Vlang

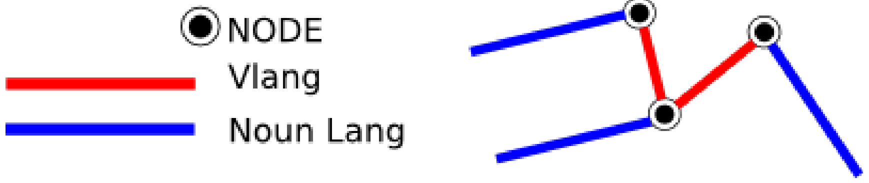
Verb Lang - 2D Verb Based Language/Writing System

Vlang attempts to utilize Rstruct in the most basic possible way. The goal is to achieve 2D language not by re-inventing language as a whole, but instead by "devolving" it back to basics.

Language, especially written language, is complex, but when it comes right down to it Vlang only has 2 types of words, Verbs and Nouns, objects and actions. Properties can been seen as "objects being something", therefore an action between objects X is Y.

Vlang assumes the role of "Verb" and imports and existing language for "Noun".

The imported language is written on nodes, while Vlang connects the nodes.



Vlang is always used between nodes, and the NounLang only connects to nodes on one side.

- Any language or writing system may be imported
- Nouns can be phrases arbitrarily long
- A NounLang verb word used alone as a node refers to the verb as if it were noun

There are many ways to use "empty connections". Connection or nodes without any Vlang/Noun-Lang could be used to describe hierarchy, grouping, sequence, etc.. for now we will leave that open.

The goal of Vlang is to explore what Vlang can do with **Rstruct**(see Page 6). Like an adjustable framework built out of an adjustable framework.

For the rest of this document we will ignore all methods of description that "bypass" Vlang, otherwise we will probably just end up with a mountain of "shortcuts" that end up conflicting with and restricting Vlang before it gets a chance to develop.

So now that we have our goal, how do we structure Vlang?
Well we could just import an existing language dictionary. We could increase the alphabet of **Qscript**(see Page 9) up to 52, assign each anti-letter to mean the same as its letter, assume anti-words are read backwards, and just use English cursive with a mirror plane...

But the function of Vlang is so different and such a huge alphabet would be sooo messy.

Vlang actually requires, for the first time in all my projects, a "true conlanging task".

I am not a conlanger, more of a conscripter, but I'll give it a go, bear with me, its just a test anyways ;)

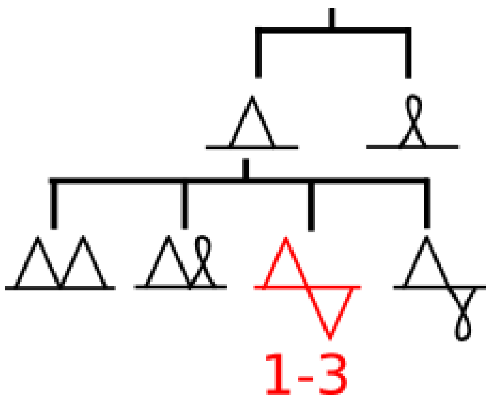
Vlang 0.1 begins by leveraging the small alphabet which creates a permutation hierarchy that starts with 2 roots (4 values... 2 values and 2 anti-values) and branches into 4 each level down. This lends itself well to a hierarchal approach the language itself, that is also easier so I will start there.

The Structure of permutations is very simple

We will assume all words start with 1 or 2, and all anti-words start with 3 or 4.

2 roots, each break into 4, each of which break into 4, etc..

There are also Mirror Words. (In red in graphic on right)
Mirror Words are special because they mirror into themselves when read upside down. (actually 180 rotate not real mirrors)



So 1-3 becomes 1-3 when rotated 180. That means

- Any verb assigned to 1-3 must not have a "direction" or be "unilateral". A verb like "eat" would be X eats Y, it has a direction between X and Y, reversing X and Y changes the meaning, but something like "is connected to" goes in both directions and works for 1-3
- Each mirror word has an anti-word that is also a mirror word.

Mirror words provide space for 2 words, but these 2 words loose a dimension of description.

Anti-words have many possible uses, for now it will be the simplest application.

Imagine you always rotate the paper so that you are reading left to right
X → Vlang → Y
All words are designed to operate between 2 objects.

