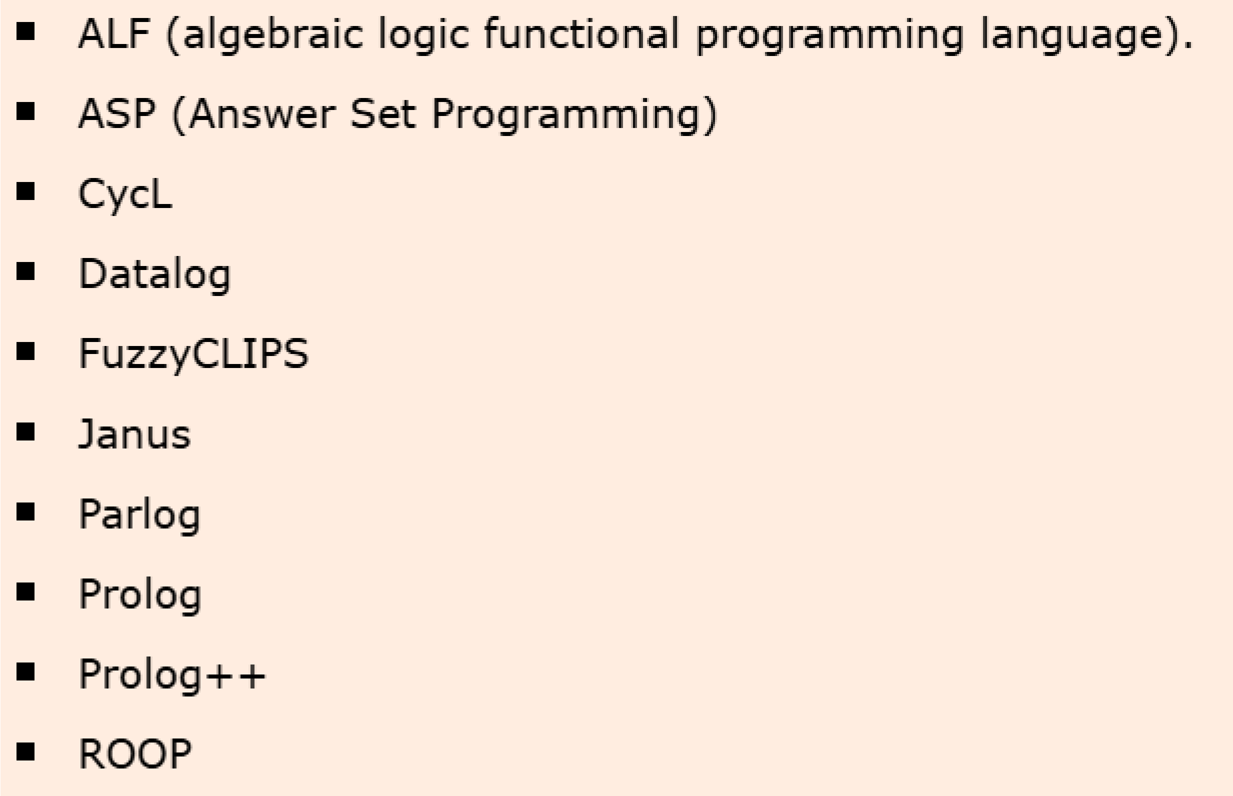
PROLOG

ProLog: Programming in Logic

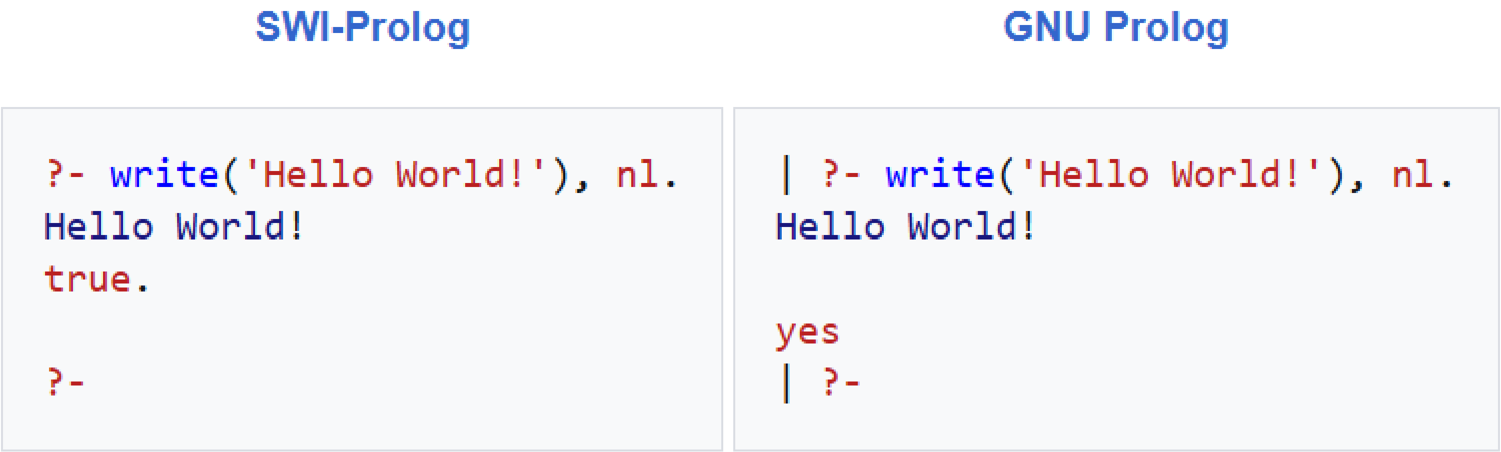
Logic Programming is one of the Computer Programming Paradigm, in which the program statements express the facts and rules about different problems within a system of formal logic.



