1. Do exercise 8.6. Note that you're using the *syntax* of the language of your choice, but *imagining* that the language uses each of these parameter passing modes. You'll need to read section 8.3.2 (Call-by-Name) from the book's virtual "CD", which is the file 8d_clbnm.pdf in the data/chapters directory of the CD zip file.

```
class CallBy {

      public static void main(String[] Args){
            int test = call(2+3);
            system.out.println(test);
      }

      public static void call(int a){
            System.out.println(
            a = 3;
      }
}
```

This will be different when testing call by value vs call by reference. In call by value this will print 0. In call by reference this will print 3.

2. Do exercise 9.21.

   If foo is an abstract class in a C++ program, why is it acceptable to declare variables of type foo* , but not of type foo ?

   You cannot instantiate an abstract class, so creating a variable of type foo is strictly prohibited. You can create a pointer to an abstract class, and that is what foo* is doing.

3. Do exercise 10.6. Your versions should also be in Scheme. The repl.it Scheme interpreter calls integer division div instead of quotient.
   a.
```
(define (log2' n)
     (if (= n 1)
      0
      (+ 1 (log2' (div (+ n 1) 2))))
)
```
   b. `(define (min’ l);;still thinking about this.`

4. Explain the exception-handling mechanism in your pet language, with an example. (If your language has no special mechanism for this, explain what a programmer would do to, e.g., let a file-reading subroutine to recover from a missing file.

   The exception handling in CoffeeScript is very similar to Java, we have try, catch, and finally. Ex:

```
try
      destroyComputer()
catch error
      print "Did you expect I would destroy myself?"
finally
      reassemble()
```

5. Does your pet language have more than one sense of equality? Explain, with examples.

Coffeescript is great, it gets rid of the JavaScript concept of comparing type and value. You simply type == for normal equal and != for not equal. In JavaScript == does not check type, so you could have "3" == 3 . Similarly, you can have 3 !== 3.0 because you are comparing a double to an int.