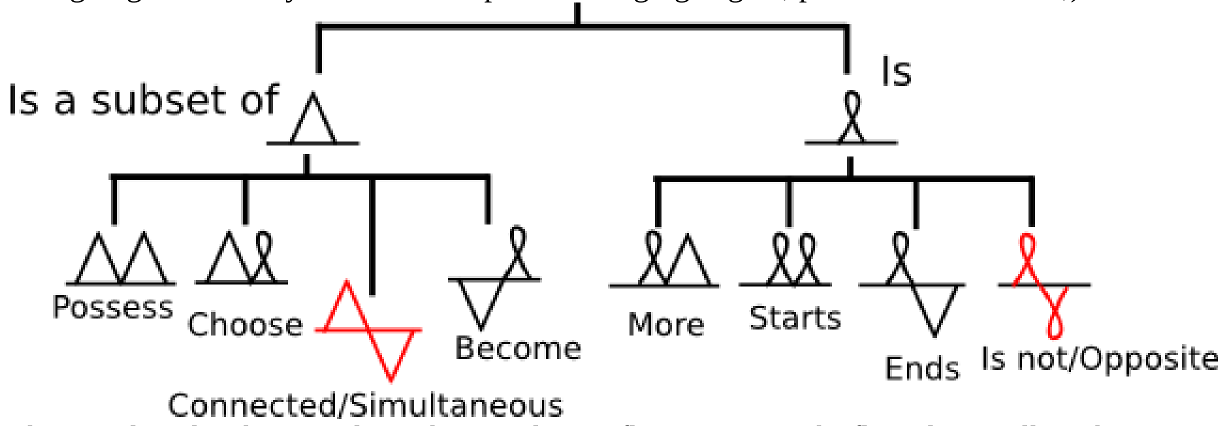
An anti-word simply reverses the X and Y.
That way when you read "backwards" the original meaning is entirely preserved.

eg.	Forwards	Frank → Vlang(eats) → Fruit
	Backwards	Fruit ->Vlang(is eaten by) → Frank

Here we create the Vlang word "eats" which automatically creates the anti-word "is eaten by".

Anti-words are not antonyms, they just have "reverse grammar". So the anti-word for "love" is not "hate" but instead "be loved by".

Now I'm going to make my humble attempt at conlanging. Again, please bare with me ;)



*When words end with an anti-letter then word or it's flip-mirror must be flipped vertically eg become/end.

IS A BUBSET OF – X is a subset of Y

Posses – X Possesses Y

Choose – X Chooses Y

Mirror a)X and Y are **connected**

b)X and Y are **simultaneous**

Become – X Becomes Y

IS – X is Y

More – X is more than Y

Starts – X starts Y

Ends – X ends Y

Mirror a)X **is not** Y

b)X and Y are **opposites**

Each of the words can further break up.

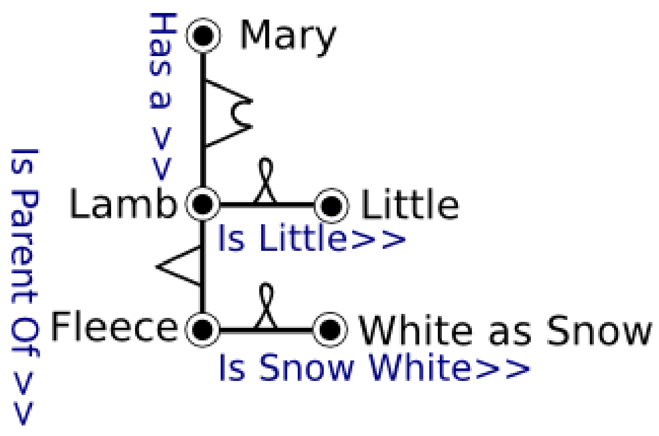
We will avoid 3 letter long words because they are a pain to divide in half. There are plenty of ways of using them, but we will just sweep them aside for now and go straight to 4 letter words.

