

Forum: [Hack Forums](#)

Thread: [Calculate primes](#)

maarten1012 - November 26, 2013, 11:44 am

Hey,

I was bored today, so i made a little program that calculates primes..
it works, but my question is how can i make it more efficient?

Here is the source:

Code class Program

```
{
    private static int num = 1;
    private static bool unlimited = false;

    static void Main()
    {
        while(true)
        {
            if (IsPrime(num))
            {
                Console.WriteLine(num);
                if (!unlimited)
                {
                    Console.ReadKey();
                    Console.Clear();
                }
            }
            num += 1;
        }
    }

    public static bool IsPrime(int number)
    {
        double boundary = Math.Floor(Math.Sqrt(number));

        if (number == 1) return false;
        if (number == 2) return true;

        for (int i = 2; i <= boundary; ++i)
        {
            if (number % i == 0) return false;
        }

        return true;
    }
}
```

Thanks in advance!

Ban - November 26, 2013, 1:39 pm

Here, quick modification of yours:

<http://pastebin.com/KUTjX1Nr>

By the way, `.Equals()` tests equality and `==` tests reference. `.Equals` can be slower however it can also be far more effective than `==` when comparing objects. Practically there really is no difference since it's a defined type and being someone who also programs in Java, it's just a little more natural for me to see `.Equals`.

I do it both ways however. Just letting you know. ^_^

0xDEAD10CC - November 26, 2013, 7:28 pm

None of these are good at all. Here is a few pointers:

1. No primes below 2.
2. 2 is the only even prime number, you shouldn't have to check even numbers other than 2 for being prime.
3. There are better algorithms out there for generating primes than just a simple looping through numbers in the x-y range strategy.

Here is my prime function:

Code static bool Prime(int n)

```
{
    if (n == 2) return true;
    if (n < 2 || (n & 1) == 0) return false;
    int limit = (int) Math.Sqrt(n);
    for (int i = 3; i <= limit; i += 2) if (n % i == 0) return false;
    return true;
}
```

But for a more efficient method of generating primes, I would do something like the following:

Code static void GenSieveAtkin(int max, ref List<int> primes)

```
{
    if (max > 3) { primes.Add(2); primes.Add(3); }
    bool[] p = new bool[max];
    int sqrtLimit = (int) Math.Sqrt(max);
    for (int x = 1; x <= sqrtLimit; x++)
    {
        int x2 = x * x;
        for (int y = 1; y <= sqrtLimit; y++)
        {
            int y2 = y * y;
            int n = (4 * x2) + (y2);
            if (n <= max && (n % 12 == 1 || n % 12 == 5))
                p[n] ^= true;
            n = (3 * x2) + (y2);
            if (n <= max && n % 12 == 7)
                p[n] ^= true;
            n = (3 * x2) - (y2);
            if (x > y && n <= max && n % 12 == 11)
                p[n] ^= true;
        }
    }
    if (max < 5) return;
}
```

```

int a = 5;
for (; a <= sqrtLimit; a += 2)
{
    if (p[a])
    {
        int a2 = a*a;
        for (int i = a2; i < max; i += a2) p[i] = false;
        primes.Add(a);
    }
}
for (; a < max; a += 2) if (p[a]) primes.Add(a);
}

```

```

public static void MyMethod()
{
    const int max = 1000;
    List<int> primes = new List<int>(max);
    GenSieveAtkin(max, ref primes);
    foreach (var prime in primes)
    {
        Console.WriteLine(prime);
    }
}

```