in Functional Programming, we have to define the procedures, and the rule how the procedures work. These procedures work step by step to solve one specific problem based on the algorithm. On the other hand, for the Logic Programming, we will provide knowledge base. Using this knowledge base, the machine can find answers to the given questions, which is totally different from functional programming.

In functional programming, we have to mention how one problem can be solved, but in logic programming we have to specify for which problem we actually want the solution. Then the logic programming automatically finds a suitable solution that will help us solve that specific problem.

Imperative Programming vs. Functional Programming

Imperative programming is a paradigm of computer programming where the program describes steps that change the state of the computer. Unlike declarative programming, which describes "what" a program should accomplish, imperative programming explicitly tells the computer "how" to accomplish it. Programs written this way often compile to binary executables that run more efficiently since all CPU instructions are themselves imperative statements.

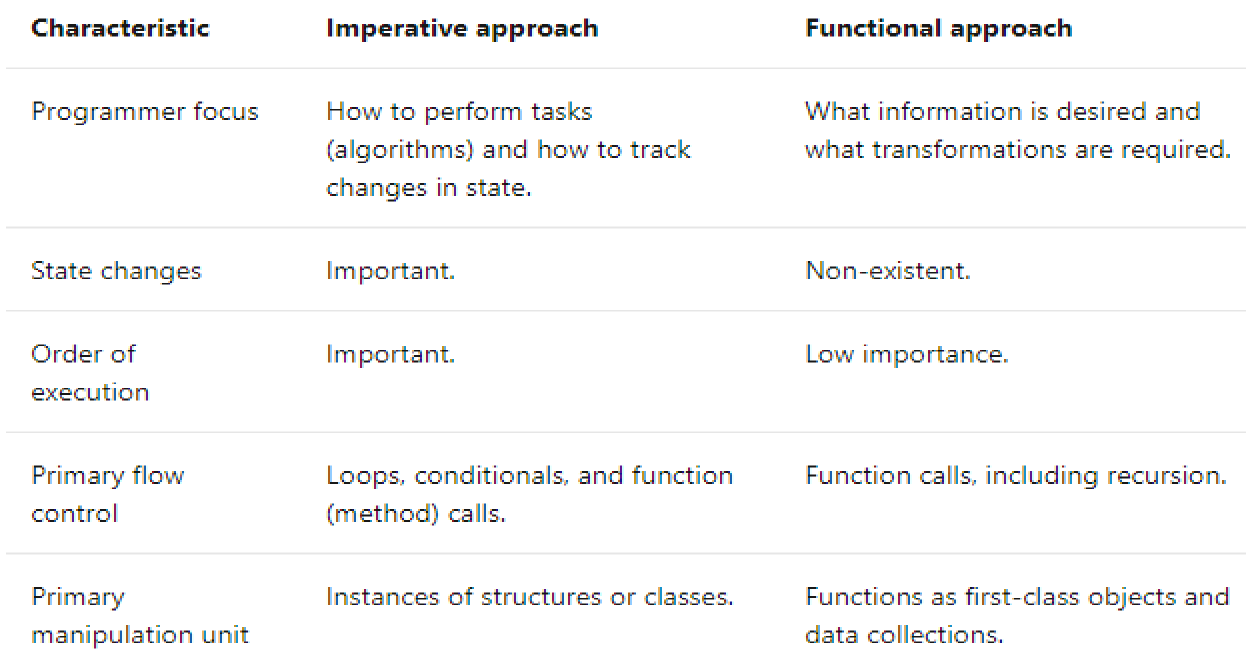
To make programs simpler for a human to read and write, imperative statements can be grouped into sections known as code blocks. In the 1950s, the idea of grouping a program's code into blocks was first implemented in the ALGOL programming language. They were originally called "compound statements," but today these blocks of code are known as procedures. Once a procedure is defined, it can be used as a single imperative statement, abstracting the control flow of a program. The process allows the developer to express programming ideas more naturally. This type of imperative programming is called procedural programming, and it is a step towards higher-level abstractions such as declarative programming.

Imperative programming languages

- Ada
- ALGOL
- Assembly language
- BASIC
- C
- C#
- C++
- COBOL
- FORTRAN
- JAVA
- MATLAB
- Python

The *functional programming* paradigm was explicitly created to support a pure functional approach to problem solving. Functional programming is a form of *declarative programming*. In contrast, most mainstream languages, including object-oriented programming (OOP) languages such as C#, Visual Basic, C++, and Java, were designed to primarily support *imperative* (procedural) programming.

