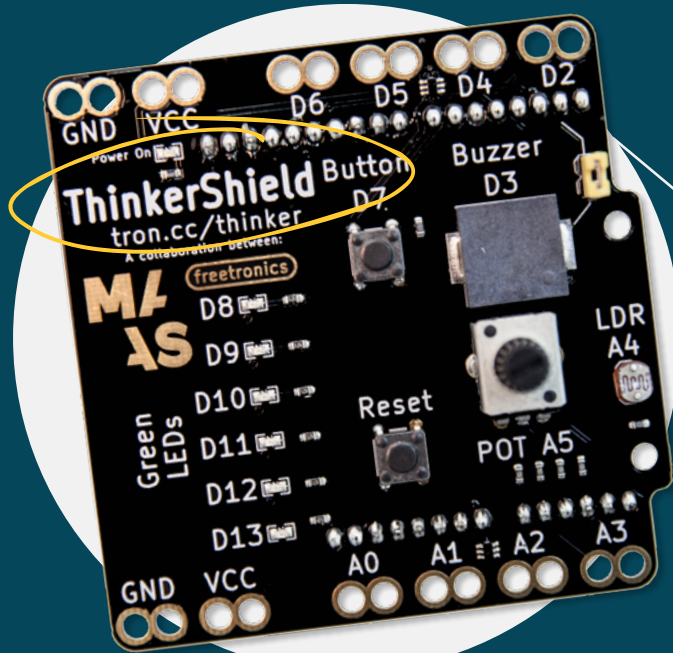
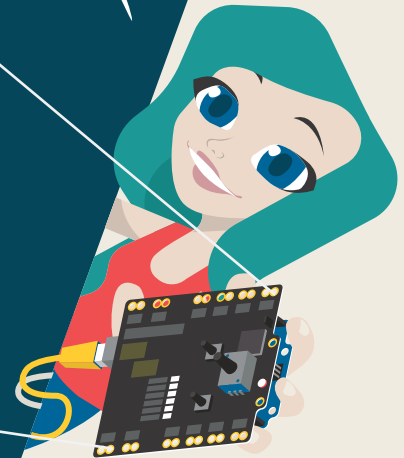


# GET.(ON).WITH.IT

ThinkerShield for Arduino *quick-start* guide



18 easy to follow  
physical computing  
activities for the  
Thinkershield.  
Get on with it!



Peter Mahony / James Oliver



## Acknowledgements

Making and documenting something like the ThinkerShield requires input from lots of clever people — probably too many to thank. We would however like to acknowledge some key individuals who made major contributions to the project: John Hirsch, Arturo Rivillo, Robbie Mudrazija, Joy Suliman, Samuel Bruce, Mark Scarcella, Curtis Black, Lucy McGinley, Fil Bartkoviak and Craig Browne. Thank you guys! Thanks also to the countless young enthusiastic participants who helped us test the product in our Thinkspace workshops.

We would also like to acknowledge the creative electronics expertise and support provided by the team over at **Freerionics**: Jonathan Oxer, Marc Alexander and Angus Gratton.

While care has been taken in the preparation of this book, the MAAS and the authors assume no liability for errors or omissions, or for damages or loss resulting from the use of the information contained in its contents. In addition no responsibility is taken for any information or services which may appear on any linked websites.

You may reproduce this document for your personal and educational use. You may also reproduce and share your projects and code that include information contained in this book provided that you acknowledge this book as the source. You may not however imply that the MAAS endorses you or your use of the material without expressed written permission.

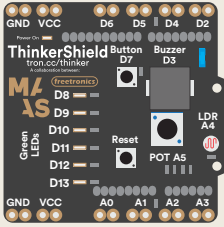
The Museum of Applied Arts and Sciences is an executive agency of the NSW Government.

©2015-2016 Copyright Museum of Applied Arts and Sciences. 'Arduino' is a brand of Arduino LLC.

# Contents

Introduction . . . . .	5	Push.Button.Basics . . . . .	30
Get.Connected . . . . .	6	Get.A.Toggle.On . . . . .	32
What's.On.Board?. . . . .	9	Buzzer.Basics . . . . .	34
In.A.Blink . . . . .	10	Pitch.Changer . . . . .	36
What's in an Arduino program? . . .	12	Play.A.Song . . . . .	38
Getting.Flashy . . . . .	14	Electronic.Dice . . . . .	40
Even.Flashier . . . . .	16	Buzzing.Light.Meter . . . . .	44
Pot.Basics . . . . .	18	LED.Magic.Sign. . . . .	48
A.Light.Dimmer. . . . .	20	Connect.A.LED . . . . .	52
Decode the light dimmer code . . .	22	Traffic.Lights . . . . .	54
POT.LED.Bouncer . . . . .	24	Digital.Spoon.Piano . . . . .	56
LDR.Basics . . . . .	26	What.Next? . . . . .	60
LDR.Night.Light . . . . .	28		

# All you need:\*



## ThinkerShield board

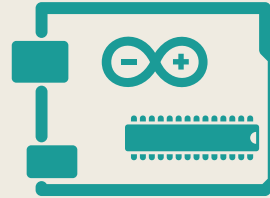
A quick-start board made up of a unique blend of on-board components to make it easy to start doing computer controlled activities. It can also be expanded using standard Arduino compatible components.



## Personal computer running Arduino software

Arduino software is free to download and use from:

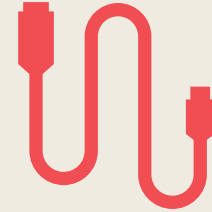
[www.arduino.cc](http://www.arduino.cc)



## Arduino board

Such as:

- Arduino Uno
- Freetronics Eleven
- Genuino Uno
- or any Arduino compatible board that has a standard Arduino UNO header layout.

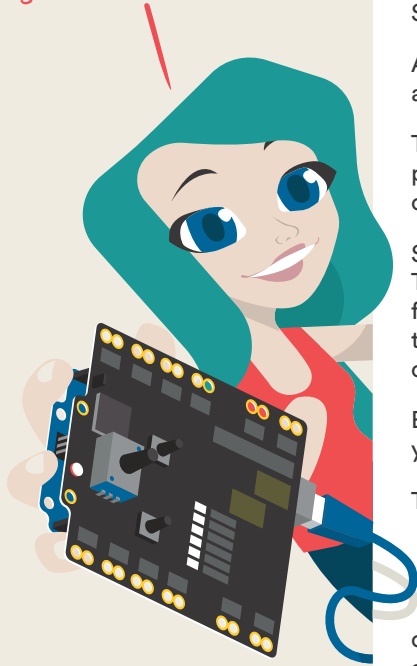


## USB cable

To connect your Arduino board to your computer. This provides power to the board and allows program upload.

\*Well, almost! You will need a few other components for the **Let's.Get.Beyond.It** projects: 3 LEDs (red, green, yellow), 3 resistors (220 ohm) and 9 wires with alligator clips. Get'em from any electronics shop for just a few dollars.

Don't spend too long  
on this page. Just grab  
your Thinkershield and  
Arduino and let's  
get on with it!



## Introduction

The ThinkerShield for Arduino is a learn-by-making product by Thinkspace at the Museum of Applied Arts and Sciences in Sydney, Australia.

At Thinkspace we love learning-by-making. We think everybody should be able to dive in and just start playing and learning right away.

The Arduino board and programming language is a worldwide phenomenon bringing the exciting world of electronics and computer control to anyone with a computer and a USB cable.

So, together with the clever folks at Freetronics, we created the ThinkerShield for Arduino. The ThinkerShield makes it easy for you, your family, your schoolmates and friends to start programming and controlling things with your computer in minutes! No need for any wiring or soldering or program knowledge.

Even if you have never seen a computer program before, we guarantee you will be making things flash, buzz, beep and respond in no time.

Then, when you are ready to move on to bigger things, the ThinkerShield has 8 built-in external connectors that make it easy for you to connect all manner of devices like switches, lights, motors and sensors.

So, don't just watch what's going on in the world of electronics and computing. Take a step to start understanding it. Grab your ThinkerShield and **get.on.with.it!**

# Get.Connected

## 1. Install Arduino

Installing the Arduino software for your computer is usually pretty straightforward.

Just go to the Arduino website and follow the links to the downloads page. Select the latest version for your computer (Windows, Mac or Linux).

[www.arduino.cc](http://www.arduino.cc)

The site covers everything you need to get up and running and to troubleshoot any problems you may have.

## 2. Install drivers

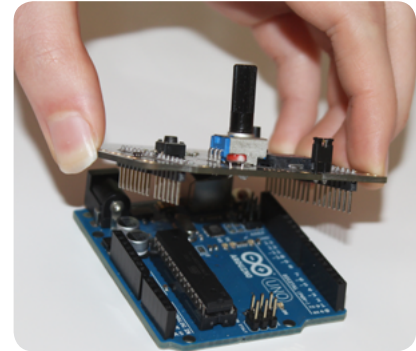
Depending on your computer and operating system, you may need to install a driver for your Arduino board. Again, the best advice we can give you is to follow the steps set out on the Arduino site. Just search 'install drivers' or go straight to:

[www.arduino.cc/en/Guide/Howto](http://www.arduino.cc/en/Guide/Howto)

Of course, if you are using an Arduino compatible board you should install the drivers recommended by the board's manufacturer.

## 3. Connect your ThinkerShield to your Arduino and computer

Carefully align all your ThinkerShield's leg pins with the sockets on your Arduino and press them together.



*Make sure everything is aligned, otherwise you may bend the pins or the sockets.*





## 5. Download ThinkerShield program code files

All activities in this book have associated program code for you to load and complete.

Before you start your Arduino software, download all the program code files from our site:



[ma.as/  
thinkershieldresources](http://ma.as/thinkershieldresources)

Copy the files to a convenient location on your computer. The code files are divided into two folders: *activity* and *completed*. The files in the activity folder are for working through the activities. The completed versions are ready to be uploaded and run at any time.

## 6. Launch!

Launch the Arduino application (`arduino.exe`) you have previously downloaded.

When it opens:

- 1/ go to File/Open...
- 2/ navigate your way to your ThinkerShield folder you just downloaded
- 3/ open the folder called  `ts_In_A_Blink`
- 4/ double-click on the file  `ts_In_A_Blink.ino`
- 5/ Get on with it!

[Turn to the first activity]

## What is Arduino?

Arduino is an open-source electronics prototyping platform based on flexible, easy-to-use hardware and software. It's intended for artists, designers, hobbyists and anyone interested in creating interactive objects or environments.

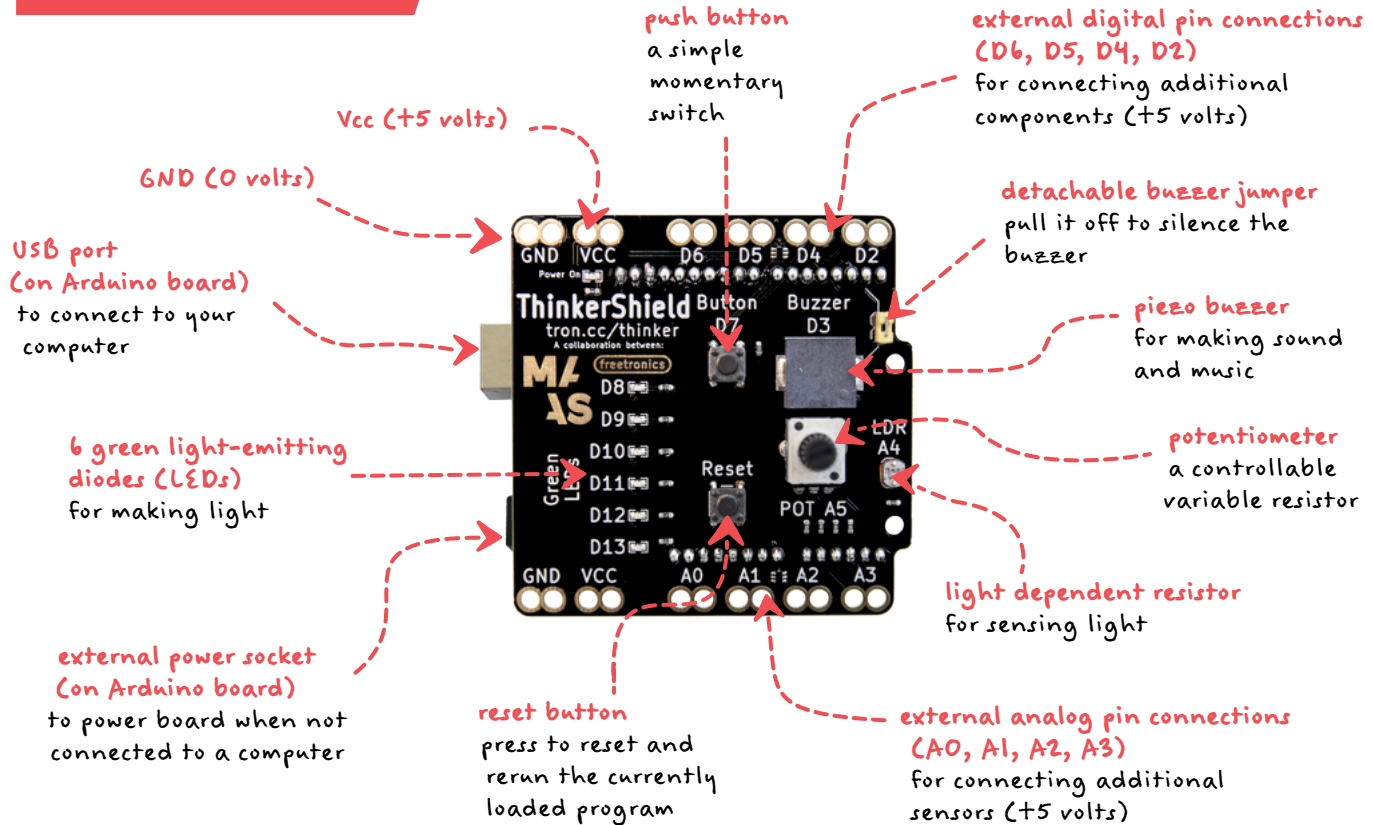
Arduino can sense the environment by receiving input from a variety of sensors and can affect its surroundings by controlling lights, motors and other actuators.

