Prolog Quick Reference

INSTALL SWI-PROLOG ON UBUNTU LINUX

```
  % sudo apt-add-repository ppa:swi-prolog/stable
  % sudo apt-get update
  % sudo apt-get install swi-prolog
```

STARTING SWI-PROLOG ON LINUX

```
  kbarroga@OSDLOCKBOX002A ~
  $ swipl
  Welcome to SWI-Prolog (Multi-threaded, 32 bits,
  Version 6.6.6)
  ...
```

YOU CAN EXIT PROLOG WITH <CTRL>+<D> OR WITH THE HALT COMMAND

```
  ?- halt.
```

CLEAR SCREEN

```
  <CTRL>+<L>
```

INTERACTIVE SHELL THAT YOU CAN USE DIRECTLY BY ASSERTING FACTS AND ASKING QUERIES

```
  ?- Y is 10, X is Y + 3.

  Y = 10
  X = 13
```

SIMPLE PROLOG PROGRAM: likes.pl

```
  likes(kevin,food).
  likes(kevin,beer).
  likes(mandie,wine).
  likes(mandie,kevin).
```

LOADING PROGRAMS

*Make sure you are in the correct directory

```
  kbarroga@OSDLOCKBOX002A /home/apes-0.2.0/src
  $ swipl

  ?- [likes].
  ...
  % main compiled 0.08 sec, 275 clauses
  true.

  ?- ['likes.pl'].
  ...
  % main.pl compiled 0.00 sec, 1 clauses
  true.

  ?- consult('likes.pl').
  ...
  % main.pl compiled 0.00 sec, 1 clauses
  true.
```

EXECUTING A QUERY

```
  ?- likes(kevin,beer).
  true.

  ?- likes(kevin,wine).
  false.

  ?- likes(kevin,X).
  X = food ;
  X = beer.

  ?- likes(mandie,X).
  X = wine ;
  X = kevin.

  ?- likes(mandie,beer).
  false.

  ?- likes(mandie,wine).
  true .
```

THE HELP FUNCTION CAN BE USED TO LOOK UP COMMANDS IN THE MANUAL, FOR EXAMPLE

```
  help(consult).
```

THE LISTING COMMAND CAN BE USED TO LIST THE PREDICATES SPECIFIED IN THE PROGRAM

```
  ?- listing(likes).
  likes(kevin, food).
  likes(kevin, beer).
  likes(mandie, wine).
  likes(mandie, kevin).

  true.

  ?- listing(consult).
  :- meta_predicate consult(:).

  system:consult(D:A) :-
      A==user, !,
      flag('$user_consult', B, B+1),
      C is B+1,
      atom_concat('user://', C, E),
      load_files(D:E, [stream(user_input)]).
  system:consult(A) :-
      load_files(A, [expand(true)]).

  true.
```

TO GET HELP DURING A RUN, ENTER H YOU WILL GET A HELP LIST SHOWING THE COMMANDS AVAILABLE TO YOU

```
  ?- likes(X,Y).
  X = kevin,
  Y = food

  Actions:
  ; (n, r, space, TAB): redo t: trace & redo
  b: break c (a, RET): exit
  w: write p print
  h (?): help

  Action?
```

USEFUL COMMANDS ARE DEBUG AND TRACE. IT ALLOWS YOU TO SPY ON THE INTERPRETER AS IT ATTEMPTS TO SATISFY GOALS

```
  ?- trace.
  true.

  [trace] ?- likes(kevin,wine).
    Call: (6) likes(kevin, wine) ? creep
    Fail: (6) likes(kevin, wine) ? creep
  false.

  [trace] ?- likes(kevin,beer).
    Call: (6) likes(kevin, beer) ? creep
    Exit: (6) likes(kevin, beer) ? creep
  true.

  [trace] ?- likes(X,Y).
    Call: (6) likes(_G1174, _G1175) ? creep
    Exit: (6) likes(kevin, food) ? creep
  X = kevin,
  Y = food ;
    Redo: (6) likes(_G1174, _G1175) ? creep
    Exit: (6) likes(kevin, beer) ? creep
  X = kevin,
  Y = beer ;
    Redo: (6) likes(_G1174, _G1175) ? creep
    Exit: (6) likes(mandie, wine) ? creep
  X = mandie,
  Y = wine ;
    Redo: (6) likes(_G1174, _G1175) ? creep
    Exit: (6) likes(mandie, kevin) ? creep
  X = mandie,
  Y = kevin.

  [trace] ?-
```

