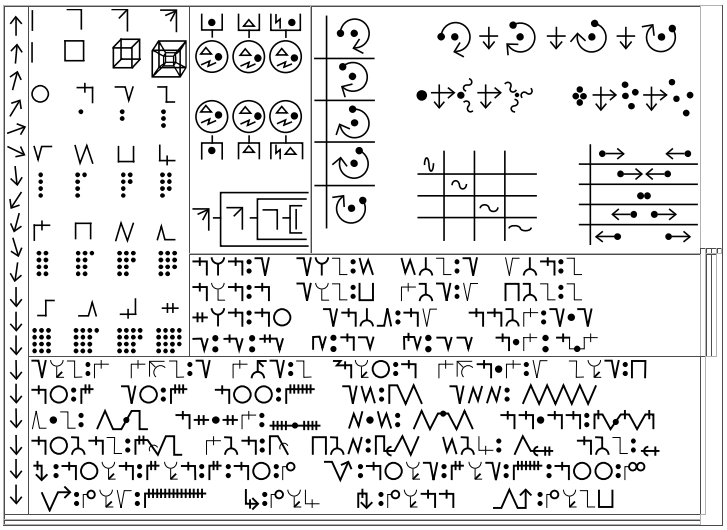
Yes.. there are **still** , even after so so so many passes, additions, edits, and revision happening as this is made.

The flow of definitions shows it, there are still, even now, definitions that come after 'their group' and other definitions that would benefit from being shifted backwards so they can use other definitions that exist later.

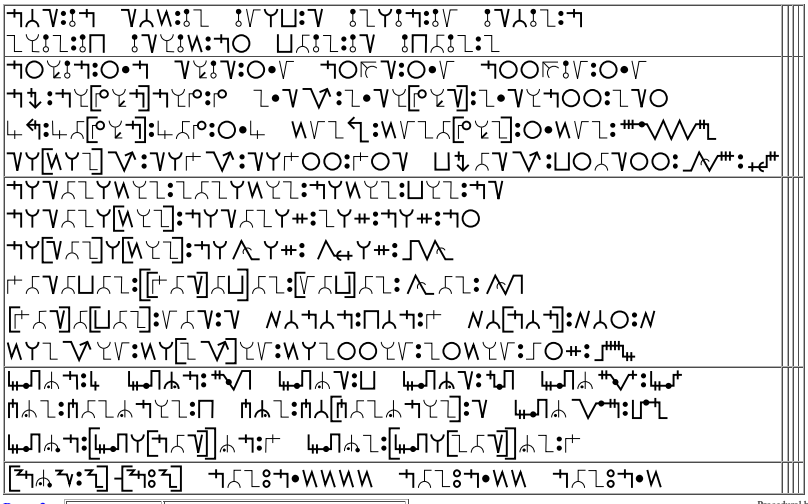
But I am finally satisfied and confident that this version and these tools can come together into a solid enough framework that I can move on to the 'expansion plug-in' I have been holding onto. I have been getting impatient, I don't want to start adding plugins to a framework that hasn't even been well defined itself. But I think this version will be able to flesh out the math, procedural, and physics base solidly... plus move on to more communicative and abstracted layers and define them all with a strong enough base to justify adding 'plug-ins'. Now I just need to keep picking at it until it gets there.... big project, must not rush or get impatient hehe.

General flow is

1. **Math**
   Obvious first step. numbers and basic arithmetic are super easy to define visually from scratch
2. **Procedural**
   With math it is then easy to define sequence of operations, loops, brackets, arrays, etc..
3. **Geometry**
   Geometry is easily defined using some constants like pi, equations like $a^2+b^2=c^2$, functions like sin(), cos(), etc..
4. **Calculus**
   Using functions and arrays we can use geometric concepts to define slopes, areas under curves, etc..
5. **Physics**
   Starting with fundamental particles, a table of mass values is universal (not in unit values, units are arbitrary... but ratios between the particles is universal). With particles we can then define units like distance, time, forces etc.. as well as molecules, interactions, etc..
6. **Astronomy**
   Using physics we can define the other side of the universal spectrum. The very small is universal (particles, forces, etc...) and the very large is also universal (Astronomy). So we can define stars, planets, blackholes, galaxies, etc... (funnily, what is not universal is all the things that seem most significant in our lives, the human scale is not universal. I suppose we find ourselves in this range precisely because this is where permutations and possibilities are so vast.)
7. **Communication and Abstraction**
   At this point move on to building communication tools for exchanging ideas, asking questions, etc... The building blocks established in previous layers can be used directly, or as abstractions. Establish ways to abstract with metaphor, analogy, etc... We can create things akin to 'x is like y', 'x is y-ish' etc...
8. **Plug-ins**
   Plug in other universal definition from visual example. Add other methods fo defining from scratch.