The microcontroller on the board is programmed using the Arduino programming language (based on Wiring) and the Arduino development environment (based on Processing). Arduino projects can be stand-alone or they can communicate with software running on a computer (eg Flash, Processing, MaxMSP).

[www.arduino.cc](http://www.arduino.cc)  
source: Arduino site

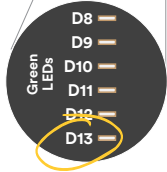
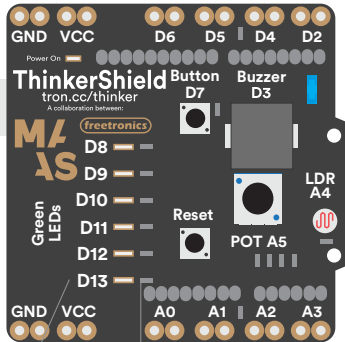


# What's On Board?



# In.A.Blink

Start by flashing a LED



LED on pin D13

We think the best way to make sure your ThinkerShield is connected correctly and everything is working properly is to make it do something — straight away! So let's make one of those cute little LEDs blink!

Connect your ThinkerShield to your Arduino and then to your computer with the USB cable.

Upload the program to your ThinkerShield.

Check that everything works and start experimenting in a blink!



Open a very simple Arduino program that will make the LED on pin D13 flash on and off.

The words 'Compiling sketch' should appear momentarily in the status bar as the green upload bar updates. If everything works, you will see the message 'Done uploading' and the D13 LED will flash twice to let you know it has finished uploading.

## Ingredients



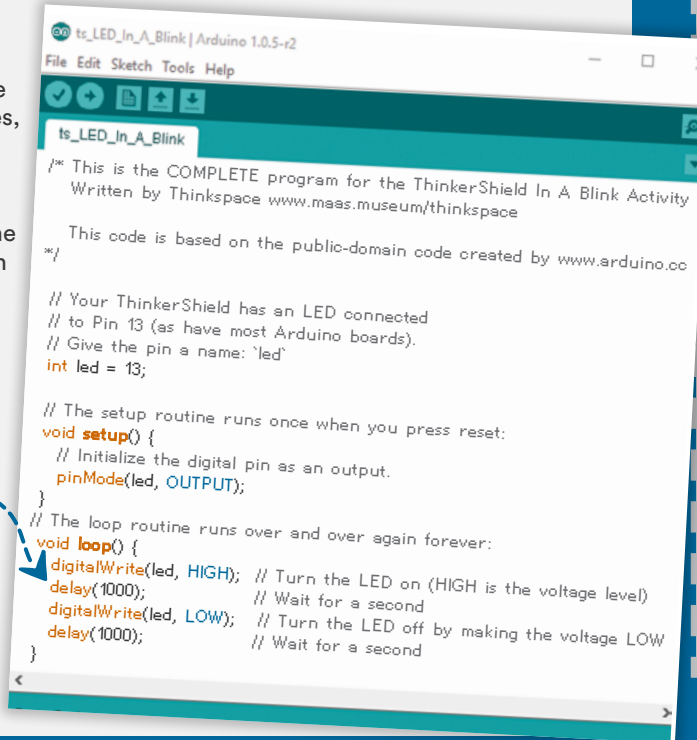
## LET'S.GET.(ON).WITH.IT

- 1/ Set up and connect your ThinkerShield and Arduino software as shown in the previous pages.
- 2/ Open the file  `ts_LED_In_A_Blink.ino`
- 3/ Upload your program by clicking on the  icon.
- 4/ The LED on pin D13 should begin to flash slowly on and off once every second. If it does, everything is working fine (yippee! — move on to step 5). If not check everything in the 'Not working?' column until you solve the problem.

### 5/ Try this:

Look at the program code on your screen and find this section.

Try changing the `delay(1000)` to another number such as `delay(500)` and click upload to make the LED flash at a different rate!



```
ts_LED_In_A_Blink | Arduino 1.0.5-r2
File Edit Sketch Tools Help

ts_LED_In_A_Blink

/* This is the COMPLETE program for the ThinkerShield In A Blink Activity
   Written by Thinkspace www.maas.museum/thinkspace

   This code is based on the public-domain code created by www.arduino.cc
   */

// Your ThinkerShield has an LED connected
// to Pin 13 (as have most Arduino boards).
// Give the pin a name: 'led'
int led = 13;

// The setup routine runs once when you press reset:
void setup() {
  // Initialize the digital pin as an output.
  pinMode(led, OUTPUT);
}

// The loop routine runs over and over again forever:
void loop() {
  digitalWrite(led, HIGH); // Turn the LED on (HIGH is the voltage level)
  delay(1000); // Wait for a second
  digitalWrite(led, LOW); // Turn the LED off by making the voltage LOW
  delay(1000); // Wait for a second
}
```



### Not working?

- ✓ Is the Arduino getting power? If it is connected correctly you should see a tiny green LED next to the word ON. If not, check your USB connection.
- ✓ Is your ThinkerShield connected correctly to your Arduino board? Make sure all legs are nicely aligned and embedded in the black sockets.
- ✓ Have you selected the correct board type from the 'Tools > Board' menu on your Arduino software?
- ✓ Have you selected a serial port from the 'Tools > Serial Port' menu on your Arduino software?
- ✓ Is the program uploading successfully? Do you see the message 'Done uploading'?

### Still not working?

- ✓ Shutdown the Arduino program on your computer and unplug the USB from your computer and go through steps 1 to 3 again.
- ✓ Go to: [www.arduino.cc](http://www.arduino.cc) click on Forum and look for troubleshooting.

```
ts_in_A_Blink | Arduino 1.0.5-r2
File Edit Sketch Tools Help
ts_in_A_Blink

/* This is the COMPLETE program for the ThinkerShield In A Blink Activity
   Written by Thinkspace www.maas.museum/thinkspace
   This code is based on the public-domain code created by www.arduino.cc
*/

// Your ThinkerShield has an LED connected
// to Pin 13 (as have most Arduino boards).
// Give the pin a name: 'led'
int led = 13;

// The setup routine runs once when you press reset:
void setup() {
  // Initialize the digital pin as an output.
  pinMode(led, OUTPUT);
}

// The loop routine runs over and over again forever:
void loop() {
  digitalWrite(led, HIGH); // Turn the LED on (HIGH is the voltage level)
  delay(1000); // Wait for a second
  digitalWrite(led, LOW); // Turn the LED off by making the voltage LOW
  delay(1000); // Wait for a second
}

Done Saving
Binary sketch size: 1,004 bytes (of a 32,256 byte maximum)
24
Arduino Uno on COM4
```

**Comments:** appear in grey. They are explanatory notes used to tell yourself (and other programmers) what the next bit of code is supposed to do or where it is from. Single line comments begin with `//`. Blocks of comments are enclosed between `/* */`

**keyWords:** These words have particular meaning in the Arduino language. They always appear in orange.

**void setup ( ) {...}** This runs once after the program is uploaded and run — you will always use this to setup pinModes.

**void loop ( ) {...}** This section of the program runs continuously after the setup ( ) function has run.

**Semicolon:** Each line of code ends with a ;

**{ }** Curly braces: Your code must be enclosed between curly braces.

**Constants:** Are predefined expressions in the Arduino language (like **HIGH**, **LOW**, **INPUT**, **OUTPUT**, **true**, **false**). They will appear in blue or orange.

**Debugger area:** Any errors will be displayed here.

**Status bar:** Displays the current board type and which COM port is being used.

## What's in an Arduino program?

There are zillions of Arduino based programs all over the web that you can download, try out and try to understand. And you can be pretty sure that they will all have the basic elements and structure shown here on these pages.



**Sketch:** Arduino people like to call their programs 'Sketches'.

**Upload program:** Click this icon to transfer your program to the ThinkersShield. This little arrow will quickly become your favourite button!

**Verify program:** Check your program for errors before uploading. You probably won't need to verify before uploading programs in this book, but it might help as you create larger programs.

**Serial monitor:** Open the serial monitor for watching serial communication.



Arduino software is free and allows you to program the micro-controller on your ThinkersShield. It doesn't get easier or better than that!

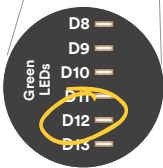
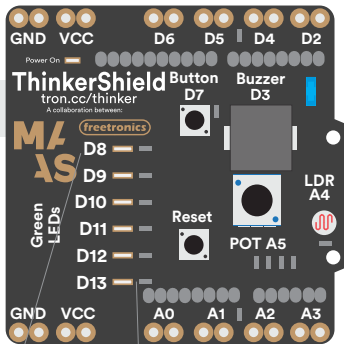
**Need help, or just want to start exploring on your own?**

Whatever your question, chances are the answer is waiting for you on the official Arduino website. It's easy to follow and filled with solutions, forums and ideas for everything Arduino!

[www.arduino.cc](http://www.arduino.cc)

# Getting Flashy

## Controlling a LED



Light-emitting diode (LED) on digital pin D12

Now let's start talking to your ThinkerShield and take control of those LEDs so you can make any of them flash how you want them to!

We first create a variable that we can use to hold the number of the digital pin of the LED we want to control. We'll start with pin 12.

Then we use a function called `pinMode()` to set the pin as an OUTPUT. Otherwise it will default to an input.

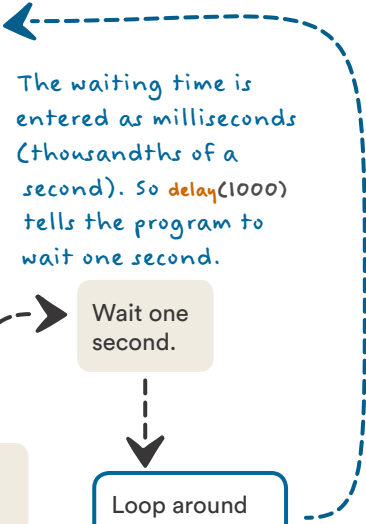
Next we do what's called a `digitalWrite` to the `ledPin` and send it HIGH to turn it on.

Wait one second.

Do another `digitalWrite` to the `ledPin` and send it LOW to turn it off.

Wait one second.

Loop around forever.



### Ingredients



## LET'S.GET.(ON).WITH.IT

1/ Open the file  `ts_LED_Getting_Flashy.ino`

2/ Look for the comment `// Setup LED variables`

```
// Setup LED variables.  
// The setup routine runs once when you press reset:  
void setup()
```

and insert the following line of code:

```
int ledPin = 12;
```

This creates a variable called 'ledPin' that we can use to refer to the LED on pin 12.

3/ Find the comment `// Set the LED pin as an output`

```
// Set the LED pin as an output.  
  
}
```

and insert the following line of code:

```
pinMode(ledPin, OUTPUT);
```

This gets the LED on pin 12 ready for flashing.


