```
Prolog : Logical Languages
    Logical predicates and deduction
    Recall Discrete math

    works on closed world view
        any fact that you tell it is true
        anything else is false
        be careful:
            blue(car)
            not_blue(car)
            compiler thinks this makes sense

Facts : Truths
    sunny
    sky_is_blue


2 parts to a program  : Database and Queries


Database :
    some kind of text file
    name.pl (.pro .P)


Query :
    sunny
        yes
    warm
        uncaught exception, identity not found...
            used to ouput no

Facts with arguments :
    warm(la).
        ie: it is warm in LA
    warm(chicago).
        ie: is warm in Chicago

    fly(chicago, la).
        trying to establish a relationship between chicago and la
        can fly from chicago to la
        no enforced way of reading this (origin destination, destination origin)
        no specified way of reading or writing relations
            you have to define it yourself

    likes(a, b).
```

```
        have not enforced :
        a->b
        b->a
        a<->b
        NEED TO WRITE COMMENTS ABOUT WHAT YOU MEAN BY THIS RELATION
        compiler just establishes relation between the two

Capital letter : Variable
    A
    likes(X,b).   (how can I fill in the variable X so that this relataionship is true)
        prolog says, this is possible if X = a
        if many:
            esc : exit query
            a : show all the results
            ; : show the next result
        KEEP TRACK OF ORDER

        likes(b,X)
            no (can't prove this statement)


Clauses : in databse
    prove subfacts to prove this fact
    fun(x) :-
        blue(x),
        car(x).

        combined with logical ands

    to do an or: write a new clause
    fun(x) :-
        red(x),
        candy(x).

    something is fun if it is (blue and a car)
    or (red and candy)


    clauses connected with ors
    subclauses connted with ands


    fun(truck).

    must also have
    blue(truck),
```

```
    car(truck).

    if you don't have this : probably an error
        make sure you don't say a truck is an apple



Unification : similar to matching in OCaml


List
[1, 3, 4]
    [ H | T ]
    H: 1
    T: [3,4]

[] won't unify
    tail can unify with empty
    but head can not

can use concrete things to unify
    [1 | T]


don't really have return values, so have to have argument
to "store" the result


append([],List,List)
append([Head|Tail], List2, [Head|Result]) :-
    append(Tail, List2, Result).   (the recursive call)

    the Heads are same (variable names local to clause they're written in)


    if this subclause is true, then this superclause is true

append(L1, L2, [1,2,3])
    []          [1,2,3]
    [1]         [2,3]
    [1,2]       [3]
    [1,2,3]     []


if result is a permutation of L1 what properties do I know are true
(takes L1 and returns some permutation of L1)
```

```
what do I know is true :
    first element in permutation has to be some element in our original list

will return true if [H|T] is permutation of L


perm([],[])
perm(L, [H|T]) :-
    append(V, [H|U], L),
    append(V,U,W),
    perm(W,T).

this is how you prove something is a permutation of something else



negation:

not_blue(x) :-
    \+(blue(x))

    \+ : negation of this


comparisons (all true)
    1 = 1
    2 < 3
    3 > 2
    1 \= 3

    L= [ H | T ]
        ( can use _ | _ if you don't care)

arithmetic assignment to variables
    X is 3 + 1


len([],0).
len([H|T], N) :-
    X is N-1,    // because N-1 is not naturally evaluated as an arithmentic expression
    len(T, X).



len(L1,5)
    [_,_,_,_,_,_];
```

```
    if you want more, it will then stack overflow


loading a database into interpreter

consult('db.pl')



tower(5,
          [[2,3,4,5,1],
           [5,4,1,3,2],
           [4,1,5,2,3],
           [1,2,3,4,5],
           [3,5,2,1,4]],[
    [2,3,2,1,4],
            [3,1,3,3,2],
            [4,1,2,5,2],
            [2,4,2,1,2]]).


tower(5,T,[
    [2,3,2,1,4],
            [3,1,3,3,2],
            [4,1,2,5,2],
            [2,4,2,1,2]]).



plain_tower(5, T,
          counts([2,3,2,1,4],
                 [3,1,3,3,2],
                 [4,1,2,5,2],
                 [2,4,2,1,2])).
```