Page	defs	desc	todo
Page 1	dims … numbers	some intro vague conceptuals like arrow and subset, time/entropy, dimensions There are mirror versions of non-commutative operations the reverse the operation direction Operation direction is arbitrary anyways. For example in Chinese the normal way of expressing fractions is inverted. 1/3 on third is 三分之一 (3 split of-that 1), 三分之二 is 2/3, etc.. This will also help with design expressions and 'hiding brackets' later.	Add examples of inverted non-commutative operations add example of scinote left shift (only added right shifting so far) *string bin digits is all high or all low should include mark to show high/low if not clear *maybe fill in arrow triangle to add-mass to tip *sub example maybe clarify that all can be sub or any permutation *numbers(hex, bin-string,tens,fractions),+-/*,exp,root,log *maybe rotate number dots so they follow reading dir l-r t-b *maybe modify string numbers so that the digits around the radix point are inferred form position 1_.._1 = 11.11 0-.-0 = 00.00 0-_.1= 01.11 0-.-_ = 00.01 0-\(dot in line)_1 = 00.11 etc... !! not satisfied with ten and hun symbols.. maybe change them or delete them
	fractions sci-note		
	ten hun thou :=		
	+ - * / exp root log alt exp		
	sub()		
	subtract (3 subtract 1 = 2) subtracts (1 subtracts 3 = 2)		
	the dot in the subtract symbol represents the side kept / measured		
	x / (divide-by) y x \ (divides) y dim corner points to the 'segment length to divide into'		
	x to-the-power-of y x powers-of y dimension corner points to power		
	x rooted-by y x roots y dims/split points at number of dimensions to fracture into		
	x logged-by y x logs y dims corner points at dimensional length to fracture into		
			
Page 2	negative numbers	negative numbers, scinote usage, op sequence, int/frac/var symbols	add more approx-equals exmaple I think the way the number is thirteen should imply tolerance and certainty. 1.23 means there is tolerance on the 0.3 but not on the 0.2.. otherwise use the range/tolerance symbol. As for large numbers like 1000 with a tolerance of hundreds, then write the number using sci-note so you dont actually draw the hundreds digit could add 2d drawing style like upper exp, division stacks, root sign, etc.. not getting into that stuff now.. strictly linear bracketed for now will establish ints, rationals, and later irrationals.. probably stop there for 1d numbers to start, can use clauses to establish subsets of those when needed, or establish a symbol later on if actually necessary or needed for convenience
	floor mod round	floor, mod, round, aprox equal(a rounded number)	
	braks and op order (left to right. no priority based on operations type)	no bedmas, just left to right, except for scinote I admit bedmas helps hide brackets.. just like root and fraction symbols help hide brackets, or how the expression in a power is calculated before the power to hide brackets. Uscript doesn't use bedmas as a 'braket-hiding-tool', because that is a very human, historical, and culture specific system. admittedly the hidden brackets in roots, powers, and division are visually elegant, but that's not true for bedmas in general. Uscript sticks with sequential operations and brackets... we will find other ways to "hide brackets" (more visual and intuitive ones, akin to division, powers and roots in our current system), This will also provoke more divergence from our current systems, representation, and thinking.	

≈ approximately equal to.



Page 3

true false | bool ops (and/or/xor/etc..)

