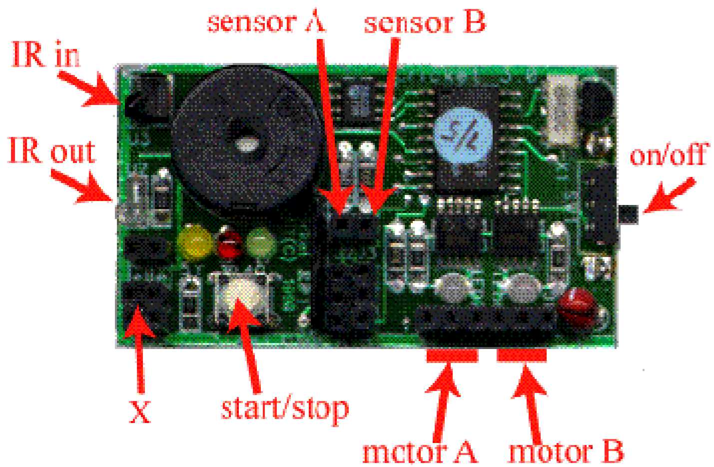
# Cricket Logo Language Reference
# "Blue Dot" Version




Cricket Logo was designed and implemented by Brian Silverman with help from Fred Martin and Robbie Berg. This language reference was written by Fred Martin and Robbie Berg. A less technical introduction can be found in the document Getting Started With Crickets[1]. For more information about Crickets visit the Cricket Home Page on the World Wide Web at:

`http://lcs.www.media.mit.edu/people/fredm/projects/cricket/`

## 1. Overview

Cricket Logo has the following features:

- control structures like `if`, `repeat`, `wait`, `waituntil` and `loop`
- motor and sensor primitives
- global and local variables
- global arrays
- procedure definition with inputs and return values
- primitives for infrared communication
- a mulitasking `when` primitive
- a 16-bit number system (addition, subtraction, multiplication, division, comparison);
- timing functions, a tone-playing function, and a random number function
- data recording and playback primitives

---

[1] Crickets are part of an ongoing research project carried out by:

*The Epistemology and Learning Group*
*MIT Media Laboratory*
*20 Ames Street Cambridge, MA 02139*

When using Cricket Logo, user programs are entered on a desktop computer and compiled into tokens which are beamed via infrared to the Cricket. (An “interface”, connected to the desktop computer’s serial port, translates these tokens into infrared signals. To download programs, both the interface and the Cricket must be turned on and their infrared ports must face each other.) Cricket Logo is a procedural language; procedures are defined using Logo `to` and `end` syntax:

```
to procedure-name
     procedure-body
end
```

When the Cricket is idle, pressing its **start-stop** push-button causes it to begin executing remote-start line 1 on the Cricket Logo screen.

When the Cricket is running a program, pressing the **start-stop** button causes it to halt program execution.

## 2. Motors

The Cricket has two motors, which are named "A" and "B". A bi-color LED indicates the state of each motor.

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.).

| | |
|---|---|
| `a,` | Selects motor A to be controlled. |
| `b,` | Selects motor B to be controlled. |
| `ab,` | Selects both motors to be controlled. |
| `on` | Turns the selected motors on. |
| off | Turns the selected motors off. |
| `brake` | Actively applies a brake to the selected motors. |
| `onfor` *`duration`* | Turns the selected motors on for a *duration* of time, where duration is given in tenths-of-seconds. E.g., `onfor 10` turns the selected motors on for one second. |
| `thisway` | Sets the selected motors to go the ``thisway" direction, which is defined as the way that makes the indicator LEDs light up green. |
| `thatway` | Sets the selected motors to go the ``thatway" direction, which is defined as the way that makes the indicator LEDs light up red. |
| `rd` | Reverses the direction of the selected motors. Whichever way they were going, they will go the |

opposite way.

`setpower` *`level`* — Sets the selected motor(s) power level. Input is in the range of 0 (coasting with no power) to 8 (full power).

## 3. Timing and Sound

The timing and sound commands are useful to cause the Cricket to do something for a length of time. 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.

