1. The following Scala program uses exceptions to propagate an error condition. Rewrite the program to use the `Maybe` type instead following the model on slide 6.

```scala
case object Failure extends Exception

def f (a:Int) : Int =
  if (a > 10) a-10 else throw Failure

def g (b:Int) : Int =
  f(b+b) + 10

def h (c:Int) =
  try {
    println(g(c)*5)
  } catch {
    case Failure => println("oops")
  }
```

2. Consider a function `samevals(t1, t2)` that takes two binary trees, not necessarily of the same shape, and returns a boolean saying whether or not their values as produced by an in-order traversal are the same. For example, `samevals` should return true for these trees:

```
       6
      / \
     9   1
    /   /\  \
   7   4 9  7
      /   \
     4    6
```

It is surprisingly difficult to write this function using ordinary recursive procedures (try it!). One simple approach is to extract the values from each tree into a list and then compare the lists, but this always requires $O(n)$ extra storage and $O(n)$ time, even if there is a mismatch in values early on.

But the problem (a variant of the well-known “same fringe” problem) is easy to solve using coroutines. Write a version of the `samevals` function using the facilities of Lua coroutines that are illustrated on lecture 5a slice 16, including the `walk` function. Pseudo-code is fine; there is no need to get the details of Lua syntax right—although if you want to try, Lua is very easy to learn: there's an online book at https://www.lua.org/pil/contents.html and you can test your code at https://www.lua.org/cgi-bin/demo.