`Xcas` reference card

1 How to install `Xcas`

`Xcas` is a free software (GPL), you can download it at :
`http ://www-fourier.ujf-grenoble.fr/~parisse/giac.html` CAS (Computer Algebra System) means exact, formal or symbolic calculus.

2 Interface

Interface	
`File Edit Cfg...`	is the main menu
`session1.xws` or `Unnamed`	is the name of the current session or if the session has not been saved
`?`	open the help command index
`Save`	save the session
`Config : exact real...`	open the CAS configuration
`STOP`	interrupt a computation
`Kbd`	show/hide keyboard
`X`	close the session
`1`	is a commandline

You can write your first command (click to have the cursor in the commandline) : `1+1`, then "Enter" (or "Return" depending on your keyboard). The result appears below in an expression editor, as well as a new commandline (numbered 2) for the next command. `Xcas` has different data types : integers (`2`), fractions (`3/2`), float numbers (`2.0`,`1.5`), formal parameters (`x,t`), variables (`a:=2`), expressions (`x^2-1`), functions (`f(x):=x^2-1`), lists (`[1,2,3]`), sequences (`1,2,3`), strings (`"na"`) and geometric objects.

An expression is a combination beteween numbers and variables connected with operators. A function associates a variable to an expression.
For example, `a:=x^2+2*x+1` defines an expression `a` but `b(x):=x^2+2*x+1` defines a function `b` and `b(0)=subst(a,x=0)=1`.

A matrix is a list of lists with same length, a sequence can't contains sequence.

Ponctuation symbols	
`.`	between the integer part and the decimal part
`,`	between the terms of a list or of a sequence
`;`	ends each instruction of a program
`:;`	ends an instruction whose answer will not be displayed
`!`	n ! is the factorial of n (4 !=$1 \cdot 2 \cdot 3 \cdot 4 = 24$)
`:=`	`a:=2` affectation instruction that stocks `2` into the variable `a`
`[]`	list delimitations (`L:=[0,2,4]` and `L[1]` returns `2`)
`""`	string delimitations (`C:="ba"` and `C[1]` returns `"a"`)

3 Configurations

Configurations	
`Cfg▶CAS config`	open the CAS configuration
`Cfg▶graph config`	open the default graphic configuration
`Cfg▶general configuration`	open the general configuration
`cfg` (Graph)	open the configuration of this graphic level
`Config : ...`	open the CAS configuration
`Sheet config` :	open the sheet configuration

You can change the aspect of the interface and save your changes for the next sessions using the `Cfg` menu.

4 Levels

Each session is composed of numbered levels which are : command line for cas commands, interactive geometry screen (2-d et 3-d), formal spreadsheet, turtle drawing, programm editor etc...

Levels	
`Alt+c`	new comment
`Alt+d`	new turtle graphic
`Alt+e`	new expression editor
`Alt+g`	new 2-d geometry figure
`Alt+h`	new 3-d geometry figure
`Alt+n`	new commandline
`Alt+p`	new program editor
`Alt+t`	new spreadsheet

5 Help

All commands are sorted in alphabetical order in the help index (`Help▶Index`) and several manuals with exercises in `Help▶Manuals▶...` and examples in `Help▶Examples`.

Help	
`Help▶Index`	open the command index
`Help▶Manuals▶...`	open one of manuals in your navigator
`?`	open the command index
`ce` `?`	open the command index at `ceil`
`ce` `F1`	open the command index at `ceil`
`ce` `⇄`	open the command index at `ceil`
`?ceil`	open the browser detailled help for `ceil`
`Cmds▶Real▶Base▶ceil`	print `ceil` short help in `msg` opened with `Cfg▶Show▶msg` or `Kbd` ▶ `msg`

`Xcas` reference card : basic CAS

- Type Enter to execute a commandline.
- Numbers may be exact or approx.
- Exact numbers are constants, integers, integer fractions and all expressions with integers and constants.
- Approx numbers are written with the scientific standard notation : integer part followed by the decimal point and the fractional part, optionally followed by `e` and an exponent.