`wait` *`duration`* — Delays for a duration of time, where *duration* is given in tenths-of-seconds. E.g., wait 10 inserts a delay of one second.

`beep` — Plays a short beep

`timer` — Reports value of free-running elapsed time device. Time units are reported in 1 millisecond counts.

`resett` — Resets elapsed time counter to zero.

`note` *`pitch duration`* — Plays a note of a specified *pitch* and *duration*. Increasing values of the pitch create lower tones (the *pitch* value is used as a delay counter to generate each half of the tone's square wave). The duration 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

| Pitch Number | 119 | 110 | 110 | 105 | 100 | 100 | 94 | 89 | 84 | 84 | 79 | 74 | 74 | 70 | 66 | 66 | 62 | 59 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Musical Notation | c | c# | db | d | d# | eb | e | f | f# | gb | g | g# | ab | a | a# | bb | b | c2 |

For example,

```
note 119 5
```

will play a middle “c” for half a second. Alternatively, the musical notation can be used directly:

```
note c 5
```

does the same thing.

## 4. Sensors

The Cricket has two sensors, named "A" and "B".

| | |
|---|---|
| `sensora` | Reports the value of sensor A, as a number from 0 to 255 |
| `sensorb` | Reports the value of sensor B, as a number from 0 to 255. |
| `switcha` | Reports “true” if the switch plugged into sensor A is pressed, and “false” if not. |
| `switchb` | Reports “true” if the switch plugged into sensor B is pressed, and “false” if not. |

## 5. Control

Cricket 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 calculated value. |
| `if `*`condition`*` [`*`body`*`]` | If *condition* is true, executes *body*. Note: a condition expression that evaluates to zero is considered “false”; all non-zero expressions are “true”. |
| `ifelse `*`condition`*<br>`[`*`body-1`*`] [`*`body-2`*`]` | 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. |

## 6. Multitasking

Blue Dot Cricket Logo contains a `when` primitive that allows for simple multitasking:

| | |
|---|---|
| `when`<br>`[`*`condition`*`] [`*`body`*`]` | Launches a parallel process that repeatedly checks *condition* and executes *body* whenever *condition* changes from false to |

| | |
|---|---|
| | true. The `when` rule is "edge-triggered" and remains in effect until it is turned off with the `whenoff` primitive. Only one `when` rule can be in effect at a time; if a new `when` rule is executed by the program, this new rule replaces the previous rule. |
| `whenoff` | Turns off any existing `when` 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
     resett
     when [timer > 1000] [beep resett]
     loop [a, onfor 1 rd]
end
```

## 7. Numbers

The "Blue Dot" version of Cricket Logo uses 16-bit integers between -32768 and + 32767

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

| | |
|---|---|
| `+` | Infix addition. |
| `-` | Infix subtraction. |
| `*` | Infix multiplication |
| `/` | Infix division. |
| `%` | Infix modulus (remainder after integer division). |
| `and` | Infix logical "and" operation (bitwise and). |
| `or` | Infix logical or operation (bitwise or). |
| `not` | Prefix bitwise not operation. |
| `random` | Reports pseudo-random number from 0 to 32767. |

## 8. Global Variables

Global variables are created using the global [*variable-list*] directive at the beginning of the procedures window. E.g.,

```
global [foo bar]
```

creates two globals, named `foo` and `bar`. Additionally, two global-setting primitives are created: `setfoo` and `setbar`. Thus, after the global directive is interpreted, one can say

```
setfoo 3
```

to set the value of foo to 3, and

```
setfoo foo + 1
```

to increment the value of `foo`.

## 9. Global Arrays

Global arrays are created in the Blue Dot version of Cricket Logo using the

`array [`*`array1-name, array1-length, array2-name, array2-length, etc.`*`]`

directive at the beginning of the procedures window. *E.g.*,

```
array [foo 50 bar 25]
```

creates two arrays, one named `foo`, which can hold 50 numbers and another named `bar`, which can hold 25 numbers. Elements in the array are set and read using the `aset` and `aget` primitives:

| | |
|---|---|
| `aset` *`array-name item-number value`* | sets the *item-number*th element of *array-name* to *value* |
| `aget` *`array-name item-number`* | reports the *item-number*th element of *array-name* |

Thus, for example

