Arithmetic

`a + b`	*addition*
`a - b`	*subtraction*
`a * b`	*multiplication*
`a / b`	*division*
`a /. b`	*integer division*
`a^1|2`	*exponentiation to fractional power*

Comparisons

`a < b`	*less than*	
`a <= b`	*less than or equal to*	≤
`a == b`	*equal to*	
`a <> b`	*not equal to*	≠
`a >= b`	*greater than or equal to*	≥
`a > b`	*greater than*	

Logical operations

`a and b`	*logical AND*
`a or b`	*logical OR*
`a xor b`	*exclusive OR*
`not a`	*logical NOT*

Data types

`any`	*any type*
`bits`	*bit string*
`bytes`	*byte string*
`changelist`	*find/replace list*
`color`	*color*
`func`	*function pointer*
`image`	*bitmap image*
`meas`	*physical measurement*
`num`	*number or enum*
`ptr`	*pointer*
`regexp`	*regular expression pattern*
`sound`	*sound effect*
`str`	*character string*
`tree`	*a tree*
`video`	*movie*
`yesno`	*Y, N, U or ERR*

Definitions and flow of control statements

```
assets (local:name) (remote:name)
    folder:name into:dest (pattern:str)
          index: head|tail|seq|name
    file:name (url:"...") label:name

const name (:type) = expression
type name = type_expression
var name (:type) (= expression)

family name base:name (abbrev: name)
    (dimensions: unit^n | n | nonlinear)
unit doz family:scalar ratio:1 doz = 12 each

record name
    field1 (:type)
    :recordname

if expression
    ..statements
elif expression
    ..statements
else
    ..statements

case expression
| 10, 12
    ..statements
|>10
    ..statements
else
    ..statements

loop index:id path:id val:id label:id
    count:id rev:yesno root:yesno kind:kind
    from:expr to:expr (swap:yesno) (by:expr)
    across:tree across^2:tree across^3:tree
    top_down:tree bottom_up:tree
    recursive:tree via:field
    sort: (index|val|field: id) (func: id)
    while:expr until:expr reps:expr trap:yesno

continue (loopname)
exit (loopname)
return (expr)
nop
log (+/-) "msg" (on:flag) (cat:ident (level:num))
```

Single value operations

`=`	*copy to left*	⇐
`=>`	*copy to right*	⇒
`+=>`	*add then copy*	+⇒
`-=>`	*subtract then copy*	-⇒
`*=>`	*multiply then copy*	*⇒
`/=>`	*divide then copy*	/⇒
`&=>`	*concatenate then copy*	&⇒

```
append expr => dest
insert expr => dest
prepend expr => dest
link a.rel <=> b.rel          .LINK follows
swap src <=> dest

dec dest
inc dest
toggle dest
touch dest
adr dest                      take address of a subtree
```

Tree operations

dest `<===` *src*	*copy tree to left*	⇚
`copy` *src* `===>` *dest*	*copy tree to right*	⇛
`prepend` *src* `===>` *dest* (`index:`*ident*)		
`append` *src* `===>` *dest* (`index:`*ident*)		
`insert` *src* `===>` *dest*		
`merge` *src* `===>` *dest*		
`move` *src* `===>` *dest*		
`swap` *src* `<===>` *dest*		⇚⇛

```
clear  ===> dest
remove ===> dest
renum  ===> dest
trunc  ===> dest
```

Other operations

f `|>` *g* `|>` *h*	*function chaining*	➤
path `^^`	*pointer indirection*	↑
`ditto`	*repeat function parms*	

Calculation and drawing blocks

```
calc name ( `description`
    parm: type `description`
    ) : returntype

calc name ditto  // repeat the same parameters as function above
```

```
draw name
    layer axes:kind area:box inset:spec skew:expr
          rotate:expr translate:expr matrix:matrix
          pin:expr dpi:expr
```

Plain draw block with optional sub-layer

```
(horz|vert) (slice|scroll) name
    add expr units funcname (order:expr)
    skip expr units
```