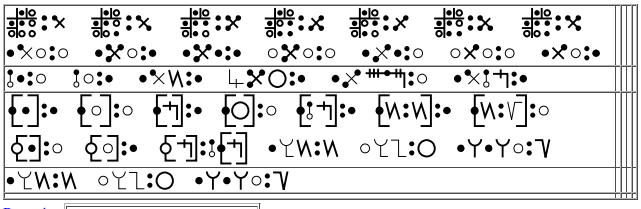
eval | evalneg | invert bool

in math like +/-/*/etc.. true=1 false=0

Procedural basics

booleans and gates

*maybe clarify that eval mode is supplemented by gt / lt cooperators which can perform actual comparisons like > < == etc.. if they return 0 then things are equal



Page 4

int frac and var types

wave/undefined/uncertain how vars work

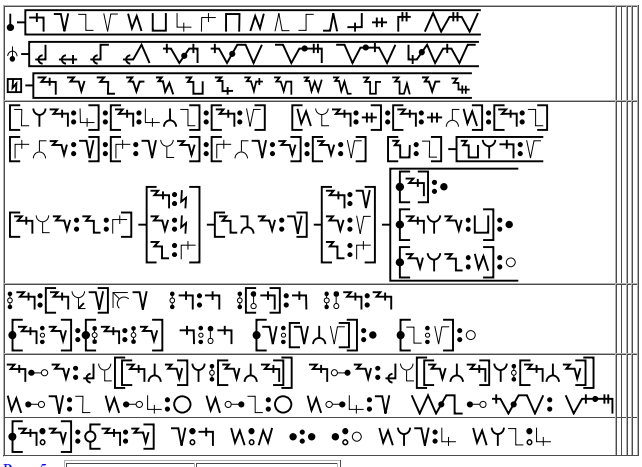
absolute | absolute equal | !=

> < gt/lt.by how much. not just bool

next do approx equal to!

introducing real numbers, fraction and variables as categories of "numbers"

variables, or "wave-numbers" are like using a,b,c,...x,y,z as variables, except that that are wave-1, wave-2, wave-3, etc..



Page 5

varset :> <: | swap var values

add right to left | add left to right

subtract right from left | subtract left from right

set left to right value. then zero the right value

set right to left value. then zero the left value

Move-left. add right value into left var. then zero right var

Move-right. subtract left value from right var. then zero left var

Varset defined
we only define the case of a single varset per expression, and it causes both sides to be fully evaluated before the varset

weird cases are undefined for now. like multiple varsets per expression, or varsets inside of brackets.

!!!todo
more examples of variable setters
add a version of varset where there is a center braketed or bin sring. that means the the center value is applied to both sides (eg "v1 < sub : 3 : add > v2" = "sub 3 from v1 and add 3 to v2")
add expression vars

procedural intro
variable setting, if conditionals, sequential operations

We now have varset so we can distinguish between statements of x is equal to y and instructions of set x to be y. so we don't need to rely on bracketing and subsets to show the results of variable actions.

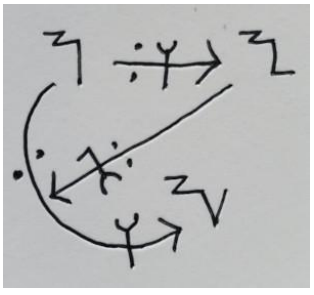
IF evaluates the raw result of conditional

IFF (if false) inverts the conditional evaluation
yes, there is redundancy
it is a byproduct lower level design

it allows us to connect multiple outputs to a single conditional (can connect both result cases , it allows adding an else)

you can also consider it as optional ways to offer different contextual emphasis
if and iffn both do the same

add examples where more than 2 vars are involved, a center expression that can be applied in different ways to many variable (eg value 3 can applied in different ways to many different variables with many varset arrows all originating form the same central bracket each with different operators.) *not sure if i will include that in this version



Remove-left. subtract right value from left var. then zero right var

Remove-right. move left value into right var. then zero left var

sub right from left and add left to right. Simultaneous actions.

sub left from right and add right to left. Simultaneous actions.

