Initial setup conversion from text to/from various list formats.

> $StringToVects("Helloooo", 3);$

$$\left[\begin{bmatrix}40\\69\\76\end{bmatrix},\begin{bmatrix}76\\79\\79\end{bmatrix},\begin{bmatrix}79\\79\\94\end{bmatrix}\right] \quad (1)$$

> $v := \langle 1, 2, 7\rangle;$

$$v := \begin{bmatrix}1\\2\\7\end{bmatrix} \quad (2)$$

> $convert(v, list);$

$$[1, 2, 7] \quad (3)$$

> $op(\%);$

$$1, 2, 7 \quad (4)$$

> $map(v \rightarrow convert(v, list), [\langle 1, 2\rangle, \langle 3, 4\rangle]);$

$$[[1, 2], [3, 4]] \quad (5)$$

> $map(v \rightarrow op(convert(v, list)), [\langle 1, 2\rangle, \langle 3, 4\rangle]);$

$$[1, 2, 3, 4] \quad (6)$$

> $op(v);$

$$3, \{1=1, 2=2, 3=7\}, datatype = anything, storage = rectangular, order = Fortran\_order, shape = [\ ] \quad (7)$$

> $L := StringToVects("Helloooo", 2);$

$$L := \left[\begin{bmatrix}40\\69\end{bmatrix},\begin{bmatrix}76\\76\end{bmatrix},\begin{bmatrix}79\\79\end{bmatrix},\begin{bmatrix}79\\79\end{bmatrix}\right] \quad (8)$$

> $A := Matrix(\langle\langle 1|2\rangle, \langle 3|4\rangle\rangle);$

$$A := \begin{bmatrix}1 & 2\\3 & 4\end{bmatrix} \quad (9)$$

> $p := length(Alphabet);$

$$p := 95 \quad (10)$$

> $A.L[1]$ **mod** $p;$

$$\begin{bmatrix}83\\16\end{bmatrix} \quad (11)$$

> $VectsToString([\%]);$

"s0" (12)

> $A.L[2]$ **mod** $p;$

$$\begin{bmatrix}38\\57\end{bmatrix} \quad (13)$$

> $VectsToString([\%]);$

(14)

"FY" (14)

> $Ainv := MatrixInverse(A)$ **mod** p;

$$Ainv := \begin{bmatrix} 93 & 1 \\ 49 & 47 \end{bmatrix} \quad (15)$$

> $Ainv.\langle 83, 16\rangle$ **mod** p;

$$\begin{bmatrix} 40 \\ 69 \end{bmatrix} \quad (16)$$

> $VectsToString([\%], 2)$;

"He" (17)

> $Dimension(A)$;

2, 2 (18)

Affine Matrix cipher

> $with(LinearAlgebra)$:

```
> AffineMat := proc (plain::string, A::Matrix, b::Vector,
  {decrypt:=false})
   local L, S, p, m,n;
   global Alphabet;
   p := length(Alphabet);
   m,n:=Dimension(A);
  ######### need to check that A is invertable.
  # if (gcd(p, a)>1) then
  #    error (a, " is not relatively prime to length of
  Alphabet", p);
  #  fi;

   L := StringToVects(plain,n);
   if (decrypt) then  ### not done yet
     S:=map(x->(x-b)/a mod p, L); # apply the inverse if
  decrypting
   else
     S := map(x->(A.x+b) mod p, L);
    fi;
   return VectsToString(S,n);
  end:
```

> $AffineMat(\text{"No Change"}, \langle\langle 1, 0\rangle, \langle 0, 1\rangle\rangle, \langle 0, 0\rangle)$;

Error, (in rtable/Sum) invalid arguments

> $debug(AffineMat)$;

AffineMat (2.1)

```
> AffineMat("No Change", <<1,0>,<0,1>>, <0,0>);
{--> enter AffineMat, args = No Change, Matrix(4, 1, {(1, 1) =
1, (2, 1) = 0, (3, 1) = 0, (4, 1) = 1}), Vector(2, {(1) = 0, (2)
= 0})
```

$$p := 95$$

$$m, n := 4, 1$$

$$L := [[46], [79], [0], [35], [72], [65], [78], [71], [69]]$$

```
<-- ERROR in AffineMat (now at top level) = invalid arguments}
```

```
Error, (in rtable/Sum) invalid arguments
```