4/ Look for the comment `// Turn the LED on and off`

```
{  
  // Turn the LED on and off.  
  
}
```

and add the following 4 lines of code:

```
digitalWrite(ledPin, HIGH); // LED on  
delay(1000); // Wait 1 sec  
digitalWrite(ledPin, LOW); // LED off  
delay(1000); // Wait 1 sec
```

Turn the LED on by setting the voltage HIGH. Wait, then set the voltage LOW (and the LED goes off).

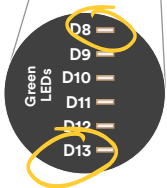
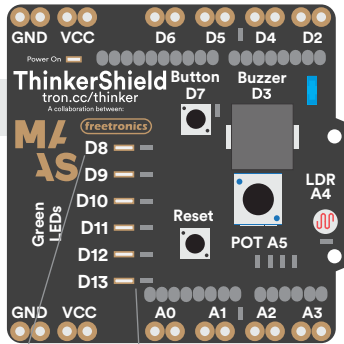
 5/ Upload your program. The LED on pin 12 should flash on and off each second.

I know this is pretty much the same as the first activity, but now you know how to write the code to initialise a digital pin and set it as an output using `pinMode`. Try changing the ledpin number!



## Even.Flashier

Controlling multiple LEDs at the same time



Light-emitting diodes (LEDs) on digital pins D8 and D13

Your ThinkerShield has six LEDs (on digital pins D8-D13). With some simple changes to your program code you can easily control them — one at a time, all at once or in patterns and sequences. Let's start with two of them!

Set up variables for each of the digital pins for each of the LEDs you will be flashing. We chose pins 8 and 13.

`void setup()`  
Then, in `void setup()`, use `pinMode` to set each LED pin as an OUTPUT.

`void loop()`  
Next, in `void loop()`, do various patterns of `digitalWrite`s to send each LED high or low according to the on/off pattern you want to create.

Get flashier and flashier and flashier!

### Ingredients





## LET'S.GET.(ON).WITH.IT

1/ Open the file `ts_LED_Even_Flashier.ino`

2/ Look for the comment `// Setup LED variables`

```
// Setup LED variables  
int led1Pin = 13;
```

and insert the following code:

```
int led2Pin = 8;
```

This will define our second LED pin.

3/ Find the comment `// Set LED pins as digital outputs`

```
// Set LED pins as digital outputs  
pinMode(led1Pin, OUTPUT);  
}
```

and insert the following code:

```
pinMode(led2Pin, OUTPUT);
```

Our second LED is now ready for use as an output.

4/ Look for the comment `// Turn the LEDs on and off`

```
// Turn the LEDs on and off  
  
}
```

and add the following code:

```
digitalWrite(led1Pin, HIGH); // LED1 on  
digitalWrite(led2Pin, LOW); // LED2 off  
delay(500);  
digitalWrite(led1Pin, LOW); // LED1 off  
digitalWrite(led2Pin, HIGH); // LED2 on  
delay(500);
```

Turn LED1 on and LED2 off (it won't matter if it's already off).

Do the opposite.

5/ Upload your program. The LEDs on pins D13 and D8 should flash on and off every half second.

PICK A PATH

Page 48

Make a LED magic sign

Next Page

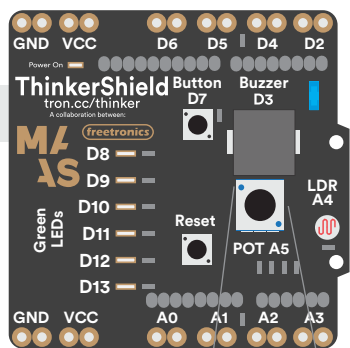
Learn to use the potentiometer

I added a third LED to my program. Then I re-arranged the code so that all the LEDs were blinking together and with different patterns and different delays!



# Pot.Basics

## Reading the potentiometer



Potentiometer on analog pin A5



Learn to read the ThinkerShield's potentiometer (a variable resistor) and how to print the values to the serial monitor so you can see them on your computer screen.

Create a variable to hold the potPin value.

Send the value to the serial monitor.

Then find out how the values change as you turn the potentiometer back and forth.

A potentiometer is an analog device so we use the `analogRead()` function to get a value from it.

Turn on the serial port with `Serial.begin` so we can see the values from the potentiometer on analog pin A5.

Analog devices like potentiometers (or 'pots') are often used as faders on lights.



### Ingredients



# LET'S.GET.(ON).WITH.IT

- 1/ Open the file  `ts_Pot_Basics.ino`
- 2/ Find the comment `// Define potentiometer pin`

```
// Define potentiometer pin
```

Create a variable for analog pot pin A5.

and insert the following code:

```
int potPin = A5;
```

- 3/ Find the comment `// Turn the serial port on`

```
{  
  // Turn the serial port on.
```

Activate the serial monitor so you can use it to see values on your computer screen.

and insert the following code:

```
Serial.begin(115200);
```

- 4/ Locate the comment `// Read the input pin`

```
// Read the input pin
```

This will put the value of the potentiometer into a variable called 'potValue'. It will return values between 0 and 1023.

and insert the following code:

```
potValue = analogRead(potPin);
```

- 5/ Lastly, find the comment `// Print the contents of ...`

```
// Print the contents of the variable to the serial monitor.  
  
// Short delay before reading again.
```

This line prints the value to the serial monitor so you can see it. Once the program is running, click this icon!

and insert this line of code:

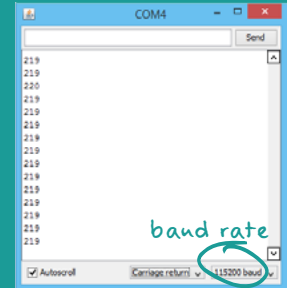
```
Serial.println(potValue);
```

- 6/ Upload your program and open the serial monitor. Turn the pot back and forth and see how the values change!



See also:  
cereal monster

The serial monitor lets you see the values coming from your components.

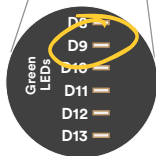
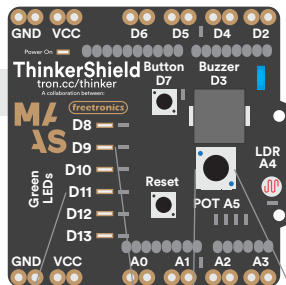


## Not working?

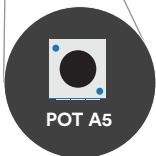
- Check your code for missing ; or ( ) or . or { }
- Make sure potPin = 5 (not some other number).
- Do the baud rates in the code and on the serial monitor match?

# A.Light.Dimmer

Make a light dimmer control



Light-emitting diode (LED) on digital pin D9



Potentiometer on analog pin A5

Now that we know how to read the values from the potentiometer we can use it to control the brightness of the ThinkerShield's LEDs — just like a light dimmer.

We start by initialising one of the LED pins that supports a technique called PWM. On ThinkerShield these are pins 9, 10 and 11. This will be our light.

Do an `analogWrite( )` to the LED pin. This gives us more control than just a `digitalWrite( )` which can only switch between high and low.

Then we insert a line of code that uses the `map( )` function to convert values coming from the (analog) potentiometer so we can send them to the (digital) LED.

Experiment with different potentiometer positions to see the effect on the LED brightness and create some romantic mood lighting!

## Ingredients



## LET'S.GET.(ON).WITH.IT

1/ Open the file `ts_Pot_Light_Dimmer.ino`

2/ Look for the comment // Define pins

```
// Define pins.
```

```
int potPin = A5;
```

This creates a variable for digital LED pin D9.

and insert the following code:

```
int ledPin = 9;
```

3/ Look for the comment // Map the function as PWM supports 0 to 255 not 0 to 1023

```
// Map the function as PWM supports 0 to 255 not 0 to 1023.
```

```
// Write the value to the LED.
```

This will convert our potentiometer values, which have a range of 0-1023 to PWM friendly values in the range of 0-255.

and insert the following:

```
value = map(value, 0, 1023, 0, 255);
```

4/ Find the comment // Write the value to the LED

```
// Write the value to the LED.
```

```
// Short delay before reading again.
```

This will send our value to our selected LED pin.

and insert the following code:

```
analogWrite(ledPin, value);
```

5/ Upload your program.

6/ Try turning the potentiometer. You should see the brightness of the LED change in response.

Be careful it doesn't get too romantic! 😊



The light dimmer is using something called Pulse Width Modulation (or PWM) to send varying amounts of power to the LED — which changes its brightness. PWM works by switching the power on and off very fast. The longer it is on compared to off, the higher the amount of power supplied (and the brighter your LED!)

On the ThinkerShield you can do PWM on digital pins 3, 5, 6, 9, 10, 11. And we have already have a LED on 9, 10 and 11.

PICK A PATH

Page 26

Jump straight to the LDR

Next Page

Gimme more POT stuff

# Let's decode the Light Dimmer Code

```
ts_Pot_Light_Dimmer

/* This is the COMPLETE program
for the ThinkerShield Pot Light Dimmer Activity
Written by Thinkspace www.meas.museum/thinkspace
*/

// ThinkerShield has a potentiometer connected to analog pin 5
// Pins that support PWM are 3,5,6,9,10,11. We already have LEDs
// permanently connected to pins 9,10,11.

// Define pins.
int potPin = A5;
int ledPin = 9;

// Create a variable to hold the data we read in.
int value = 0;

void setup()
{
  // Turn the serial port on.
  Serial.begin(115200);

  // Initialize the LED pin as a digital output.
  pinMode(ledPin, OUTPUT);
}
```

`int ledPin = 9;`  
Declares a variable called 'ledPin' and assigns it the value '9' which corresponds to the number of the pin that our LED is connected to. We did a similar thing for the potPin but we used A5 so we know we are referring to an analog pin.

`void setup( ) { ... }`  
Set up the serial port and the baud rate so we can send data to the serial monitor.

By default all pins are set as **INPUT**. So, we need to set the **pinMode** for the ledPin as **OUTPUT**. The potPin will default to input as we need it to be.

In programming, a variable is a value that can change, depending on conditions or on information passed to the program. You will use variables all the time in your code to make it easier to do things.



continued on opposite page

Everything between the { } after the `void loop()` function runs repeatedly until the program is stopped.

continued...

```
void loop()
{
  // Read the input pin.
  value = analogRead(potPin);

  // Print the contents of the variable to the serial monitor.
  Serial.println(value);

  // Map the function as PWM supports 0 to 255 not 0 to 1023.
  value = map(value, 0, 1023, 0, 255);

  // Write the value to the LED.
  analogWrite(ledPin, value);

  // Short delay before reading again.
  delay(25);
}
```

`value = analogRead(potPin);`  
This is a very common command used to read the current value on an analog pin and store it in the variable called 'value'.

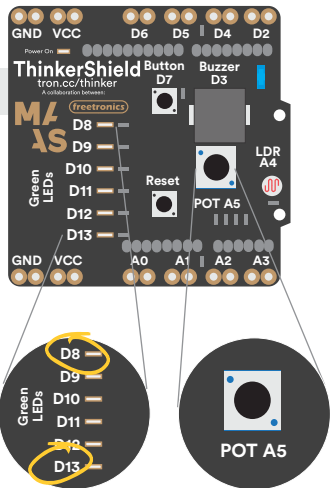
`value = map(value, 0, 1023, 0, 255);`  
The map function makes it easy to scale, or map, one range of values to another range. In this case, it converts from the potentiometer's values that range from a lowest value of 0 and highest value of 1023 to a range from a lowest of 0 to a highest of 255. So, if the value from the pot was 1023 it will be mapped to 255. If it was say 512 (half-way between 0 and 1023) it will be mapped to a value of 127 (half-way between 0 and 255).

`delay(25);`  
This literally means do nothing for 25 milliseconds. It stops us getting zillions of readings.

`analogWrite(ledPin, value);`  
The opposite of an `analogRead`, this writes a value to the analog pin. Because this pin supports PWM it means that we can send different amounts of power to the LED.

## POT.LED.Bouncer

Using analog values to switch LEDs on and off



Light-emitting diodes (LEDs) on digital pins D8 and D13

Potentiometer on analog pin A5

With just a little bit of extra code you can turn the potentiometer into a switch-type control and use it to bounce between two of the ThinkerShield's LEDs.

Start by initialising two LED pins as outputs. We picked pins D8 and D13.