With an imperative approach, a developer writes code that specifies the steps that the computer must take to accomplish the goal. This is sometimes referred to as *algorithmic* programming. In contrast, a functional approach involves composing the problem as a set of functions to be executed. You define carefully the input to each function, and what each function returns. The following table describes some of the general differences between these two approaches.



Characteristic	Imperative approach	Functional approach
Programmer focus	How to perform tasks (algorithms) and how to track changes in state.	What information is desired and what transformations are required.
State changes	Important.	Non-existent.
Order of execution	Important.	Low importance.
Primary flow control	Loops, conditionals, and function (method) calls.	Function calls, including recursion.
Primary manipulation unit	Instances of structures or classes.	Functions as first-class objects and data collections.

PROLOG

Prolog is an AI programming language. It belongs to the family of logic programming languages. Prolog is a declarative language, in which computations are carried over by running queries over the relations (which represent program logic), which are defined as rules and fact. Developed in 1970, prolog is one of the oldest logic programming languages and one of the most popular AI programming languages today (along with Lisp). It is a free language, but many commercial variants are available. It was first used for natural language processing, but now it is been used for various tasks such as expert systems, automated answering systems, games and advanced control systems. Prolog has only one data type called the term. A Term can be an atom, number, variable or a compound term. Numbers can be float or integers. Prolog supports lists and string as collection of items. Prolog defines relations using clauses. Clauses can be either rules or facts. Prolog allows iteration thorough its recursive predicates.

Prolog or PROgramming in LOGics is a logical and declarative programming language. It is one major example of the fourth generation language that supports the declarative programming paradigm. This is particularly suitable for programs that involve symbolic or non-numeric computation. This is the main reason to use Prolog as the programming language in Artificial Intelligence, where symbol manipulation and inference manipulation are the fundamental tasks.

In Prolog, we need not mention the way how one problem can be solved, we just need to mention what the problem is, so that Prolog automatically solves it. However, in Prolog we are supposed to give clues as the solution method.

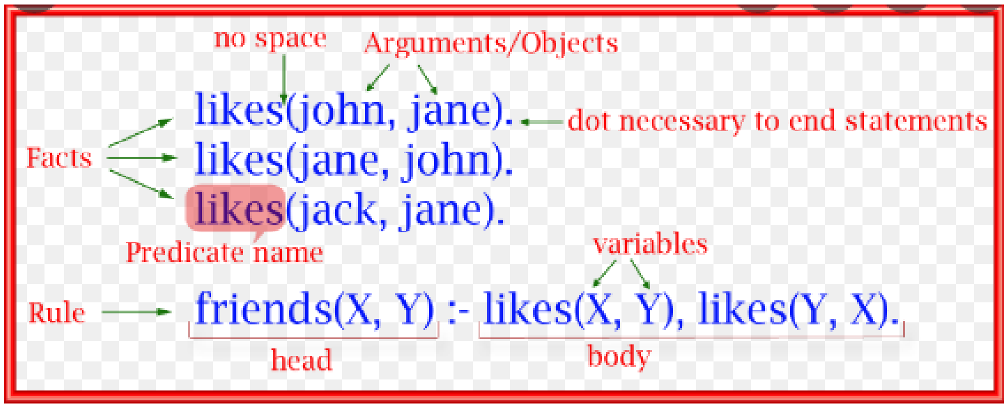
Prolog language basically has three different elements −

Facts − The fact is predicate that is true, for example, if we say, “Tom is the son of Jack”, then this is a fact.

Rules − Rules are extinctions of facts that contain conditional clauses. To satisfy a rule these conditions should be met. For example, if we define a rule as −

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 father of Z.



Other 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.

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'

The heritage of prolog includes the research on theorem provers and some other automated deduction system that were developed in 1960s and 1970s. The Inference mechanism of the Prolog is based on Robinson’s Resolution Principle, that was proposed in 1965, and Answer extracting mechanism by Green (1968). These ideas came together forcefully with the advent of linear resolution procedures.

The explicit goal-directed linear resolution procedures, gave impetus to the development of a general purpose logic programming system. The **first** Prolog was the **Marseille Prolog** based on the work by **Colmerauer** in the year 1970. The manual of this Marseille Prolog interpreter (Roussel, 1975) was the first detailed description of the Prolog language.

