



The Playful Invention Company



PicoCricket Reference Guide

Version 1.0

Credits



Design Team

Mitchel Resnick
Brian Silverman
Paula Bontá
Robbie Berg
Natalie Rusk



Industrial Design: Smart Design



Graphic Design: Big Blue Dot

We would like to give special thanks to:



PICO staff members Chad Burt and Mike Durcak for their invaluable ideas and enthusiasm.



Diana Gee Silverman for the on-line help, Danny Lutz for the sounds, and Lawrence Shubert for guiding PICO through the intricate aspects of compliance testing and manufacturability.



staff from the Playful Invention and Exploration (PIE) project for developing new Cricket activities, particularly Keith Braafladt, Stephanie Hunt, Chip Lindsey, Hideki Mori, Kristen Murray, Mike Petrich, Margaret Pezalla-Granlund, Natalie Rusk, Michael Smith-Welch, Karen Wilkinson, and Diane Willow.



MIT researchers and students who contributed to Cricket R&D, particularly Andy Begel, Robbie Berg, Rahul Bhargava, Rick Borovoy, Fred Martin, Bakhtiar Mikhak, Mitchel Resnick, and Brian Silverman.



the LEGO Company for its continuing support, particularly Kjeld Kirk Kristiansen, Lisbeth Valther Pallesen, Jens Maibom, and Erik Hansen.

Contents

Credits	2
Introduction	4
PicoBlocks	8
Blocks Summary	12
Blocks - Detailed by Category	14
Melody and Rhythm Editors	30
The PicoBlocks Text Language	35
Error Messages	48

Introduction

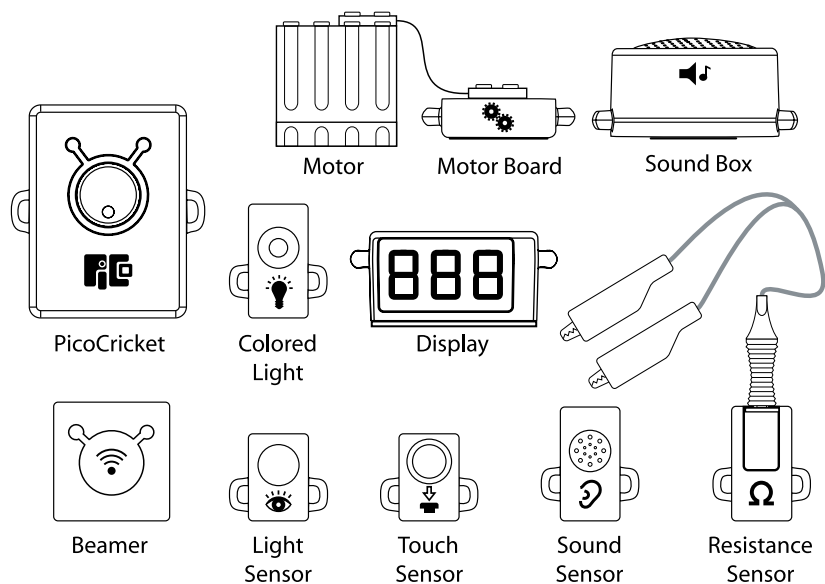
Welcome! This Reference Guide provides an overview of the PicoCricket family of parts and the PicoBlocks software used to program the PicoCricket.

If you are just getting started with the PicoCricket and PicoBlocks, please go through the Setup Guide and Getting Started booklet first, to get an overall introduction. If you want more detailed information, then come back to this Reference Guide.

We are continually updating the Reference Guide. You can download the latest version from www.picocricket.com/support

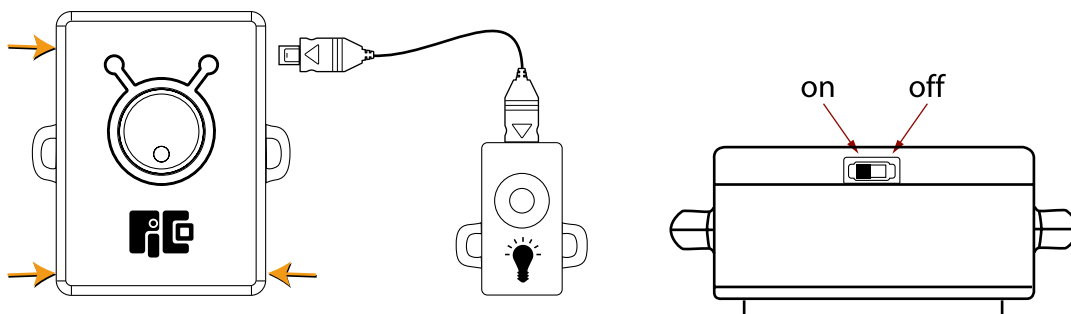
PicoCricket Family

The PicoCricket family includes the following parts:

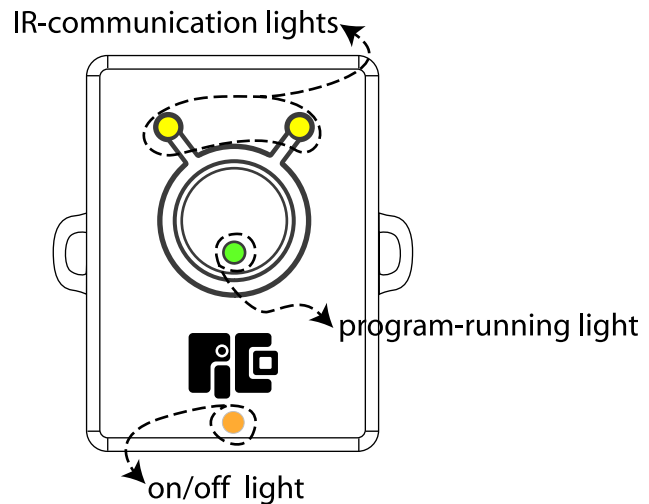


PicoCricket

The PicoCricket is a tiny computer that can control things in the world. The Cricket has four ports where you can plug in other devices, such as sensors, motors, and lights. You can plug any of these devices into any of the Crickets ports.



The Cricket has three types of indicator lights:

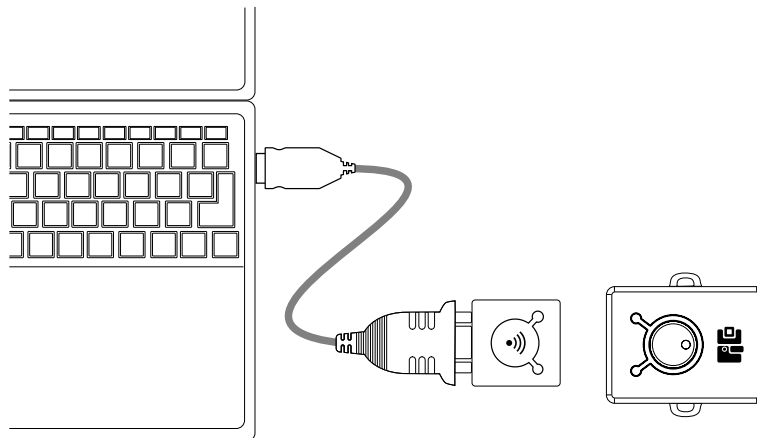


- The **on-off indicator light** turns on (orange) when you turn on the Cricket. If the batteries are low, the light flashes rapidly when you turn on the Cricket.
- The **program-running indicator light** turns on (green) when the Cricket is running a program, and it changes color to amber when the Cricket is running two programs at the same time.
- The **IR-communication indicator lights** flash (yellow) when the Cricket is communicating through its infrared transmitter.

If you push the Cricket button while the Cricket is running a program, the program will stop. If you push the button when no program is running, the Cricket will re-run the last program that it ran.

Even when you turn the Cricket off, it remembers the last program that it ran. So if you run a program, turn off the Cricket, turn it back on later, and push the button, the Cricket will run the program again.

Beamer



The Beamer transmits information (via infrared signals) from your computer to your Cricket. When you write a program using PicoBlocks software on your computer, you send the program to the Cricket via the Beamer.



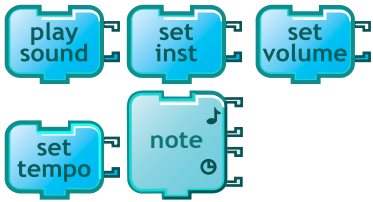

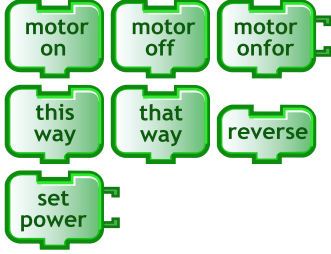



The PicoCricket Kit includes a USB-serial cable for connecting the Beamer to a USB port on your computer. Alternatively, you can connect the Beamer to a serial port, using a standard serial cable (not included).

When you are sending information to the Cricket, make sure that the antennas on the Beamer are facing towards the antennas on the Cricket. While the Beamer and Cricket are communicating, the indicator lights in the antennas will flash yellow (on both the Beamer and the Cricket).

The Beamer has a range of about a 1 meter. Note that a Beamer can communicate with one Cricket at a time. If you have multiple Crickets, make sure that only one of them is in range of the Beamer at any time.











Actuators

Actuators are devices that make things happen in the world. The PicoCricket Kit includes four types of actuators: colored lights, sound box, motor (with motor board), and numeric display.

Name	Description	Programming Blocks	Icon
Light	You can set the color of the light across the entire rainbow, from red to violet. You can also set the brightness of the light.		
Sound Box	The sound box is a small synthesizer that can play notes, melodies, sound effects, and rhythms. PicoBlocks software includes special editors for creating melodies and rhythms. You can set and change which instrument will play notes and melodies.		
Motor and Motor Board	The motor board connects to the Cricket, and the LEGO motor connects to the motor board. You can turn the motor on and off, and set its power and direction.		
Display	The three-digit display can show any number between 0 and 999. It is especially useful for displaying sensor values.		

Sensors

Sensors allow your Cricket to respond to sounds, lights, or other changes in its environment.

Name	Description	Programming Blocks	Icon
Light Sensor	Reports the brightness of light. For example, you can use the light sensor to detect when someone casts a shadow with their hand – or when the sun shines through the window.		
Sound Sensor	Reports the loudness (or volume) of sound. For example, you can use the sound sensor to detect when someone claps their hands – or when someone sings into the sensor.	 	
Touch Sensor	Reports if the button is pressed. For example, you can write a program that makes something happen (a motor spinning or a light changing color) when someone presses the touch sensor.		
Resistance Sensor	Reports the resistance in the circuit formed by the alligator-clip cables. If you put different materials (for example, play-dough or a banana) between the alligator clips, the resistance will change. You can also use the resistance sensor to detect when the alligator clips are touching each other (or connected through a conductive material, such as aluminum foil).	 	

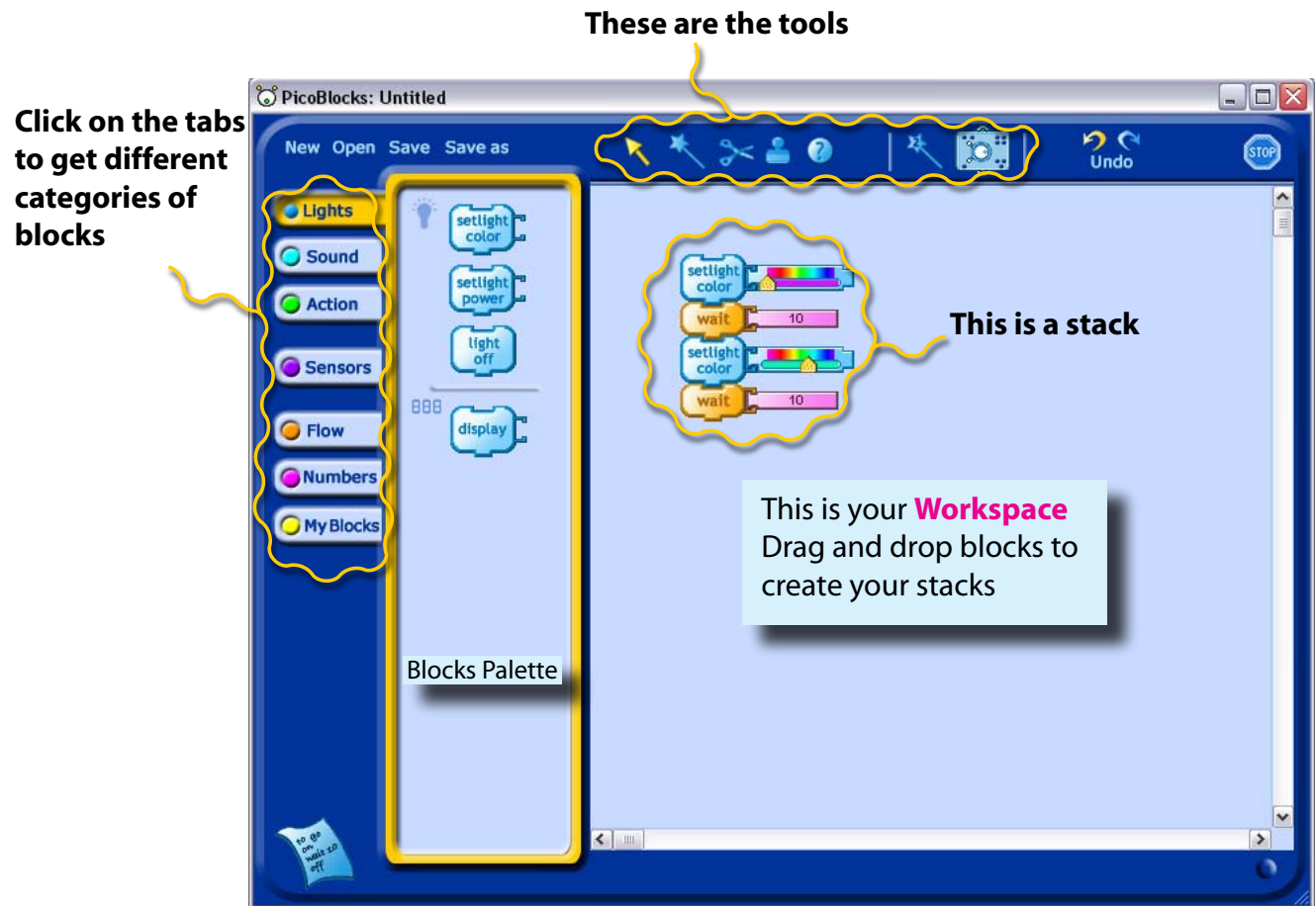
Each of the programming blocks will be described further in the following pages. For examples of how to use the actuators and sensors, [see “Magic Lantern” and other sample project placemats](#).

Cricket Care

1. Remember to turn off the Cricket when you aren't using it.
2. If the orange on-off indicator light flashes, it means that your Cricket is running out of batteries. To change the batteries, follow the instructions in the Setup Guide.
3. Be gentle with the cable connectors. If you push the connectors up and down with too much force, the connectors will break.
4. Keep your Cricket devices away from water and other liquids.
5. If your Cricket is misbehaving, check the Troubleshooting section in our website: www.picocricket.com/troubleshooting

Overview

PicoBlocks is a programming language designed specifically for programming your Cricket. With PicoBlocks, you create programs by snapping graphical blocks together into stacks.




