# FUNCTIONAL PROGRAMMING 2015

MOISE GABRIEL

## QUESTION 4

(a) "Lazy evaluation" is a method used by Haskell in which expressions are only evaluated when their results are needed by other processes. One advantage is that this way we can easily work with infinite lists, or pass undefined values to functions through pattern-matching. It can also be more time-efficient when we create a lot of values for a function, but only a small number of them are actually needed and used.

(b)

```
> allTriples :: [(Integer, Integer, Integer)]
> allTriples = concat [ triples s | s <- [0..]]

> triples :: Integer -> [(Integer, Integer, Integer)]
> triples s = [(a,b,c) | a <- [0..s], b <- [0..s], c <- [0..s], (a+b+c) == s]
```

(c)

```
> sieve :: [Integer] -> [Integer]
> sieve xs = filter (\x -> (x `mod` h) /= 0) xs
>           where h = head xs

> primesFunct :: [Integer] -> [Integer]
> primesFunct (x:xs) = x : primesFunct (sieve (x:xs))

> primes :: [Integer]
> primes = primesFunct [2..]
```

(d) By using Euler's formula to generate primitive Pythagorean triples we have

- $a = \min(m^2 - n^2, 2mn)$
- $b = \max(m^2 - n^2, 2mn)$
- $c = m^2 + n^2$
- $\gcd(m,n) = 1$
- $m > n > 0$
- m and n cannot be both odd

Here, we use tail to not take into consideration the triple (0, 1, 1):

```
> pythagora :: [(Integer, Integer, Integer)]
> pythagora = tail [(min (m*m-n*n) (2*m*n), max (m*m-n*n) (2*m*n), m*m+n*n) |
                    (m,n) <- allDoubles, m>n, gcd m n == 1, m*n `mod` 2 == 0]
```

```
> allDoubles :: [(integer, integer)]
> allDoubles = concat [ doubles  s | s <- [0..]]

> doubles :: [(integer, integer)]
> doubles s = [(a, s-a) | a <- [0..s]].
```