Prolog is also considered as a fourth generation programming language supporting the declarative programming paradigm. The well-known Japanese Fifth-Generation Computer Project, that was announced in 1981, adopted Prolog as a development language, and thereby grabbed considerable attention on the language and its capabilities.

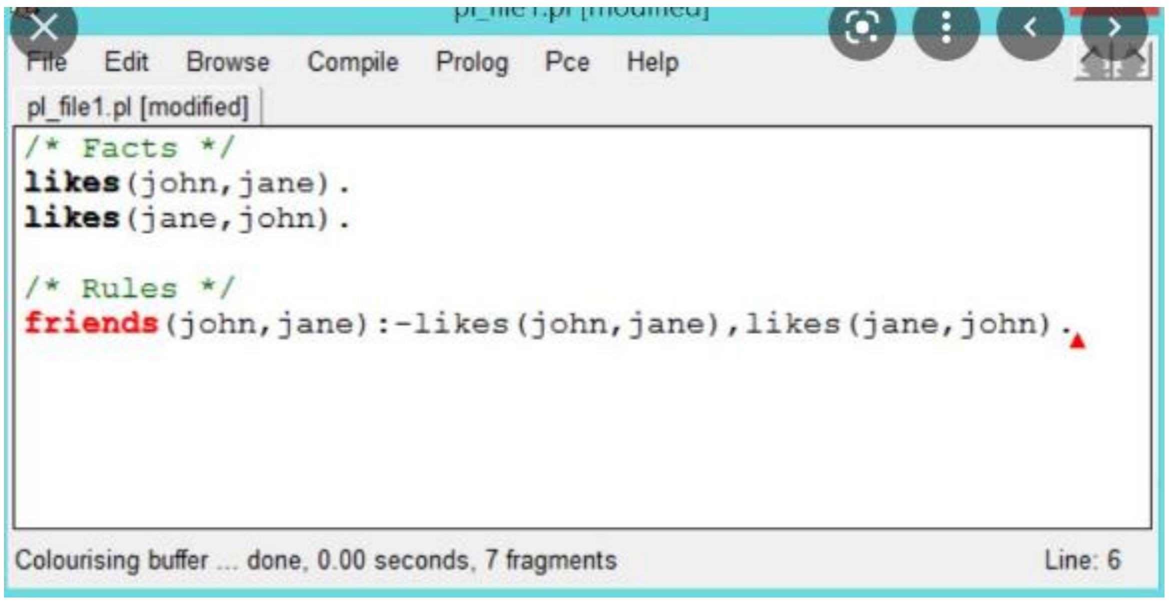
Some Applications of Prolog

Prolog is used in various domains. It plays a vital role in automation system. Following are some other important fields where Prolog is used −

- Intelligent Database Retrieval
- Natural Language Understanding
- Specification Language
- Machine Learning
- Robot Planning
- Automation System

- Problem Solving

Prolog is a logic programming language associated with AI and computational linguistics. It has its roots in first-order logic, the formal logic, unlike many other programming languages. Prolog is mainly a declarative programming language. It is possible to express the program logic as a set of relations, facts and rules. Therefore, a computation can be initiated by running a query over these relations.



```
/* Facts */
likes(john,jane).
likes(jane,john).

/* Rules */
friends(john,jane):-likes(john,jane),likes(jane,john).
```

