Programming with TI-Nspire

By:

Brian Olesen

BO@haslev-gym.dk

Pilot teacher at

Haslev Gymnasium and HF

Skolegade 31

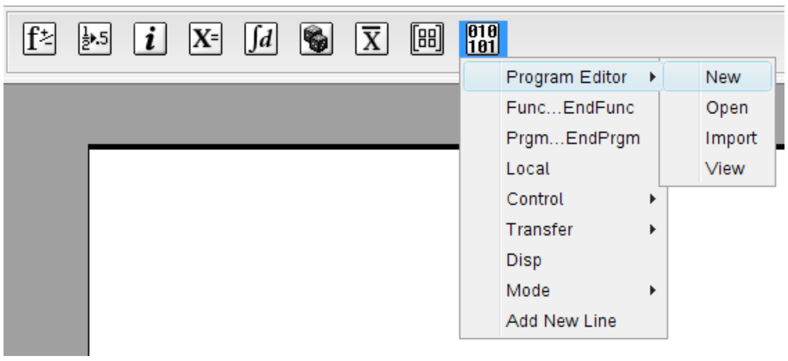
4690 Haslev

Denmark

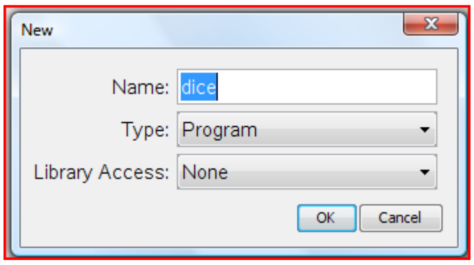
1. Introduction to the Program Editor

Functions and Programs are found in the menu of the **Calculator** application. We will only be working with **Programs** in this introduction.

Let us construct our first simple program where we will simulate a throw of a dice. To get started choose **New** from the **Program Editor** in the menu.



In the pop-up window we are able to name our program **Dice**. We choose **Program** under **Type** and **None** under **Library access** (the possibilities are **None**, **LibPriv**, and **LibPub**). If we chose **LibPub** the program would be accessible in other documents from the **Catalog**. We will return to this later but will not be working with **LibPriv** in this introduction:



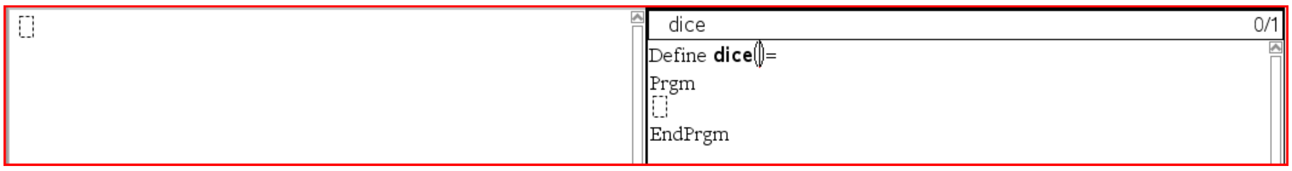
New

Name: dice

Type: Program

Library Access: None

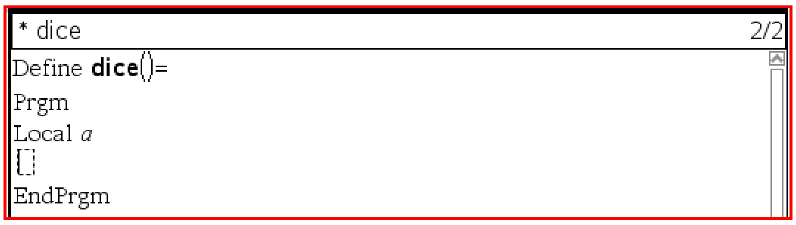
The page is split into two with a **Calculator** application to the left and a **Program Editor** to the right:



```
dice                                   0/1
Define dice()=
Prgm

EndPrgm
```

In the **Program Editor** we notice that the cursor is placed just behind the name of our program in the parenthesis. Our first program does not depend on any given input and we therefore jump to the line under **Prgm**. This is the first and only line, which is reflected in the top of the Program Editor: 1/1.

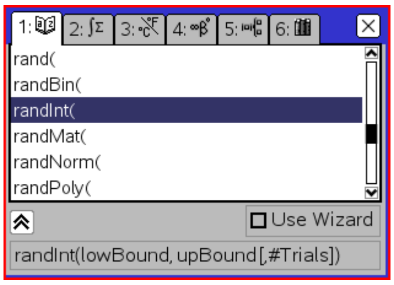
Here we define a local variable, ***a***, via **Local** from the menu **Define Variables**. Write, ***a***, after **Local** and press enter to insert a new line:



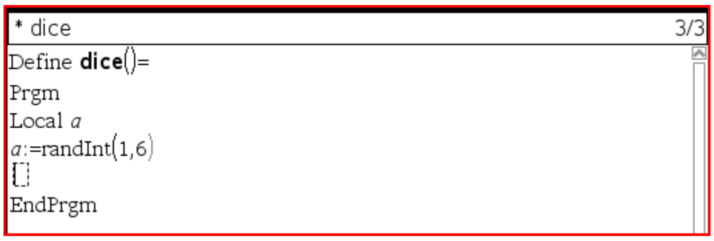
```
* dice                                                  2/2
Define dice()=
Prgm
Local a

EndPrgm
```

Notice the star (*) in front of the name of the program. This reflects a change of the program since it was last stored. We choose **Check syntax & store** from the menu **Check syntax & store** and the star disappears. In the top to the right we now see that the cursor is placed in the second line out of two: 2/2.

With the command **randInt(1,6)** a random number between 1 and 6 is given. We can use this command to simulate a throw of a dice and the command is found in **Catalog**:

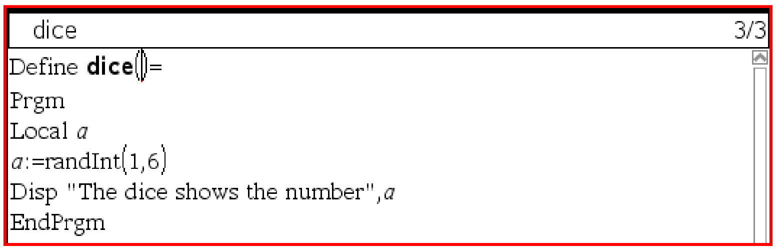


In the second line we define our local variable as ***a* := randInt(1,6)** and press enter to insert a line:



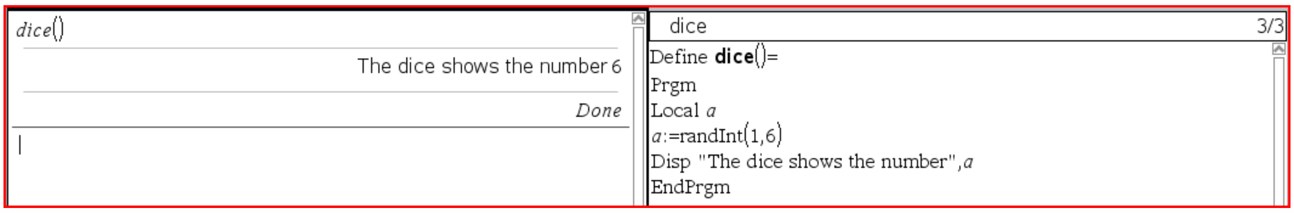
```
* dice                                                  3/3
Define dice()=
Prgm
Local a
a:=randInt(1,6)

EndPrgm
```

Our program is almost finished. We just need to ensure that the program will display our output. In the third line we choose **Disp** from the menu: **I/O** and after the command **Disp** we write: **"The dice shows the number", *a***. The syntax is checked and the program is stored. We are now ready to test it in the **Calculator** application:



```
  dice                                                  3/3
Define dice()=
Prgm
Local a
a:=randInt(1,6)
Disp "The dice shows the number",a
EndPrgm
```

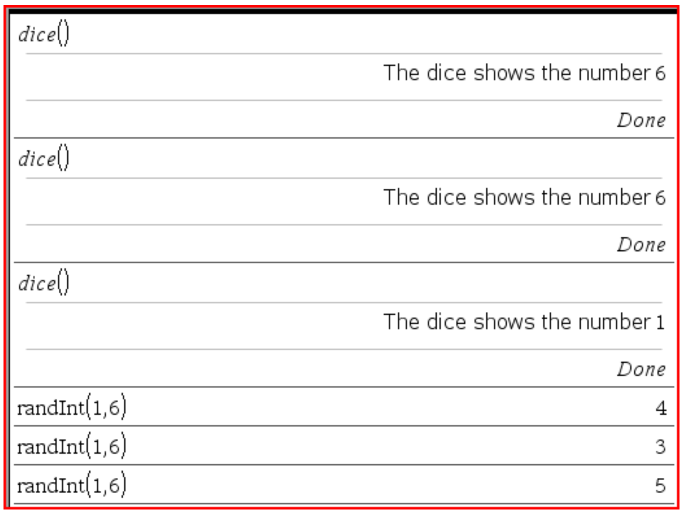
In the calculator we write **dice()**:



```
dice()
                  The dice shows the number 6
                                         Done
```