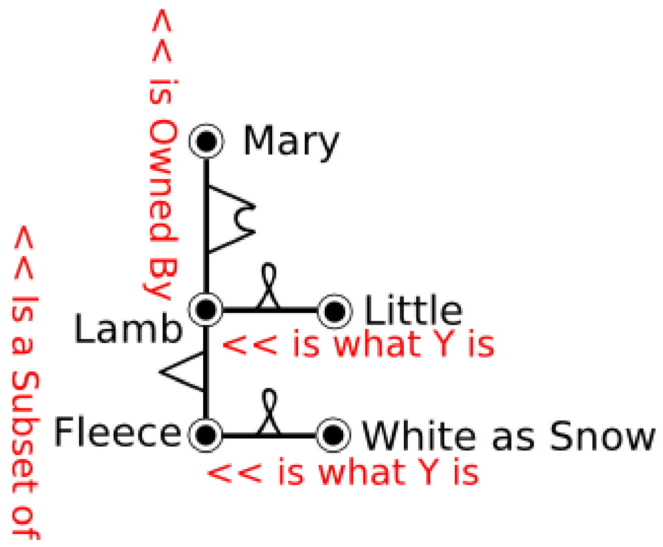
Each 2nd level root(2 letters) will have 16 4th level sub-words. The dictionary is still very small, and grows slowly because many words are imported by the Noun-Lang.

The examples are in standard English text. I personally enjoy using Chinese characters and English written in Dscript for the Noun-Lang, but simplicity first.

For those interested, OUWI (http://www.ouwi.org/) is a2D writing system and language that also embraces importing vocab. The language structure is much more unique in OUWI whereas here we try to retain more similarities to standard alphabetical language.

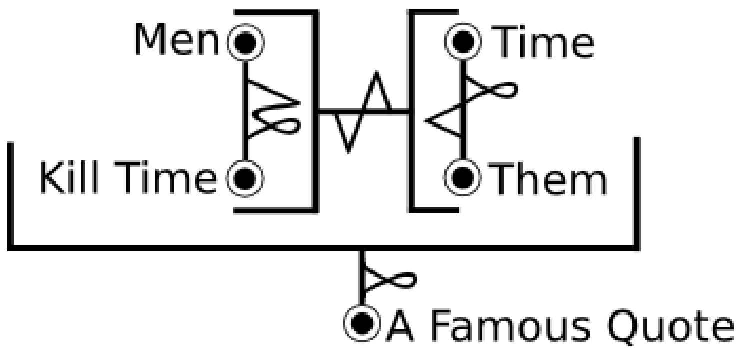
Example 1





Mary (has a) Lamb (which is) Little,
The Fleece (subset of of the) Lamb (is) White as snow
…..or reading backwards using the anti-words …...
White as snow (is the state of the) Fleece (which is a subset of) Lamb
Little (is the state of) the Lamb (which is owned by) Mary

Next we have “brackets”, these allow us to refer to entire spacial regions of the writing space as a noun.



Here is a basic version of the quote “men choose to kill time while time quietly kills them”

For simplicity in this example we will skip the word quietly for now, and reduce “kill time” into a noun.

Also, here the entire phrase is in brackets which describe it as being a famous quote.

```
|        Men Vlang(choose) Kill Time--------|                                  |
|                                           }---------Simultaneous             |
|        Time Vlang(Ends) Them--------------|                                  |
--------------------------------------------------------------------------------
                                        |
                                    Vlang(is)
                                  A famous quote
```

Now let's look at “Vlang Phrases” or “compound verbs”.
Normally Verbs affect 2 noun nodes. eg. nounX verb nounY

In Vlang multiple words are nested as opposed to sequential. We do not want the verbs to act individually on the same nouns (that can be done with multiple Vlang connections per verb, or perhaps invent rules to allow sequential phrases for this purpose, for now lets just assume it is handled with multiple Vlang connections)

A string of words in Vlang looks like this

nounX - Vlang_verb_1_part_a - Vlang_verb_2 - Vlang_verb_1_part_b - nounY

So we want one verb to target another verb as opposed to targeting the nouns.
Here I choose to have the center(last) verb be dominant. The center verb is the main action/relationship between nouns.

I choose the center because the center verb will never be split in halves, it will always be “whole”, the center value is also the only place that odd number length words (1,3,5,7,etc..) can exists according to the rules already laid out.

This choice is rather arbitrary, there is equally good reason to make the outside the main verb, the outside is read first and is in closest proximity to the nouns.

