COMPSCI 367
Tutorial 8: More Prolog!
Jonathan Rubin.

1) Lists

List syntax in prolog: [ann, tennis, tom, skiing]

(1) the first item, called the *head* of the list;
(2) the remaining part of the list, called the *tail*.

E.g. *head* = ann
tail = [tennis, tom, skiing]

In general, the *head* of the list can be anything (any prolog object). The *tail* has to be a list.

It is often practical to treat the whole tail as a single object:

e.g:

```
        L = [a, b, c]
=>
        L = [a | Tail]
        Tail = [b, c]
```

We can also list any number of elements followed by '|' and the list of the remaining items:

e.g:

```
        L = [a, b, c]
=>
        L = [a, b | Tail]
        Tail = [c]
```

2) Membership

Lets implement,

member(X, L)

where, X is an object and L is a list. The goal member(X, L) is true if X occurs in L.
e.g:

member(b, [a, b, c]) *is true,*

member(b, [a, [b, c]]) *is not true,*

member([b,c], [a, [b, c]]) *is true.*

So,

X is a member of L if either:
(1) X is the head of L, or
(2) X is a member of the tail of L.

This can be written in two clauses; the first is a simple fact and the second is a rule:

```
member(X, [X | Tail]).

member(X, [Head | Tail]) :-
        member(X, Tail).
```

3. Concatenation

For concatenating lists we will define the relation:

```
conc(L1, L2, L3)
```

where, L1 and L2 are two lists and L3 is their concatenation
e.g:

conc([a, b], [c, d], [a, b, c, d])   *is true*,

conc([a, b], [c, d], [a, b, a, c, d])   *is false.*

Again we have two cases in the definition of conc:

```
conc([], L, L).

conc([X | L1], L2, [X | L3]) :-
        conc(L1, L2, L3).
```

We can use this for concatenating lists:

```
?- conc([a, b, c], [1, 2 3], L).
L = [a, b, c, 1, 2, 3]
```

We can also use conc in the inverse direction for *decomposing* lists:

```
?- conc(L1, L2, [a,b,c]).
L1 = []
L2 = [a,b,c];

L1 = [a]
L2 = [b,c];

L1 = [a,b]
L2 = [c]

…
```

Or, look for patterns:

E.g: find all the names before sam:

```
?- conc(L1, [sam | _], [bob, rob, sam, pam]).
L1 = [bob, rob].
```

4. Arithmetic

+, -, *, /

**	power
//	integer division
mod	modulo

Can use the *is,* built-in procedure. Forces an expression to be evaluated.

```
?- X is 1 + 2.
```

```
X = 3
```

Requires variables to be instantiated before use.

```
?- X is 1 + A.
```

Comparison Operators:

>, <, >=

=<	Less than or equal to
=:=	Equal
=\=	Not Equal

5. Debugging

?- trace.   Information regarding a goals satisfaction is displayed during execution.

?- notrace.   Stop *tracing*.

?- spy(P).   Trace only for a specified predicate, P.

?- nospy(P).   Stop *tracing* predicate P.

6. Not \+

```
?- not(Goal)
```

if the Goal succeeds then not(Goal) fails,
otherwise not(Goal) succeeds.

Alternatively, written as: \+ Goal.

E.g.

```
        likes(mary, X) :-
                animal(X),
                \+ snake(X).
```

Have to be careful as not doesn't directly correspond to negation in mathematical logic.