- ALF (algebraic logic functional programming language).
- ASP (Answer Set Programming)
- CycL
- Datalog
- FuzzyCLIPS
- Janus
- Parlog
- Prolog
- Prolog++
- ROOP

SWI Prolog

GNU Prolog



SWI-Prolog

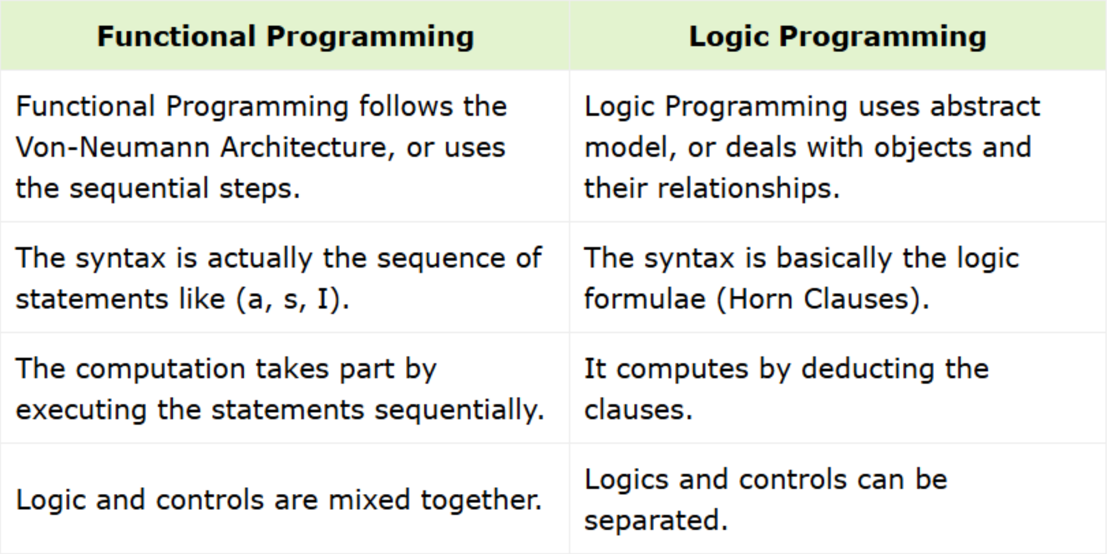
```
?- write('Hello World!'), nl.
Hello World!
true.

?-
```

GNU Prolog

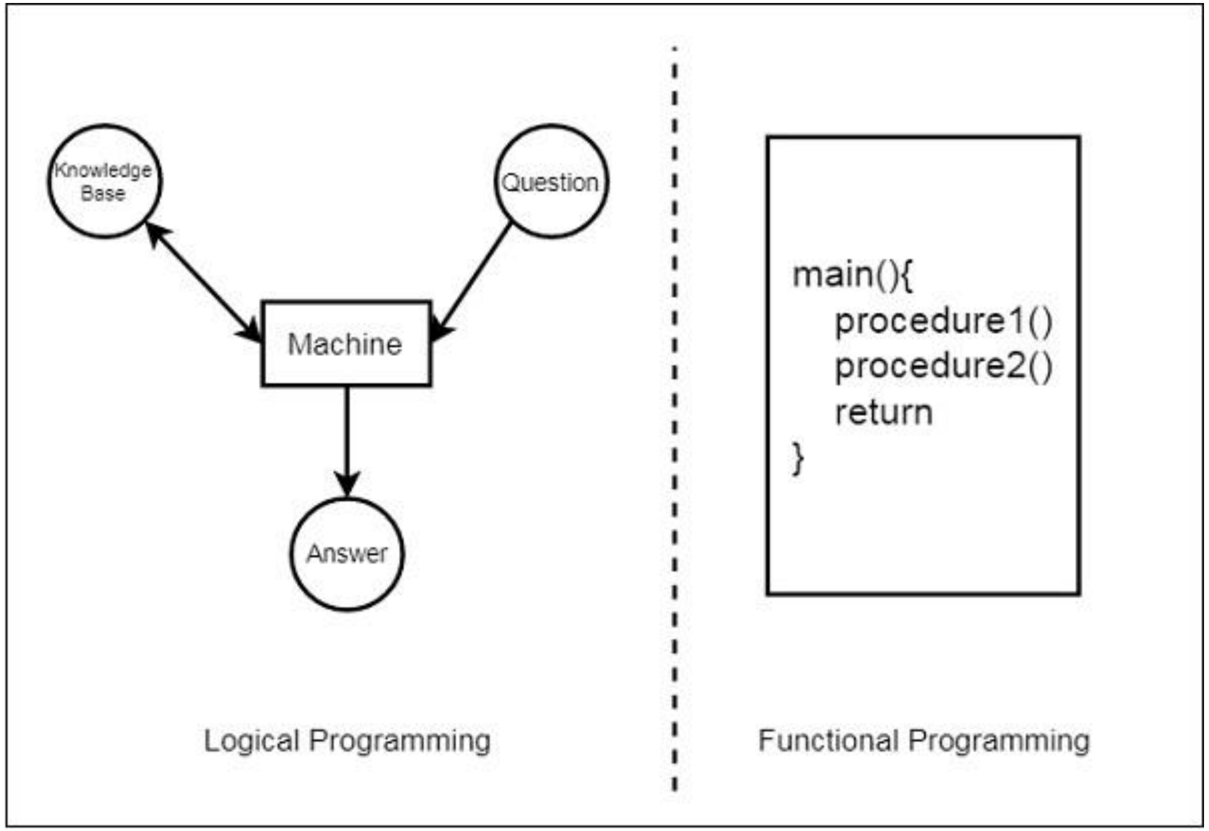
```
| ?- write('Hello World!'), nl.
Hello World!

yes
| ?-
```

Computer programming in Prolog consists of:

