# Miller Rabin primality test

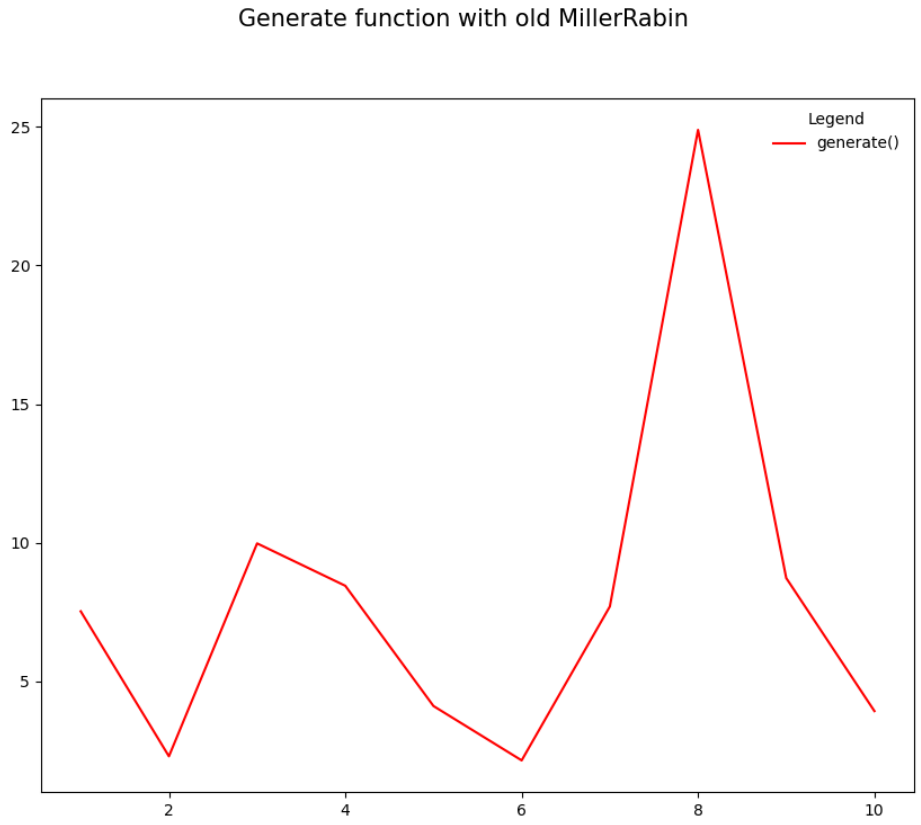
The first algorithm I wanted to improve was the Miller-Rabin primality test, the algorithm that check the primality of a given number. The keypair generator is mainly based on this test, the remaining code is about picking a random number and verify some RSA properties. My observation is the following: the time of generation for a keypair vary a lot from one to another. For instance one keypair can take 4s and another one 20s. Which is logical because the point of my algorithm is to, first, pick a random number then verify is primality. Which doesn't give us the guarantee to find quickly a prime number. It can be instantly as it can be endless. Our goal is to optimize the algorithm so it can verify more number is a shorter time to improve our chance to find a prime number.

Let's start. This what the Miller-Rabin look like at the beginning:

In [ ]:

```
from random import randint

def miller_rabin(p, r):
    """Credit: https://rosettacode.org/wiki/Miller%E2%80%93Rabin_primali
    s = 0
    d = p-1
    while d%2==0:
        d>>=1
        s+=1
    assert(pow(2, s) * d == p-1)

    def trial_composite(a):
        if pow(a, d, p) == 1:
            return 0

        for i in range(s):
            if pow(a, pow(2, i) * d, p) == p-1:
                return 0
        return 1

    for _ in range(r): #number of trials
        a = randint(2, p)
        if trial_composite(a):
            return 0
    return 1
```

And this is how long it takes to generate 10 keypairs. In ordinate you see the time generation in second and in ascissa the number of keypair:

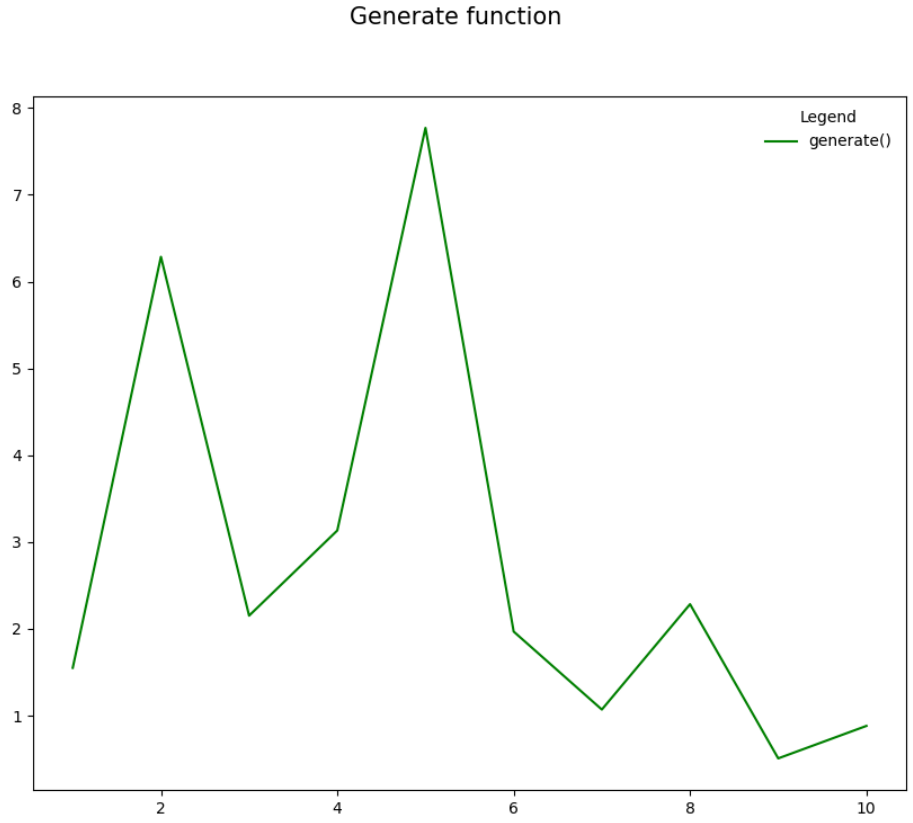



As we can see, the Miller-Rabin test is quite heavy. There are a lot of loop, condition and even a function in the main function. I've been looking for a lot of different implementation on the internet since I started coding this function and I didn't find better than this one. So I asked chatGPT to improve my algorithm and indeed this function into a function was the issue. He gave me this correction:

In [ ]:

```python
def miller_rabin(p, r):
    s, d = 0, p - 1
    while d % 2 == 0:
        d >>= 1
        s += 1

    for _ in range(r):
        a = randint(2, p - 2)
        x = pow(a, d, p)
        if x == 1 or x == p - 1:
            continue
        for _ in range(s - 1):
            x = pow(x, 2, p)
            if x == p - 1:
                break
        else:
            return 0

    return 1
```

It simply removing the function with a loop. The new function look a bit different from the old one but they remains the same thing. Here is a speed test of the new function:




As we can clearly see, this small change improved a lot the time of generation. My program can now generate many keypairs in a few second only! The goal has been reached and I can merge the benchmark branch into the main.

# Conclusion:

Be simple in your code, don't try to create weird alternative like a function in another function. It make the code harder to read and as you saw, lead to unexpected behaviour. A word about the use of chatGPT: I consider chatGPT like a very useful tool but in my case the correction was small and the requested task wasn't too big. You should keep it that way because if you ask chatGPT important request, it then become very confusing and lead nowhere. I already had bad experience that became a total waist of time because chatGPT isn't a magical tool that can handle everything. In resume chatGPT is very good, but with straight request ans small task. I actually recommend to use it in those cases, you gain time.

Archie.