2019-12-05 ... Last Class :-(or maybe :-p or ;-). depends on you.?
Maple has a page about RSA built in. See

```
> ?RSA Encryption
>
```

Continuing from last time, let's actually write maple procedure that does RSA encryption.
As usual, want to steal StringToList, ListToString from Crypto.mw

```
> with(StringTools) :
```

StringToList, ListToString

StringToList converts a string into a list of numbers representing the position of each character in the Alphabet.
ListToString converts such a list back into a text string.

Note that this differs from what we did in class on Nov 12 and 14, in that Alphabet[n] is represented by n-1. This will be more convenient.
These routines have been revised to allow for specification of an alternative base, as on Nov.21's class, but the changes are backward compatible. Also ignores characters not found in the Alphabet, issuing a warning. This behavior is controlled by a switch keepbad (default: false)

```
> StringToList:=proc(str::string, {base::posint:=1},
  {keepbad::truefalse:=false})
    local numlist;
    global Alphabet;
    numlist:=map( s->SearchText(s,Alphabet)-1, Explode(str));
    if (not(keepbad)) then
      numlist:=remove(x->x<0,numlist);
      if (nops(numlist)<length(str)) then
        WARNING("%1 characters have been ignored because they
  don't occur in the Alphabet", length(str)-nops(numlist));
      fi;
    fi;
    if (base>1) then
      numlist:=convert(numlist, ':-base', length(Alphabet),
  base);
    fi;
    return(numlist);
  end:
> ListToString:=proc(nums::list(nonnegint), {base::posint:=1})
    local numlist;
    global Alphabet;
    if (base>1) then
      numlist:=convert(nums, ':-base', base, length(Alphabet));
    else
      numlist:=nums;
    fi;
    return(Implode(map(k->Alphabet[k+1],numlist)) );
  end:
```

```
> Alphabet := Select(IsPrintable, convert( [seq(i, i = 1 ..127) ], bytes) );
```

$$Alphabet := \text{" !\"\#\$\%'()*+,-./0123456789:<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]\^_`abcdefghijklmnopqrstuvwxyz\{|\}~"} \quad (1)$$

Let's make RSA go, with 100-digit or so primes. We'll do this first "by hand", then write a general purpose procedure to do it.

```
> with(NumberTheory) :
  with(RandomTools) :
```

RSA relies on the fact that factoring the product of two big primes is computationally hard. With current technology, 150-600 digit primes is considered safe.

How can we get a big prime? Maple has a nextprime function that gives the next prime bigger than something, so

```
> nextprime(10^100)
```

$$100000000000000000000000000000000000000000000000000000000000000000000000000\backslash$$
$$00000000000000000000267 \quad (2)$$

is the first prime bigger than 10^{100}, that is, the smallest 101-digit prime.

So to get a 100-digit prime, we generate a random 100-digit number, and take the next prime bigger. We might as well a Blum-Blum-Shub generator, since it is a cryptographically secure random number generator.

```
> rando := BlumBlumShub:-NewGenerator(range = 10^99 .. 10^100) :
> rando( ); rando( )
```

$$30377124371726666464582043148101155151709419296032239642603151061097241955758\backslash$$
$$81673391517010513162324$$
$$43337745069224282727095779078959190670767279546685307309200103335839457358873\backslash \quad (3)$$
$$20424645664157371173623$$

```
> randprime100 := ( ) → nextprime(rando( ));
```

$$randprime100 := ( ) \mapsto nextprime(rando( )) \quad (4)$$

```
> p := randprime100( ) ;
  q := randprime100( );
  n := p·q;
```

$$p := 48661708531942715641275970037658480505701201717887090032847225983343477856\backslash$$
$$60335224098531972647085107$$

$$q := 90354613888655766380635935700849448864556729392617410527384291278174278857\backslash$$
$$19851845425551383904169539$$

$$n := 43968098855659900987812214951285303434450758632031627777162332999583467674\backslash \quad (5)$$
$$74129294996972095468321372927173295436921203552986083387541364976862948058\backslash$$
$$58406891179055498164237625342889974876285318899556 73$$

```
> phiN := (p − 1)·(q − 1);
```

$$phiN := 43968098855659900987812214951285303434450758632031627777162332999583467674\backslash \quad (6)$$
$$74129294996972095468321371537010071230936383333867026002462071274283636953\backslash$$
$$53906330947538236664648091154101927963545175338701028$$

We also need a random exponent, but it doesn't need to be prime, only relatively prime to phi(n). So,

let's generate a random number, and check if it is relatively prime to phiN. If not, try again.

```
> a := rand( ); # lets not bother with BBS.
```

$$a := 730616292946 \quad (7)$$

```
> gcd(a, phiN)
```

$$2 \quad (8)$$

```
> a := rand( ); gcd(a, phiN)
```

$$a := 106507053657$$

$$1 \quad (9)$$

Note that we could do the reverse: Pick a, and then if it isn't relatively prime to phi(n), pick a new p,q pair. Either way is fine.