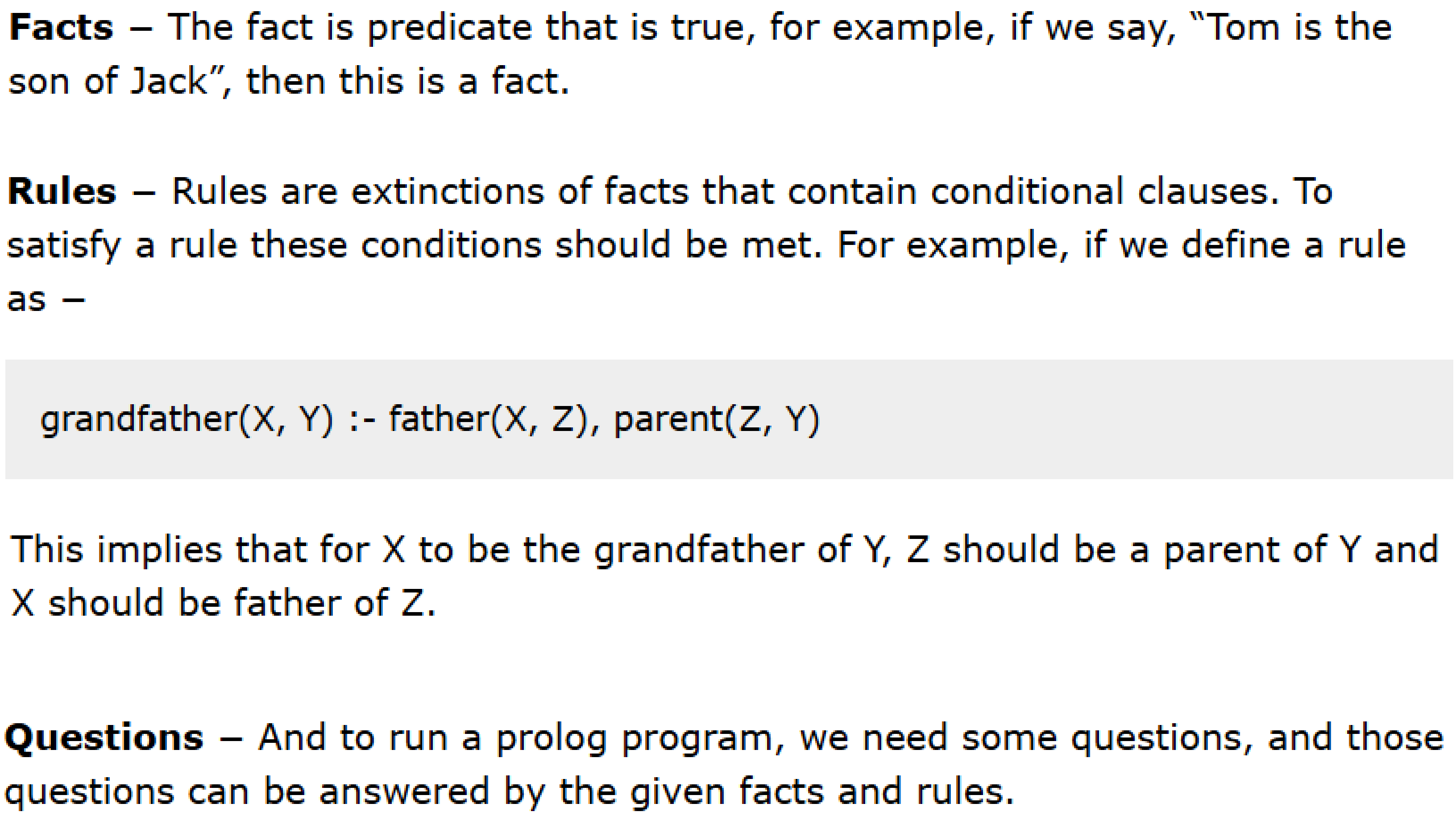
- specifying some *facts* about objects and their relationships,
- defining some *rules* about objects and their relationships, and
- asking *questions* about objects and their relationships.



Functional Programming	Logic Programming
Functional Programming follows the Von-Neumann Architecture, or uses the sequential steps.	Logic Programming uses abstract model, or deals with objects and their relationships.
The syntax is actually the sequence of statements like (a, s, I).	The syntax is basically the logic formulae (Horn Clauses).
The computation takes part by executing the statements sequentially.	It computes by deducting the clauses.
Logic and controls are mixed together.	Logics and controls can be separated.



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



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

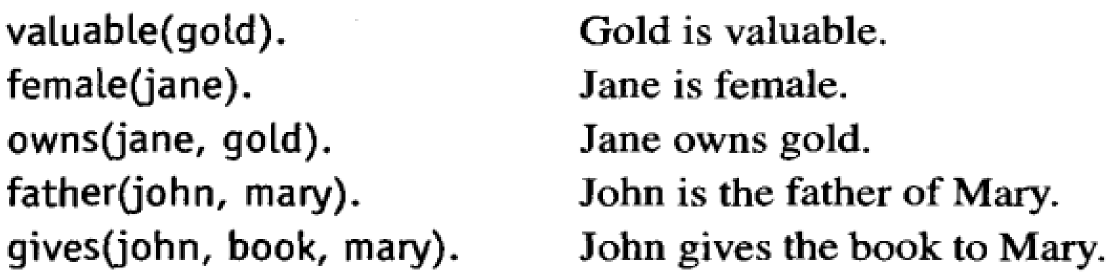
Rules – Rules are extinctions of facts that contain conditional clauses. To satisfy a rule these conditions should be met. For example, if we define a rule as –

```
grandfather(X, Y) :- father(X, Z), parent(Z, Y)
```

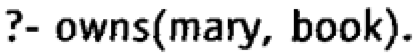
This implies that for X to be the grandfather of Y, Z should be a parent of Y and X should be father of Z.