add R to L. subtract L from R twice. simultaneous

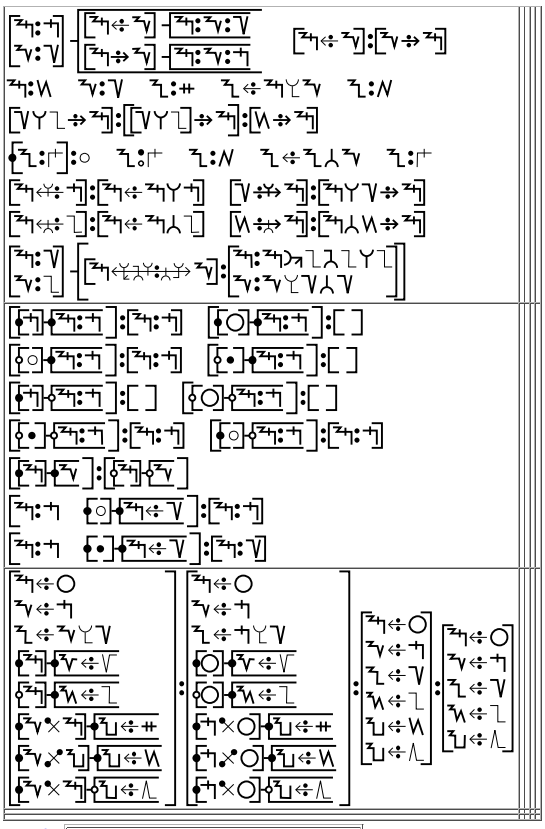
L = L + R. R = R * L - L. All ops use the original var values

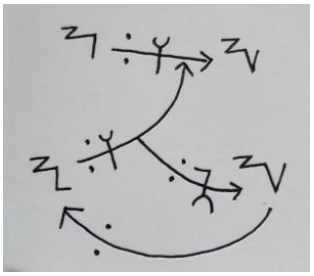
L = L exp R divides R + R aka.L=(R/(L^R))+R.... R = R * L - L

ops are 'loaded with values then pushed into the vars' so they go from 'outside inwards'

if( ){ } ifn( ){ }

iff( ){ } iffn( ){ }





but this is definitely something to flesh out later. time time time.. so much to do, so little time.

add expression vars

Page 6

array def ([][][]...)

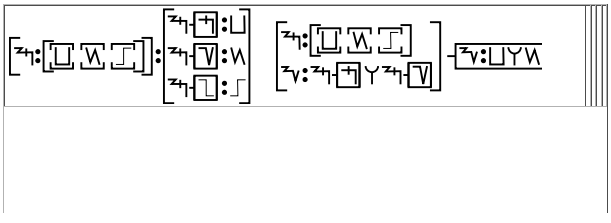
array ref v1 s[1]

ops across all elements of array

sum() count() avg()

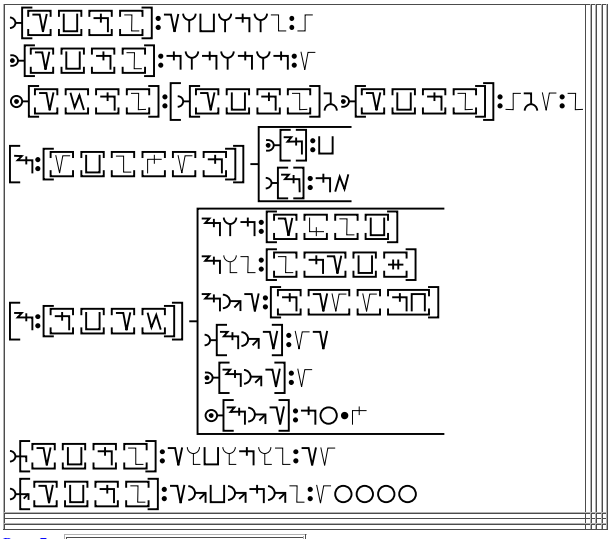
prod([1][2][3])

expo([1][2][3])



arrays

Some of these definitions would be easier if I add array pop here first, instead of later



