Table of Contents

[jLogo Introduction 1](#__RefHeading__106_1432511736)

[Manipulating words 2](#__RefHeading__108_1432511736)

[Variables 2](#__RefHeading__110_1432511736)

[Numbers 3](#__RefHeading__112_1432511736)

[Overriding the default number of arguments 3](#__RefHeading__114_1432511736)

[Sentences 3](#__RefHeading__116_1432511736)

[Defining procedures 5](#__RefHeading__118_1432511736)

[Terminating procedures and outputing values 6](#__RefHeading__120_1432511736)

[Local variables 7](#__RefHeading__122_1432511736)

[Dynamic scope 8](#__RefHeading__124_1432511736)

[Control structures 8](#__RefHeading__126_1432511736)

[Property lists 10](#__RefHeading__128_1432511736)

[Other jLogo notes 11](#__RefHeading__130_1432511736)

[Primitive Reference 13](#__RefHeading__132_1432511736)

[Complex Primitives 13](#__RefHeading__134_1432511736)

[Control Primitives 14](#__RefHeading__136_1432511736)

[Defining Primitives 16](#__RefHeading__138_1432511736)

[File Primitives 18](#__RefHeading__140_1432511736)

[Math Primitives 20](#__RefHeading__142_1432511736)

[System Primitives 23](#__RefHeading__144_1432511736)

[Ticker Primitives 26](#__RefHeading__146_1432511736)

[Word and List Primitives 26](#__RefHeading__148_1432511736)

[Serial Primitives 30](#__RefHeading__150_1432511736)

[TCP Primitives 30](#__RefHeading__152_1432511736)

[Image Primitives 31](#__RefHeading__154_1432511736)

[Wav Primitives 32](#__RefHeading__156_1432511736)

# jLogo Introduction

jLogo is an interpreter for the Logo programming language written in Java.

Logo is intended for use by people with no prior experience programming,

and allows the programmer to ignore distinctions that other languages

require, typically for efficiency. In Logo, there is no conceptual

difference between an identifier, a string, and a number. They are all

"words." The difference is how they are used.

\* A word by itself is interpreted as a procedure name to be called.

\* A word with a quote before it refers to the word itself.

\* A word with a colon before it is interpreted as a variable name, and the value is retrieved.

\* Numbers evaluate to themselves.

For example, in:

print 1

1

The word "print" is interpreted as a procedure to call, the word "1"

is a number that evaluates to itself, and which is passed to PRINT.

PRINT then prints out 1.

## Manipulating words

In:

print "hello

hello

The expression "hello evaluates to the word "hello" itself, which is

passed to PRINT.

FIRST and BUTFIRST, when passed a word as an argument, return the first

character, or everything but the first character. Similarly, LAST and

BUTLAST return the last character or everything but the last character.

print first "hello

h

print butfirst "hello

ello

WORD combines multiple words (2 by default) into a new word:

print word "abc "xyz

abcxyz

## Variables

Variables are assigned with MAKE. Make takes a word and a value as

arguments, and assigns the value to the word. THING takes a word as an

argument and returns the value of the word.

For example:

make "x 1

print thing "x

1

assigns the value 1 to the variable x and then fetches and prints the

value. In the MAKE statement, "x is quoted so that the word x itself

will be passed to MAKE. In the print statement, "x is quoted so that

the word itself will be passed to THING.

Fetching the value of a variable is so common that there is an

abbreviation for it. :x is the same as THING "x.

make "x 1

print :x

1

## Numbers

Logo allows the programmer to pretend that numbers are actually words

composed of decimal digits. Strings and numbers are converted back and

forth as necessary.

For example:

print first 15 + 9

2

15 and 9 are added by + to form 24, and FIRST takes the first character of 24, which is 2.

Logo also automatically converts words into numbers as necessary:

print (word "1 "2) + 13

25

## Overriding the default number of arguments

Some Logo procedures (SUM, PRODUCT, AND, OR, MIN, MAX, EQUAL?, WORD, SE,

LIST) can be passed an arbitrary number of arguments. For example, WORD

takes two arguments by default, but it can take any number of arguments.

To pass a different number of arguments than the default, enclose the

argument in parenthesis.

print (word "a "b "c)

abc

print (word "a)

a

print (word)

<-- Blank line.

The last statement prints out the empty word.

## Sentences

The primary data structure in Logo is the sentence. Traditionally,

Logo sentences have been implemented as linked lists, but in jLogo they

are implemented as arrays. This difference is only noticeable when

sentences are mutated.