Questions – And to run a prolog program, we need some questions, and those questions can be answered by the given facts and rules.

Example 1: Facts, Rules, Questions

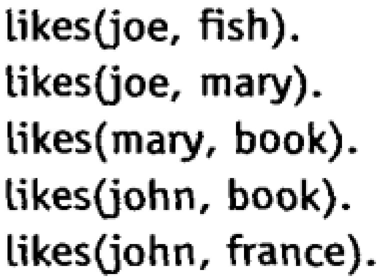


```
valuable(gold).             Gold is valuable.
female(jane).               Jane is female.
owns(jane, gold).           Jane owns gold.
father(john, mary).         John is the father of Mary.
gives(john, book, mary).    John gives the book to Mary.
```

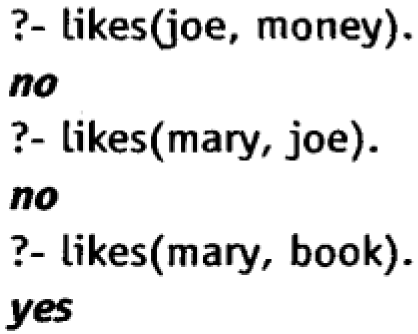


```
?- owns(mary, book).
```

Example 2: Facts, Rules, Questions

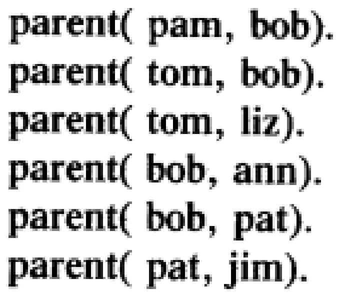


```
likes(joe, fish).
likes(joe, mary).
likes(mary, book).
likes(john, book).
likes(john, france).
```

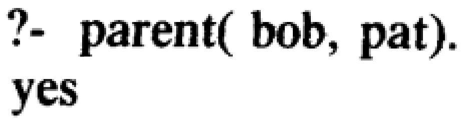


```
?- likes(joe, money).
no
?- likes(mary, joe).
no
?- likes(mary, book).
yes
```

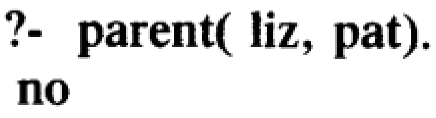
Example 3: Facts, Rules, Questions



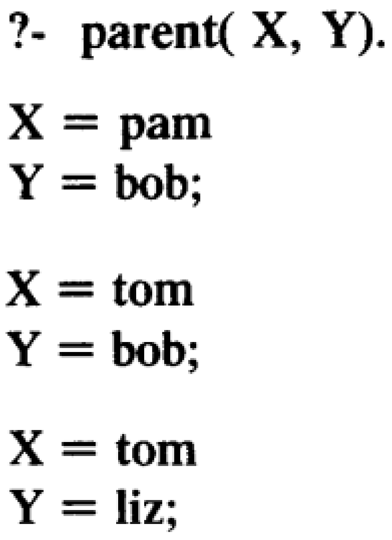
```
parent( pam, bob).
parent( tom, bob).
parent( tom, liz).
parent( bob, ann).
parent( bob, pat).
parent( pat, jim).
```



