16 April 2024

The stuff we did before is taken from Crypto.mw in the daily/extras directory. As mentioned earlier, I will update this as we go along in the class (and clean things up after class).

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

Alphabet Setup

Let's define some default Alphabet by selecting all printable characters from the ASCII sequence. Probably you will change this depending on various things.

```
> # Alphabet := Select(IsPrintable, convert([seq(i,i=1..127)],
  bytes)):
  # Alphabet := Select(IsUpper, convert([seq(i,i=1..127)], bytes)):
  Alphabet := cat(Select(IsAlpha, convert([seq(i,i=1..127)], bytes)
  )," .");
  printf("Our %d-character Alphabet is \n%s\n",length(Alphabet),
  Alphabet);
```

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

```
Our 54-character Alphabet is
ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz .
```

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 differ slightly from what we did in class, in that Alphabet[n] is represented by n-1. This will be more convenient when doing arithmetic.

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

Remind me what numlist::list(nonnegint) means
It enforces the type of the argument.

```
> ListToString([1, 2, 3, cat, 0])
Error, invalid input: ListToString expects its 1st argument,
numlist, to be of type list(nonnegint), but received [1, 2, 3, cat,
0]
> ListToString([1,2,3,5,0]);
```

$$\text{"BCDFA"} \qquad \textbf{(2.1)}$$

Caesar (shift) cipher.

To encrypt with a shift of x, use `Caesar(plaintext, x);`
Decrypt with `Caesar(crypttext, -x)` or `Caesar(crypttext, x, decrypt)`

This has been adjusted to account for the adjustment in the indexing of the Alphabet string, and to add the `decrypt` keyword.

```
> Caesar:= proc(msg::string, shift::integer, {decrypt::truefalse :=
  false})
    global Alphabet;
    local Alen:=length(Alphabet);
    local shifted;
    if (decrypt) then
       return(Caesar(msg,-shift));
    fi:
    shifted:= map(x-> modp(x+shift,Alen), StringToList(msg));
    return(ListToString(shifted));
  end:
```

```
> Caesar("Et tu, Brute?  12345", 5)
```

"JyDyzEDGwzyjEDDEEEEE" **(3.1)**

```
> Caesar(%,-5);
```

"Et tu. Brute. ....." **(3.2)**

```
> crypt := Caesar("Et tu Brute", 5)
```

crypt := "JyDyzDGwzyj" **(3.3)**

```
> Caesar(crypt,-5);
```

"Et tu Brute" **(3.4)**

Putting a function parameter in a curly bracket means it is a named parameter, which is optional.

```
> Caesar(crypt, 5, decrypt)
```

"Et tu Brute" **(3.5)**

What do you mean by \" ? Answer: it is a way to have the character " within a string, since string values are delimited by double-quote marks.
We can also put other special characters in there, for example \n is a a newline:

```
> StringContainingAQuote:="Baby Sally said \"uh-oh!\" \nThen she fell
  down."
```

StringContainingAQuote := "Baby Sally said "uh-oh!"
Then she fell down." **(1)**

```
>  StringContainingAQuote[15..20]
```

"d "uh-" **(2)**

If we want to "build a list from nothing", we can start with an empty list, then add things to it. This isn't optimal, but it works.

```
> mylist:=[];
```

mylist := [ ] **(3)**

```
> mylist;
```

[ ] **(4)**

```
> mylist:=[op(mylist), 1];
```

$$mylist := [1] \quad (5)$$

```
> mylist:=[op(mylist), 2];
```

$$mylist := [1, 2] \quad (6)$$

OK, so let's start building the Vignere routine.

```
> Vignere:=proc(msg::string, key::string)
    global Alphabet;
    local Alen:=length(Alphabet);
    local numlist:=StringToList(msg);
    local keynums:=StringToList(key);
    local shifted, i;

    shifted:=[];
    print(numlist, keynums);
    for i from 1 to length(msg) do
      shifted:=[op(shifted), modp(numlist[i]+keynums[i], Alen)];
      print(i, shifted);
    od;
  end:
```

