2019-12-03 Public key encryption

idea behind RSA is that factoring is hard.
But checking primes is easy.

```
> isprime(10^200 + 123456777)
                                false                                  (1)
```

Don't ask maple to factor $10^{200}+123456777$... if you do, be prepared to wait a LOOOOONG time.

```
> isprime(10^40 + 123456789);
                                false                                  (2)
> isprime(10^40 + 123456829)
                                true                                   (3)
> ifactor(10^40 + 123456777)
        (7) (6208565384257) (7787686710839471) (29546240513)           (4)
```

Observe that the time needed to factor $10^{40}+123456777$ was way longer than the time needed to check if it was prime.
This is because primaility testing is easy, but factoring is hard.

We know that in integers mod n, if gcd(k,n)=1, then k is invertible mod n.

RSA (and its friends) rely on computing $x^a \bmod n$ and being able to invert it.

if n is prime, everything is good, that is everything in [0,1,2,3,....,n-1] has an inverse mod n, and powers $x^a \bmod n$ are all invertible.

What if n = p*q, both p and q prime? want to find power a so that $x^a \bmod n$ has an inverse for all x <n
This works for some powers, not for others.

Fermat's little theorem   says that for p prime, any 0< $x < p$, we have $x^{p-1} = 1 \bmod p$.

```
> p := nextprime(2000)
                              p := 2003                                (5)
> 47^(p-1) mod p
                                  1                                    (6)
```

what if we do this with n, with n=p*q?

Let's just try some.

```
> Some := [3, 5, 8, 19, 6, 42 , 18]
                    Some := [3, 5, 8, 19, 6, 42, 18]                   (7)
> q := nextprime(1234);
                             dfq := 1237                               (8)
> n := p·q;
                            n := 2477711                               (9)
```

```
> map(x→x^(n-1) mod n, Some)
        [2185444, 1976834, 454503, 1762278, 1174228, 2352680, 2133445]        (10)
```
So raising to n-1 power and reducing mod n doesn't work. But as we know, it works with a prime.
```
> map(x→x^(p-1) mod p, Some)
                          [1, 1, 1, 1, 1, 1, 1]                                (11)
```
Euler's theorem: for n=p*q with p,q prime any 0< $x < n$, we have $x^{\mathrm{phi}} = 1 \bmod n$. where phi=(p-1)*(q-1).

Phi(n) is called "Euler's Totient function". For a prime p, phi(p)=p-1. For a product of two primes p and q,
Phi(n) is (p-1)*(q-1). For more complicated composites n , phi(n) is more complicated, but that is irrelevant for us here.

```
> with(NumberTheory) :
> Totient(p); p
                                 2002
                                 2003                                          (12)
> Totient(n); (p-1)·(q-1)
                               2474472
                               2474472                                         (13)
> phiN := Totient(n);
                           phiN := 2474472                                     (14)
> map(x→x^phiN mod n, Some)
                          [1, 1, 1, 1, 1, 1, 1]                                (15)
```

The idea of RSA is that factoring is hard and so computing phi(n) is hard if we just know n, not its factorization.

Here's how to do it:

1) Choose 2 primes p and q Let n=p*q.
(in practice, p and q need to be about 150 digits long, maybe more. These days, RSA uses primes whose product is 1024, 2048, or 4096 bits, so p and q are between 150 and 600 decimal digits long. We will use shorter ones).
2) Choose exponent a with gcd(a,phi(n)) = 1 (a=3 is pretty common, but whatever makes you happy.)
3) public key is (n, a)

4) let d=1/a mod phi.
5) private key is (n, d)

To encrypt, represent your message as a list of numbers < n
for each number x, encryption is $y = x^a \bmod n$

to decrypt y, compute $y^d \mathbf{mod}\ n$.

why?

since d=1/a mod phi,   d*a = 1 mod phi, so d*a = 1+ k *phi

Write [x] for x mod n,, so $y = x^a$ and remember that by Euler $[x^{\mathrm{phi}}] = [1]$

$[y^d] = [(x^a)^d] = [x^{a \cdot d}] = [x^{1+k \cdot \mathrm{phi}}] = [x] \cdot [x^{k \cdot \mathrm{phi}}] = [x][x^{\mathrm{phi}}]^k = [x] \cdot 1$

```
> Some;
```
$$[3, 5, 8, 19, 6, 42, 18] \tag{16}$$

```
> p := 11; q := 17; n := p·q
```
$$p := 11$$
$$q := 17$$
$$n := 187 \tag{17}$$

```
> phiN := Totient(n)
```
$$phiN := 160 \tag{18}$$

```
> a := 11
```
$$a := 11 \tag{19}$$

so our public key is the pair (n,a) = (187, 11)

```
> crypted := map(x→x^a mod n, Some)
```
$$crypted := [58, 181, 19, 8, 39, 53, 18] \tag{20}$$

```
> d := 1/a mod phiN;
```
$$d := 131 \tag{21}$$

```
> map(x→x^d mod n, crypted)
```
$$[3, 5, 8, 19, 6, 42, 18] \tag{22}$$

In fact, all this can be done using Charmichael's Lambda function instead of Euler Totient, which can give smaller exponents (so less work). The Lambda function divides the Totient and the result is a power of 2. For primes p and q, the Lambda function is just the least common multiple of (p-1) and (q-1), but the result of the Euler theorem also holds if we use the Lambda function instead.

```
> CarmichaelLambda(n)
```
$$80 \tag{23}$$

```
> lcm(p − 1, q − 1)
```
$$80 \tag{24}$$

```
> d2 := 1/a mod CarmichaelLambda(n)
```
$$d2 := 51 \tag{25}$$

```
> map(x→x^d2 mod n, crypted)
```
$$[3, 5, 8, 19, 6, 42, 18] \tag{26}$$

Either works, but using Lambda instead of Phi is computationally easier, since 51 is less that 131.
Sometimes they are equal (for example, if a=7, 1/7 mod 80 = 23 = 1/7 mod 160) but often using Lambda gives a smaller decrypting exponent, and never larger.