Subdivide into horizontal or vertical slices (stack scrolls)

```
grid name
    horz slice
      add expr units funcname (order:expr)
      skip expr units
    vert slice
      add expr units funcname (order:expr)
      skip expr units
    under
          // optional code to draw underneath grid
    cell
          // code to draw each cell
    over
          // optional code to draw on top of table
```

Create a 2-dimensional grid

```
report name
    add expr units    // define raw columns
    skip expr units
    rowkind name      // define row kinds
      background
            // background drawing..
      span expr
            // drawing for this row section..
      skip expr

    build     // now build the rows
      add expr units row_id:expr (fieldname:expr)...
      skip expr units

track (eventkind)
    ...tracking logic for draw block...
```

Create a tabbed report that has a fine columnar grid, and each row can use a different set of the micro-columns

Block subdivision Units

```
al -- aliquots (proportions)
pc -- picas (1/12 of an inch, 2mm)
pt -- points (1/72 of an inch, 1/3mm)
px -- device pixels
```

Punctuation in identifiers

`$`, `_` (underscore), `*` (export marker suffix)

Comments

```
//          line comment
--          line comment
---         documentation comment
====        line comment

/* ... */   block comment
(* ... *)   block comment alternate

`comment`   metaprogramming comment
```

Required first line

```
beads level num (program | library
| monitor | system) name (export_all)
(ver literal) (title literal)
```

Preprocessor commands

```
@alias name = tokens...
@favorite
@index  "cat" / "subcat"
@option ..compiler options..
@partial func1(a) = func2(arg1, a, arg2)

@+          continue a line until @-

@if condition @then code
@elif condition @then code
@else code @endif

@< @>       indent, dedent for generated code
@;          line break for generated code
@(  @)      〔 〕 nested parentheses
```

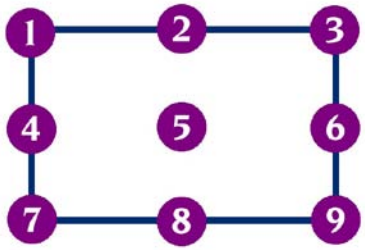
Alternate braces

```
@a{ @a}   【 】
@b{ @b)   « »
@c{ @c}   〈 〉
@d{ @d}   《 》
@e{ @e}   「 」
```

To calculate a rectangle given a series of constraints:

solve_rect(*basis:a_rect, pin, cx, cy, dx, dy, top_left, left, top, right, bottom, width, height, aspect, inset, inset_n, inset_s, inset_e, inset_w, inset_x, inset_y***):a_rect**

Rectangle pin positions



To calculate a point given a basis rectangle:

solve_point(*basis:a_rect, pin:num=5***):a_xy**

To calculate a value using linear interpolation, given an input value, an input range, and output range:

interpolate(*value, input1, input2, output1, output2, roundflag=N, clamp=N***):num**

To convert a number into a string with optional overrides for thousands mark, decimal point, length, parentheses for negative, etc.:

to_str(*val, base=10, currency=N, currency_cc:="$", decimal_cc=".", thou=N, thou_cc=",", round_k=N, max=999, min=1, neg_paren=N, pos_plus=N, percent=N, digits=U, show_u=N, zero_pad=N***):str**

To draw text on the screen:

draw_str(*box:a_rect, text:str, xy, backcolor, color=BLACK, fill, size=10 pt, leading=12 pt, indent, just=CENTER, vert=0.5, opacity, corner, border, border_color=BLACK, font="_sans", html, bold, italic, und, strike, wrap, hide_u, shrink=Y, shrink_min=6 pt, var metrics***)**

To draw a rectangle that can be stroked, filled with a color, pattern or gradient, with optional rounded corners:

draw_rect(*box, color, grad, patt, corner, corner_tl, corner_tr, corner_bl, corner_br, opacity, stroke=BLACK, width=0, pos=0***)**

To draw an oval (or circle if bounding box is square) that can be stroked and/or filled with a color, pattern or gradient:

draw_oval(*box, color, grad, patt, opacity, stroke=BLACK, width=0, pos=0***)**

To draw a circle given a center point and radius or diameter, that can be stroked and/or filled:

draw_circle(*center, x, y, radius, diam, color, grad, patt, opacity, stroke=BLACK, width=0, pos=0***)**