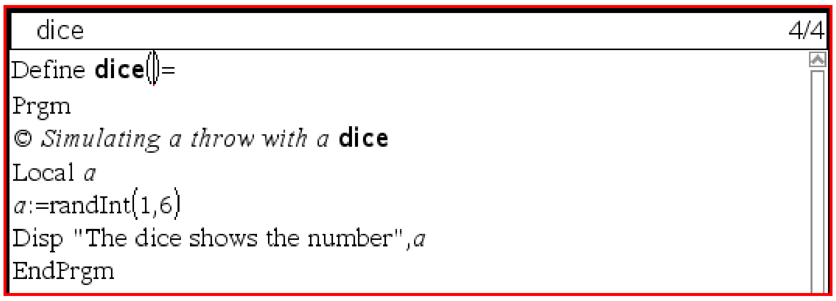
```
dice                                      3/3
Define dice()=
Prgm
Local a
a:=randInt(1,6)
Disp "The dice shows the number",a
EndPrgm
```

Press enter a couple of times to check that our simulation works. Our first program is only a trivial expansion of the command **randInt()**:



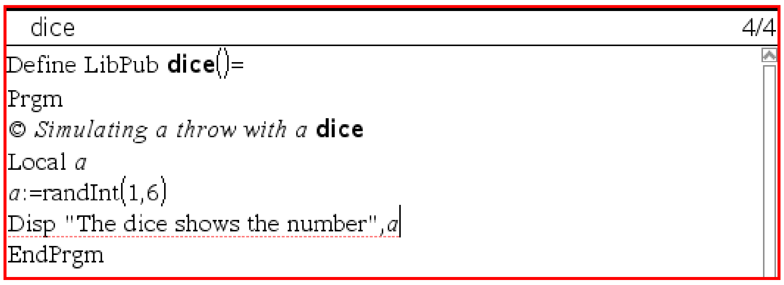
```
dice()
                  The dice shows the number 6
                                         Done
dice()
                  The dice shows the number 6
                                         Done
dice()
                  The dice shows the number 1
                                         Done
randInt(1,6)                                4
randInt(1,6)                                3
randInt(1,6)                                5
```

In the first line of the program we insert a comment by choosing **Insert comment** from the menu **Actions**. After the symbol © we write ***Simulating a throw with a dice*** as description of our program:



```
dice                                      4/4
Define dice()=
Prgm
© Simulating a throw with a dice
Local a
a:=randInt(1,6)
Disp "The dice shows the number",a
EndPrgm
```

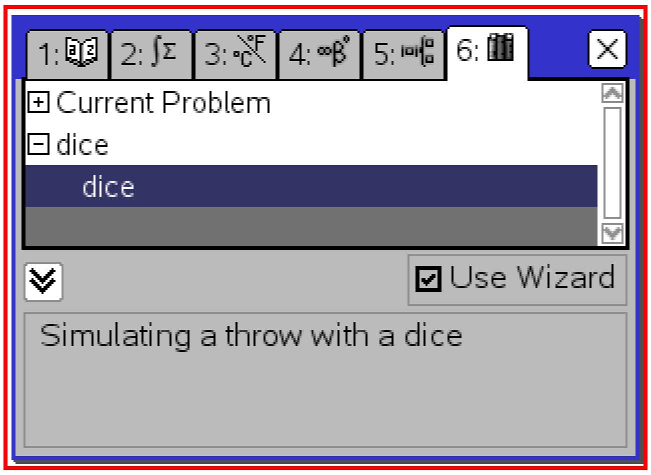
For now we can only use the program in the current problem. Let us change this so that the program is accessible in every new TI-Nspire document from the **Catalog**. From the menu **Actions** we can select **LibPub** from **Change library access**:



```
dice                                      4/4
Define LibPub dice()=
Prgm
© Simulating a throw with a dice
Local a
a:=randInt(1,6)
Disp "The dice shows the number",a
EndPrgm
```

We need to store the file in the folder **Mylib** in the directory: **Documents > TI-Nspire > MyLib**. Finally we must update the library by pressing the icon: (or using **Refresh Libraries** from the menu **Tools**).

If we now open a new document we should be able to access the program from the **Catalog**:



Simulating a throw with a dice

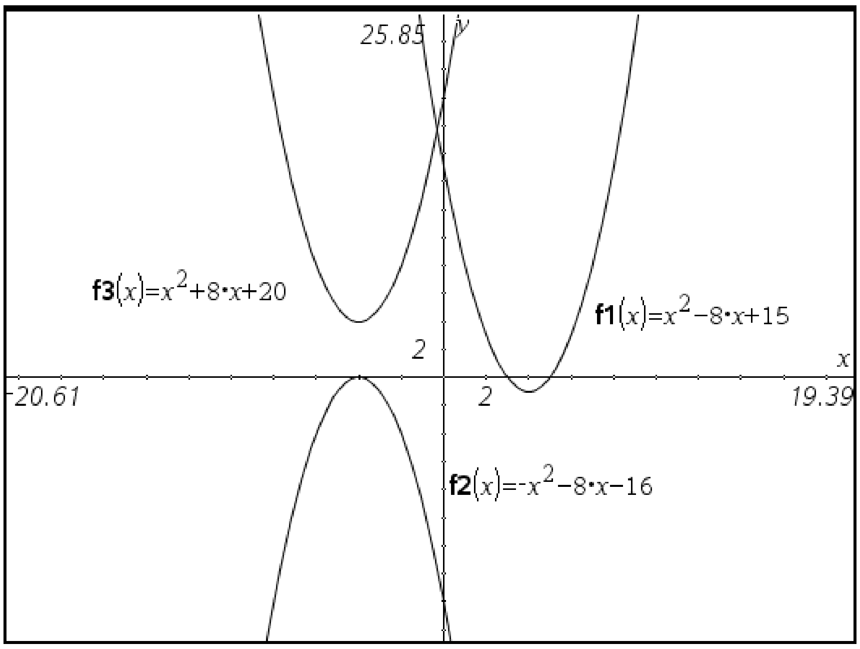
Notice our description of the program is included.

1.1 Exercise: Sum of the throw of two dices

Construct a program that simulates the throw of two dices. The program shall display the outcome of each dice and the sum of the dices.

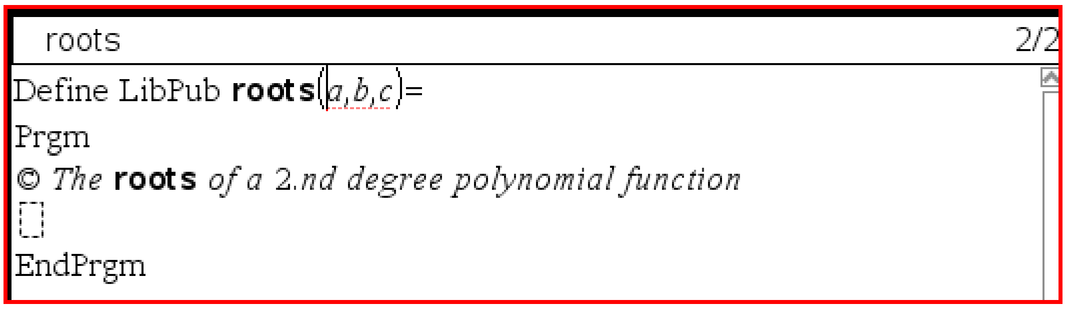
2. Programming with "if… then… endif" and "elseif… then…"

Let us now construct a more interesting and relevant program. We will be working with second degree polynomial functions $f(x) = a \cdot x^2 + b \cdot x + c$. The graph is called a parabola which can have up to two intersection points with the x-axis:



If we solve the second degree equation $0 = a \cdot x^2 + b \cdot x + c$ we are able to determine the intersection points. A solution to the equation is called a root. The number of roots can be determined by calculating the discriminant $d = b^2 - 4 \cdot a \cdot c$. If $d < 0$ there are no roots, if $d = 0$ there is a double root and if $d > 0$ there are two distinct roots.