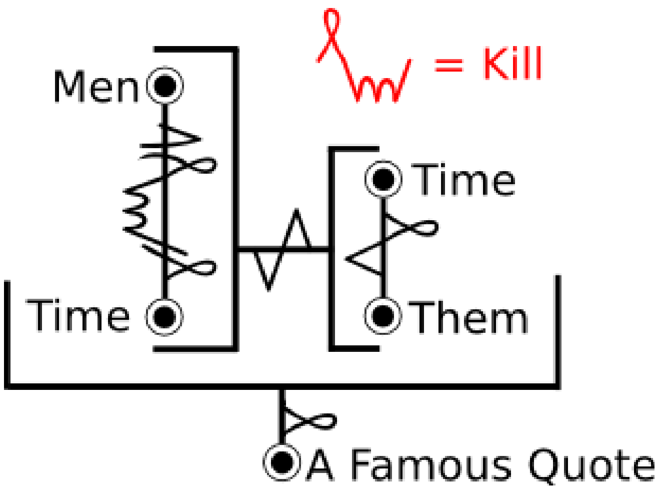
For now just consider compound verbs similar to adverbs that come before a verb “swiftly run”, impatiently wait” etc..

The only truly significant effect this choice has is “can odd length words be dominant in compounds or are they always adverbs in compounds?”, and this is only because of a previous arbitrary rule.

On the right here we add the word "kill" to the vlang dictionary.

Technically we probably shouldn't preserve the expression so literally. Something like "pass" or "waste" might be better, but for now we can just assume that Vlang "Choose-Kill" (or "cho-kill-ose") means "to willfully waste/end/terminate"

Using the alphabet and pronunciation we have set this line would be pronounced:
"men **s'** **keenee** **'o** time"



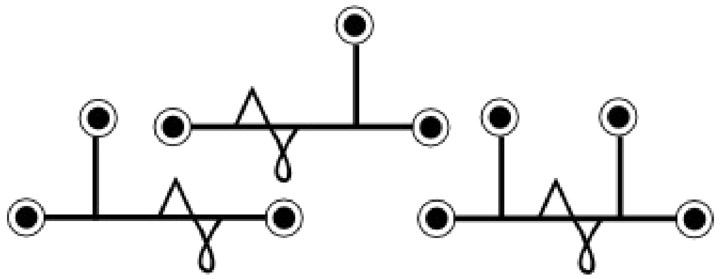
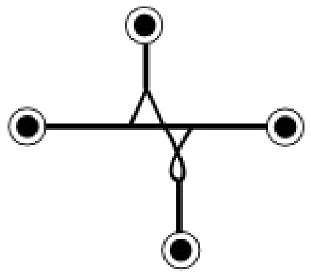
S' = the S sound with no distinct vowel 'O=the O sound alone

When we divide words 2 letters long the reading is not as straightforward. The "'O" at the end requires that you know it is the second half of an odd number length half(2=1 and 1, 1 is an odd number). This is also a problem for 6 letter long words, 10 letter long word, etc..

As per why all odd length words have been omitted, this is because even though there could be some rather simple rules to account for them (eg. allow them to split with one side larger than the other) such rules would cause conflict with potential future features like compound words (perhaps three letter words are not words, but instead a one letter word and a 2 letter word, or two 2 letter words compressed together). For now we will just try to leave as many doors open as we can and make these choices later.

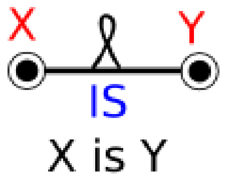
Next there are multi connections, verbs with more than 2 targets, these can be broken into 2 types.

- Forks that occur on the verb line
- Forks that occur on the Vlang text itself

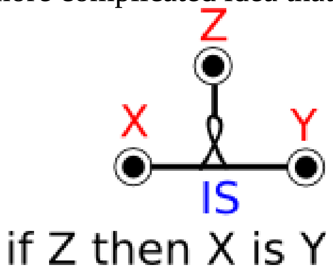



This area is still largely unexplored. So far it's still up in the air, but I have found it works well to simply consider forks on the verb line as "AND", simply apply the verb to plural pronouns of all nouns that stem from each side eg nounX=(nounX1 and nounX2 and nounX3 and etc..)

