Orderly Chaos: Prime numbers, part 2

Martin Roelfs

April 8, 2020

1 Section Title

1.1 Introduction

Part 2 of this exercise assumes you have completed the exercises in prime numbers part 1.

You will build upon the skills you have gained in the previous exercises and get a feeling for how to approach the same problem in a different way, and how different approaches to a problem affect the performance of your code.

2 Exercises

Exercise 2: Sieve of Eratosthenes

The method in Exercise 1 is inefficient because it does not use anything it might have learned in the previous iterations of the loop. For example, once you know that 2 is a prime number, you also know that all other even numbers can't be prime. So why bother with checking other even numbers? If you were to stop checking other even numbers, you would save practically 50% of the work. And for all multiples of 3 you would save 33.33...%, etc. The oldest algorithm that makes use of this knowledge is known as the sieve of Eratosthenes, a polymath from Alexandria.

The basics of the algorithm are as follows:

1. Create an ordered list from 2 to the maximal value to be checked.
2. Take the smallest value from the list.
3. Remove all values which are a multiple of this value, but not the value itself.
4. Take the next number in the list, and repeat the previous step.

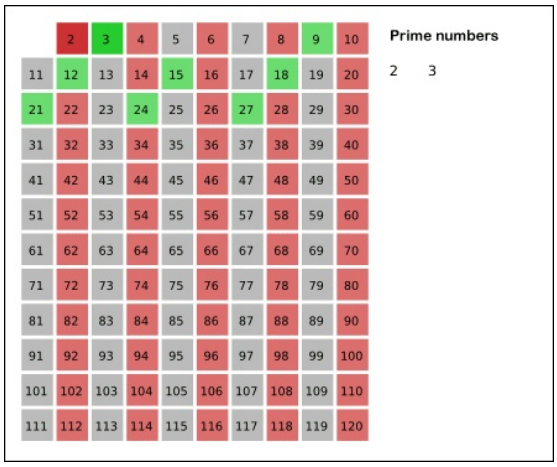

Figure 1: The sieve throws away any multiples of primes. Image courtesy of Wikipedia

pros Probably the best way to find prime numbers without the help of a computer.

cons On a computer, you have the great difficulty that you have to keep a list from [2, max] in memory. This means that with very large numbers, you simply can't hold that in working memory. The maximum size for a Python list on a 32 bit system is 536870912 elements. Only an estimated ± 26708310 of these will be prime numbers, which is about 4.97%. So we are taking up a lot of memory with numbers we aren't even interested in. And this discrepancy will only get worse as values increase, because the density of primes decreases the higher you get.

To make this Sieve, I suggest you follow the following steps.

1. Make a function that replaces all multiples of primes in a list (range(2,max)), by 0. The list is needed to make sure range really returns a list of numbers.

   ```
   def crude_sieve(maximum):
   ```

2. Make a function that removes all the zero's from the list.

   ```
   def remove_zeros(number_list):
   ```

3. Finally, combine these to form a function sieve(maximum), which gives you back a list of all the primes under the value of max.

   ```
   def sieve(maximum):
   ```

2.1 Exercise 3

The Sieve of Eratosthenes is not commonly used as a way to find primes because of the drawbacks mentioned above. However, it should also be clear that it has some great strengths over the simple brute-force way of exercise 1.

What makes the sieve a strong tool is the fact that in a way, it remembers the primes it has already divided by. In doing so it will not make the mistake of dividing by a number which is itself a multiple of prime numbers.

1. Make a script that stores any primes it finds, and divides new numbers only by these primes. That way, we greatly reduce the memory space used, as well as the number of calculations needed.

2. As a further optimisation, keep in mind that the largest number any number can be divided by, is its square root. Therefore, it is not necessary to check for divisibility by any number which is larger than the square root of the target number. Incorporate this idea into your code.