Now, test the value to see if it is either 1023 or 0. These values tell us that the pot is turned all the way to the left or all the way to the right.

If it is one of these values we use `digitalWrite()` to change the state of the two LEDs ... the one that's on gets turned off and the one that's off gets turned on.

Read the value coming from the pot on pin A5 and store the value in a variable.

Wait a moment and loop around again.

So, when you turn the knob all the way from one side to the other, the LEDs bounce back and forth!

### Ingredients





# LET'S.GET.(ON).WITH.IT

1/ Open the file `ts_Pot_LED_Bouncer.ino`

2/ In the `void setup` section, find the comment `// Initialise the LED pins as digital outputs`

```
// Initialise the LED pins as digital outputs.  
[ ]  
// Turn the serial port on.
```

and insert these two lines of code:

```
pinMode(led1Pin, OUTPUT);  
pinMode(led2Pin, OUTPUT);
```

Set the LED pins  
as OUTPUTS

3/ Find the comment `// Read the input pin`

```
// Read the input pin.  
[ ]  
// Print the contents of the variable to the serial monitor.
```

and insert the following code underneath:

```
value = analogRead(potPin);
```

Use `analogRead()`  
to get the current  
value from the  
pot and store it  
in 'value'.

4/ Find the comment `// Use if statement to turn LEDs on and off`

```
// Use if statement to turn LEDs on and off.  
[ ]  
// Short delay before reading again.
```

and insert these ten lines of code:

5/ Upload your program.

6/ Try turning the potentiometer from left to right and see if the LEDs bounce back and forth.

7/ If you turn on the serial monitor you will be able to see the values as they change!

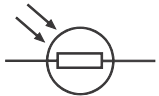
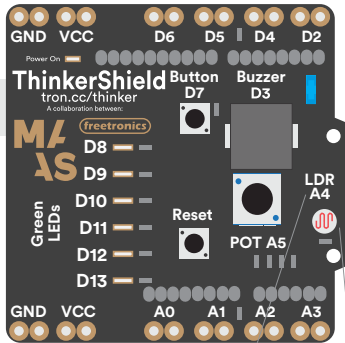
Test to see if the value equals 1023. If it does, do the first bit (led1 on, led2 off). If not, test to see if it equals zero. If it does, it does the second bit. If it doesn't equal either 1023 or 0 don't change anything.

```
if(value == 1023)  
{  
  digitalWrite(led1Pin, HIGH);  
  digitalWrite(led2Pin, LOW);  
}  
else if(value == 0)  
{  
  digitalWrite(led1Pin, LOW);  
  digitalWrite(led2Pin, HIGH);  
}
```

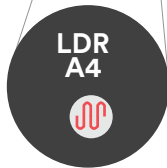
`==` means 'is equal to'

# LDR.Basics

Read the LDR to sense light



Light dependent resistor (LDR) on analog pin A4



Learn to read the values of the ThinkerShield's light sensor (called a **light dependent resistor**) and see how it responds to different light conditions.

Turn on the serial port with `Serial.begin` so we can see the values from the sensor.

Send the value to the serial port.

The LDR is an analog device so we use the `analogRead()` function to get a value from it.

Then have some fun testing with different amounts of light on the sensor.

## Ingredients



# LET'S.GET.(ON).WITH.IT

1/ Open the file `ts_LDR_Basics.ino`

2/ Find the comment // Turn the serial port on

```
// Turn the serial port on.  
}  
}
```

This will activate the serial monitor so you can use it to read values.

and insert the following code:

```
Serial.begin(115200);
```

3/ Find // Read the input pin

```
// Read the input pin.  
}
```

This will put the value of the light sensor into a variable called 'LDRvalue'. It will return values between 0 and 1023.

and insert the following code:

```
LDRvalue = analogRead(LDRPin);
```

4/ Lastly, locate the comment // Print the contents of the variable ...

```
// Print the contents of the variable to the serial monitor.  
}  
// Short delay before reading again.  
delay(25);
```

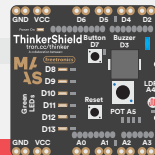
This will print the value to the serial monitor so you can see it.

and insert this line of code:

```
Serial.println(LDRvalue);
```

5/ Upload your program and then open the serial monitor by clicking the magnifying glass icon in the top right corner. Make sure the baud rate is set to 115200.

6/ Expose the light sensor to varying amounts of light and discover how the values change.



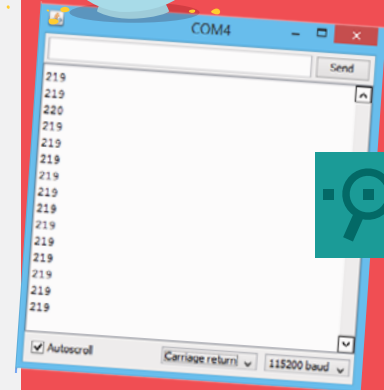
PICK A PATH

Page 36

Use the POT to control the buzzer

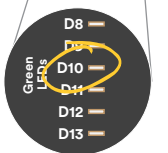
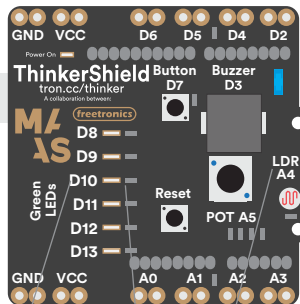
Next Page

More LDR: scare some roaches!



# LDR.Night.Light

Make an automatic night light



Light-emitting diode (LEDs) on digital pin D10

Light dependent resistor (LDR) on analog pin A4

Read the analog values from the LDR and use changes in the values to control the state of the ThinkerShield's LEDs — just like an automatic night light!

First we do a few tests to find out the range of values that your LDR produces — this will always be different depending on the lighting conditions and may vary a tiny bit between ThinkerShields.

We'll need to use the Arduino map() function so we can send the right values to the LED pins.

Then we can use what we find out to decide when it's day and when it's night.



And then we do a digitalWrite to an LED pin to turn it on and off as the values change.



## Ingredients



## LET'S.GET.(ON).WITH.IT

- 1/ Open the file  `ts_LDR_NightLight.ino`
- 2/ Upload your program and then open the serial monitor by clicking on this icon 
- 3/ Try to find the highest and lowest values that the light sensor will produce in the place you are sitting, by exposing it to different amounts of light.

- 4/ Look for the comment // Map the values

```
// Map the values.  
[ ]  
// Use conditional statement to turn LEDs on and off.
```

and insert this line of code:

```
value = map(LDRvalue, 97, 330, 0, 100);
```

Change these two numbers to the low and high values you get from your sensor.

- 5/ Look for the comment // Use conditional statement to turn LEDs on and off

```
// Use conditional statement to turn LEDs on and off.  
[ ]  
// Short delay before reading again.
```

and insert these eight lines of code:

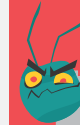
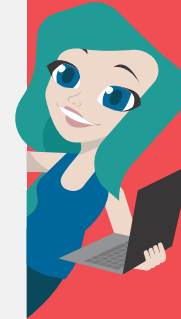
```
if(value < 50)  
{  
  digitalWrite(ledPin, HIGH);  
}  
else  
{  
  digitalWrite(ledPin, LOW);  
}
```

After uploading your program, the LED should switch on when the light sensor detects darkness or low light levels! In our code a low light level is anything that gives a mapped value of <50.

- 6/ Upload again and start scaring some roaches!



You can find the lowest value by covering the LDR with your finger or a piece of thick cardboard. To find the highest value put it in full sunlight or another really bright light.

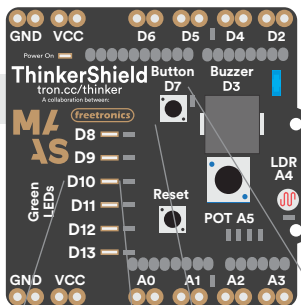


### Serial port errors?

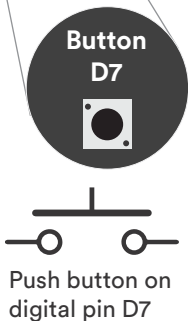
If you ever get errors such as 'Serial Port COM 4 already in use' or 'Serial Port not available', try closing all Arduino programs, unplug the USB from your computer and start everything again.

# Push.Button.Basics

Program the push button as a momentary switch



Light-emitting diodes (LEDs) on digital pins D8-D13



We start by defining and initialising the buttonPin as INPUT and an LED pin as an OUTPUT.

Initialise a boolean variable called buttonState to hold the state of the button. Boolean variables are either true or false.

Next we do a digitalWrite() to set the buttonPin HIGH to make it ready for use (by turning on a pull-up resistor).

Then we just keep reading the state of the buttonPin and act when it changes.

If buttonState is true it means it's being pressed and we use a digitalWrite() to turn on the LED.

more about this on the opposite page

### Ingredients



# LET'S.GET.(ON).WITH.IT

1/ Open the file `ts_PushButton_Basics.ino`

2/ Find the comment `// Variables`

```
// Variables.  
[ ]  
void setup()
```

Set up a boolean (true or false) variable for the buttonState

and insert the following line of code:

```
boolean buttonState = false;
```

3/ Look for the comment `// Initialise button pin as INPUT`

```
// Initialise button pin as INPUT.  
[ ]  
}
```

and insert these 3 lines of code:

```
pinMode(buttonPin, INPUT);  
digitalWrite(buttonPin, HIGH);  
pinMode(ledPin, OUTPUT);
```

4/ In the void loop section, find `// Read button value`

```
// Read button value.  
[ ]  
}
```

➔ and insert this code, then **upload** and run it!

```
buttonState = digitalRead(buttonPin);  
// Turn the LED on and off  
if (buttonState == true)  
{  
  digitalWrite(ledPin, HIGH);  
}  
else  
{  
  digitalWrite(ledPin, LOW);  
}
```

This conditional statement will switch the LED on if the buttonState variable is true and keep it off if it's not.



The electronics in the ThinkerShield are very sensitive to tiny changes in current and could at times give random readings.

So, it is good practice to use a pull-up resistor that will 'pull' the buttonPin to HIGH when it is not receiving any input.

That way we can be sure that our LED will only come on when the button is pressed.

## PICK A PATH

Page 34

Show me how to use the buzzer

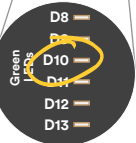
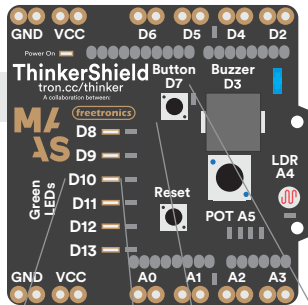
Next Page

More push button: make a toggle switch

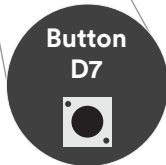
Component: push button

## Get.A.Toggle.On

Make the push button into a toggle switch



Light-emitting diodes (LEDs) on digital pins D8-D13



Push button on digital pin D7

Program the push button to behave as a toggle switch — press once for on, press again for off. And, if you want, you can try some debouncing!

This time we will start by creating a couple of boolean variables to store the current state of the button and of the LED.

If the LED was off, we flip our two boolean variables so that it is turned on (sent HIGH). If it was on, we will invert the variables to turn it off.

Then, when the button is pressed, we check to see what state things were in the last time it was pressed and act accordingly.


Now! My debouncing code really works!

Ingredients





## LET'S.GET.(ON).WITH.IT

- 1/ Open the file  `ts_PushButton_GetAToggleOn.ino`
- 2/ Find the comment `// Boolean (or true/false) variables` and insert these two lines of code:

```
// Boolean (or true/false) variables.  
boolean lastButton = LOW;  
boolean ledOn = false;
```

Boolean variables can be either HIGH/LOW, true/false or even just I/O.

- 3/ In the void loop section, look for the comment `// Read button value` and insert this conditional code:

```
// Read button value.  
  
if (digitalRead(buttonPin) == HIGH && lastButton == LOW)  
{  
  ledOn = !ledOn; // This inverts (switches) the value.  
  lastButton = HIGH; // This stores the last button state.  
}  
else  
{  
  lastButton = digitalRead(buttonPin);  
}
```

- 4/ Have a look at the code that starts with `if (ledOn)`. Can you understand what this bit of code does? Nothing to add, just have a quick read of the code.

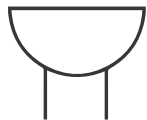
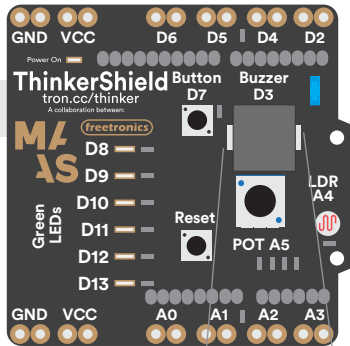
- 5/ Upload and give it a try. Your button should now behave as a toggle switch.