Tools



The tools are used for taking different actions on the blocks, such as copying blocks, deleting blocks, or sending blocks to the Cricket. When you click on a tool, the cursor turns into that tool.

Arrow

Use the Arrow to drag blocks from the Blocks Palette to the Workspace, or within the Workspace. To move an entire stack, drag from the top block. Dragging a block from the middle of a stack will also move any blocks attached beneath the one you are dragging.

You can get rid of blocks by dragging them off of the Workspace, in any direction. If you delete blocks by mistake, click Undo 

Magic Wand

Use the Magic Wand to run a program. When you click on a block or stack with the Magic Wand, it sends the program to the Cricket and tells the Cricket to run the program.



You can use the Magic Wand to click on blocks in the Blocks Palette, the same way you click on blocks in the Workspace. This feature is useful for testing out a block.

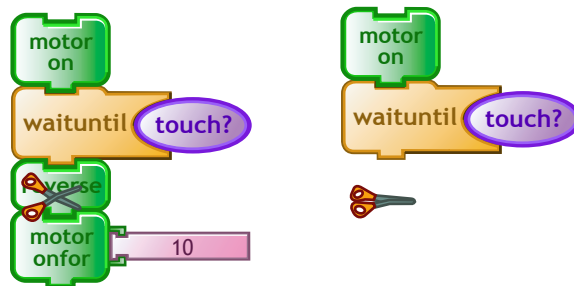
Another way to run a program is by double-clicking with the Arrow, instead of single-clicking with the Magic Wand.

Scissors

Use the Scissors to get rid of blocks. Clicking on a block in a stack will also cut all blocks attached below it.




If you delete blocks by mistake, click Undo .

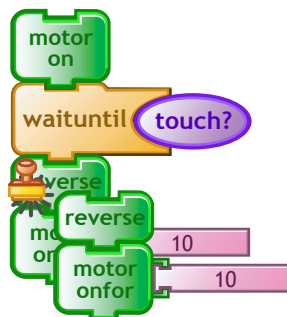
You can also get rid of blocks by dragging them out of the Workspace.



Stamper

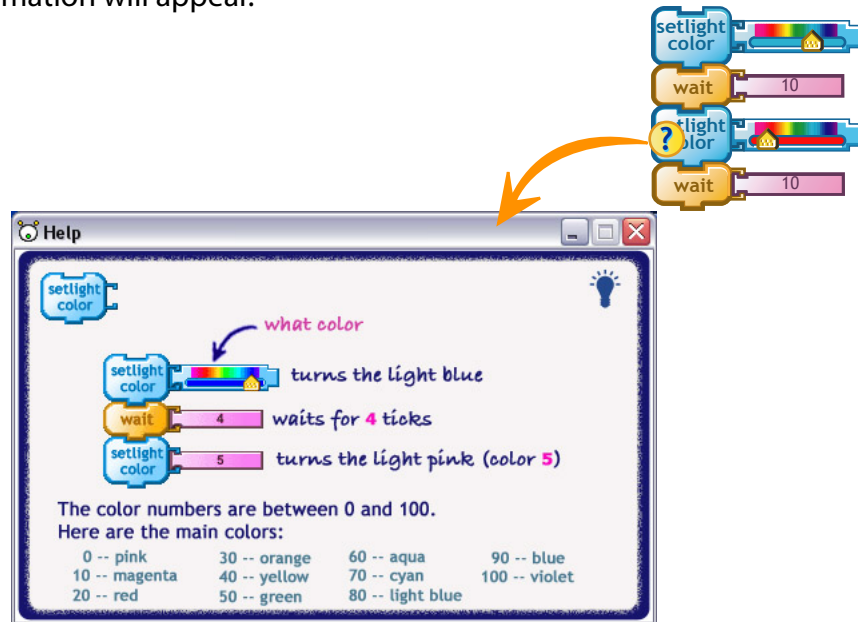
Use the Stamper to copy blocks and stacks. Click once to copy, then move to a new location and click a second time to paste. When you click on a block in a stack with the Stamper, it also copies all blocks attached below it.

You can switch to the Stamper temporarily using Ctrl + , or  + .



Help

Use the Help tool to get more information. If you click on a block with the Help tool, a window with more information will appear.



Second Wand

Use the Second Wand to tell the Cricket to run a second stack at the same time as it is running something else. To run two stacks, click on one with the Magic Wand, then the other with the Second Wand.

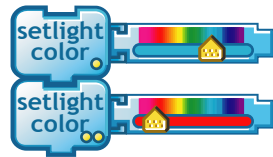
For example, you can use the Second Wand to start background music or display sensor values while another program is running.



Tags

Use the Tags tool when you want to tell two devices of the same type to do different things. For example, if you plug two colored lights into a Cricket, and then send a setlight-color command to the Cricket, the Cricket will change the color of both lights. If you want to turn each light to a different color, you need to use Tags.

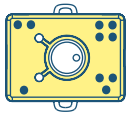
Each of the four ports on the Cricket is labelled with a different number of dots. If you want to send a command to a particular port, then you need to “tag” the block with the appropriate number of dots. For example, if you plug colored lights into ports ● and ●●, then you can set the color of each light individually with the following blocks:



To tag a block, click on the appropriate dot in the Tags tool, then click on the block that you want to tag.



To remove a tag from a block, click in the center of the Tags tool and then click on the block.



Stop


Click on the Stop sign to stop all programs running on the Cricket. Make sure that the Cricket is “in sight” of the Beamer. Alternatively, you can stop programs on the Cricket by pressing the white button on the Cricket. Note that stopping programs on the Cricket does not turn off lights attached to the Cricket. If you want to turn off the lights, slide the on-off switch into the off position.


Blocks Summary


This list shows all of the basic blocks in PicoBlocks.


 sets the light's color

 sets the light's power


 turns the light off



 displays a value on the display

 makes the Cricket chirp


 plays a note (pitch and duration)


 plays a sound



 sets an instrument for the next note or melody


  sets the tempo or volume for the next note, melody or rhythm


  turns motor on or off


 turns the motor on for a certain period of time


 reverses the motor's direction


  sets the motor's direction


 sets the motor's power

 reports true if the touch sensor is pressed


 reports a value from the light sensor


 reports true if the sound sensor detects a loud sound


 reports a value from the sound sensor


 reports true if the resistance sensor detects a resistance of 0

 reports a value from the resistance sensor

 beams a value to another Cricket


 reports a value beamed by infrared from another Cricket

 reports true if another Cricket has beamed a new value by infrared


 reports the Cricket's timer value in 100ths of a second

 sets the Cricket's timer to 0

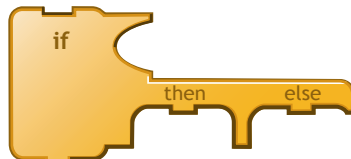
 waits a number of ticks (where 10 ticks = 1 second)


 waits until the condition is true

 repeats a block or stack forever

 repeats a block or stack a certain number of times

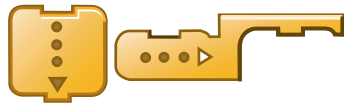
 runs a block or stack if the condition is true

 if condition is true, run one stack; if not, run the other stack

 stops everything

 stops this stack

Blocks Summary - Continued



creates space between blocks,
so that they don't overlap one another

specifies a number (click to highlight, and then type in a number)



computes arithmetic operations



picks a random value between the two numbers



makes a comparison between two numbers
and reports true or false



reports true if both conditions are true



reports true if either condition is true



reports the opposite of a condition



sits on top of stack, and allows you to type in name for the stack



box1

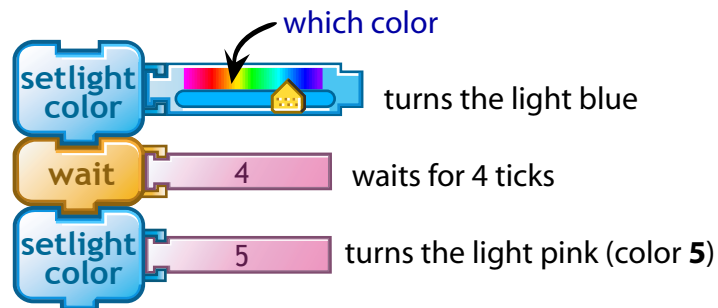
set and report values for variables



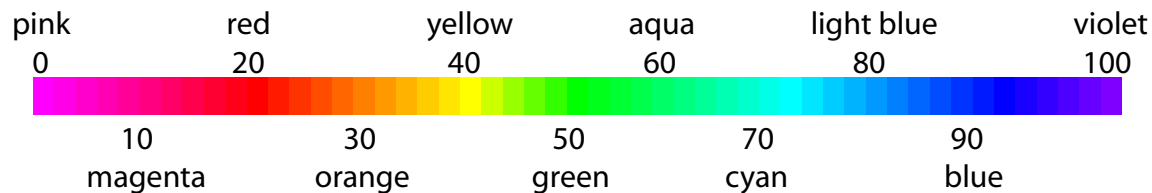
box2

Blocks - Detailed by Category

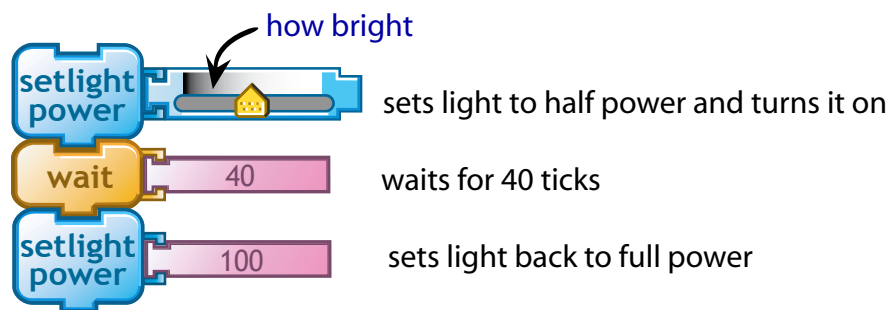
Light Blocks



Sets the LED's color and turns it on. You can set the color by using the color slider, or choosing a number between 0 and 100. Colors are numbered as follows:

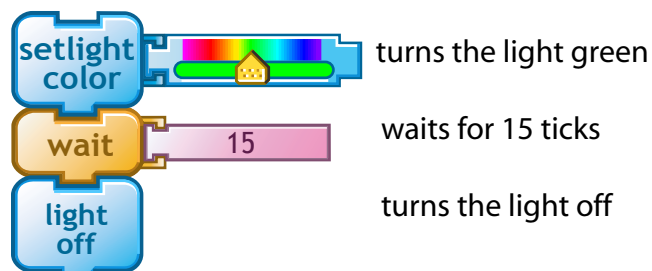


Default value: **setlightcolor 0**

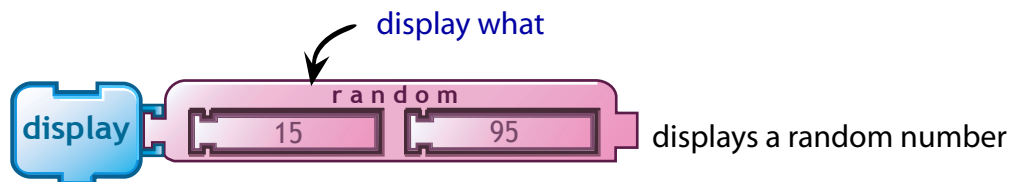


setlightpower can take numbers between 0 (no light) and 100 (full power). You can set the power by using the slider or a number block.

Default value: **setlightpower 100**



lightoff turns the light off.



Displays a value on the display

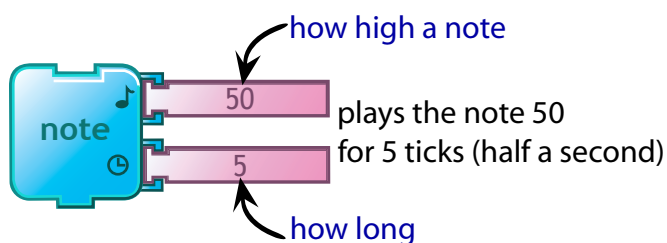
Sound Blocks

The Sound palette starts with six basic blocks (shown below). You can create additional blocks with the Melody and Rhythm editors. See [page 30](#) for information on creating melodies and rhythms.



makes the Cricket chirp once

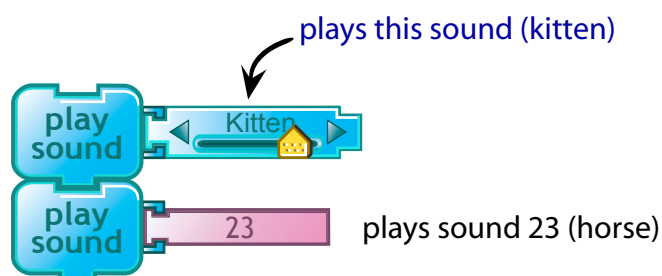
Making the Cricket **chirp** is a good way to test whether your computer is communicating properly with the Cricket.



note can play notes between 0 (lowest) and 100 (highest).

You can play notes with the Cricket speaker or with the Sound Box (but it will sound much better with the Sound Box). For composing melodies, we suggest using the [Melody Editor](#) rather than putting individual note blocks together.

Default value: **note 50 2**



Select a sound with the slider or enter a number between 0 and 100. There are 24 sounds available (see table below).

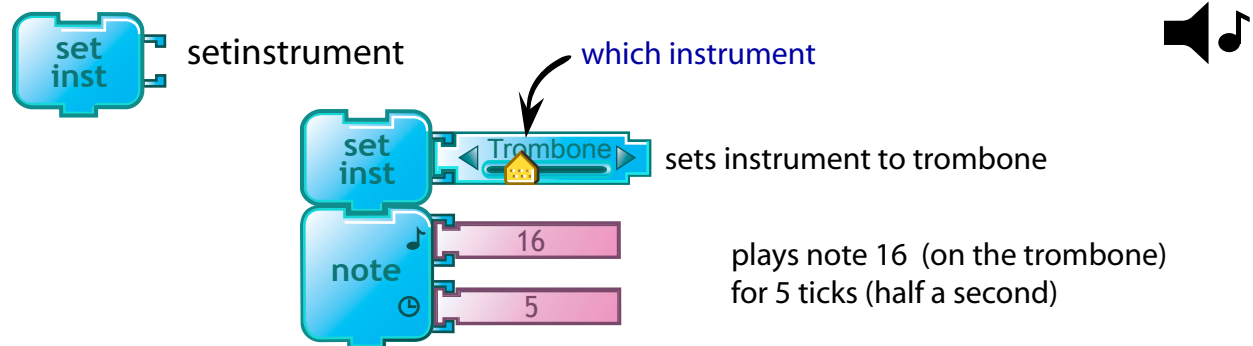
#	Sound
1	Piano
2	Flute
3	Pizzicato
4	Strings
5	Trombone
6	Vibes
7	Agogo
8	Cabasa

#	Sound
9	Claps
10	Cowbell
11	Maracas
12	Slap
13	Timbale
14	Wood Block
15	Guiro
16	Bloops

#	Sound
17	Cricket
18	Fairydust
19	HighQ
20	Dog
21	Rooster
22	Goose
23	Horse
24	Kitten

All these sounds require the Sound Box. Sound 0 is the Cricket chirp. For numbers greater than 24, playsound plays the next sound in the list: 25 plays sound 1, 26 plays sound 2, ... and so on.

