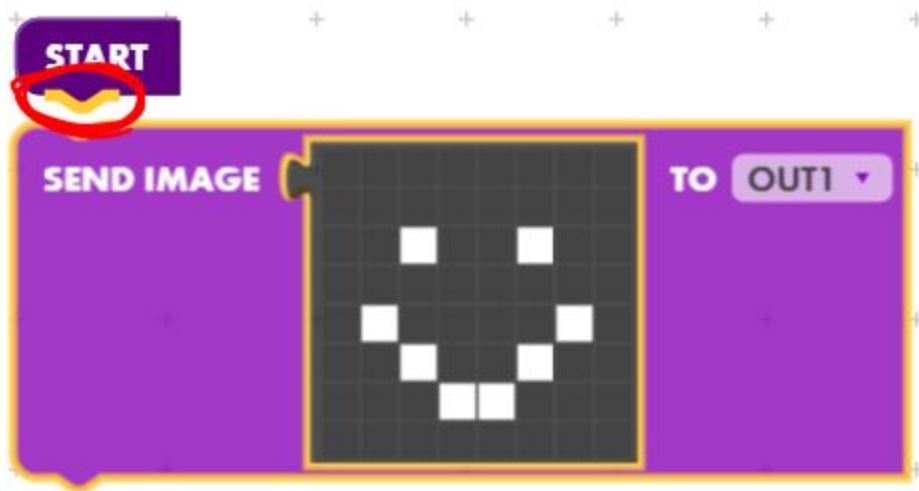


LittleBits Blockly Key

Blockly Intro	1
LittleBits Code Kit	2
Images	3
Colors	6
Text	7
Sound	8
Input/Output	9
Timing	10
Logic	12
Loops	14
Math	16
Variables	18
Functions	20
Code Examples	22
#1	22
#2	23
#3	24
#4	24
#6	27
#7	28
#8	29
#9	30

Blockly Intro

Blockly code fits together like blocks!

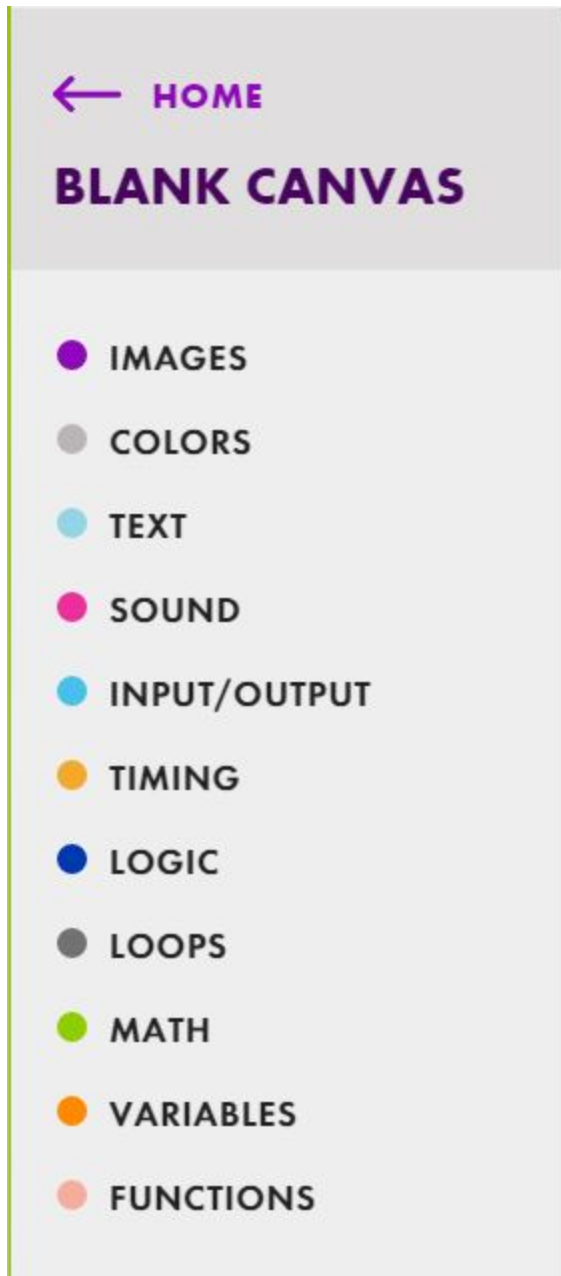


Watch for the **highlighted** connection, and snap blocks together to write programs.



Some Blocks don't fit together, that's just Blockly telling you to try another method!

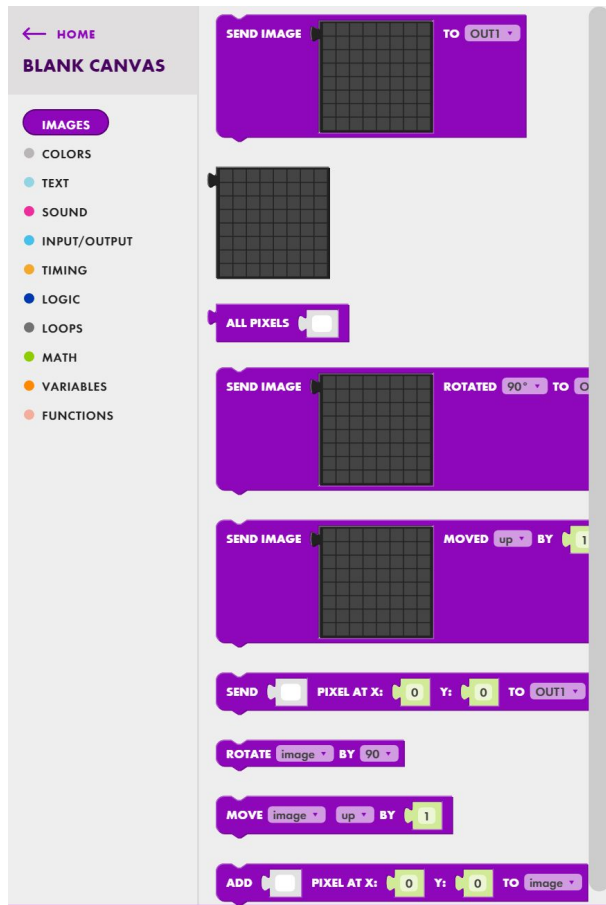
LittleBits Code Kit

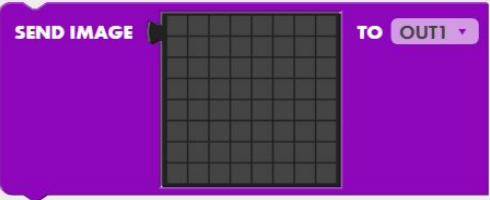


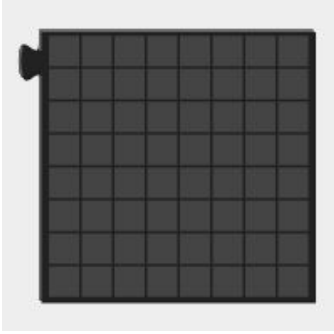
When you open LittleBits code kit, you will see this menu to the left of your screen. All the blocks you will need to program your LittleBits are located here!

This guide will take you through each block, its description, and its use in a sample program. Additional sample programs and links to download them are provided below for your convenience.

