

# Logo Documentation

## What is Logo

Logo is a programming language that dictates the drawing of a sketch.

The drawing vehicle is an avatar that can be moved inside the 3-dimensional space, leaving a trail behind it. The avatar is initiated at the origin with orientation towards the y axis.

Movement commands indicate the movement of the avatar with reference to its current position and orientation. Exceptionally, the set commands can place the avatar to a specific position with reference to the absolute coordinate system of the space.

## Movement Commands

These are the basic commands that control the movement of the avatar.

All following commands do not return any value

Command Name	Arguments	Description	Example
fd or forward	1 (steps distance)	Move forwards	fd 50
bk or backward	1 (steps distance)	Move backwards	bk 50
rt or right	1 (degrees)	Turn right	rt 90
lt or left	1 (degrees)	Turn left	lt 90
up	1 (degrees)	Turn up	up 90
dn or down	1 (degrees)	Turn down	dn 90
rr or roll_right	1 (degrees)	Roll right	rr 90
rl or roll_left	1 (degrees)	Roll left	rl 90
home	-	Move to home position	home
setx	1 (position)	Move to specific x coordinate	setx 50
sety	1 (position)	Move to specific y coordinate	sety 50
setz	1 (position)	Move to specific z coordinate	setz 50
setxyz	3 (position)	Move to specific point (x,y,z)	setxyz 50 50 50

## Position Commands

These are the commands that provide information about the current position of the avatar.

All following commands do not accept any arguments.

Their return value can be used by any Logo command accepting arguments

Command Name	Return value unit	Description	Example
getx	coordinate	returns the current position on x axis	getx
gety	coordinate	returns the current position on y axis	gety
getz	coordinate	returns the current position on z axis	getz

## Trail appearance Commands

These are the commands that configure whether the avatar movement leaves a trail and how it should appear. Once called, they affect all consecutive movement commands until another change is made.

All following commands do not return any value

Command Name	Arguments (unit)	Description	Example
penup	-	Avatar stops leaving trail	penup
pendown	-	Avatar starts leaving tail	pendown
setpensize	1 (pixels)	Sets the trail width	setpensize 4
color	3 (0-255 r g b)	Sets the trail color in RGB space	color 255 20 40

## Output Commands

These are the commands that enable output to the user

All following commands do not return any value

Command Name	Arguments (unit)	Description	Example
print	1	Prints a value to the terminal	print "starting"
label	1	Displays a value on the sketch	label "corner"

## Arguments

An argument in Logo can be:

- Any number. E.g. 2, 3.14, 2.76e3
- The value of a defined variable name, using the prefix ':'. E.g. fd :var
- A word literal, using the prefix ' '. Only a few commands can accept this kind of argument  
E.g. print "helloWorld label "corner
- Any function or command that returns a value e.g. print getx
- Any expression with a combination of arguments and arithmetic operators (+, -, \*, /) or comparison operators (<, >, <=, >=, =)  
E.g. fd :n + 10 rt 360 / 5 print :k <= 5  
The result of a comparison is 1 if the comparison is true and 0 if it is false
- Any expression can be sub-grouped using parentheses '()'.  
This is also useful to determine the priority of operations

## Variables

Variables are places in memory that we can store a numeric value.

The variables can be assigned names and values by the user, using the **make** command

After setting it, the value of that variable can be accessed by using the prefix ‘:’:

*Syntax:*

```
make <variable name literal> <variableValue>
:< variable name literal>
```

*Example :*

```
Make `var 50
fd :var (now equivalent to fd 50)
```

## Program Flow Control

### Conditionals

A block of code can be executed conditionally, by using the command **if**

*Syntax:*

```
If <condition> [ <commands to execute if condition is true> ]
<condition>: An argument expression that can be evaluated as true or false
< commands to execute if condition is true > : any program code
```

*Example:*

```
If :n < 5 [ print `lessThanFive ]
```

Similarly, two different blocks of code can be executed, depending on the truth value of a condition, using **ifelse**

*Syntax*

```
Ifelse <condition > [ <commands to execute if condition is true> ] [ <commands to execute if condition is false> ]
```

*Example*

```
Ifelse :n < 5 [ print `lessThanFive ] [ print `higherOrEqualToFive ]
```

## Loops

A block of code can be executed repeatedly n times, with the command **repeat**:

Syntax:

Repeat <number of executions> [ <commands> ]

Example:

```
repeat 4 [ fd 10 rt 90 ]
```

A block of code can be executed repeatedly, as long as a condition is true, with the command **while**

Syntax:

While <condition> [ <commands>]

Example:

```
make "n 4
while :n > 0 [ fd 10 make "n :n - 1]
```

Similarly, a block of code can be executed repeatedly, as long as a condition is false, with the command **until**

Syntax:

until <condition> [ <commands>]

Example:

```
make "n 4
until :n = 0 [ fd 10 make "n :n - 1]
```

## Functions

- A function is a part of code (called function's body) that can be given a name and can be executed whenever this name is called inside the program.
- A function can accept any predefined number of parameters and use them inside its body as variables. The definition of the parameters names is by using the prefix ':'
- A function can optionally return a value to the command that called it. The return statement can be at any position in the body and the execution will stop once it reaches it
- Functions are called by using their names, followed by their parameters arguments
- ***In fact, all Logo commands can be considered as functions***

Syntax:

to <functionName> <list of parameter names> <body> <return statement> end

Example:

```
to square :side repeat 4 [ fd :side rt 90 ] end
to add :a :b return :a + :b end
square 50           – will draw a square
print add 2 3       – will print the number '5' on the terminal
square add 25 25    – will first call function add with parameters 25, 25 and then call square with
parameter the output of function add, which will be 50
```

## Mathematical commands

These are some commands that are useful for performing mathematical calculations

Name	Arguments	Return value units	Description	Example
sqrt	1		compute square root	sqrt 4
pow	2 (base, exponent)		raises the base to the exponent	pow 2 3
mod	2 (Divisor, divider)		remainder of integer division	mod 4 3
cos	1 (degrees)		cosine of angle	cos 60
sin	1 (degrees)		sine of angle	sin 30
tan	1 (degrees)		tangent of angle	tan 30
arccos	1	degrees	inverse cosine	arccos 0.5
arcsin	1	degrees	inverse sine	arcsin 0.5
arctan	1	degrees	inverse tangent	arctan 4
ln	1		natural logarithm	ln 7
log	1		logarithm with base 10	log 150
exp	1		e raised to value	exp 2

## Logical commands/functions

These are some commands that are useful for performing logical operations between arguments that can be evaluated for their truth value

Name	Arguments	Description	Example
or	2	Returns true if any of the arguments is true	or :n < 5 :n > 10
and	2	Returns true if both of the arguments are true—	and :n < 5 :k < 5
not	1	Returns true if the argument is false	not :n = 5

## Random generation

The following command is useful to produce random numbers

Name	Arguments	Description	Example
rand	1	Returns an integer random number in the range of [0, n) where n is the argument*	rand 100

\*If the drawing is repeated in every frame of the display, the same random number will be returned at every execution. Alternatively, use **randcrazy** for a different output at each frame

## Comments

Comments are notes on the source code that are used for human readability and are not part of the actual program.

Comments in Logo start with the semicolon ';'. Any appearance of the semicolon will make the interpreter ignore the rest of the specific line when it appeared

## Example

```
Fd 20 ;move a bit forward
```