Question 1

The most basic way of representing structure in linguistic expressions is bracketing. Some examples:

(1) a. Black coffee machine.
  b. *Reading 1:* ((Black coffee) machine)
  c. *Reading 2:* (Black (coffee machine))

(2) a. Old dogs and cats are happy.
  b. ((Old (dogs and cats)) (are happy)).
  c. (((Old dogs) and cats) (are happy)).

(3) a. She wished John knew some poems.
  b. *Only reading:* (She (wished (John (knew (some poems))))).

(4) a. She asked John to read some poems.
  b. *Only reading:* (She ((asked John) (to (read (some poems))))).

Provide brackets for the following sentences (some are from Winter 2016, p. 37, ex. 2):

(a) I read that Dan published an article in the newspaper.

(b) Sue is blond or tall and thin.

(c) I saw the man with the telescope.

(d) Sue told some man that Dan liked the story.

(e) Because the man that the police interrogated didn't call his lawyer was concerned.

(f) It's safe to believe the people the people you believe are smart believe are smart are smart, but not safe to trust the people the people you trust trust. (Hint: look for structures of the form *the man the woman likes*).

Question 2

Wouldn't it be nice to generate a random model for a given domain of entities (Winter's set E) and a vocabulary – a set of individual and predicate names? For such a task, you will need some utility functions that construct set-theoretic objects. You are asked to write some of these this week. (If you are using Python, don't use its `set` type; quite incredibly, Python sets are not recursive: you can't have sets as elements of sets.)

Implement the following:

- A predicate that checks whether a given list is a set (=no repetitions).
- A function that returns a random element from a set;
- A function that returns a random subset from a set; give user the option to specify the size of the list, if she doesn't want to pick a random size.
- A function that returns a random set of n-tuples generated from a given set. It might be handy to have a function taking the Cartesian product of all the sets in a set of sets; then you can randomly pick a subset from this product.

Find a sample interaction on the next page.

```
* (defparameter *test* '(a b c d))

*TEST*
* (random-pick *test*)

C
* (pick-a-subset *test*)

(A B)
* (pick-a-subset *test*)

(B D C)
* (pick-a-subset *test* 3)

(B A C)

* (cartesian-product '((a b c) (1 2 3)))

((B 3) (B 2) (B 1) (A 1) (A 2) (A 3) (C 1) (C 2) (C 3))
* (cartesian-product '((a b c) (1 2 3) (x y)))

((C 2 Y) (C 2 X) (A 3 Y) (A 3 X) (A 1 Y) (A 1 X) (B 2 Y) (B 2 X) (B 3 X)
 (B 3 Y) (B 1 X) (B 1 Y) (A 2 X) (A 2 Y) (C 1 X) (C 1 Y) (C 3 X) (C 3 Y))

* (generate-tuples *test* 2)

((D A) (C D) (C A) (A A) (B A) (A D) (D D) (C C) (A C) (D B) (B B) (D C))
* (generate-tuples *test* 3)

((A C C) (D D B) (A D D) (D A B) (D C D) (A D C) (B A A) (B C C) (B D D)
 (D B D) (B B D) (D A A) (D B A) (D D A) (B A C) (C C C) (C C A) (C D D)
 (C B B) (A B A) (A A A) (D B C) (B D B) (A B C) (C C B))
* (generate-tuples *test* 3)

((D B A) (B D B) (B B A) (A A D) (D D B) (B A C))
```