Images



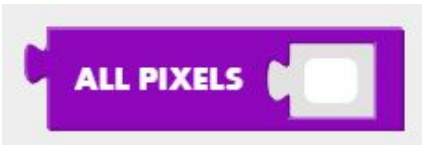
Block	Description	Code Example
	<p>Send an image you designed to your LED matrix.</p>	<p>#2</p>



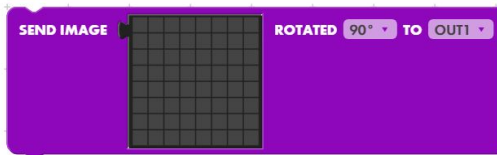
This grid represents your LED matrix. Choose your pixel colors here.

#2

(you will often see this block nested within others)



A quick way to set all the pixels on the LED matrix to the same color.



Send the image you specify rotated:

90°	turned to the left
180°	turned upside down
270°	turned to the right



Send the image you specify moved up/down/left/right by one pixel.




#3



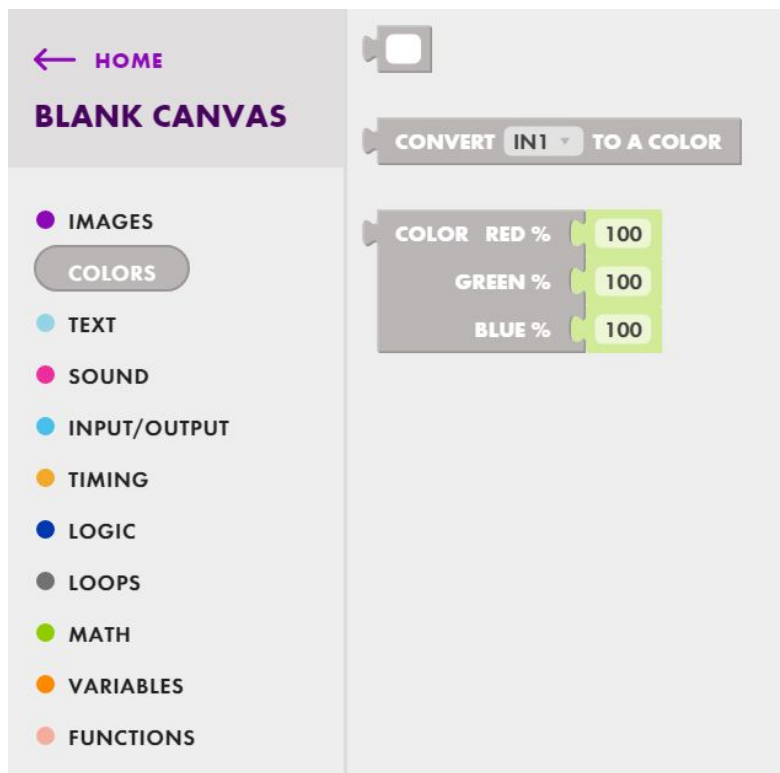
Set the color of a specific pixel on the LED matrix, using this coordinate system:




#5

0	1	2	3	4	5	6	7
1							
2							
3							
4							
5							
6							
7							

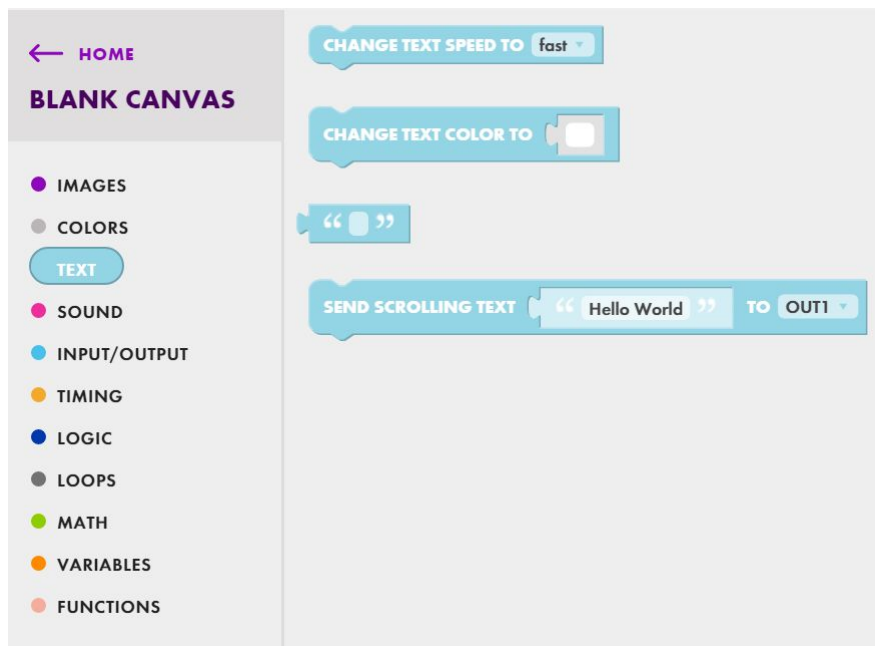
	<p>Rotate your image at the angle you specify.</p>	<p>#4</p>
	<p>Send the image you specify moved up/down/left/right by the number of pixels you specify.</p>	<p>#3</p>
	<p>Set the color of a specific pixel in your LED matrix. (uses the same coordinate system from above)</p>	

Colors



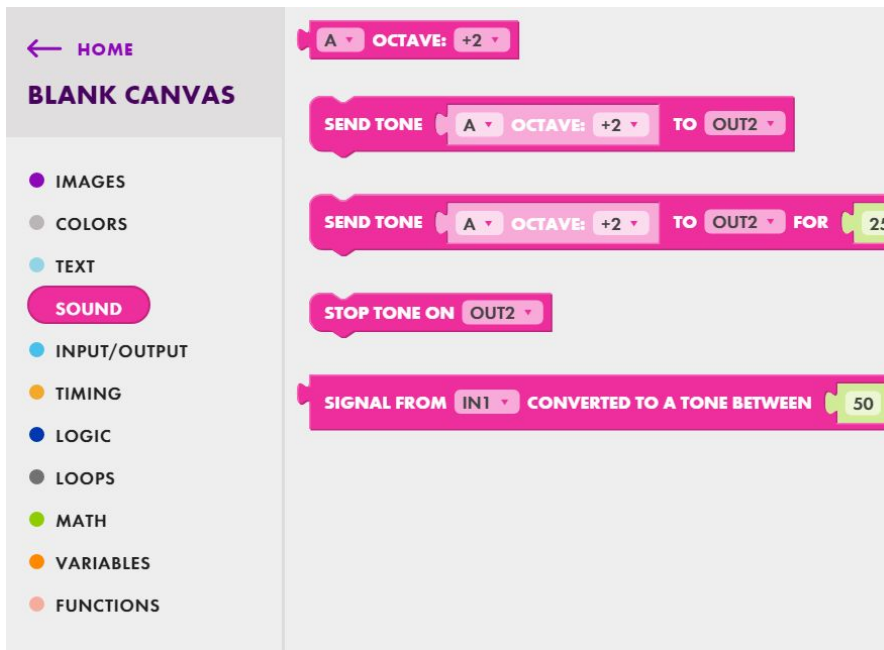
Block	Description	Code Example
	This block represents a color.	
	Convert the signal from one of the CodeBit's inputs to a color value.	#6
	Create a custom color using RGB values.	#6






