**Unification.** In Prolog we compute using expressions. Variables represent the portion of an expression that hasn’t been bound to a value yet. Unification is the process where two expressions are matched by trying to fill in the unbound portions of one expression with the corresponding partially bound portion of another. A unification fails if the corresponding portions of two expressions are bound to different constants.

```prolog
unify_example1(A, B) :- A = B.
?- unify_example1(X, f(Y)).
X = f(Y)
?- f(g(h(i)), k(X)) = f(g(X), k(j)).
no
?- f(g(h(i)), k(X)) = f(g(Y), k(j)).
X = j
Y = h(i)
unify_example2(A, B) :- unify_with_occurs_check(A, B).
?- unify_example2(X, f(X)).
no
```

**Lists are represented in square bracket notation. However, this notation is really short hand for a nesting of the functor.**

```prolog
?- (a, (b, [])) = X.
X = [a,b]
```

**The last item of a list should be the empty list []. When it is not, the notation is used to indicate the last item of the list.**

```prolog
?- (a, (b, c)) = X.
X = [a,b|c]
```

**atom_codes lets us convert back and forth between an atom in Prolog (similar to a symbol in Ruby) and its corresponding string name.**

```prolog
?- atom_codes(X, "ab").
X = ab
?- atom_codes(ab, "ab").
yes
```

**Strings are lists of character codes.**

```prolog
?- atom_codes(ab, X).
X = [97, 98]
?- X = "ab".
X = [97, 98]
?- atom_chars(ab, X).
X = [a,b]
```

**Prolog for logic databases**

```prolog
fact(one).
fact(two).
fact(three).
number1(A) :- fact(A).
```

**Introducing not \+\**

```prolog
notNumber1(A) :- \+ number1(A).
?- notNumber1(five).
yes
?- notNumber1(one).
no
?- notNumber1(X).
no
number2(A) :- fact(A).
```
number2(four).
| ?- number2(X), \+ number1(X).
X = four

**** Implementing forAll iterator with not and call

number1(X) :- fact(X).
number2(X) :- fact(X).
number2(four).
notNumber1(X) :- \+ number1(X).

forAll(Goal1, Goal2) :- \+ (call(Goal1), \+ call(Goal2)), call(Goal1), !.
| ?- forAll(number1(X), number2(X)).
X = one
| ?- forAll(number2(X), number1(X)).
X = one
| ?- forAll(number1(X), number1(X)).
X = one
| ?- forAll(number1(Y), number1(Y)).
X = one
| ?- forAll(number1(X), number2(Y)).
X = one
| ?- forAll(number2(X), number1(Y)).
X = one

thrice(A, B) :- B is A * 3.

**** Behavior of fail.

testInterpret2 :-
  thrice(4, A),
  write([A]),
  nl,
fail.
| ?- testInterpret2.
[12]
no

**** Writing a prolog interpreter. clause builtin searches database for rules
that match head or tail.

interpret(true) :- !.
interpret((A, B)) :-
  !,
  interpret(A),
  interpret(B).
interpret(A) :-
  clause(A, B),
  interpret(B).

parent(michael, cathy).
parent(melody, cathy).
parent(charles_grondon, michael).
parent(hazel, michael).
parent(jim, melody).
parent(eleanor, melody).

grandparent(A, B) :- parent(C, B), parent(A, C).
testInterpret :- interpret(grandparent(A, B)), write([A, B]), nl, fail.
| ?- testInterpret.
[charles_grondon, cathy]
[hazel, cathy]
[jim, cathy]
[eleanor, cathy]
no

prove2 :- clause(parent(michael, cathy), A), write([A]), nl, fail.
| ?- prove2.
[true]
no

greatgrandparent(A, B) :- parent(C, B), parent(D, C), parent(A, D).
testInterpret3 :- interpret(greatgrandparent(A, B)), write([A, B]), nl, fail.
yes

**** Adding stuff to the database at runtime using asserta

testInterpret4 :- asserta(parent(cathy, sam)), testInterpret3.

| ?- testInterpret4.
[charles_grondon, sam]
[hazel, sam]
[jim, sam]
[eleanor, sam]
no