For example, in jLogo:

make "a [x y z]

make "b fput "foo :a

setitem 1 :a "bar

print :a

bar y z

print :b

foo x y z

In traditional Logo interpreters, the last print statement would have shown "foo bar y z"

FIRST, BUTFIRST, LAST, and BUTLAST all work on sentences as well as words.

print first [x y z]

x

print butfirst [x y z]

y z

ITEM fetches a particular character from a word, and a particular item

from a sentence.

print item 2 "hello

e

print item 1 [the quick brown fox]

the

Notice that items are numbered starting with 1.

SETITEM will modify a member of a sentence.

make "a [the quick brown fox]

print :a

the quick brown fox

setitem 2 :a "slow

print :a

the slow brown fox

Be careful with SETITEM. If you create a recursive data structure it is easy to crash the interpreter.

make "a [x]

setitem 1 :a :a

print :a

java.lang.StackOverflowError

Sentences are used both as a data structure, and to hold lists of

commands. This ability to treat code as data and data as code is one

of the great sources of power for languages like Lisp and Logo.

LIST creates a sentence containing its arguments. It takes two arguments

by default, but can accept any number of arguments.

print list "a "b

a b

print (list "a "b [x y z] 123)

a b [x y z] 123

SENTENCE (also called SE) builds a sentence by combining its arguments.

It differs from LIST in that any list passed as an argument to SENTENCE

will have its elements separately added to the output.

print se "a "b

a b

print se [a b] "c

a b c

print se "a [b c]

a b c

print (se "a "b [x y z] 123)

a b x y z 123

## Defining procedures

jLogo treats input from the terminal differently than input from a file.

When reading from a file, lines beginning with "to" and "define" are

processed and create new procedures.

For example:

to foo

print 1

end

defines a procedure named "foo" that prints the number 1.

to bar :x

print :x

end

defines a procedure named "bar" that prints its input.

If the above definition is placed in a file named "test.logo" then these procedures can be loaded and used as follows:

load "test

foo

1

bar 5

5

Notice that the file extension ".logo" is not specified in the LOAD command.

Within a jLogo procedure, newlines are treated the same as spaces,

and commands do not need to be on the same line as their arguments.

The following works just fine:

to print1

print

1

end

When the interpreter begins, it automatically loads the file

"startup.logo" and then automatically runs the procedure STARTUP.

Procedures can also be defined using DEFINE. DEFINE can be used within

a file, or from the terminal input, but it must be called differently

in each case. Within a file, DEFINE is passed an unquoted symbol,

a list of variables, and a list of commands.

For example, within a file:

define baz [x] [

print :x

]

From the terminal, or within the body of a definition, the first argument

to DEFINE must be quoted.

For example:

define "double [x] [print :x + :x]

double 21

42

## Terminating procedures and outputing values

Procedures can terminate early using the command STOP, and can output

values using the command OUTPUT.

The procedure:

to demonstrate\_stop

print 1

stop

print 2

end

will print 1 and then return to its caller. The "print 2" command will

never be executed.

demonstrate\_stop

1

The procedure:

to demonstrate\_output

print 1

output 2

print 3

end

will print 1 and then output the value 2. The "print 3" command will

never be executed.

print demonstrate\_output

1

2

## Local variables

Procedures can create local variables using LET. Let takes one argument,

a list containing pairs of local variable names and expressions for the

initial values for the variables.

For example, this procedure:

to double.twice :x

let [doubled :x + :x

doubledoubled :doubled + :doubled]

output :doubledoubled

end

will double its input twice, like this:

print double.twice 1

4

A LET statement must never attempt to create a local variable that is

already an argument or that has previously been made local in the same

procedure by another LET statement. Doing so will cause the variable

to be improperly restored when the procedure returns.

An expression in a LET statement may reference the variable being defined.

For example, with these definitions:

to testlet

let [x 1]

sublet

end

to sublet

let [x :x + 1]

print :x]

end

Running TESTLET will print 2:

testlet

2

## Dynamic scope

jLogo uses dynamic scope. This means that a procedure has access to

the variables in its callers.

For example, in:

to printx

print :x

end

to bindx :x

printx

end

PRINTX is able to access the value of x that is bound in BINDX, so a

call to BINDX will print its argument:

bindx 5

5

uLogo note: uLogo does NOT use dynamic scope.

