



# Daily Coding Problem #244

## Problem

This problem was asked by Square.

The Sieve of Eratosthenes is an algorithm used to generate all prime numbers smaller than N. The method is to take increasingly larger prime numbers, and mark their multiples as composite.

For example, to find all primes less than 100, we would first mark [4, 6, 8, ...] (multiples of two), then [6, 9, 12, ...] (multiples of three), and so on. Once we have done this for all primes less than N, the unmarked numbers that remain will be prime.

Implement this algorithm.

Bonus: Create a generator that produces primes indefinitely (that is, without taking N as an input).

## Solution

Despite being very old, the Sieve of Eratosthenes is a fairly efficient method of finding primes. As described above, here is how it could be implemented:

```
def primes(n):
    is_prime = [False] * 2 + [True] * (n - 1)

    for x in range(n):
        if is_prime[x]:
            for i in range(2 * x, n, x):
                is_prime[i] = False

    for i in range(n):
        if is_prime[i]:
            yield i
```

There are a few ways we can improve this. First, note that for any prime number  $p$ , the first useful multiple to check is actually  $p^2$ , not  $2 * p$ ! This is because all numbers  $2 * p, 3 * p, \dots, i * p$  where  $i < p$  will already have been marked when iterating over the multiples of  $2, 3, \dots, i$  respectively.

As a consequence of this we can make another optimization: since we only care about  $p^2$  and above, there is no need for  $x$  to range all the way up to  $N$ : we can stop at the square root of  $N$  instead.

Taken together, these improvements would look like this:

```
def primes(n):
    is_prime = [False] * 2 + [True] * (n - 1)

    for x in range(int(n ** 0.5)):
        if is_prime[x]:
            for i in range(x ** 2, n, x):
                is_prime[i] = False

    for i in range(n):
        if is_prime[i]:
            yield i
```

Finally, to generate primes without limit we need to rethink our data structure, as we can no longer store a boolean list to represent each number. Instead, we must keep track of the lowest unmarked multiple of each prime, so that when evaluating a new number we can check if it is such a multiple, and mark it as composite. This is a good candidate for a heap-based solution.

Here is how we could implement this. We start a counter at 2 and incrementally move up through the integers. The first time we come across a prime number  $p$ , we add it to a min-heap with priority  $p^2$  (using the optimization noted above), and `yield` it. Whenever we come across an integer equal with this priority, we pop the corresponding key and reinsert it with a new priority equal to the next multiple of  $p$ .

For integers between 2 and 10, we would perform the following actions:

```
2: push [4, 2], yield 2
3: push [9, 3], yield 3
4: pop [4, 2], push [6, 2]
5: push [25, 5], yield 5
6: pop [6, 2], push [8, 2]
7: push [49, 7], yield 7
8: pop [8, 2], push [10, 2]
9: pop [9, 3], push [12, 3]
```

An important thing to note is that at any given time the next composite number will be first in the heap, so it suffices to check and update only the highest-priority element.

```

import heapq

def primes():
    composite = []
    i = 2

    while True:
        if composite and i == composite[0][0]:
            while composite[0][0] == i:
                multiple, p = heapq.heappop(composite)
                heapq.heappush(composite, [multiple + p, p])

            else:
                heapq.heappush(composite, [i*i, i])
                yield i

        i += 1

```

The time complexity is the same as above, as we are implementing the same algorithm. However, at the point when our algorithm considers an integer  $N$ , we will already have popped all the composite numbers less than  $N$  from the heap, leaving only multiples of the prime ones. Since there are approximately  $N / \log N$  primes up to  $N$ , the space complexity has been reduced to  $O(N / \log N)$ .

© Daily Coding Problem 2019

[Privacy Policy](#)

[Terms of Service](#)

[Press](#)