

# RCX Brick basic command reference

This command reference was modified from the Cricket Logo language reference (<http://el.www.media.mit.edu/people/mikhak/sd98/cricketlogo.html>) written by Fred Martin and Robbie Berg. Cricket Logo was designed and implemented by Brian Silverman with Robbie Berg and Fred Martin. This document is a reduced version containing only a set of basic commands. Edited by Arnan (Roger) Sipitakiat.

## Motors

Motor commands are used by first selecting the motor (using **a**, **b**, or **ab**,) and then telling it what to do (e.g., **on**, **off**, **rd**, etc.).

<b>a</b> ,	selects motor A to be controlled.
<b>b</b> ,	selects motor B to be controlled.
<b>c</b> ,	selects motor C to be controlled
<b>ab</b> ,	selects motor A and B to be controlled.
<b>ac</b> ,	selects motor A and C to be controlled.
<b>bc</b> ,	selects motor B and C to be controlled.
<b>abc</b> ,	selects motor A, B and C to be controlled.
<b>on</b>	turns the selected motors on.
<b>off</b>	turns the selected motors off.
<b>onfor</b> <i>duration</i>	turns the selected motors on for a <i>duration</i> of time, where <i>duration</i> is given in tenths-of- seconds. E.g., <b>onfor 10</b> turns the selected motors on for one second.
<b>thisway</b>	sets the selected motors to go the "thisway" direction.
<b>thatway</b>	sets the selected motors to go the "thatway" direction.
<b>rd</b>	reverses the direction of the selected motors. Whichever way they were going, they will go the opposite way.
<b>setpower</b> <i>level</i>	sets the selected motor(s) power <i>level</i> . Input is in the range of 0 (coasting with no power) to 8 (full power).

## Timing and Sound: timer/resett, wait, beep, and play a note

The timing and sound commands are useful to cause the RCX to do something for a length of time. The timing primitives are timer/resett and wait.

<b>timer</b>	reports value of free-running elapsed time device. Time units are reported in 1 millisecond counts.
<b>resett</b>	resets elapsed time counter to zero.
<b>wait</b> <i>duration</i>	delays for a <i>duration</i> of time, where duration is given in tenths-of-seconds. E.g., <b>wait 10</b> inserts a delay of one second.

For example, one might say

**ab, on wait 20 off**

to turn the motors on for two seconds. This is equivalent to

**ab, onfor 20**

Please note that there are two different reference values for timing: 0.1 second units, used in wait and in note, and 0.001 second units, used in timer.

Sound primitive, beep and note, can also be used to cause delays.

<b>beep</b>	plays a short beep.
<b>note</b>	plays a note of a specified <i>pitch</i> and <i>duration</i> . The <i>pitch</i> <i>duration</i> value is specified in tenths-of-seconds units.

The correspondence between the numbers to define the pitch and the musical notes in the octave between middle c and high c is shown in the table below

Musical Notation	c	c#	db	d	d#	eb	e	f	f#	gb	g	g#	ab	a	a#	bb	b	c2
------------------	---	----	----	---	----	----	---	---	----	----	---	----	----	---	----	----	---	----

For example,

**note c 5**

will play a middle “c” for half a second.

## Sensors

The RCX has three sensors, named "1", "2" and "3".

- sensor1** reports the value of sensor 1, as a number from 0 to 1023.
- sensor2** reports the value of sensor 2, as a number from 0 to 1023.
- sensor3** reports the value of sensor 3, as a number from 0 to 1023.
- switch1** reports "true" if the switch plugged into sensor 1 is pressed, and "false" if not.
- switch2** reports "true" if the switch plugged into sensor 2 is pressed, and "false" if not.
- switch3** reports "true" if the switch plugged into sensor 3 is pressed, and "false" if not.

## Control

RCX Logo supports the following control structures:

- loop**  
[*body*]      repetitively executes *body* indefinitely.
- repeat**  
*times*  
[*body*]      executes *body* for *times* repetitions. *times* may be a constant or a calculated value.
- if**  
*condition*  
[*body*]      if *condition* is true, the RCX executes *body*. Note: a *condition* expression that evaluates to zero is considered "false"; all non-zero expressions are "true".
- ifelse**  
*condition*  
[*body1*]  
[*body2*]      if *condition* is true, executes *body-1*; otherwise, executes *body-2*.
- waituntil**  
[*condition*]      loops repeatedly testing *condition*, continuing subsequent program execution after it becomes true. Note that *condition* must be contained in square brackets; this is unlike the conditions for **if** and **ifelse**, which do not use brackets.
- stop**      terminates execution of procedure, returning control to calling procedure.
- output**  
*value*      terminates execution of procedure, reporting *value* as result.

