2019-11-21: I neglected to save the worksheet from this class, so I will try to recreate it from memory. Sorry about that.

The primary goal for today is to modify (some of) the routines from Crypto.mw so that they can deal with multi-letter alphabets.
That is, treat our string in bigger chunks of letters (that is, pairs of letters, triples of letters, etc.) rather than just single letters at a time.
While this is not so useful for a Caesar cipher, it makes frequency analysis (looking for common letters) much more difficult. That is, while it is easy to look for common single letters (such as e, t, o, etc.), there are so many more common triples or 4-tuples of letters that this is much harder. Also, it will be useful later.

Basic setup...

```
> with(StringTools):
> Alphabet := cat(" .",Select(IsAlpha, convert([seq(i,i=1..127)],
  bytes))):
  length(Alphabet);
```

$$Alphabet := \text{" .ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz"}$$

$$54 \quad (1)$$

Now let's copy some things from Crypto.mw that we will be revising.

```
> StringToList:=proc(str::string)
    global Alphabet;
    return(map( s->SearchText(s,Alphabet)-1, Explode(str)));
  end:
> ListToString:=proc(nums::list(nonnegint))
    global Alphabet;
    return(Implode(map(k->Alphabet[k+1],nums)) );
  end:
```

Now, notice when we use StringToList, we get a list of numbers....

```
> StringToList("This is a string.")
```

$$[21, 35, 36, 46, 0, 36, 46, 0, 28, 0, 46, 47, 45, 36, 41, 34, 1] \quad (2)$$

We want to put the numbers together in some way, that is, group the 21 with the 35, the 36 with the 46, the 0 with the 36, etc. to represent the pair "Th", the pair "is", the pair " i", and so on. We can concatenate them (as in 2135, 3646, etc.) but that will lead to some issues when we do arithmetic with the pairs, since not all pairs correspond to letter pairs (for example, in the current 54-letter alphabet, 2*2135 = 4270, but while 42 corresponds to "o", 70 is not the code for a letter.

So, a little aside about bases...

# About representing numbers in various bases

When we write the number 2043 we mean $3 + 4 \cdot 10 + 0 \cdot 10^2 + 2 \cdot 10^3$. But of course we could use other bases. For example, in base 2, we would represent, say, 11 as a sum of powers of 2. We see that $11 = 1 + 1 \cdot 2 + 0 \cdot 2^2 + 2 \cdot 2^3$, so we say that 11 (base 10) is written as 1011 (base 2).

Base 8 is called octal, and $11 = 3 + 1 \cdot 8$, so in octal, we have 11 (base 10) is 13 (base 8). Or using powers of 3, we have $11 = 2 + 0 \cdot 3 + 1 \cdot 3^2$, so we say that 11 (base 10) is 102 (base 3).
Of course, Maple can do this for us.

```
> convert(11,binary)
```

$$1011 \quad (1.1)$$

```
> convert(11,octal)
```

$$13 \quad (1.2)$$

The bases 2 (binary), 8 (octal), 10 (decimal), and 16 (hexadecimal or hex) have special names. (To write hexadecimal, we use the numbers 0-9, A, B, C, D, E, and F; where A=10, B=11, C=12, D=13, E=14, and F=15, so 11 is written as B in hexadecimal, and 26 is 16+10 written as 1A (base 16).

```
> convert(26,hex)
```

$$1A \quad (1.3)$$

This representation goes by the name "big-endian" because we write the most significant (or largest) numbers first. In some contexts (eg, computer memory), numbers are written little-endian, with the smallest digits first.

For arbitrary bases, we can represent the "digits" as a sequence of numbers between 0 and one less than the base. Maple writes the least significant digits first (that is, little-endian). For example, writing 11 base 3 would be [2,0,1] or [3,1] in base 8. We can represent 26 in base 16 as [10,1] and in base 10 as [6,2].

```
> convert(11,base,3)
```

$$[2, 0, 1] \quad (1.4)$$

```
> convert(11,base,8)
```

$$[3, 1] \quad (1.5)$$

```
> convert(26,base,16)
```

$$[10, 1] \quad (1.6)$$

```
> convert(26,base,10)
```

$$[6, 2] \quad (1.7)$$

We can convert between bases as follows:

```
> convert([2,0,1],base,3,8)
```

$$[3, 1] \quad (1.8)$$

That is, the arguments are a list represented in the first base, the word 'base', the first base, and the base to convert it to.