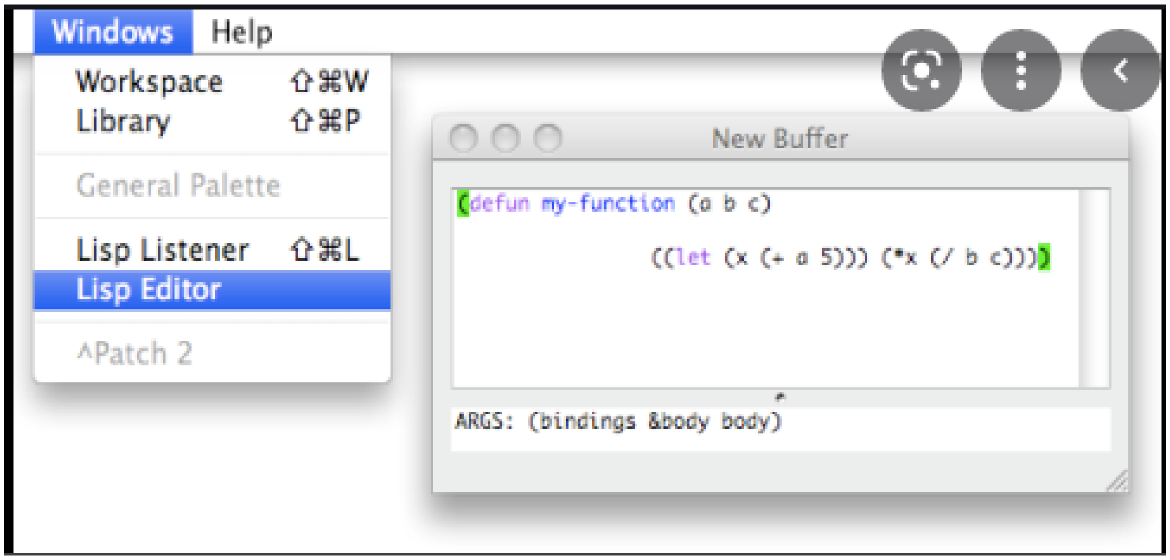
```
Colourising buffer ... done, 0.00 seconds, 7 fragments                Line: 6
```

Prolog was one of the first logic programming languages. It helps various tasks such as theorem proving, expert systems, term rewriting, type systems, natural language processing and automated planning. It also helps to create GUIs, administrative and networked applications. Furthermore, Prolog is suitable for rule-based logical queries such as searching databases, filling templates and voice control systems.

LISP

Lisp is a family of computer programming languages. And the most famous Lisp dialects used for general purpose programming today are Common Lisp and Scheme. The name LISP comes from “LISt Processing” and as it hints, Lisp’s major data structure is the linked list. In fact the whole source is written using lists (using prefix notation), or more correctly parenthesized lists (called s-expressions). For example, a function call is written as (f a1 a2 a3), which means function f is called using a1, a2 and a3 as input arguments for the function. Therefore it is called an expression oriented language, where all data and code are written as expressions (there is no distinction between expressions and statements in Lisp). This nice feature is very special to Lisp, where it could be used to extend the language to the problem at hand by writing helpful macros. Although tail-recursion is used by programmers to express loops, all frequently seen Lisp dialects do include control structures like loop. Furthermore, Common Lisp and scheme have mapcar and map that are examples of functions, which provide looping functionality by applying the function successively to all its elements and then collects the results in to a list.

Lisp is a computer programming language with a long history and a distinctive, fully parenthesized prefix notation. The programmer writes all program code in s-expressions or parenthesized lists. Furthermore, a function call or syntactic form can be written as a list with the function or operator's name first.



```
(defun my-function (a b c)
                    ((let (x (+ a 5))) (*x (/ b c)))
```

```
ARGS: (bindings &body body)
```