> $AffineMat(\text{"No Change"}, \langle\langle 1, 0\rangle | \langle 0, 1\rangle\rangle, \langle 0, 0\rangle);$

```
{--> enter AffineMat, args = No Change, Matrix(2, 2, {(1, 1) =
1, (1, 2) = 0, (2, 1) = 0, (2, 2) = 1}), Vector(2, {(1) = 0, (2)
= 0})
```

$$p := 95$$

$$m, n := 2, 2$$

$$L := \left[\begin{bmatrix}46\\79\end{bmatrix}, \begin{bmatrix}0\\35\end{bmatrix}, \begin{bmatrix}72\\65\end{bmatrix}, \begin{bmatrix}78\\71\end{bmatrix}, \begin{bmatrix}69\\94\end{bmatrix}\right]$$

$$S := \left[\begin{bmatrix}46\\79\end{bmatrix}, \begin{bmatrix}0\\35\end{bmatrix}, \begin{bmatrix}72\\65\end{bmatrix}, \begin{bmatrix}78\\71\end{bmatrix}, \begin{bmatrix}69\\94\end{bmatrix}\right]$$

```
<-- exit AffineMat (now at top level) = No Change~}
```

"No Change~" **(19)**

> $AffineMat(\text{"Mix me"}, \langle\langle 0, 1\rangle | \langle 1, 0\rangle\rangle, \langle 0, 0\rangle);$

```
{--> enter AffineMat, args = Mix me, Matrix(2, 2, {(1, 1) = 0,
(1, 2) = 1, (2, 1) = 1, (2, 2) = 0}), Vector(2, {(1) = 0, (2) =
0})
```

$$p := 95$$

$$m, n := 2, 2$$

$$L := \left[\begin{bmatrix}45\\73\end{bmatrix}, \begin{bmatrix}88\\0\end{bmatrix}, \begin{bmatrix}77\\69\end{bmatrix}\right]$$

$$S := \left[\begin{bmatrix}73\\45\end{bmatrix}, \begin{bmatrix}0\\88\end{bmatrix}, \begin{bmatrix}69\\77\end{bmatrix}\right]$$

```
<-- exit AffineMat (now at top level) = iM xem}
```

"iM xem" **(20)**

```
> AffineMat := proc (plain::string, A::Matrix, b::Vector,
  {decrypt:=false})
   local L, S, p, m,n, Ainv;
   global Alphabet;
   p := length(Alphabet);
   m,n:=Dimension(A);
  ######### need to check that A is invertible.
  # if (gcd(p, a)>1) then
  #    error (a, " is not relatively prime to length of Alphabet",
  p);
  #  fi;

   L := StringToVects(plain,n);
   if (decrypt) then
     Ainv := MatrixInverse(A) mod p;
     S:=map(x->Ainv.(x-b) mod p, L); # apply the inverse if
  decrypting
   else
     S := map(x->(A.x+b) mod p, L);
    fi;
   return VectsToString(S,n);
  end:
```

> $crypt := AffineMat(\text{"Mix me"}, \langle\langle 0, 1\rangle | \langle 1, 0\rangle\rangle, \langle 0, 0\rangle);$

$$crypt := \text{"iM xem"} \quad (21)$$

> $AffineMat(crypt, \langle\langle 0, 1\rangle | \langle 1, 0\rangle\rangle, \langle 0, 0\rangle, decrypt);$

$$\text{"Mix me"} \quad (22)$$

> $A := \langle\langle 1, 2, 3, 4\rangle | \langle 1, 0, 1, 0\rangle | \langle 1, 1, 2, 3\rangle | \langle 19, 47, 5, 1\rangle\rangle;$
> $b := \langle 1, 2, 3, 4\rangle;$

$$A := \begin{bmatrix} 1 & 1 & 1 & 19 \\ 2 & 0 & 1 & 47 \\ 3 & 1 & 2 & 5 \\ 4 & 0 & 3 & 1 \end{bmatrix}$$

$$b := \begin{bmatrix} 1 \\ 2 \\ 3 \\ 4 \end{bmatrix} \quad (23)$$

> $crypt := AffineMat(\text{"aaaaaaaaaaaa"}, A, b);$

$$crypt := \text{"\&6UQ\&6UQ\&6UQ"} \quad (24)$$

> $AffineMat(crypt, A, b, decrypt);$

$$\text{"aaaaaaaaaaaa"} \quad (25)$$

> $crypt := AffineMat(\text{"aaaaaaaaaaaa"}, \langle\langle 0, 0\rangle | \langle 1, 0\rangle\rangle, \langle 0, 0\rangle);$

$$crypt := \text{"a a a a a a "} \quad (26)$$

> $AffineMat(crypt, \langle\langle 0, 0\rangle | \langle 1, 0\rangle\rangle, \langle 0, 0\rangle, decrypt);$

