facts - declarations
bigger(horse, elephant). - fact because stating something that is unconditionally true

rules -
- like if statement
- also called clause for the head
- can be rep as a tree (head's children are predicates in the body)
- head :- body is a rule stating that head is true if body true

Operators -
, - and operator
; - or operator
=:= - equal to operator
is - assignment operator, but can't reassign. If a variable is bound, error (?)
= - unification operator. if unbounded, assign, else compare lhs and rhs and return t/f
A = dog, A = lion. is false
A = dog. A = lion. means A is lion.

queries - is something true or false
bigger(horse, X). - find X (coz variable) st bigger(horse,X) is true - a query

knowledge base - what all is true - a collection of facts+rules

term - Starts with lowercase, enclosed in single quotes

atom - starts with capital/underscore
if underscore - anonymous variable eg - _ itself a variable.

compound term - atom with arguments - like a function in C - actually a predicate (number of args - arity)

everything ends in a dot

logical/rule based language

prolog uses interpreter
Prolog program - facts, then rules
fail - a predicate that is always false - kinda like an instruction to force backtracking in dfs

cut - ! - stop backtracking beyond a particular point

prolog has inbuilt search engine, uses dfs
common(lion,tiger,Z) :- eats(lion,Z), eats(tiger,Z)

above query uses dfs to find the first Z among the rules, then stops. to find all such values, we make it fail after one found, so it continues searching
i.e.
common(lion,tiger,Z) :- eats(lion,Z), eats(tiger,Z), fail

predicates in body of rule are connected left to right in corresponding tree, ie processed left to right
eventually stops because all predicates in body become false, so head becomes false so stop.
the problem here is that we can't write another rule after this rule which executes this rule as it fails
solution - common. after common(...) :- ....
this is second definition of common as a fact this time, so it's always true now. (actually all statements in sequence are OR (unless ,), that's why this happens)

recursion
order matters, recursion call should be at end

downcounting in english
if x is 0, stop the process
else decrement X and call process again

downcounting in prolog
countdown(0). % state that countdown predicate is true for 0 as fact
countdown(N) :- writeln(N), N1 is N-1, countdown(N1). %says countdown exists for all N, and is defined as a rule ie writeln.... N1 not N. Different N1 in each call to countdown because call stack

lisp - list processor - very powerful

lists in prolog
first term of list is head, rest elements form tail (so head is one element, tail is a list)
head and tail of empty list? - no head or tail defined for empty list
can contain heterogenous data types including list, compound terms, predicates, without definitions like python
getHead(X,[H|T]) :- X is H. -- a rule to get head of list
get_head(H,[H|_]). -- a fact to get head of list (problem - T is unused variable, so use anonymous variable)

member(X,[X|T]). -- recursive way to find if X is member of list or not
member(X, [H|T]) :- member(X, T)

?- query mode of IDE

dynamic predicates can be retracted (delete a pred) or asserted (add a pred) during runtime, so evolvable programs can be written in prolog. program can be changed at runtime

:-dynamic name/arity. - to define dynamic predicate

then assert(name(h)). adds name(h).
retract(name(h)). removes name(h).
asserta()- add at top of name facts
assertz() - at bottom
retract(name(_)) - delete all name facts
retractall(name(_)) - same thing (?)
retract() - delete first of the facts
retractall() - deletes all of the facts

AI winter
if somehow can combine powers of langs like C and those of prolog/lisp

Examples

carnivore(Tiger). - meaning - anything put in it is true/anything is a carnivore - semantically incorrect, otherwise correct
carnivore, carnivOre - different predicates
carnivore(lizard) - incorrect - no .
Bigger(Mosquito, horse). - incorrect B should be small
bigger(X,Y) - correct (semantically incorrect)
bigger(_,_). - still variables, correct (semantically incorrect) - anonymous variables don't use space

A is 5. % A is now bound to 5 and can't be reassigned anything else