```
1?-[myprogram.P]
```
loads program in

To launch prolog at dos promopt
```
     > xsb
     1?- halt      //exits prolog
     ctrl-d        // exits prolog
```

Simple prolog program example

```
/* this is a prolog comment */
/* prolog has 2 o-ary predicates true/fail        */
bird(ostrich).            /* all lower case letters means a constant       */
bird(penguin).            /* variables begin with uppercase letters        */
bird(seagull).
bird(eagle).
flies(W) :- bird(W),    /* flies W if W is a bird, but not an ostrich or penguin
            W \= ostrich              /*       /= is not equals in prolog        */
            W \= penguin
loves(jane, X) :- flies(X).     /* jane loves things that fly    */
loves(penguin, jane).           /* penguin loves jane  */
loves(aadvark, jane).           /* aadvark loves jane  */
      /* r1 :- c1, … cn                       /* this is a rule */
      /* f1 :- …                              /* this is a fact/clause */
      /* bird, loves, flies are called predicates
      /* number of slots predicate has called arity.  Often when describing a predicate add arity after
         name
      /* bird /1   means bird has parity 1 or bird(x)
      /* bird /2   means bird has parity 2 or bird(x,y)
      /* r1 :- c1, c…, cn    the r1 is the head of the rule, c1, c2, c3, … is the tail of the rule
1?- bird(seagull).
yes.
1?- bird(duck).
no.
1?- bird(X), loves(X, jane).
X = penguin    /* if you put a semicolon at the end of this line and hit return, compiler looks for more
               solutions */
no.
1?- loves(X, Y).
X = jane
Y = seagull;
X = jane
Y = eagle;
X = penguin
Y = jane;
X = aadvark
Y = jane;
no.
```

Lists in prolog

Looks different than scheme, but roughly same idea

[ ] = empty list
[a, b, c] = commas between items like C
[dogs, cats, marbles, mix]
[root, [11, 12], [13]] = list of lists

code to append two lists

```
append([ ], L, L).        empty list appended with list L gives just list L
append( [X | L1], L2, [X | L3] )        :- append(L1, L2, L3)
                    X denotes first element of list, | denotes rest of list
```

```
1?- append([a, b], [c], Z).
[a, b, c]
```

```
/*      this tries to match the 2nd rule, X = a, L1 = [b], L2 = [c]
                tries to compute append([b], [c], L3)
                matches 2nd rule
                X = b
                L1 = [ ]
                L2 = [c]
                        tries to compute append([ ], [c], L3)
                        matches 1st rule
                        L = [c]
```