TURN TRACING OFF

```
  ?- notrace.
```

PROGRAM SYNTAX

Prolog program syntax consists of predicates and clauses. A predicate consists of one or more clauses. A clause is a base clause if it is unconditionally true, if it has no "if" part

```
 <program> ::= <predicate> | <program><predicate>

 <predicate> ::= <clause> | <predicate><clause>

 <clause> ::= <base clause> | <nonbase clause>
```

Prolog Quick Reference

Two clauses belong to the same predicate if they have the same name(functor)and the same arity.

likes(kevin) and likes(kevin,beer) are different predicates.

```
  <base clause> ::= <structure> .

  <nonbase clause> ::= <structure> :-
                       <structures> .

  <structures> ::= <structure> | <structure> ,
                                 <structures>
```

A structure is a functor followed by zero or more arguments; the arguments are enclosed in parentheses and separated by commas. If there are no arguments, the parentheses are omitted.

```
  <structure> ::= <name> | <name> ( <arguments> )

  <arguments> ::= <argument> | <argument> ,
                               <arguments>
```

Arguments may be any legal Prolog values or variables. A variable is written as a sequence of letters and digits, beginning with a capital letter. The _ underscore is considered to be a capital letter.

An atom is any sequence of letters and digits, beginning with a lowercase letter. Alternatively, an atom is any sequence of characters, enclosed by single quotes ('atom')

Examples:

```
beer, dawg11, max_value, maxValue, 'max value'
```

Inline comment begin with the %, percent sign

Block comments begin with the characters /* and end with */

UNIFICATION AND INSTANTIATION

Unification and instantiation can be performed explicitly with the '=' operator, or implicitly via parameter transmission. Unification is a symmetric operation (X=Y is the same as Y=X), and is not the same as assignment.

Any value can be unified with itself.

Example:

```
likes(beer) = likes(beer).
```

A variable can be unified with another variable. The two variable names thereafter reference the same variable.

Example:

```
X = Y, X = 2, write(Y). % Writes the value 2.
```

A variable can be unified with any Prolog value; this is called instantiating the variable. A variable is fully instantiated if it is unified with a value that does not itself contain variables.

Example:

```
X = foo(bar, [1, 2, 3]). % X is fully
instantiated.
```

Example:

```
Pa = husband(Ma). % Pa is partially instantiated.
```

Two different values can be unified if there are unifications for the constituent variables which make the values the same.

Example:

```
mother(mary, X) = mother(Y, father(Z)).
[Also results in the unifications mary=Y and
X=father(Z)].
```

It is legal to unify a variable with an expression containing itself; however, the resultant value cannot be printed, and must otherwise be handled with extreme care.

Example:

```
X = foo(X, Y).
```

BUILT-IN PREDICATES

CONTROL PREDICATES

not X

  (Sometimes written \+X or not(X)) Succeed only when X fails.

true

  Succeed once, but fail when backtracked into.

repeat

  Always succeed, even when backtracked into.

fail

  Never succeed.

!

  "cut". Acts like true, but cannot be backtracked past, and prevents any other clauses of the predicate it occurs in from being tried.

abort

  Return immediately to the top-level Prolog prompt.

ARITHMETIC PREDICATES

X is E

  Evaluate E and unify the result with X.

X + Y

  When evaluated, yields the sum of X and Y.

X - Y

  When evaluated, yields the difference of X and Y.

X * Y

  When evaluated, yields the product of X and Y.

X / Y

  When evaluated, yields the quotient of X and Y.

X mod Y

  When evaluated, yields the remainder of X divided by Y.

X =:= Y

  Evaluate X and Y and compare them for equality.

X =\= Y

  Evaluate X and Y and succeed if they are not equal.

X < Y

  Evaluate X and Y and succeed if X is less than Y.

X =< Y

  Evaluate X and Y and succeed if X is less than or equal to Y.

X > Y

  Evaluate X and Y and succeed if X is greater than Y.

X >= Y

  Evaluate X and Y and succeed if X is greater than or equal to Y.

Sources:
http://www.cse.unsw.edu.au/~billw/prologdict.html
http://www0.cs.ucl.ac.uk/staff/mahmed/teaching/intro.html
http://www.swi-prolog.org/build/Debian.html
http://www.cis.upenn.edu/~matuszek/Concise%20Guides/Concise%20Prolog.html