When you run this, you may notice that the button doesn't always behave exactly as expected. Using our button in this way is a little like turning a hose off at the tap — sometimes there is still some electricity that will leak through at the last minute — just like water in a hose. The problem can be solved using a technique called **debouncing**. Maybe you can work out a way to do it. You can also find an example in the: `File\sketchbook\Digital` menu in your Arduino software.

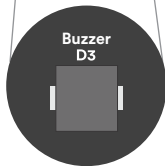


## Buzzer.Basics

Learn to make sounds with the buzzer



Buzzer or piezo speaker on digital pin D3



We've learned how to make some light with the LEDs, let's make some sound with the ThinkerShield's on-board buzzer (a piezo electric speaker).

We initialise the pin that our buzzer is connected to, digital pin D3, and set it as an OUTPUT.

Then we use a nice little function called `tone()` to tell the program what frequency (ie what pitch) to play and how long to play that tone for.

That's it. Simple.



Inside your ThinkerShield's buzzer is a disk made from a special piezo electric material.

When you apply an electrical signal to piezo electric substances they stretch or compress, according to the frequency of that signal. This shape changing produces sound of a corresponding frequency and duration.

### Ingredients



## LET'S.GET.(ON).WITH.IT

- 1/ Open the file `ts_Buzzer_Basics.ino`
- 2/ Find the `// Define pin` comment and define the buzzer pin by adding the following line of code:

```
// Define pin.  
int buzzerPin = 3;
```

- 3/ In the void loop section, add these lines of code:

```
// Play a tone with the buzzer.  
tone(buzzerPin, 659, 20);  
delay(200);
```

- 4/ That's it! Upload and make some sound!  
Experiment with different frequency values.

- 5/ Make music!

We can now start to construct a musical phrase by using different frequencies, note durations and delays.

Experiment with multiple lines of code like these:

```
tone(buzzerPin, 659, 20);  
delay(200);  
tone(buzzerPin, 165, 50);  
delay(100);  
tone(buzzerPin, 400, 80);  
delay(200);
```

You will also see in the code that the `pinMode` is set to `OUTPUT`.

This will tell the buzzer to play a tone with a frequency (or pitch) of 659 Hertz for 20 milliseconds. The delay of 200 milliseconds creates some silence between the notes.

You don't actually need to put a duration. If you don't the sound will keep on playing until you either give it another tone command or use the `noTone(pin)` command.

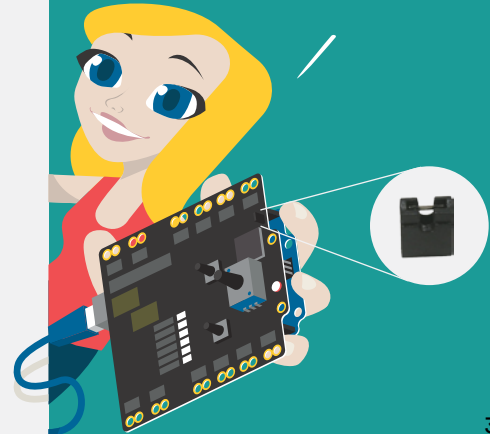
PICK A PATH

Page 38

Jump straight to making music!

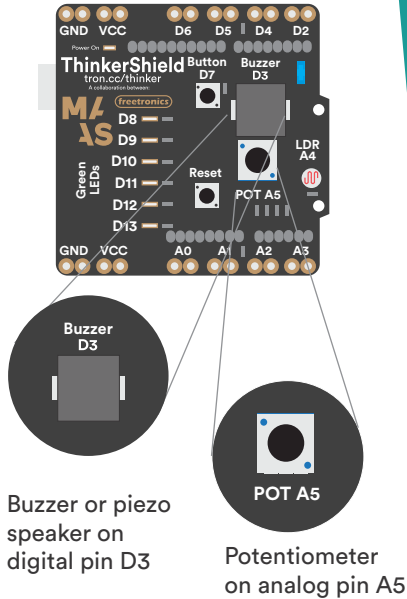
Next Page  
More buzzer: control the frequency

Using the buzzer is fun, but if you (or those around you) need a break from the beeping, you can temporarily slide off the little 'jumper' that normally connects the buzzer to the power.



# Pitch.Changer

Control the buzzer frequency with the potentiometer



Just like we used the potentiometer to vary the power sent to an LED, let's use it to change the pitch (or frequency) produced by the buzzer.

Define our pot and buzzer pins and initialise a couple of variables to hold the values from the pot (potValue) and our mapped frequency (freq).

**void setup()**  
Turn on the serial port so we can see the frequencies the buzzer is producing and set buzzer pinMode to output.

**void loop()**  
Do an analogRead of the potPin and store it in potValue.  
Map the potValue to our desired frequency range of 30-5000Hz.

Use tone() to write the frequency value (freq) to the buzzer and listen to the results.

## Ingredients



Ear muffs?

## LET'S.GET.(ON).WITH.IT

- 1/ Open the file `ts_Buzzer_Pitch_Changer.ino`
- 2/ Find the comment `// Create variables to hold data we read` and insert these lines:

```
// Create variables to hold data we read.  
int potValue = 0;  
int freq = 0;
```

- 3/ In void loop, find the comment `// Read the input pin` and add the following:

```
// Read the input pin.  
potValue = analogRead(potPin);
```

- 4/ Find the comment `// Map the potValue for PWM` and add the following code:

```
// Map the potValue for PWM.  
freq = map(potValue, 0, 1023, 30, 5000);
```

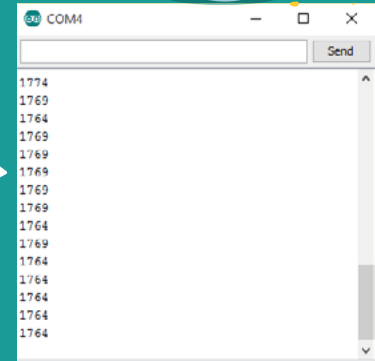
- 5/ Then, under `// Write the value to the buzzer` add the following:

```
// Write the value to the buzzer.  
tone(buzzerPin, freq, 20);
```

- 6/ Upload your program and try changing the pitch of the buzzer by turning the potentiometer back and forth.
- 7/ Try modifying the frequency values in your map function and the delay value in the tone function. Listen to the results!

The map function will take the pot value which will be in the range 0-1023 and convert it to a range of 30-5000 which will give us a good scale of pitch change.

If you turn on the serial monitor you will be able to see the frequency (in Hertz) as you hear it.



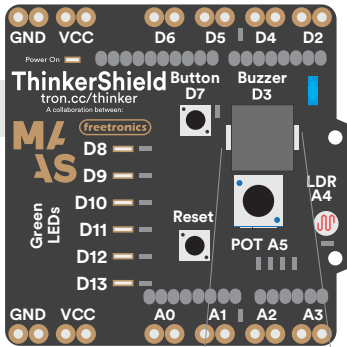
PICK A PATH

Page 44  
Make a  
buzzing  
light meter

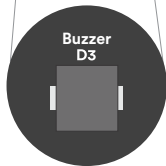
Next Page  
More  
buzzer:  
play a song

# Play.A.Song

Make the buzzer  
make music



Buzzer or piezo  
speaker on  
digital pin D3



Create musical notes and string them together in arrays to make music. Hear your ThinkerShield play your favourite songs!

Use #define to create constants for each of the musical notes we will be using.

Create a special function we call buzzerPlay() that will read the array values and write them to the buzzer in sequence.

Call the buzzerPlay() function, then loop around and play it again.

Set up our pins and pinModes.


Initialise a melody[] array to hold the sequence of notes for our song. Initialise a noteDuration[] array to hold the duration of each of the notes.

Start changing the note and duration array values to play your favourite songs!

## Ingredients



## LET'S.GET.(ON).WITH.IT

- 1/ Open  `ts_Buzzer_Play_A_Song.ino`
- 2/ In this file we have created variables that represent musical notes that will be played by the buzzer. Each note looks like this:  
`#define NOTE_G3 196`
- 3/ Look for the comment `// Notes in the melody`. Below this line is an array (or list) of notes. This is a list of all the notes that go together in a particular order to form a melody. Changing the first note in the list will change the first note of the melody.
- 4/ Look for the comment `// Note durations`:  
`4=quarter note, 8=eighth note ...`  
Below this line is another array which contains the note durations (or lengths). Each of the note lengths in this list corresponds to one of the notes in our melody. Once again changing the first note length in the list will alter the length of the first note in our melody.
- 5/ Upload the program and listen. Do you recognise the tune?
- 6/ Now, try editing each of the arrays in order to change the music. Making changes to the notes array will change the pitch of the notes. Making changes to the durations array will affect the rhythm of the melody.

value name                      value

↙                                      ↘

```
#define NOTE_G3 196
```

We are using a useful technique that let's us give a name to a **CONSTANT** value at the start of the program. We can then use the name instead of the value. This is perfect for storing musical notes.

Take a look at the code that starts with:

```
void buzzerPlay()
```

This is a special function we have included that our program uses (or 'calls') to play the notes and durations in the arrays.

```
NOTE_C4 262
NOTE_D4 294
NOTE_E4 330
NOTE_F4 349
NOTE_G4 392
NOTE_A4 440
NOTE_B4 494
NOTE_C5 523
```



An array is a special type of variable that stores lists of values. Arrays are very useful in programming. You create (or declare) an array with the following syntax:

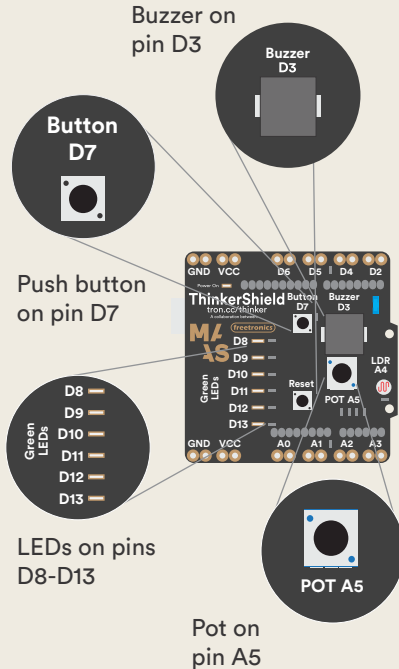
```
type name[] = {item1, item2, ...};
```

Here's the melody array from our code:

```
int melody[] = { NOTE_G3, NOTE_G3,
NOTE_G3, NOTE_DS3, NOTE_AS3,
NOTE_G3, NOTE_DS3, NOTE_AS3,
NOTE_G3, NOTE_D4, NOTE_D4, NOTE_DS4,
NOTE_DS4, NOTE_AS3, NOTE_FS3,
NOTE_DS3, NOTE_AS3, NOTE_G3};
```

# Electronic.Dice

Make a working six light electronic dice



Combine all six LEDs, the buzzer, the pot and the push button to make a cool flashing and beeping variable roll electronic dice to use with your games.

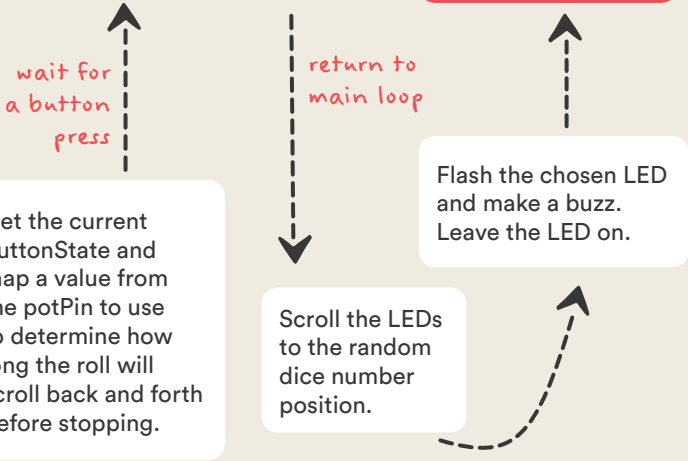
All pin variables are defined. For convenience, LEDs are defined using an array. We also declare a 'float' variable to use with the randomSeed() function.

Randomise the random number generator, set all pinModes are required and start the serial monitor.