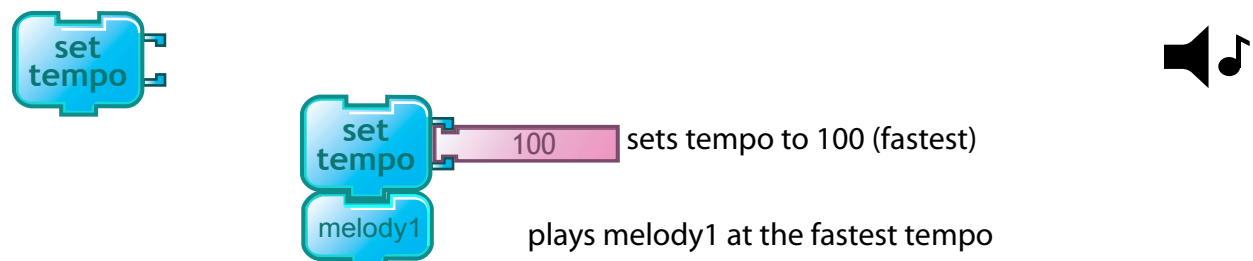
Default value: **playsound 24** (kittten)



Sets the instrument to be used for subsequent notes or melodies. You can select the instrument with the slider or attach a number between 0 and 100. **setinstrument 0** sets the note or melody with the Cricket's beeper.

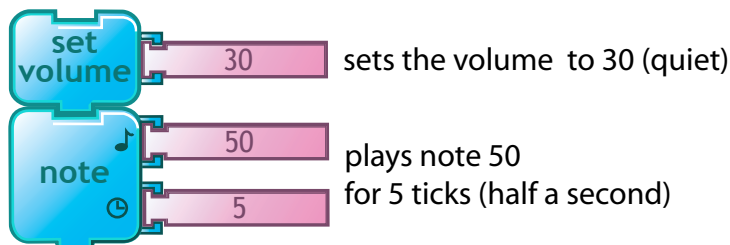
The inputs for **setinstrument** are the same as for **playsound** (shown in table above). You can use any sound as an instrument, but they don't all work well.

Default value: **setinstrument 1** (piano)



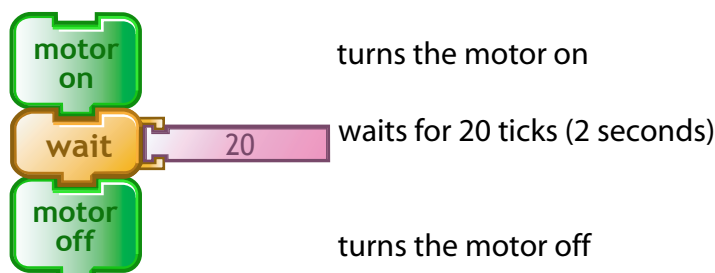
settempo takes values between 0 (slowest) and 100 (fastest). The fastest tempo (100) equals to 240 bpm (beats per minute), while the slowest (0) equals to 60 bpm.

Default value: **settempo 50** (120bpm)

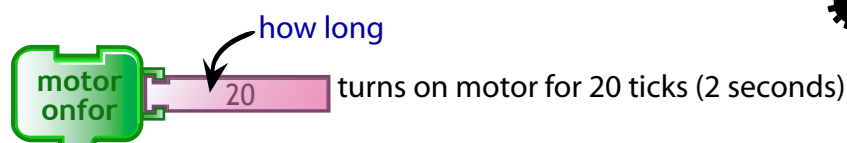


setvolume takes values between 0 (no sound) and 100 (loudest). **Setvolume** only works with the Sound Box. Default value: **setvolume 80**

Action Blocks



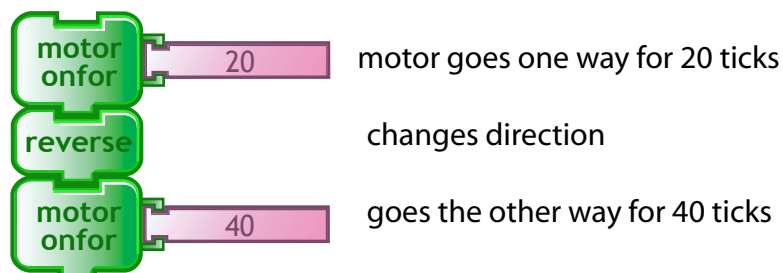
motoroff makes the motor coast to a stop. If you want the motor to stop more abruptly or more gently, try using the **brake** and **coast** commands in the [PicoBlocks Text Language](#).



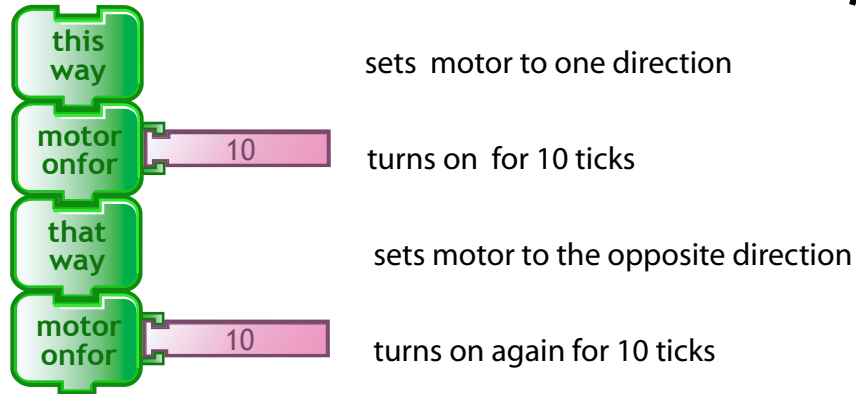
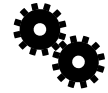
10 ticks = 1 second

Turns the motor on for the specified number of ticks, then turns it off. **motoronfor** does the same thing as a stack with **motoron**, then **wait**, then **motoroff**.

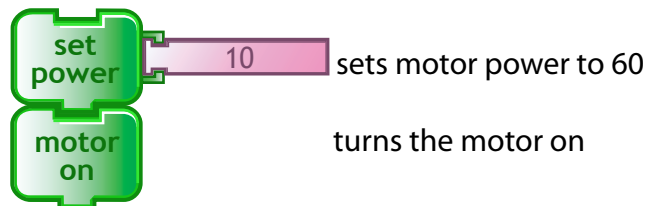
Default value: **motoronfor 10**



reverse causes the motor to change the direction in which it is spinning. **reverse** does not turn the motor on, it only changes the direction.



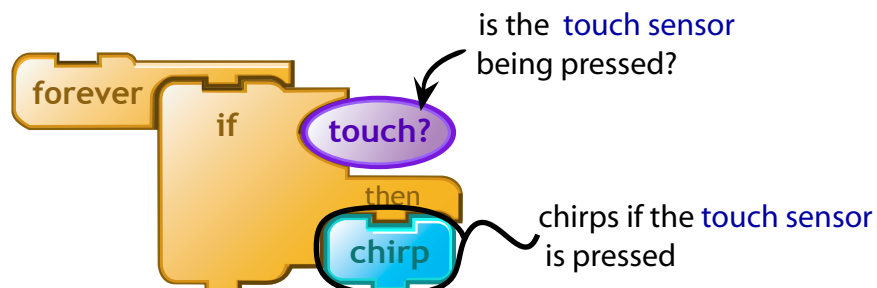
thisway sets the motor in one direction, **thatway** sets it in the other direction. Which direction is which depends on how the motor is plugged in. **thisway** and **thatway** do not turn on the motor, they only set the direction.



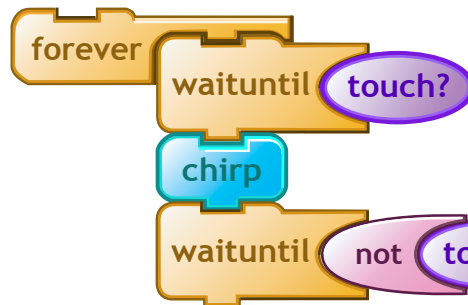
setpower takes values between 0 (no power) and 100 (strongest power). **setpower** does not turn on the motor, it only sets the power.

Default value: **setpower 100**

Sensors Blocks

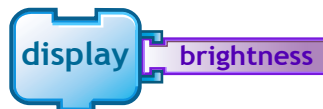


touch? reports true when the touch sensor is being pressed. The example above will make the Cricket chirp continuously while you are pressing the button on the touch sensor. If you want to chirp only once each time you push the button, you should make a stack similar to the one shown below:



waits until the **touch** sensor is released

brightness



displays the number reported by the **light** sensor

brightness reports numbers between 0 (no light) and 100 (lots of light)

loud?

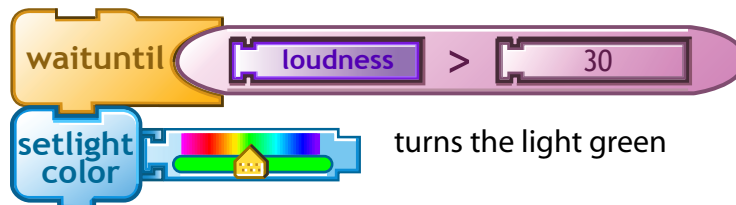


waits until the sound sensor detects a sound

turns the light pink

loud? reports true if the sound sensor detects a sound with loudness greater than 15.

loudness

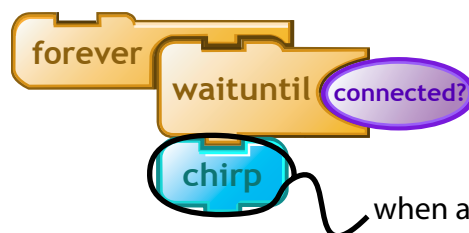


waits until the sound sensor detects a clap

turns the light green

loudness reports numbers between 0 (no sound) and 100 (loud sound).

connected?



waits for resistance sensor to detect a connection

when a connection is sensed, it chirps

connected? reports true if the resistance sensor detects a resistance of less than 50.



displays the number reported by the resistance sensor

resistance reports numbers between 0 (no resistance) and 100 (lots of resistance). If you are getting numbers in a very narrow range, you should adjust the sliders as follows:



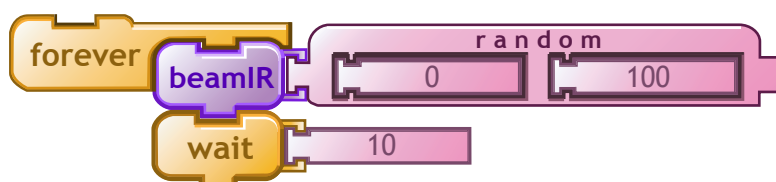
try this if you aren't getting low numbers



try this if you aren't getting high numbers

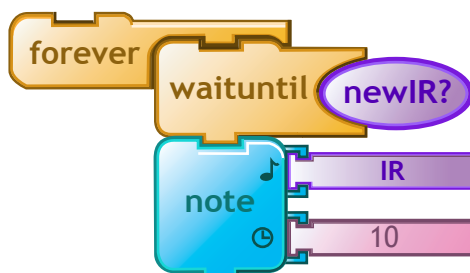


Remember to use the magic wand after moving the sliders



beams a number between 0 and 100

beamIR beams a number to another Cricket via infrared (IR). To use this command, you need two Crickets: one beaming the IR signal, the other receiving the IR signal. See example for **IR** block below. We recommend using numbers in the range 0 to 100. Numbers outside this range may produce unexpected results.

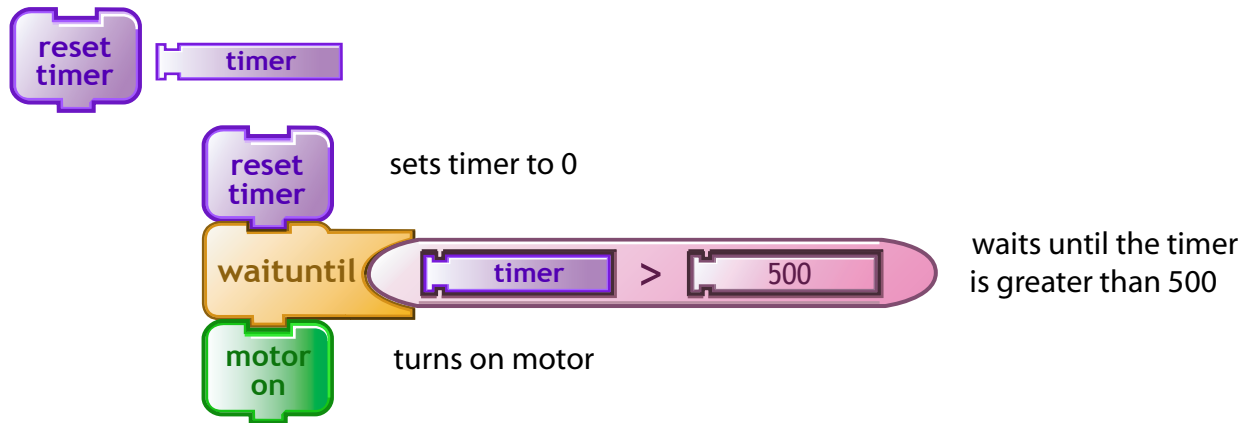


waits for a new IR message from another Cricket

plays a note based on the number received

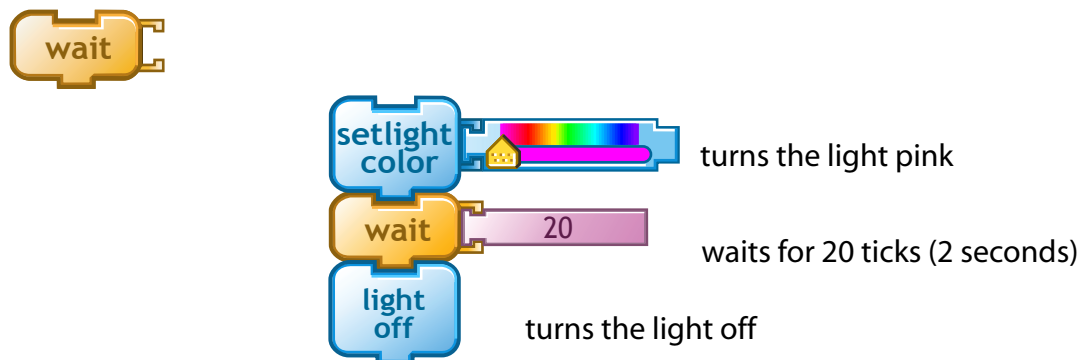
IR stands for Infra Red.

newIR? reports true if a new infrared (IR) signal has been received. **IR** reports the value of the latest infrared signal received.

