Primality tests

Pseudoprimes. Recall that n is a pseudoprime to the base a coprime to n if $a^{n-1} \equiv 1 \pmod{n}$ but n is composite.

The following program makes a list of pseudoprimes to the base 2. Change 2000 by the upper limit of the list.

```
#
# pseudoprimes to the base 2
#
for n in range(3,2000,2):
    if Mod(2,n)^(n-1)==1:
        if is_prime(n)==False:
            print n, 'passes the test but is not prime'
```

Preserving 2000 the obtained list is short:

```
341 passes the test but is not prime
561 passes the test but is not prime
645 passes the test but is not prime
1105 passes the test but is not prime
1387 passes the test but is not prime
1729 passes the test but is not prime
1905 passes the test but is not prime
```

and it is even shorter if we impose n to be also pseudoprimes to the base 3

```
#
# pseudoprimes to the base 2 and 3
#
for n in range(5,2000,2):
    if (Mod(2,n)^(n-1)==1) and (Mod(3,n)^(n-1)==1):
        if is_prime(n)==False:
            print n, 'passes the test but is not prime'
```

```
1105 passes the test but is not prime
1729 passes the test but is not prime
```

A variation is considering a list of bases

```
#
# pseudoprimes to the bases of a list
#

def pseud(n):
    for k in list:
        if Mod(k,n)^(n-1) != 1:
            return False
    return True

list =[2,3,5,7]
for n in range(5,50000,2):
    if (is_prime(n)==False) and (pseud(n)==True):
        print n, 'passes the test but is not prime'
```

The result is again meagre although the upper limit is now 50000.

```
29341 passes the test but is not prime
46657 passes the test but is not prime
```

Strong pseudoprimes. Given n odd we can write $n-1=2^kq$ with q odd.

The following lines of code compute q and k for a given n

```
def q_and_k(n):
     q = n-1
     k = 0
     while Mod(q,2)==0:
          k += 1
          q = q/2
     return q,k
```

Running

```
print q_and_k(23)
print q_and_k(57)
print q_and_k(65537)
```

we get

`(11, 1)`		$23-1=2^1\cdot 11$
`(7, 3)`	meaning	$57-1=2^3\cdot 7$
`(1, 16)`		$65537-1=2^{16}$

A *strong pseudoprime to the base* a is an odd composite number n such that either $a^q\equiv 1 \pmod{n}$ or $\exists\, 0\le t<k$ such that $a^{2^tq}\equiv -1 \pmod{n}$.

With the following program we check if a number n passes to test to be a strong pseudoprime to the base a.

```
def strong_pseud(n,a):
    (q,k) = q_and_k(n)
    b = Mod(a,n)^q
    if b==1:
        return True

    for i in range(k):
        if b==-1:
            return True
        b = Mod(b,n)^2
    return False
```

For instance, the output of

```
print strong_pseud(172947529,17)
print strong_pseud(172947529,23)
```

is `True` and `False`. We have $172947529=307\cdot 613\cdot 919$.

Miller-Rabin test. This is a test broadly employed in practice. It simply checks if an odd number passes the test for strongs pseudoprime to many bases a.

These bases are commonly chosen as the first primes or as random numbers.

In the second version it is known that assuming a conjecture in analytic number theory (the Generalized Riemann Hypothesis) there are not composite numbers n passing the test for more than $2(\log n)^2$.

This two flavors of the test are contained in the following functions:

```
#
# Miller-Rabin
#
def miller_rabin(n,secur):
    for p in primes_first_n(secur):
        # Small primes
        if Mod(n,p)==0:
            if n==p:
                return 'Prime'
            return 'Composite'
        # Check strong pseudoprimes
        if strong_pseud(n,p)==False:
            return 'Composite'
    return 'likely prime'

def miller_rabin2(n,secur):
    a = 0
    for i in range(secur):
        a = 2+ZZ.random_element( n-3 )
        if Mod(n,a)==0:
            return 'Composite'
        if strong_pseud(n,a)==False:
            return 'Composite'
    return 'likely prime'
```

The parameter `secur` indicates the number of bases taken in consideration.

With `secur = 2` we get a couple of faliures of the primality tests for $n < 50000$

```
secur = 2
for n in range(3,50000,2):
    if (is_prime(n)==False)and(miller_rabin2(n,secur)=='likely prime'):
        print n, 'passes the test but is not prime'
```

```
5461 passes the test but is not prime
31621 passes the test but is not prime
```

and these exceptions disappear taking `secur = 3`.

For `secur = 9` it can be checked that there are not exceptions less than $3.8 \cdot 10^{18}$. In fact the first exception is $3825123056546413051 = 149491 \cdot 747451 \cdot 34233211$ that can be ruled out with `secur = 12`.

Although its efficiency in practice, Miller-Rabin test is not a *deterministic* primality test. Its output is 'composite' or 'very likely prime'.

There are some deterministic tests, the most important is the *cyclotomic test* (Adleman-Pomerance-Rumely 1983) that runs in time $(\log n)^{O(\log\log\log n)}$. It is not very simple and requires methods of algebraic number theory.

Commonly the best option (regarding to performance) is to apply very sophisticated primality test only when we have tried direct division for some number and Miller-Rabin test because they allow to rule out the most of the composite number with almost no effort.