```
>
```

Initial setup and utility functions.

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

PrintRuler is just to make a little ruler under the Alphabet so we can easily see what character has which position.

```
> PrintRuler:=proc(n)
   local j;
   for j from 0 to n-1 do
    if  (j mod 10 = 0) then printf("%d",trunc( modp(j,100)/10));

    elif (j mod 10 = 5) then printf("+");
    else printf(".");
     fi;
   od;
  end:
```

Let's define our Alphabet by selecting all printable characters from the ASCII sequence.

```
> Alphabet := Select(IsPrintable, convert([seq(i,i=1..127)],
  bytes)):
  printf("Our Alphabet is \n%s\n",Alphabet); PrintRuler(length
  (Alphabet));
Our Alphabet is
 !"#$%&'()*+,-./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]
^_`abcdefghijklmnopqrstuvwxyz{|}~
0....+....1....+....2....+....3....+....4....+....5....+....6....+....7....+
```

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.

```
> StringToList := proc (str::string)
    global Alphabet;
    return([seq(SearchText(str[i], Alphabet)-1, i = 1 .. length
  (str))]);
  end:
  ListToString := proc (l::list(nonnegint))
    global Alphabet;
    return(cat(seq(Alphabet[l[i]+1], i = 1 .. nops(l))))
  end:
```

Affine cipher

To encrypt using (integers) a and b, use Affine(plaintext, a, b);

To decrypt a message encrypted with a and b, use Affine(crypttext, a, b, decrypt);

The encrypting key a must be relatively prime to the length of Alphabet.

```
> Affine := proc (plain::string, a::integer, b::integer,
  {decrypt:=false})
   local L, S, len;
   global Alphabet;
   len := length(Alphabet);
```

```
    if (gcd(len, a)>1) then
       error (a, " is not relatively prime to length of Alphabet",
  len);
     fi;

    L := StringToList(plain);
    if (decrypt) then
      S:=map(x->(x-b)/a mod len, L); # apply the inverse if
  decrypting
    else
      S := map(x->(a*x+b) mod len, L);
     fi;
    return ListToString(S);
  end:
```

```
> Affine("Once upon a midnight dreary, while I pondered weak and weary", 47, 81);
      "*J ~qvIyJq!qz|OJ|}MGqOH~!HtkquM|K~q-qIyJO~H~Oqu~!{q!JOqu~!Ht"        (2.1)
> crypto := "*J ~qvIyJq!qz|OJ|}MGqOH~!HtkquM|K~q-qIyJO~H~Oqu~!{q!JOqu~!Ht";
   crypto := "*J ~qvIyJq!qz|OJ|}MGqOH~!HtkquM|K~q-qIyJO~H~Oqu~!{q!JOqu~!Ht"  (2.2)
```

Hint: the first two characters are "On".

```
>
> length(Alphabet);
                                   95                                     (2.3)
```

We know that "*J" decrypts to "On". This means we have the following correspondence

```
> StringToList("·J"), "becomes", StringToList("On");
                      [10, 42], "becomes", [47, 78]                       (1)
```

that is, 10 -> 47 and 42 -> 78

in other words, we must solve 10x+y=47 mod 95, 42x+y=78 mod 95   for x and y.

```
>

> msolve({10·x + y = 47, 42·x + y = 78}, 95 );
                             {x = 93, y = 67}                             (2)
> Affine(crypto, 93, 67);
        "Once upon a midnight dreary, while I pondered weak and weary"    (3)
```

Now let's do something else.

```
> Text := "Why isn't English like Chinese?";
                  Text := "Why isn't English like Chinese?"               (4)
> StringToList(Text);
[55, 72, 89, 0, 73, 83, 78, 7, 84, 0, 37, 78, 71, 76, 73, 83, 72, 0, 76, 73, 75, 69, 0, 35, 72, 73, 78,   (5)
    69, 83, 69, 31]
>
```

Think of each 3 characters as a "big character", so "Why" is one character, " is" is another, "n't" is another, etc.

"Why" = $55*95^2 + 72*95 + 89$ or $89*95^2 +72*95+55$ (either read left-to-right or right-to-left).

```
> convert(46, binary);
                                   101110                                  (6)
> convert(46, base, 15)
                                   [1, 3]                                  (7)
> convert(44, base, 15);
                                   [14, 2]                                 (8)
> convert(46, base, 2);
                             [0, 1, 1, 1, 0, 1]                            (9)
> convert([14, 2], base, 15, 2);
                             [0, 0, 1, 1, 0, 1]                           (10)
> convert(12345, base, 100);
                                [45, 23, 1]                               (11)
> bas95 := StringToList(Text);
bas95 := [55, 72, 89, 0, 73, 83, 78, 7, 84, 0, 37, 78, 71, 76, 73, 83, 72, 0, 76, 73, 75, 69, 0, 35,   (12)
    72, 73, 78, 69, 83, 69, 31]
> convert(bas95, base, 95, 95^3);
   [810120, 756010, 758843, 707465, 666116, 6923, 683886, 315944, 710957, 630679, 31]   (13)
> convert([810120], base, 95^3, 95);
                                [55, 72, 89]                              (14)
> StringToKgraph:=proc(text::string, k::posint)
    local numlist, p;
    global Alphabet;

    p:=length(Alphabet);
    numlist:=StringToList(text);
    return(convert(numlist,base, p, p^k));
  end:
> StringToKgraph(Text, 3);
   [810120, 756010, 758843, 707465, 666116, 6923, 683886, 315944, 710957, 630679, 31]   (15)
> KgraphToString:=proc(numlist::list, k::posint)
    local  p;
    global Alphabet;

    p:=length(Alphabet);
    ListToString(convert(numlist,base, p^k, p));
  end:
> KgraphToString([810120, 756010, 758843, 707465, 666116, 6923, 683886, 315944, 710957,
      630679, 31], 3);
                       "Why isn't English like Chinese?"                  (16)
> BigAffine := proc (plain::string, a::integer, b::integer,
  k::integer, {decrypt:=false})
   local L, S, len;
   global Alphabet;
   len := length(Alphabet);
   if (gcd(len, a)>1) then
      error (a, " is not relatively prime to length of Alphabet",
  len);
    fi;
  L := StringToKgraph(plain,k);
  if (decrypt) then
    S:=map(x->(x-b)/a mod len^k, L); # apply the inverse if
decrypting
  else
    S := map(x->(a*x+b) mod len^k, L);
  fi;
  return KgraphToString(S,k);
end:
```

```
> BigAffine(Text, 1, 0, 4); # identity
                    "Why isn't English like Chinese?"                    (17)
> BigAffine(Text, 1, 1, 4); # identity+1

                    "Xhy jsn'u Enhlisi lile Ciinete?"                    (18)
> BigAffine(Text, 1234567, 9998, 3);
                 "UZe7D`e-"7Ls\(sypbpeVguo'U-gXjzd]"                   (19)
> BigAffine(%, 1234567, 9998, 3, decrypt);
                    "Why isn't English like Chinese?"                    (20)
>
```