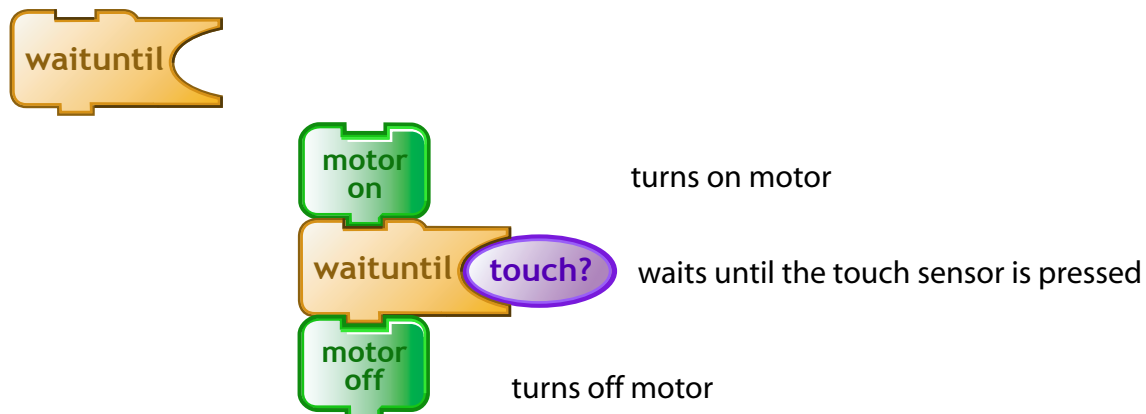


100 ticks of the timer equals 1 second

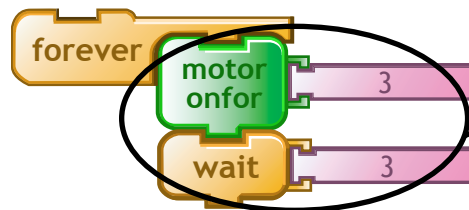
Flow Blocks



10 ticks equals 1 second
Default value: **wait 10**



Waits until the condition is true.

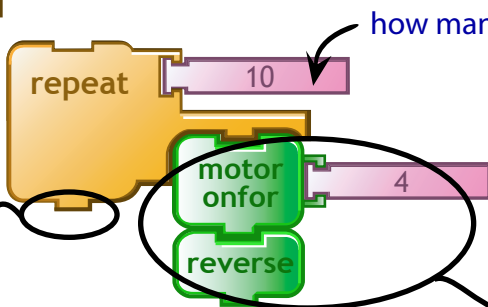


repeat this forever

To stop the forever, push the button on your Cricket or click on 

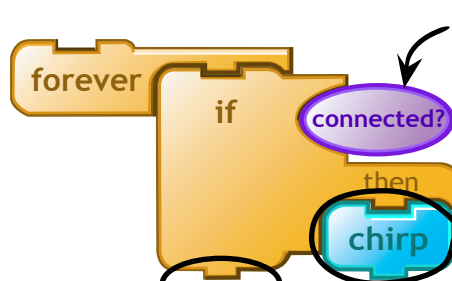
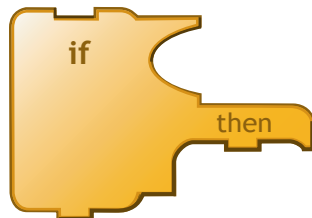


place here
whatever you want
to do after the
repeat finishes



how many times

repeat this 10 times

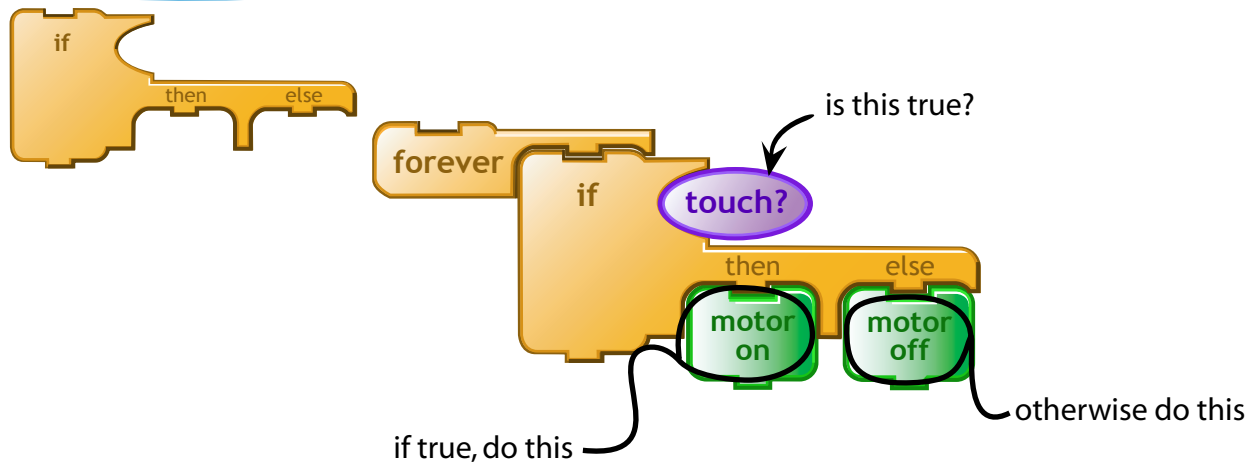


if this is true...

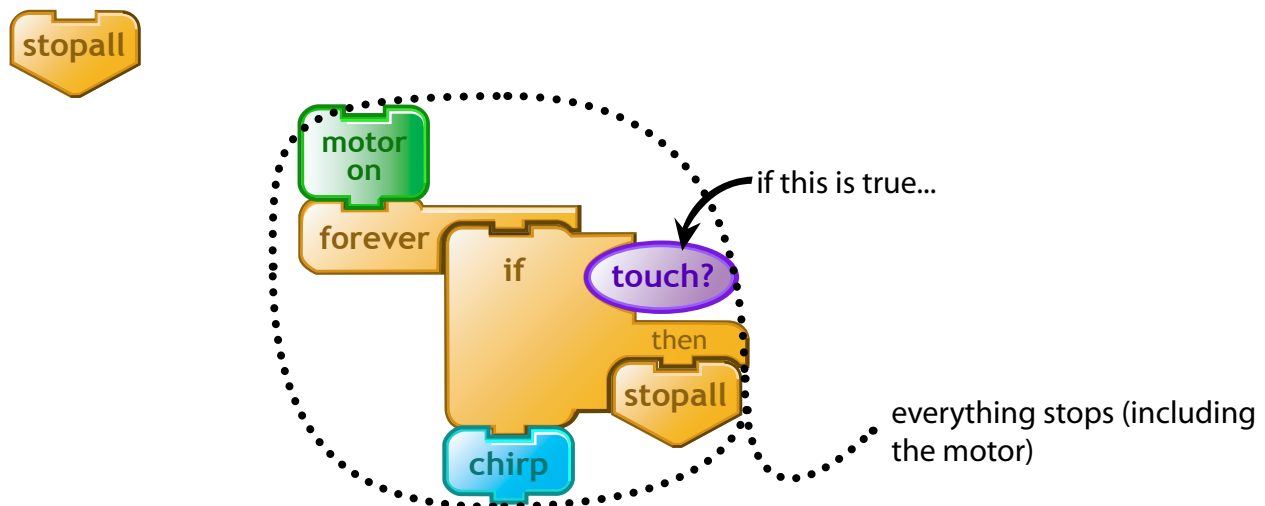
this will happen

place here whatever you want
to do after the **if** finishes

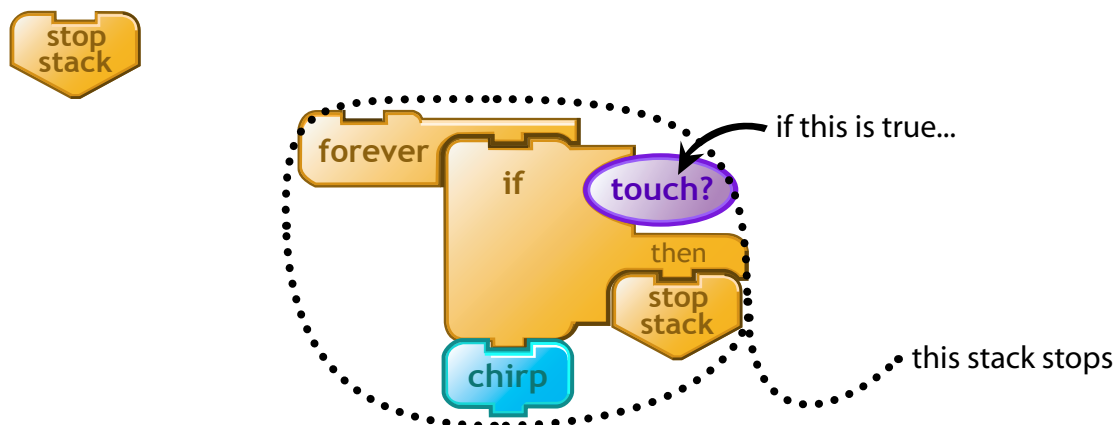
If the condition reports true, then the Cricket will run the blocks below the word "then."



If the condition reports true, then run the blocks under the word “then”; if not, run the blocks under “else”.



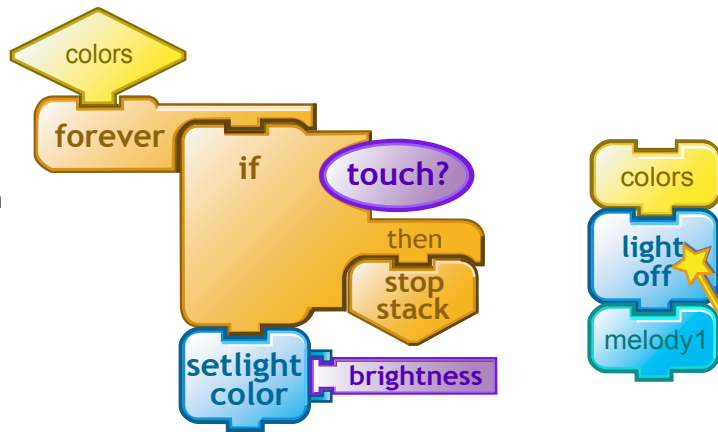
Stops all programs and motors (but leaves lights on). **stopall** is equivalent to pressing the button on the Cricket.



stopstack stops only the current stack. You can use this block to stop one stack while continuing another stack.

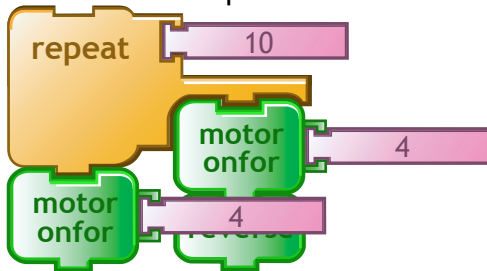
For example:

colors will stop when the touch sensor is pressed, yet the other stack will continue. The light will be turned off and melody1 will be played.

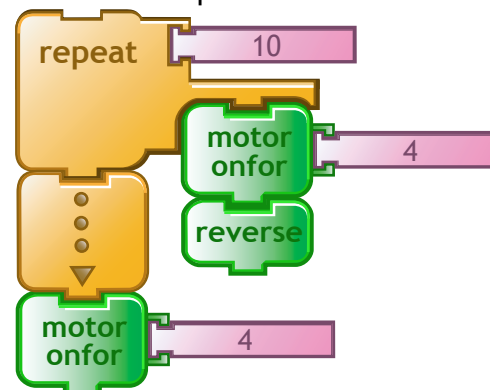


Vertical Spacer

Without Spacers



With Spacers

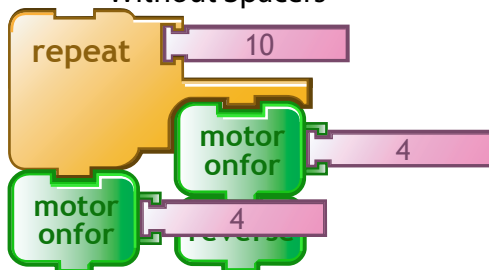


Use spacers to keep the blocks from overlapping. (Adding spacers does not change the way the program runs.)

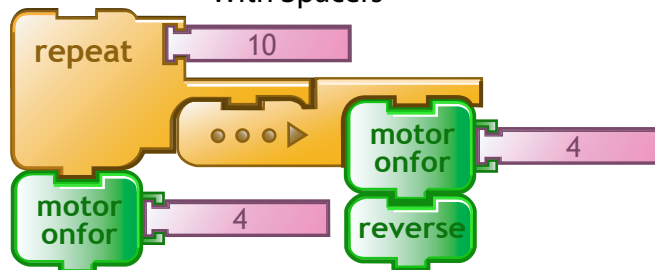


Horizontal Spacer

Without Spacers



With Spacers



Use spacers to keep the blocks from overlapping. (Adding spacers does not change the way the program runs.)

Numbers Blocks





50

sets motor power to 50 (half power)



20

turns on motor for 20 ticks (2 seconds)

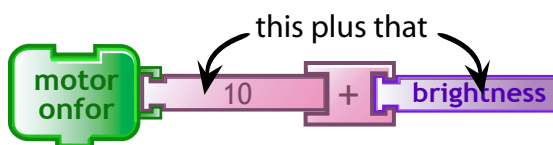
Click on  to type in a number.  accepts negative numbers.

Default Value: **10**

Numbers in PicoBlocks are limited to integers between -32768 and +32767.



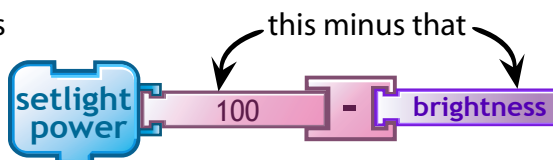
adds two values



turns on motor for 10 plus the value of the brightness sensor (for example, if the brightness sensor is 30, motor turns on for 40 ticks).



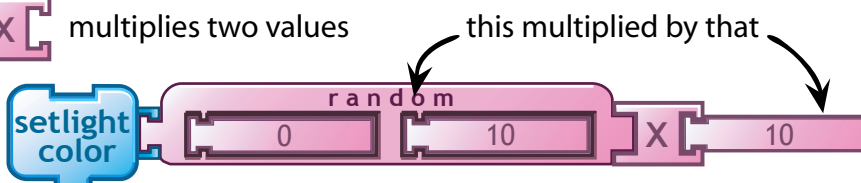
subtracts two values



turns on light with a power of 100 minus the value of the brightness sensor (for example, if the brightness sensor is 60, light power will be 40).



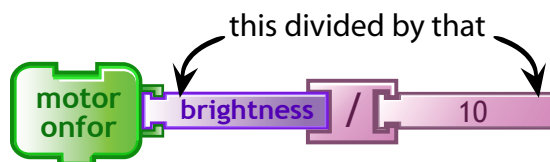
multiplies two values



Sets the light's color to a random number multiplied by 10 (for example, if the random number is 7, sets the color to 70 which is cyan).



divides the first value by the second value



turns on motor for the value of brightness sensor divided by 10 (for example, if the brightness sensor is 50, motor turns on for 5).

