1. **Do exercise 3.7. Hint: in part (a), there's something Brad expects to happen that isn't happening. In part (b), there's something happening that Brad doesn't expect to happen.**
    a. Brad is expecting automatic garbage collection. There is a function available to delete the list_node struct, but he never calls it in his main function.
    b. Brad doesn't realize that the delete_list function is deleting the data in each of the nodes in L, so when Brad says that L = T, L has all of the structure of the node in memory, but the values in the nodes are freed memory, so each of them can be rewritten leading to serious instability of the program.
2. **Describe two mechanisms in your pet language for exiting a deeply nested control structure (e.g., nested loops). Java has return statements and exceptions.**
    a. Similar to javascript and java, there is the keyword "**break"** to exit a loop.
    b. There are also return statements in Coffeescript, returning in the middle of a loop will exit the method entirely.

3. **In each of Java, Haskell, and your pet language, name a left-associative and a right-associative operator. For each operator, give an example using parentheses to force the other order of association. (If this question does not apply to your pet language, explain.)**
    a. Java
        i. - **R->L** Subtraction. example:
        `5-3-1 = 1; SWITCH ORDER 5-(3-1) = 3;`
        ii. / **L->R** Division. example:
        `5/3/2 = 6.666666; SWITCH ORDER (5/3)/2= .8333333;`
    b. Haskell
        i. % **L->R** modulo Example:
        `10 % 6 % 4 = 1; SWITCH ORDER 10 % (6 % 4) = 0;`
        ii. ^ **R->L** carat. Example:
        `2^3^2 = 512; SWITCH ORDER (2^3)^2 = 64;`
    c. CoffeeScript
        i. + **L->R** addition Example:
        `2+5+3 = 10; SWITCH ORDER 2+(5+3) = 10;`
        ii. / **R->L** division Example:
        `5/3/2 = 6.666666; SWITCH ORDER (5/3)/2= .8333333;`

4. **Do exercise 6.5.**
    a. In Lisp the following would be ambiguous `(- 48 * 5 2 3)` because it could be evaluated as `(- 48 (* 5 2 3)) = (- 48 30) = 18.` It could also be evaluated as `(- 48 (* 5 2) 3) = (- 48 10 3) = 35.`
    When using prefix notation to avoid ambiguity a compiler would have to look for operators that are not surrounded by parentheses.
    Issues of precedence and associativity do not arise with prefix or postfix notation assuming the compiler actually knows how to handle prefix notation.