Error, (in LinearAlgebra:-MatrixInverse) singular matrix

```
> AffineMat := proc (plain::string, A::Matrix, b::Vector,
  {decrypt:=false})
   local L, S, p, m,n, d, Ainv;
   global Alphabet;
   p := length(Alphabet);
   m,n:=Dimension(A);
  ######### need to check that A is invertible.
   d:=Determinant(A);
   if (d=0) then
      error ("matrix is not invertible");
    fi;

   L := StringToVects(plain,n);
   if (decrypt) then
     Ainv := MatrixInverse(A) mod p;
     S:=map(x->Ainv.(x-b) mod p, L); # apply the inverse if
  decrypting
   else
     S := map(x->(A.x+b) mod p, L);
    fi;
   return VectsToString(S,n);
  end:
```

> $crypt := AffineMat(\text{"aaaaaaaaaaaa"}, \langle\langle 0, 0\rangle | \langle 1, 0\rangle\rangle, \langle 0, 0\rangle);$

Error, (in AffineMat) matrix is not invertible

> $crypt := AffineMat(\text{"aaaaaaaaaaaa"}, \langle\langle 0, 5\rangle | \langle 1, 0\rangle\rangle, \langle 0, 0\rangle);$

$$crypt := \text{"aHaHaHaHaHaH"} \tag{27}$$

```
> AffineMat(crypt, <<0, 5>|<1, 0>>, <0, 0>, decrypt);
Error, (in Matrix) the modular inverse does not exist
```

```
> AffineMat := proc (plain::string, A::Matrix, b::Vector, 
  {decrypt:=false})
   local L, S, p, m,n, d, Ainv;
   global Alphabet;
   p := length(Alphabet);
   m,n:=Dimension(A);
  ######### need to check that A is invertible.
   d:=Determinant(A);
   if (gcd(d,p)<>1) then
      error ("matrix is not invertible mod",p);
    fi;

   L := StringToVects(plain,n);
   if (decrypt) then
     Ainv := MatrixInverse(A) mod p;
     S:=map(x->Ainv.(x-b) mod p, L); # apply the inverse if 
  decrypting
   else
     S := map(x->(A.x+b) mod p, L);
    fi;
   return VectsToString(S,n);
  end:
```

```
> crypt := AffineMat("aaaaaaaaaaaa", <<0, 5>|<1, 0>>, <0, 0>);
Error, (in AffineMat) matrix is not invertible mod, 95
```

```
> crypt := AffineMat("aaaaaaaaaaaa", <<0, 0>|<1, 0>>, <0, 0>);
Error, (in AffineMat) matrix is not invertible mod, 95
```

```
>
```

a diversion on optional arguments.

```
> Try :=proc( a :: integer, b :: integer, {c :: integer := 3})
    print(a, b, c);
   end:
```

```
> Try(1);
Error, invalid input: Try uses a 2nd argument, b (of type 
integer), which is missing
```

```
> Try(1, 2, c=7);
```

$$1, 2, 7 \tag{28}$$

```
> Try(1 , 2);
```

$$1, 2, 3 \tag{29}$$

```
> Try2 :=proc( a :: integer, b :: integer := 15, {c :: integer := 3})
    print(a, b, c);
   end:
```

```
> Try2(1, 2, c=7);
```

$$1, 2, 7 \tag{30}$$

```
> Try2(1);
```

$$1, 15, 3 \tag{31}$$

```
> Try2(1, c = 18);
                                  1, 15, 18                                  (32)
> debug(Try2);
                                    Try2                                     (33)
> Try2(1);
{--> enter Try2, args = 1
                                  1, 15, 3
<-- exit Try2 (now at top level) = }
> msolve( 3·x + 5 = 0, 97 );
                                  {x = 63}                                   (34)
> msolve( {3·x + y = 0, 2·x + y = 18}, 97 );
                              {x = 79, y = 54}                               (35)
> printf("%s has %d characters %s", "joe", length("joe"), "yo1");
joe has 3 characters yo1
> print("%s has %d characters %s", "joe", length("joe"), "yo1");
            "%s has %d characters %s", "joe", 3, "yo1"                       (36)
> printf("joe"); printf(" sez "); printf("hi!");
joe sez hi!
> print("joe"); print(" sez "); print("hi!");
                                    "joe"
                                   " sez "
                                    "hi!"                                    (37)
>
```