Operators	
`+`	addition
`-`	substraction
`*`	mutiplication
`/`	division
`^`	power

Constants	
`pi`	$\pi \simeq 3.14159265359$
`e`	$e \simeq 2.71828182846$
`i`	$i = \sqrt{-1}$
`infinity`	∞
`+infinity` or `inf`	$+\infty$
`-infinity` or `-inf`	$-\infty$
`euler_gamma`	Euler's constant

Sequences, lists, vectors	
`S:=a,b,c`	S is a sequence of 3 elements
`S:=[a,b,c]`	S is a list of 3 elements
`S:=NULL`	S is an empty sequence
`S:=[]`	S is an empty list
`dim(S)`	returns the size of S
`S[0]`	returns the first element of S
`S[n]`	returns the $n+1$-th element of S
`S[dim(S)-1]`	returns the last element of S
`S:=S,d`	appends the element `d` at the tail of a sequence S
`S:=append(S,d)`	appends the element `d` at the tail of a list S

Strings	
`S:="abc"`	S is a string of 3 characters
`S:=""`	S is a string of 0 character
`dim(S)`	is the length of S
`S[0]`	returns the first character of S
`S[n]`	returns the $n+1$-th character of S
`S[dim(S)-1]`	returns the last character of S
`S:=S+d`	appends the character d at the tail of the string S
`"ab"+"def"`	concats the two strings and returns "abdef"

Fractions	
`propfrac`	returns integer part+fractional part
`numer getNum`	numerator of the fraction after simplification
`denom getDenom`	denominator of the fraction after simplification
`f2nd`	[numer, denom] of the fraction after simplification
`simp2`	simplifies a pair
`dfc`	continued fraction expansion of a real
`dfc2f`	converts a continued fraction expansion into a real

Usual functions			
`evalf(t,n)`	num. approx. of t with n decimals	`sign`	sign (-1,0,+1)
`max`	maximum	`min`	minimum
`round`	nearest integer	`frac`	fractional part
`floor`	greatest integer $\leq$	`ceil`	smallest integer $\geq$
`re`	real part	`im`	imaginary part
`abs`	norm or absolute value	`arg`	argument
`conj`	conjugate	`affix`	affix
`factorial !`	factorial	`binomial`	binomial coefficient
`exp`	exponential	`sqrt`	square root
`log10`	common logarithm (base 10)	`ln log`	natural logarithm
`sin cos`	sinus cosine	`csc sec`	1/sinus 1/cosine
`tan`	tangent	`cot`	cotangent
`asin`	arcsinus	`acos`	arccosine
`atan`	arctangent	`acot`	arccotangent
`sinh`	hyperbolic sinus	`cosh`	hyperbolic cosine
`asinh`	hyperbolic arcsine	`acosh`	hyperbolic arccosine
`tanh`	hyperbolic tangent	`atanh`	hyperbolic arctangent

Arithmetic on integers	
`a%p`	a mod p
`powmod(a,n,p)`	a^n mod p
`irem`	euclidean remainder
`iquo`	euclidean quotient
`iquorem`	[quotient,remainder]
`ifactor`	factorization into prime factors
`ifactors`	list of prime factors
`idivis`	list of divisors
`gcd`	greatest common divisor
`lcm`	lowest common multiple
`iegcd`	extended greatest common divisor
`iabcuv`	returns $[u, v]$ such as $au + bv = c$
`ichinrem`	chinese remainders for integers
`is_prime`	test if n is prime
`nextprime`	next pseudoprime integer
`previousprime`	previous pseudoprime integer

Transformations			
`simplify`	simplifies	`tsimplify`	simplifies (less powerful)
`normal`	normal form	`ratnormal`	normal form (less powerful)
`expand`	expands	`partfrac`	partial fraction expansion
`factor`	factorizes	`convert`	converts into a specified format

Transformations and trigonometry			
`tlin`	linearize	`tcollect`	linearizes and collects
`texpand`	expands $\exp, \ln$ and trig	`trig2exp`	trig to $\exp$
`hyp2exp`	hyperbolic to $\exp$	`exp2trig`	$\exp$ to trig

`Xcas` reference card : statistics and spreadsheet

Probabilities	
`comb(n,k)`	$\binom{n}{k} = C_n^k$
`binomial(n,k,[p])`	returns comb$(n,k) * p^k(1-p)^{n-k}$ or `comb(n,k)`
`perm(n,p)`	A_n^p
`factorial(n), n!`	`n!`
`rand(n)`	random integer p such that $0 \leq p < n$
`rand(p,q)`	random real t such that $t \in [p,q]$
`randnorm(mu,sigma`	random real t according $N(\mu, \sigma)$

1-d statistics	
`mean`	mean of a list
`median`	median of a list
`quartiles`	[min,quartile1,median,quartile3,max]
`boxwhisker`	whisker boxes of a statistical series
`variance`	variance of a list
`stddev`	standard deviation of a list
`histogram`	histogram of its argument