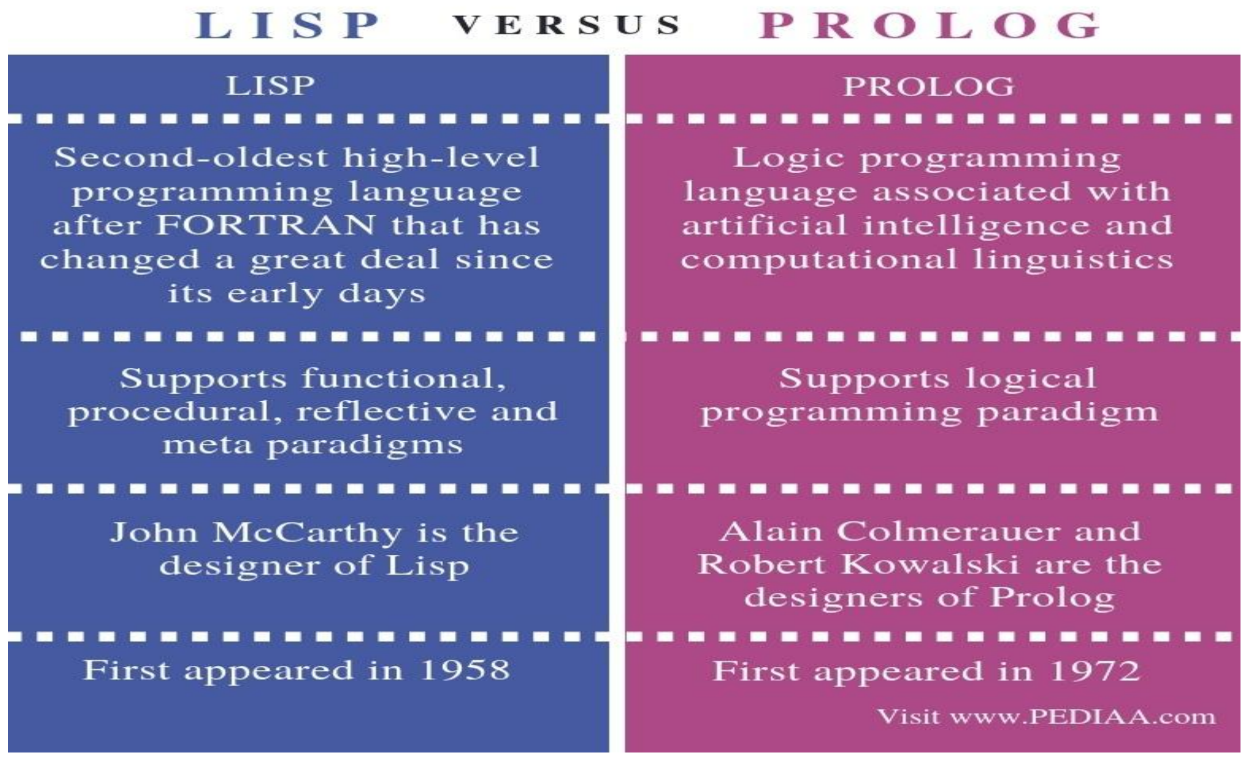
Lisp is an old, high-level programming language. The main objective of using Lisp is to represent mathematical notations for computer programs. Some popular Lisp dialects are Clojure, Common Lisp and Scheme. Moreover, it also helps to develop Artificial Intelligence (AI) applications.

LISP Vs. Prolog

Prolog and Lisp are two of the most popular AI (Artificial Intelligence) computer programming languages today. They are built with two different programming paradigms. Prolog is a declarative language, while Lisp is a functional language. Both are used for various AI problems but Prolog is used most for logic and reasoning problems, while Lisp is used for problems with rapid prototyping needs.

Although, Prolog and Lisp are two of the most popular AI programming languages, they have various differences. Lisp is a functional language, while Prolog is a logic programming and declarative languages. Lisp is very flexible due to its fast prototyping and macro features, so it actually allows extending the language to suit the problem at hand. In the areas of AI, graphics and user interfaces, Lisp has been used extensively because of this rapid prototyping ability. However, due to its inbuilt logic programming abilities, Prolog is ideal for AI problems with symbolic reasoning, database and language parsing applications. Choice of one over the other completely depends on the type of AI problem that need to be solved.

The main difference between Lisp and Prolog is that Lisp is a computer program language that supports functional, procedural, reflective and meta paradigms while Prolog is a computer programming language that supports logic programming paradigm.
Generally, Artificial Intelligence (AI) is a way of making a computer, robot, software or a machine to work intelligently similar to a human. It is a discipline that covers various fields, including Mathematics, Computer Sciences, Engineering, Philosophy etc. Overall, Lisp and Prolog are two programming languages that help to write and develop AI-based applications. Furthermore, Lisp is an older language than Prolog.



LISP VERSUS PROLOG

LISP	PROLOG
Second-oldest high-level programming language after FORTRAN that has changed a great deal since its early days	Logic programming language associated with artificial intelligence and computational linguistics
Supports functional, procedural, reflective and meta paradigms	Supports logical programming paradigm
John McCarthy is the designer of Lisp	Alain Colmerauer and Robert Kowalski are the designers of Prolog
First appeared in 1958	First appeared in 1972

Visit www.PEDIAA.com

Introduction Prolog Operator

- The prolog operator is a function to operate and works on the programming operand or variable.
- The prolog operator is a symbolic character to work arithmetic, logical, and comparison operations.
- It is a symbol to take action between two values and objects for a programming language.
- The prolog operator is an expression to perform two or more than the two values in the arithmetic, logical, comparison, and another format.
- The prolog operator is a symbolic function to operate the “pl” files value in the console.
- The prolog operator is a function for the prolog console and operates files variable, objects, and given operand.