Get the current buttonState and map a value from the potPin to use to determine how long the roll will scroll back and forth before stopping.

Jump to the roll() function which will scroll the LEDs back and forth for the selected time and then return a random dice number.

Go back to the top of the loop and wait for the next button press.



### Ingredients





## LET'S.GET.(ON).WITH.IT

- 1/ Open the file `tsproj_Electronic_Dice.ino`
- 2/ Find the comment `// Declare ledPins array` and insert the following line of code:

```
// Declare ledPins array.  
int ledPins[] = {8, 9, 10, 11, 12, 13};
```

- 3/ Now, in void setup find the comment `// Set the seed to be random by reading in an unconnected pin` and insert these two lines of code:

```
// Set the seed to be random by reading in  
// an unconnected pin.  
seed = analogRead(0);  
randomSeed(seed);
```

- 4/ Find the comment `// Set all leds to OUTPUT` by looping through the array immediately underneath your last entry and add the following:

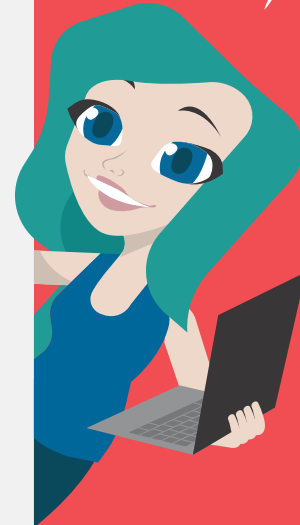
```
// Set all leds to OUTPUT by looping through  
the array.  
for (int i=0; i<6; i++)  
{  
  pinMode(ledPins[i], OUTPUT);  
}
```

By storing the pin values for each of the LEDs in an array, we can easily cycle through them later in the program.

These two lines of code start up and randomise the random number generator in the microprocessor.

Here we use the ledPins array to set the pin mode for all the LEDs in just a few lines of code!

The electronic dice project is totally cool but it does have some pretty advanced code that you may not completely understand straight away. But don't worry, you can still enter it in and play around changing things here and there to try to discover how it works!



5/ Move down into the void loop section, locate the comment // Read in the button and pot values and carefully insert these two lines:

```
// Read in the button and pot values.  
buttonState = digitalRead(buttonPin);  
scrollMaxDelay = map(analogRead(potPin), 0, 1023, 0, 500);
```

6/ Now find the comment // Check the button state and carefully insert this if statement and its opening { curly bracket:

```
// Check the button state.  
if (buttonState == HIGH)  
{
```

Don't do anything until the button is pressed.

7/ Under // Jump to the dice roll function add the following call to the dice roll function:

```
// Jump to the dice roll function.  
diceValue = roll();
```

8/ Finally, scroll all the way to the bottom of the program (past the last curly bracket). Now carefully enter the big block of code that makes up the dice roll function (on the next page)\*

➡ 9/ Upload your program and start rolling some dice!

10/ Try turning the pot to different locations to see how it affects the roll rate.

The map function is being used to convert the current value from the potPin value from its range of 0-1023 to 0-500. This number is then stored in a variable we called scrollMaxDelay.



\* Put this code all the way down at the bottom

```
//Dice roll function.
int roll()
{
  // Clear all leds.
  for (int i=0; i<6; i++)
  {
    digitalWrite(ledPins[i], LOW);
  }
  // Scroll the leds up and down until
  // ScrollMaxDelay is reached.
  while (scrollDelay < scrollMaxDelay)
  {
    for (int i=0; i<6; i++)
    {
      Serial.println(i+1);
      digitalWrite(ledPins[i], HIGH);
      tone(buzzerPin, 100, 20);
      delay(scrollDelay);
      digitalWrite(ledPins[i], LOW);
      scrollDelay += 10;
    }
    for (int i=4; i>0; i--)
    {
      Serial.println(i+1);
      digitalWrite(ledPins[i], HIGH);
      tone(buzzerPin, 100, 20);
      delay(scrollDelay);
      digitalWrite(ledPins[i], LOW);
      scrollDelay += 10;
    }
  }
  // Return a random integer between 0 and 6.
  return random(0,6);
}
```

Loop through the LED array to turn all the LEDs off.

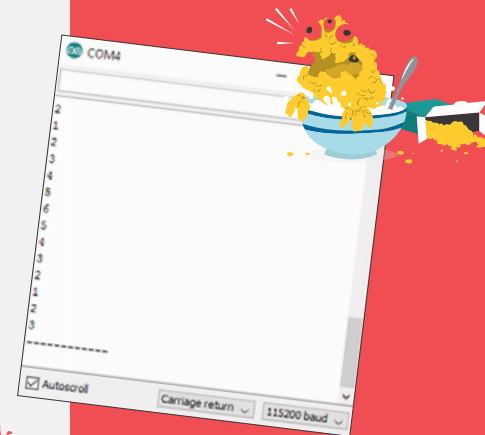
The value of scrollMaxDelay comes from the current position of the pot.

Send dice number to the monitor.  
Turn on the first LED in the array.  
Make a buzz.  
Wait for 'scrollDelay' milliseconds.  
Turn off the same LED.  
+10 to the scrollDelay value.

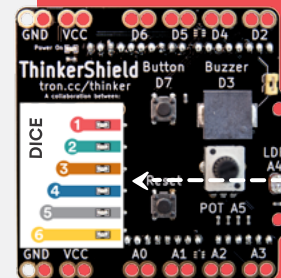
Scroll backwards through LEDs 5,4,3 and 2.

Then, when all the fancy scrolling is done, return a random dice number. Back in the main program the LEDs will slowly scroll to stop on this chosen position.

See the dice value on the serial monitor.

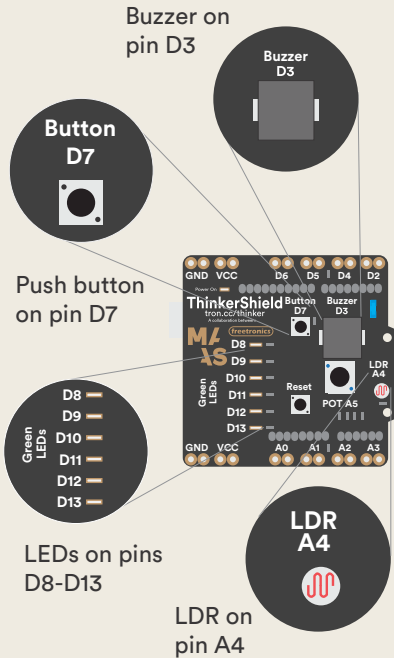


You might like to make a little paper template like this to make it easy to see the dice numbers.



# Buzzing.Light.Meter

Make a device to scan the light levels in your environment



Combine the light sensing capacity of the LDR, all six LEDs and the buzzer to make a great light level meter or scanner.

Begin by defining all required pins: LDR, buzzer and all 6 LEDs. Create 3 variables to hold the raw sensor and mapped values.

Read the LDR to get a light level. Then map that value to a range that we can use for beeping and turning on LEDs.

Use if-then statements to turn on the right number of LEDs and send a tone to the buzzer.

Read the LDR to get a light level. Then map that value to a range that we can use for PWM on the buzzer.

Delay for a time based on the light value.


Initialise the serial port and set all LED pins and the buzzer pin as OUTPUTS.

Turn off the LEDs, send a noTone() to the buzzer, then loop around.

## Ingredients



## LET'S.GET.(ON).WITH.IT

- 1/ Open  `tsproj_Buzzing_Light_Meter.ino`
- 2/ Find the comment `// Define pins` and add the following:

```
// Define pins.  
int LDRPin = A4;  
int buzzerPin = 3;  
int led8 = 8, led9 = 9, led10 = 10;  
int led11 = 11, led12 = 12, led13 = 13;
```

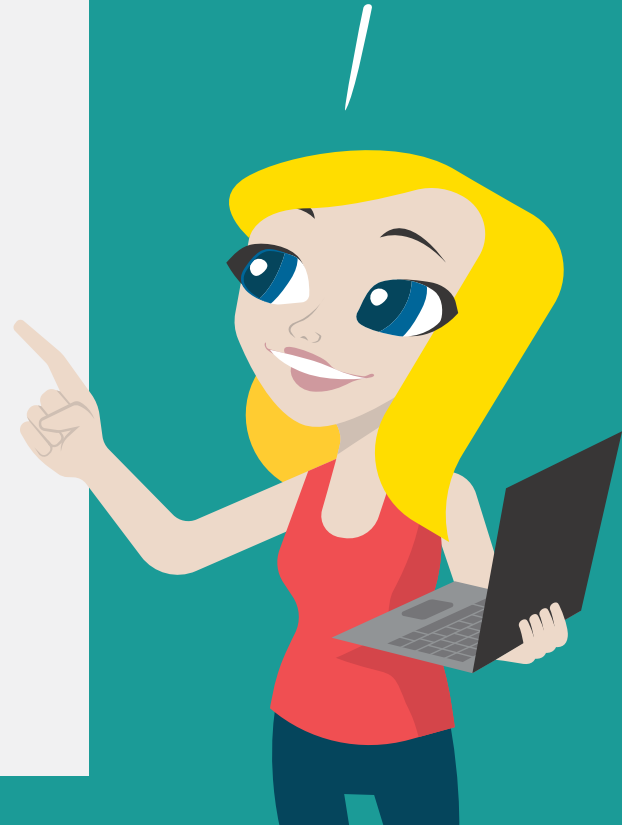
- 3/ Find the comment `// Create variables to hold data` and add the following code:

```
// Create variables to hold data.  
int LDRvalue = 0;  
int value = 0;  
int beepRate = 0;
```

- 4/ Then, under `// Initialise the LEDs` set the LED pinModes by inserting the following six lines:

```
// Initialise the LEDs.  
pinMode(led8, OUTPUT);  
pinMode(led9, OUTPUT);  
pinMode(led10, OUTPUT);  
pinMode(led11, OUTPUT);  
pinMode(led12, OUTPUT);  
pinMode(led13, OUTPUT);
```

We've done pretty much all of this sort of stuff before. No need for any explanations!



- 5/ In **void loop**, add the following line to read the LDR:

```
// Read the LDR pin  
LDRvalue = analogRead(LDRPin);
```

- 6/ Use the map function to convert the LDRvalue to a convenient range we can use for switching the LEDs and the beepRate of the buzzer.

Add the following line of code:

```
// Map the LDRvalue to a convenient range  
value = map(LDRvalue, 97, 330, 100, 350);
```

- 7/ Now we will add a series of if-then statements to switch on between one and all six LEDs according to the mapped value.

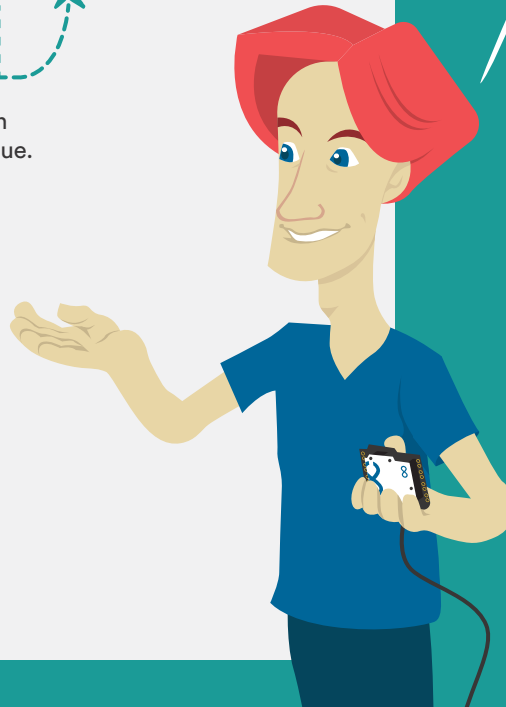
Carefully add the following lines of code:

```
// Light up the leds to show the level.  
if (value > 100) {  
    digitalWrite(led13, HIGH);  
}  
if (value > 150) {  
    digitalWrite(led12, HIGH);  
}  
if (value > 200) {  
    digitalWrite(led11, HIGH);  
}  
if (value > 250) {  
    digitalWrite(led10, HIGH);  
}
```

continued on next page \*

You will definitely want to play around with these first two values in this map function (97, 330) to tune the meter for your light conditions. But start with these ones first.

You might be able to come up with better, more compact ways to do this LED switching, but this is fine for our purposes at the moment.



```

-> * if (value > 300) {
      digitalWrite(led9, HIGH);
    }
    if (value > 350) {
      digitalWrite(led8, HIGH);
    }
  
```

The tone frequency of 650 is a nice frequency with good volume. But feel free to change this value and see what happens.