Now let's compute the decryption exponent.

```
> d := 1/a mod phiN
```

$$d := \quad (10)$$

$$20493617153027644254444389956710774029634734307497638232949804470658282005\backslash$$
$$49450012364307850605733514083757822189268212045583400811833991032383297960\backslash$$
$$94447336340225371769652788981571986037406526744232 05$$

Let's just confirm it works with a test case.

Since our numbers are big (maybe hundreds of digits), it is crazy inefficient to compute x^a and then reduce **mod** n. Instead, we reduce mod n as we calculate the powers. Maple knows how to do this, but we have to give it a hint that we want it to do this way, using the "inert form" of exponentiation &^ instead of just ^. This can't be entered in the regular "math mode" input, so we have to use "maple mode" (control-M)

```
> crypt:= 123456789 &^a mod n;
  decrypt:= crypt &^d mod n;
```

$$crypt :=$$

$$13645882161342979184244363788691730876990523021759679035917121219450789063\backslash$$
$$96460207265896181271943462226212077521445402490860775809299750962759496918\backslash$$
$$83483822574039551432733782850269906683585951130788 69$$

$$decrypt := 123456789 \quad (11)$$

It seems to work.

Now let's write some proc to give us a public and private key.
We can allow specifying the number of digits for the primes and the encryption key, with a default value if omitted.

```
> MakeRSAkeys:=proc(primesize::posint:=100, keysize::posint:=15)
    with(RandomTools):
    local p,q,rando1,rando2, n, a, d, phiN;
    rando1 := BlumBlumShub:-NewGenerator(range = 10^(primesize-1) .
  . 10^primesize);
    rando2 := BlumBlumShub:-NewGenerator(range = 10^(keysize-1) ..
  10^keysize);
    p:= nextprime(rando1()); # generate primes
    q:= nextprime(rando1());
    n:=p*q;
    phiN:=(p-1)*(q-1);  # or lcm(p-1,q-1) is OK, too
    a:=rando2();
```

```
    # make sure a is ok; if not, keep trying
    while gcd(a,phiN)> 1 do
      WARNING("Tried %1 for a, no good.",a);  # don't need this,
  but why not?
       a:=rando2();
    od;
    d:=1/a mod phiN;
    return( [n,a], [n,d]);
  end:
```

```
> pub, priv := MakeRSAkeys( ) :
```

Warning, Tried 660852372526504 for a, no good.

to encrypt:
1) convert your message to numbers less than n.
2) compute $x \rightarrow x^a \bmod n$ on each number.

```
> n := pub[1]; a := pub[2]; d := priv[2];
```

$$n := 1298561429244713161262956868286628794496246094001566777332158129779986144580669003401361284041469650031446916224232930130890832233969673496088256848789326043465396845484618226340828355606689481384788178932604346539684548461822634082835560668948138478811$$

$$a := 369390885449921$$

$$d := 3462787497219153009624576233659376064075031093770057857631307077188045037362287299275001249660811571501217639278554417816179207202387974178459344138276668917735041107503004528186867480971841533081877 \quad (12)$$

If we are using base n for our message, we can convert our string to blocks of rather large numbers. We can fit about 100 characters into a single numeric block.

```
> nums :=
      StringToList("Who put the benzedrine in Mrs. Murphy's ovaltine?  Sure was a shame, don't
      know who's to blame, she didn't even get his name.", base=n − 1)
```

$$nums := [1242760253243280208438584507732259227323217419604804005273409625237738430205776728149853698404244106293070289914347328170622431599372009947104283693826712445763171306069585143596389166182376071195233019616091076100168748578310337084732247756126128319616091076100168748578310337084732247756126128319$$

Have to tell maple to compute powers in modular arithmetic (ie, reducing as you go) rather than power first, or you'll get an overflow.

```
> crypt:= map(x-> x &^ a mod n, nums)
```

$$crypt := [7301700704718875825485407408585412555545988627324534798687483594911016283834105900103635020223260426162342678260017490606114924462781623467281624064964242920309907322569892098824436977376492047462933,3242151165644249139319638083314346947025342734422062668746332714919045118332421511656442491393196380833143469470253427344220626687463327149190451183$$

```
    10404229881457499539591504013666245155041664165206486434480106262335425110\
    3636661236762199421574150322731252765670091230213131
```

```
> decrypt:= map(x-> x &^ d mod n, crypt)
```

decrypt :=

```
    [
    1242760253243280208438584507732259227323217419604804005273409625237738430 2\
    0577672814985369840424410629307028991434732817062243159937200994710428369 3\
    8267124457631713060695851435963891661823760711952330,
    1961609107610016874857831033708473224775612612831 ]
```

(15)

```
> ListToString(%, base=n-1)
```

"Who put the benzedrine in Mrs. Murphy's ovaltine? Sure was a shame, don't know who's to blame, she didn't even get his name." (16)

```
> doRSA:=proc(numlist::list, keypair::list)
    local n,a;
    a:=keypair[2];
    n:=keypair[1];
    return(map(x-> x &^ a mod n, numlist));
  end:
```