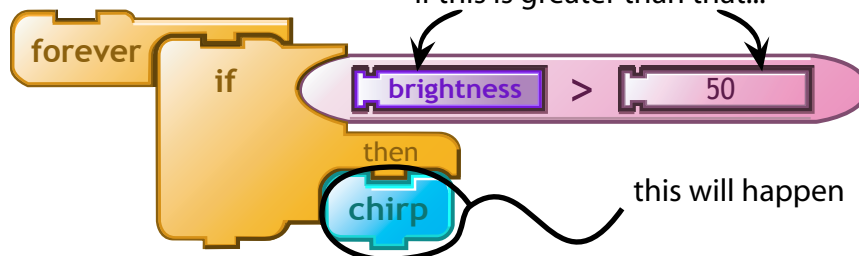


reports a random number
between this and that

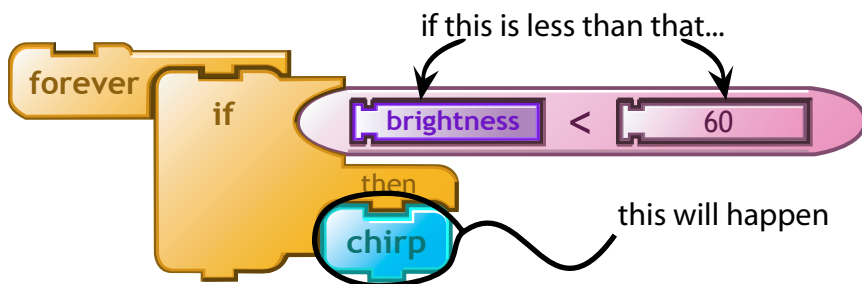


turns on motor for a random number of
ticks (in this case it could be 10, or 50, or
anything in between...)

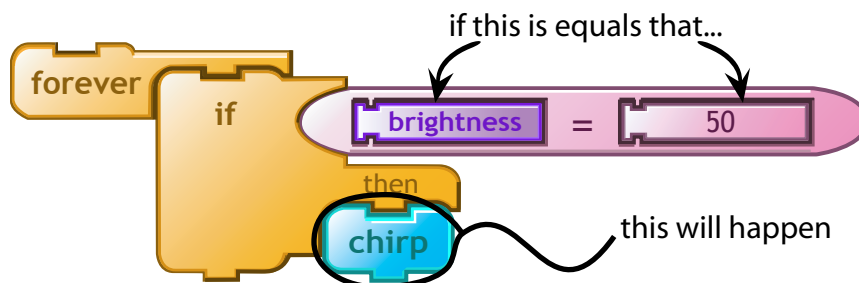
Default value: **random 0 100**



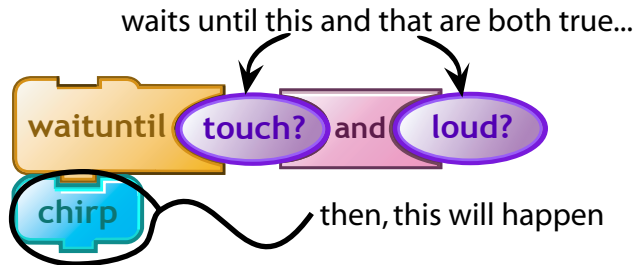
number1 > number2 reports true if number1 is greater than number2.



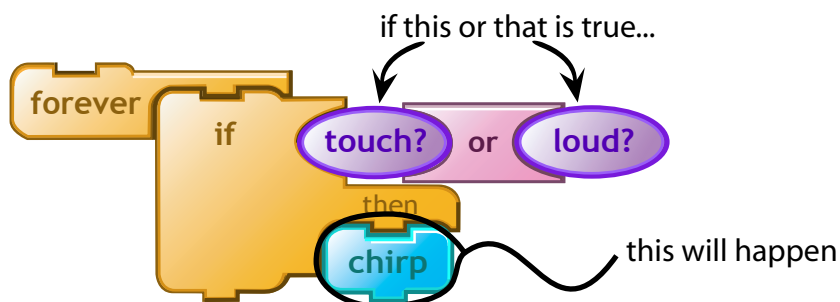
number1 < number2 reports true if number1 is less than number2.



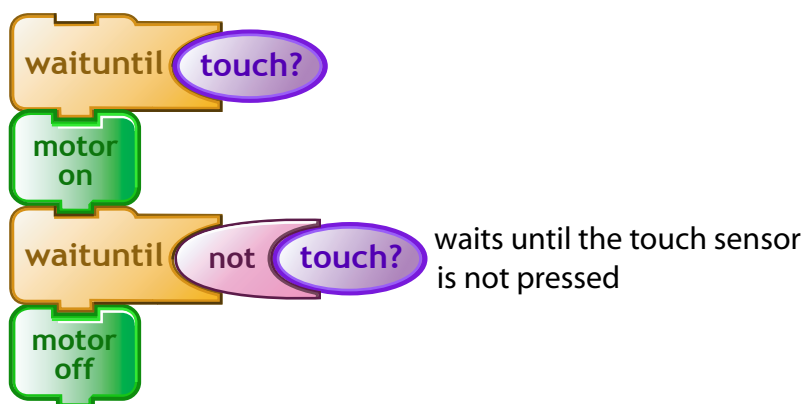
number1 = number2 reports true if the two numbers are equal.



and reports true if the result of the first condition (**touch?**) and the second condition (**loud?**) are both true.



or reports true if the result of the first condition (**touch?**) or the second condition (**loud?**) is true.



not reports true if the condition is false. not reports false if the condition is true.

My Blocks

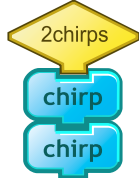


You can use this block to give a name to a stack of blocks – and create a new block that does the same thing as the entire stack. Here's how:

1. attach the block at the top of the stack you want to name



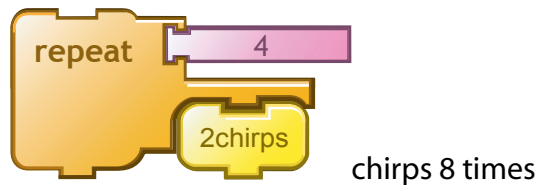
2. click and type a name





3. a new block with that name appears in

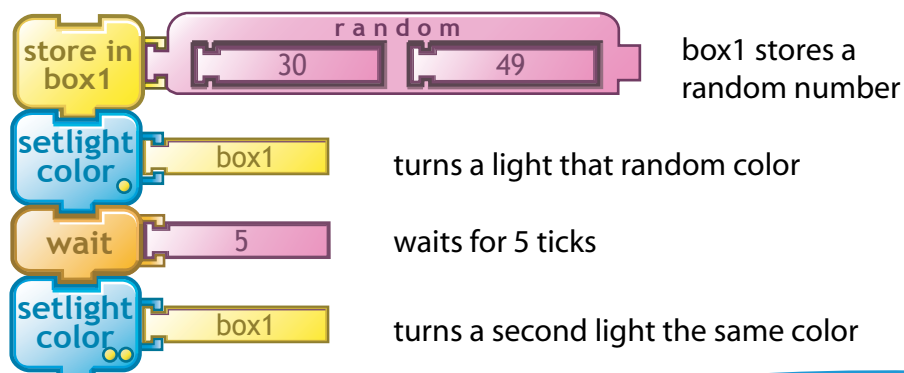


4. Use it in other stacks

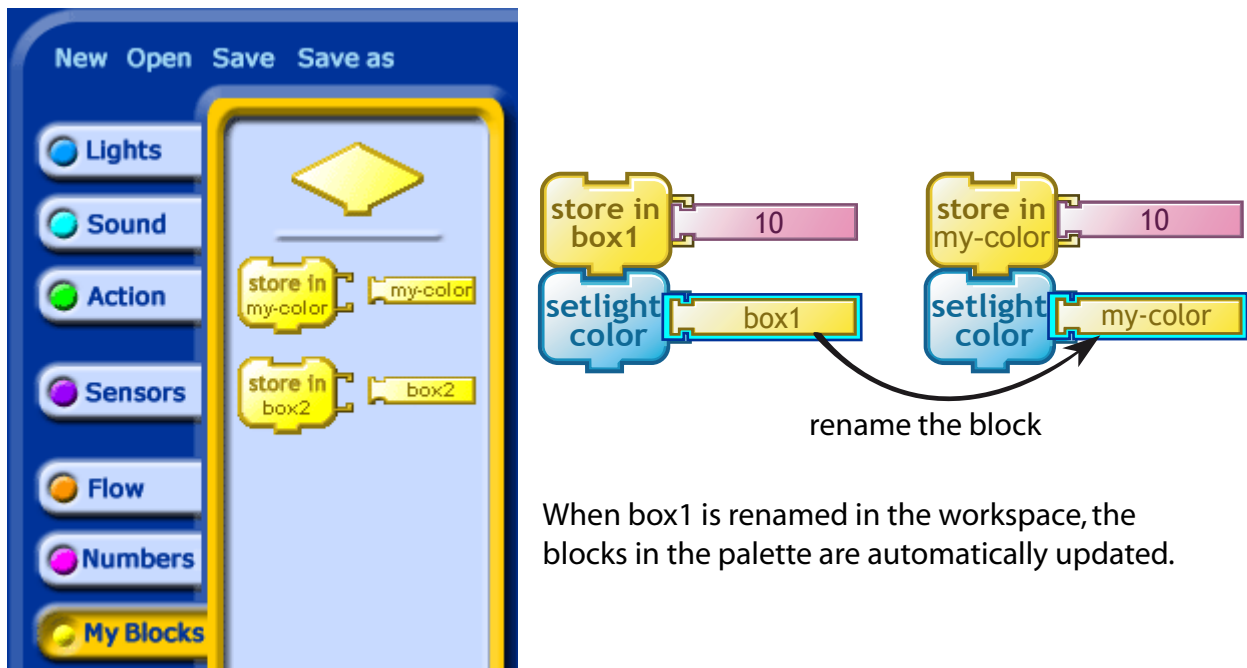


Use these blocks for setting and retrieving the values of variables.

Use  to store a value, and use  to retrieve it.



To change the name of a variable, drag it into the workspace, click on it, and type in the new name.

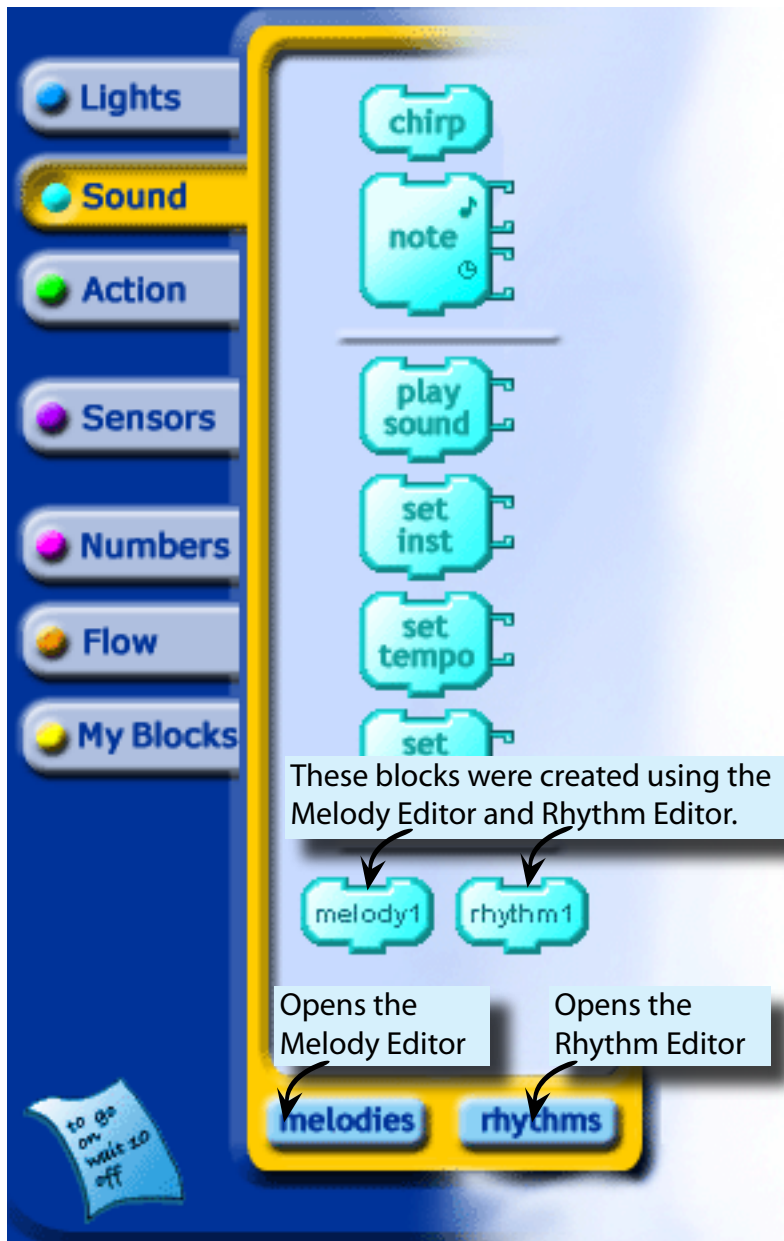


Melody and Rhythm Editors

PicoBlocks has two editors for creating music: a Melody Editor and a Rhythm Editor.

To create a new melody or rhythm (or to edit an existing one), first click on the Sound tab. At the bottom of Sound palette are two buttons, one for opening the Melody Editor, the other for opening the Rhythm Editor.

After you create a new melody or rhythm, it will appear as a new block in the Sound palette. You can drag these music blocks into the workspace, and connect them to stacks, just as you would with any other block in PicoBlocks.



Melody Editor

Click on **melodies** to open the Melody Editor.

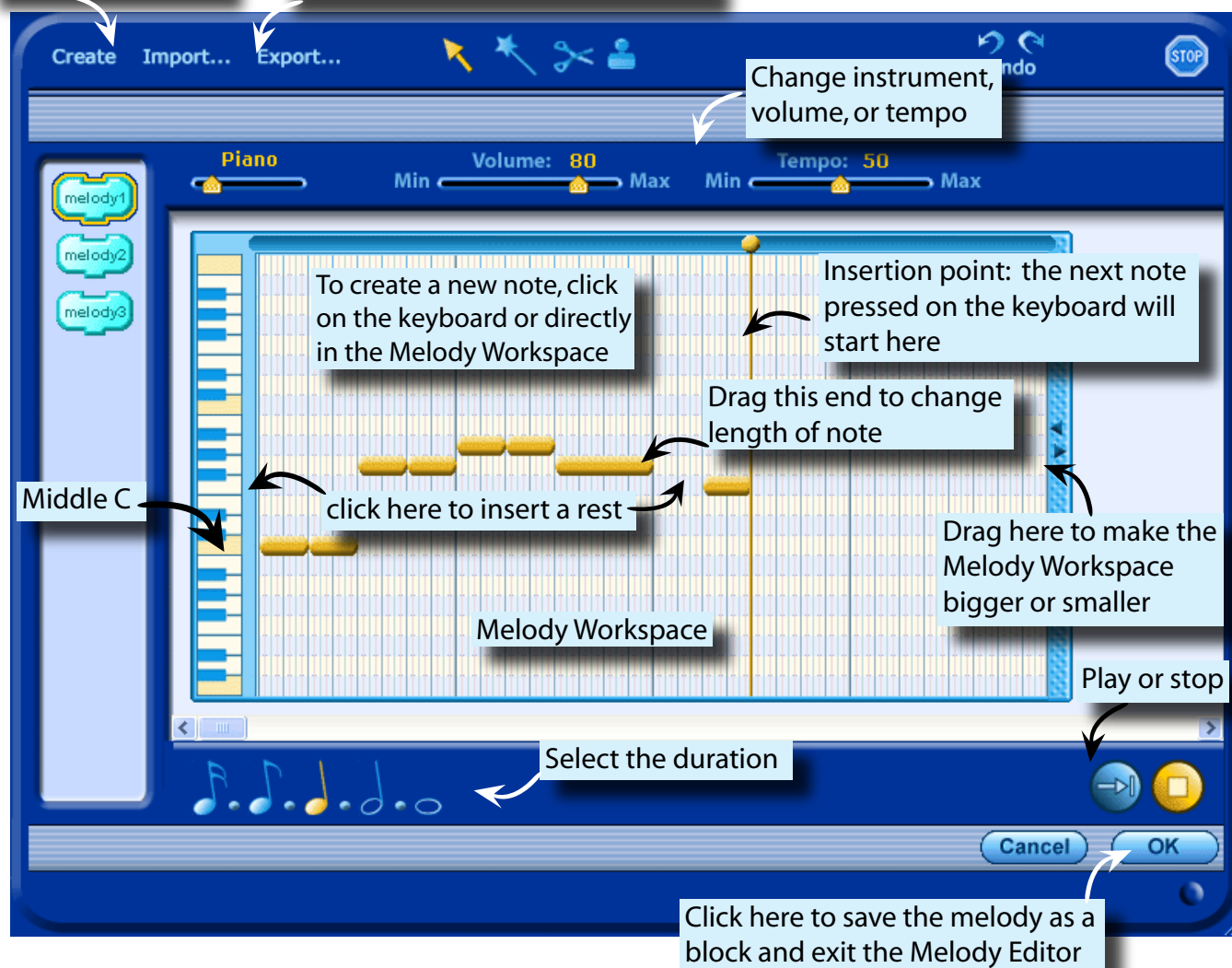
If this is your first melody, you will see a blank Melody Workspace for creating your melody. If you have previously created melodies, you will see one of your existing melodies in the Melody Workspace. You can edit this melody – or select one of your other existing melodies to edit (by clicking on a block in the palette on the left side) – or create a new melody (by clicking on the “Create” button at the top-left).