As another example, suppose we represent the number 123456 in base 10, and then in base 100.

```
> num:=convert(123456,base,10)
```

$$num := [6, 5, 4, 3, 2, 1] \quad (1.9)$$

```
> convert(num,base,10,10^2)
```

$$[56, 34, 12] \quad (1.10)$$

Now, in base 1000

```
> convert(num,base,10,10^3)
```

$$[456, 123] \quad \textbf{(1.11)}$$

That is, writing the number in powers of the original base groups them together into blocks of the size of the power.

With that discussion in hand, we return to encryption related discussions.

When we convert our strings to a list, we can view our string as a number written in the base that is the length of the alphabet.
So, for example, if we are using the 26-letter alphabet consisting of the letters a-z, with a=0, z=25, we can think of it as a number in base 26.

```
> Alphabet:="abcdefghijklmnopqrstuvwxyz"; alen:=length(Alphabet);
```

$$Alphabet := \text{"abcdefghijklmnopqrstuvwxyz"}$$

$$alen := 26 \quad \textbf{(3)}$$

```
> zip1:=StringToList("zippity");
```

$$zip1 := [25, 8, 15, 15, 8, 19, 24] \quad \textbf{(4)}$$

```
> convert(zip1,base,26,26^2)
```

$$[233, 405, 502, 24] \quad \textbf{(5)}$$

Observe that 25+8*26=233, 15+15*26=405, 502=8+19*26, and 24=24+0*26.
Hence we can convert our message to blocks of 2 letters at a time by converting from base 26 to base $26^2$.

Now let's turn to modifying StringToList and ListToString (as well as other routines) so that they can use lists with different bases (that is, multi-byte blocks of letters instead of single letters). I want to do this in such a way that it will still work as before if one doesn't ask for a different base.

So let's start modifying StringToList. First, let's add an optional keyword parameter called base, which defaults to being the length of the Alphabet if it isn't specified. Since we can't actually do a computation in this way, we'll set it to 1 and if the base is 1, then that means use the length of the Alphabet.
Also, we will break this up from essentially a one-line procedure to multiline. Changes are marked in green. For now, let's just make a note that we will use a different base.

```
> StringToList:=proc(str::string, {base::posint:=1})
    local numlist;
   global Alphabet;
    numlist:=map( s->SearchText(s,Alphabet)-1, Explode(str));
    if (base>1) then
      printf("the base is now %d, so there is work to do but I
  didn't do it yet.\n",base);
    fi;
   return(numlist) ;
  end:
```

Let's confirm that it still works as before, and that the switch tells us something.

```
> numlist:=StringToList("zippity");
```

$$numlist := [25, 8, 15, 15, 8, 19, 24] \quad (6)$$

```
> StringToList("zippity",base=alen^2);
the base is now 676, so there is work to do but I didn't do it
yet.
```

$$[25, 8, 15, 15, 8, 19, 24] \quad (7)$$

Now to discus what should be done. The idea is to just convert numlist to the new base, using convert, as in

```
> convert(numlist, base, alen, alen^2)
```

$$[233, 405, 502, 24] \quad (8)$$

Let's do that, but it isn't going to work....

```
> StringToList:=proc(str::string, {base::posint:=1})
    local numlist;
   global Alphabet;
    numlist:=map( s->SearchText(s,Alphabet)-1, Explode(str));
    if (base>1) then
       numlist:=convert(numlist, base, length(Alphabet), base);
    fi;
   return(numlist) ;
  end:
> StringToList("zippity",base=alen^2)
Error, (in StringToList) invalid input: convert expects its 2nd
argument, form, to be of type name, but received 676
```

This failed because the first base in the call to convert has the value 676, so in effect we are trying to issue the command

```
> convert(numlist,676,alen,676);
Error, invalid input: convert expects its 2nd argument, form,
to be of type name, but received 676
```

which naturally doesn't make any sense. We have to "protect" the word base in some way. An easy fix is to just call it something else, as in

```
> StringToList:=proc(str::string, {bas::posint:=1})
    local numlist;
    global Alphabet;
    numlist:=map( s->SearchText(s,Alphabet)-1, Explode(str));
    if (bas>1) then
       numlist:=convert(numlist, base, length(Alphabet), bas) ;
    fi;
   return(numlist);
  end:
```