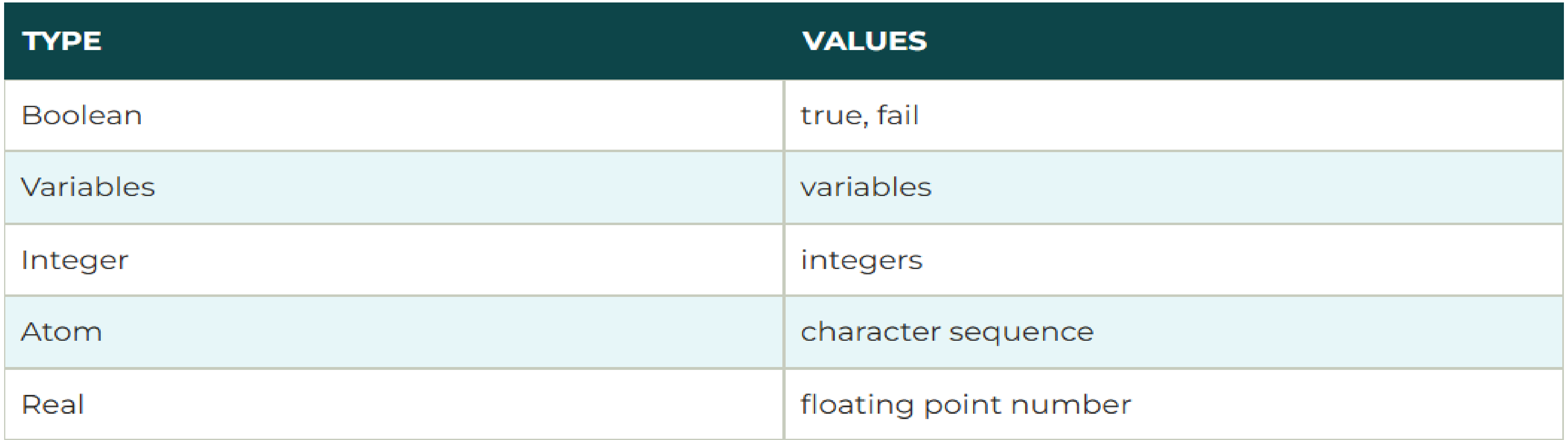
```
?-  parent( bob, pat).
yes
```



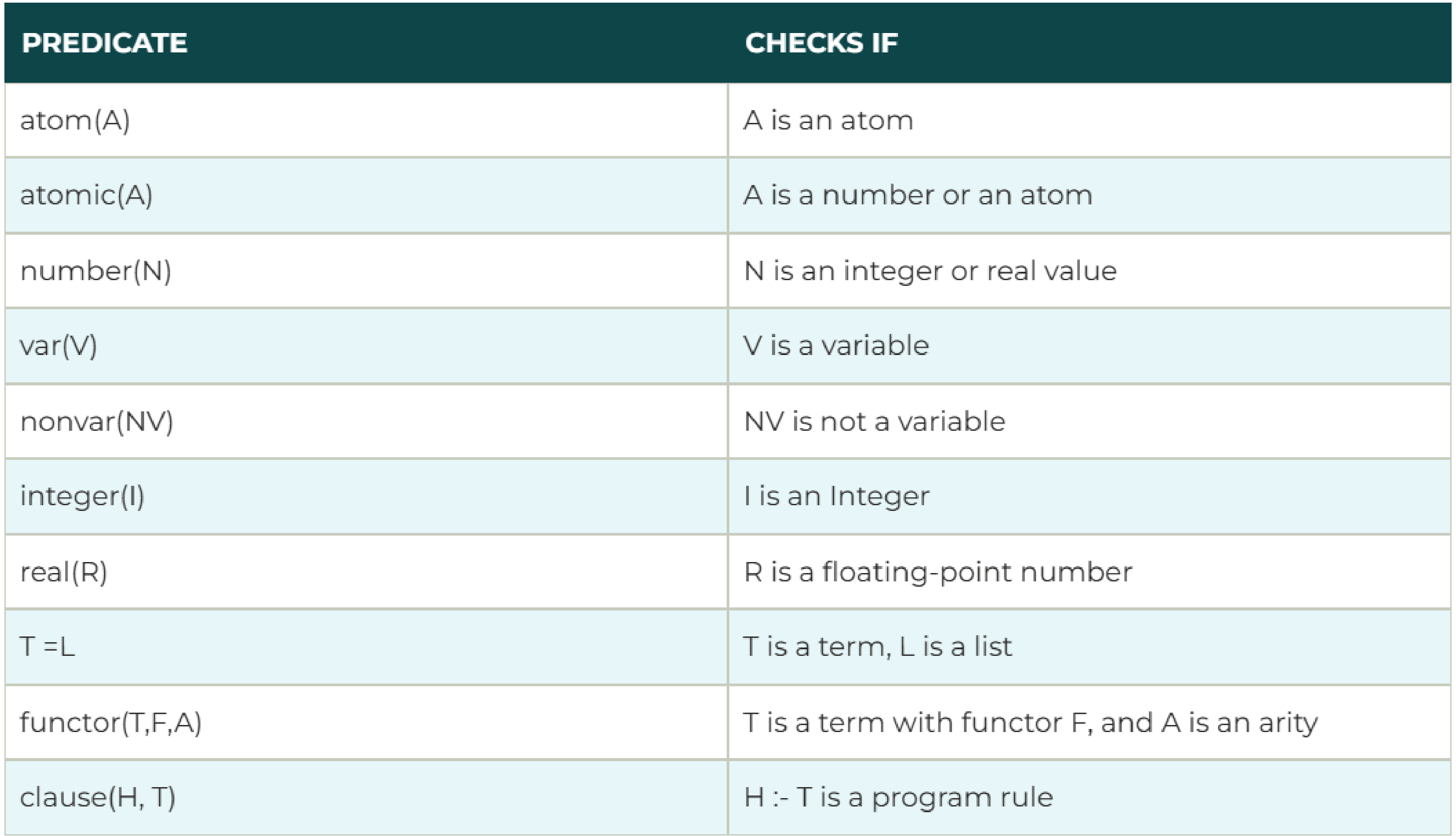
```
?-  parent( liz, pat).
no
```



```
?-  parent( X, Y).

X = pam
Y = bob;

X = tom
Y = bob;

X = tom
Y = liz;
```



TYPE	VALUES
Boolean	true, fail
Variables	variables
Integer	integers
Atom	character sequence
Real	floating point number



PREDICATE	CHECKS IF
atom(A)	A is an atom
atomic(A)	A is a number or an atom
number(N)	N is an integer or real value
var(V)	V is a variable
nonvar(NV)	NV is not a variable
integer(I)	I is an Integer
real(R)	R is a floating-point number
T =L	T is a term, L is a list
functor(T,F,A)	T is a term with functor F, and A is an arity
clause(H, T)	H :- T is a program rule

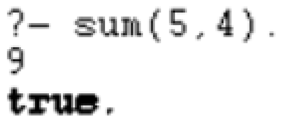
1. Write a prolog program to calculate the sum of two numbers.

SOLUTION

PROGRAM

```
sum(X,Y):-
   S is X+Y,
   write(S).
```

OUTPUT



```
?- sum(5,4).
9
true.
```

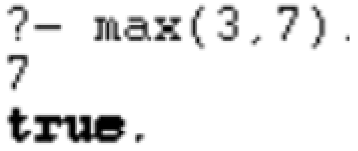
2. Write a prolog program to find the maximum of two numbers.

SOLUTION

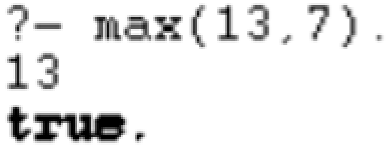
PROGRAM

```
max(X,Y):-
(
 X=Y ->
   write('both are equal')
 ;
 X>Y ->
  (
    Z is X,
    write(Z)
  )
  ;
  (
    Z is Y,
    write(Z)
  )
).
```

OUTPUT



```
?- max(3,7).
7
true.
```



```
?- max(13,7).
13
true.
```

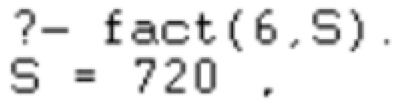
3. Write a prolog program to calculate the factorial of a given number.