8/ Send a tone to the buzzer and make it beep according to the mapped value from the LDR by adding the following:

```

// Write the value to the buzzer.
tone(buzzerPin, 650);
beepRate = value;
delay(beepRate);
noTone(buzzerPin);
  
```

Try playing around with the `beepRate = value` code to create different effects. Maybe something like: `beepRate = value/5;` Can you figure out how this will change the beeping?

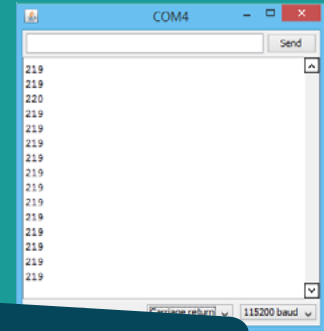
9/ Wait a jiffy then turn off the LEDs.

```

// Wait a jiffy then turn off the LEDs.
delay(60);
digitalWrite(led13, LOW);
digitalWrite(led12, LOW);
digitalWrite(led11, LOW);
digitalWrite(led10, LOW);
digitalWrite(led9, LOW);
digitalWrite(led8, LOW);
  
```

10/ Upload and start exposing the LDR to different light levels. Try to find the highest and lowest light levels in your room.

Turn on the serial monitor to see the light value coming from the LDR and see how it corresponds to the beep rate.

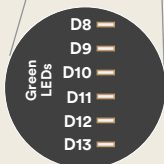
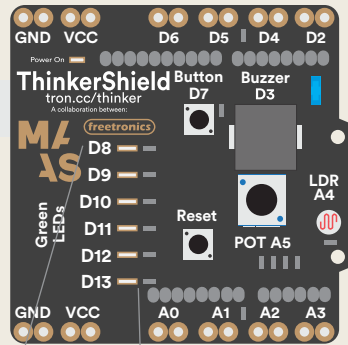


### GET.(ON).WITH.MORE

- Make the beep slow down when it's dark and speed up when it's light.
- Change the beep tone as the light changes instead of the beep rate.
- Can you work out a way to add the potentiometer into the code to use as a sensitivity dial?

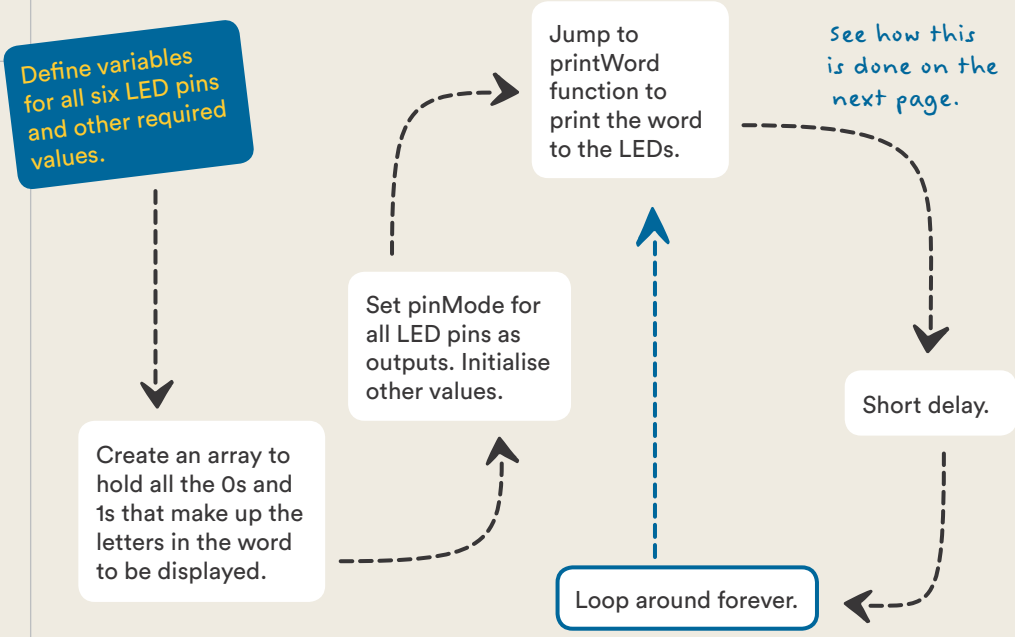
# LED.Magic.Sign

Make a cool magic sign using all six LEDs



Light-emitting diodes (LEDs) on digital pins D8-D13

Take advantage of the persistence of vision effect to create a magic sign to spell out words. Have fun making up your own characters.




## Ingredients





## LET'S.GET.(ON).WITH.IT

- 1/ Open  `tsproj_LED_Magic_Sign.ino`
- 2/ Upload the program and you will see the LEDs start to flash rapidly.
- 3/ Move the ThinkerShield back and forth in a waving motion as shown. You should start to see the word 'THINK' appearing.  
If you are having trouble seeing the word, try varying your speed or movement until the word becomes clear.
- 4/ Take a look at the code and find the comment `// The word to display`. Locate the blocks of zeroes and ones like these:

```
0,0,0,0,0,0, // T
0,0,0,0,0,1,
1,1,1,1,1,1,
0,0,0,0,0,1,
0,0,0,0,0,0,
```

This particular block represents the letter T. Have a look at the arrangement of the zeroes and ones.

Can you see how the letter is represented? You might need to twist your head to the side!

- 5/ In a new Arduino window, open the second project file:

 `tsproj_LED_Magic_Sign_Alphabet.ino`

- 6/ Let's try modifying our existing example by copying and pasting one of these letter code blocks. We'll change the word 'THINK' into 'THANK'.

This file contains code blocks to produce all the letters in the alphabet.

When waving the shield back and forth, be careful not to pull the USB cable out (and make sure you don't hit anyone!)



7/ Find the code block for the letter A. It will look like this:

```
0,0,0,0,0,0, // A
1,1,1,1,1,0,
0,0,1,0,0,1,
0,0,1,0,0,1,
1,1,1,1,1,0,
0,0,0,0,0,0,
```

*Make sure you don't forget this last comma on each of the letter blocks.*

8/ Copy this A block from the alphabet file to the clipboard. Then, back in the main file, find the code block for the letter I. Let's replace the I in THINK with the code for the letter A.

➔ 9/ Upload and try it out!

10/ Now that you have changed our existing example, try creating your own word from scratch. It's easy. Just copy and paste the letters you need from the alphabet file into the main file.

With a little creativity it's easy to create your own symbols or patterns. Use this grid (showing the letter A) as a guide and see what other shapes, letters and symbols you can come up with.

0	0	0	0	0	0
1	1	1	1	1	0
0	0	1	0	0	1
0	0	1	0	0	1
1	1	1	1	1	0
0	0	0	0	0	0

■	■	■	■	■	
		■			■
		■			■
■	■	■	■	■	

```
1,1,1,1,1,1,
1,0,0,0,0,1,
1,0,0,0,0,1,
1,0,0,0,0,1,
1,0,0,0,0,1,
1,1,1,1,1,1,
```

```
0,0,1,1,0,0,
0,0,1,1,0,0,
1,1,1,1,1,1,
1,1,1,1,1,1,
0,0,1,1,0,0,
0,0,1,1,0,0,
```

```
0,0,1,1,0,0,
0,0,1,1,1,0,
1,1,1,1,1,1,
1,1,1,1,1,1,
0,0,1,1,1,0,
0,0,1,1,0,0,
```


*Make up your own characters and symbols on a simple 6x6 grid. Just put a '0' where you want a space and a '1' in each space you want to be solid. Try these!*

```

tsproj_LED_Magic_Sign

void setup()
{
  pinMode(led1Pin, OUTPUT);
  pinMode(led2Pin, OUTPUT);
  pinMode(led3Pin, OUTPUT);
  pinMode(led4Pin, OUTPUT);
  pinMode(led5Pin, OUTPUT);
  pinMode(led6Pin, OUTPUT);

  columnDelay = 2.5;
  sizeWord = NUM_ELEM(phrase);
}

// Main loop which runs infinitely.
void loop()
{
  // Display the word using the printword function.
  printWord(phrase);
  delay(5);
}

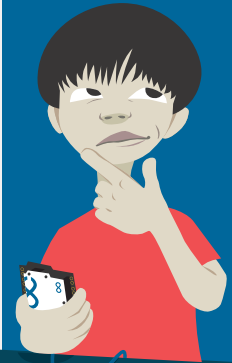
void printWord(int wordVar[])
{
  int numRows = sizeWord / 6;
  for(int j = 0; j < numRows; j++)
  {
    for(int i = 0; i < 6; i++)
    {
      digitalWrite(ledPinArray[i], wordVar[(i+j)*6]);
    }
    delay(columnDelay);
  }
}

```

How does the `printWord` function work?

The `void loop` section repeatedly runs the special function in your code that we have called `printWord`. Each time it runs it passes the data 'phrase' which is our word.

The variable, `sizeWord`, is kind of like the number of 'pixels' that will make up the phrase. As each character is six 'pixels' high, `numRows` becomes the number of rows of 'pixels' needed to make the phrase. Then, we used two nested loops to cycle through each row, for each 'pixel' and turn on the appropriate LED.



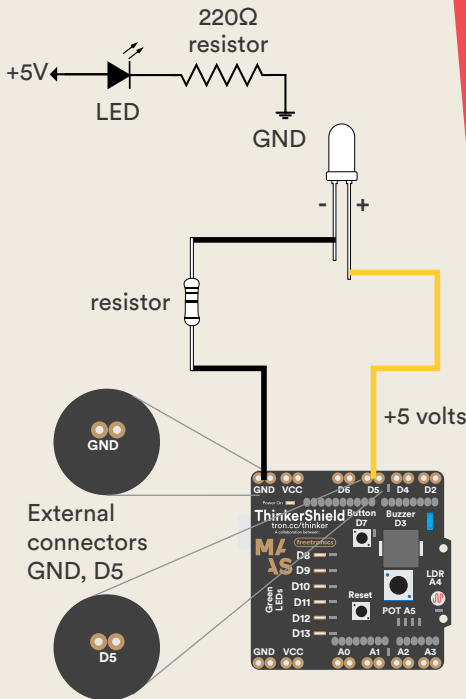
### GET.(ON).WITH.MORE

- Create your own characters
- Work out a way to make two separate words come up one after the other
- Try varying the delay amounts to suit different waving speeds
- Find a way to combine sound with your sign.

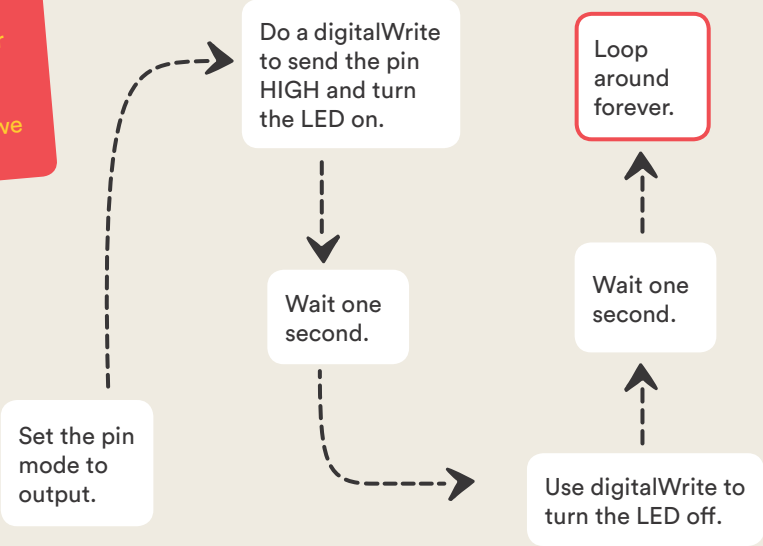
# Connect.A.LED

Move beyond the shield and flash an external LED

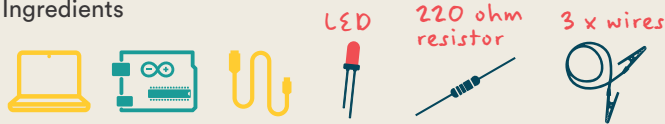
Learn to use your ThinkerShield's external pin connections to control components connected by clips and wires.



Create a variable to hold the number of the external digital pin we are connecting to — we chose D5.