2-d statistics	
`polygonplot`	polygonal line
`scatterplot`	scattered points
`polygonscatterplot`	polygonal pointed line
`covariance`	covariance of 2 lists
`correlation`	correlation of 2 lists
`exponential_regression`	(m,b) for exponential fit $y = be^{mx}$
`exponential_regression_plot`	graph of the exponential fit $y = be^{mx}$
`linear_regression`	(a,b) for linear fit $y = ax + b$
`linear_regression_plot`	graph of the linear fit $y = ax + b$
`logarithmic_regression`	(m,b) for logarithmic fit $y = m\ln(x) + b$
`logarithmic_regression_plot`	graph of the logarithmic fit $y = m\ln(x) + b$
`polynomial_regression`	$(a_n, ..a_0)$ for polynomial fit $y = a_nx^n + ..a_0$
`polynomial_regression_plot`	graph of the polynomial fit $y = a_nx^n + ..a_0$
`power_regression`	(m,b) for power fit $y = bx^m$
`power_regression_plot`	graph of the power fit $y = bx^m$

Statistic commands may be typed in a commandline or selected from the `Cmds▶Proba_stats` menu. They may be selected from the `Graphic▶Stats` menu using dialog boxes.
The easiest way is however to open a spreadsheet enter data there, select the data with the mouse, open the spreadheet `Maths` menu and fill the dialog boxes.

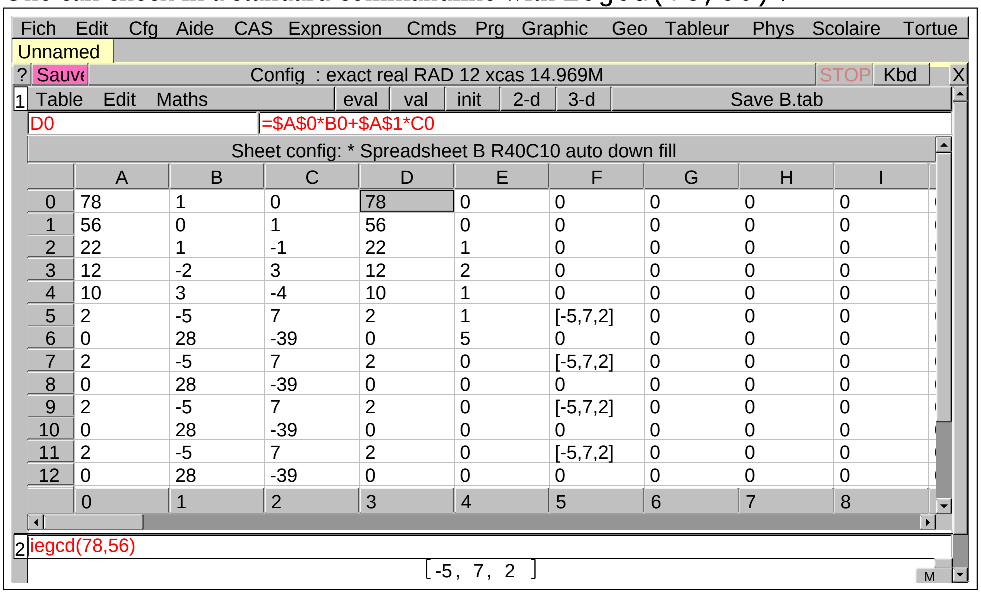
The Xcas spreadsheet is a symbolic spreadsheet (in addition to numeric values and formula (beginning with =), cells may contain exact value, complex numbers, expressions, ...) where Xcas commands and user-defined functions may be used. Note that litteral entries must be quoted as strings, for example "Result", otherwise they will be parsed as identifiers or may generate errors. The Xcas spreadsheet uses standard conventions (columns are refered with letters starting at `A`, rows with numbers starting at 0, references are relative except if the column or row number is prefixed with `$`). Note that :

- the `Table, Edit, Maths` menu may be obtained by a right-click mouse
- the `eval val 2-d 3-d` buttons (reeval the spreadsheet, show the value instead of formula, show 2-d or 3-d graph displaying cells with a graphic object value in a window)
- the “goto” input-value (top-left) let you go to a cell or select a cell range if you fill it in. It is filled if you make a mouse event
- the commandline to input cells values or formulas
- the configuration button : shows the current config, click to change the sheet configuration : you may select to view all 2-d graphic objects of the spreadsheet below or right to the sheet (Landscape mode)

Example : extended gcd, given a and b find u and v such that $au + bv$ =gcd(a, b)

- Enter the value of a and b in `A0` and `A1` for example 78 and 56
- We will fill column `A` with remainders r_n, set `A2` to `=irem(A0,A1)` and copy down (Ctrl-d).
- Column `E` will contain the quotients, set `E2=iquo(A0,A1)` and copy down
- Columns `B` and `C` will contain values of u_n and v_n such that $au_n + bv_n = r_n$, enter 1 and 0 for `B0`, `C0`, 0 and 1 for `B1` and `C1`, `=B0-E2*B1` for `B2`, copy down `=C0-E2*C1` for `C2`, copy down
- Column `D` is $au_n + bv_n$, hence should be identical to column `A`, set `D0` to `=B0*$A$0+B1*$A$1` and copy down
- Column `F` will contain the answer or 0, set `F0` to :
  `=if A0==0 then [B0,C0,D0] else 0 fi` and copy down.