SOLUTION

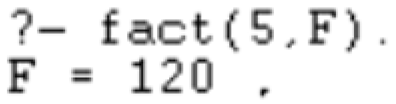
PROGRAM

```
fact(0,1).
fact(N,F):-
(

 % The below is for +ve factorial.
 N>0 ->
 (
  N1 is N-1,
  fact(N1,F1),
  F is N*F1
 )
 ).
```

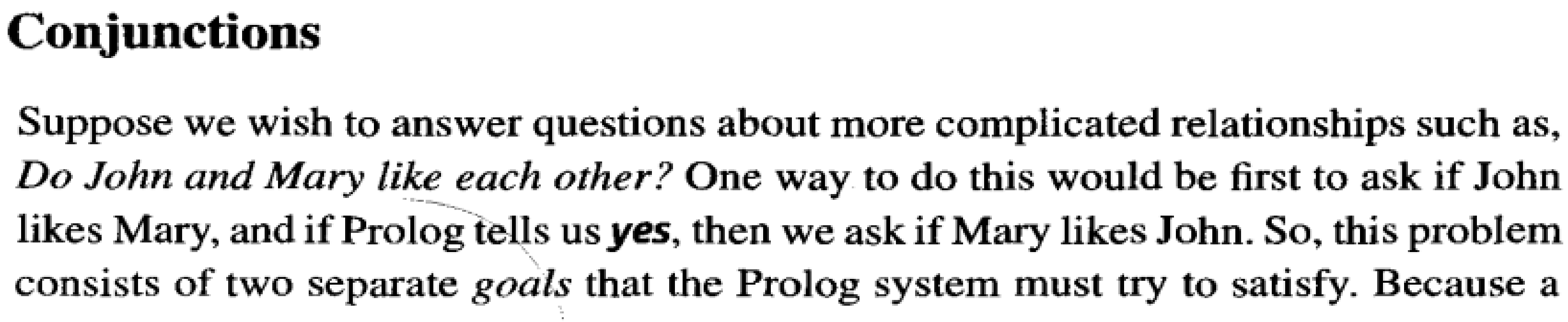
OUTPUT



```
?- fact(6,S).
S = 720 .
```

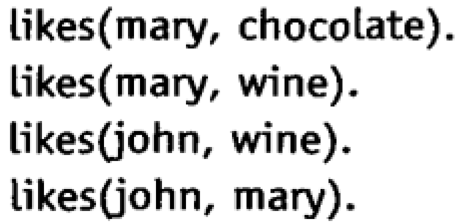


```
?- fact(5,F).
F = 120 .
```

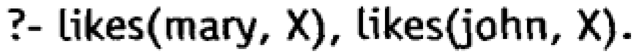


Conjunctions

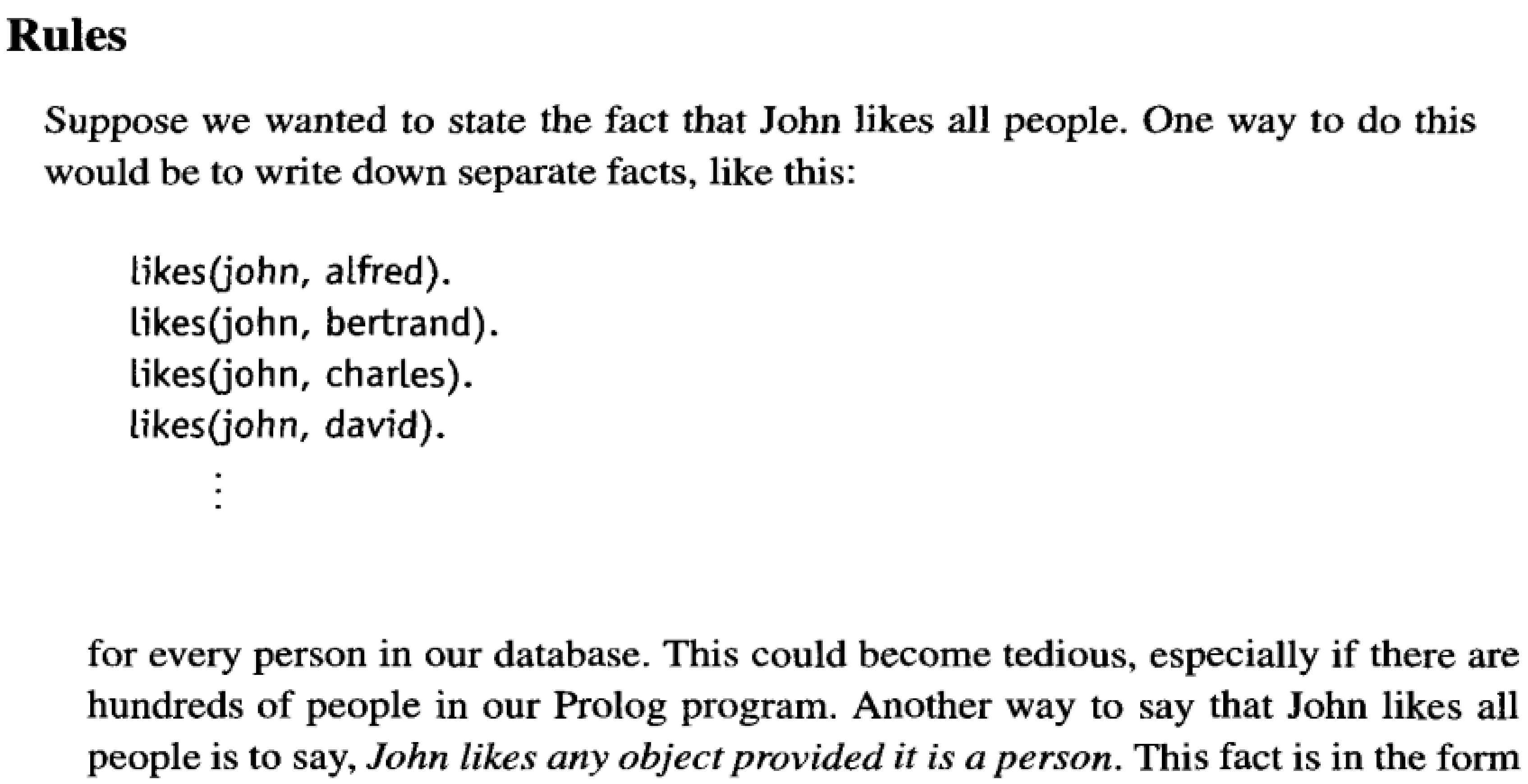
Suppose we wish to answer questions about more complicated relationships such as, *Do John and Mary like each other?* One way to do this would be first to ask if John likes Mary, and if Prolog tells us ***yes***, then we ask if Mary likes John. So, this problem consists of two separate *goals* that the Prolog system must try to satisfy. Because a



```
likes(mary, chocolate).
likes(mary, wine).
likes(john, wine).
likes(john, mary).
```



```
?- likes(mary, X), likes(john, X).
```