Forks on verbs is a much more complicated idea that is still up in the air. But it shows promise.

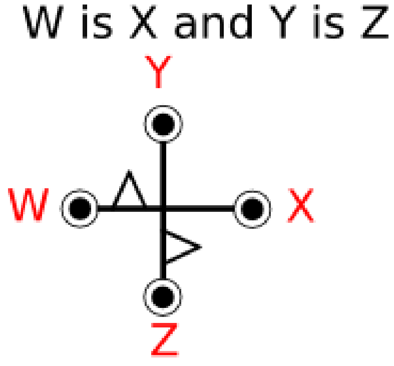


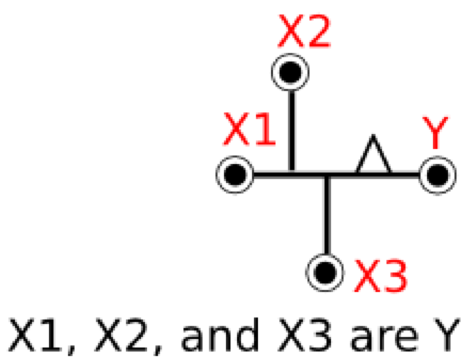
eg.



There is also the matter of anti-words vs. words in compound verbs. Basically "if you cut a verb in half, it now applies to its parent verb, not X and Y", so that basically leaves us with unused definition, word vs. anti-word no longer means X/Y, so it can mean some other binary switch as long as that switch is directional. We will skip this and leave it unused for now, both ways just mean the same thing applied to the parent verb which determines the application of the verb to the nouns.

And Finally Cross-Overs, where lines intersect .. this is... completely fine :)
It does however mean that forks cannot extend from both sides of the same point of a line.





Ands thats where it is so far. This version was designed with the aim of leaving as many option open and creating a flexible framework while exploring possibilities.

The system works very well with Dscript. Dscript works with any language and allows the text of the nouns to be flexible and cursive solid strings and glyphs that attach to the nodes making both language "melt into giant seamless spider-web/network", some visual art of this to follow of course ;)

More info on Dscript Alphabetical : http://dscript.org/dscript.pdf

Chemical Calligraphy 2.0 : http://dscript.ca/chem2.pdf
Chemical Calligraphy 1.0 : http://dscript.org/chem.pdf

Cscript – Computer Human Bi-Friendly Writing System
http://dscript.ca/cscript.pdf
think of Cscript as "somewhere between QR codes and handwriting"

Nscript – Hammer & Nail Based Layered Writing System
http://dscript.org/nailscript.pdf

Wscript – Wire Based 2D/3D Writing System
http://dscript.org/wirescript.pdf
"Mad Science"/"Technology Art" *inventions and experiments. Great DIY fun.*
http://dscript.org/inventions.pdf

Rstruct

Reversible Orthography Structure

Rstruct is a framework for designing a written language capable of reversibility.

The design of an “omni-directional script” or a script capable of being written in all directions and orientations brings up first the question of legibility. Can people actually read in all directions? Sure, there are left to right and there are right to left writing systems, but none that just go in various directions.

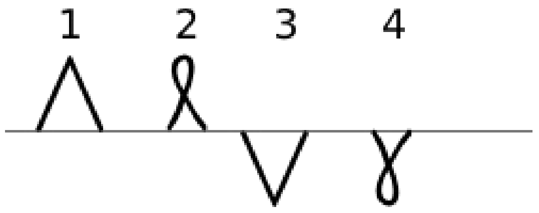
For legibility there are some questions that are key in designing such a system.

1. How easily can it be read when it appears in random orientations.
2. Is the reader expected to “rotate the visual rules” or simply rotate the paper?

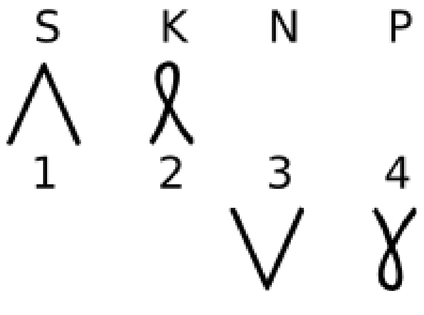
For problem 1, there is no solid answer in my humble opinion. What we can and cannot do is merely a matter of what we practice for the most part. The Rstruct “solution”, if you can call it that, is a simplistic but expandable approach. It can be summarized as “the axis is a mirror, each unit also has a mirror unit, an exact opposite exists for everything unit and compound alike”.