## Control structures

Most control structures in Logo are passed a sentence containing commands

to run.

IF takes two arguments, a boolean that must be either "TRUE or "FALSE,

and a sentence of commands to be run if the boolean is "TRUE.

print 1 < 0

false

print 1 > 0

true

if 1 > 0 [print [1 is greater than 0]]

1 is greater than 0

IFELSE is like IF, but it also takes an alternate list of commands to

run if the condition is false.

ifelse 1 < 0 [print "less] [print [not less]]

not less

ifelse 1 > 0 [print "greater] [print [not greater]]

greater

IFELSE can be used as an expression if both of the sentences contain

expressions.

print ifelse "true [1] [0]

1

print ifelse "false [1] [0]

0

RUN runs its argument once.

run [print 1]

1

If RUN is used as an expression, then its argument must contain one expression.

print run [0]

0

REPEAT takes a number and a list of commands, and runs the commands that

number of times.

repeat 3 [print "hello]

hello

hello

hello

DOTIMES takes a list containing a variable and a count, and a list of

commands to run multiple times.

dotimes [i 3] [print :i]

0

1

2

DOLIST takes a list containing a variable and an expression that will

return a list, and another list of commands to run multiple times.

dolist [i [a b c]] [print :i]

a

b

c

define "opl [] [output [a b c]]

dolist [i opl] [print :i]

a

b

c

SELECTQ uses a value to decide what command to run.

selectq "b [a [print [It is a]] b [print [It is b]] c [print [It is c]]]

It is b

SELECTQ returns a value of the command returns a value.

print selectq "a [a [1]]

1

DISPATCH uses a number to decide what command to run.

dispatch 1 [[print [It is 0]] [print [It is 1]] [print [It is 2]]

It is 1

DISPATCH returns a value of the command returns a value.

print dispatch 0 [ [1] ]

1

NOTE: Unlike ITEM and SETITEM, which consider the first element of a

list to be element number 1, DISPATCH considers the first option to be

option number 0.

LOOP runs its argument over and over forever, until it is stopped by a

STOP, OUTPUT, or an error.

define "foo [] [let [i 0] loop [print :i make "i :i + 1 if :i > 2 [stop]]

foo

0

1

2

## Property lists

Every word has a property list associated with it on which the programmer

can store properties with arbitrary values. Traditionally, in Lisp and

Logo, property lists were stored as lists, so storing large numbers of

properties on a word could incur a performance penalty, but jLogo uses

hash tables to store property lists, so there is no performance penalty

associated with storing a large number of properties on a word.

This PUT statement stores the value 1 into the property "BAR on the word

"FOO.

put "foo "bar 1

This GET expression fetches the "BAR property of the word "FOO.

print get "foo "bar

1

GET returns the empty sentence when the word does not contain the property

in question.

This PLIST expression fetches the entire property list for the word "FOO.

print plist "foo

bar 1

ERPLIST erases the property list for its argument.

erplist "foo

print plist "foo

<-- Blank line.

## Other jLogo notes

jLogo does not optimize tail calls.

define "foo [] [foo]

foo

java.lang.StackOverflowError

If RUN must return a value, then it is only allowed to contain one

expression. No commands may be before the expression. If run as a

command, then RUN may be passed multiple commands.

For example:

run [print 1 print 2]

1

2

print run [1]

1

print run [print 1 2]

1

print didn't output to run

Arguments and infix operators must be separated by spaces, but parenthesis

do not need to be separated by spaces from other tokens.

Unused symbols are never garbage collected. This is handled by not

interning the strings resulting from WORD, FIRST, BF, LAST, BL, CHAR,

ITEM, REVERSE, SUBSTRING, UCASE, REPLACE, SPLIT, etc. Words are only

interned explicitly with INTERN or when used as variables, property lists,

or procedure names.

Symbols are case insensitive, but they preserve the case that was used

the first time they were created.

For example:

print "FOO

FOO

print "foo

FOO

print "bar

bar

print "BAR

bar

jLogo does not have a "!=" operator, even though uLogo does.

When jLogo is run using jl.sh, bash collapses multiple spaces into just

one space.

For example:

print "|x x|

x x

This can be fixed by adding this to jl.sh: IFS=""

LET adds to a restore list, so repeated LET's inside of a loop will

consume large amounts of memory.

The restore list is added to at the end, but processed from the start,

which creates some strange behavior if you use multiple LET's in the

same procedure on the same variable.