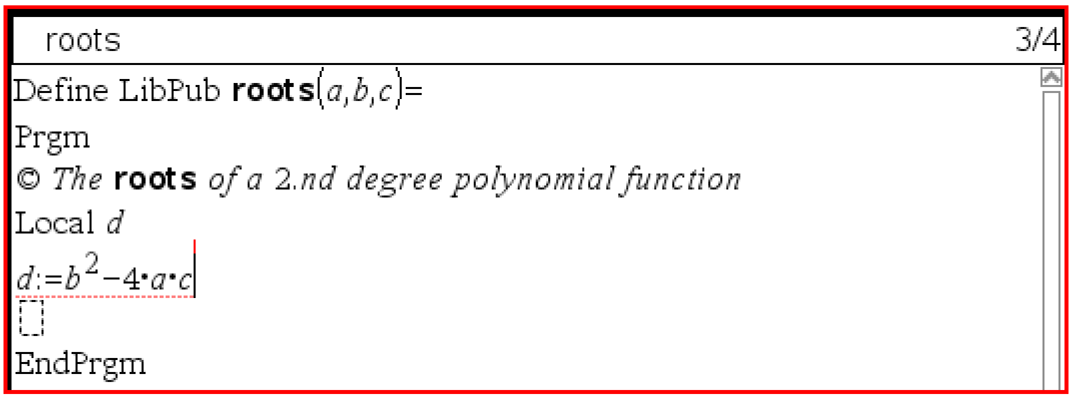
We will construct a program that determines the roots for any given second degree polynomial function. The program is called **roots** and we want to add it to the **Catalog**:



```
roots                                          2/2
Define LibPub roots(a,b,c)=
Prgm
© The roots of a 2.nd degree polynomial function

EndPrgm
```

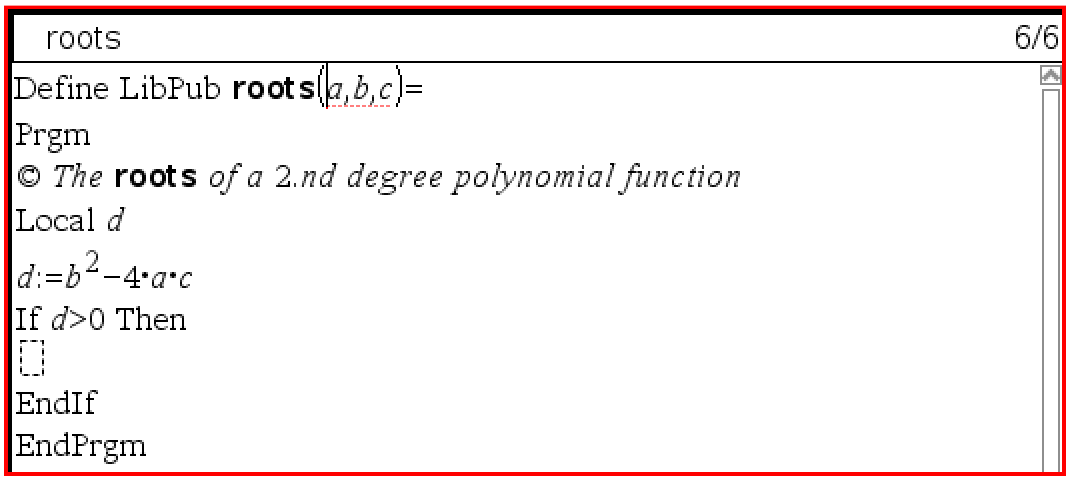
Notice that the program as input depends on the coefficients a, b and c.

The program will be constructed step vice. Let us first calculate the discriminant where we define a local variable ***d***:



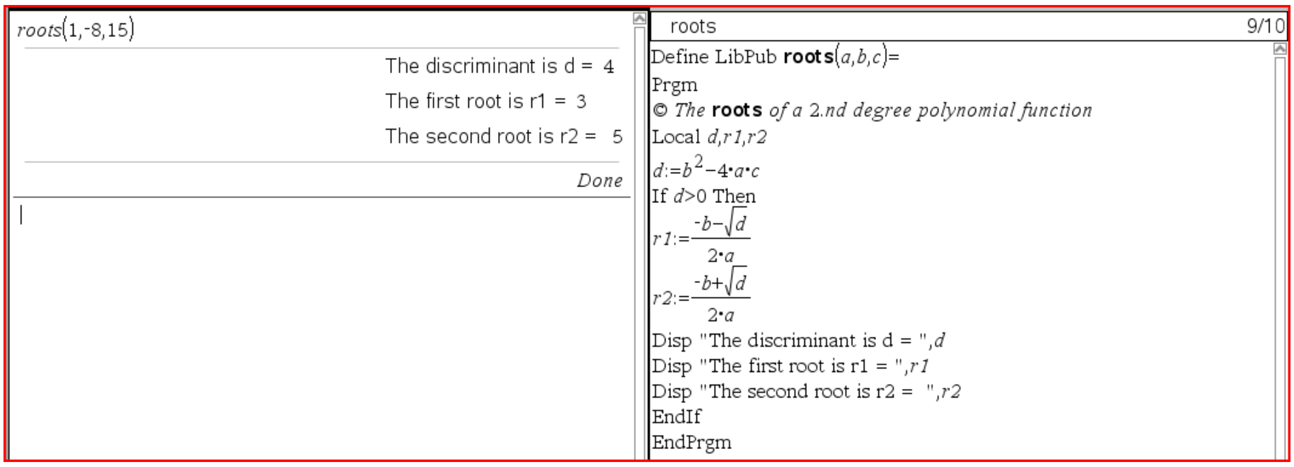
```
roots                                                    3/4
Define LibPub roots(a,b,c)=
Prgm
© The roots of a 2.nd degree polynomial function
Local d
d:=b^2−4•a•c

EndPrgm
```

We now need an **If**-command to separate the three possible numbers of roots. We choose **If…Then…endif** from the menu. After **If** we write: $d > 0$. In this case there are two distinct roots ***r1*** and ***r2*** which we define as local variables:



```
roots                                                    6/6
Define LibPub roots(a,b,c)=
Prgm
© The roots of a 2.nd degree polynomial function
Local d
d:=b^2−4•a•c
If d>0 Then

EndIf
EndPrgm
```

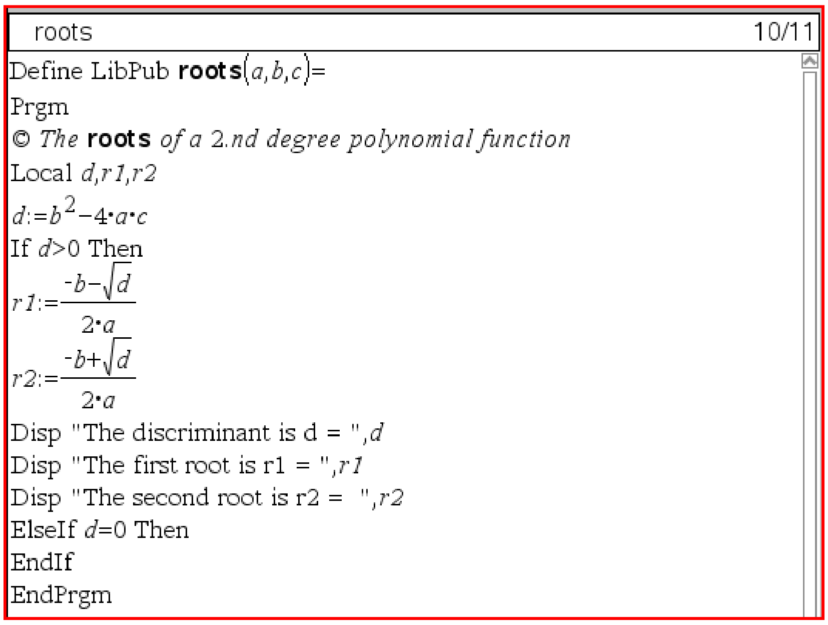
The two roots are given as either $r1 = \frac{-b+\sqrt{d}}{2a}$ or $r2 = \frac{-b-\sqrt{d}}{2a}$ which will be calculated in the program. Finally we want the program to display the discriminant and the roots:



```
roots(1,-8,15)
                         The discriminant is d =  4
                         The first root is r1  =  3
                         The second root is r2 =   5
                                                Done
```

