THE NOTA LANG SPEC

ver 0.4

An Interpreted
English-like
markup language

NOTA

Notational[1], Functional, and oddly Multi-paradigmatic

INTRO

A simple mark-up language with a little extra capability; a bit of fun in the sun and salad days like some long lost responsibility. Simply add NOTA to any .txt, and watch what SCRIBE does.

Each line of text is evaluated as a sequence of characters, in essence, as a statement. A group of statements without white space inbetween, evaluates as a block.

Different sequences evaluate to different widgets and objects in ARTIFICIUMS domain.

Below are element tables, from simple bi-gram codes, to more complex sequences. While some flexibility is allowed, the language is fairly strict in terms of statement evaluation, such that SCRIBE will throw hands if you don't play close attention.

Headings act to limit scope. All functions can Get anywhere, but can often only be Set from a higher heading level. White space only matters on the vertical axis (interrow), not the horizontal (intrarow).

SECTION 1: TYPOGRAPHIC MARKS

TITLE HEADINGS	
>	Heading 1
>>	Heading 2
>>>	Heading 3
>>>>	Heading 4
>>>>>	Heading 5
>>>>>>	Heading 6

COMMENTS	
//	Begins a commented line

LINE BREAKS	
---	Creates a horizontal line break

[1] Refers to a mark-up language with extra capability, similar to how C++ is like C with classes.

SECTION 2: DATATYPING

BASIC FORMATS	
DATATYPE	Must be all caps Strict Evaluation
Variable	Is case-sensitive Lazy Evaluation
char	A single letter
num	A single Number

LESS BASIC FORMATS	
"String"	String: A string literal Must be enclosed in double quotes 2560 characters max per string
short ######## or -########	Short int: 1 to 8 digit Number Can be positive or negative
long ################ or -################	Long int: 1 to 16 digit number

LIST I: SIMPLE LIST	

`# string;`
`1 string || short || long`
`…`
`n string || short || long`
	Creates a simple, single entry list
	Don't need to use quotes in the function body or in header/ declaration.

LIST II: PAIR LIST	

`## "string" :`
`DATATYPE;`
`1] "string" :`
`DATATYPE`
`…`
`n] "string" :`
`DATATYPE`
	Creates a pair list with a key to value entry pair.
	Can be accessed by indices
	Enclosed with statement operators
	Strings must be in double quotes but are case-sensitive
	DATATYPE automatically inferred

`## "string";`
`1] string:`
`>> string`
`…`
`n] string:`
`>> string`
	Pair List Special notation A string-string list

SECTION 3: CONTEXTUAL OPERATORs

Relational Operators

Statement Operators

Monograms
Simple list, n list variable
: link variable, . link attribute
| link block

Bi-grams
pair list, n] linked variable
!! declare, == equivocate, =] summation,
?_ question, |_ link statement

Logical Operators

Monogram
! not

Bi-grams
&& and, || or, &| and/or

Math Operators

Monograms
+ plus, - minus, = equals,
* multiply, / divide

Bigrams

Trigrams
n^n exponent

Specialist Operators	
Evaluative Operators	
Monograms	
;	end a statement or a block signature
String Operators	

SECTION 4.0: FUNCTIONAL KEYWORDs

Functional Keywords
FUNC begins a function
PACT begins a propositional function
CACT begins a conditional function
PALGO begins an propositional algorithm
CPROP begins a conditional proposition algorithm

SECTION 4.1: CONTENT FUNCTIONs

QUESTION BLOCK	
Procor ?_"Question goes here"; |_"Any amount of text goes here" |_"Simply use the provided bi-" |_"Grams and quotes to enclose" |_"He string."	1. Displays a quote in a block 2. Somewhat simplistic but useful for notes

SECTION 4.2: FORMAL FUNCTIONs

DECLARATIVE FUNCTIONS	
`!! Variable == Variable;`	1. declares equivalence between two variables of the same type 2. Variable names are case-sensitive 3. Math operators allowed on both sides
`=] Variable;`	1. displays the value of a variable in a widget 2. variable names are case sensitive

ITERATING LOOPS	
FOR TBD: UNDEFINED	1. Definition rules go here
IF TBD: UNDEFINED	1. Definition rules go here

COLOUR CODE KEY

Colour	Represented Object
	Typography
	Formatting Mark
	Comment
	Datatyping
	Datatype (constant)
	Variable
	Integer
	String
	Operators
	Relational Operators
	Statement Operator
	Logical Operator
	Mathematical Operator
	Special Operators
	Evaluative Operator
	String Operator
	Functional Keyword