Prolog Operator Types

- The prolog operator is a function to work arithmetic, logic, comparison, and other operations.
- The prolog operator categorizes several operators for different operations.

- It’s types and its subcategory are given below.

Prolog Arithmetic operator

This operator is working for arithmetic expressions such as addition, subtraction, and so on.

The arithmetic operator types shown below.

- Addition (+) operator
- Subtraction (-) operator
- Multiplication (*) operator
- Division (%) operator
- Power (**) operator
- Integer division (//) operator
- Modulus (mod) operator
- Square root (sqrt) operator
- maximum (max) operator

Prolog Comparison operator:

- This operator is working for comparison between two operands and variables such as equality.

The comparison operator types shown below.

- Greater than (>)operator
- Less than (<) operator
- Greater than equal to (>=) operator

- Less than equal to (=<) operator
- Equal (=:=) operator
- Not equal (=\=) operator

Trigonometric operator:

• This operator works for operating operand and variable to find tangent and cotangent values.

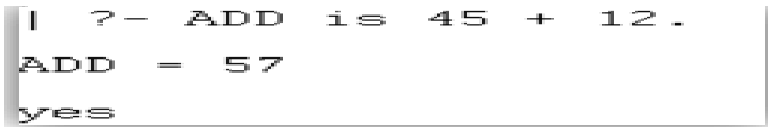
• The trigonometric operator types are shown below.

1. SIN operator
2. COS operator

Explain Operator Types with Example

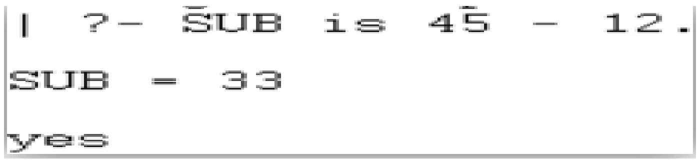
Prolog Arithmetic operator:

Addition (+) operator:



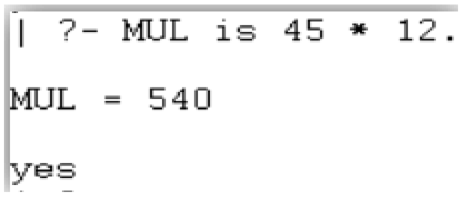
```
| ?- ADD is 45 + 12.
ADD = 57
yes
```

Subtraction (-) operator:

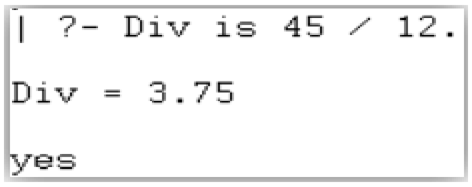


```
| ?- SUB is 45 - 12.
SUB = 33
yes
```

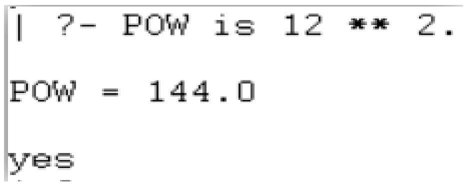
Multiplication (*) operator:

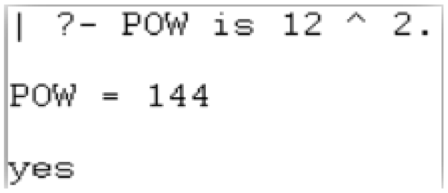

```
| ?- MUL is 45 * 12.

MUL = 540

yes
```

Division (/) operator:

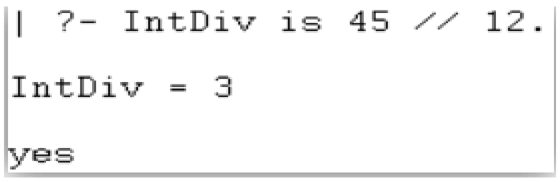

```
| ?- Div is 45 / 12.

Div = 3.75

yes
```

Power (**) operator:

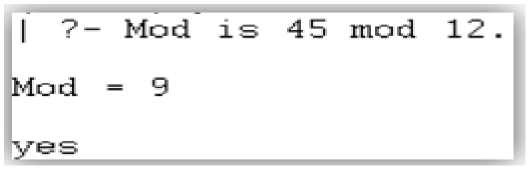

```
| ?- POW is 12 ** 2.

POW = 144.0

yes
```


```
| ?- POW is 12 ^ 2.

POW = 144

yes
```

