Laziness

Much like how in the real world you have people that are lazy and people that are not, when it comes to evaluation order in programming languages you have lazy languages and eager languages. Haskell is lazy. It won't do something unless it absolutely has to. You can think of this much like students and lecturers:

```
student :: Question -> Solution

lecturer :: Int -> [Question] -> [Solution]
lecturer n xs = take n (map student xs)
```

Think of a `student` as a function that will turn a `Question` into a `Solution`, and think of a `lecturer` as a function that assigns a certain number (`n`) of `Questions` (`xs`) to `student` in order to get a list of `Solutions`. In the world of Haskell everyone is lazy so here is how the `Solutions` would be extracted. Say the `lecturer` is going to assign 3 `Questions`, i.e. `n = 3`. `lecturer` sees that `n` isn't zero yet so it says to `(map student xs)` 'Okay I need aSolution'', and thestudentreluctantly replies spitting out oneSolution. Nown = 2, thelecturerneeds moreSolutions so it kicks the inside function ofstudentdemanding that they do more work, and this pattern continues until thelecturer' is satisfied. Like taking blood from a stone.

If this was not Haskell and we were working with an eager language things would be very different. The `student` would be a very keen bean and immediately do all the `Questions` in the list, regardless of how many the `lecturer` assigns. This seems every diligent but think about what would happen if the list of `Questions` was infinite. The function would never terminate. Our poor `student` would work themselves to the bone perpetually answering `Questions`. Not very healthy.