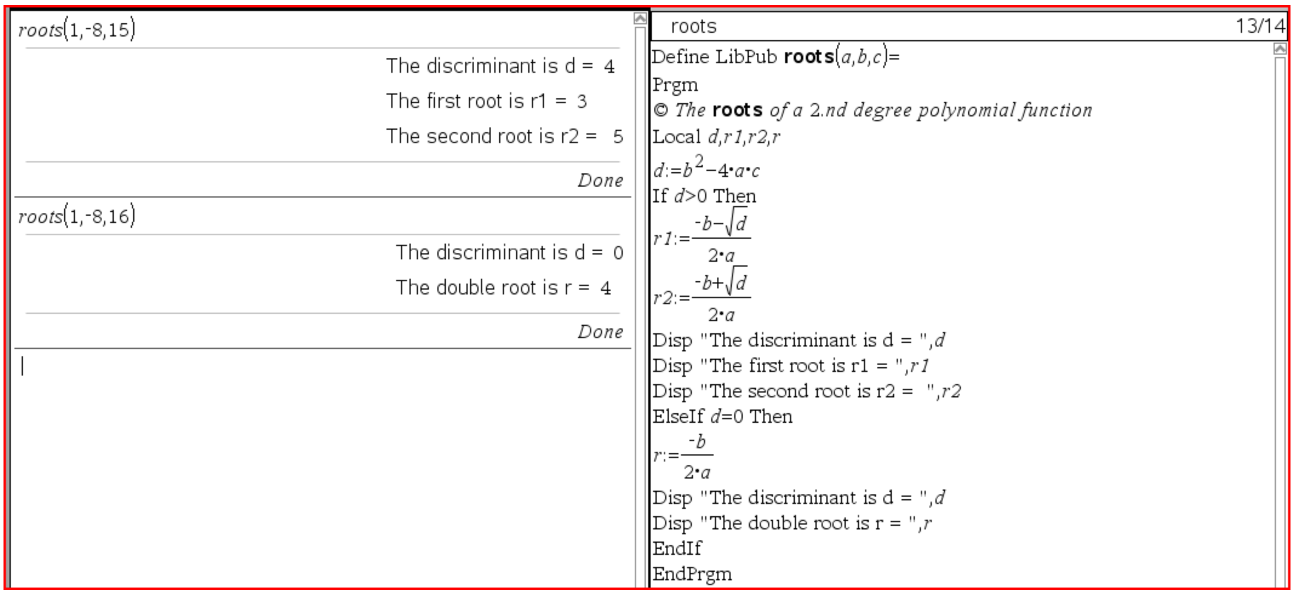
```
roots                                                    9/10
Define LibPub roots(a,b,c)=
Prgm
© The roots of a 2.nd degree polynomial function
Local d,r1,r2
d:=b^2−4•a•c
If d>0 Then
r1:=(-b−√d)/(2•a)
r2:=(-b+√d)/(2•a)
Disp "The discriminant is d = ",d
Disp "The first root is r1 = ",r1
Disp "The second root is r2 =  ",r2
EndIf
EndPrgm
```

We add the case where $d = 0$. In line 10 (just above **Endif**) we insert **Elseif…then** which can be chosen from the menu **Control**. After **Elseif** we write $d = 0$:



```
roots                                              10/11
Define LibPub roots(a,b,c)=
Prgm
© The roots of a 2.nd degree polynomial function
Local d,r1,r2
d:=b^2−4·a·c
If d>0 Then
r1:=(-b−√(d))/(2·a)
r2:=(-b+√(d))/(2·a)
Disp "The discriminant is d = ",d
Disp "The first root is r1 = ",r1
Disp "The second root is r2 =  ",r2
ElseIf d=0 Then
EndIf
EndPrgm
```

As mentioned earlier there is a double root in this case and it is given by calculating $r = \frac{-b}{2a}$ which is written into the program. The discriminant and the root are displayed while we must remember to add ***r*** to the list of local variables:

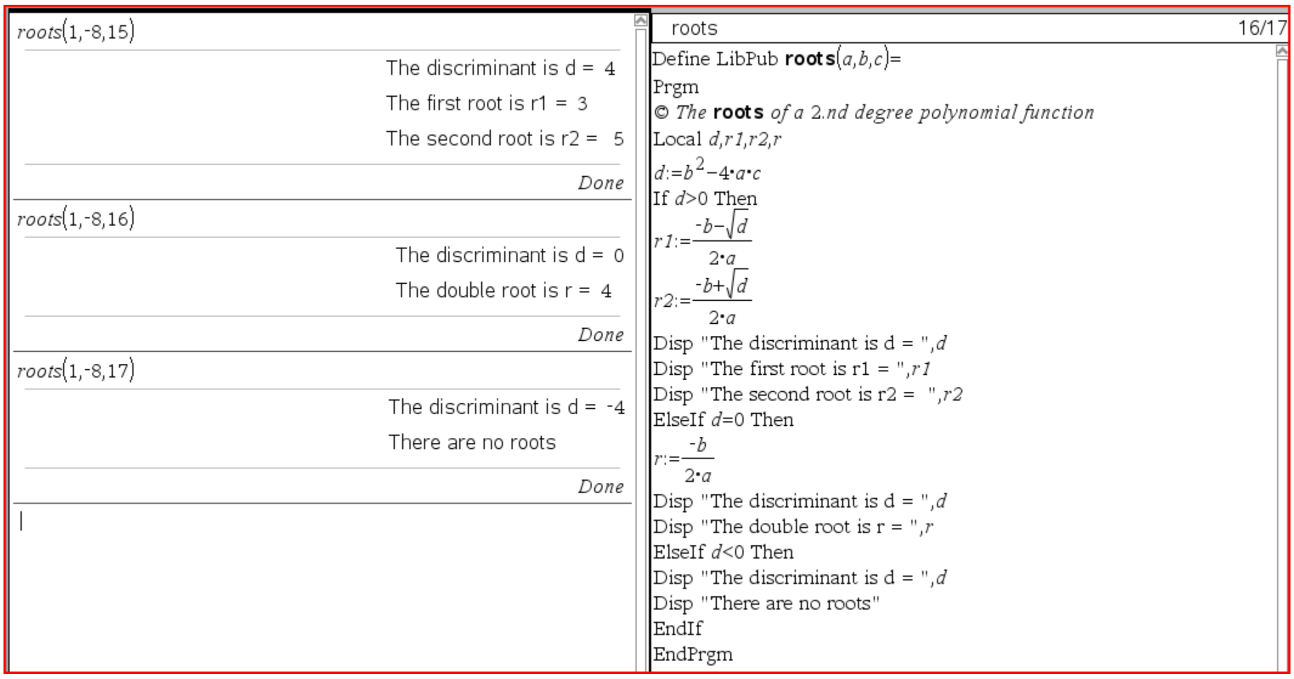


```
roots(1,-8,15)
                    The discriminant is d =  4
                    The first root is r1 =  3
                    The second root is r2 =  5
                                          Done
roots(1,-8,16)
                    The discriminant is d =  0
                    The double root is r =  4
                                          Done
```

```
roots                                              13/14
Define LibPub roots(a,b,c)=
Prgm
© The roots of a 2.nd degree polynomial function
Local d,r1,r2,r
d:=b^2−4·a·c
If d>0 Then
r1:=(-b−√(d))/(2·a)
r2:=(-b+√(d))/(2·a)
Disp "The discriminant is d = ",d
Disp "The first root is r1 = ",r1
Disp "The second root is r2 =  ",r2
ElseIf d=0 Then
r:=(-b)/(2·a)
Disp "The discriminant is d = ",d
Disp "The double root is r = ",r
EndIf
EndPrgm
```

Now, the last case where $d < 0$. Again we add the command **Elseif…then** in a line above

Endif and we write $d < 0$ after **Elseif**. We want the program to display the message that there are no roots together with the discriminant:



```
roots(1,-8,15)
                    The discriminant is d =  4
                    The first root is r1 =  3
                    The second root is r2 =  5
                                          Done
roots(1,-8,16)
                    The discriminant is d =  0
                    The double root is r =  4
                                          Done
roots(1,-8,17)
                    The discriminant is d =  -4
                    There are no roots
                                          Done
```

```
roots                                                    16/17
Define LibPub roots(a,b,c)=
Prgm
© The roots of a 2.nd degree polynomial function
Local d,r1,r2,r
d:=b^2-4·a·c
If d>0 Then
r1:=(-b-√(d))/(2·a)
r2:=(-b+√(d))/(2·a)
Disp "The discriminant is d = ",d
Disp "The first root is r1 = ",r1
Disp "The second root is r2 =  ",r2
ElseIf d=0 Then
r:=(-b)/(2·a)
Disp "The discriminant is d = ",d
Disp "The double root is r = ",r
ElseIf d<0 Then
Disp "The discriminant is d = ",d
Disp "There are no roots"
EndIf
EndPrgm
```

2.1 Exercise: The minimum/maximum of the parabola

Add the determination of the minimum/maximum to the program **roots**.

