# Prolog - reminders, p. 1

[source: Scott, Ch. 11, pp. 547-548]

- "...a Prolog interpreter runs in the context of a **database** of **clauses** (**Horn clauses**) that are assumed to be true."
  - sometimes this is called its **knowledge base**
- **Terms** may be **constants**, **variables**, or **structures**
- A **constant** is an **atom** or a **number**
  - `foo my_Const + 'Hi' -37.5`
- A **variable** begins with an uppercase letter
  - `Foo  My_var  X`
- A **structure** can be considered as either a **logical predicate** or a **data structure**

# Prolog - reminders, p. 2

[source: Scott, Ch. 11, pp. 547-548]

- "Structures consist of an atom called the **functor** and a list of arguments: [which can be any terms -- constants, variables, structures, nested structures, etc.!]

```
rainy(arcata)

teaches(tuttle, cs335)

bin_tree(foo, bin_tree(bar,arc))
```

- We use the term "**predicate**" to refer to the combination of a **functor** and an "**arity**" (number of arguments).
  - The predicate `rainy` has arity 1.
  - The predicate `teaches` has arity 2."

# Prolog - reminders, p. 3

[source: Scott, Ch. 11, pp. 547-548]

- A Prolog database contains **facts** and **rules**;
- A **fact** is a Horn clause with no RHS:

```
rainy(arcata).
```

- A **rule** has a RHS: (read a comma as "and")

```
snowy(X) :- rainy(X), cold(X).
```

- Variables that appear in the head of a Horn clause are **universally** quantified:
  - **for all X**, X is snowy if X is rainy and X is cold.
- A **query** or **goal**, a clause with an empty LHS, does not go in a knowledge base, but is given to the Prolog interpreter or compiled program to try to prove.

# Prolog - Boolean operators

[source: no-longer-available tutorial: http://www.cse.msu.edu/~cse440/Programming1/programming1tut.html]

- use a **comma** `[,]` for boolean AND
- use a **semicolon** `[;]` for boolean OR
- use **backslash** and **plus** `[\+]` for boolean NOT
- (and parentheses for grouping ARE permitted)

# = and == in Prolog

- In a Prolog rule, = means "unified with"
  - we see that in swipl's responses to our queries;

```
?- likes(A, pie).
A = eve ;  /* corrected after class */
false.
```

- If you'd like to ask -- say, in a rule -- whether two variables happen to be unified to the same value, you can use == for that:

```
?- likes(A, eve), A == al. /* corrected*/
A = al ;
false
/* DIDN'T unify A with eve */
```