Prolog notes

Prolog is based on FOL predicate calculus
Works on Horn clauses: disjunct literals with at most one positive literal

$\sim a \vee \sim b \vee \sim c \vee d$
can be turned into:
$(a \wedge b \wedge c) \Rightarrow d$

Prolog defines facts & rules which are then queried

Query types are what and T/F:

```
?- transport(windsor, X).
?- transport(windsor, toronto).
```

Variables start with uppercase.
Atoms + predicates are lowercase.

Fact defined as:

```
transport(toronto, windsor).
transport(toronto, X).
```
← can have variables

```
transport(windsor, ottawa) :- transport(windsor, toronto),
                              transport(toronto, ottawa).
```
$\uparrow$
if

Rules can have variables.

```
transport(windsor, X) :- transport(toronto, X).
```

Unification is essentially matching.
Prolog can:
1.) Unify atoms if they are identical
2.) Unify variables to anything.
3.) Two "function" will unify if they have the same name, number of args, and their atoms and variables follow above rules.

Backtracking is essentially DFS of possible unifications to try and satisfy some predicate.

Recursion:
Base case
Recursive case

the quick brown fox jumps over the lazy dog

```
on_route(rome).  ← base case
on_route(P):-    ← recursive case
    move(P, M, NP),
    on_route(NP).

move(home, taxi, halifax).
move(halifax, train, gatwick).
move(gatwick, plane, rome).
```

List processing

List can be anything.

1.) (a, b, c)
2.) ([a, b], c)

List bar separates last element.
(this is a sublist)

[a, b, c] is equivalent to [a|[b, c]]

Head/Tail:

```
[1, 2, 3, 4] = [H, T]
H= 1
T= [2, 3, 4]
```

Membership check:

```
member(X, [X|_]).
```
This means true if X is at the head of the list.
Underscore means rest of the list doesn't matter.

```
member(X, [_|T]):- member(X, T).
```
If first condition fails, this will check whether X is a member of the rest of the list.

= - unifies

Consult(filename) - loads program

trace, notrace - trace

>, <, =<, >= - arithmetic operators

is - evaluates RHS and unifies with LHS

3 is 2+1. ✓ [compares 3 with 2+1]

3 = 2+1. ✗ [compares 3 with '2+1']

Write to a file:

open('filename', write, OutputStream), write(X, OutputStream), close(OutputStream).

X==Y - checks to see whether X and Y are bound to the same value

not(Q) or \+ - negation

beep(boop).

?- \+ beep(boop). false