Atoms:
identifiers starting with lower case:
a, cisc260, catInHat
numbers: 31, -1.2
single-quoted strings: 'Mickey Mouse'

Variables:
names starting with upper case:
Fred, CatInHat

Some Simple Examples:
% facts
taken(george,cisc121,b).
taken(fred,cisc121,b).
taken(ron,cisc24,a).
teaches(lamb,cisc260).
teaches(mcleod,cisc121).

% rules
a_student(S) :- taken(S,_Course,a).
taught(Prof,Student) :-
teaches(Prof, Course), % comma = "and"
\begin{itemize}
\item taken(Student, Course, _).
\end{itemize}
good(S) :-
taken(S, _, a); % semicolon = "or"
taken(S, _, b).

Prolog queries using above facts & rules:
?- taken(S,cisc121,Mark).
S = george,
Mark = c ;
S = fred,
Mark = b.
?- taken(S,_,d).
false.
?- good(ron).
true .

?- taught(mcleod,fred).
true.

Facts & Queries Using Structures:
course(cisc204, prof(george,smith),
\begin{itemize}
\item range(time(14,30), time(15,20)),
\item location(nicol,321))
\end{itemize}
meetsIn(C,Bldg) :-
course(C,_,_,location(Bldg, _)).

Examples of Lists:
[1,2,3]
\begin{itemize}
\item [peter,susan,edmund,lucy]
\item [name(harry, potter), 42, [a, b]]
\end{itemize}
Comparisons:
1. = is structural equality, \= is structural inequality
   ?- Y=X+3.
      Y = X+3.
   ?- Y+2=3*5.
      false.
   ?- 2+2 \= 3+1.
      true.

2. :-= is arithmetic equality, =\= is arithmetic inequality
   both sides evaluated and compared. All variables must be bound
   ?- 5+7 =:= 3*4.
      true.
   ?- 9+2 =\= 12.
      true.

3. "is" means evaluate right side, assign to or compare with left side. Left side must be a single variable.
   ?- X is 4*2.
      X = 8.
   ?- X=9, X is 3*3.
      X = 9.
   ?- X=9, X is 2+6.
      false.

4. Inequality operators evaluate both sides
   ?- 2+3 < 6.
      true.
   ?- 3*4 > 13.
      false.
   ?- 1+2 >= 5.
      false.
   ?- 2+3 =< 6.
      true.

Arithmetic Operators:
+, -, *, /, //, mod
?- X is 17 / 3.
   X = 5.66666666666667.
?- X is 17 // 3.
   X = 5.
?- X is 17 mod 3.
   X = 2.

Example Using Arithmetic:
% count(Item, List, N) means Item occurs exactly N times in List.
count(_, [], 0).
count(Item, [X|Xs], Result) :-
   Item = X,
   count(Item, Xs, TailCount),
   Result is TailCount+1.
count(Item, [X|Xs], TailCount) :-
   Item \= X,
   count(Item, Xs, TailCount).

Cuts:
! this goal means Prolog can't backtrack over it – either to consider a different binding for a variable or to try a different rule.

An example using a cut for efficiency:
count(Item, [], 0).
count(Item, [Item|Tail], N) :-
   !,
   count(Item, Tail, TailCount),
   N is TailCount + 1.
count(Item, [Other|Tail], N) :-
   % the following check no longer needed
   % Item \= Other,
   count(Item, Tail, N).

Negation:
not(goal) means that there is no possible way to bind the variables in goal to make it true.

Examples using not:
?- not(member(d, [a,b,c])).
   true.
?- not(member(b, [a,b,c])).
   false.
?- not(member(X, [a,b,c])).
   false.
?- not(member(X, [])).
   true.

Finding all solutions:
?- bagof(X, member(X, [5,1,4,1]), Xs).
   Xs = [5, 1, 4, 1].
?- setof(X, member(X, [5,1,4,1]), Xs).
   Xs = [1, 4, 5].