Glossary of commands

This file holds a list of all the commands used in MATH1110 Calculus 1.

Remember that if you are unsure of how to use a command, you can type its name with a ? and Sage will recall information regarding the command.

Contents:

A

abs ~ absolute value of a number or expression.

```
sage: abs( -7 )
```

C

clear_vars ~ used to reset all previously ran and stored variables.

No arguments needed. If variables specified, only those within the parentheses will be cleared.

```
sage: clear_vars()
```

D

derivative ~ compute the derivative of a function.

Required argument: (1) function that is being differentiated.

Second and third arguments: (2) the variable that is being differentiated with respect two, (3) the order of the derivative.

```
sage: derivative(f, x, 1)
```

desolve ~ solve a differential equation for the function being derived.

Required arguments: (1) differential equation, (2) function that is being solved for

Additional argument: (3) initial conditions given for the variable and the function

```
sage: desolve( eqn, y, ics = [1, 2] )
```

diff ~ alternative for derivative (see above).

```
sage: diff(f, x, 1)
```

E

expand ~ expand a factored expression.

Required argument: expression that is being expanded.

```
sage: expand( (x-1)*(x+1) )
```

F

factor ~ factor an expression.

Required argument: expression that is being factored.

```
sage: factor( x^2 - 1 )
```

factorial ~ takes the factorial (!) of a number or expression.

```
sage: factorial(4)
```

find_local_minimum ~ find the local minimum value of a function in a given range.

Required arguments: (1) expression/function, (2) lower bound, and (3) upper bound within which the minimum is being located.

```
sage: find_local_minimum( f, 0, 10 )
```

find_root ~ find the roots of an expression (i.e. where the function is equal to zero).

Required arguments: (1) expression for which the roots are being located, (2) lower bound, and (3) upper bound within which a root is being located.

```
sage: find_root( f, -5, 0 )
```

function ~ define a symbol as a function.

Required argument: symbol that is being defined as a function.

In the example below, f is defined as a function of a (previously defined) variable x.

```
sage: f = function('f')(x)
```

I

infinity ~ positive infinity, also denoted 'oo'.

```
sage: infinity
```

Implicit_plot ~ plot an implicitly defined curve.

Required arguments: (1) the equation that is being plotted, (2) the first variable and its domain, (3) the second variable and its domain.

Additional arguments: see options listed for the 'plot' funciton.

```
sage: implicit_plot( x*y^3 == tan(y) , (x, -10, 10), (y, -10, 10) )
```

integral ~ compute the integral of a function.

Required arguments: (1) the function that is being integrated, (2) the variable that is being integrated with respect to.

Additional arguments: (3) the lower bound of the definite integral, (4) the upper bound of the definite integral.

```
sage: integral(f, x)
```

integrate ~ alternative for integral (see above).

```
sage: integrate(f, x)
```

L

lim ~ alterative for limit (see below).

```
sage: lim(f, x = oo)
```

limit ~ take the limit of an expression.

Required arguments: (1) the function for which the limit is being evaluated, (2) the independent variable assigned to the value which it is approaching.

```
sage: limit(f, x = oo)
```

line ~ create a line between two points.

Required argument: the location of the start and end of the line, represented as two ordered pairs in a larger set of square brackets.

Additional arguments: linestyle, color, alpha, and soforth.

```
sage: line( [(-pi, 5), (pi, 5)], color = 'green' )
```

ln ~ the natural logarithm function.

```
sage: ln(x)
```

log ~ the logarithmic function.

The default base is e, **not** base 10, making it equivalent to the natural logarithm.

Required argument: number or expression within the logarithm.

Second argument: the base.

```
sage: log(x, 10)
```

N

N ~ express a number as a decimal.

Additional argument: number of digits desired.

```
sage: N( pi, digits = 4 )
```

O

Operators, assorted

- + is the addition operator
- - is the subtraction operator
- * is the multiplication operator
- / is the division operator
- ^ (or **) is the exponent operator
- sqrt(_) is the square root operator

- (), [], {} are parentheses, square brackets, and curly brackets

- = is the assignment operator
- == is an equality
- > is greater than
- \< is less than
- >= is greater than or equal to
- \<= is less than or equal to

P

parametric_plot ~ plot a parametrically defined curve.

Required arguments: (1) parametric functions, (2) parameter (variable) and its domain.

Additional arguments: see options listed for the 'plot' function.

```
sage: parametric_plot( (sin(t), cos(t)), (t, -pi, pi) )
```

plot ~ create a graph of an expression.

Required argument: (1) function that is being plotted.

Additional arguments: (2) independent variable, (3) the lower bound of domain, (4) the upper bound of domain, and the scaling on the y-axis (ymin / ymax).

Graphs can be customized with titles, legends, text, and lines of different colours, styles, thicknesses, etc.

See https://doc.sagemath.org/html/en/reference/plotting/sage/plot/plot.html (https://doc.sagemath.org/html/en/reference/plotting/sage/plot/plot.html) for more options.

```
sage: plot(f, x, -10, 10, ymin = -10, ymax = 10, color = 'violet', title = 'Plot of
a function')
```

point ~ create a point or multiple points on a plot.

Required argument: location of point(s), inputted as an ordered pair.

Additional arguments: color, size, and shape of the point.

In the first example, only one point is made at (1, 1). The second example demonstrates correct syntax when creating multiple points.

```
sage: point( [1, 1], size = 30, color = 'red')
sage: point( [ [1, 1], [2, 2], [3, 3] ], size = 40, color = 'black')
```

polar_plot ~ plot a curve in polar coordinates.

Required arguments: (1) the expression being plotted, (2) the independent variables with its domain.

Additional arguments: see options listed for the 'plot' function.

```
sage: ploar_plot( sin(theta), (theta, -2*pi, 2*pi) )
```

print ~ print desired numbers, symbols, or text in the code output.

```
sage: print('This is a piece of text.')
```

R

reset ~ wipe any stored meaning of a previously defined symbol(s).

No required arguments. If none are specified, all previously defined symbols will be cleared. In the example below, only the variable x will be wiped.

```
sage: reset('x')
```

round ~ round a number to a specified decimal place.

Required argument: the number being rounded.

Second argument: the number of decimal places desired.

```
sage: round( sqrt(2), 3)
```

S

show ~ display input in nicer script.

```
sage: show( f(x) )
```

solve ~ solve an equation for one variable.

Required arguments: (1) equation, (2) variable to be isolated.

```
sage: solve( x^2 - 1 == 0, x)
```

sqrt ~ take the square root of a number and/or variable.

Required argument: item being rooted.

```
sage: sqrt( 81 )
```

sum ~ add a sequence of terms together.

Required arguments: (1) the expression to be summed, (2) the variable, (3) starting value, (4) ending value.

```
sage: sum( x^2, x, 0, 10 )
```

T

Trigonometry functions, assorted

```
sin(x), cos(x), tan(x)

csc(x), sec(x), cot(x)

arcsin(x), arccos(x), arctan(x)

arccsc(x), arcsec(x), arccot(x)
```

V

var ~ define a symbol as a variable.

Required argument: symbol that is being defined as a variable

```
sage: x = var('x')
```