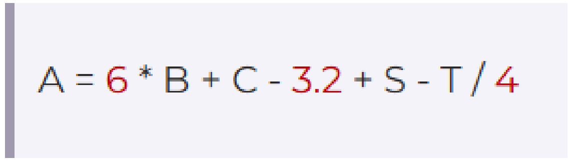
Rules

Suppose we wanted to state the fact that John likes all people. One way to do this would be to write down separate facts, like this:

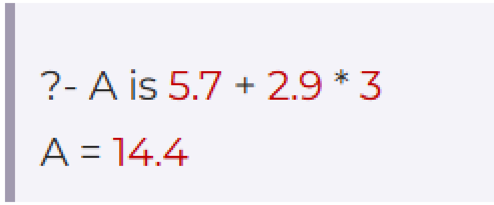
```
likes(john, alfred).
likes(john, bertrand).
likes(john, charles).
likes(john, david).
        ⋮
```

for every person in our database. This could become tedious, especially if there are hundreds of people in our Prolog program. Another way to say that John likes all people is to say, *John likes any object provided it is a person*. This fact is in the form

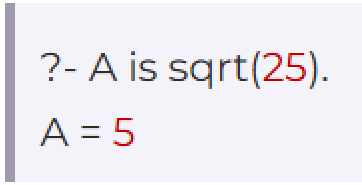
Arithmetic in Prolog



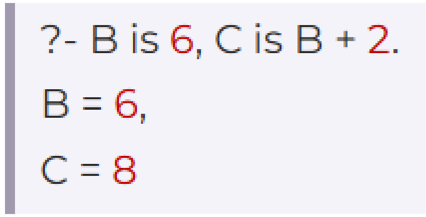
```
A = 6 * B + C - 3.2 + S - T / 4
```



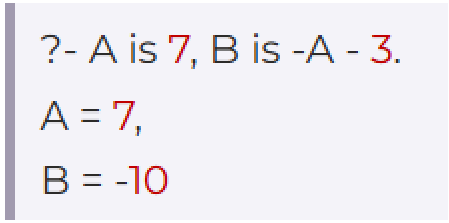
```
?- A is 5.7 + 2.9 * 3
A = 14.4
```



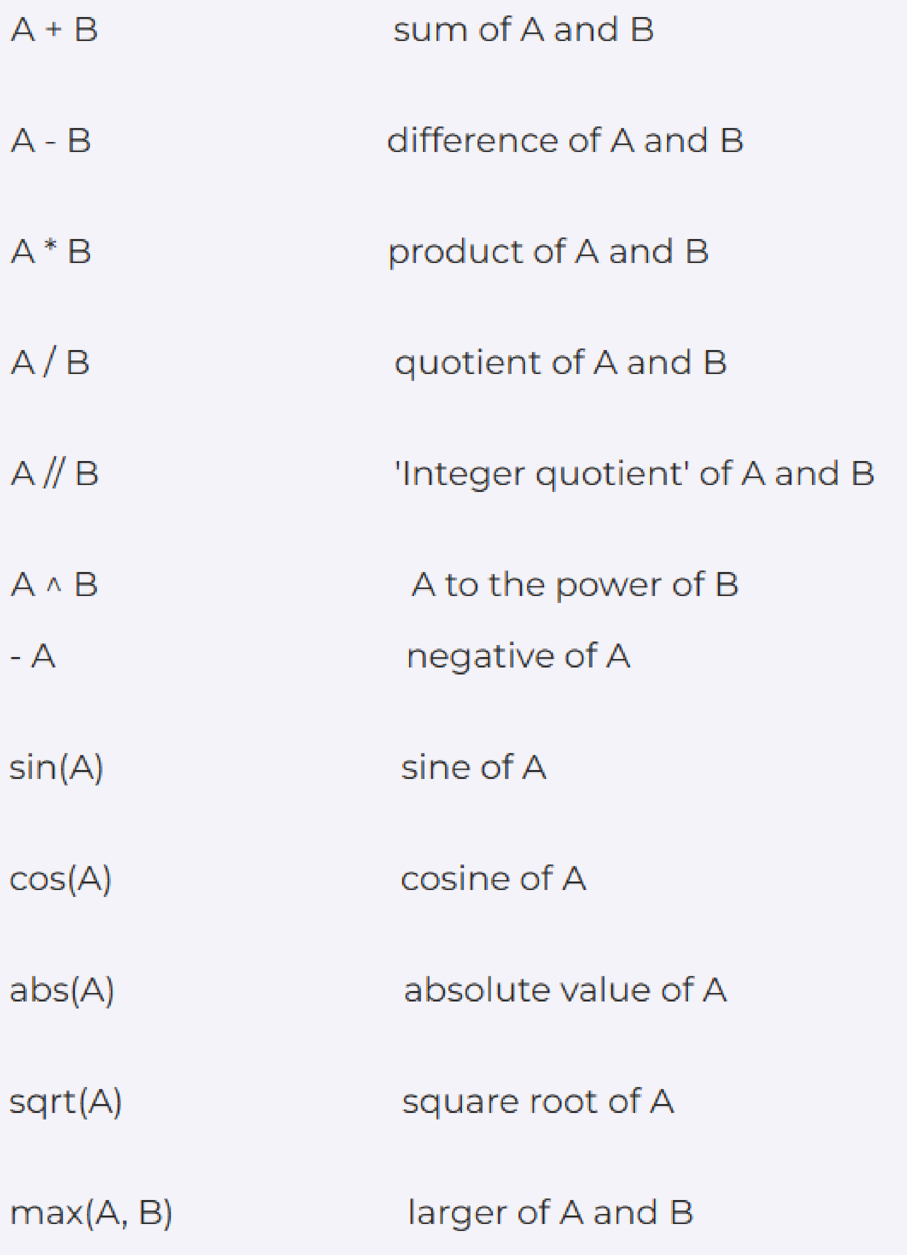
```
?- A is sqrt(25).
A = 5
```



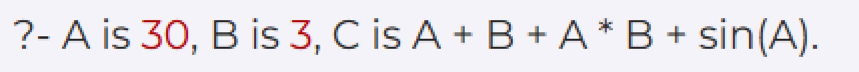
```
?- B is 6, C is B + 2.
B = 6,
C = 8
```



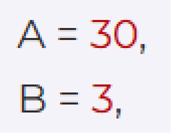
```
?- A is 7, B is -A - 3.
A = 7,
B = -10
```



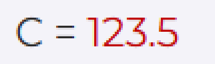
A + B	sum of A and B
A - B	difference of A and B
A * B	product of A and B