This will work, but I really want to call the parameter base.

```
> StringToList("zippity",bas=alen^2);
```

$$[233, 405, 502, 24] \quad (9)$$

We can try surronding the word base by single quotes (that works in some contexts, but not here), but that doesn't quite do the trick either.

```
> StringToList:=proc(str::string, {base::posint:=1})
```

```
    local numlist;
    global Alphabet;
    numlist:=map( s->SearchText(s,Alphabet)-1, Explode(str));
    if (base>1) then
        numlist:=convert(numlist,'base', length(Alphabet), base);
     fi;
    return(numlist);
  end:
> StringToList("zippity",base=alen^2);
```

Error, (in StringToList) invalid input: convert expects its 2nd argument, form, to be of type name, but received 676

It is not obvious, but we have to write it as ':-base'. This is described in the "Keyword Matching" section of the help page on argument processing.

```
> StringToList:=proc(str::string, {base::posint:=1})
    local numlist;
    global Alphabet;
    numlist:=map( s->SearchText(s,Alphabet)-1, Explode(str));
    if (base>1) then
        numlist:=convert(numlist,':-base', length(Alphabet), base);
     fi;
    return(numlist);
  end:
```

Now it works:

```
> StringToList("zippity",base=alen^2);
```

$$[233, 405, 502, 24] \quad (10)$$

Now let's adjust ListToString to undo this.

```
> 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:
```

Let's give it a try:

```
> Alphabet := cat(" .",Select(IsAlpha, convert([seq(i,i=1..127)],
  bytes))): length(Alphabet);
```

$$54 \quad (11)$$

```
> numlist:= StringToList("I like big base and I can not lie.",
  base=length(Alphabet)^4)
```

$$numlist := [5782438, 4568222, 4568328, 95824, 92638, 4496482, 6733085, 5782475, 86] \quad (12)$$

```
> ListToString(numlist,base=length(Alphabet)^4)
```

"I like big base and I can not lie." (13)

Note that although a base which is the $n^{th}$ power of the length of the Alphabet corresponds to taking letters in blocks of size $n$, we can, in fact, use any integer base as long as we are consistent. For example:

```
> numlist:= StringToList("I like big base and I can not lie.",
  base=4096)
```

$$numlist := [3974, 3622, 3538, 3906, 1120, 1491, 1001, 3465, 2285, 2548, 1817, 2244, 2198, 2755, 2794, 1545] \quad (14)$$

```
> ListToString(numlist,base=4096);
```

"I like big base and I can not lie." (15)

To finish, let's adjust the Affine cipher to deal with the bigger bases. Similar changes work for the other ciphers as well, but are less useful.

```
> Affine:=proc(msg::string,a::integer, b::integer:=0,
  {decrypt::truefalse:=false}, {base::posint:=1})
    global Alphabet;
    local modbase, numlist;
    if (base>1) then
      modbase:=base;
    else
      modbase:=length(Alphabet);
    fi;
    if (gcd(a, modbase)<>1) then
      error(sprintf("%d is not coprime to the modulus base (%d);
  cannot decrypt",a, modbase) );
    fi;
    if (decrypt) then
      return(Affine(msg, modp(1/a,modbase), modp(-b/a, modbase),
  ':-base'=modbase) );
    fi;
   numlist:=StringToList(msg, ':-base'=modbase) ;
   numlist:=map(x->a*x+b mod modbase, numlist);
   return(ListToString(numlist, ':-base'=modbase) );
  end:
```

Let's give this a try. First, for the original affine encryption, observe that a repeated letter always has the same encryption, so "oo"->"tt", "kk"->"dd", and "ee"->"FF".

```
> crypto:=Affine("bookkeeper",31,41);
  Affine(crypto,31,41,decrypt);
```

$$crypto := \text{"UttddFFWFe"}$$

"bookkeeper" (16)

Using a different base, this is unlikely to be true (since pairs or triples of double letters are unlikely to align on block boundaries).

```
> crypto:=Affine("bookkeeper",31,41,base=54^3);
  Affine(crypto,31,41,base=54^3,decrypt);
```

$$crypto := \text{"UVcdKoFAreY"}$$

(17)

Maybe I managed to (nearly) recreate the class. At least this is (roughly) what I remember doing. Soon I will update Crypto.mw to include these changes.