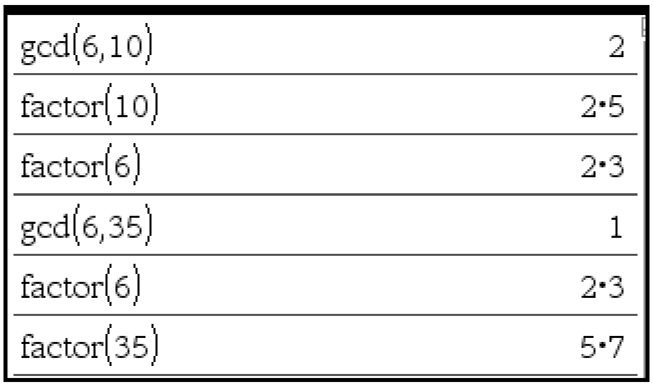
2.2 Exercise: Heads or tails

Construct a program that simulates a flip of a coin ***n*** times. Input must be the number of times the coin is flipped. The program must display the different outcomes together with the number of both **heads** and **tails**.

3. Number theory with TI-Nspire

We will not cover the number theory needed to understand the affine cryptosystem but only discuss the use of TI-Nspire in this connection. This material can be used as supplement to a mathematical learning of the relevant number theory.

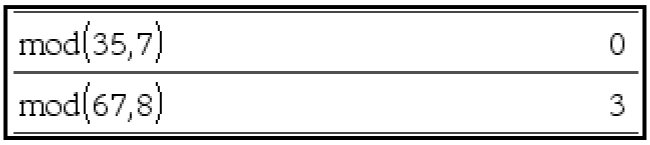
Greatest common divisor **gcd** is a built-in command in TI-Nspire:



```
gcd(6,10)          2
factor(10)       2·5
factor(6)        2·3
gcd(6,35)          1
factor(6)        2·3
factor(35)       5·7
```

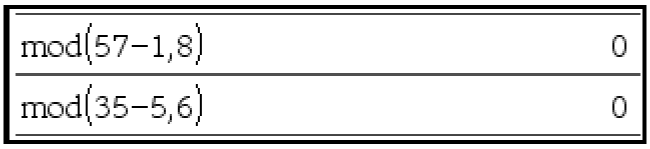
We see that 2 is a factor in both $10 = 2 \cdot 5$ and $6 = 2 \cdot 3$ and it is the greatest common factor. We also see that the numbers 6 and 36 are relatively *prime to each other* as their greatest common divisor is the unity 1.

By modulus calculation we see that 7 is a divisor in 35 as the remainder is zero and we see that the principal remainder of 67 divided by 8 is 3:



```
mod(35,7)          0
mod(67,8)          3
```

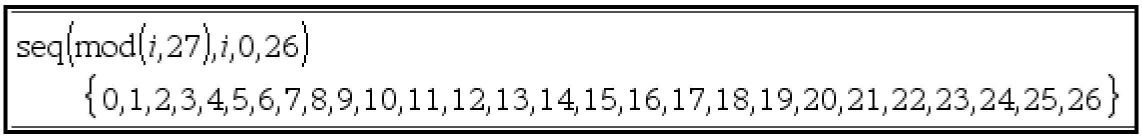
In this way we can show that two numbers are congruent:



```
mod(57−1,8)        0
mod(35−5,6)        0
```

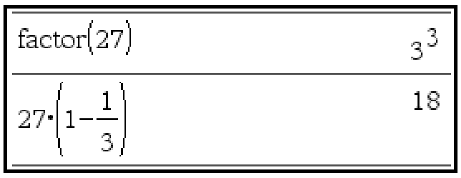
Here 57 is congruent with 1 modulus 8 as 8 is a divisor in $57 - 1$. This is written: $57 \equiv 1 \pmod 8$ as $8 \mid (57 - 1)$.

With a sequence we can determine the set of different remainders when dividing with 27 which typically is written $\mathbb{Z}_{27}$:



```
seq(mod(i,27),i,0,26)
      {0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26}
```

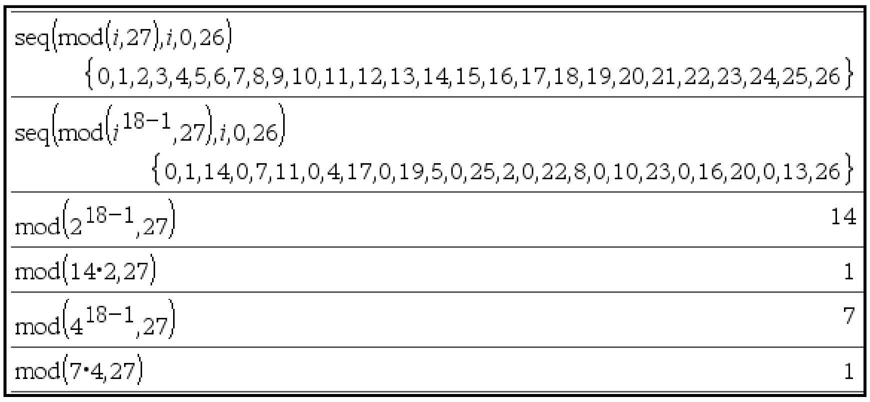
Using Eulers formula we can determine the number of remainders that are prime to 27 in $\mathbb{Z}_{27}$ (this corresponds to the Euler φ-function $\varphi(n) = n \cdot \left(1-\frac{1}{p_1}\right) \cdot \left(1-\frac{1}{p_2}\right) \cdot \ldots \cdot \left(1-\frac{1}{p_k}\right)$, where p_1, p_2, ..., p_k are the different prime factors associated with the number n):



$\text{factor}(27)$	3^3
$27\cdot\left(1-\frac{1}{3}\right)$	18

First we make a factorization of the number 27 and see that the only distinct prime number is 3. By calculation $\varphi(27) = 18$.

Using Eulers theorem $a^{\varphi(n)} \equiv 1(\text{mod}\, n)$ provided a and n are relatively prime, the different inverse elements to the remainders that are relatively prime to 27 can be determined:



$\text{seq}(\text{mod}(i,27),i,0,26)$	{0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26}
$\text{seq}(\text{mod}(i^{18-1},27),i,0,26)$	{0,1,14,0,7,11,0,4,17,0,19,5,0,25,2,0,22,8,0,10,23,0,16,20,0,13,26}
$\text{mod}(2^{18-1},27)$	14
$\text{mod}(14\cdot2,27)$	1
$\text{mod}(4^{18-1},27)$	7
$\text{mod}(7\cdot4,27)$	1

We see that 14 is inverse element to 2 modulus 27 because $14 \cdot 2 \equiv 1$ (mod 27). Here the inverse element to 2 is determined by the calculation: $2^{\varphi(27)-1} \equiv 2^{18-1} \equiv 2^{17} \equiv 14$ (mod 27).

And the inverse element to 4 modulus 27 is 7.

From the sequence command we see that 3 has no inverse elements because $3^{18-1} \equiv 0$ (mod 27) while 3 is not relatively prime to 27. The same accounts for 6, 9, 12...

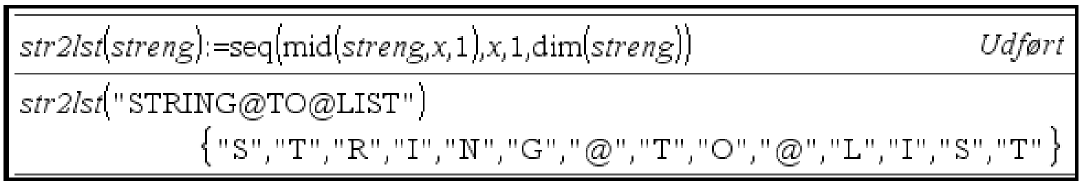
3.1 Exercise: Number theory

a) Determine the principal remainder of 674 divided by 13
b) Determine the greatest common divisor of the numbers 349 og 154
c) Show that -9 is congruent to 3 modulus 4 and that 54 is congruent to 11 modulus 6
d) Determine $\mathbb{Z}_{30}$ and $\varphi(30)$
e) Determine if 16 has an inverse element modulus 27. If so, what is the inverse element?

4.1 Affine cryptosystems with TI-Nspire

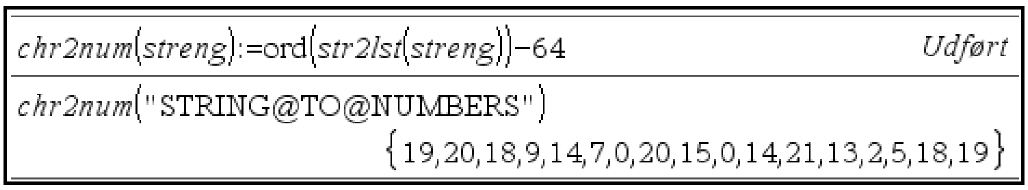
The starting point for our treatment of the affine cryptosystem is the English alphabet and messages in English. Characters are written in capital and the symbol @ is used for space between words. Therefore 26 + 1 signs are available and we are working in $\mathbb{Z}_{27}$ with 18 remainders that are relatively prime to 27. Before we start encrypting messages we will look at some commands.