One can check in a standard commandline with `iegcd(78,56)` :



`Xcas` reference card : Algebra

Polynomials	
`normal`	normal form (expanded and reduced)
`expand`	expanded form
`ptayl`	Taylor polynomial
`peval horner`	evaluation using Horner's method
`genpoly`	polynomial defined by its value at a point
`canonical_form`	canonical form of a second degree polynomial
`coeff`	coefficient or list of coefficients
`poly2symb`	list polynomial to symbolic polynomial
`symb2poly`	symbolic polynomial to list polynomial
`pcoeff`	polynomial from it's roots
`degree`	degree
`lcoeff`	coefficient of the monomial of highest degree
`valuation`	degree of the monomial of lowest degree
`tcoeff`	coefficient of the monomial of lowest degree
`factor`	factorizes a polynomial
`cfactor`	factorizes a polynomial on $\mathbb{C}$
`factors`	list of irreducible factors and multiplicities
`divis`	list of divisors
`collect`	factorization on the coefficients field
`froot`	roots with their multiplicities
`proot`	approx. values of roots
`sturmab`	number of roots in an interval
`getNum`	numerator of a rational fraction (unsimplified)
`getDenom`	denominator of a rational fraction (unsimplified)
`propfrac`	returns polynomial integer part + fractional part
`partfrac`	partial fraction expansion
`quo`	euclidean quotient
`rem`	euclidean remainder
`gcd`	greatest common divisor
`lcm`	lowest common multiple
`egcd`	extended greatest common divisor
`chinrem`	chinese remainder
`randpoly`	random polynomial
`cyclotomic`	cyclotomic polynomial
`lagrange`	Lagrange polynomial
`hermite`	Hermite polynomial
`laguerre`	Laguerre polynomial
`tchebyshev1`	Tchebyshev polynomial (1st type)
`tchebyshev2`	Tchebyshev polynomial (2nd type)

Matrices	
`M:=[[a,b,c],[f,g,h]]`	M is a matrix with 2 rows and 3 columns
`dim(M)`	returns dimensions as a list [nrows, ncols]
`M[0]`	returns the first line of M
`M[n]`	returns the $n+1$-th line of M
`row(M,n)`	returns the $n+1$-th line of M
`col(M,n)`	returns the $n+1$-th column of M
`M[dim(M)[0]-1]`	returns the last line of M
`M[n..p]`	returns the sub-matrice of M with lines in $[n..p]$
`append(M,[d,k,l])`	appends the line $[d,k,l]$ at the end of M
`M[dim(M)[0]]:=[d,k,l]`	appends the line $[d,k,l]$ at the end of M
`border(M,[d,k])`	appends the column $[d,k]$ at the end of M

Operators on vectors and matrix	
`v*w`	scalar product
`cross(v,w)`	dot product
`A*B`	matrix product
`A .* B`	term by term product
`1/A`	inverse
`tran`	transposes a matrix
`rank`	rank
`det`	determinant
`ker`	kernel basis
`image`	image basis
`idn`	identity matrix
`ranm`	matrix with random coefficients

Linear systems	
`linsolve`	linear system solver
`rref`	Gauss-Jordan reduction
`rank`	rank
`det`	determinant of a system

Matrix reduction	
`jordan`	eigenvalue/characteristic vectors (Jordan reduction)
`pcar`	characteristic polynomial
`pmin`	minimal polynomial
`eigenvals`	eigenvalues
`eigenvects`	eigenvectors

Xcas reference card : Calculus

Derivatives	
diff(E) or E'	expression derivative of an expression E with respect to x
diff(E,t) or (E,t)'	expression derivative of an expression E with respect to t
diff(f) or f'	function derivative of the function f
diff(E,xn,ym)	expression partial derivative $\frac{\partial E}{\partial x^n \partial y^m}$ of an expression E
grad	gradient
divergence	divergence
curl	rotationnal
laplacian	laplacian
hessian	hessian matrix

Limits and series expansion	
limit(E,x,a)	limit of an expression E at $x = a$
limit(E,x,a,1)	limit of an expression E at $x = a^+$
limit(E,x,a,-1)	limit of an expression E at $x = a^-$
series(E,x=a,n)	series expansion of E at a with relative order=n
taylor(E,a)	series expansion of E at $x = a$ with relative order=5

