Lesson Thirteen

Math 6080 (for the Masters Teaching Program), Summer 2020

Euclid's Proof of Euclid's Theorem. Euclid's proof that there are infinitely many primes is constructive. Start with the smallest prime 2. Then:

$$2 + 1 = 3$$

is also prime. Next,

$$2 * 3 + 1 = 7$$

is a new prime that we one adds to the list:

$$2 * 3 * 7 + 1 = 43$$

is prime, to be added to the list as well. Now,

$$2 * 3 * 7 * 43 + 1 = 1807 = 13 * 139$$

is **not** a prime, but each prime factor of is distinct from the primes on the list. Let's add the smallest one to the list and proceed.

$$2 * 3 * 7 * 13 * 43 + 1 = ...$$

This can be run indefinitely, generating a new prime with each iteration. This is effectively Euclid's original proof for the infinitude of the primes. The primes in this list, however, grow in size extremely quickly.

Implementing this on Python involves picking out the smallest factors of very large numbers, which very quickly becomes prohibitively difficult. Is there a way to keep the sizes of the new primes down, so that Python can generate more primes? One answer would be to keep the new prime 13 above and discard the larger prime 43, culling the list down to $[2, 3, 7, 13]$ and then continue:

$$2 * 3 * 7 * 13 + 1 = 547$$

is prime. This gets one a bit farther:

$$2 * 3 * 7 * 13 * 547 + 1 = 97 * 3079$$

so we can add 97 to the mix and remove 547....

This also blows up, producing numbers that are so large that testing for a factor becomes intractable. But do we really need to factor them?

See the following code and figure out what cleverness was employed. Run it for fun! It does a good job of finding small primes until it blows up... Try to imagine other schemes for using Euclid's idea to produce even more small primes using Python.