The command **str2lst(streng)** converts a string to a list:



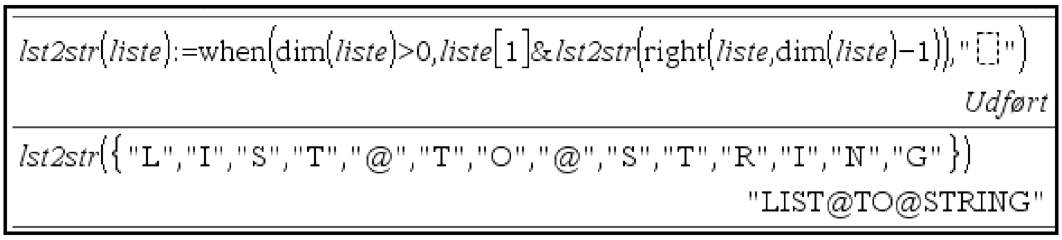
```
str2lst(streng):=seq(mid(streng,x,1),x,1,dim(streng))                Udført
str2lst("STRING@TO@LIST")
              {"S","T","R","I","N","G","@","T","O","@","L","I","S","T"}
```

The above command is used in the command **chr2num(string)** which converts a string to a list of numbers so that **@** is assigned the value **0**, **A** the value **1**, **B** the value **2**, and so on:



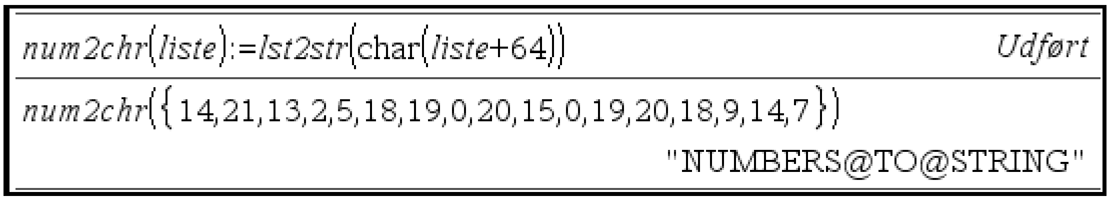
```
chr2num(streng):=ord(str2lst(streng))−64                       Udført
chr2num("STRING@TO@NUMBERS")
                       {19,20,18,9,14,7,0,20,15,0,14,21,13,2,5,18,19}
```

The command **lst2str(liste)** converts a list to a streng:



```
lst2str(liste):=when(dim(liste)>0,liste[1]&lst2str(right(liste,dim(liste)−1)),"⬚")
                                                                       Udført
lst2str({"L","I","S","T","@","T","O","@","S","T","R","I","N","G"})
                                                            "LIST@TO@STRING"
```

The command **lst2str(liste)** is used in the command **num2chr(liste)** to convert a list of numbers to a streng:



```
num2chr(liste):=lst2str(char(liste+64))                          Udført
num2chr({14,21,13,2,5,18,19,0,20,15,0,19,20,18,9,14,7})
                                                   "NUMBERS@TO@STRING"
```

4.1.1 Exercise

Investigate the commands **str2lst(streng)** and **chr2num(string)** and determine how they work.

4.2 Affine encryption og decryption with TI-Nspire

In this part we are able to number the characters in the English alphabet from 0 to 26 corresponding to the order of the characters. Spaces between words are symbolized by the sign @ and given the number 0.

Encryption with the affine cryptosystem is determined by linear functions:

$$E(x) \equiv a \cdot x + b \text{ (mod 27) where } a, b \in \mathbb{Z}_{27}$$

Decryption is possible if a is prime to 27, that is if $a \in \mathbb{Z}_{27}^{*}$, because a then has an inverse element a^{-1}. By solving the equation, where $E(x) = y$, we get the following:

$$y \equiv a \cdot x + b \text{ (mod 27)}$$

$$y - b \equiv a \cdot x \text{ (mod 27)}$$

$$a^{-1} \cdot (y - b) \equiv a^{-1} \cdot a \cdot x \text{ (mod 27)}$$

$$x \equiv a^{-1} \cdot (y - b) \text{ (mod 27)}$$

Therefore the decryption function is given by:

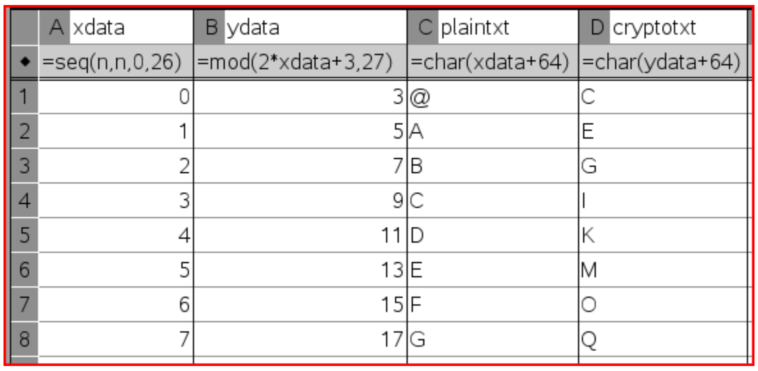
$$D(y) \equiv a^{-1} \cdot (y - b) \text{ (mod 27)}$$

4.2.1 Example

Let us watch an example of an affine encryption function:

$$E(x) \equiv 2 \cdot x + 3 \text{ (mod 27)}$$

The starting point is the English alphabet, where every character is represented by a number from 0-26:

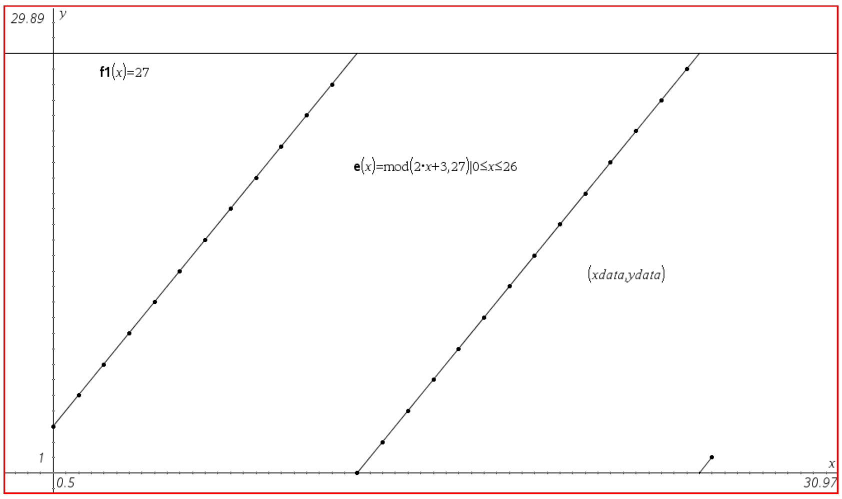


	A xdata	B ydata	C plaintxt	D cryptotxt
◆	=seq(n,n,0,26)	=mod(2*xdata+3,27)	=char(xdata+64)	=char(ydata+64)
1	0	3	@	C
2	1	5	A	E
3	2	7	B	G
4	3	9	C	I
5	4	11	D	K
6	5	13	E	M
7	6	15	F	O
8	7	17	G	Q

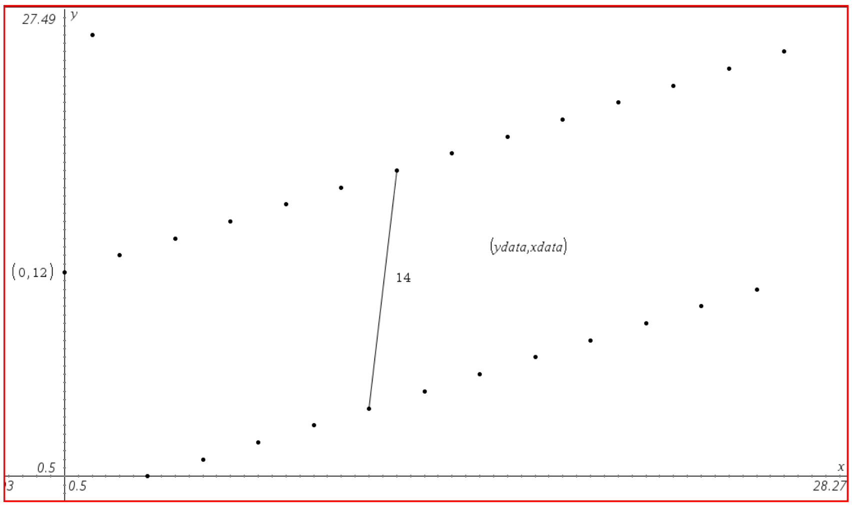
The **xdata** are the numbers 0-26 that represent the characters **@**, **A**, **B**, **C**, …, **X**, **Y**, **Z** from the plaintext (**plaintxt**). The **ydata** are determined by modulus calculation and represents the cryptotext (**cryptxt**).

