Negation as Failure

1) animal(dog).
2) animal(cat).
3) animal(dove).
4) animal(python).
5) snake(python).

6) likes(mary, X) :- animal(X).

?- likes(mary, python).

Yes

Can we express that

mary likes all animals except snakes

6) likes(mary, X) :- animal(X), notsnake(X).

6a) notsnake(X) :- snake(X), !, fail.
6b) notsnake(X) :- true.

We can use ";" to indicate an alternative:
(shorthand for 2 rules with the same head)

notsnake(X) :- snake(X), !, fail; true.
Define "not P" to mean:

not P succeeds if P fails, and
not P fails if P succeeds.

not P :- P, !, fail.
not P :- true.

SWI-Prolog has "not" built-in as a prefix op.

Instead of
likes(mary, X) :- animal(X), notsnake(X).

We can write
likes(mary, X) :- animal(X), not snake(X).

and the rules for "notsnake"
are no longer necessary.
Goals with variables

1) odd(7).  3) integer(2).
2) prime(2).  4) integer(7).
5) even(X) :- not odd(X).

Goals containing variables can produce incorrect results with negation.

We ask for an integer that is even & prime

?- integer(X), even(X), prime(X).
   X = 2
   yes

Now ask in the other order:

?- even(X), prime(X), integer(X).
   no ( Why ? )

Solving "integer(X)" bound X to a constant, so "even(2)" was the goal invoking rule 5.

But invoking rule 5 with "even(X)" allowed a successful attempt to prove "odd(X)", leading to failure.
Computing with several solutions

1) animal(dog).
2) animal(cat).
3) four_legs(dog).
4) four_legs(cat).
5) animal(dove).
6) two_legs(dove).
7) animal(python).
8) snake(python).

9) likes(mary, X) :- animal(X), not snake(X).

?- likes(mary, X).
   X = dog;
   X = cat;
   X = dove;
   no

Can we get the list of animals mary likes?
?- setof(X, likes(mary, X), L).
   X = _0
   L = [dog, cat, dove].

How about the quadripeds mary likes?

10) like_4(X,Y) :- four_legs(Y), likes(X,Y).

?- setof(Y, like_4(X,Y), L).
   X = mary
   Y = _0
   L = [dog, cat];
   no