```
aset foo 31 1000
```

sets the 31st element of `foo` to have a value of 1000

while

```
send aget foo 31
```

causes the value of the 31st element of `foo` to be transmitted via infrared. There is no error-checking to prevent arrays from overrunning their boundaries.

## 10. 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.

## 11. Data Recording and Playback

In addition to the user defined arrays mentioned above there is a single global array for storing data which holds 1024 one-byte numbers. There is no error checking to prevent overrunning the data buffer. The following primitives are available for data recording and playback:

| | |
|---|---|
| `setdp` *`number`* | Sets the value of the data pointer. |
| `record` *`value`* | Records value in the data buffer and advances the data pointer. |
| `recall` *`value`* | Reports the value of the current data point and advances the data pointer. |
| `erase` | Sets the value of all 1024 elements of the data array to zero and then sets the data pointer to zero. Because the process of recording data is relatively slow (about 20 milliseconds per data point) it takes about 20 seconds for the erase command to be executed |

For example the procedure `take-data` can be used to store data recorded by a sensor once every second:

```
to take-data
     erase beep
     repeat 1024 [record sensora wait 10]
end
```

The data can be "replayed" using the following `send-data` procedure:

```
to send-data
     setdp 0
     repeat 1024 [send recall wait 5]
end
```

This causes the data to appear in the monitor box on the Cricket Logo screen on the desktop, updating twice a second. The Cricket Logo desktop also contains built-in graphing capabilities for rapidly uploading, graphing, and analyzing data.

## 12. Recursion

Cricket 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
```

### 13. Infrared Communication

Crickets can send infrared signals to each other using the **send** primitive, and receive them using the **ir** primitive. The **newir?** primitive can be used to check if a new infrared signal has beenreceived since the last time **newir?** primitive was used

| | |
|---|---|
| **send** *value* | transmits *value* via infrared. |
| **ir** | reports the byte most recently received by the infrared detector. Note that the Blue Dot crickets do not clear the infrared buffer. Thus the ir primitive reports the most recent byte received. |

**newir?** reports “true” if a new byte has been received by the infrared detector since last time **newir?** was used, and “false” if not. It does not effect the contents of the infrared buffer. For example, consider the following use of the **newir?** primitive:

```
to thing1-or-thing2
      waituntil [newir?]        ;checks for new infrared byte
      if ir = 1 [thing1]
      if ir = 2 [thing2]
end

to thing1
   . . .
end

to thing2
   . . .
end
```

In this example nothing happens until a new infrared byte is received.

There are cases when you may want to use an alternate form of the *thing1-or-thing2* procedure:

```
to thing1-or-thing2
if newir?                 ;checks for new infrared byte
      [if ir = 1 [thing1]
      if ir = 2 [thing2]
      ]
end
```

In this case we do not wish to hold everything up until a new infrared byte is received; we only want thing1 or thing2 to happen if a new infrared byte is received.

Infrared codes in the range 128 to129 are interpreted to launch remote-start menu items 1 or 2 on the cricket logo screen. These codes can be generated by pressing buttons #1 or #2 on the interface respectively. Household TV/VCR remotes may be used to cause the cricket to launch its two remote-start lines. Use a sony remote, or a universal remote set to talk to a Sony TV, and use the keys numbered 1 and 2.

Infrared codes in the range 130 to 140 are used by the underlying cricket operating system as escape codes for infrared program download. Therefore please restrict general purpose user broadcast of ir codes to the ranges of 1 to 127 or 141 and above.

Received infrared values issued with the **send** primitive are displayed on the cricket logo screen in the small text box next to the download button.

## 14. Other Details

If a Cricket is not running a program (which is indicated by the green run light being off), infrared codes from 128 to 129 are interpreted to launch remote-start menu items 1 or 2 on the Cricket Logo screen. These codes can be generated by pressing buttons #1 or #2 on the interface respectively. Household TV/VCR remotes may also be used to cause the Cricket to launch its two remote-start lines. Use a Sony remote, or a universal remote set to talk to a Sony TV, and use the keys numbered 1 and 2.