Notice that we here use the **char** command to convert at number to a character.

We make a scatter plot of our **xdata** and **ydata**. Then we draw the graph of our encryption function:



Now we are going to graphically determine the decryption function, which can be used as a method in crypto analysis. We do this by interchanging the independent and dependent variables by drawing a scatter plot with **ydata** and **xdata** along the **x**- and **y**-axis:



We are searching for a linear decryption function:

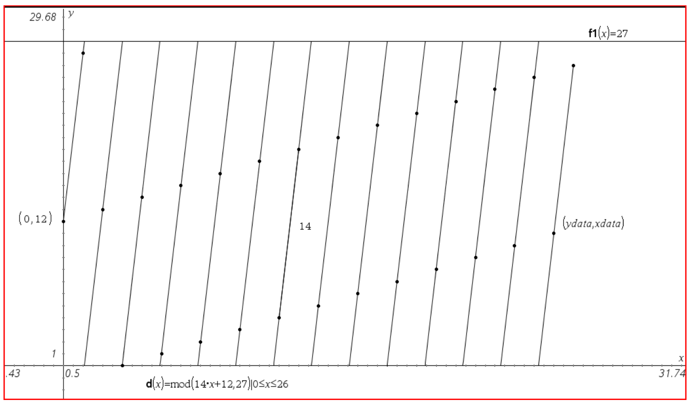
$$D(x) \equiv a_d \cdot x + b_d \pmod{27} \text{ where } a_d, b_d \in \mathbb{Z}_{27}$$

We observe that when **ydata** equals 0 then **xdata** equals 12 which correspond to the intersection between the linear line and the y-axis. Therefore $b_d = 12$. The slope is determined by connecting two neighboring points with a line segment and measuring the slope. Therefore $a_d = 14$.

Our guess of an affine decryption function is

$$D(x) \equiv 14 \cdot x + 12 \pmod{27}$$

We now draw the graph of our idea of a decryption function together with **ydata** and **xdata**:



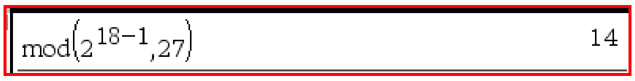
As mentioned this method to determine a decryption function can be used in crypto analysis in the attempt to break an encrypted message. Let us be sure that the decryption function equals the one that could be determined analytic if the encryption function was known:

$$E(x) \equiv 2 \cdot x + 3 \ (\text{mod } 27)$$

From the theory we know that the decryption function is given by:

$$D(y) \equiv 2^{-1} \cdot (y - 3) \ (\text{mod } 27)$$

From the part about number theory we know that by calculation we get:



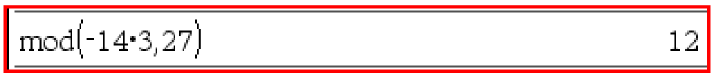
$\text{mod}(2^{18-1},27)$ 14

Therefore we have: $D(y) \equiv 14 \cdot (y - 3) \ (\text{mod } 27)$

And by multiplication we get:

$$D(y) \equiv 14 \cdot (y - 3) \equiv 14 \cdot y - 14 \cdot 3 \equiv 14 \cdot y + 12 \ (\text{mod } 27)$$

Because:



$\text{mod}(-14 \cdot 3,27)$ 12

This equals the decryption function we determined graphically.

4.2.2. Exercise

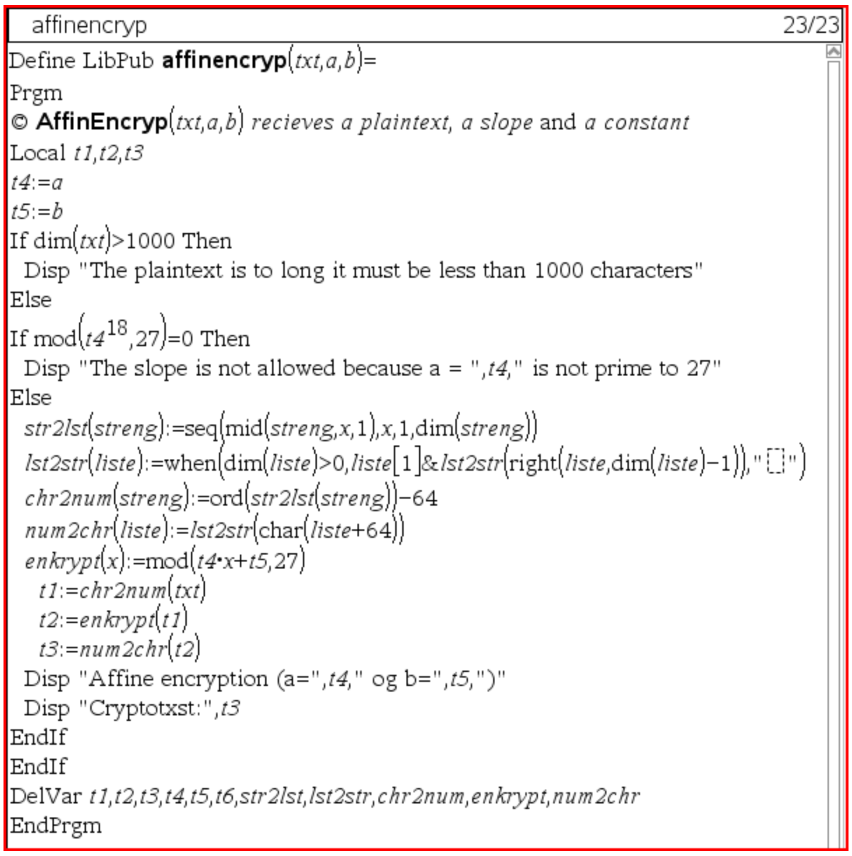
Determine graphically the decryption function associated with:

$$E(x) \equiv 5 \cdot x + 8 \ (\text{mod } 27)$$

Compare with the analytically determined decryption function.

4.3 A program for affine encryption

This program can be used for **affine encryption** of English plain text. The program receives a **plaintext** as a string a **slope,** and a **constant** of the linear encryption function:



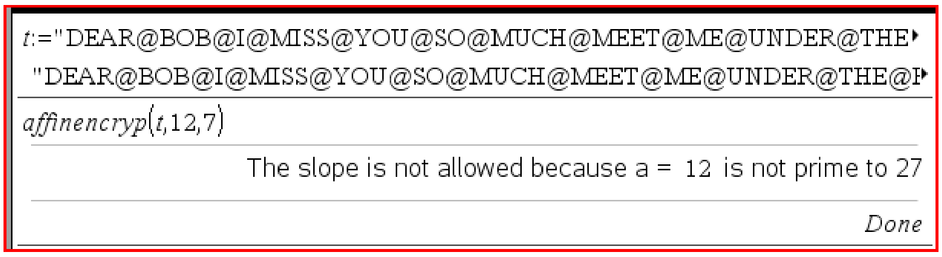
```
affinencryp                                                        23/23
Define LibPub affinencryp(txt,a,b)=
Prgm
© AffinEncryp(txt,a,b) recieves a plaintext, a slope and a constant
Local t1,t2,t3
t4:=a
t5:=b
If dim(txt)>1000 Then
  Disp "The plaintext is to long it must be less than 1000 characters"
Else
If mod(t4^18,27)=0 Then
  Disp "The slope is not allowed because a = ",t4," is not prime to 27"
Else
  str2lst(streng):=seq(mid(streng,x,1),x,1,dim(streng))
  lst2str(liste):=when(dim(liste)>0,liste[1]&lst2str(right(liste,dim(liste)-1)),"⬚")
  chr2num(streng):=ord(str2lst(streng))-64
  num2chr(liste):=lst2str(char(liste+64))
  enkrypt(x):=mod(t4·x+t5,27)
    t1:=chr2num(txt)
    t2:=enkrypt(t1)
    t3:=num2chr(t2)
  Disp "Affine encryption (a=",t4," og b=",t5,")"
  Disp "Cryptotxst:",t3
EndIf
EndIf
DelVar t1,t2,t3,t4,t5,t6,str2lst,lst2str,chr2num,enkrypt,num2chr
EndPrgm
```

Let us imagine that ALICE wants to send a secret massage to BOB by using the affine crypto system. She wants to send this message:

DEAR BOB I MISS YOU SO MUCH MEET ME UNDER THE BRIDGE AT TEN PM TOMORROW LOVE ALICE

Alice writes the plaintext as a string where the spaces are represented by @ and she decides to use this encryption function:

$$E(x) \equiv 12 \cdot x + 7 \pmod{27}$$



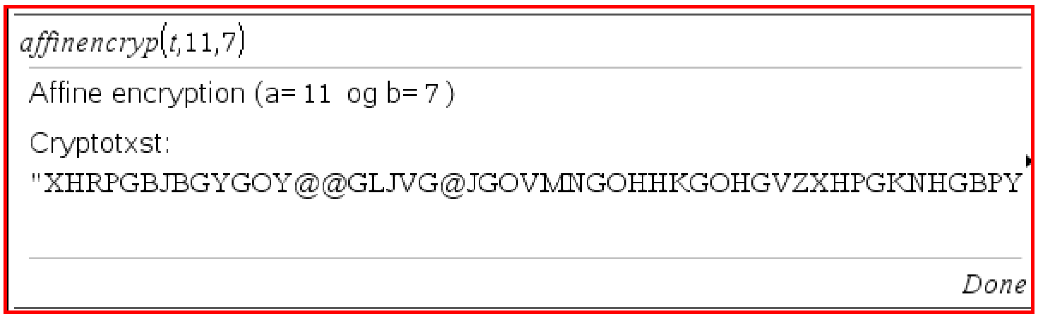
```
t:="DEAR@BOB@I@MISS@YOU@SO@MUCH@MEET@ME@UNDER@THE▸
 "DEAR@BOB@I@MISS@YOU@SO@MUCH@MEET@ME@UNDER@THE@P▸
affinencryp(t,12,7)
               The slope is not allowed because a =  12  is not prime to 27
                                                                       Done
```

She realizes that this function does not work because the slope does not have an inverse element modulus 27.

With the function:

$$E(x) \equiv 11 \cdot x + 7 \pmod{27}$$

it goes much better:



```
affinencryp(t,11,7)
Affine encryption (a= 11  og b= 7 )
Cryptotxst:
"XHRPGBJBGYGOY@@GLJVG@JGOVMNGOHHKGOHGVZXHPGKNHGBPY
                                                    Done
```

The crypto text that she can send to BOB is:

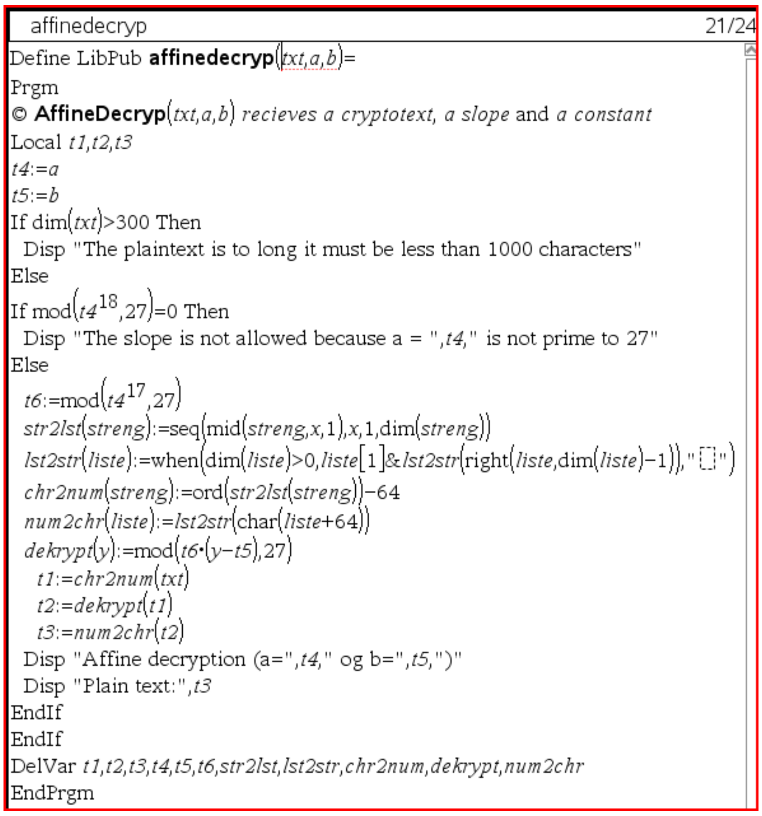
XHRPGBJBGYGOY@@GLJVG@JGOVMNGOHHKGOHGVZXHPGKNHGBPYXC
HGRKGKHZGUOGKJOJPPJQGDJFHGRDYMH

Via a secure communication line BOB is told which encryption function ALICE has used.

Notice that the program will give you a warning if the plaintext exceeds 1000 characters.

4.3 A program for affine decryption

This program can be used to make an **affine decryption** of an English encrypted plain text. The program receives a **crypto text** as a string, a **slope** and a **constant** of the linear encryption function:



```
affinedecryp                                                    21/24
Define LibPub affinedecryp(txt,a,b)=
Prgm
© AffineDecryp(txt,a,b) recieves a cryptotext, a slope and a constant
Local t1,t2,t3
t4:=a
t5:=b
If dim(txt)>300 Then
  Disp "The plaintext is to long it must be less than 1000 characters"
Else
If mod(t4^18,27)=0 Then
  Disp "The slope is not allowed because a = ",t4," is not prime to 27"
Else
  t6:=mod(t4^17,27)
  str2lst(streng):=seq(mid(streng,x,1),x,1,dim(streng))
  lst2str(liste):=when(dim(liste)>0,liste[1]&lst2str(right(liste,dim(liste)−1)),"⬚")
  chr2num(streng):=ord(str2lst(streng))−64
  num2chr(liste):=lst2str(char(liste+64))
  dekrypt(y):=mod(t6·(y−t5),27)
    t1:=chr2num(txt)
    t2:=dekrypt(t1)
    t3:=num2chr(t2)
  Disp "Affine decryption (a=",t4," og b=",t5,")"
  Disp "Plain text:",t3
EndIf
EndIf
DelVar t1,t2,t3,t4,t5,t6,str2lst,lst2str,chr2num,dekrypt,num2chr
EndPrgm
```

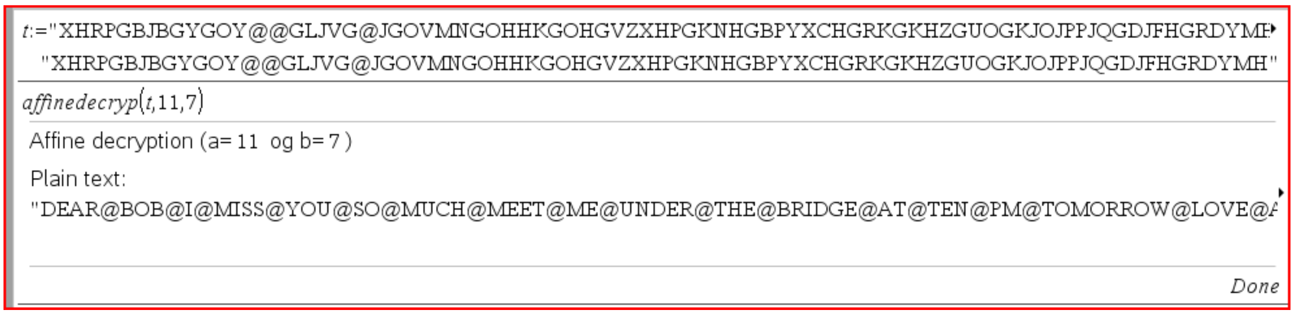
Imagine that BOB has received the message from Alice which is encrypted by the affine system. He receives this crypto text:

XHRPGBJBGYGOY@@GLJVG@JGOVMNGOHHKGOHGVZXHPGKNHGBPYXC
HGRKGKHZGUOGKJOJPPJQGDJFHGRDYMH

BOB is by a secure communication line told that ALICE has used the encryption function:

$$E(x) \equiv 11 \cdot x + 7 \pmod{27}$$

Now it is easy for BOB to decrypt the message from ALICE:



```
t:="XHRPGBJBGYGOY@@GLJVG@JGOVMNGOHHKGOHGVZXHPGKNHGBPYXCHGRKGKHZGUOGKJOJPPJQGDJFHGRDYMH▸
  "XHRPGBJBGYGOY@@GLJVG@JGOVMNGOHHKGOHGVZXHPGKNHGBPYXCHGRKGKHZGUOGKJOJPPJQGDJFHGRDYMH"
affinedecryp(t,11,7)
 Affine decryption (a= 11  og b= 7 )
 Plain text:
 "DEAR@BOB@I@MISS@YOU@SO@MUCH@MEET@ME@UNDER@THE@BRIDGE@AT@TEN@PM@TOMORROW@LOVE@▸
                                                                              Done
```