To draw a line. cap styles are **CAP_BUTT, CAP_ROUND, CAP_SQUARE***:*

draw_line(*p1, p2, x1, y1, x2, y2, dx, dy, opacity, color, width, rel, cap***)**

To draw a bitmap image. The image can be sized to fit the specified box:

draw_image(*image, box, indent, horz, vert, fit_horz, fit_vert, aspect, xy, x, y, pin, origin, originx, originy, border_color, border_width, angle, corner, opacity, blend***)**

To add a callback event into the Loom:

timer_start(*function, count=1, interval=U, rate=U, delay=0 sec, pre=U, group=U, time=U,...arguments for callback...***):num**

To count how many items are in an array:

count(*array***):num**

Character string syntax

To concatenate use ampersand (&):

```
""hello" & "goodbye"
```

Use braces to embed an expression:

```
"total count is {count}"
```

Use triple quotes to define multiple line strings:

```
'''Love all,
trust a few,
do wrong to none'''
```

Use escape characters to access special characters or block special interpretation:

```
'my dog\'s name'
```

Localization suffixes look like this:

```
"my name"~[123]
```

String escape sequences

{expr}	embedded expression
****	single backslash
\n	newline
\r	carriage return
\t	tab
\uAAAA	unicode character by hex code
\0	null character
\'	single quote (')
\"	double quote (")
_	space (explicit spaces in translations)
\~	non-breaking space
\-	em dash
\!	inverted exclamation (¡)
\?	inverted question mark (¿)
\<	left guillemet («)
\>	right guillemet (»)
\{	left brace (not embedded expr.)
\}	right brace (not embedded expr.)
\.	bullet (•)

String handling functions from the str library

subset(*str, from=1, to=U, len=U, rev=N***):str** *extract a substring from a string*

to_upper(*str***):str** *switch to all upper case letters*
to_lower(*str***):str** *switch to all lower case letters*

to_char(*num***):str** *convert unicode number to character*
from_char(*'c'***):num** *convert a the first character of a string to its unicode number*

pad_left(*str, len, pad=' '***):str** *pad a string on the left side to a specified length*
pad_right(*str, len, pad=' '***):str** *pad a string on the right side to a specified length*

to_str(*num, base=10, currency=N, currency_cc="$", decimal_cc=".", u_cc="?", thou=N, thou_cc=",", round_k=N, max=999, min=1, neg_paren=N, pos_plus=N, percent=N, zero_pad=N***) :str**

str_begins(*haystack, needle***):yesno** *test if a long string (haystack) begins with a small string (needle)*
str_del(*str, from=1, to=U, len=U, rev=N***):str** *remove a range of characters from a string*
str_ends(*haystack, needle***):yesno** *test if a long string (haystack) ends with a small string (needle)*
str_find(*str, pattern, startpos=0, ignorecase=N, rev=N, regexp=N, count=1***):a_find**
str_ins(*starting, insert, to=U, len=U, rev=N***):str** *insert a string into another, maybe deleting some characters*
str_len(*str***):num** *get string length*
str_localize(*str, lang=U, index=U, qty=U, fallback=N***):str**
str_repeat(*str, n***):str** *repeat a string n times*
str_replace (*haystack, needle, replacement, start=***0):str**
str_replace_multiple (*haystack, changes, trace=N***):str**
str_reverse(*str***):str** *reverse the characters in a string*
str_split_lines(*str,result,delimiter=***TAB)** *split a string into lines using delimiters*
str_split_lines_words(*str,result,delimiter=***TAB)** *split a string into lines and words*
str_split_words(*str,result, delimiter=***'')** *split a string into words*
str_strip_quotes(*str***):str** *strip single or double quotes from a string*
str_to_enum(*str***):num** *convert a string back to the enum, U if not found*
str_to_num(*str***):num** *convert a string into a number,* **ERR** *if incorrectly formatted*
str_to_tree(*json, tree***)** *convert a string into a tree*
str_trim*(str)* **:str** *trim a string on both left and right sides*

enum_to_str(*num***):str** *convert an enum into string form*
json_to_tree(*json, tree***)** *convert a JSON string into a tree*
meas_to_str(*meas, styles...***):str** *convert a units of measurement into string form*
tree_to_json(*tree, limit=INFINITY***):str** *convert a tree into a JSON-compatible string, with an optional term limit*
tree_to_str(*tree, limit=INFINITY***):str** *convert a tree into a string*