We can represent our string in any base we want, as long as it is less than n. Here n is about 10^200.

```
> evalf(log[10](n));
```

199.1134625 (17)

Ok, so it is a bit bigger than 10^199.

```
> cry:=
  doRSA(StringToList("This is a test. This is only a test. If
  this were a real emergency, you would be dead by now so who
  cares?", base=10^50), pub):
> ListToString(cry,base=10^50)
```

"3{I_G$#O i6_NKGj0@[6'BXWXsg 0j]ga*!=V.7\,xrO1.R)F||j#p5! ! XANkR&uu9bx|FDezk7D})L\~h"y! nCD{*VFwlC 2v@zPqO -eUjO q_ypHMm0b$kC a3ykTC m2p$(C)psl,gZ7OA/h65-SLHah9sF#sz_gN.Q_an|po:iUt7Es! @H"2`NNdI! R8-kSC v4ZAOO/G0" (18)

```
> cry
```

[(19)

```
    9079451339774785302172870726903083161184336549282106147608597852130888067 4\
    4366129126422240655943004099127172107488087007520152430359451784817386233 7\
    6303539025547125017784317387993085915341162418614 39,
    9023789531789650956404836530354059193119957526907160376520492377132671959 3\
    3611670852262899884844470664885339765396288617007260984226094196560071993 1\
    4479031772396763016264326070423299529039765625494 50,
    5397948110922651914914239513133359843305012825402903194569485931045419048 2\
    7404169016682058148827927692450315301585399770732908153617954705456414852 7\
    7749757537512642271048633325012864499310612208222 20,
    1040576592512783495850479713211901081156373881159362678128968125080998182 3\
    6851445894225346452394670389988527624419534831025339193674947801373583477 1\
```

```
    859563467951759092851754561113747064952282705259891 4,
    546528257951781154511428463927950173100205676898036597427004140114492937 99\
    525432975047406365521483848920738020569993121782790453414053602978820752 53\
    878265141091405700792502854072538004470115664740304 ]
```

> $ListToString(doRSA(cry, priv), base = 10^{50})$ # *decrypting the message*

"This is a test. This is only a test. If this were a real emergency, you would be dead by now so who cares?" **(20)**

Let's try again with a base of 10^199. Still should work.

```
> cry2:=
  doRSA(StringToList("This is a test.  This is only a test.  If
  this were a real emergency, you would be dead by now so who
  cares?", base=10^199), pub):

  ListToString(doRSA(cry2,priv), base=10^199)
```

"This is a test. This is only a test. If this were a real emergency, you would be dead by now so who cares?" **(21)**

But this will break if we use too big of a base (remember, 10^199 < n < 10^200) :

```
> cry3:= doRSA(
    StringToList("This is a test.  This is only a test.  If this
  were a real emergency, you would be dead by now so who cares?",
  base=10^200),
    pub):

  ListToString( doRSA(cry3,priv), base=10^200)
```

"-GtGVLru/DQ]o%JRl7qt^?RJNp|*-9.Ru:uY3U/!s\#82~UdHr4]~:j~|[C}8#jrvQrGC]4.K] eH~pdwqa|DuI!~02_@?'C RFS! #w~bares?" **(22)**

One other thing discussed here is that RSA can be used to digitally sign a message.

That is, assume my public key is (n,a) and my private key is (n,d). My public key needs to be obtainable publically, for example, on my web page.

Suppose I want to send you a message (not encrypted, in the clear), but signed so that you know it was really from me and not forged. I could, of course, just encrypt the whole message with my private key, and anyone (including you) could decrypt it. Since I am the only person who can encrypt that message (since I am the only one who knows d), you know it came from me. But maybe I want the message to be readable by anyone, and those that want to know wasn't forged can decrypt a shorter message to check.

This leads to the idea of a cryptographic hash function (most common are MD5 and SHA ... MD5 is quite common, but was cracked in about 2010; SHA is its replacement)..... the has function gives a string that changes if even one bit of its input is changed, and figuring out two messages with the same hash is believed to be computationally impossible. (if we don't specify the method, we get MD5)

```
> Hash("this and that",method=sha1)
```

"4f813651453b11db74cbc36ccddea13f109e8412" **(23)**

```
> Hash("This and that",method=sha1)
```

"e4edcc5d05086e6a667096fcb9d19681ddf0d773" **(24)**

So, to sign a message, I can compute the hash of the message and RSA encrypt that with my private key, then include that with the message as a digital signature.
If you want to check that I sent it and it wasn't tampered with, you can use my public key to decrypt the signature, and compare it to the hash of the message. If the two agree, I sent it and it wasn't tampered with. If they disagree, it was tampered with or I didn't send it.

Of course, you need to be sure it is MY public key, and not the key of some forger.

OK, that's it for this semester of MAT331. I hope you got something out of it.
Please do your course evaluations, especially the comment section.