11/19/2019 Some 7th grade teacher's 41st birthday.

I'll start saving the routines we write in the file Crypto.mw which is at
http://www.math.stonybrook.edu/~scott/mat331.fall19/daily/extras/
This will get updated as the class progresses.

Here is some stuff I stole from there. I'll be doing that in most of the classes.

First note that there was a change from previous classes in that it will be useful to use 0..length(Alphabet)-1 rather than 1..length(Alphabet) for our numeric representations of text. It will make life easier later.

```
> with(StringTools):
> StringToList:=proc(str::string)
    global Alphabet;
    return(map( s->SearchText(s,Alphabet)-1, Explode(str)));
  end:
> ListToString:=proc(nums::list)
    global Alphabet;
    return(Implode(map(k->Alphabet[k+1],nums)) );
  end:
> Caesar:= proc(msg::string, shift::integer, {decrypt::truefalse := false})
    local numlist, alen, shifted;
    global Alphabet;
    if (decrypt) then
      return(Caesar(msg,-shift));
    fi:
    alen:=length(Alphabet);
    numlist:=StringToList(msg);
    shifted:= map(x-> modp(x+ shift,alen), numlist);
    return(ListToString(shifted));
  end:
```

OK, so let's set up our Alphabet. a..z will do for now.

```
> Alphabet := "abcdefghijklmnopqrstuvwxyz";
```

$$Alphabet := \text{"abcdefghijklmnopqrstuvwxyz"} \quad \textbf{(1)}$$

One thing to notice is that I added a "decrypt" option to all the routines. This means we can either decrypt a message by using the inverse of the encryption key, or by telling it to decrypt with the encryption key. That way, we don't need to calculate the decryption key in order to decrypt.

```
> crypt := Caesar("salad", 3)
```

$$crypt := \text{"vdodg"} \quad \textbf{(2)}$$

```
> Caesar(%,-3)
```

$$\text{"salad"} \quad \textbf{(3)}$$

```
> Caesar(crypt, 3, decrypt)
```

$$\text{"salad"} \quad \textbf{(4)}$$

Now we extend from doing addition to doing multiplication. But some care is needed:

```
> L := [$0..25]
```

(5)

$L := [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25]$ **(5)**

```
> a := 3;
```

$$a := 3 \quad \textbf{(6)}$$

```
> map(x->a*x mod 26, L)
```

$[0, 3, 6, 9, 12, 15, 18, 21, 24, 1, 4, 7, 10, 13, 16, 19, 22, 25, 2, 5, 8, 11, 14, 17, 20, 23]$ **(7)**

```
> a := 2;
```

$$a := 2 \quad \textbf{(8)}$$

```
> map(x->a*x mod 26, L)
```

$[0, 2, 4, 6, 8, 10, 12, 14, 16, 18, 20, 22, 24, 0, 2, 4, 6, 8, 10, 12, 14, 16, 18, 20, 22, 24]$ **(9)**

Note that in multiplying by 3, every number is represented, but in multiplying by 2, we lose half the numbers and get repeats.
This is because 2 and 26 share a factor, but 3 and 26 are relatively prime.

One way to represent the multiplicative inverse is by dividing mod n.

```
> 1/3 mod 26
```

$$9 \quad \textbf{(10)}$$

```
> 1/2 mod 26
```

Error, the modular inverse does not exist

So now let's write an affine encryption. This will be in stages, as usual The first pass will have issues.

```
> Affine:=proc(msg::string,a::integer, b::integer,
  {decrypt::truefalse:=false})
    global Alphabet;
    local alen, numlist;
    alen:=length(Alphabet);
     #
    numlist:=StringToList(msg);
    numlist:=map(x->a*x+b mod alen, numlist);
    return(ListToString(numlist));
  end:
> crypt:=Affine("affinity",3,1)
```

$$crypt := \text{"bqqzozgv"} \quad \textbf{(11)}$$

The decrypting key for [3,1] is [9,-9] (or [9, 17] if you prefer, since -9 mod 26 = 17).

```
> Affine(crypt,9,-9)
```

$$\text{"affinity"} \quad \textbf{(12)}$$

We won't be able to decrypt if we use 2.

```
> Affine("abcznop",2,0)
```

"aceyace" **(13)**

Note that "abc" and "nop" both encrypt to "ace", so we won't be able to decrypt that.

```
> gcd(2,26)
```

2 **(14)**

```
> gcd(3,26)
```

1 **(15)**

So let's change it so that we can't encrypt with a number we can't decrypt from.

```
> Affine:=proc(msg::string,a::integer, b::integer,
  {decrypt::truefalse:=false})
    global Alphabet;
    local alen, numlist;
    alen:=length(Alphabet);
     if (gcd(a,alen)<>1) then
       error(sprintf("%d is not coprime to alphabet length (%d), so
  just go die",a,alen));
     fi;
    numlist:=StringToList(msg);
    numlist:=map(x->a*x+b mod alen, numlist);
    return(ListToString(numlist));
  end:
> Affine("iamdumb",2,0)
```

Error, (in Affine) 2 is not coprime to alphabet length (26), so just go die

```
> Affine("thisisbetter",15,7)
```

"gixrxrwpggpc" **(16)**

Now let's make decrypting work

Observe that if $y = a \cdot x + b \bmod N$ then as long as a and N are coprime, we can solve for x as $x = \frac{y-b}{a} \bmod N = \frac{1}{a} y - \frac{b}{a} \bmod N$

```
> Affine:=proc(msg::string,a::integer, b::integer,
  {decrypt::truefalse:=false})
    global Alphabet;
    local alen, numlist;
    alen:=length(Alphabet);
    if (gcd(a,alen)<>1) then
      error(sprintf("%d is not coprime to %d, can not encode, go
  die",a,alen));
     fi;
    if (decrypt) then
      return(Affine(msg, modp(1/a,alen), modp(-b/a, alen)));
     fi;
    numlist:=StringToList(msg);
    numlist:=map(x->a*x+b mod alen, numlist);
    return(ListToString(numlist));
  end:
```

```
> crypt:=Affine("thisisatest",17,9)
```

$$crypt := \text{"uypdpdjuzdu"} \quad (17)$$

```
> Affine(crypt,17,9,decrypt)
```

$$\text{"thisisatest"} \quad (18)$$

We can do this with vectors and matrices, too. Let's do that next time, then move on to bigger alphabets (eg, digraphs, k-graphs).