Orderly Chaos: Prime numbers, part 1

Martin Roelfs

April 8, 2020

1 Section Title

1.1 Introduction

Prime numbers have fascinated mathematicians since the Ancient Greeks. These numbers are only divisible by 1 and by themselves. Dividing by any other number will give a remainder. Prime numbers are the building blocks of numbers, because every other number can be made by factoring primes. For example, $24 = 2^3 3^1$ and 2 and 3 are prime numbers.

For a long time prime numbers where nothing but a mathematical curiosity. Fun for mathematicians, but of little use to anyone else. That all changed with the introduction of the computer. It wasn't long before there was a great need to encrypt data so it is hidden from unwanted eyes. And when it comes to encryption, nothing beats the primes.

In this assignment you will create a Python script to find your own prime numbers. This will be done in a couple of ways to give you a feeling for how the performance of your prime-finder can be improved by using a different algorithm.

2 Exercises

Exercise 1

The simplest way to check if a number is prime, is to divide it by all smaller numbers to see if one of them divides it. If this works for one of those smaller numbers, the number is not prime. If however, the number can't be divided by any other number, it is prime.

1. Think of a good check for divisibility. Try to use as little code as possible. You should be able to do all the work with a single operator. *Hint: 6 does not divide 7, but 7 does divide 7. How could a program spot the difference?*

2. Write a for loop that divides a number of your choosing by all numbers smaller than itself, staring from 2. For example, let's pick 8. If a certain number divides your target, print() the message "2 divides 8". If however a number does not divide your target, print() the message "3 does not

divide 8". *Hint: look into the* range() *function to find out how to loop over all the values smaller than your target but not smaller than* 2.

Try to run this loop with several different targets to ensure yourself that it distinguishes primes from non-primes properly.

3. Of course this is way to much information. We only want to know if a number is prime or not, we do not need to know which number was to blame for dividing it. We will therefore write a function is_prime(target) that will return True if the target is indeed prime, or False otherwise. The code for this function should basically be the same as your code for the previous question, but turned into a function. This step is called "Encapsulation and Generalization".

   ```
   def is_prime(target):
   ```

4. You probably used a for loop to create your is_prime(target) function. Although the for loop is easy to understand, it should also be possible to do the same thing with a while loop. Make the same function again, but now use a while loop. Give it a different name because you will need both versions for the next question. P.S. if you get stuck in an infinite loop, you can always escape by pressing ctrl+break on Windows, or ctrl+c on Unix.

5. It is now possible to check large ranges of numbers for primes within those ranges. Create a function primes_between(start, end) that checks a range of your liking for primes. It should print() the primes that it finds, and keep quiet if a number is not a prime.

   ```
   def primes_between(start, end):
   ```

6. Now it is time for a very basic optimisation. It is not necessary to divide by any number which is larger than the square root of the target you are checking. Change the is_prime(target) function that you prefer to take this into account.

   Theory If a target number can be divided by a certain number, it can automatically be divided by the 'complement' of that number. For example, $\frac{20}{2} = 10$ and so 10 also divides 20. Therefore, the biggest unique value a number can be divided by, is its square root.

7. Create a function largest_prime(end) that returns the largest possible prime smaller than a certain maximal value.

   ```
   def largest_prime(end):
   ```

   It might be smart to think twice about how you build this function. When using this brute-force approach there is a right and a very wrong way to do this. *Hint: keep in mind that you might want to know the largest prime under* 10^{100}.