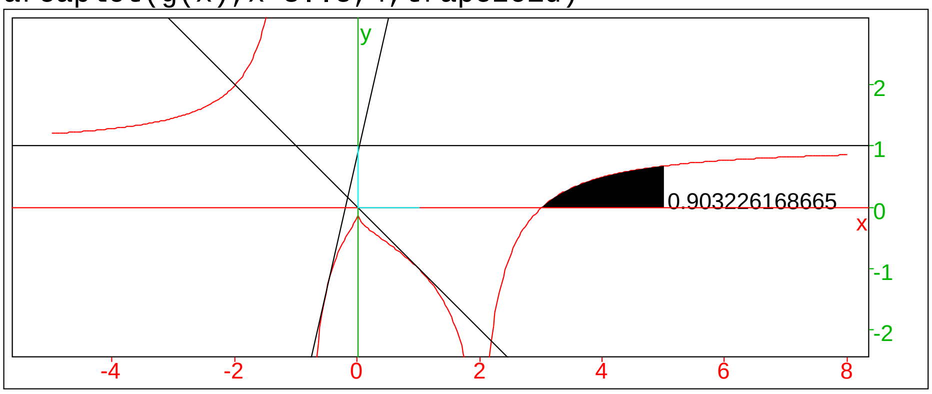
Integrals	
int(E,x)	antiderivative of an expression E
int(f)	antiderivative function of a function f
int(E,x,a,b)	integration of an expression E from $x = a$ to $x = b$
romberg(E,x,a,b)	approximate value of int(E,x,a,b)

Equations	
solve(eq,x)	exact $\mathbb{R}$-solution of a polynomial equation
solve([eq1,eq2],[x,y])	exact $\mathbb{R}$-solution of a list of polynomial equations
csolve(eq,x)	exact $\mathbb{C}$-solution of a list of polynomial equations
csolve(eq1,eq2],[x,y])	exact $\mathbb{C}$-solution of a list of polynomial equations
fsolve(eq,x=x0)	approx solution of an equation (x0=xguess)
fsolve([eq],[var],[val])	approx solution of a list of equations(val=xguess)
newton	Newton's method
linsolve	linear system solver
proot	approx roots of a polynomial

Ordinary Differential Equations (ODE)	
desolve	exact solution of an ODE
odesolve	approx solution of an ODE
plotode	plot the approx solution of an ODE
plotfield	plot the field of an ODE
interactive_plotode	plot an ODE field and solutions through mouse clicks

Curves	
`plot`	plots a 1-d expression
`tangent`	draws the tangent lines to a curve
`slope`	slope of a line
`plotfunc`	plots a 1-d or 2-d expression
`...,color=...)`	chooses the color of a plot
`areaplot`	displays the area below a curve
`plotparam`	plot a parametric curve
`plotpolar`	plot a polar curve
`plotimplicit(f(x,y),x,y)`	implicit plot of $f(x, y) = 0$

Example Define the function f over $\mathbb{R} - \{-1, 0, 1, 2\}$ by : $f(x) = \frac{\ln(|2 - x|)}{\ln(|x|)}$.
We will show that f can be extended to a continuous function on $\mathbb{R} - \{-1, 2\}$, draw the graph of f, and the tangents at $x = -1/2$, $x = 0$ and $x = 1$. We will give an approximate value of the area between $x = 3, x = 5, y = 0$ and the curve, using the trapezoid rule with 4 subdivisions.
Input : `f(x) :=ln(abs(x-2))/ln(abs(x))`
`limit(f(x),x,1)` answer `-1`. `limit((f(x)+1)/(x-1),x,1)` answer `-1`
Hence we can extend f at $x = 1$ and the slope of the tangent at (1,-1) is -1
`limit(f(x),x,0)` answer `0`, `limit(f(x)/x,x,0,1)` answer `-infinity` and `limit(f(x)/x,x,0,-1)` answer `+(infinity)`. Hence we can extend f at $x = 0$ and the tangent at (0,0) is the y-axis
`limit(f(x),x,-1)` answer `infinity`, so $x = -1$ is an asymptote.
`limit(f(x),x,2)` answer `-infinity`, so $x = 2$ is an asymptote.
`limit(f(x),x,inf)`, `limit(f(x),x,-inf)` answer `(1,1)`. We conclude that the line $y = 1$ is an asymptote to the curve.
To extend f to a continuous function defined on $\mathbb{R} - \{-1, 2\}$, input :
`g :=when(x==0,0,when(x==1,-1,f(x)))`
To get the graph, input : `G :=plotfunc(g(x),x=-5..8,color=red);`, `line(y=1),tangent(G,-1/2),line(1-i,slope=-1),`
`areaplot(g(x),x=3..5,4,trapezoid)`



In order to approximate the area with 4 trapezoids, type :
`Digits :=3;0.5*(f(3)/2+f(3.5)+f(4)+f(4.5)+f(5)/2)`
it will return 0.887.