.define "foo [] [let [x 1] let [x 2]]

.make "x 10

.foo

.print :x

1

When a procedure returns, the parameters are restored before the LET

restore list, which creates strange results if the same variable is both

a parameter and a LET variable.

.define "bar [x] [let [x 1]]

.make "x 10

.bar 20

.print :x

20

# Primitive Reference

## Complex Primitives

COMPLEX real imaginary

Returns a complex number with the given real and imaginary parts.

RE complex

Returns the real part of a complex number.

IM complex

Returns the imaginary part of a complex number.

MAG complex

Returns the magnitude of a complex number.

PHASE complex

Returns the phase angle of a complex number.

CSUM complex1 complex2

Returns the sum of two complex numbers.

CDIFF complex1 complex2

Returns the difference of two complex numbers.

CPROD complex1 complex2

Returns the product of two complex numbers.

CQUOT complex1 complex2

Returns the quotient of two complex numbers.

CLIST length

Returns a list filled with 0+0i.

CNTH position complex\_list

Returns the complex number at the given position in the given list of

complex numbers. position is zero based.

CSETNTH position complex\_list new\_value

Sets the complex number at the given position in the given list to

new\_value. Converts new\_value into a complex number if it is not already

one.

FFT complex\_list

Computes a Fast Forier Transform on the values in complex\_list and

returns the result.

IFFT complex\_list

Computes an inverse Fast Forier Transform on the values in complex\_list

and returns the result.

FNTH position byte\_array

Fetches a floating point value, represented as four consecutive bytes

in the byte\_array, from positions (position\*4) through (position\*4+3).

FSETNTH position byte\_array new\_value

Stores a floating point value, represented as four consecutive bytes,

in the byte\_array, from positions (position\*4) through (position\*4+3).

## Control Primitives

REPEAT count block

Runs the block repeatedly, count number of times.

IF condition block

Runs the block if condition is TRUE.

IFELSE condition block1 block2

Runs block1 if condition is TRUE and block2 if condition is FALSE.

Can be used as an expression if block1 and block2 each contain

one expression.

STOP

Terminates the current procedure without outputting a value.

OUTPUT value

Terminates the current procedure and outputs the given value.

DOTIMES [variable count] block

Runs block count times, giving variables the values 0 through

count-1 each time.

DOLIST [variable list] block

Runs block once for each member of list, assigning each member to

variable in turn.

CAREFULLY block1 block2

Runs block1. If an error occurs while running block1, then run block2.

Within block2, the error that terminated block1 can be retrieved using

ERRORMESSAGE.

ERRORMESSAGE

Can be used within the second block of a CAREFULLY statement.

Returns the error that terminated the execution of the first block.

UNWIND-PROTECT block1 block2

Runs block1. Then runs block2.

Will run block2 even if block1 is terminated by an error.

ERROR error\_message

Throws an error that includes the given error\_message.

DISPATCH number list\_of\_blocks

Runs one of the blocks in list\_of\_blocks based on the value of

number. Zero based. For example:

dispatch 1 [[print "a] [print "b] [print "c]]

b

Can be used as an expression.

print dispatch 1 [["a] ["b] ["c]]

b

RUN block

Runs the block. Can be used as an expression if block contains

one expression.

LOOP block

FOREVER block

Runs the block repeatedly until the current procedure is terminated

with STOP or OUTPUT.

SELECTQ item list\_of\_values\_and\_blocks

Tests item against each value, and runs the corresponding block if

a match is found.

selectq "b [a [print 123] b [print 456] c [print 789]]

456

Can be used as an expression:

print selectq "b [a [123] b [456] c [789]]

456

If a value begins with a : then it will be interpreted as a variable.

make "x "b

print selectq "b [a [123] :x [456] c [789]]

456

Contains a bug that causes strings to not match symbols.

selectq (word "b) [a [print 123] b [print 456] c [print 789]]

<--- No output.

Can be worked around with INTERN:

selectq (intern (word "b)) [a [print 123] b [print 456] c [print 789]]

456

STOPME

Throws an empty error. Equivalent to: ERROR "||

## Defining Primitives

MAKE name value

Assigns the value to the name.

DEFINE name argument\_list code

Defines a procedure associated with name, taking the given arguments,

and executing the given code.

define "double [x] [output :x \* 2]

print double 5

10

LET [variable expression ...]

Takes any number of variable/expression pairs. For each pair, evaluates

the given expression, creates a new local variable by the given name,

and assigns to it the value of the expression.