Text



Block	Description	Code Example
	Choose the speed you want your text to scroll	#8
	Set the color of your text.	#8
	Type text you want to use in your program. (not limited to the LED matrix)	
	Type the text you want to see on the LED Matrix, and send it.	#8





Sound



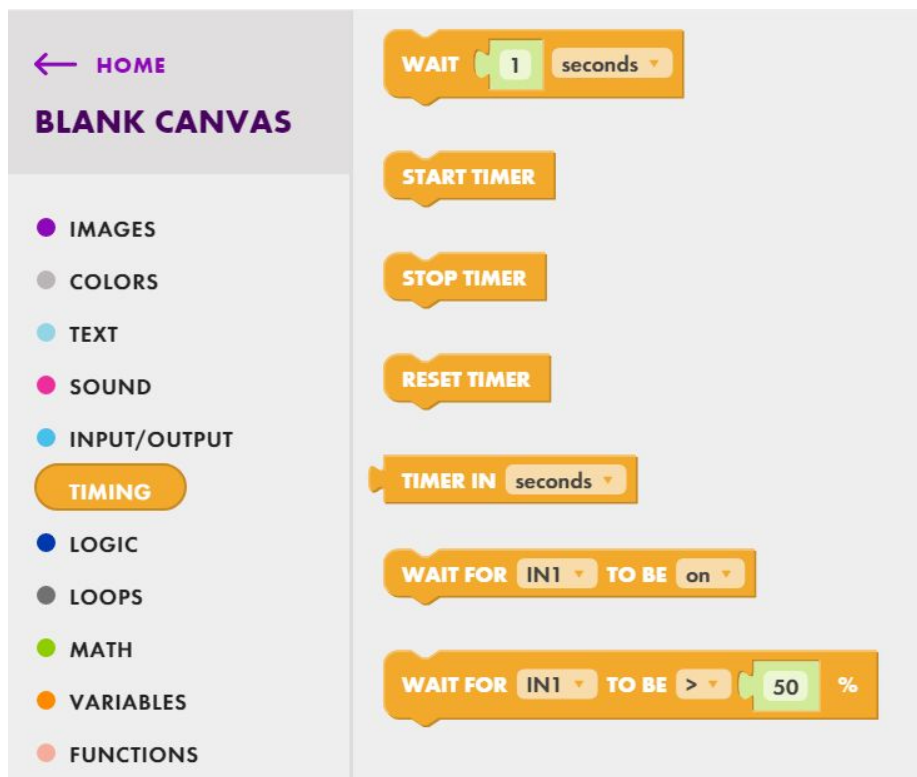
Block	Description	Code Example
	Specify a tone for oscillator to play.	(similar to #9)
	Send a tone you specify to be played.	(similar to #9)
	Send a tone for a specific amount of time.	#9
	Stop any tones that are playing.	
	Convert an input signal to a tone.	(similar to #7)





Input/Output




The screenshot shows the Scratch 'INPUT/OUTPUT' category interface. On the left is a sidebar with a 'HOME' button and a 'BLANK CANVAS' button. Below these are several category buttons: IMAGES, COLORS, TEXT, SOUND, INPUT/OUTPUT (highlighted), TIMING, LOGIC, LOOPS, MATH, VARIABLES, and FUNCTIONS. On the right, four code blocks are displayed: 'SIGNAL FROM IN1', 'SIGNAL FROM IN1 IS ON', 'SEND 100 % SIGNAL TO OUT1', and 'SEND on SIGNAL TO OUT1'.

Block	Description	Code Example
	This represents whatever signal may come from an input on the codeBit.	#7
	Returns TRUE if the input is over 50%, FALSE if not.	#8
	Sends a signal % you choose to an output on the codeBit.	
	Sends an ON/OFF signal to an output on the codeBit.	

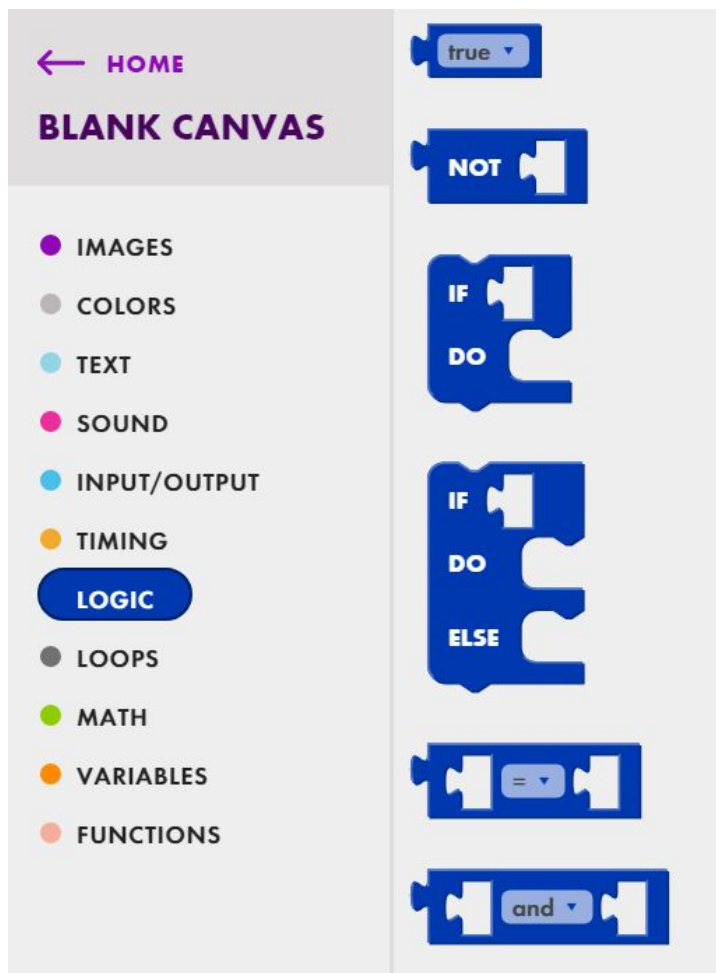
Timing



Block	Description	Code Example
	Pause the program for the amount of time you specify.	#2, #3, #4
	Starts a new timer or continues a timer that has been stopped.	
	Stops the timer that is currently counting.	
	Set the timer back to 0.	

	<p>Decide whether you want to measure time in seconds or milliseconds.</p>	
	<p>Pauses the program until the input you specify is ON/OFF.</p>	#1
	<p>Pauses the program until the input you specify is greater than/less than a % value you specify.</p>	

Logic









← HOME

BLANK CANVAS

- IMAGES
- COLORS
- TEXT
- SOUND
- INPUT/OUTPUT
- TIMING
- LOGIC**
- LOOPS
- MATH
- VARIABLES
- FUNCTIONS

Logic blocks available:

- true
- NOT
- IF DO
- IF DO ELSE
- =
- and

Block	Description	Code Example				
	<p>True/False blocks are useful to add to other logical blocks.</p> <p>Ex: IF (condition) = TRUE</p>					
	<p>This block negates whatever you put inside it.</p>					
	<p>IF (condition is met) DO (thing you choose)</p>	#7				
	<p>IF (condition is met) DO (thing you choose) (or if the condition is not met) ELSE (do something else)</p>	#8				
	<p>This block compares two values on the basis of whether one is =, ≠, ≤, <, ≥, or > the other.</p>	#7				
	<p>Puts two blocks together within an if statement:</p> <table border="1" data-bbox="748 1635 1159 1803"> <tbody> <tr> <td>and</td> <td>Both must be true</td> </tr> <tr> <td>or</td> <td>At least one must be true</td> </tr> </tbody> </table>	and	Both must be true	or	At least one must be true	
and	Both must be true					
or	At least one must be true					