[Page 7]

nlcon connect to preceding line

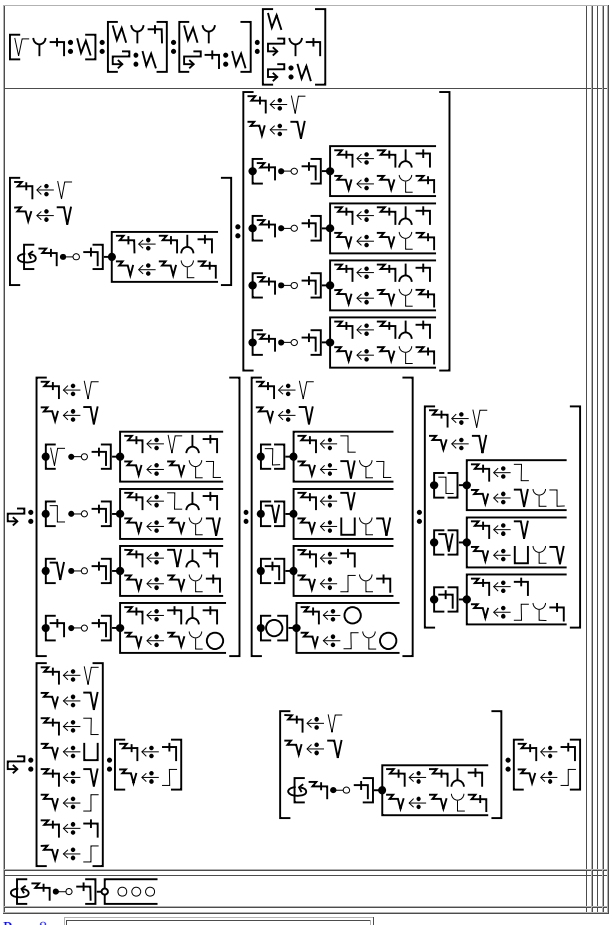
define line connector. It allows be linked to be an extension of the previous, essentially saying ignore previous carriage return.

while (...) { ... }

while loop definition

while-false loop end op

while-false is not fully defined yet
it will mean "execute once at the end when the loop breaks. This allows both the loop and an exit condition to be draw as a single unit
it also creates a single symbol for "wait until a condition, then do this"
yes you can just add code underneath a while loop. Uscript will use these in more abstract linguistic environments, so codifying these things connected units is useful



[Page 8]

foreach(...) { ... }

foreach and execution brak

exe(...)

return

define that when inside a loop, or anything, varset 'into-nothing' has the function of setting a return value for the loop, or if, etc..

Seeing as I added reverse pop all loops, should probably add reverse foreach loop

a symbol to stand in for variable of the current iteration

loopvar symbols dont address nested loops, maybe give it a way to reference levels
I think just allow the loopvar to be treated as an array, but make a special call were you reference an array element using sci-note
eg loopvar[scinote+1] = up one nested loop level

next loopval

prev loopval

need to add example of all the extra loopvars (next,prev,original,etc..)

loopval 2 loops ahead. 3 loops ahead. 4. 5. etc..

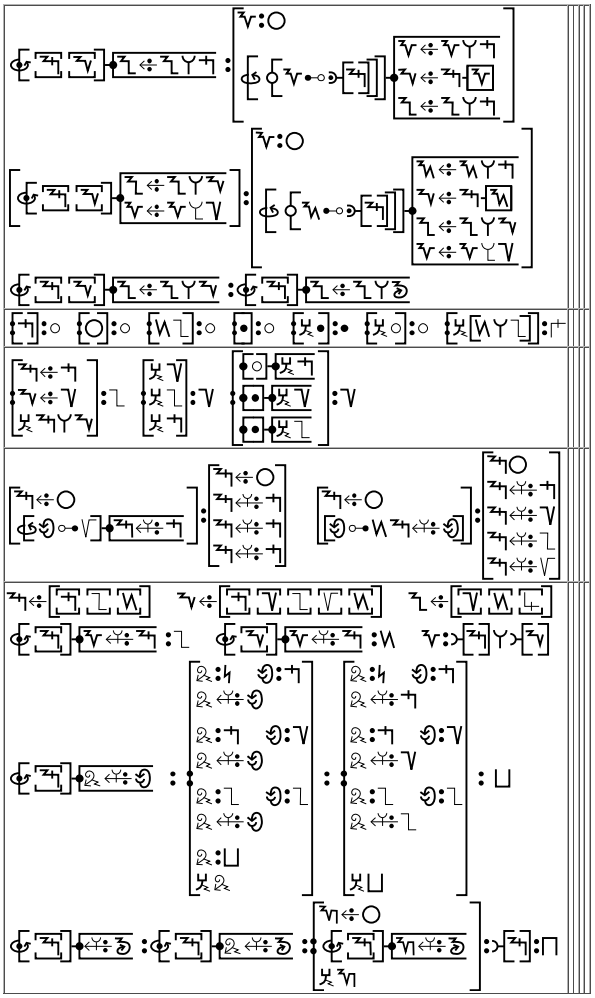
loopval 2 loops back. 3 loops back. 4. 5. etc.