Due to a bug in the interpreter, LET must NEVER be used more than once

with the same variable in the same procedure. LET must also NEVER be

used on a variable that is a parameter to the current procedure.

THING name

Outputs the value assigned to the given name.

make "x 10

print thing "x

10

PUT name property value

Associates the given value with the given property on the given name's

property list.

put "foo "bar 10

print get "foo "bar

10

print plist "foo

bar 10

GET name property

GETP name property

Outputs the value of the given property of the given name.

See example under PUT.

PLIST name

Returns the property list for the given name. This consists of

alternating property names and values. See the example under PUT.

ERPLIST name

Erases the property list for the given name.

NAME? name

Returns TRUE of name has a value as a variable. Otherwise returns FALSE.

print name? "foo

false

make "foo 1

print name? "foo

true

DEFINED name

Returns TRUE if name is defined as a procedure. Otherwise returns FALSE.

print defined? "foo

false

define "foo [x] [print :x]

print defined? "foo

true

CLEARNAME name

Removes any value assigned to name.

print name? "x

false

make "x 1

print name? "x

true

clearname "x

print name? "x

false

QUOTE thing

Returns thing as is if thing is a list.

Adds quote marks to the beginning of thing otherwise.

Useful when creating lists to be executed with RUN or similar

primitives.

print quote first [foo]

"foo

print quote 6

"6

INTERN string\_or\_symbol

Forces the argument to be an interned symbol. Very rarely needs to be

called explicitly. Sometimes needed to address a bug in SELECTQ.

NARGS procedure\_name

Returns the number of arguments expected by the given procedure.

print nargs "make

2

## File Primitives

FILETOSTRING filename

Returns the contents of the named file as a string.

RESOURCETOSTRING resource\_path

Returns the contents of the named resource (file that was packaged into

the running .jar file) as a string.

LOAD filename

Loads jLogo TO and DEFINE definitions from the named file.

RELOAD

Reloads startup.logo.

STRINGTOFILE filename string

Overwrites the named file with the contents of the string.

FILE? filename

Returns TRUE if the named file exists.

SETREAD string

Sets the string as the source to be read by READLINE and EOT?

The actual contents of the string are parsed. The string is not

treated as a filename.

READLINE

Returns the first line of the string being read (as set by SETREAD) as

a string.

EOT?

Returns TRUE of READLINE is at the end of the input text (as set by SETREAD).

LINEBACK

Unreads the last line read by READLINE.

FILENAMEFROMPATH path

Returns the file portion of the path (the part after the last /).

DIRNAMEFROMPATH path

Returns the directory portion of the path (the part before the last /).

DIR directory\_path

Returns a list of files contained in the directory indicated by

directory\_path. Only the file name itself is returned for each file.

FILES

Returns a list of files contained in the directory indicated by

directory\_path. The full path to each file is returned.

SETFREAD filename

Make the file indicated by filename be the source of text for following

calls to FREADLINE, FEOT? and FCLOSE.

FREADLINE

Returns a line of the file previously opened by SETREAD.

FEOT?

Returns TRUE if there are no more lines for FREADLINE to read.

FCLOSE

Closes the file previously opened by SETREAD.

PRFOPEN filename

Opens the named file for appending with PRFPRINT.

PRFPRINT string

Appends the string and a newline to the file previously opened with PRFOPEN.

PRFCLOSE

Closes the file previously opened with PRFOPEN.

LOGOPEN filename

Begins logging the contents of the terminal to the named file.

Equivalent to DRIBBLE in other Logo implementations.

LOGCLOSE

Stops logging the contents of the terminal.

Equivalent to NODRIBBLE in other Logo implementations.

ERFILE filename

Erase the named file.

SERIALIZE filename object

Writes a serialized version of object to filename.

BYTESTOFILE filename byte\_array

Writes the given byte\_array to the given file. Overwrites the file.

FILETOBYTES filename

Creates a byte array containing the contents of the named file.

FILEPARTTOBYTES filename offset length

Creates a byte array containing the length bytes from the given

offset in the named file.

SETMODDATE filename moddate

Sets the modification date for the named file.

MKDIR dirname

Creates the named directory.

FILELEN filename

Returns the length of the named file.

## Math Primitives

SUM num1 num2

(SUM num ...)

num1 + num2

Returns the sum of two numbers.

REMAINDER num1 num2

num1 % num2

Returns the remainder when num1 is divided by num2.