Constants

Y	*yes, true, on*
N	*no, false, off*
U	*undefined*
ERR	*error*
INFINITY	*positive infinity*
-INFINITY	*negative infinity*
PI	*π*
TAU	*2 π*
E	*euler's constant 2.718...*
GOLDEN_RATIO	*golden ratio 1.618...*
BEEP : sound	*system beep*
META : tree	*for introspection*

System Variables

runtime : a_runtime

System Records

meas	*mag, unit*
a_date	*date_year, date_month, date_day, date_hour, date_minute, date_second*
a_event	*evkind, when, x, y, z, global_x, global_y, keycode, unicode, is_shift, is_ctrl, is_alt, is_cmd*
a_xy	*x, y*
a_xyz	*x, y, z*
a_rect	*left, top, width, height*
a_runtime	*args, app_version, air_version, os_version, os_language, os_kind, cpu_kind, full_screen, window_horz, window_vert, screen_horz, screen_vert, screen_dpi, touch_kind, hardware_id, notch_height, notch_width, major_stepx, major_steps, micro_steps*

Math functions

abs(*n*):num
distance(*a_xy, a_xy*):num
distance_xyz(*a_xyz, a_xyz*):num
exp(*n*):num
exp_minus_1(*n*):num
epsilon(*n*=1.0):num
fract(*n*):num
idiv(*input, divisor,* one=N):num
lg(*n*, base=E):num
min(*a, b, ...*):num
max(*a, b, ...*):num
mod(*input, divisor,* one=N, neg=N):num
next_float(*n*):num
power(*base, exponent*):num
prev_float(*n*):num
sign(*n*):num
sqrt(*n*):num
uzero(*n*):num

Trigonometry functions

arc_cos(*n*):Angle
arc_sin(*n*):Angle
arc_tan(*n*):Angle
arc_tan2(*y, x*):Angle
cos(*angle*):num
hypot(*a, b*):num
sin(*meas*):num
tan(*meas*):num

Machine functions

machine_spawn(*name*)
machine_pause(*name*)
machine_resume(*name*)

Misc. functions

halt(*errcode*=0)
type_of_val(*val*):num
type_of_ptr(*path*):num

Tree functions

tree_count(*tree*):num
tree_hi(*tree*):num
tree_lo(*tree*):num
tree_next_hi(*tree*):num
tree_next_lo(*tree*):num
tree_sibling_hi(*ptr*):ptr
tree_sibling_lo(*ptr*):ptr
tree_index(*ptr, keys...*):ptr
tree_deep_index(*ptr, keys*):tree

Coordinate functions

local_to_global(*x,y*):a_xy
global_to_local(*x,y*):a_xy
measure_table_column(*kind,col*):num

Rounding functions

round(*n*, multiple=1):num
round_up(*n*, multiple=1):num
round_down(*n*, multiple=1):num
round_zero(*n*, multiple=1):num

Yesno functions

is_enum(*n*):yesno
is_err_u(*n*):yesno
is_err_enum(*n*):yesno
is_even(*n*):yesno
is_field_in_record(*field, rec*):yesno
is_finite(*n*):yesno
is_infinite(*n*):yesno
is_landscape(*a_rect*):yesno
is_numeric(*n*):yesno
is_odd(*n*):yesno
is_portrait(*a_rect*):yesno

Sound functions

sound_pause()
sound_play(*sound, loop, notify...*)
sound_play_file(*path*)
sound_resume()