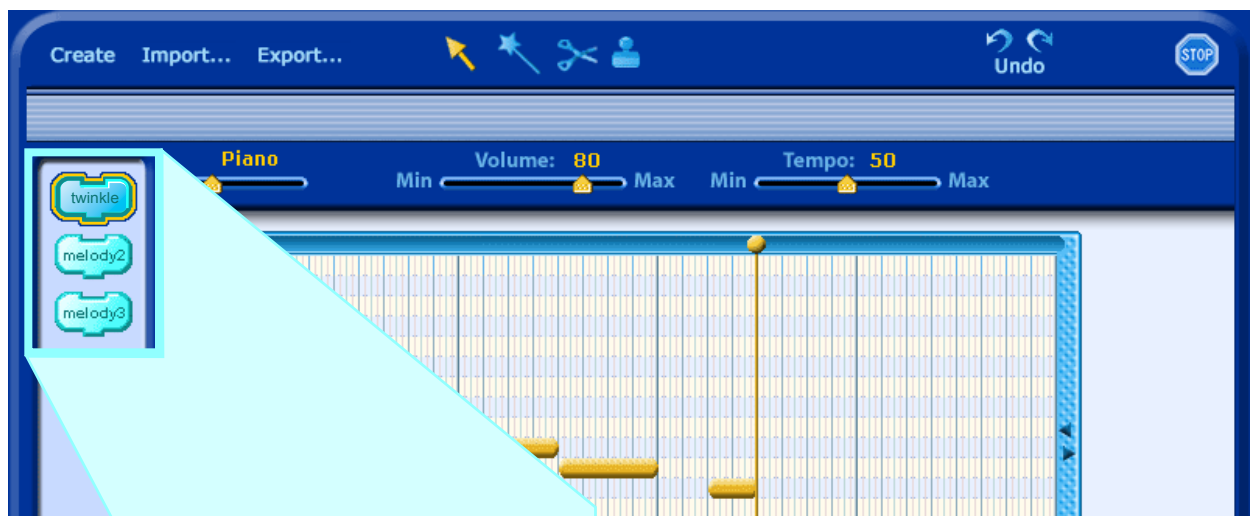
To add notes to your melody, either click on the keyboard, or click directly in the Melody Workspace.

When you are finished creating your melody, click **OK** to turn your melody into a block and exit the Melody Editor.

Click here to create a new melody



You can import and export melody files with extension .pbm

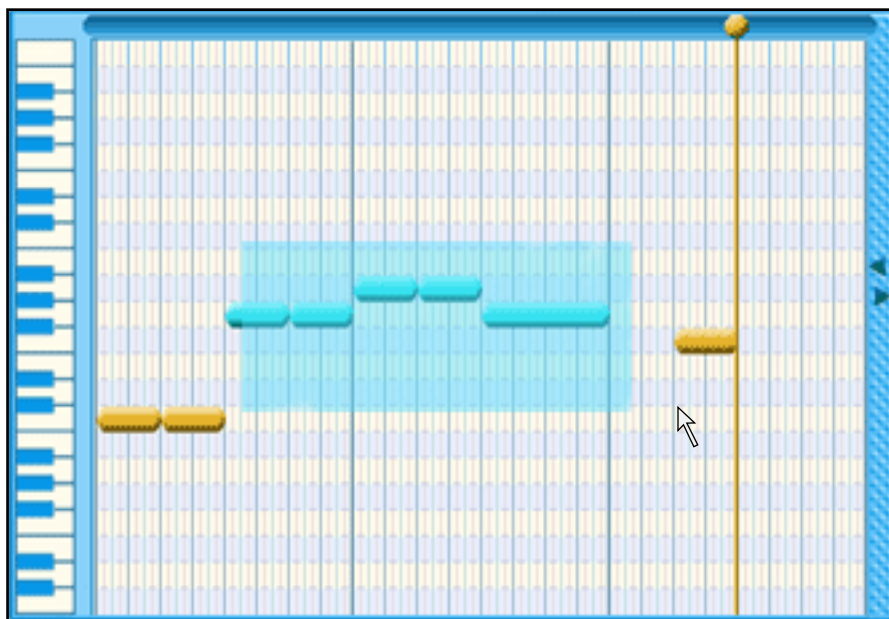







All of your melodies appear at the left of the Melody Editor. Click on the melody that you want to edit.

To change the name of a melody, click on the block and type a new name.

Use  to make a copy of a melody. Use  to delete melodies.



Drag on the background to select notes. Move the selected notes by dragging them to other location.

You can also copy , cut  and play  the selection.

You can also play the selection by double clicking on it.

Rhythm editor

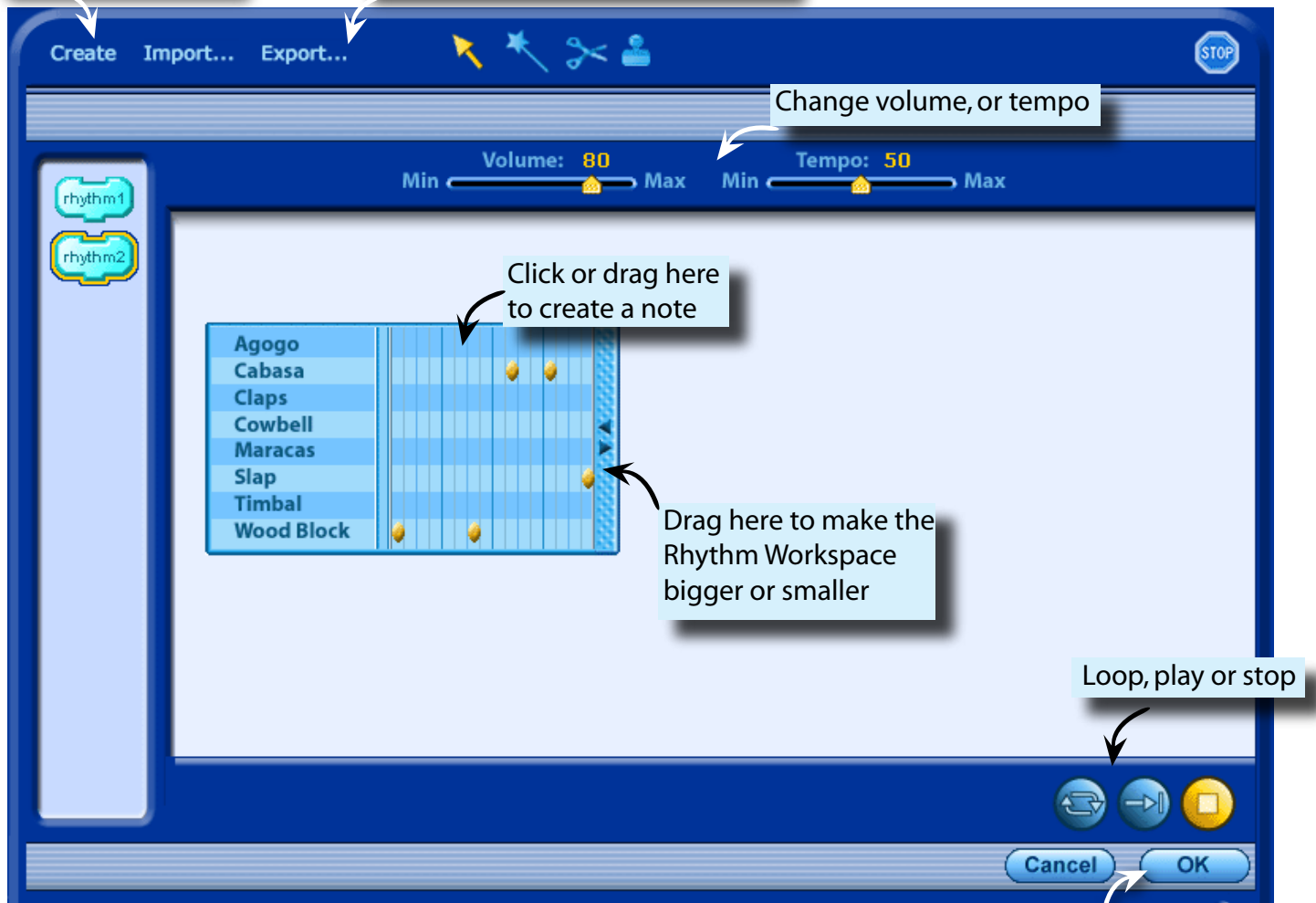
Click on **rhythms** to open the Rhythm Editor.

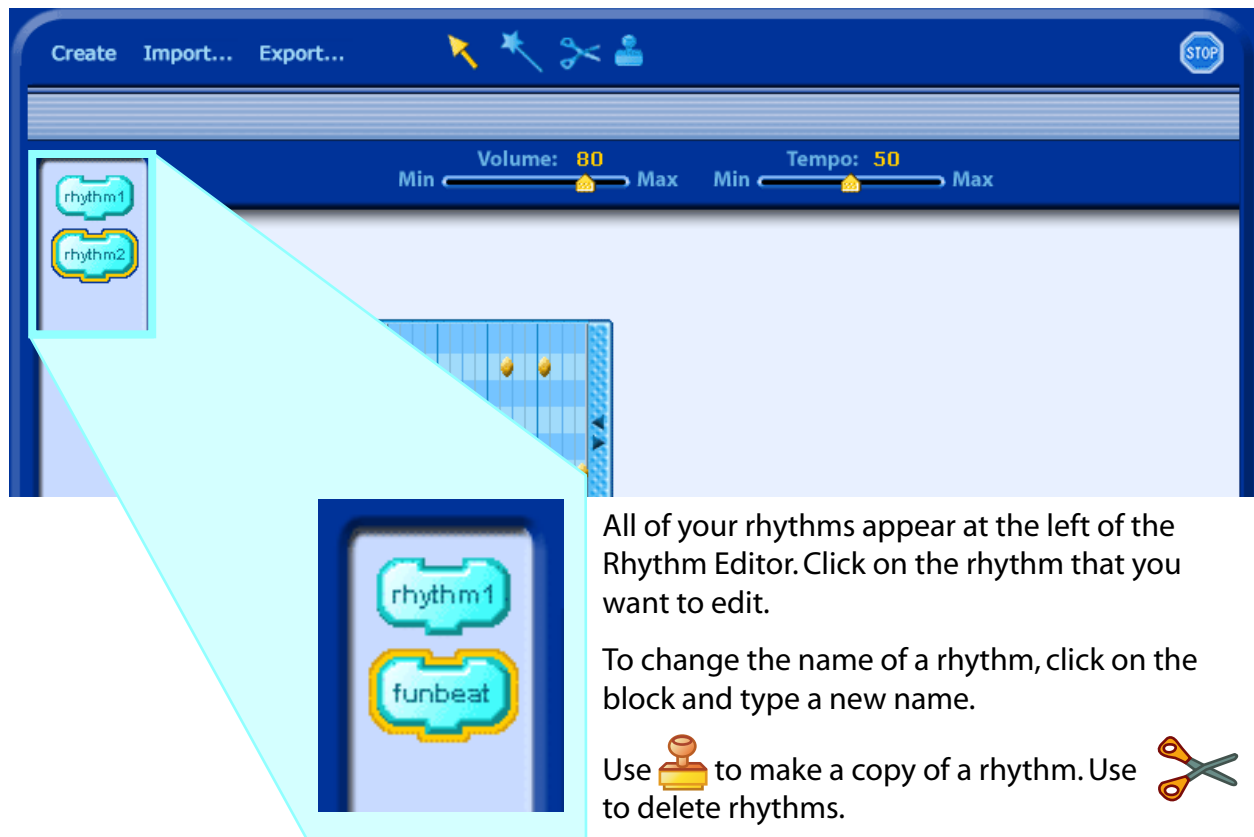
If this is your first rhythm, you will see a blank Rhythm Workspace for creating your rhythm. If you have previously created rhythms, you will see one of your existing rhythms in the Rhythm Workspace. You can edit this rhythm – or select one of your other existing rhythms to edit (by clicking on a block in the palette on the left side) – or create a new rhythm (by clicking on the “New Rhythm” button at the top-left).

When you are finished creating your rhythm, click **OK** to turn your rhythm into a block and exit the Rhythm Editor.