original loop vals before the loop executed

original loop vals are for reference only. constant values during loop

loop-counter starts at one increments after each cycle

the return value which can be modified when inside the loop.
setting this does not break the loop.
the return is the final value after the loop finishes.



Page 9

array append

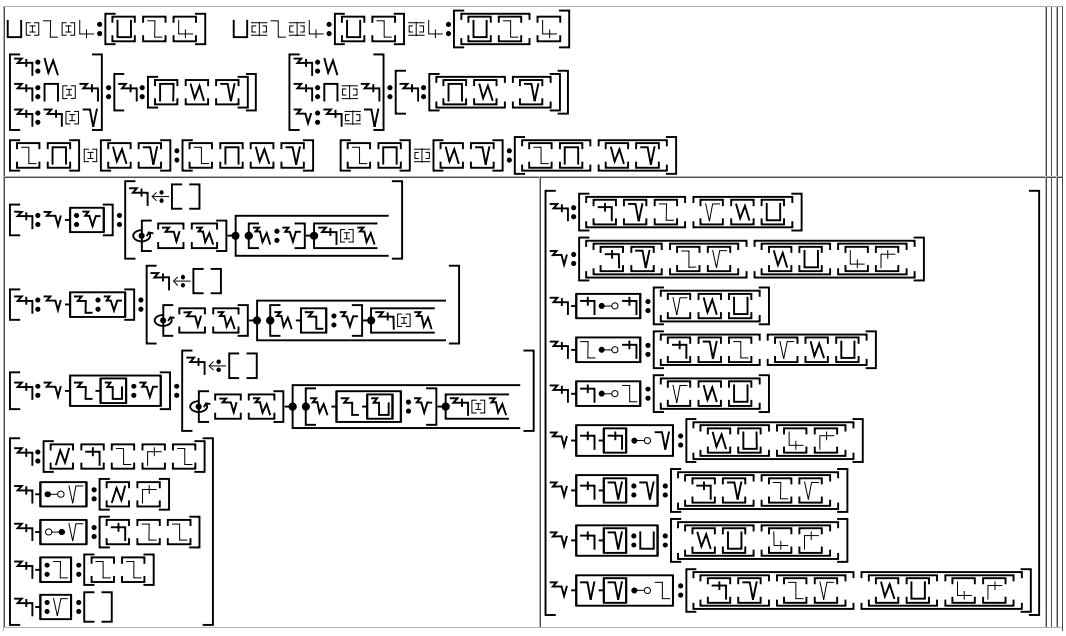
array combine
array append
array subset

**could do so many more array ops, but drawing the line for now

**add count/sum/avg onto array extract for quick ops(just add small icon prefix to the box protrusion). that way you can ask "how many meet this conditon" etc..

array combine (array of arrays)

... = Condition to check elements.
Qualifying elements are included in output array