File operations

file_exists(*path*):yesno
file_read_bytes(*path*):bytes
file_read_str(*path*):str
file_read_tree(*path*):tree
file_write_tree(*path,tree*):yesno
launch_file(*file*)
launch_url(*url*)
path_extract_folder(*pathstr*):str
path_extract_file(*pathstr*):str
pick_dir(*title, callback*)
pick_file_open(*title, callback, ...*)
pick_file_save(*title, callback*)

Time functions

elapsed():num
now():num
seconds_to_date(*sec, city*=U)
date_to_seconds(*a_date*):num
days_in_month(*yy,mm*):num
day_of_week(*date, ...*):num
day_of_year(*yy,mm,dd*):num

Random functions

random():num
random_color():color
random_range(*start, stop*):num
random_range_int(*start, stop*):num
random_parm(...items):any
random_word4():str

Color functions

rgb(*r,g,b,a*):color
rgb1(*r,g,b,a*):color
hsv(*h,s,v*):color
hsv_to_color(*hsv*):color
color_to_hsv(*color*):hsv
color_r(*color*):num ..g ..b

Unit families and their units

Angle — *angle*

- deg
- gradian
- radian
- rev

Area — *len^2*

- acre
- hectare
- sq_cm
- sq_ft
- sq_in
- sq_mi
- sq_yd

Energy — *$len^2 \cdot mass / time^2$*

- BTU
- calorie
- ev
- gigajoule
- hp_hour
- joule
- kw_hr
- therm

Force — *$len \cdot mass / time^2$*

- lbf
- newton

Frequency — *1/time*

- hz
- rpm

Length — *len*

- angstrom
- cm
- dm
- foot
- inch
- km
- m
- mile
- mm
- nautmile
- nm
- um
- yard

Mass — *mass*

- grain
- gram
- kg
- ounce
- pound
- slug
- ton
- tonne
- troy_ounce
- troy_pound

Power — *$len^2 \cdot mass / time^3$*

- gigawatt
- hp
- kilowatt
- megawatt
- milliwatt
- watt

Pressure — *$mass / len \cdot time^2$*

- bar
- pascal
- psi
- torr

Scalar

- dozen
- each
- gross

Speed — *len / time*

- ft_per_min
- ft_per_sec
- km_per_hr
- m_per_min
- m_per_sec
- mi_per_hr

Temperature — *temp*

- deg_c
- deg_f
- deg_k

Time — *time*

- day
- hour
- microsec
- millisec
- minute
- month
- nanosec
- picosec
- sec
- week
- year

Volume — *len^3*

- cu_ft
- cu_yd
- cup
- gal
- l
- ml
- oz
- pint
- quart
- tbsp
- tsp

Literals

Use single braces to create a tree literal:

```
var t : tree = { name:"fred", age:12 }
```

Use single brackets to create an array literal, semicolons indicate go to next level:

```
var two_by_two = [ 2, 3; 4 6]
```

Use the [<...>] notation to create a literal that is an array of records; the first row is the names of the fields:

```
var people : array of people = [<
    name, age
    "fred", 22
    "sarah", 44 >]
```

Use # to prefix a color literal:

`#aabbcca` *-- a hex color constant*

Implied variables

In draw blocks the implied variable `b:a_block` *is defined:*

```
record a_block
    box      : a_rect  // local bounds of the drawing area
    matrix   : tree    // current transform matrix in effect
    blabel   : str     // block label if any
    dest     : num     // FOR_PRINT, FOR_SCREEN, etc.

    // fields that are set inside a table
    row_kind : num  // TRACK, the kind of row we tapped on
    row_box  : a_rect // box for the whole row
    row_id   : num  // the unique id of the row
    col_id   : num  // the unique id of the column

    // fields that are set inside a grid and table
    cell     : a_point  // cell col and row (as .x, .y)
    cell_id  : a_point  // id of the cell
    ncols    : num  // number of columns in the grid
    nrows    : num  // number of rows in the grid or table
```

Regular expression definitions

regexp *name (ignore_case | global | multiline | starts | ends) (parm1:str parm2:str ...)*
...regular terms...