Enter `areaplot(g(x),x=3..5)` to compute the area with Romberg's method (an acceleration of the trapezoid method) ; 3 digits are displayed. For more digits, enter `romberg(g(x),x,3,5)`, it returns 0.903226168665 if `Digits :=12;`.

`Xcas` reference card : geometry

2-d geometry	
`point`	point given by its coordinates or its affix
`...,display=...)`	attributs for a graphic object (last argument)
`legend="..."`	set the legend of a graphic object
`segment`	returns the segment given by 2 points
`line(A,B)`	returns the line AB
`line(a*x+b*y+c=0)`	returns the line $ax + by + c = 0$
`triangle(A,B,C)`	returns the triangle ABC
`bissector(A,B,C)`	returns the bissector of $\widehat{BAC}$
`angle(A,B,C)`	returns the angle measure (in rad or deg) of $\widehat{BAC}$
`median\_line(A,B,C)`	draws the median-line through A of the triangle ABC
`altitude(A,B,C)`	draws the altitude through A of the triangle ABC
`perpen\_bisector(A,B)`	draws the perpendicular bisector of AB
`square(A,B)`	draws the direct square of side AB
`circle(A,r)`	draws the circle with center A and radius r
`cercle(A,B)`	draws the circle with diameter AB
`radius(c)`	gives the radius of the circle c
`center(c)`	gives the center of the circle c
`distance(A,B)`	returns the distance from A to B (point or curve)
`inter(G1,G2)`	returns the list of points in $G1 \cap G2$
`inter_unique(G1,G2)`	returns one of the points in $G1 \cap G2$
`assume`	add a symbolic parameter (or an hypothesis)
`element`	add a numeric parameter
`polygon`	draws a polygon
`open\_polygon`	draws an open polygon
`coordinates`	coordinates of a point
`equation`	cartesian equation
`parameq`	parametric equation
`homothety(A,k,M)`	image of M by the homothety of center A and coefficient k
`translation(B-A,M)`	image of M by the translation $\overrightarrow{AB}$
`rotation(A,t,M)`	image of M by the rotation of center A and of angle t
`similarity(A,k,t,M)`	image of M by the similarity of center A, coefficient k and angle t
`reflection(A,M)`	image of M by the reflection (w.r.t. point or line A)

You can either type a geometric command with the keyboard, or select it in the `Geo` menu. Additionnally, inside a figure, you can select a geometric object shape in `Mode`, and click with the mouse to construct it. Clicks will by default build geometric objects with approx coordinates unless you uncheck $\boxed{\sim}$. If you choose `Landscape`, the graphic screen will be larger and the commandlines will be below the figure. If you modify one commandline and press Enter, all the following commandlines will be re-evaluated and the figure will be synchronized.

Example, draw a triangle ABC, the perpendicular bissector to AB and the circumcircle to ABC.

– Choose `Mode▶Polygon▶triangle`. Click at the desired position for the point A, move the mouse (a segment joining to the first point is displayed) and click at the desired second point position, move the mouse (a triangle following the mouse is displayed) and click again at the desired position for C. The triangle is now constructed and a few commandlines appear at the left of the figure (`A:=point(...)`, ...).
– Choose `Mode▶Line▶perpen_bisector`. Click on `A`, move the mouse to `B` (a perpendicular bisector will follow the move), click, the perpendicular bissector to AB is constructed and the corresponding commandline is added at the left of the figure
  `E:=perpen_bissector(A,B,display=0)`
– Choose `Mode▶Circle▶circumcircle`, click on A, move, click on B, move (a circle follows the mouse move) and click on C, the circumcircle is constructed and the corresponding commandline is added at the left of the figure
  `F:=circumcircle(A,B,C,display=0`
– Choose `Mode▶Pointer`. In this mode you can drag one of the point A, B or C and see the consequences on the figure.