Loops

← HOME
BLANK CANVAS





- IMAGES
- COLORS
- TEXT
- SOUND
- INPUT/OUTPUT
- TIMING
- LOGIC
- LOOPS**
- MATH
- VARIABLES
- FUNCTIONS

DO FOREVER

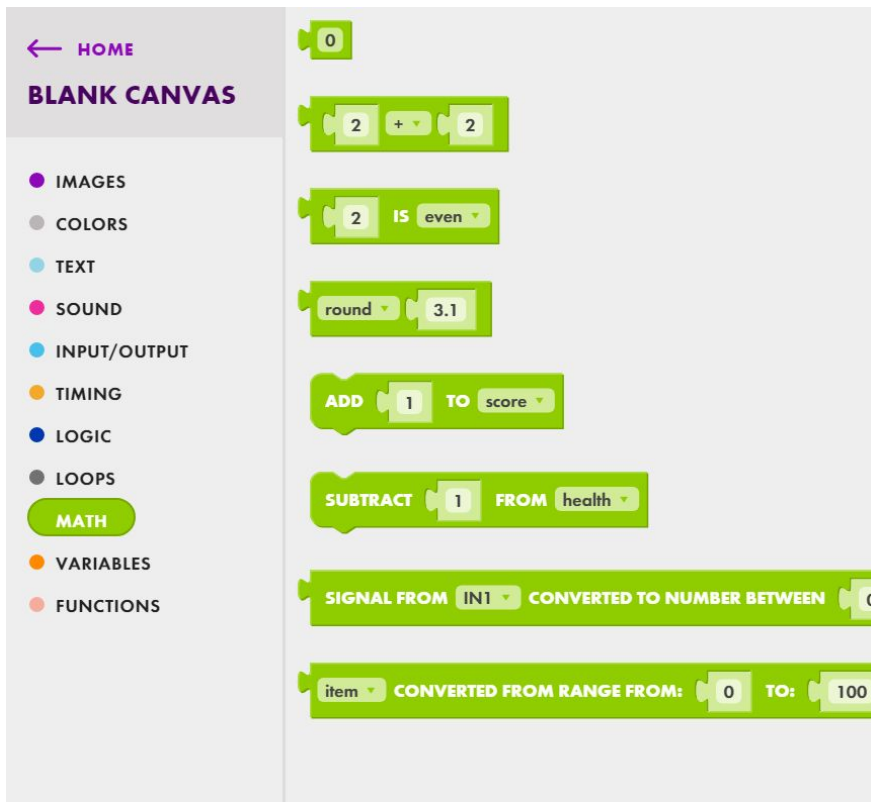
REPEAT 10 TIMES
DO




REPEAT while
DO






COUNT WITH i FROM 1 TO 10 BY 1
DO

Block	Description	Code Example
	<p>Repeat the blocks inside the loop over and over forever.</p>	<p>#7, #8</p>
	<p>Repeat the blocks inside the loop however many times you specify.</p>	<p>(similar to #5)</p>
	<p>Repeat the blocks inside the loop while a condition is met (or until a condition is met).</p>	
	<p>Repeat the blocks inside the loop however many times you specify.</p> <p>This loop is special because it allows you to use “i” as a variable within itself that increments along with the loop.</p>	<p>#5</p>

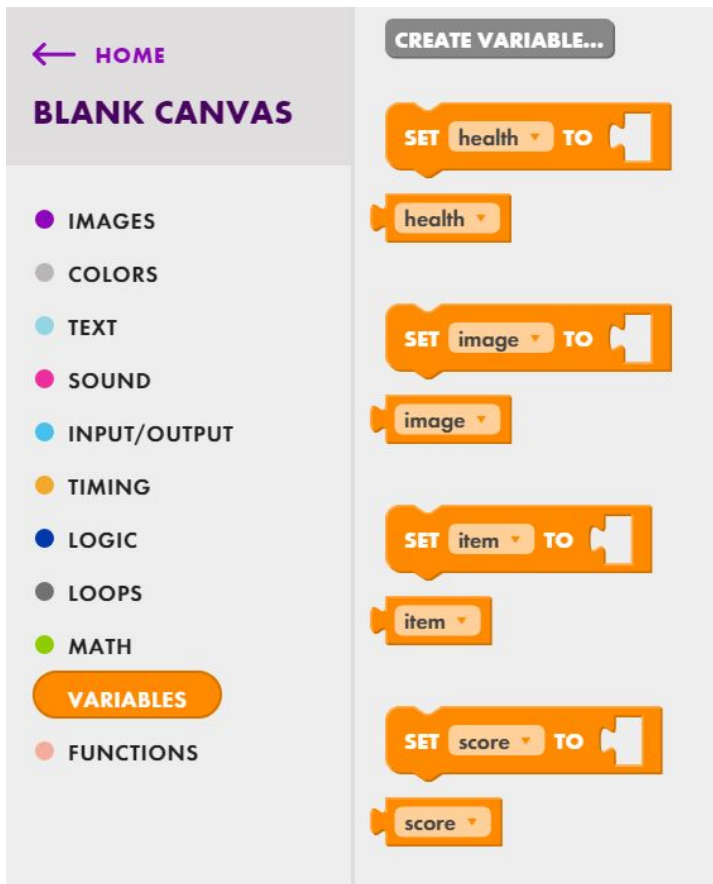
Math








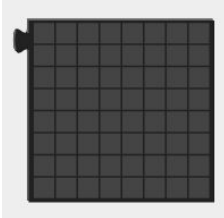

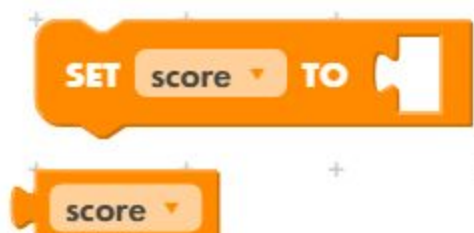
Block	Description	Code Example
	This is just a number.	#7
	Add two numbers together.	
	Check if a number is even/odd.	