The result will have the same sign as num1.

DIFFERENCE num1 num2

DIFF num1 num2

num1 - num2

Returns num1 minus num2.

PRODUCT num1 num2

(PRODUCT num ...)

num1 \* num2

Returns num1 times num2.

QUOTIENT num1 num2

num1 / num2

Returns num1 divided by num2.

GREATER? num1 num2

num1 > num2

Returns TRUE if num1 is greater than num2.

LESS? num1 num2

num1 < num2

Returns TRUE if num1 is less than num2.

INT num

Returns the integer part of num, discarding the fractional part.

Equivalent to truncating num.

MINUS num

Returns a number of the same magnitude as num but with opposite sign.

Equivalent to (0 - num).

ROUND num

Rounds num to the nearest integer. Numbers ending in .5 are rounded

towards positive infinity.

SQRT num

Returns the square root of num.

SIN num

COS num

TAN num

Returns the sin, cos, or tangent of num.

num is measured in degrees.

ABS num

Returns the absolute value of num.

Equivalent to IFELSE num < 0 [minus num] [num]

POWER base exponent

Returns the base raised to the exponent'th power.

ARCTAN num

Returns the arctan of num in degrees between -180 and 180.

PI

Returns the value of pi.

EXP num

Returns e raised to the power of num.

ARCTAN2 y x

Returns the angle in degrees represented by y/x. Uses the signs of

y and x to determine the quadrant of the resulting angle.

ARCTAN2 0 1 => 0

ARCTAN2 1 0 => 90

ARCTAN2 0 -1 => 180

ARCTAN2 -1 0 => -90

LN num

Returns the natural logarithm of num.

LOGAND num1 num2

LOGIOR num1 num2

LOGXOR num1 num2

Returns the bitwise and, inclusive or, or exclusive or of num1 and num2.

LSH num shift

Returns num shifted by shift bits to the left. shift may be a negative

value.

AND bool1 bool2

(AND bool ...)

OR bool1 bool2

(OR bool ...)

Returns the logical and or or of bool1 and bool2.

Can accept an arbitrary number of arguments.

NOT bool

Returns TRUE of bool is FALSE, and FALSE if bool is TRUE.

RANDOM int

Returns a random integer in the range of 0 through int-1.

MIN num1 num2

(MIN num ...)

MAX num1 num2

(MAX num ...)

Returns the maximum or minimum value of num1 and num2.

Can accept an arbitrary number of arguments.

NUMBER? thing

Returns TRUE if thing is a number.

EQUAL? thing1 thing1

(EQUAL? thing ...)

EQ thing1 thing1

thing1 = thing2

Returns TRUE if thing1 and thing2 are equal to each other.

EQUAL? can accept an arbitrary number of arguments.

Performs recursive comparison on lists.

## System Primitives

RESETT

Resets a millisecond timer.

TIMER

Returns the number of milliseconds that have elapsed since the last

time RESETT was called.

WAIT tenths\_of\_a\_second

Pauses execution for tenths\_of\_a\_second tenths of a second.

TRUE

Returns the word "TRUE.

FALSE

Returns the word "FALSE.

HEXW value width

Returns a string containing a hexadecimal representation of value

that is zero padded to be at least width characters long.

width may not be greater than 9.

OCTW value width

Returns a string containing an octal representation of value

that is zero padded to be at least width characters long.

width may not be greater than 18.

DECW value width

Returns a string containing a decimal representation of value

that is zero padded to be at least width characters long.

width may not be greater than 18.

TAB

Returns a word containing just one tab character.

Equivalent to CHAR 9

CLASSOF thing

Returns the Java class used internally to represent thing.

CLASS classname

Returns the Java class object for the Java class named by classname.

STRING thing

Converts thing into a string. For most objects, this will be the same

string that would normally be printed to the terminal by PRINT, but for

byte arrays it will become a string with each element in the byte array

being converted into a character in the string.

%NOTHING%

Returns an instance of the Java class Nothing. Not particularly useful.

CPRINT thing

Prints thing directly to stdout. The printed value does not appear

in the any open log file.

PRINT thing

Prints thing. The printed value does appear in any open log file.

PRINTC thing

Prints thing directly to stdout, without a trailing newline.

The printed value does not appear in the any open log file.

HEXPARSE hex\_string

Parses hex\_string as a hexadecimal number and returns the appropriate

value.

SCANHEX hex\_string result\_list format\_list

Command.