Page 10

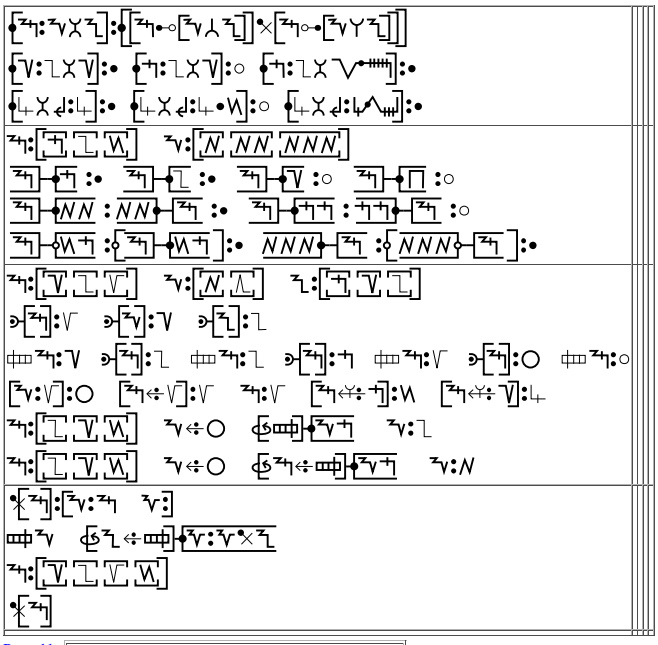
tolerance

... has o | o is in ...

... doesn't have o | o is not in ...

pop first array element | pop last array element

and left. L = L and R | or left . L = L or R



tolerance
array pop
bool varsets

tolerance here means "deviation must be less than" not "deviation acceptable up to".

pops will serve well as the idea of "taking from" later

establish that a unilateral varset operation can be evaluated and check. its value is equal to resulting new value of the variable.

bool ops can be used with varset the op using the establish patterns might need more examples, but the patter is clear i think

5 tolerance 2 include 3.0000001 but does not include 3 or 7

all/non/etc... these benefit more form expression variables which come later

**should probably add invert-array symbol(rotate-array)

**add other form of tolerance deviation up to and including)

**add examples of tolerance where numbers are drawn above or below to manipulate the upper and lower bounds independently

Add defintion that declare "a larger number contains a smaller number" using has. this is the behavior if the checked input is not an array. this way it can be abstracted better into linguistic usage.

Page 11

expression vars hold expressions

expression holding braket

allpop last to first. A foreach that empties as it goes

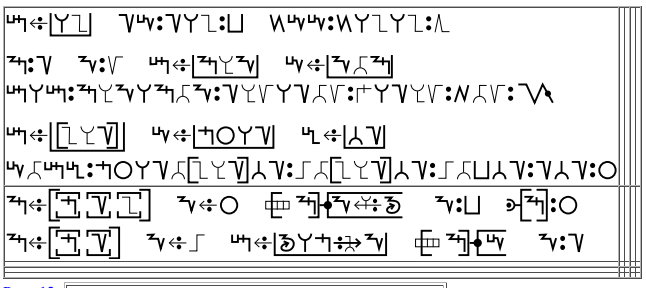
popall first to last. A foreach that empties as it goes

apair. array pairs. first to last.

paira. array paris last to first

arraypair runs for 'gap between elements'
there's no current loopvar just prev and next

array sort. small -> large | reverse sort. large -> small



expression vars

expression vars hold expressions,

we ONLY define simple usage here

we do NOT address expressions containing expressions vars, or any other complex applications

Yes! introducing expression vars opens the door to a lot of questions... "how do I put the value from a variable into an expression as a fixed value instead of variable reference?" how do I build a dynamic expression then lock it down so I can now change the vars used to create it? etc..

the answer is "Those are questions for another day.. those behaviors are undefined for today"

pop loops - they popall elements and handle each. There is still a 'if true dot' on the sub brak becaus ethsi way we can add a false one to run at the end.

Pop loops are different from foreach loops, pop loops empty the original array, this means they can better represent situation when things are handle or processed. Many physical process of "take all of X and process them" convert or destroy the original source material.So this is actually a useful distinction for abstracting into language.

array sorts simple value sorts on simple 1d arrays for now
later can add bracketing and nuance to sorting

array pair loop examples

array sort exmaples

The varset arrow can connect to the bracket, can add that later

then some cool 2D webs of expression building can be constructed. even skipping the expression variables ad just pointing to where the expressions should be inserted.

add inner-array-loop ... runs between all all the elements instead of on each element. loopvar will be empty but loopnext and loop prev will have values.