Integer division (//) operator:


```
| ?- IntDiv is 45 // 12.

IntDiv = 3

yes
```

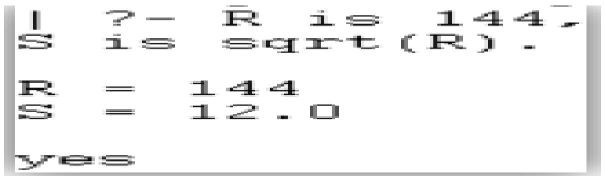
Modulus (mod) operator:



```
| ?- Mod is 45 mod 12.

Mod = 9

yes
```

Square root (sqrt) operator:

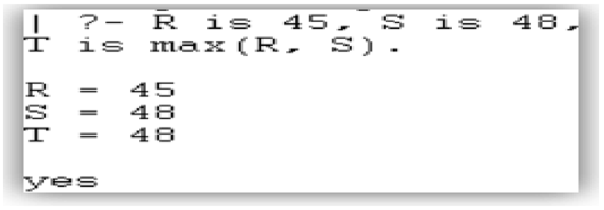
Output:



```
| ?- R is 144,
S is sqrt(R).

R = 144
S = 12.0

yes
```

maximum (max) operator:

Output:



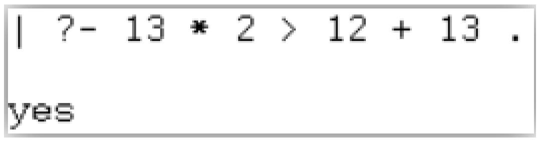
```
| ?- R is 45, S is 48,
T is max(R, S).

R = 45
S = 48
T = 48

yes
```

Prolog comparison operator:

The comparison operator types examples show below.

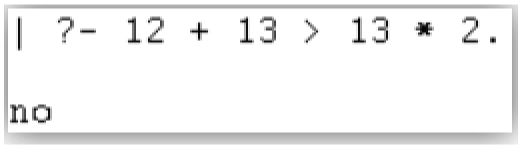
Greater than (>) operator

Example 1



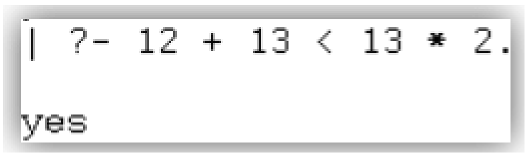
```
| ?- 13 * 2 > 12 + 13 .

yes
```

Example 2



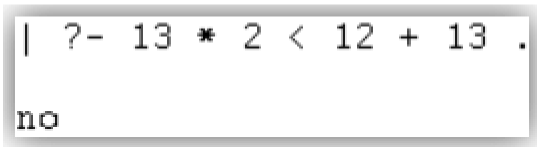
```
| ?- 12 + 13 > 13 * 2.

no
```

Less than (<) operator



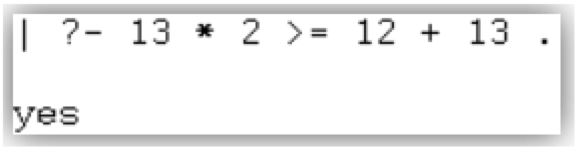
```
| ?- 12 + 13 < 13 * 2.

yes
```

Example 2



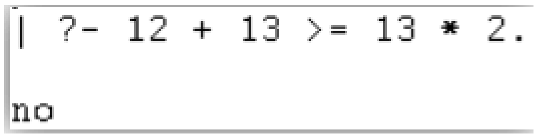
```
| ?- 13 * 2 < 12 + 13 .

no
```

Greater than equal to (>=) operator

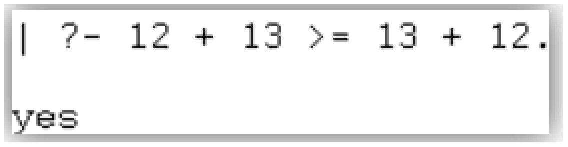


```
| ?- 13 * 2 >= 12 + 13 .

yes
```

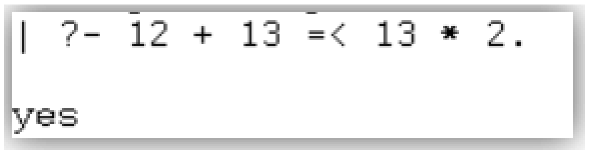
Example 2


```
| ?- 12 + 13 >= 13 * 2.

no
```