	<p>Round to the nearest whole number.</p>	<p>#7</p>
	<p>Add to a variable (only works if the variable represents a number).</p>	
	<p>Subtract from a variable (only works if the variable represents a number).</p>	
	<p>Convert a signal to a number in the range of your choice.</p>	<p>#7</p>
	<p>Convert the value of a number from one range to another.</p>	<p>(similar to #7)</p>

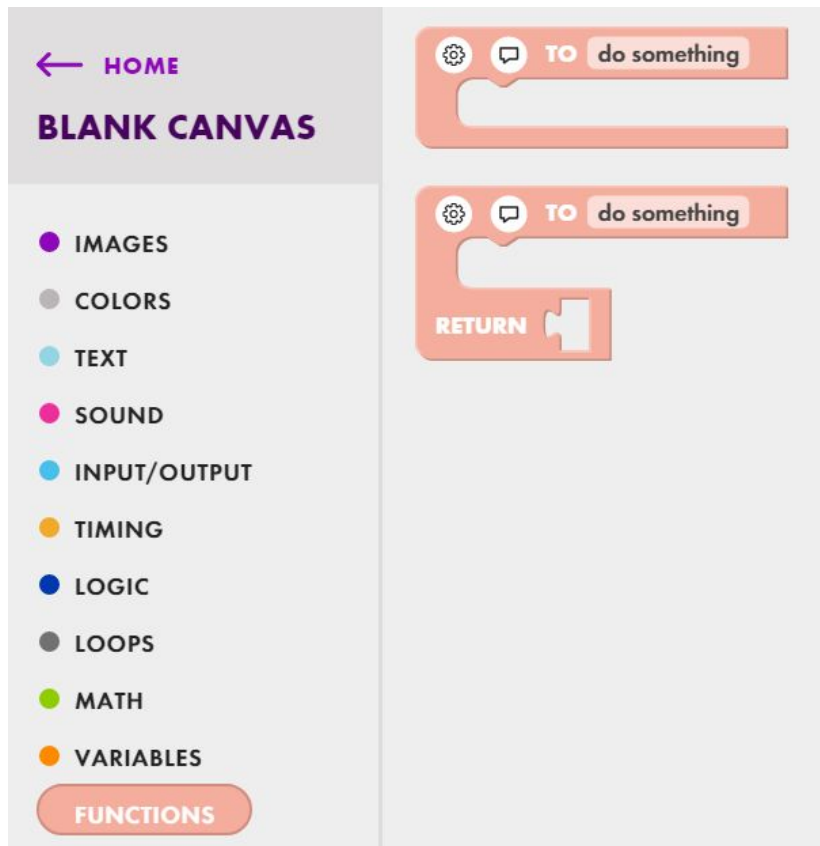
Variables


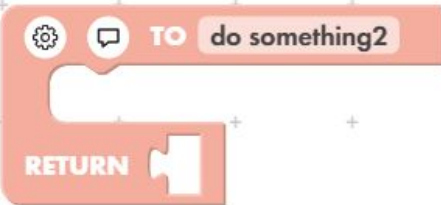




Block	Description	Code Example
	<p>Create a new variable, name it, and see it appear in the list with the others.</p>	<p>#9</p>
	<p>In programming, a variable is a place where we store information. First you “set” it by giving it a name and a value, then you can “reference” it by using it later in the program.</p> <p>You can change the value at any time in your program.</p>	

	<p>This variable is useful within this loop:</p>  <p>As your program loops through the code, this variable represents the iteration it is on.</p> <p>Ex: if you started at 0, and loop 3 times, i=2</p>	<p>#5</p>
	<p>This variable represents an image you have created using the LED matrix. Set it using this block:</p>  <p>And refer to it with the other.</p>	<p>#4</p>
	<p>This variable is more generic, set it to anything you want.</p>	
	<p>This variable is more generic, but a variable named score should probably be a number.</p>	

Functions

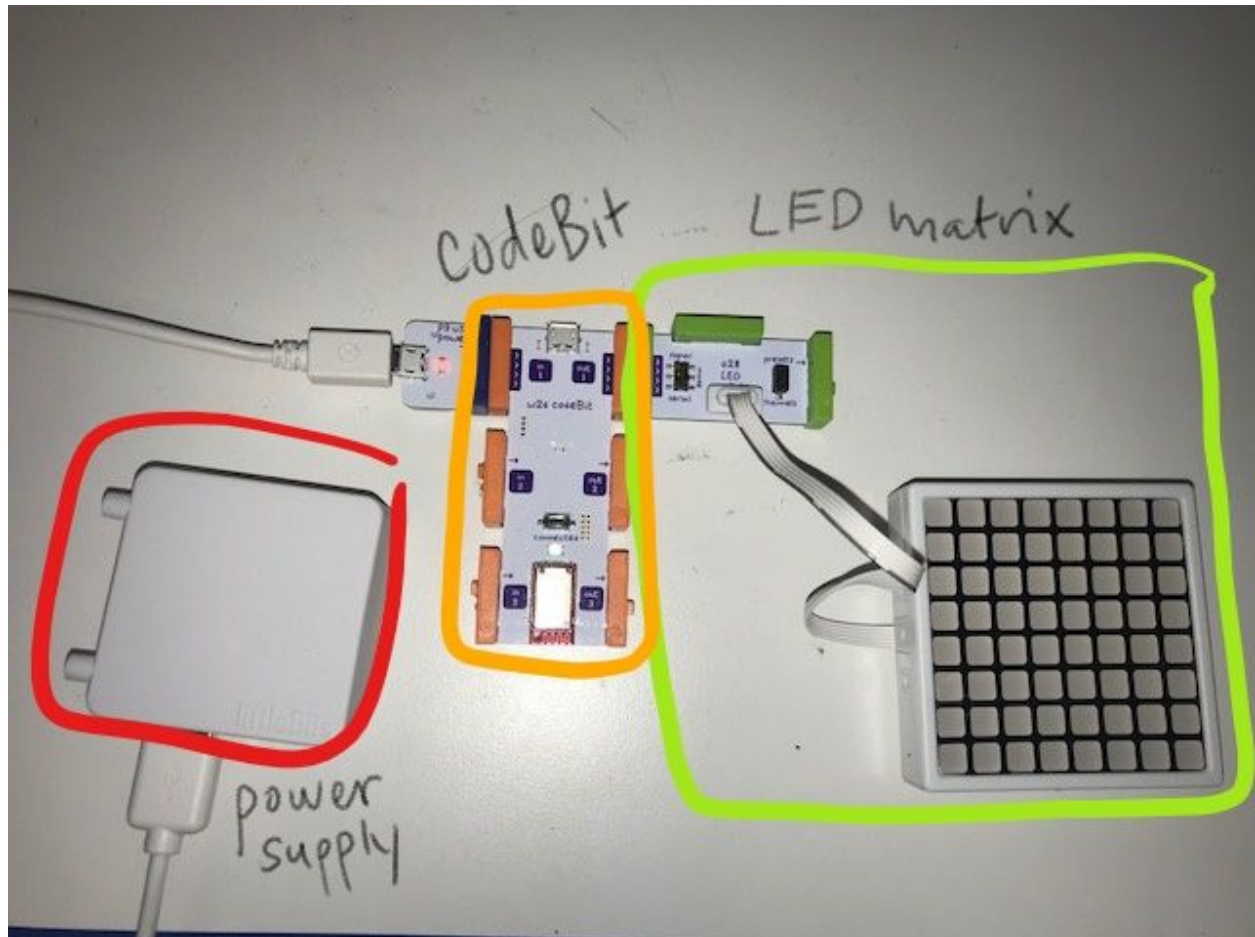


Block	Description	Code Example
	<p>A function is a program that only runs (or starts) when it is called.</p> <p>Write the function here, and give it a descriptive name. Then call it using the third block below.</p>	#9
	<p>When you run this function, it gives you back a value (usually something you determined within it).</p>	(similar to #8)

	<p>You can use the same function multiple times within your code without having to copy and paste a huge block of code each time.</p> <p>Once you wrote the function, place this block in your code and it will run.</p>	#9
	<p>Once you wrote the function, place this block in your code and it will run.</p>	