## Multitasking: when, whenoff

RCX Logo contains a when primitive that allows for simple multitasking:

<b>when</b> [ <i>condition</i> ] [ <i>body</i> ]	launches a parallel process that repeatedly checks condition and executes <i>body</i> whenever <i>condition</i> changes from false to true. The <b>when</b> rule is "edge-triggered" and remains in effect until it is turned off with the <b>whenoff</b> primitive. Only one <b>when</b> rule can be in effect at a time; if a new <b>when</b> rule is executed by the program, this new rule replaces the previous rule.
<b>whenoff</b>	turns off any existing <b>when</b> rule.

For example, the following program will beep once every second, while reversing the motor direction every tenth of a second:

```
to beep-and shake
  reset
  when [timer > 1000] [beep reset]
  loop [a, onfor 1 rd]
end
```

**Note:** Do not use the **stop** primitive inside the body of the **when** command. **stop** only has meaning inside of a procedure, and the **when** command executes in its own context, outside of any procedure scope.

## Recursion

RCX Logo supports tail recursion to create infinite loops. For example:

```
to beep-forever
  beep wait 1
  beep-forever
end
```

is equivalent to

```
to beep-forever
  loop [beep wait 1]
end
```

The recursive call must appear as the last line of the procedure and cannot be part of a [control](#) structure like **if** or **waituntil**. Thus the following is not valid:

```
to beep-when-pressed
  beep wait 1
  if switcha [beep-when-pressed]
end
```

# Numbers

All arithmetic operators must be separated by a space on either side. E.g., the expression 3+4 is not valid. Use 3 + 4.

<b>+</b>	infix addition
<b>-</b>	infix subtraction
<b>*</b>	infix multiplication
<b>/</b>	infix division
<b>%</b>	infix modulus (remainder after integer division)
<b>and</b>	infix logical "and" operation (bitwise and)
<b>or</b>	infix logical "or" operation (bitwise or)
<b>xor</b>	infix logical "xor" operation (bitwise xor)
<b>not</b>	prefix logical not operation. use only with boolean values (1 and 0).
<b>random</b>	reports pseudo-random number.

# Variables

	creates a variable and assigning a value to it. For example
<b>let</b>	<i>let [foo 10]</i>
<b>[variable value]</b>	creates a variable named foo with an initial value of 10.
	<i>let [foo 10 bar 20]</i>
	creates two variables foo and bar with initial values of 10 and 20 respectively.
	Assigns a value to a variable created by let.
<b>make</b>	<i>make "foo 50</i>
<b>"variable value</b>	assigns 50 to variable foo
	<i>make "foo :foo + 1</i>
	increases the value of variable foo.

Please note that you need to put a (" in front of the variable name you want to access its value. For example

```
let [foo 10]  
repeat :foo [beep wait 5]
```

## Procedure Inputs and Outputs

Procedures can accept arguments using Logo's colon syntax. E.g.,

```
to wiggle :times  
ab,  
repeat :times [on wait 2 rd]  
end
```

creates a procedure named wiggle that takes an input which is used as the counter in a **repeat** loop.

Procedures may return values using the **output** primitive; e.g.:

```
to go  
ab,  
repeat third [on wait 10 rd]  
end  
  
to third  
if sensora < 20 [output 1]  
if sensora < 50 [output 2]  
output 3  
end
```

The go procedure will execute 1, 2, or 3 times depending on the value of sensor A.

## RCX Brick full command list

stop	loop	print	sensor1'
output	repeat	lcd-frob	prgm
if	waituntil	lcd-fritz	launch
ifelse		setprgm	every
=	a,	beep	when
>	b,		random
<	c,	send	
	ab,	ir	record
	ac,	newir?	recall
ew	bc,	powerdown	resetdp
dw	abc,		setdp
eb		sensor1	dp
db	on	sensor2	erase
	onfor	sensor3	
+	coast	switch1	aset
-	rd	switch2	aget
*	thisway	switch3	vbat
/	thatway		resetpd
%	off	resetc1	note
abs	setpower	counter1	
		resetc2	make
not	wait	counter2	waituntil
and	timer	resetc3	when
or	resett	counter3	let
xor			note