This doesn't work:

```
> Vignere("Some stuff", "zyx");
```

$$[18, 40, 38, 30, 52, 44, 45, 46, 31, 31], [51, 50, 49]$$

$$1, [15]$$

$$2, [15, 36]$$

$$3, [15, 36, 33]$$

Error, (in Vignere) invalid subscript selector

Let's try to fix it

```
> Vignere:=proc(msg::string, key::string)
    global Alphabet;
    local Alen:=length(Alphabet);
    local numlist:=StringToList(msg);
    local keynums:=StringToList(key);
    local shifted, i, keylen:=length(key);

    shifted:=[];
    print(numlist, keynums);
    for i from 1 to length(msg) do
      shifted:=[op(shifted), modp(numlist[i]+keynums[modp(i,keylen)],
  Alen)];
      print(i, shifted);
    od;
  end:
> Vignere("Some stuff", "zyx");
```

$$[18, 40, 38, 30, 52, 44, 45, 46, 31, 31], [51, 50, 49]$$

1, [15]

2, [15, 36]

Error, (in Vignere) invalid subscript selector

Problem is that 3 mod 3 is 0. We can fake it by shifting things, applying mod, then shifting back.

```
> (-1 mod 3) +1
```

3 **(7)**

```
> ((3-1) mod 3)+1
```

3 **(8)**

```
> Vignere:=proc(msg::string, key::string)
    global Alphabet;
    local Alen:=length(Alphabet);
    local numlist:=StringToList(msg);
    local keynums:=StringToList(key);
    local shifted, i, keylen:=length(key);

    shifted:=[];
    print(numlist, keynums);
    for i from 1 to length(msg) do
      shifted:=[op(shifted), modp(numlist[i]+keynums[modp(i-1,keylen)
  +1], Alen)];
      print(i, shifted);
    od;
  end:
```

```
> Vignere("Some stuff", "zyx");
```

[18, 40, 38, 30, 52, 44, 45, 46, 31, 31], [51, 50, 49]

1, [15]

2, [15, 36]

3, [15, 36, 33]

4, [15, 36, 33, 27]

5, [15, 36, 33, 27, 48]

6, [15, 36, 33, 27, 48, 39]

7, [15, 36, 33, 27, 48, 39, 42]

8, [15, 36, 33, 27, 48, 39, 42, 42]

9, [15, 36, 33, 27, 48, 39, 42, 42, 26]

10, [15, 36, 33, 27, 48, 39, 42, 42, 26, 28] **(9)**

Looks like it works.

```
> Vignere:=proc(msg::string, key::string)
    global Alphabet;
    local Alen:=length(Alphabet);
    local numlist:=StringToList(msg);
    local keynums:=StringToList(key);
    local shifted, i, keylen:=length(key);
```

```
    shifted:=[];
    # print(numlist, keynums);
    for i from 1 to length(msg) do
      shifted:=[op(shifted), modp(numlist[i]+keynums[modp(i-1,keylen)
  +1], Alen)];
      # print(i, shifted);
    od;
    return(ListToString(shifted));
  end:
> Vignere("Hello there", "cat");
                                "jCcLMrTFVRC"                          (10)
```

It takes some effort to figure out with the decrypting key for "cat" is... note that it is just the negative of the key; let's be lazy and just write that since time is going fast.

```
> unVignere:=proc(msg::string, key::string)
    global Alphabet;
    local Alen:=length(Alphabet);
    local numlist:=StringToList(msg);
    local keynums:=StringToList(key);
    local shifted, i, keylen:=length(key);

    shifted:=[];
    # print(numlist, keynums);
    for i from 1 to length(msg) do
      shifted:=[op(shifted), modp(numlist[i]-keynums[modp(i-1,keylen)
  +1], Alen)];
      # print(i, shifted);
    od;
    return(ListToString(shifted));
  end:
> unVignere("jCcLMrTFVRC","cat");
                                "Hello there"                          (11)
```

I will put these together into one in the crytpo.mw file. I can clean this up all nice.