Solution 1 : everything has its own unique mirror value which is obtained by 180 degree rotation.

For Problem 2, we will just dodge/avoid the issue and say “well.... um... you could start by rotating the paper and hope you get better and have to do it less over time.. I suppose... kinda just hoping the start simple approach will give buffer for whatever problems come up”. ;)



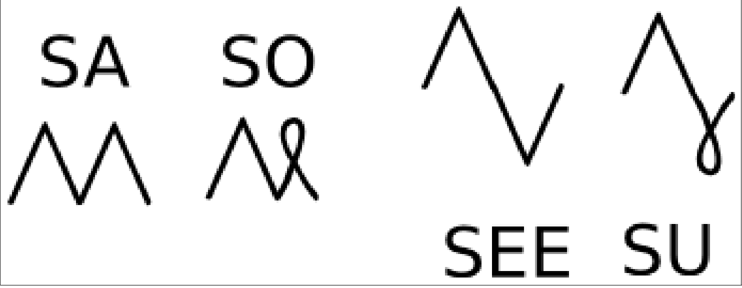
So...Everything is mirrored....
Lets look at Qscript now



As you can see Qscript has exactly 2 values and a mirror plane producing 4 total values.

This is the bare minimum for reversible orthography.
sure you could have only 1 value and 1 mirror, but then you would have very long strings. Adding more is easy, but for now.... KISS(keep it simple stupid)

So we will treat these as 4 “letters”.

	Next we add a second round of letters, this time vowels, for the second value in strings. 1=s, 11=sa, 111=sas, etc.. In reverse that would be 3=n,33=nee, 333=neen

Now we can construct words, knowing that every word has an “anti-word”, if you draw the anti-word, its mirror is the word.

So we can handle compounds of units, or words, which is great, but if we string them together then the sequence is also reversed. This requires some complex type of grammar that is reversible or a more fundamental overhaul.

Rstruct performs a fundamental overhaul in an attempt optimize for reversibility.

As oppose to stringing words together in a line, Rstruct instead nests them within each.

“Pete Sees It” for example would stead be “Pe Se It Es Te”
Pe Se It Es **Te=Pete**
Pe **Se** It **Es** Te=**Sees**
Pe Se **It** Es Te=**It**

Can people even read like that?
Considering the wide range of “ways people can read” and the flexibility my visual system seems to afford in learning to visually decipher things every day, I think it is well within the realm of possibility....this is not a complex pattern, it has very clear simple rules with no exceptions....Can it be effective or would it just be a cumbersome burden with no significant payoff? Who knows...

Now when we reverse read our string we will get the anti-words in the same sequence as the original.

To accommodate this structure Rstruct uses a few rules to prevent confusion or over-complication.

1. All values have anti-values(mirror versions)
2. All strings(words) have anti strings(words)
3. Strings 1 unit long can only stand alone or in the end(center) of a sentence
4. Strings 2 units long can exists anywhere in the sentence
5. Strings 3 units long can only stand alone or in the end(center) of a sentence
6. Strings 4 units long can exists anywhere in the sentence
7. etc...

As you can see odd number length strings are “not very welcome” they can only exists alone or at the end of a sentence.

Permutations work out as follows

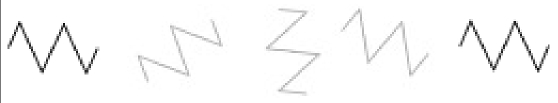
Length	**# of pairs** **(word+anti-word)**
1	2
2	8
3	32
4	128
5	512
6	2048 etc.....

So we are not at risk of running out of words, although the odd length words are a bit restrictive and those might be kinda “leftover excess” later.

Rules can be added to allow odd numbered words to encapsulate, but again, keep it simple to start ;)

*note that units that split into even number length units (eg.4=2 and 2, 8=4 and 4) are simpler to read

Last but not least we have the “mirror words”. Mirror words are words that read the same when they are reversed. The string 1-3, when revered will remain 1-3.