eg
foreach ((1,2,3) as x) { = x } == 3
foreach ((1,2,3) as x) { += x } == 6

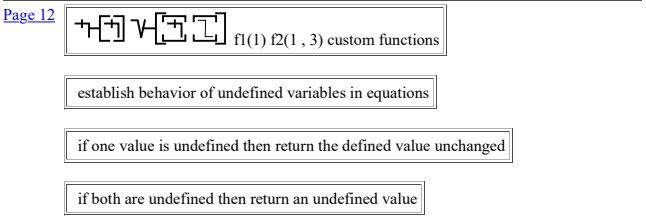
Page 12

f1(1) f2(1 , 3) custom functions

establish behavior of undefined variables in equations

if one value is undefined then return the defined value unchanged

if both are undefined then return an undefined value



all/any/none/any-dont
those which
min/max

Custom function
Define loop return values and how to manipulate them
default return is number of loops executed. but setting it at any point overrides that behavior

establish that undefined variables have no effect in equations. an undefined variable acts as 0 in additions and subtraction. as 1 in * / ^, in 'x divides y' an undefined y acts as x^2 etc..
1.if one of the values is defined then you get the defined value as a result
2.with 2 undefiuned values the result is an undefined value.

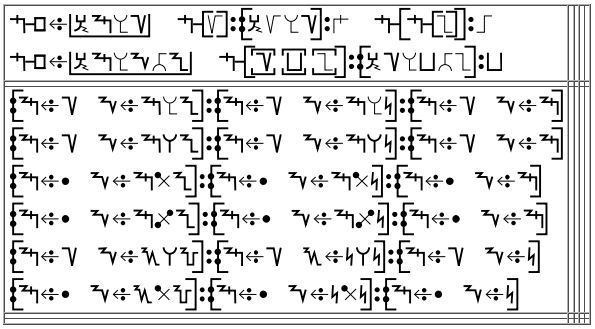
For custom functions you can use any symbol as the brac symbol. but I have only added custom alignment for number 1 to 6

I simple number might be best,.

of course custom symbols can be designed and drawn. but for general purpose function definition and usage, I think just labeling them the same everything is given an ID is the simplest solution for now

should add more loop return behavior examples

there now alot of what is needed for calculus
we can even draw the procedural logic quite elegantly
function, array pair loops can draw trapezoidal areas under iterated functions
...debating the order and sequence of how to add calculus and geometry before moving on physics and astronmy



Page 13

all - the and bool array brak
Bool array braks use boolean op between all array elements
all(array(a , b , c)) = a && b && c

none - uses falsefalse between all array elements
falseflase returns true is both are false

any - uses the OR between all array elements

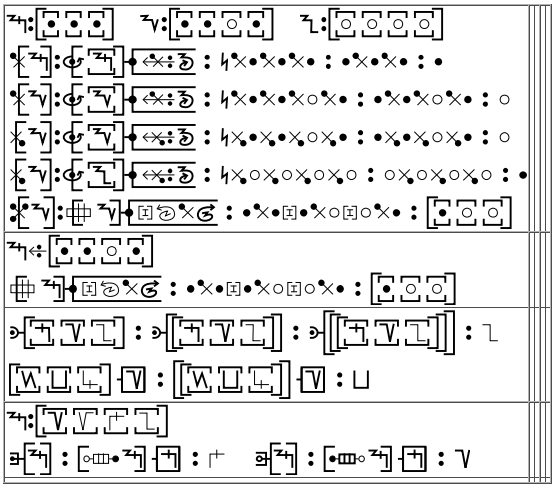
min - returns the smallest value in the array

max - returns the largest value in the array

all/any/none/any-dont
min/max

establish that reg brackets don't add depth to arrays. they are only used to wrap them for assignments. double or triple wrapping them does do anything different than single wrapping.

add bit wise bool ops
I have not added them yet because they don't have immediate use, but considering procedural is such a fundamental layer, and numbers are bit constructed, it feels obvious that bit-wise operators should be included in Uscript.



Page 14

pi | radius | diameter | circumference

geometry

a simple thing like pi, defined by the value, using basic math expressions we can build up so much.

a quick define that strings of solid symbols (like bin string numbers, variable, fractions) are multiplication by default.