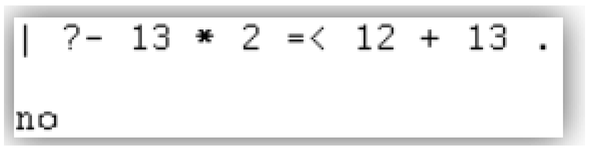
Example 3


```
| ?- 12 + 13 >= 13 + 12.

yes
```

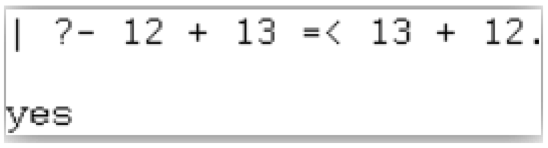
Less than equal to (=<) operator


```
| ?- 12 + 13 =< 13 * 2.

yes
```

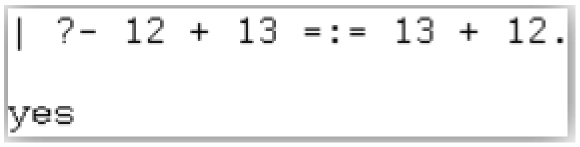
Example 2


```
| ?- 13 * 2 =< 12 + 13 .

no
```

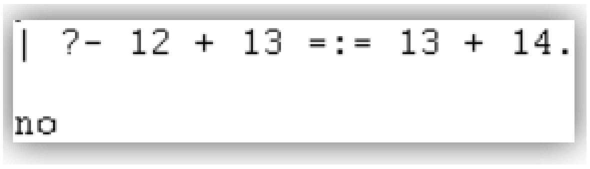
Example 3


```
| ?- 12 + 13 =< 13 + 12.

yes
```

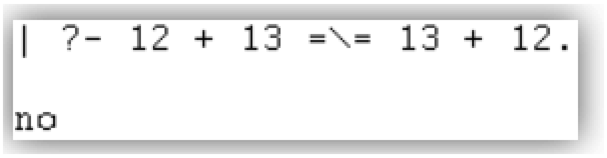
Equal (=:=) operator


```
| ?- 12 + 13 =:= 13 + 12.

yes
```

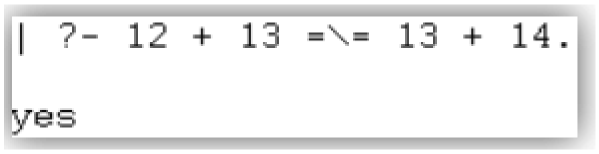
Example 2


```
| ?- 12 + 13 =:= 13 + 14.

no
```

Not equal (=\=) operator


```
| ?- 12 + 13 =\= 13 + 12.

no
```

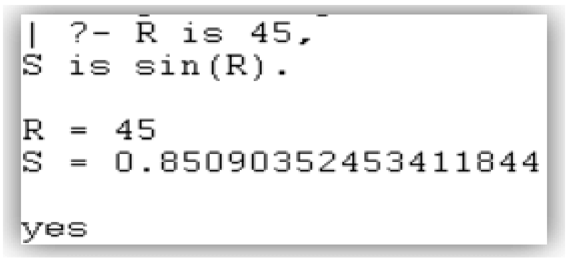
Example 2


```
| ?- 12 + 13 =\= 13 + 14.

yes
```

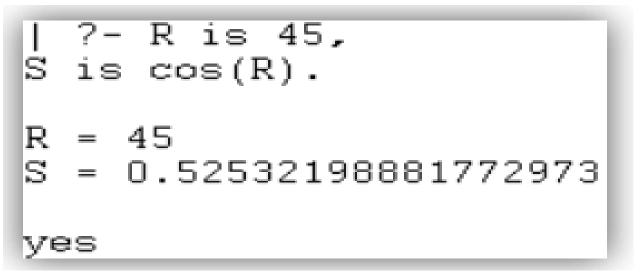
Prolog trigonometric operator:

The trigonometric operator types examples show below.

sin() operator


```
| ?- R is 45,
S is sin(R).

R = 45
S = 0.85090352453411844

yes
```

cos() operator



```
| ?- R is 45,
S is cos(R).

R = 45
S = 0.52532198881772973

yes
```