Code Examples

You will need this LittleBits setup for examples #1-5

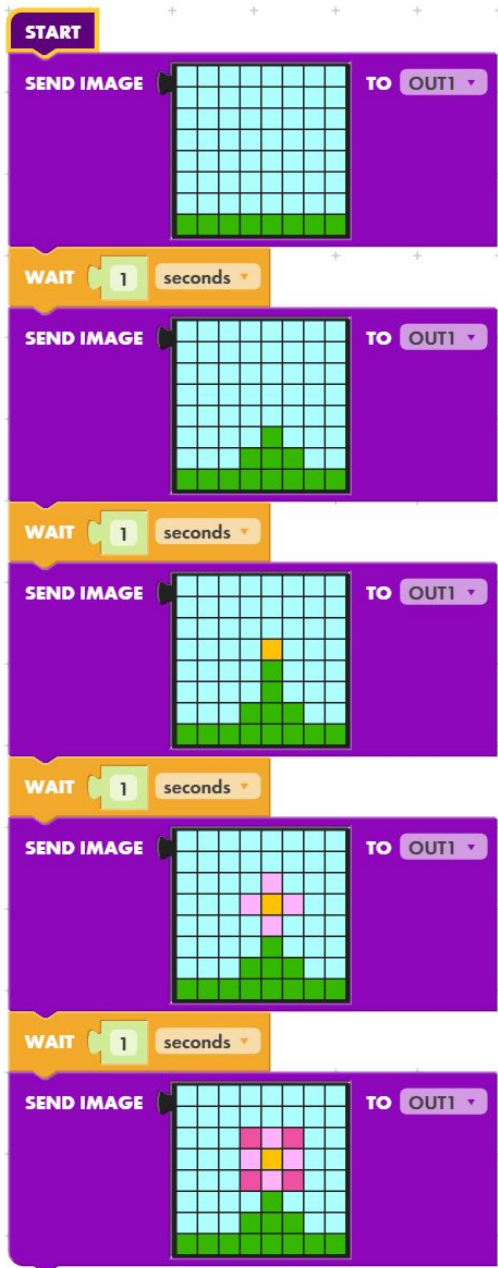


#1



This program simply waits for a signal from IN1 on the codeBit, then displays your message scrolling on the LED matrix.

#2



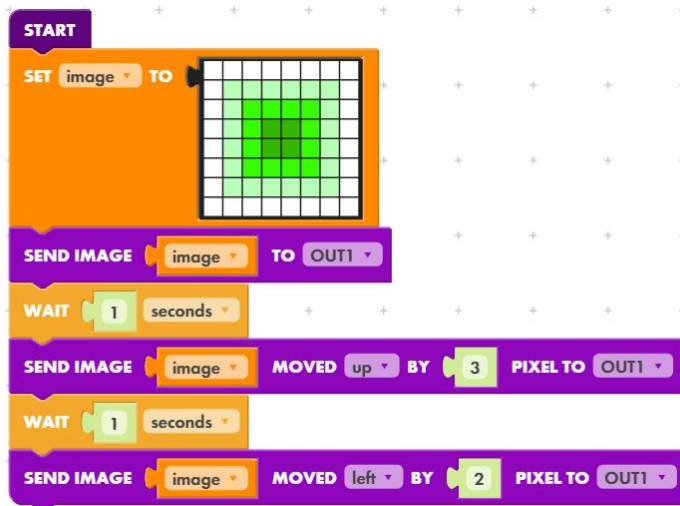
This program sends a series of images to the LED matrix. When they are sent in quick succession, it looks like a flower growing.

(Think of it like the programming version of a flipbook)

To make the animation look more smooth, reduce the wait time.

The image at the end will remain on the screen indefinitely (unless you add a blank image to the end of the program).

#3



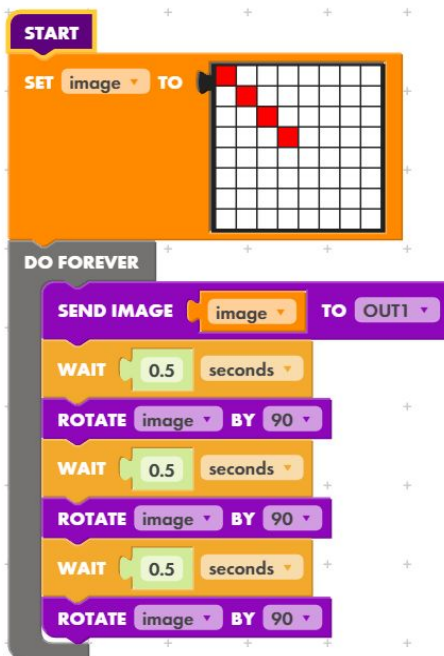
This program sets a variable called “image” to refer to the green LED matrix pattern. In the rest of the program, the block called image refers to that pattern. Run this code to see the pattern move around the screen:

First in the center

Then up (revealing 3 rows of blank space)

Then to the left (revealing 2 columns of blank space)

#4



This program sets a variable called “image” to refer to the red LED matrix pattern. In the rest of the program, the block called image refers to that pattern.

Run this code to see the pattern move around the screen (repeating forever due to the forever loop) :

First display in the original orientation

Then turned 90° to the left

Turned again

Turned again

The result is a spinning pinwheel effect!

#5

This code repeats 8 times, counting from 0 to 7 each time it loops, and changing the value of the variable “i” every iteration. Counting this way is useful because we can use “i” within the program. Here it is used as the x-coordinate where we will place our yellow pixel.

```
START
COUNT WITH i FROM 0 TO 7 BY 1
DO
  SEND PIXEL AT X: i Y: 0 TO OUT1
  WAIT 0.5 seconds
```

