▶ Initial setup & conversion of text to/from various list formats.

For RSA,
choose big primes: p,q.
Let n=p*q, phi=(p-1)*(q-1), choose e rel prime to phi, and d=1/e mod phi.

encrypt m as x=m^e mod n, and decrypt x as x^d mod n = m.

>

>

We can convert text with StringToList or with convert(...,bytes).

```
> StringToList("I don't remember");
```

[41, 0, 68, 79, 78, 7, 84, 0, 82, 69, 77, 69, 77, 66, 69, 82] **(1)**

```
> convert("I don't remember", bytes);
```

[73, 32, 100, 111, 110, 39, 116, 32, 114, 101, 109, 101, 109, 98, 101, 114] **(2)**

```
> printf("%s", Alphabet);
```

```
 !"#$%&'()*+,-./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]
^_`abcdefghijklmnopqrstuvwxyz{|}~
```

```
> nums := [seq(i, i=1..128)];
```

nums := [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, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99, 100, 101, 102, 103, 104, 105, 106, 107, 108, 109, 110, 111, 112, 113, 114, 115, 116, 117, 118, 119, 120, 121, 122, 123, 124, 125, 126, 127, 128] **(3)**

```
> Alphabet := convert(nums, bytes);
```

Alphabet := "□□□□□□□□□□□□ !"#$%'()*+,-./0123456789:<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_abcdefghijklmnopqrstuvwxyz{|}~□□" **(4)**

```
> StringToList("I don't remember");
```

[72, 31, 99, 110, 109, 38, 115, 31, 113, 100, 108, 100, 108, 97, 100, 113] **(5)**

```
> Alphabet := ".ABCDEFGHI"; length(Alphabet);
```

Alphabet := ".ABCDEFGHI"

10 **(6)**

```
> StringToList("DEADBEEF");
```

[4, 5, 1, 4, 2, 5, 5, 6] **(7)**

```
> StringToKgraph("DEADBEEF", 5);
```

[24154, 655] **(8)**

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

```
Our 95-character Alphabet is
```

```
 !"#$%&'()*+,-./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]
^_`abcdefghijklmnopqrstuvwxyz{|}~
0....+....1....+....2....+....3....+....4....+....5....+....6...
.+....7....+....8....+....9....
```

```
> p := nextprime(654); q := nextprime(729);
                                  p := 659
                                  q := 733                                  (9)

> n := p·q;
                                 n := 483047                                (10)

> Alphabet[-1];
                                   "~"                                      (11)

> StringToKgraph("~~~~", 4);
                                 [81450624]                                 (12)

> StringToKgraph("~~~~", 3);
                                [857374, 94]                                (13)

> StringToKgraph("~~~~", 2);
                                [9024, 9024]                                (14)

> log[95](n);
                                 ln(483047)
                                 ----------                                 (15)
                                  ln(95)

> evalf(%);
                                2.874005941                                 (16)

> floor(%);
                                    2                                       (17)

> ceil(7.223);
                                    8                                       (18)

> k := floor(log[length(Alphabet)](n));
                                   k := 2                                   (19)

> plain := "Vanilla ice cream is (not totally) plain unless you add stuff.";
          plain := "Vanilla ice cream is (not totally) plain unless you add stuff."   (20)

> nums := StringToKgraph(plain, k);
nums := [6229, 7013, 7296, 65, 6438, 69, 7857, 6244, 77, 7958, 760, 7583, 84, 7589, 6259,   (21)
    7296, 944, 7600, 6251, 7483, 8075, 7298, 7954, 83, 7594, 85, 6525, 68, 8063, 6735, 1400]

> KgraphToString(%, k);
              "Vanilla ice cream is (not totally) plain unless you add stuff."   (22)

> e := 7;
                                   e := 7                                   (23)

> crypto:=map( m-> modp(m&^e, n), nums);
crypto := [156885, 157457, 195127, 292681, 307044, 189729, 133840, 37736, 152272,   (24)
    102131, 169061, 380361, 160589, 330777, 172852, 195127, 100777, 84311, 345330,
    83486, 361205, 376962, 177628, 339476, 378462, 14455, 238473, 87213, 348175, 324275,
    266777]

> DoRSA:=proc(plain::string, n::posint, e::posint)
    local k, nums;
    global Alphabet;
```

```
    k:=floor(log[length(Alphabet)](n));
    nums:=StringToKgraph(plain, k);
    return( map( m-> modp(m&^e, n), nums));
  end:
```

```
> DoRSA(plain, n, 7);
```

[156885, 157457, 195127, 292681, 307044, 189729, 133840, 37736, 152272, 102131, 169061, 380361, 160589, 330777, 172852, 195127, 100777, 84311, 345330, 83486, 361205, 376962, 177628, 339476, 378462, 14455, 238473, 87213, 348175, 324275, 266777] (25)

```
> UndoRSA:=proc(crypt::list, n::posint, e::posint)
    local k, nums;
    global Alphabet;
    k:=floor(log[length(Alphabet)](n));
    KgraphToString(map( m-> modp(m&^e, n), crypt), k);
  end:
```

```
> d := 1/e mod ((p - 1)·(q - 1));
```

Error, the modular inverse does not exist

```
> phi := (p - 1)·(q - 1)
```

$$\phi := 481656 \quad (26)$$

```
> ifactor(phi); e := 11;
```

$$(2)^3\ (3)\ (7)\ (47)\ (61)$$

$$e := 11 \quad (27)$$

```
> crypto := DoRSA(plain, n, e);
```

crypto := [309305, 223163, 162471, 482166, 177665, 251073, 234017, 3285, 358668, 322953, 69841, 447659, 446559, 453362, 201730, 162471, 56393, 390320, 37175, 382599, 14343, 51545, 347796, 176089, 107661, 44427, 12743, 77309, 101887, 266282, 111248] (28)

```
> d := 1/e mod phi;
```

$$d := 43787 \quad (29)$$

```
> UndoRSA(crypto, n, d);
```

"Vanilla ice cream is (not totally) plain unless you add stuff." (30)

```
>
```