Alternatively, one can also enter the commands directly in the commandline at the left of the figure

```
A:=point(-1,2);
B:=point(1,0);
C:=point(-3,-2);
D:=triangle(A,B,C);
E:=perpen_bisector(A,B);
F:=circumcircle(A,B,C);
```

3-d geometry	
`plotfunc`	surface $z = f(x, y)$ given by $f(x, y)$
`plotparam`	parametric surface or 3-d parametric curve
`point`	point given by the list of its 3 coordinates
`line`	line given by 2 equations or 2 points
`inter`	intersection
`plane`	plane given by 1 equation or 3 points
`sphere`	sphere given by center and radius
`cone`	cone given by vertex, axis and half-angle
`cylinder`	cylinder given by axis and radius, [altidude]
`polyhedron`	polyhedron
`tetrahedron`	regular direct tetrahedron or pyramid
`centered_tetrahedron`	regular direct tetrahedron
`cube`	cube
`centered_cube`	centered cube
`parallelepiped`	parallelepiped
`octahedron`	octahedron
`dodechedron`	dodecahedron
`icosahedron`	icosahedron

`Xcas` reference card : programmation

1. How to write a function

You have to :

- choose a syntax, we describe here the `Xcas` syntax :
  - either with the menu `Cfg▶Mode(syntax)▶xcas`,
  - or press on the button `Config :..` to open the CAS configuration window and choose `Xcas` in `Prog style`,
- open a program editor either with `Alt+p`, or with the menu `Prg▶New program`. Note the `:;` at the end.
- write the function with the instructions separated by `;`
  Check that the name of the function, arguments and variables are not reserved keywords (they should be written in black, programming key words are in blue and the commandnames in brown), this can be achieved by beginning the function name by a Capital,
- click `OK` or press `F9` to compile the program.
- you are now ready to test your program in a commandline, write it's name followed by parenthesis, with the argument values separated with commas.

2. The `add` menu of a program editor

This menu may be used to remind the syntax of a function, of a test and of loops.

Syntax of a function :

```
f(x,y):={
  local z,a,...,val;
  instruction1;
  instruction2;
  val:=...;
  .....
  instructionk;
  return val;
}:;
```

Example, Bezout's algorithm :

```
Bezout(a,b):={
 local la,lb,lr,q;
  la:=[1,0,a];
  lb:=[0,1,b];
 while (b!=0){
   q:=iquo(la[2],b)
   lr:=la+(-q)*lb;
   la:=lb;
   lb:=lr;
   b:=lb[2];
 }
 return la;
}:;
```

3. Compilation If compilation is successfull, you should see `Done` (if the program ends with `:;`) or the translation of your program

For the example, click `OK` (or `F9`), you should obtain `// Parsing Bezout// Success compiling Bezout` and `Done`. Then input `Bezout(78,56)` which should return `[-5,7,2]` (-5*78+7*56=2=gcd(78,56)).

4. Step by step You can run a program line by line (for debugging or pedagogical illustration) using the debug command, like e.g. :

`debug(Bezout(78,56))`

A new window opens, press `sst` (shortcut F5) to run the next instruction.

Instructions	
affectation	a:=2;
input expression	input("a=",a);
input string	textinput("a=",a);
output	print("a=",a);
returned value	return a;
quit a loop	break;
alternative	if <condition> then <inst> end_if;
	if <condition> then <inst1> else <inst2>end_if;
for loop	for j from a to b do <inst> end_for;
	for j from a to b by p do <inst> end_for;
repeat loop	repeat <inst> until <condition>;
while loop	while <condition> do <inst> end_while;
do loop	do<inst1> if (<condition>)break;<inst2>end_do;

C-like instructions	
affectation	a:=2;
input expression	input("a=",a);
input string	textinput("a=",a);
output	print("a=",a);
returned value	return(a);
stop	break;
alternative	if (<condition>) {<inst>};
	if (<condition>) {<inst1>} else {<inst2>};
for loop	for (j:= a;j<=b;j++) {<inst>};
	for (j:= a;j<=b;j:=j+p) {<inst>};
repeat loop	repeat <inst> until <condition>;
while loop	while (<condition>) {<inst>};
do loop	do <inst1> if (<condition>) break;<inst2> od;

Ponctuation symbols	
.	between the integer part and the decimal part
,	between the terms of a list or of a sequence
;	ends each instruction of a program
:;	ends an instruction whose answer will not be displayed
!	n ! is the factorial of n

<table>
<tr><th colspan="4">Operators</th></tr>
<tr><td>+</td><td>addition</td><td>-</td><td>substraction</td></tr>
<tr><td>*</td><td>mutiplication</td><td>/</td><td>division</td></tr>
<tr><td>^</td><td>power</td><td>a mod p</td><td>a modulo p</td></tr>
<tr><td>==</td><td>tests equality</td><td>!=</td><td>tests difference</td></tr>
<tr><td><</td><td>strictly less</td><td><=</td><td>less or equal</td></tr>
<tr><td>></td><td>strictly greater</td><td>>=</td><td>greater or equal</td></tr>
<tr><td>||, or</td><td>boolean infixed operator</td><td>\&\&, and</td><td>boolean infixed operato</td></tr>
<tr><td>not</td><td>logical not</td><td>!(..)</td><td>logical not</td></tr>
<tr><td>true</td><td>is the boolean true or 1</td><td>false</td><td>is the boolean false or 0</td></tr>
</table>