A / B	quotient of A and B
A // B	'Integer quotient' of A and B
A ∧ B	A to the power of B
- A	negative of A
sin(A)	sine of A
cos(A)	cosine of A
abs(A)	absolute value of A
sqrt(A)	square root of A
max(A, B)	larger of A and B



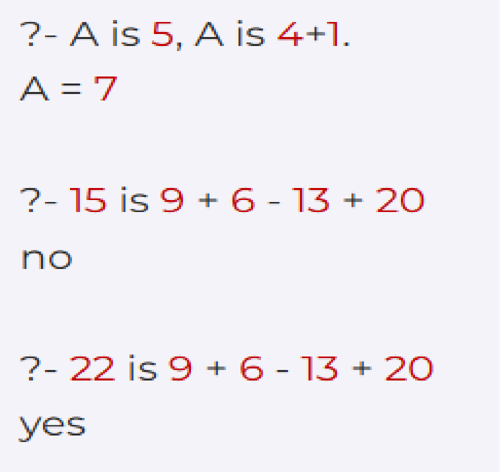
```
?- A is 30, B is 3, C is A + B + A * B + sin(A).
```



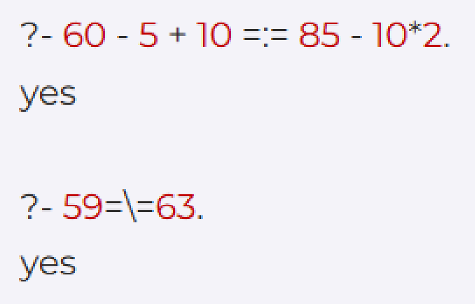
```
A = 30,
B = 3,
```



```
C = 123.5
```

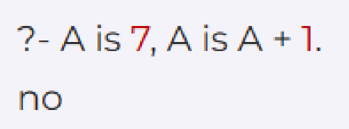


```
?- A is 5, A is 4+1.
A = 7

?- 15 is 9 + 6 - 13 + 20
no

?- 22 is 9 + 6 - 13 + 20
yes
```

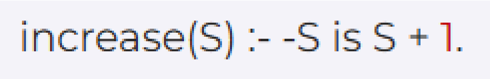


```
?- 60 - 5 + 10 =:= 85 - 10*2.
yes

?- 59=\=63.
yes
```

The goal A is A + 1 will always fail, whether or not A is bound.



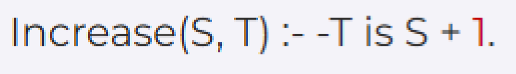
```
?- A is 7, A is A + 1.
no
```



```
increase(S) :- -S is S + 1.
```

?- increase(4).

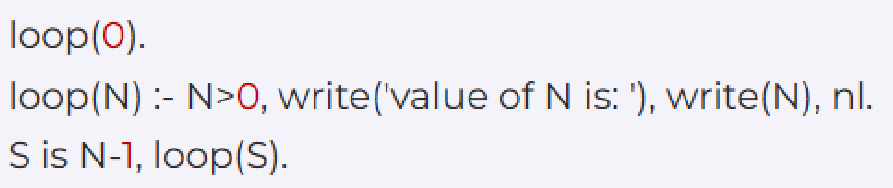
no



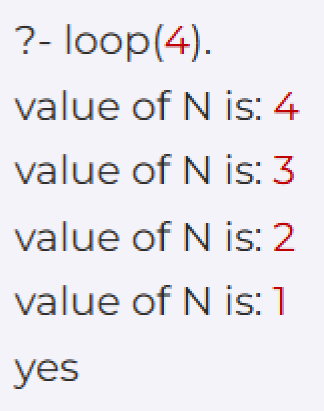
```
Increase(S, T) :- -T is S + 1.
```

?- increase(4, A).
A = 5

Loops in Prolog



```
loop(0).
loop(N) :- N>0, write('value of N is: '), write(N), nl.
S is N-1, loop(S).
```



```
?- loop(4).
value of N is: 4
value of N is: 3
value of N is: 2
value of N is: 1
yes
```