Infrared codes in the range of 130 and higher are used by the underlying Cricket operating system as escape codes for infrared program download. Therefore please restrict general purpose user broadcast of IR codes to the range of 1 to 127.

Received infrared values issued with the send primitive are displayed on the Cricket Logo screen in the small text box next to the download button.

### *Caveats:*

The maximum size of a Cricket Logo program is 768 bytes. (This number becomes smaller if arrays are used. Each array element takes up two bytes of memory. If the record primitive is not used, programs as long as 1792 bytes are possible.) In addition, a maximum of 16 different global variables may be used.

When a program is downloaded, its size is displayed in the “status box” near the bottom of the Cricket Logo procedures window.

## 15. Two Sample Programs

### *Dancing Crickets*

Here's a simple program written by two 10 year old boys who had seen the "dancing Crickets" and wanted to build their own (single Cricket) version:

```
to dance
   cha-cha-cha
   go-round
   shake-it
end

to cha-cha-cha
   repeat 4 [back-and-forth]
   ab, off
end

to back-and-forth
   ab, thisway onfor 3
   beep
   ab, thatway onfor 3
   beep
end

to go-round
```

```
    a, on thisway
    b, on thatway
    beep wait 1 beep wait 1 beep
    wait 60
    ab, off
end

to shake-it
    a, thisway
    b, thatway
    ab,
    repeat 10 [beep onfor 1 beep rd onfor 1 rd]
end
```

Note that these kids made their program easier to follow by nesting procedures inside of other procedures. For example, the procedure dance calls the procedure `cha-cha-cha,` which in turn calls `back-and-forth`.

### *The Wandering LEGObug*

The LEGObug is a creature with two motors connected to its two rear wheels. It also has two touch sensors connected to two "whiskers" positioned on either sides of its head and two light sensors that serve as "eyes." Detailed plans for building the LEGObug are available at the following URL:

`http://lc s.www.media.mit.edu/people/fredm/projects/legobug/`

The procedure `seek` shown below causes the creature to be attracted to bright light. It assumes that the light sensors are plugged into the Cricket's sensor-ports. The light sensors have the property that the greater the amount of light that hits them, the smaller the sensor value that is produced. (In typical indoor lighting the light sensors might give readings in the 15 - 30 range, if you shine a flashlight on them, they will produce a reading in the 1 - 5 range. It takes almost complete darkness to produce a reading of 255.)

```
to seek
loop [
     ifelse (sensora < 10) or (sensorb < 10)
     [go-forward]
     [stop-motors]
    ]
end

;the motors are each hooked up so that the "thisway" ;direction
causes them to drive forward

to go-forward
    ab, on thisway
end

to stop-motors
    ab, off
end
```

As an exercise you might try making creatures that run away from the dark, or ones that turn toward a bright light.

The procedure `wander` shown below causes LEGObug to drive straight until a whisker bumps into an obstacle. (It assumes that the touch sensors are plugged into the two sensor-ports.) In an attempt to avoid the obstacle, it the creature backs up a bit, turns a small (random) amount and continues to drive forward.

```
to wander
   go-forward
   waituntil [touch-left? or touch-right?]
   ifelse touch-left?
   [back-up turn-right]
   [back-up turn-left]
    wander
end

to go-forward
   ab, on thisway
end

;touch-left reports "true" if the sensor
;plugged into sensor-port "a" is pressed

to touch-left?
    output switcha
end

;touch-left reports "true" if the sensor
;plugged into sensor-port "a" is pressed

to touch-right?
   output switchb
end

;turns right for a random amount of time between 0 and 5
;seconds.
;the primitive random reports a random number between 0 and 255

to turn-right
   b, off 5
   a, thisway onfor (random / 5)
end

to turn-left
   a, off
   b, thisway onfor (random / 5)
end

to back-up
ab, thatway onfor 20
end
```

For more information about Crickets visit the Cricket Home Page on the World Wide Web at:

`http://lcs.www.media.mit.edu/people/fredm/projects/cricket/`