Xcas reference card : the turtle

Moves	
clear efface	clears the screen
forward	forward
backward	back
jump	jump
side_step	side step
turn_left	turns left
turn_right	turns right

Colors	
pen	gives the color of the pencil.
hide_turtle	hides the turtle
show_turtle	shows the turtle
draw_turtle(n)	draws the turtle, the shape is filled if n is 0

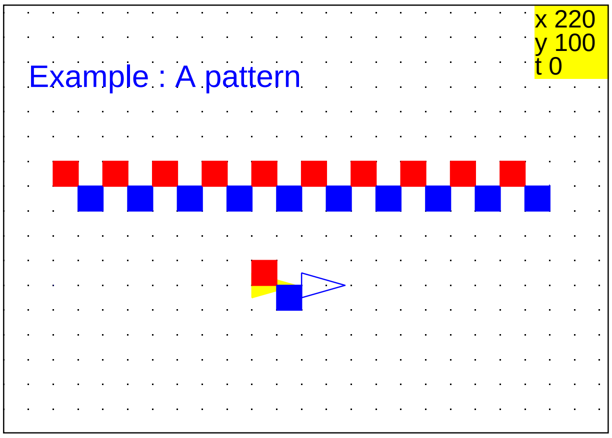
Shapes	
turtle_circle	circle or arc of circle
filled_triangle	filled triangle
filled_rectangle	filled rectangle (or square, rhombus, parallelogram)
disc	filled circle (or angle sector) tangent to the turtle.
centered_disc	circle (or angle sector) with the turtle as center
filled_polygon	fill the polygon that has just been drawn before

Legends	
write_string	write on the screen at the turtle position
signature	put a signature at the screen left botton

Turtle programs	
if <c> then <inst> end_if	$< inst >$ are done if condition $< c >$ is true
if <c> then <inst1> else <inst2> end_if	$< inst1 >$ (or $< inst2 >$) are done if condition $< c >$ is true (or false)
repeat_turtle n,<i1>,<i2>	repeat n times the instructions $< i1 >, < i2 >$
for j from j1 to j2> do <inst> end_for	$< inst >$ are done with an iteration variable j with a step=1 for the iteration
for j from j1 to j2 by p do <inst> end_for	$< inst >$ are done with an iteration variable j with a step p for the iteration
while <c> do <inst> end_while	$< inst >$ are done while condition $< c >$is true
return	return the value of a function
input(a)	get a value from the keyboard, stores it in a,
textinput(a)	get a string from the keyboard, stores it in a
write("toto",a,b)	write functions a, b in a file named $toto$
read("toto")	read the functions from the file named $toto$

Position	
`position`	give the turtle position or change it's position
`cap`	give the turtle direction or change it's direction
`towards`	put the turtle direction to a point.

There should be at most one turtle picture level in a given session.
To drive the turtle, you can write a command, use the `Turtle` menu, or click on a button below the turtle picture, each button is named after the first letters of a turtle command (`cr` button displays also all the colors). At the right of the screen, there is a small editor which records all your commands (called "recording editor"). You may change commands there and synchronize the turtle picture by running all these commands (press `F7`).



This picture is obtained by repetition of a pattern, which is isolated above (turtle start position is in yellow). Let's make first the pattern : open a turtle level (`Alt+d`) then enter in the commandlines at the left of the picture :

```
pen 1;
filled_rectangle ;
jump ;
turn_right ;
pen 4;
filled_rectangle ;
turn_left ;
jump ;
```

You can enter most commands by pressing buttons `pe, fr, ju, tr, ....` The commands are echoed in the recording editor at the right of the picture. If you make a mistake, modify the command in the small editor and press `F7` to synchronize.
Once the commands are all entered, open a program editor (`Alt+p`) and copy-paste the text from the small editor to the program editor. Replace `efface;` at the beginning by `motif():={` then add a `}` at the end before `:;` and press `F9`.
Enter in a commandline at the left of the picture :

```
repeat_turtle 10, motif()
```

You can move or zoom the picture with mouse drags and with the mousewheel.

This example shows how to make a complex picture by decomposing it in simple tasks, and how to properly use the recording editor to extract a procedure from a picture built step by step.