2,750 words

0000000000000000000000000000000000000000000000000000000000000000000000000000000

What is laziness?

Lazy programming, and or evaluation, is the process by which expressions and assignments are deferred until required. It is often times very useful because it allows for computation that in “greedy” or “strict” programming would either take too long, or unable to compute. It particularly is useful for evaluating recursive and potentially infinite functions/data structures. For example, having a function that returns the Nth value of the fibinacci sequence would be extremely expensive to program using a strict language because it would try and create the fibinacci sequence up to some arbitrary value. Assigning the same task to a lazy language would be much better because you would be able to create the sequence, and it would not be evaluated until n is defined and given to the function returning the corresponding value from the sequence.

- What does laziness cost? (Either you, the programmer, or the computer.) Are some kinds of programs particularly hard or ill-suited to lazy evaluation? How could you make laziness better for them?

Some of the issues with lazy evaluation is that it is often times diﬃcult to implement eﬃciently. This adds a greater expense to the programmer by them having to spend more time implementing algorithms. It’s magnified since the order of evaluation can be counterintuitive which makes debugging diﬃcult. Evaluation can (and likely will) be out of order, and it has to be accounted for when doing several thing, debugging being one of them. Exception handling is also essential to consider with using lazy evaluation. Since by nature of the lazy program/language expressions aren’t evaluated until the value is required, a common mistake might be to call a function inside of a Try-Catch statement, or some other exception handling tool, and the program would crash if that value wasn’t required until outside of the exception handling. For example, if the following algorithm were implemented in a lazy language:

String response = “”  
 try{

response = getTextFromPage(url)

}catch(Exception e){

Throw “error”

}

Print response

rather than the exception being caught by the catch statement, the program will crash when getTextFromPage(String) (which might throw a network exception, or timeOut error) gets response is printed since getTextFromPage(url)

Granted, however, this isn’t always the case. Also, if lazy evaluation is combined with pattern matching.

- Is it necessary for laziness to be built-in to the language, such as Haskell or Lazy Racket? How could you use laziness in a "non-lazy" language, such as C?

- Discuss how laziness relates to modularity.

- How does, or should, mutation relate to laziness?

- Are there programs you have written in the past that would have been better had you understood laziness beforehand? Explain why, why not, how, etc.

- Have you used laziness before and did not realize it? Explain.

- Is laziness "better" or more "powerful" than strictness?