Click here to create a new rhythm

You can import and export rhythm files with extension .pbr

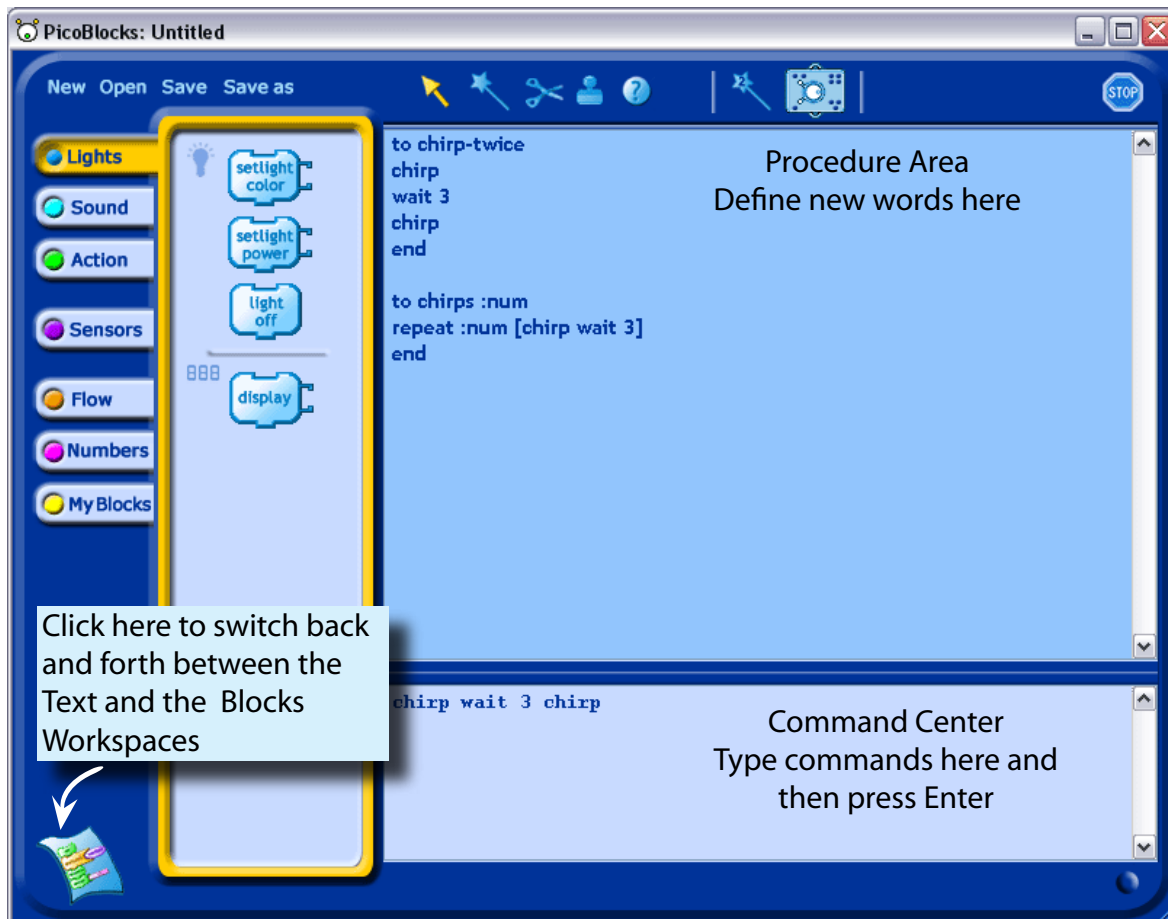
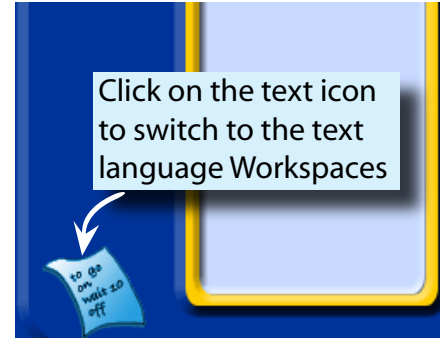




The PicoBlocks Text Language

Overview

PicoBlocks lets you construct computer programs by snapping different blocks together, like pieces of a puzzle. This approach makes it easy to get started; different kinds of blocks have different shapes and can only snap together in certain ways. The shapes of the pieces provide a lot of guidance as to how structure your programs, helping you build programs that work the way you want them to work. But if you try to write longer programs you may find that using the blocks can get cumbersome. For example, all of your stacks may no longer fit on the screen at the same time. Also, there are some advanced programming features that are not available using the blocks. As your programs become longer and more complex, you may want to consider using the Text Language that is available from within the PicoBlocks software. The PicoBlocks Text Language is similar to the computer language Logo. (For more information on Logo see www.logofoundation.org.) To begin writing programs in the Text Language, click on the text icon located in the lower left hand corner of the PicoBlocks Workspace. The Text Language Workspace, shown in the figure below, will appear.



Try typing the following commands in the Command Center. (Press the Enter key to run each line.)

```
chirp           The Cricket chirps!
chirp wait 2 chirp  The Cricket chirps twice.
```

Note that spaces are very important in the Text Language; they are used to determine where words start and end. For example, if you leave out the space between wait and 2 in the last example you will get an error message that says “I don’t know how to wait2”. This is because PicoBlocks does not understand the word `wait2`.

Each block in PicoBlocks has an associated word in the text language. The general rule for finding the name of the word in the text language that is associated with a particular block is a very simple one¹:

Just copy what’s written on the block, leaving out a space if there is one.

For example, plug a light into your Cricket and type the following in the Command Center:

```
setlightcolor 20  The light turns red
```

Note that there are no spaces in the word `setlightcolor`.

Now try typing the following line in the Command Center:

```
repeat 5 [chirp wait 2]
```

In the Text Language square brackets are used to enclose the list of words that are to be repeated. You can get guidance about how the text word corresponding to a particular block is meant to be used by clicking on that block, which you can find on the left hand side of the Text Language Workspace. For example if you click on the “repeat” block the following text will appear in the Command Center:

```
repeat 10 [ WHATEVER ]
```

In this message “WHATEVER” is meant as a place holder for the list of commands that you want repeated.

Other Examples

Plug in a touch sensor and a sound box and type:

```
waituntil [touch?] playsound 23
```

You should here a “meow” when you press the touch sensor on the touch sensor.

If you type: `forever [waituntil [touch?] playsound 23]`

you should now hear the “meow” each time you press the touch sensor. In these last two examples you will need to make sure you type the square brackets in the proper places.

Plug in a display and type: `display 7 * 9 + 6`

The Text language uses parentheses to determine the order of in which things are done. Compare the above result to what you get if you type:

```
display 7 * (9 + 6)
```

¹ Exceptions to this rule occur for the cases of the `if` and `ifelse` words; since the text word does not include “then”. For a complete list of the blocks and corresponding text words, see the [Summary of Text Language Words](#) chart at the end of this section.

Procedures

You can “teach” your Cricket to understand new words by defining procedures in the Procedures Area of the Text Language Workspace. Procedures are defined using the following general form:

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

The procedure definition starts with the keyword `to`, followed by the name of the procedure. Next comes the body of the procedure, which is a list of words that describe what the procedure is to do. The keyword `end` is used to complete the procedure definition.

For example, in the Procedure Area type:

```
to chirp-twice
  chirp wait 5 chirp
end
```

You’ve now taught your Cricket a new word; if you type the word `chirp-twice` in the Command Center, the Cricket will chirp twice.

Procedures with Inputs and Outputs

You can create procedures that take inputs or produce an output. For example, you can create a procedure named `chirps` that takes an input that is used as the counter in a repeat loop. Note that the name of the input must begin with a colon.:

```
to chirps :num
  repeat :num [chirp wait 5]
end
```

Typing

```
chirps 10
```

in the Command Center will cause the Cricket to chirp 10 times.

You can create a procedure that takes two inputs that will determine both the number of chirps and the duration of the pause between chirps:

```
to chirps2 :num :pause
  repeat :num [chirp wait :pause]
end
```

Typing

```
chirps2 10 3
```

in the Command Center will cause the Cricket to chirp 10 times, with a pause of 3 tenths of a second between chirps.

Procedures may return values using the output command. For example, you can define a procedure called `half` by:

```
to half
  output brightness / 2
end
```

Then, with a display and a light sensor plugged in, try typing in the Command Center:

```
display half
```

Or, you can define a procedure called `dark?` that returns a true or false result:

```
to dark?  
  output (brightness < 20)  
end
```

Now if try typing in the Command Center:

```
forever [if dark? [chirp]]
```

your Cricket should chirp when it is dark enough.

Make Your Own Blocks

You can use the Text Language to make different kinds of blocks, which you can then use to build programs in the PicoBlocks Workspace. This feature allows you to continue doing much of your programming using the Blocks Language, with the Text Language being used as a supplement when its advanced features are needed.

New blocks are defined by typing in the Procedures Area. The block definitions begin with the keyword `block`. For example:

```
block chirp1  
chirp wait 3 chirp  
end
```



Makes a block that chirps twice.

Blocks that you have defined will automatically be created and placed in the “My Blocks” area. You can create blocks with up to three inputs. For example, to create a block with one input:

```
block chirp2 :num  
repeat :num [chirp wait 3]  
end
```



Makes a block that chirps a number of times.

To create a block with two inputs:

```
block chirp3 :num :pause  
repeat :num [chirp wait :pause]  
end
```



Makes a block that chirps a number of times, with the ability to vary the length of the pause between chirps.

When you define a block that outputs a number, PicoBlocks will automatically create a number-shaped reporter block:

```
block bright2  
output 2 * brightness  
end
```



Reports a number twice as big as the brightness measured by the light sensor.

Use the keyword `bblock` when you want to make a block that outputs the result of a true/false condition:

```
bblock dark?  
output brightness < 10  
end
```

dark?

Using Tags in the Text Language

PicoBlocks “tags” are used when you are plugging in more than one of the same kind of part, two lights or two motors for example, and you want these parts to act differently. (See the previous [section on Tags](#) on pages 10 and 11.) In programs written in the Text Language, a word without a tag will address all parts of the same type. For example, if two lights are plugged into your Cricket, the command

```
setlightcolor 90
```

 will turn both lights blue.

If you want to refer only to the light plugged into the ● port, add a period to the end of the word, without a space:

```
setlightcolor. 90
```

Similarly, to refer only to the light plugged into the ●● port, add two periods to the end of the word, without spaces:

```
setlightcolor.. 90
```

To refer only to the light plugged into the ●●● port, add a period and a colon to the end of the word, without spaces:

```
setlightcolor.: 90
```

To refer only to the light plugged into the ●●● port just add two colons to the end of the word, without spaces:

```
setlightcolor:: 90
```

This method of tagging text words will work with other words². For example

```
display.: 57
```

will display the number 57 on a display plugged into the ●● port and

```
forever [if touch?: [chirp]]
```

will cause the Cricket to chirp if a touch sensor that is plugged into the ●● port is pressed.

Global Variables

In the “My Blocks” tab you will find blocks for storing numbers in two different “global variables” called `box1` and `box2`. In the Text Language you can use the keyword `global:` to create up to five additional global variables. For example, by including the line:

```
global: times, pause
```

² **Note:** Tags will not work with words related to the sound box.

in the Procedures Area you will create two new global variables called `times` and `pause`. Now, try typing the following in the command center:

```
storeintimes 5 storeinpause 3
repeat times [chirp wait pause]
```

The Cricket should chirp 5 times, with pauses of 3 tenths of a second between chirps.

If you click on the My Blocks tab you will also find that new blocks corresponding to the new variables `times` and `pause` have been created.

Local Variables

The word `make` allows you to change the value of an input to a procedure. For example, if you define a procedure called `rainbow` by:

```
to rainbow :a
  repeat 30 [setlightcolor :a make "a :a + 1 wait 2]
end
```

then, with a light plugged into your Cricket, typing

```
rainbow 20
```

in the Command Center will cause colors 20 through 49 to flash.




Variables of this kind are called “local variables” because they are only available for use by the procedure in which they are defined. Because PicoBlocks has a very limited number of global variables (7), local variables can be helpful because they do not use any of the global variable slots.


Tool Not Available in the Text Language







The “second wand” is not available in the Text Language.

Summary of Text Language Words








The following table shows the text word that corresponds to each PicoBlocks block and gives an example of how the word can be used.



Light		
Block	Word	Example Usage
	<code>setlightcolor number</code>	Setlightcolor takes an input from 0 to 100. <code>setlightcolor 70</code> The light changes to cyan and turns on.
	<code>setlightpower number</code>	Setlightpower takes an input from 0 to 100. <code>setlightpower 50</code> The light turns on at half power.
	<code>lightoff</code>	<code>lightoff</code> The light turns off.







Light		
Block	Word	Example Usage
	<code>display number</code>	Displays a value on the digit display: <code>display 709</code> The display shows the number 709.
Text only	<code>setrgb number number number</code>	Sets the color of the light. Setrgb takes 3 inputs (red, green, and blue), each with a range from 0 to 100. The three inputs mix to give one particular light color. <code>setrgb 0 100 100</code> The light turns cyan.




Sound		
Block	Word	Example Usage
	<code>chirp</code>	<code>chirp wait 3 chirp</code> The Cricket chirps twice.
	<code>note number number</code>	Note takes two inputs. The first input (a number between 0 and 100) determines the pitch of the note and the second input determines the duration of the note (in tenths of a second). <code>note 36 5</code> The sound box plays a note for half a second.
	<code>playsound number</code>	Plays a sound on the sound board. Playsound takes any number between 0 and 100. There are 24 built-in sounds. Playsound repeats the list in sequence for numbers greater than 24. <code>playsound 24</code> The sound box meows.
	<code>setinstrument number</code>	Sets the instrument for the next note, or melody to be played on the sound board. Setinstrument takes any number between 0 and 100: <code>setinstrument 4 note 36 5</code> The sound box plays violin note.
	<code>settempo number</code>	Sets the tempo at which the next rhythm or melody will be played on the sound board. Settempo takes any number between 0 -lowest pace- and 100 -fastest pace. <code>settempo 50 melody1</code> The sound box plays a melody at the standard tempo.
	<code>setvolume number</code>	Sets the volume at which the next note, sound, rhythm or melody will be played on the sound board. Setvolume takes any number between 0 -lowest volume- and 100 -highest volume: <code>setvolume 30 playsound 24</code> <code>setvolume 100 playsound 24</code> The sound box meows softly, then loudly.




In addition to the blocks above can add your own blocks to the Sound tab by creating melodies and/or rhythms in their respective editors. The melody or rhythm name becomes a text word that you can use.


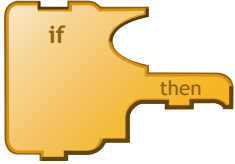
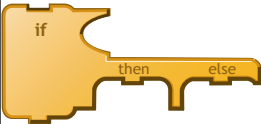




Action		
Block	Word	Example Usage
 	motoron motoroff	Turns a motor on or off. motoron wait 10 motoroff The motor spins for 1 second.
	motoronfor <i>number</i>	Turns the motor on for a set period of time (in tenths of a second). motoronfor 10 The motor spins for 1 second.
	reverse	Change the direction in which the motor spins. repeat 6 [motoronfor 3 reverse] The motor wiggles back and forth.
 	thisway thatway	Set the motor to spin in a specific direction. thisway motoronfor 10 thatway motoronfor 10 The motor spins one way for 1 second, then the other way for 1 second.
	setpower <i>number</i>	Setpower takes an input from 0 to 100. setpower 100 motoronfor 10 The motor spins quickly. setpower 30 motoronfor 10 The motor spins slowly.
text only	brake	Causes the motor to stop immediately. This is slightly different from <code>motoroff</code> , which does a short <code>brake</code> followed by a <code>coast</code> . motoron wait 10 brake The motor spins for 1 second, then immediately comes to rest.
text only	coast	Turns off the power to the motor but does not brake it, so that the motor slowly comes to rest. motoron wait 10 coast The motor spins for 1 second, then slowly comes to rest.