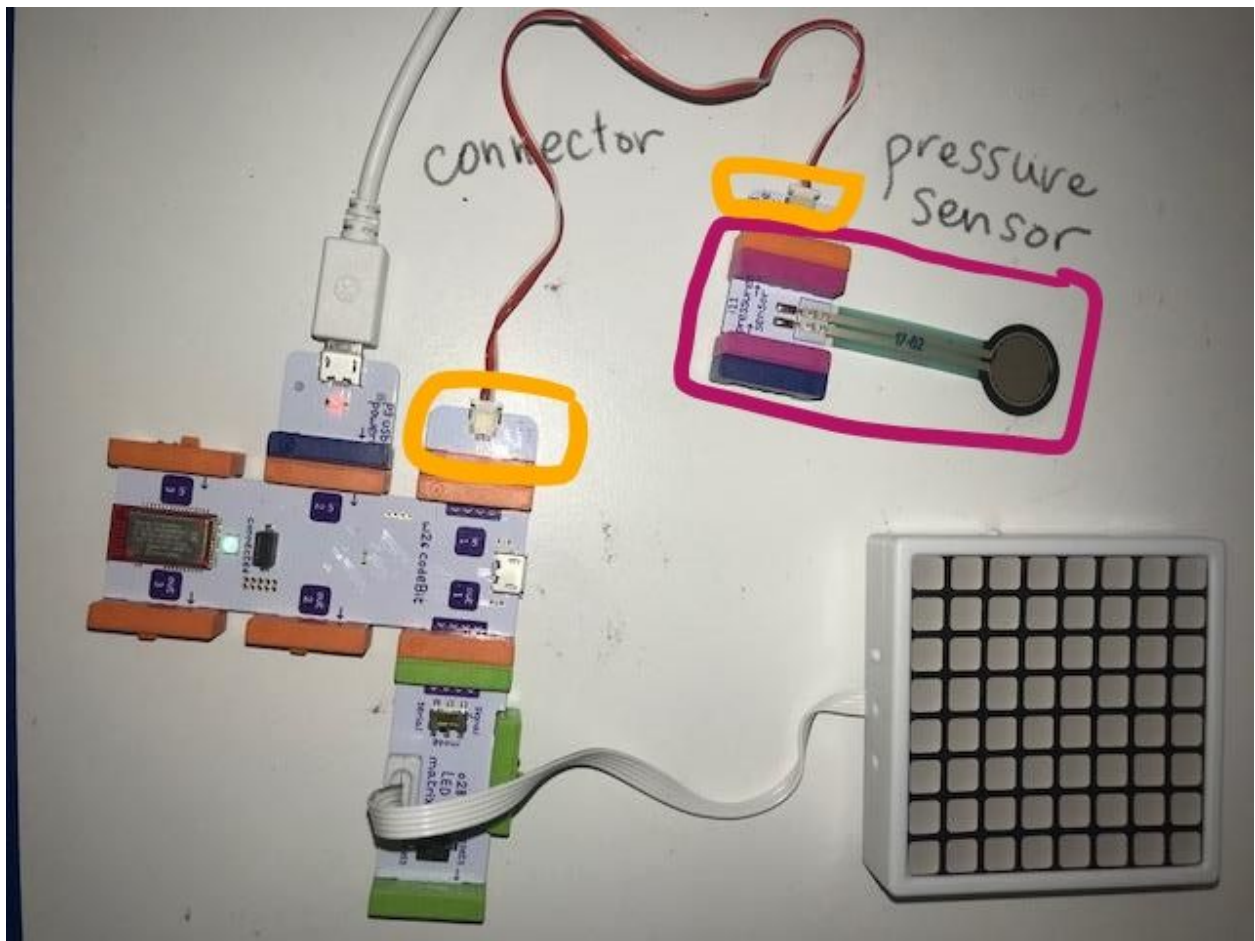
Each time the program loops, the yellow pixel moves one spot to the right

X, Y	1, 0	2, 0	3, 0	4, 0	5, 0	6, 0	7, 0
1, 0	1, 1	1, 2	1, 3	1, 4	1, 5	1, 6	1, 7
2, 0	2, 1	2, 2	2, 3	2, 4	2, 5	2, 6	2, 7
3, 0	3, 1	3, 2	3, 3	3, 4	3, 5	3, 6	3, 7
4, 0	4, 1	4, 2	4, 3	4, 4	4, 5	4, 6	4, 7
5, 0	5, 1	5, 2	5, 3	5, 4	5, 5	5, 6	5, 7
6, 0	6, 1	6, 2	6, 3	6, 4	6, 5	6, 6	6, 7
7, 0	7, 1	7, 2	7, 3	7, 4	7, 5	7, 6	7, 7

This small addition to the program makes a big change. “Nesting” the original loop inside another loop means that each time the “k” loop iterates, the “i” loop iterates eight times. This program goes left to right through every row and column.

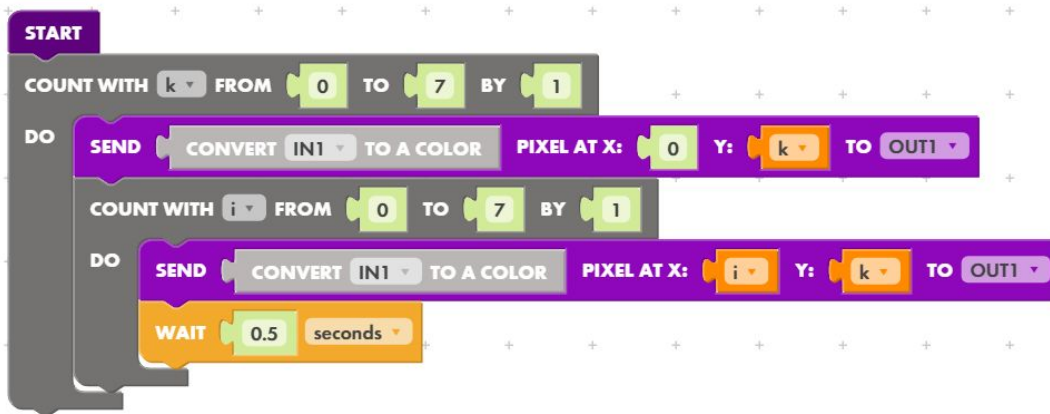
```
START
COUNT WITH k FROM 0 TO 7 BY 1
DO
  SEND PIXEL AT X: 0 Y: k TO OUT1
  COUNT WITH i FROM 0 TO 7 BY 1
  DO
    SEND PIXEL AT X: i Y: k TO OUT1
    WAIT 0.5 seconds
```

Change the LittleBits setup to this:



#6

This code is very similar to #5, but instead of the color being determined in the code, it is determined by converting the signal from IN1 to a color.



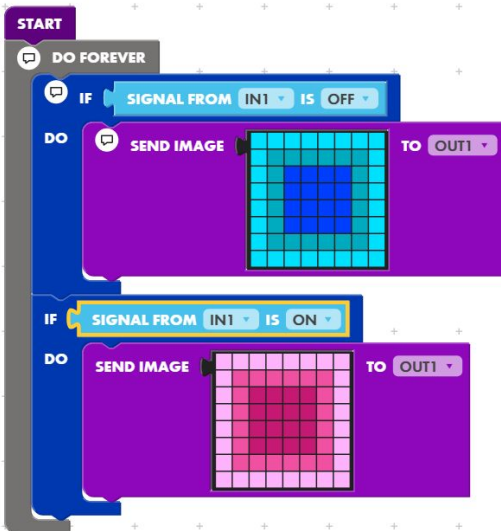
```
START
COUNT WITH k FROM 0 TO 7 BY 1
DO
SEND CONVERT IN1 TO A COLOR PIXEL AT X: 0 Y: k TO OUT1
COUNT WITH i FROM 0 TO 7 BY 1
DO
SEND CONVERT IN1 TO A COLOR PIXEL AT X: i Y: k TO OUT1
WAIT 0.5 seconds
```

Here's another way to pick your color (pressure sensor is not necessary for this program):



```
START
COUNT WITH k FROM 0 TO 7 BY 1
DO
SEND COLOR RED % 200 GREEN % 0 BLUE % 200 PIXEL AT X: 0 Y: k TO OUT1
COUNT WITH i FROM 0 TO 7 BY 1
DO
SEND COLOR RED % 200 GREEN % 0 BLUE % 200 PIXEL AT X: i Y: k TO OUT1
WAIT 0.5 seconds
```

#7



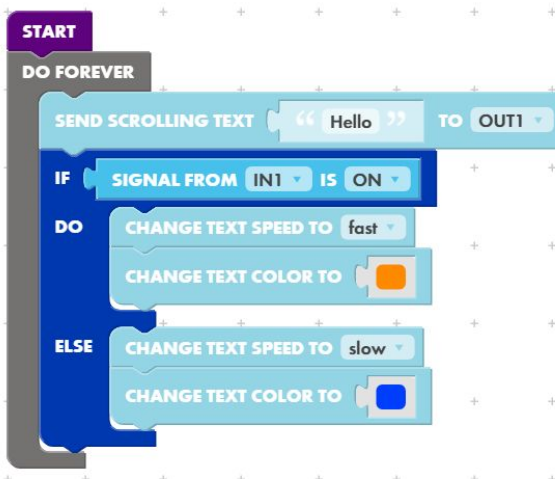
If the signal is OFF (no pressure to the sensor) then the **BLUE** pattern will appear. If the signal is ON then the **PINK** pattern will appear. The whole program is wrapped in a forever block so the program is constantly updating, rather than just running once when you press UPLOAD.