Mutates result\_list setting each element to a number created by

scanning a portion of hex\_string as described by format\_list.

Two numbers in format\_list describe the substring of hex\_string to use

for each element of result\_list.

For example:

make "a makelist 2

scanhex string "A10 :a [0 1 1 2]

print :a

10 16

In the format list, the first two elements, 0 and 1, say to set the first

element of :a to the substring of "A10 starting at offset 0 and containing

1 character, so "A is scanned and becomes decimal 10. The third and fourth

elements, 1 and 2, say to use the substring starting at offset 1 and

containing two characters, so "10 is scanned and becomes decimal 16.

EXEC command

Runs command as a shell command. Returns a string containing the

output of command.

BLINDEXEC command

Runs command as a shell command. Throws away any output.

GETPROPERTY java\_system\_property

Fetches the named java system property.

IGNORE thing

Command. Accepts one input and does nothing with it.

Useful when you wish to execute a function that returns a value

but you do not wish to do anything with the value.

QSYM string

Returns a quoted version of string.

NOW

Returns the current time in Unix format.

DATEFORMAT format time timezone

Returns a formated representation of time using the given format

and adjusted for the given timezone.

The format is interpreted by Java's SimpleDateFormat as described in

https://docs.oracle.com/javase/7/docs/api/java/text/SimpleDateFormat.html

The time zone is interpreted by Java's TimeZone class as described in

https://docs.oracle.com/javase/6/docs/api/java/util/TimeZone.html

For example, to print the current time on the west coast:

print dateformat "|y-M-d-H-m-s| now string "|America/Los\_Angeles|

To print the current time on the west coast when daylight savings time

is not in effact:

print dateformat "|y-M-d-H-m-s| now string "|GMT-0800|

To print the current time on the west coast when daylight savings time

is in effact:

print dateformat "|y-M-d-H-m-s| now string "|GMT-0700|

DATEPARSE pattern datestring

Parses a given string describing a date according to the given pattern.

CLARG argument\_number

Returns a command line argument, or the empty list if argument\_number

is greater than the number of provided command line arguments.

SETINDENT string

Sets the indent string. The indent string is printed before every

line printed by PRINT.

SETHANDLELINE bool

Turns on of off the use of HANDLE-LINE.

If bool is TRUE, then all lines not beginning with a period will be

handled by the HANDLE-LINE Logo procedure (typically for running uLogo

code), and only lines beginning with a period will be interpreted as

jLogo code.

If bool is FALSE, then input lines will be treated as jLogo code to be

interpreted immediately.

OCTAL

Sets the input base to base 8.

DECIMAL

Sets the input base to base 10.

ASK prompt

Prints the prompt without a newline, and waits for the user to enter

a value.

ANSWER

Returns the answer entered by the user during a previous call to ASK.

## Ticker Primitives

STARTTICKER milliseconds

Causes the word TICKTASK to be run repeatedly, with a delay of the

given number of milliseconds between each call.

STOPTICKER

Stops TICKTASK from being run automatically.

## Word and List Primitives

FIRST thing

LAST thing

Returns the first or last member of thing. If thing is a list, returns

the first or last member of the list. If thing is a word, returns a new

word containing the first or last letter of the word.

WORD thing1 thing2

(WORD thing ...)

Creates a new word formed by concatenating the arguments together.

BUTFIRST thing

BF thing

If thing is a list, returns a new list that does not contain the first

member of thing.

If thing is a word, returns a new word that does not contain the first

letter of thing.

BUTLAST thing

BL thing

If thing is a list, returns a new list that does not contain the last

member of thing.

If thing is a word, returns a new word that does not contain the last

letter of thing.

FPUT thing list

Returns a new list containing thing as its first element, and all of the

members of list.

LPUT thing list

Returns a new list containing thing as its last element, and all of the

members of list.

ITEM index collection

Returns the member of collection indicated by index. Index is one based.

If collection is a list, returns the indicated member of the list.

If collection is a word, returns a new word containing the indicated letter

of the word.

NTH index collection

Returns the member of collection indicated by index. Index is zero based.

If collection is a list, returns the indicated member of the list.

If collection is a word, returns a new word containing the indicated letter

of the word.

EMPTY? thing

Returns TRUE if thing is the empty list or the empty string.

COUNT collection

Returns the number of items in the collection.

If the collection is a list, then COUNT returns the number of items in the

list.

If the collection is a string, then COUNT returns the number of letters in

the string.

