# **Project 1 - Goldbach’s other conjecture**

### [**Learn You a H**](http://learnyouahaskell.com/chapters)**askell**

[1](http://learnyouahaskell.com/introduction). Introduction

[2](http://learnyouahaskell.com/starting-out). Starting out

[3](http://learnyouahaskell.com/types-and-typeclasses). Types and Typeclasses

### **Videos (**[**link to all Drake videos**](https://www.youtube.com/playlist?list=PLAYqRAte9oRIChcPR_DD4uc8mCR6d3RiJ)**)**

[1. Introduction](http://www.youtube.com/watch?v=NBKnY7Z_w3I) (3:24)

[2a. Haskell as a calculator](http://www.youtube.com/watch?v=hJGEuFjcvZ8) (4:44)

[2b. Functions, if, and let](http://www.youtube.com/watch?v=YbqxYJLyaZE) (3:26)

[2c. Lists](http://www.youtube.com/watch?v=UTn7feUNr8g) (8:11)

[2d. List comprehensions](http://www.youtube.com/watch?v=qoQsDTzBojs) (2:27)

[2e. Tuples](http://www.youtube.com/watch?v=-8vF_vXtLjI) (3:38)

[3. Types and typeclasses](http://www.youtube.com/watch?v=x3uF7fcQwWE) (6:17)

**Total**: 32:07

**Goldbach’s conjecture** (so far neither proved nor disproved): Every even integer greater than 2 can be expressed as the sum of two primes. E.g., 34 = 29 + 5; 98 = 91 + 7.

**Goldbach’s other conjecture** (disproved): Every odd composite number can be expressed as the sum of a prime and twice a square. E.g., 35 = 17 + 2\*(3^2); 99 = 67 + 2\*(4^2).

The assignment is to write a program that finds the two smallest numbers for which Goldbach’s other conjecture does not hold and then stops.

In other words, write a program that finds the first two non-prime odd numbers so that for each there is no *prime p* and integer *k > 0* such that the number is equal to *p + 2 \* k^2*.

As a head start, we will develop some code that generates the prime numbers.

The following code uses the function takeWhile, not covered in chapters 1 - 3. takeWhile takes elements from a list (and constructs a new list) while some predicate holds. For example :

> takeWhile (> 4) [8, 7, 6, 5, 4, 3, 2, 3, 4, 5, 6, 7, 8]

[8,7,6,5]

Note that takeWhile is not the same as filter, which we will look at another time.

> filter (> 4) [8, 7, 6, 5, 4, 3, 2, 3, 4, 5, 6, 7, 8]

[8,7,6,5,5,6,7,8]

In both cases (> 4) is short for the anonymous function (\x -> x > 4). You should have seen anonymous functions in Java or some other language.

> takeWhile (\x -> x > 4) [8, 7, 6, 5, 4, 3, 2, 3, 4, 5, 6, 7, 8]

[8,7,6,5]

> filter (\x -> x > 4) [8, 7, 6, 5, 4, 3, 2, 3, 4, 5, 6, 7, 8]

[8,7,6,5,5,6,7,8]

The primes may be defined as follows.

oddsFrom3 = [3, 5 .. ]

primeDivisors n = [d | d <- takeWhile (\x -> x^2 <= n) primes, n `mod` d == 0]

primes = 2 : [n | n <- oddsFrom3, null (primeDivisors n)]

> take 15 primes

[2,3,5,7,11,13,17,19,23,29,31,37,41,43,47]

Notice that primeDivisors refers to primes even though we are in the process of defining primes. How can that be?

This is another form of recursion: we are reducing the current problem to a simpler one. To determine whether a number n is primes we use the fact that we have already computed the primes up to n-2.

Initially primes = [2].

To determine whether 3 is prime we look at primeDivisors 3. But primeDivisors 3 examines elements of primes whose squares are <= 3. But 2^2 > 3. So we stop immediately. So primeDivisors 3 = [], and 3 is prime. Now primes = [2, 3].

Next we ask whether 5 is prime. The only element of primes whose square is <= 5 is 2. But 5 `mod` 2 != 0. So primeDivisors 5 = [], and 5 is prime. Etc.

**Do you understand infinite lists?**

How do these lists differ? Which evaluate in a finite amount of time?

a) listA = [3, 5 ..]

b) listB = [x | x <- [3, 5 ..]]

c) listC = [x | x <- [3, 5 .. 5999]]

d) listD = [x | x <- [3, 5 .. 6000]]

e) listE = [x | x <- [3, 5 ..], (x < 6000)]

f) listF = [x | x <- takeWhile (\x -> x < 6000) [3, 5 ..] ]

In other words, for any pair of the preceding lists, listA and listB

a) Is it true that listA == listB?

b) Will Haskell return True for: > listA == listB

*Be able to explain why or why not?*

**Debugging aid:** trace :: String -> a -> a

The function trace takes two arguments. The first must evaluate to a String, which is printed. It then evaluates and returns the second.

The function show is like Java’s toString(); it isn’t a print command. Here’s how to use trace to watch how primes works.

primes = 2 : [trace (show p ++ " is prime.") p | p <- oddsFrom3,

trace (show p)

(null (primeDivisors p))]

> take 6 primes

3

3 is prime.

5

5 is prime.

7

7 is prime.

9

11

11 is prime.

13

13 is prime.

15

17

17 is prime.

[2,3,5,7,11,13,17]

**Hints for doing the Goldbach’s other conjecture problem.**

1. The two numbers that fail Goldbach’s other conjecture are 5777 and 5993.

2. For each odd non-prime g determine if there is a p and k such that

g = p + 2 \* k^2. If there is no p and k, g is one of the numbers you are looking for.

3. *Caution*: If you use brute force to try all combinations of p and k, your program will run for a very long time. Instead, look for a prime p such that (g - p)/2 is a square.

4. You may use isASquare to determine whether an integer is a perfect square.

iSqrt n = floor (sqrt (fromIntegral n))

isASquare n = (iSqrt n) ^ 2 == n

5. You will also find it useful to write a function isPrime that returns True or False depending on whether its argument is prime. What’s wrong with this?

isPrime n = n `elem` primes

The problem is that n `elem` primes asks whether n is an element of primes. Ask yourself how long the list primes is.