If the signal from IN1 (controlled by the pressure sensor) rounds to 1, the **BLUE** pattern will appear. If it rounds to 2, the **YELLOW** pattern will appear. If it rounds to 3, the **PINK** pattern will appear.

You need to include the green round block around the signal block, because the signal is unlikely to be a whole number.

#8



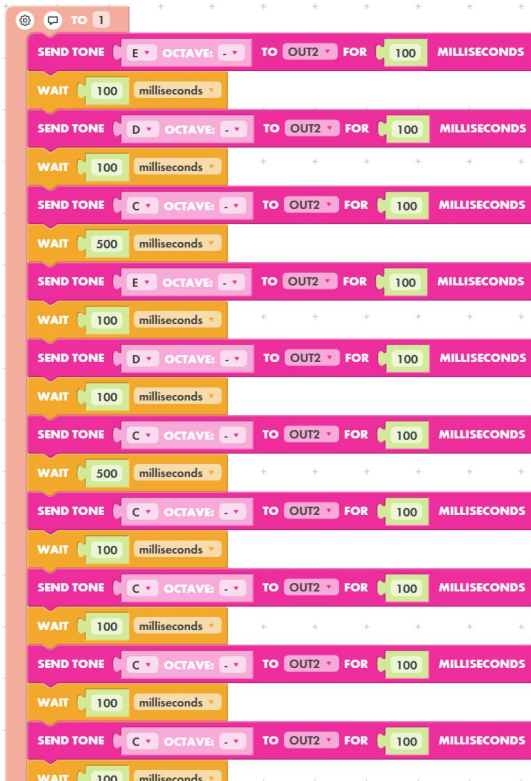
This code makes the word “hello” scroll across the LED matrix forever.

IF the sensor is pressed, it appears in **ORANGE** and scrolls fast.

Otherwise (ELSE), it appears in **BLUE** and scrolls slow.

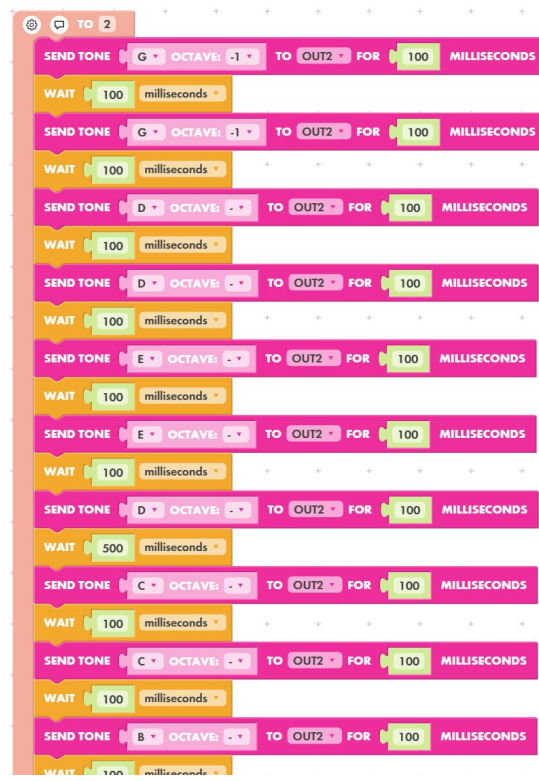
For this example, replace the LED matrix with the 026 speaker.

#9



TO 1

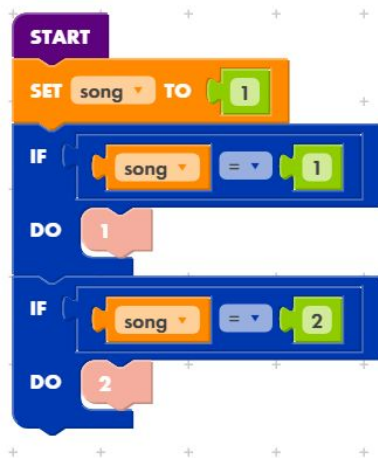
```
SEND TONE E OCTAVE: - TO OUT2 FOR 100 MILLISECONDS
WAIT 100 milliseconds
SEND TONE D OCTAVE: - TO OUT2 FOR 100 MILLISECONDS
WAIT 100 milliseconds
SEND TONE C OCTAVE: - TO OUT2 FOR 100 MILLISECONDS
WAIT 500 milliseconds
SEND TONE E OCTAVE: - TO OUT2 FOR 100 MILLISECONDS
WAIT 100 milliseconds
SEND TONE D OCTAVE: - TO OUT2 FOR 100 MILLISECONDS
WAIT 100 milliseconds
SEND TONE C OCTAVE: - TO OUT2 FOR 100 MILLISECONDS
WAIT 500 milliseconds
SEND TONE C OCTAVE: - TO OUT2 FOR 100 MILLISECONDS
WAIT 100 milliseconds
SEND TONE C OCTAVE: - TO OUT2 FOR 100 MILLISECONDS
WAIT 100 milliseconds
SEND TONE C OCTAVE: - TO OUT2 FOR 100 MILLISECONDS
WAIT 100 milliseconds
SEND TONE C OCTAVE: - TO OUT2 FOR 100 MILLISECONDS
WAIT 100 milliseconds
```



TO 2

```
SEND TONE G OCTAVE: -1 TO OUT2 FOR 100 MILLISECONDS
WAIT 100 milliseconds
SEND TONE G OCTAVE: -1 TO OUT2 FOR 100 MILLISECONDS
WAIT 100 milliseconds
SEND TONE D OCTAVE: - TO OUT2 FOR 100 MILLISECONDS
WAIT 100 milliseconds
SEND TONE D OCTAVE: - TO OUT2 FOR 100 MILLISECONDS
WAIT 100 milliseconds
SEND TONE E OCTAVE: - TO OUT2 FOR 100 MILLISECONDS
WAIT 100 milliseconds
SEND TONE E OCTAVE: - TO OUT2 FOR 100 MILLISECONDS
WAIT 100 milliseconds
SEND TONE D OCTAVE: - TO OUT2 FOR 100 MILLISECONDS
WAIT 500 milliseconds
SEND TONE C OCTAVE: - TO OUT2 FOR 100 MILLISECONDS
WAIT 100 milliseconds
SEND TONE C OCTAVE: - TO OUT2 FOR 100 MILLISECONDS
WAIT 100 milliseconds
SEND TONE C OCTAVE: - TO OUT2 FOR 100 MILLISECONDS
WAIT 100 milliseconds
SEND TONE B OCTAVE: - TO OUT2 FOR 100 MILLISECONDS
WAIT 100 milliseconds
```

These two songs are very long blocks of code, so you can put them into functions and simply refer to them as “1” and “2”. Putting them in the IF loops (see next page) would make the code appear much more complicated than it is!



```
START
SET song TO 1
IF song = 1 DO 1
IF song = 2 DO 2
```

Functions are a great way to make your code more clean and understandable. This code calls a different function depending on what number you chose to represent the variable “song”.