WORD? thing

Returns TRUE if thing is a word.

LIST? thing

Returns TRUE if thing is a list.

MEMBER? thing collection

Returns TRUE if thing is a member of collection, or if both thing and

collection are strings and thing is a substring of collection.

ITEMPOS thing collection

Returns the one based index of thing in collection if collection contains

thing, or if collection is a string and thing is a substring of collection.

Throws an error if collection does not contain thing.

SETITEM position list new\_value

Modifies list so that the given position will contain the new\_value.

position is one based.

SETNTH position list new\_value

Modifies list so that the given position will contain the new\_value.

position is zero based.

REMOVEITEM thing list

Returns a copy of LIST from which the first copy of thing has been removed.

REMOVEITEMPOS position list

Returns a copy of LIST from which the thing at the given position has

been removed.

position is one based.

SENTENCE thing1 thing2

SE thing1 thing2

(SE thing ...)

Returns a new list formed by concatenating any lists in the arguments

and including any non-lists.

For example:

print (se "a [b c] "d [e f])

a b c d e f

LIST thing1 thing2

(LIST thing ...)

Returns a list containing the arguments.

For example:

print (list "a [b c] "d [e f])

a [b c] d [e f]

MAKELIST count

Returns a list of length count. All members are initialized to the

empty list.

COPYLIST list

Creates a new list with the same members as the argument.

PARSE string

Parses the given string into a list of jLogo values.

CHAR ascii\_value

Creates a string of length one, whose only character has the given

ascii\_value.

ASCII string

Returns the numerical ascii value of the first character in the string.

REVERSE list

Returns a new list containing the same items as the argument but in

reversed order.

SUBSTRING string offset length

Returns a new string formed by skipping the first offset characters

in string and including the next length characters.

UCASE string

LCASE string

Converts a string to upper or lower case.

REPLACE string old new

Replaces all instances of old in string with new.

SPLIT string delimiter

Returns a list formed by splitting string up in each place where delimeter

appears.

SPLIT "axbxcxd "x => [a b c d]

BYTEARRAY length

Creates a new byte array of the given length.

NFORMAT format number

Returns a string containing the number formatted as specified by the format.

The format is the portion of a printf format string that goes between the %

and the f.

UNZIP byte\_array offset ilen olen

Does something with zipped data.

## Serial Primitives

OPENPORT name

Opens the named serial port.

CLOSEPORT

Closes the current serial port.

PORTNAME pid vid

Returns the name of the port with the given pid and vid.

PORTNAMES pid vid

Returns a list containing the path components to the port with the given

pid and vid.

SETPORTPARAMS baud databits stopbits parity

Sets the serial port parameters.

SEND int

Sends the character with the ascii value given by the int.

SENDL int\_list

Sends one character for each integer in the list.

RECC

Reads one byte and returns the ascii value of that byte as a number.

CLEARCOM

PORTHANDLE

USBINIT

MODEMCTRL

## TCP Primitives

CONNECT host port

Opens a tcp connection to the given host and port.

TCPRECV

Read one byte from the currently open tcp stream.

TCPRECVN count

Read count bytes from the currently open tcp stream. Returns them as a

byte array. If you want to manipulate this as a string, you need to convert

it using STRING.

TCPAVAIL

Returns TRUE if there are any characters ready to read on the tcp connection.

TCPSEND thing

If thing is a number, write one byte with that value.

If thing is a string or byte array, write the string or byte array

on the currently open tcp stream.

TCPTIMEOUT milliseconds

Sets the maximum amount of time that a TCP read will block for if there

is no data waiting to be read.

STARTSERVER port

Starts listening on the given TCP port.

IPADDRESS

Returns the local ip address.

NEWCONNECTION

When running as a server, returns TRUE if there has been a new connection

since the first call to STARTSERVER or the last call to NEWCONNECTION.

READURL url

Fetches the web page indicated by url and returns its contents.

BASE64 byte\_array

Returns a string containing the base64 encoded representation of the

byte\_array.

DIGEST string

Returns the base64 encoded version of the SHA-1 digest of the given

string.

MD5 string

Returns the hex encoded version of the MD5 digest of the given string.

URLENCODE string

URL encodes the string.

Converts characters that cannot appear in a URL into appropriate %XX

escape sequences and converts spaces into +'s.

URLDECODE string

Decodes a URL encoded string.

## Image Primitives

TODO

## Wav Primitives

TODO