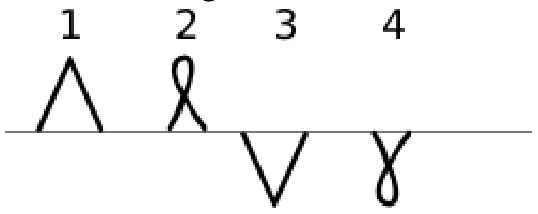
The framework itself only accomplishes a specific task, now it needs a purpose, part 3 coming soon ;)

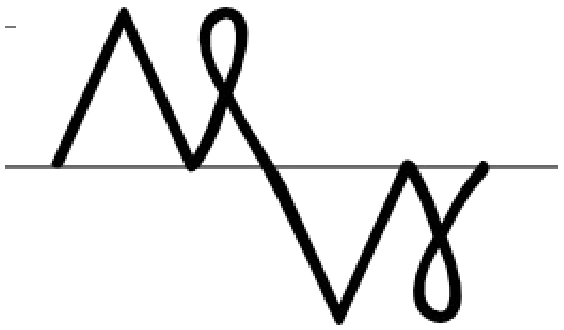
Qscript

Quaternary Script

Qscript is an elegant linear quaternary script. It is easily cursive and works great with or without lines.

The line, drawn or ruled, acts as a center, and corners or loops are drawn above or below the line. This gives 4 possibilities.

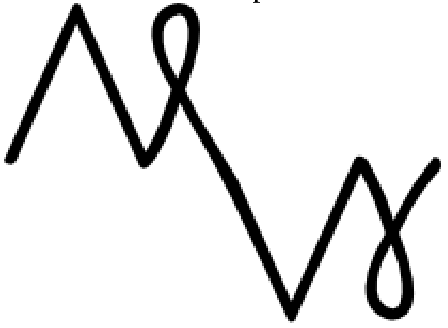


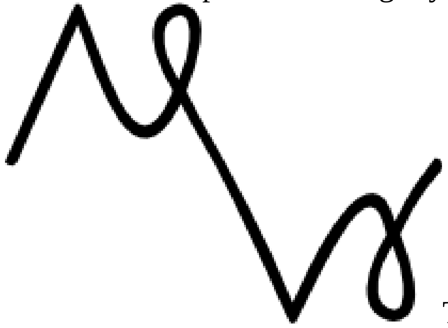


These 4 units can be combined into infinite permutations of cursive strings.

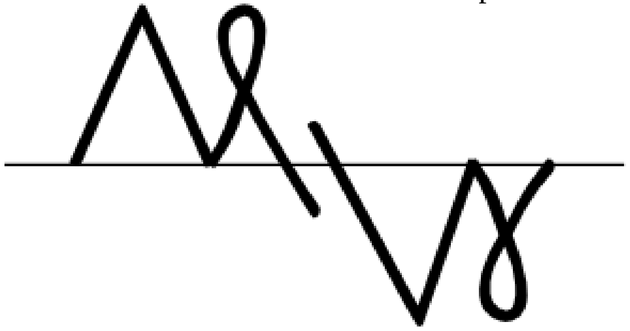
2 units =16 permutations
3 units = 64 permutations
4 units = 256 permutations
etc.....

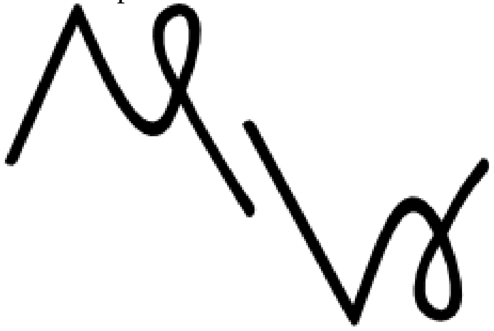
The line can be easily omitted, but when this is done units repeating the same side of the line produce small corners. I prefer to turn these into curves to avoid possible ambiguity or reading errors.





To help make cursive string terminals more obvious, I find it best to overlap the terminal past the center line. It seems to create a distinct effect for the “space character”, and also improve the visual “flow”.





Applying the script can be tricky because no languages have only 4 letters (asides perhaps DNA) and if you employ compound letters then your words cannot be made cursive (at least not without adding more rules and complexity).. that comes next ;)