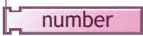





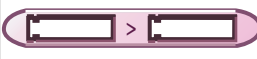


Sensors		
Block	Word	Example Usage
	touch?	Reports true if the touch sensor is pressed. waituntil [touch?] chirp The Cricket waits until the touch sensor is pressed before chirping.
	brightness	Reports the value measured by the light sensor as a number from 0 to 100. forever [display brightness wait 1] The display shows, and continually updates, the light level measured by the light sensor.




Sensors		
Block	Word	Example Usage
	loud?	Reports true (the number 1) if the value measured by the sound sensor is greater than 15. <code>waituntil [loud?] chirp</code> The Cricket will chirp after the sound sensor detects a loud sound.
	loudness	Reports the value measured by the sound sensor as a number from 0 to 100. <code>forever [display loudness]</code> The display shows, and continually updates, the sound level measured by the sound sensor.
	connected?	Reports true if the value measured by the resistance sensor is less than 50. <code>waituntil [connected?] chirp</code> The Cricket waits until the alligator clips on the resistance sensor connect before chirping.
	resistance	Reports a number from 0 to 100. The sliders are not available in text language. <code>forever [display resistance wait 1]</code> The display shows, and continually updates, the electrical resistance measured by the resistance sensor.
text only	rawresistance	Reports the value measure by the resistance sensor as a number between 0 and 1023; <code>rawresistance</code> provides finer resolution than <code>resistance</code> . <code>forever [display rawresistance wait 1]</code> The display shows, and continually updates, the electrical resistance measured by the resistance sensor on a scale of 0 to 1023. (Numbers of 1000 or greater appear as dashes.)
text only	rawrange	Rawrange is used to adjust the sensitivity of the resistance sensor. It takes as an input either the number 0, 1, or 2. An input of 0 results in the highest sensitivity and should be used when measuring resistances of 1000 ohms or less. An input of 1 is medium sensitivity and is the default setting. It works best for resistances of about 10,000 ohms. 2 is the lowest sensitivity and should be used to measure resistances of 100,000 ohms or more. <code>rawrange 0</code> <code>display rawresistance</code>
	<code>beamir number</code>	Beams a number from the Cricket's infrared transmitter. <code>beamir 4</code>
	newir?	Reports true (the number 1) if the Cricket's infrared receiver has received a new value. <code>waituntil [newir?] chirp</code> The Cricket chirps when a new number is beamed to it.




Sensors		
Block	Word	Example Usage
	ir	<p>Reports the value of the most recent number beamed to the Cricket.</p> <pre>to test waituntil [newir?] if ir = 10 [chirp] end</pre> <p>The Cricket chirps if the number beamed to it is equal to 10.</p>
	resettimer	<p>Resets to zero the value of the Cricket's internal timer. See next example.</p>
	timer	<p>Reports the Cricket's timer value. Timer measures time in hundreths of a second.</p> <pre>to test resettimer chirp waituntil [touch?] display timer end</pre> <p>The display shows the amount of time that has elapsed between when the Cricket chirps and when the touch sensor is pressed.</p>

Flow		
Block	Word	Example Usage
	wait <i>number</i>	<p>Waits a certain amount of time (in tenths of a second).</p> <pre>chirp wait 10 chirp</pre> <p>The Cricket pauses for a second between the two chirps.</p>
text only	mwait <i>number</i>	<p>Waits a certain amount of time (in milliseconds).</p> <pre>setlightcolor 20 mwait 10 lightoff</pre> <p>The light flashes very briefly.</p>
text only	no-op	<p>Does nothing for about 40 microseconds.</p> <pre>repeat 10 [no-op]</pre> <p>The Cricket waits for 400 microseconds.</p>
	waituntil [<i>condition</i>]	<p>Waits until the condition is true.</p> <pre>waituntil [touch?] chirp</pre> <p>The Cricket waits until you press the touch sensor before chirping.</p>
	forever [<i>list</i>]	<p>Repeats a list of instructions forever.</p> <pre>forever [chirp wait 3]</pre> <p>The Cricket chirps continuously.</p>

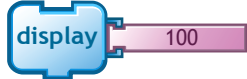
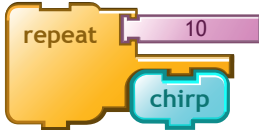
Flow		
Block	Word	Example Usage
	<code>repeat <i>number</i> [<i>list</i>]</code>	Repeats a list of instructions a set number of times. <code>repeat 5 [chirp wait 2]</code> The Cricket chirps five times.
	<code>if <i>condition</i> [<i>list</i>]</code>	Runs a list of instructions if a certain condition is met. <code>forever [if touch? [chirp wait 1]]</code> The Cricket chirps if you press the touch sensor and is quiet if you do not press it.
	<code>ifelse <i>condition</i> [<i>list1</i>] [<i>list2</i>]</code>	Runs the first list of instructions if a certain condition is met and the second list if it is not. <code>to test forever [ifelse touch? [setlightcolor random 0 99] [lightoff]] end</code> The light flashes random colors if you press the touch sensor and turns off if you do not press it.
	<code>stopall</code>	Stops everything: programs, motors, etc. Stopall is equivalent to pressing the button on the Cricket. <code>to test motoron forever [if touch? [stopall] setlightcolor random 0 99] end</code> The lights and motors stop when you press the button.
	<code>stopstack</code>	Terminates execution of a procedure, returning control to the calling procedure. <code>to test forever [if touch? [stopstack] setlightcolor random 0 99] end</code>
		There is no need for spacers in the text language.
		There is no need for spacers in the text language.





Numbers		
Block	Word	Example Usage
	<i>number</i>	Reports a number. Numbers in PicoBlocks are limited to integers between -32768 and +32767. <code>display 7</code> The display shows the number 7.
	<i>number + number</i>	Reports the sum of two numbers. <code>display 7 + 4</code> The display shows the number 11.
	<i>number - number</i>	Reports the difference of two numbers. <code>display 7 - 4</code> The display shows the number 3
	<i>number * number</i>	Reports the product of two numbers. <code>display 7 * 4</code> The display shows the number 28.
	<i>number / number</i>	Reports the quotient of two numbers, rounded down to the nearest integer. <code>display 7 / 4</code> The display shows the number 1.
text only	%	Reports the remainder when two numbers are divided. <code>display 7 % 4</code> The display shows the number 3.
	<i>random number</i> <i>number</i>	Report a random number, the two inputs determining the lower and upper limits <code>display random 0 10</code> The display shows a random number from 0 to 10.
	<i>number > number</i>	Reports true (the number 1) if the value of the first input is greater than the value of the second input. to test <code>forever [if brightness > 30 [chirp]]</code> end The Cricket chirps if the light sensor measures a value greater than 30.
	<i>number < number</i>	Reports true (the number 1) if the value of the first input is less than the value of the second input. to test <code>forever [if brightness < 30 [chirp]]</code> end The Cricket chirps if the light sensor measures a value less than 30.
	<i>number = number</i>	Reports true (the number 1) if the value of the first input is equal the value of the second input. <code>forever [if brightness = 30 [chirp]]</code> The Cricket chirps if the light sensor measures a value exactly equal to 30.




Numbers		
Block	Word	Example Usage
	<i>condition and condition</i>	<p>Reports true (the number 1) if the first and second conditions are both true.</p> <pre>to test forever [if (brightness < 30) and touch? [chirp]] end</pre> <p>The Cricket chirps if the light sensor measures a value less than 30 <i>and</i> the touch sensor is also pressed</p>
	<i>condition or condition</i>	<p>Reports true (the number 1) if the first or the second conditions are both true.</p> <pre>to test forever [if (brightness < 30) or touch? [chirp]] end</pre> <p>The Cricket chirps if the light sensor measures a value less than 30 <i>or</i> the touch sensor is pressed</p>
	<i>not condition</i>	<p>Reports true (the number 1) if the result of the condition is false (number 0).</p> <pre>to test forever [if not touch? [chirp]] end</pre> <p>The Cricket chirps if the touch sensor is <i>not</i> pressed.</p>

My Blocks		
Block	Word	Example Usage
	<i>block name</i> <i>block-definition</i> end	You can use the text language to make your own blocks. (See section Make Your Own Blocks above.)
	<i>storeinbox1 number</i>	<p>Stores a number in the global variable called box1.</p> <pre>storeinbox1 57 display box1</pre> <p>The display shows the number 57.</p>
	box1	<p>Reports the number stored in the global variable called box1.</p> <pre>storeinbox1 57</pre>
text only	global:	You can use the text language to create new global variables. (See section Global Variables above.)
text only	make "name value	Make allows you to change the value of a local variable. (See section Local Variables above.)

Common Error Messages

Error Message	Explanation
The Beamer cannot see the Cricket	<p>The Beamer is sending, but no Cricket is responding.</p> <p><u>Solution:</u> Make sure the Cricket is turned on, and that the Cricket and the Beamer are facing each other. For more information, see www.picocricket.com/troubleshooting</p>
Can't find the Beamer	<p>PicoBlocks wasn't able to open the communication port.</p> <p><u>Solution:</u> Make sure the Beamer is properly connected to your computer. If you are using Windows, make sure only one copy of PicoBlocks is running. For more information, see: www.picocricket.com/support</p>
Can't save an empty score	<p>You tried to export a melody or rhythm that is empty.</p> <p><u>Solution:</u> Make sure the block contains notes, then click Export again.</p>
Can't use ____ as a name	<p>PicoBlocks does not allow that word as a name. Numbers, names starting with % or [, and existing words are considered invalid.</p> <p><u>Solution:</u> Type another name.</p>
Communication problem	<p>There was a communication problem while PicoBlocks was downloading data to your Cricket.</p> <p><u>Solution:</u> Make sure that the Beamer and the Cricket are facing each other, then try again. If the Cricket's yellow antenna lights keep flashing even when you are not downloading, the Cricket is probably receiving infrared interference from a monitor or other device. Shield the Cricket from the monitor. For more information, see www.picocricket.com/troubleshooting</p>
Missing input to _____	<p>One (or more) of the blocks requires input.</p> <p><u>Solution:</u> Check that every block has something attached on its right side when required. (You can tell by the shape of a block whether it takes an input.)</p> <p>For example, display needs a number (or other value) attached as an input:</p>  <p>Repeat requires a number value at the top right. It also requires plus one or more blocks under its "arm" to repeat:</p>  <p>Tip: To check if a block needs inputs, click on the block with the Help tool ?</p>

Error Message	Explanation
I don't know how to _____	<p>PicoBlocks tried to run a name that doesn't exist.</p> <p><u>Solution:</u> If you name a stack, then PicoBlocks creates a block with that name. To run the block, you need to keep the named stack in the Workspace, so PicoBlocks knows what to do.</p>
The name ____ already exists	<p>There is already an existing block with the same name.</p> <p><u>Solution:</u> Type another name.</p>
Number block doesn't like _____	<p>You typed an invalid character in </p> <p><u>Solution:</u> Type only numbers.</p>
Project contains too many blocks	<p>Your project contains too many blocks.</p> <p><u>Solution:</u> Delete all stacks, melodies and rhythms that are not being used.</p>
Too many blocks in the stack	<p>The stack contains too many blocks (and uses up too much memory).</p> <p><u>Solution:</u> Use a repeat to do more with fewer blocks. Or, place a  on the top of the stack, and click on it to type in a name.</p>
Too many music blocks: Limit of 24	<p>You have made more than the maximum number of music blocks. The maximum number of melodies and rhythms is 24 blocks.</p> <p><u>Solution:</u> Delete some of the melody or rhythm blocks so that the project has no more than 24 total.</p>
Too many notes for one music block	<p>The current melody or rhythm is too long.</p> <p><u>Solution:</u> Divide the melody or rhythm into two blocks. To do this, first copy the current music block. In this block, delete the second half of the notes. In the other block, delete the first half of the notes. Now you can use these two blocks together to play the entire sequence.</p>
waituntil needs input	<p>The waituntil block needs an input block that fits.</p> <p><u>Solution:</u> Attach a block that fits onto the right side of the waituntil block.</p> <p>Examples:</p> <div style="text-align: center;">  </div> <p>or</p> <div style="text-align: center;">  </div>

Error Message	Explanation
You can't copy an empty music block	You tried to copy a melody or rhythm that is empty. <u>Solution:</u> Add notes, then copy the block.
You can't name a stack _____	The name typed on the  contains invalid characters. It cannot contain the following characters: % " + - * /] [or a space. Numbers can only be used in combination with letters. <u>Solution:</u> Type letters on the 
You don't say what to do with _____	PicoBlocks tried to run a block that returns a value, but there was no block attached to receive that value. <u>Solution:</u> Attach a block that will fit that shape (and receive the value). For example: 

Text-Only Error Messages

The following error messages only appear when using the PicoBlocks Text Language.

Error Message	Explanation
_____: a block can't have more than 3 inputs	You can't make a block with more than 3 inputs. <u>Solution:</u> Store the other values as global variables, and use the names of these variables in your program. For more information, see Global Variables in the PicoBlocks Text Language section of this reference guide.
_____ has no value	PicoBlocks could not find a value for the specified name. <u>Solution:</u> In the Text Area, check that you have correctly typed the name of the input. Inputs should have a colon preceding them, but global variables should not.
_____: output block can only have 1 input	A block that outputs can only have 1 input. <u>Solution:</u> Store the other values as global variables, and use the names of these variables in your program. For more information, see Global Variables in the PicoBlocks Text Language section of this reference guide.
() error misplaced)	A parenthesis is missing or misplaced. <u>Solution:</u> Check that there are an equal number of open and closed parenthesis (), and that they are correctly placed.

Error Message	Explanation
Can't name a global _____	<p>You need to type in a different name for the global variable.</p> <p><u>Solution:</u> Use only letters and numbers as names for global variables. See Global Variables in the PicoBlocks Text Language section of this reference guide for more information.</p>
Can't use _____ as name of an input	<p>PicoBlocks doesn't like numbers as the name of an input.</p> <p><u>Solution:</u> You need to type in a different name for the input. Avoid using numbers as input. Also, make sure to put a colon before the name of the input.</p> <p>For example:</p> <pre>block mydance :turns repeat :turns [motoronfor 10 reverse] end</pre>
Missing [Missing]	<p>There is a missing bracket:] or [</p> <p><u>Solution:</u> Make sure there are an equal number of open and closed brackets.</p>
Missing name after to Missing name after block Missing name after block	<p><u>Solution:</u> The words to, block, and block need to be followed by a valid name. For example:</p> <pre>block mylights setlightcolor 60 wait 10 lightoff end</pre>
No output from _____	<p>The word or block indicated does not provide output.</p> <p><u>Solution:</u> Make sure every block has something attached on its right side when required. See the PicoBlocks Text Language section for more on how to use output.</p>
Too many globals	<p>There are too many global variables in the project. The maximum number of variables is 7 (including box1 and box2).</p> <p><u>Solution:</u> Delete some of the global variables so that the project has no more than 7 total.</p>
waituntil needs a list as input.	<p>waituntil needs a list as input.</p> <p><u>Solution:</u> Add a list in brackets after waituntil</p> <p>For example:</p> <pre>waituntil [touch?] chirp</pre>

Error Message	Explanation
make needs a local variable	<p>Make needs to be followed by the name for a local variable.</p> <p><u>Solution:</u> Check that the word after make has a quotation mark before it, and is a local variable.</p> <p>For example:</p> <pre>block fastlight :n setlightcolor :n make "n :n / 10 wait :n lightoff end</pre>