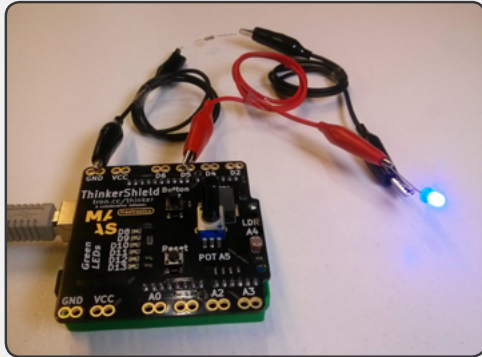
## Ingredients



## BEYOND LET'S.GET.(ON).WITH.IT

### Build it

- 1/ Attach one end of an alligator clip to the short leg of any one of the LEDs included in your kit. Attach the other end to one end of the resistor.
- 2/ Take a second wire and connect the other end of the resistor to the GND on the ThinkerShield.



- 3/ Using a third wire, attach one end to the long leg of the LED and clip the other end to the D5 on the ThinkerShield.

### Code it

- 4/ From the 'File' menu, create a new Arduino sketch and enter all the following code:
- 5/ Upload your program. If all goes well, the LED should flash.

```
// Flash external LED.  
// Define external pin.  
int ledPin = 5;  
  
// Setup.  
void setup() {  
  pinMode(ledPin, OUTPUT);  
}  
// Program loop.  
void loop() {  
  digitalWrite(ledPin, HIGH);  
  delay(1000);  
  digitalWrite(ledPin, LOW);  
  delay(1000);  
}
```



### LED won't flash?

- ✓ Check all of your connections. Make sure the clips have good contact with the copper connector rings.
- ✓ Check your code.
- ✓ Try clipping to a different external pin. Change the number in the code and upload.
- ✓ Try a different LED.

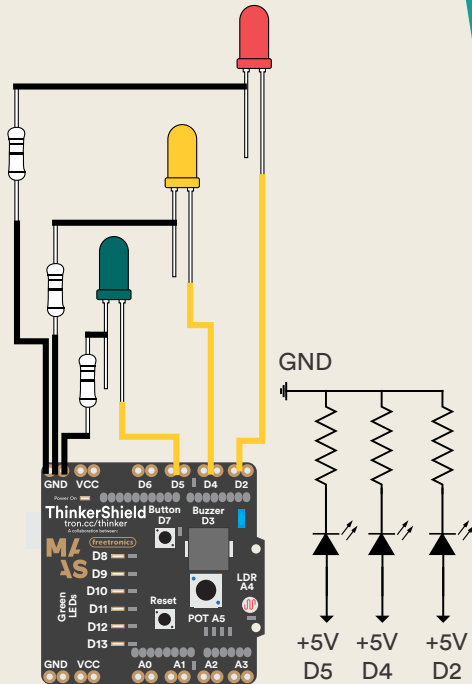
### GET.(ON).WITH.MORE

- Can you flash the external LED and one or more of the on-board LEDs?
- Use the pot to control the brightness of the LED.
- Use the external LED to effect the values from the LDR and have the buzzer buzz when the light beam is broken.



# Traffic.Lights

Control three external LEDs in a traffic light sequence



Define all required LED pins: D2, D4, D5 and set them all as OUTPUTs.

DigitalWrite to turn on the first LED.

Wait a moment.

Repeat for each LED.

DigitalWrite to turn it off.

### Make your traffic lights stand

1. Cut a piece of cardboard about 15 x 7 cm.
2. Punch three holes for the LEDs down the middle. Holes should be about 5 mm in diameter.
3. Cut and fold the flaps to make it stand.

### Ingredients

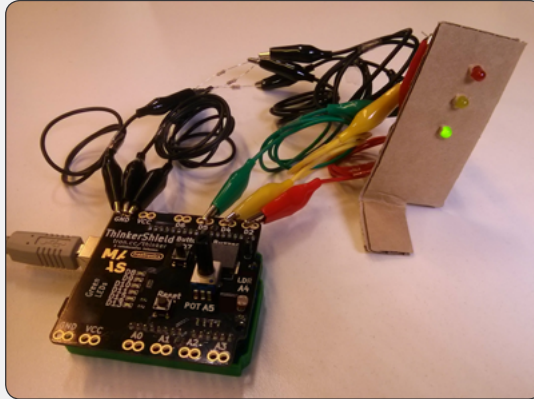
- 3 x LEDs
- 3 x resistors
- 9 x wires
- cardboard



# BEYOND LET'S.GET.(ON).WITH.IT

## Build it

- 1/ Put all three LEDs into the holes in the cardboard stand (you might want to hold the stand in place with a piece of tape).
- 2/ Using three wires and alligator clips, connect the short leg of each of the LEDs to one end of a resistor (that's three resistors, one resistor for each LED).
- 3/ Use three more wires to connect the other end of each resistor to GND on the ThinkerShield.
- 4/ Finally, use another three wires to connect the long leg of the red LED to D2, the yellow LED to D4 and the green LED to D5.



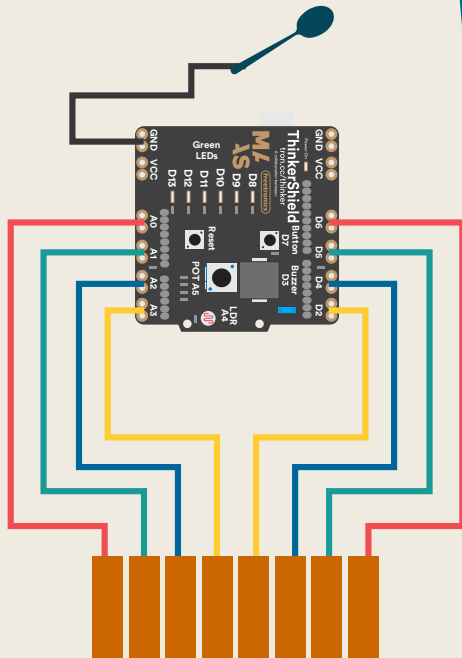
## Code it

- 5/ From the 'File' menu, create a new sketch and enter the LED traffic light code.
- 6/ Upload your file and see how it goes!

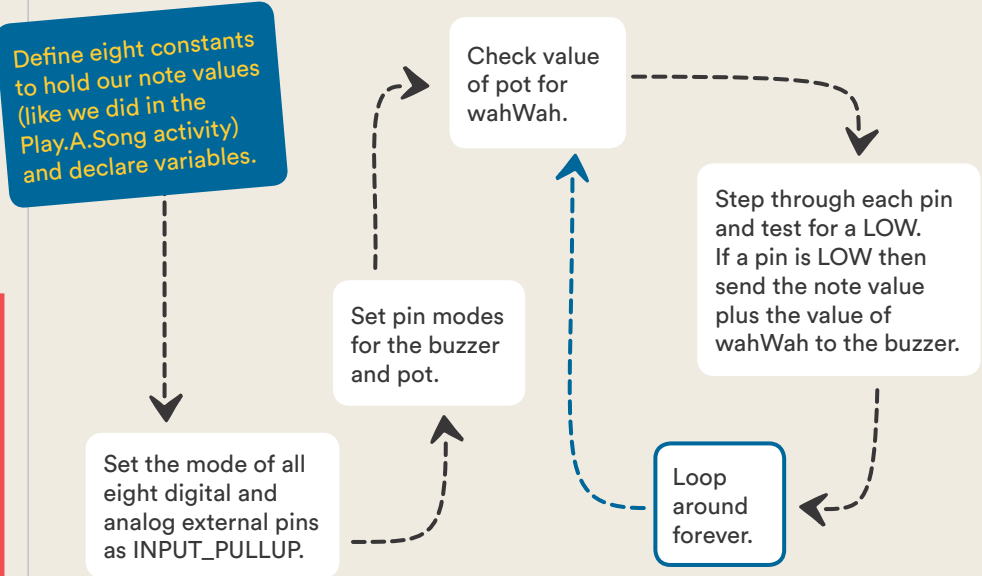
```
// LED traffic lights.  
// Define external pins.  
  
int red = 2;  
int yellow = 4;  
int green = 5;  
  
void setup()  
{  
  pinMode(red, OUTPUT);  
  pinMode(yellow, OUTPUT);  
  pinMode(green, OUTPUT);  
}  
  
void loop()  
{  
  digitalWrite(green, HIGH);  
  delay(2000);  
  digitalWrite(green, LOW);  
  digitalWrite(yellow, HIGH);  
  delay(1000);  
  digitalWrite(yellow, LOW);  
  digitalWrite(red, HIGH);  
  delay(2000);  
  digitalWrite(red, LOW);  
}
```

# Spoon Digital.Piano

Make a fun spoon-controlled piano using all eight external pins



Use all eight external pins, the buzzer and the potentiometer to make a fun digital spoon piano, complete with an awesome 'wah wah' control!



### Ingredients



aluminium foil or copper tape



metal spoon



9 x wires



cardboard

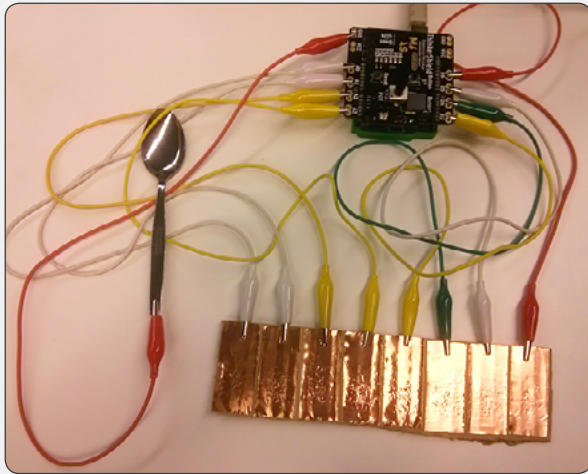




# BEYOND LET'S.GET.(ON).WITH.IT

## Build it

- 1/ Cut out eight strips of aluminium foil or copper tape about 2 x 5 cm in size. These will be your piano keys. Stick them to a piece of cardboard making sure there is a gap of about 2-3 mm between them. They must not touch each other.
- 2/ Using eight wires and alligator clips, connect the eight keys to the ThinkerShield in the order: A0, A1, A2, A3, D2, D4, D5, D6.



- 3/ With another wire, connect a metal spoon to the ThinkerShield's GND pin (you can use either one).

## Code it

- 4/ From the 'File' menu, create a new sketch and enter the following block of code to define your eight notes and declare our variables:

```
// Digital Spoon Piano.  
  
// Define notes.  
#define NOTE_C4 262  
#define NOTE_D4 294  
#define NOTE_E4 330  
#define NOTE_F4 349  
#define NOTE_G4 392  
#define NOTE_A4 440  
#define NOTE_B4 494  
#define NOTE_C5 523  
  
// Declare variables.  
int potPin = A5;  
int buzzerPin = 3;  
int wahWah = 0;
```

Of course, you can change these values to whatever you want once you know everything works.

- 5/ Now, we create our void setup () section. When connecting to external pins like this we need to initialise all eight pins as inputs and switch on their pull-up resistors using a variation of the pinMode function. Add the following code block underneath your last entries:

code on next page

```
// Set up all pin modes.
void setup()
{
  pinMode(A0, INPUT_PULLUP);
  pinMode(A1, INPUT_PULLUP);
  pinMode(A2, INPUT_PULLUP);
  pinMode(A3, INPUT_PULLUP);
  pinMode(2, INPUT_PULLUP);
  pinMode(4, INPUT_PULLUP);
  pinMode(5, INPUT_PULLUP);
  pinMode(6, INPUT_PULLUP);
  pinMode(buzzerPin, OUTPUT);
  pinMode(potPin, INPUT);
}
```

- 6/ Finally we add the loop section of the program that will actually play the notes:

```
// Program loop.
void loop()
{
  if(digitalRead(A0) == LOW) tone(buzzerPin, NOTE_C4, 100);
  if(digitalRead(A1) == LOW) tone(buzzerPin, NOTE_D4, 100);
  if(digitalRead(A2) == LOW) tone(buzzerPin, NOTE_E4, 100);
  if(digitalRead(A3) == LOW) tone(buzzerPin, NOTE_F4, 100);
  if(digitalRead(2) == LOW) tone(buzzerPin, NOTE_G4, 100);
  if(digitalRead(4) == LOW) tone(buzzerPin, NOTE_A4, 100);
  if(digitalRead(5) == LOW) tone(buzzerPin, NOTE_B4, 100);
  if(digitalRead(6) == LOW) tone(buzzerPin, NOTE_C5, 100);
}
```

- 7/ Upload and give it a try! Touch each key with the spoon to play each note.

As you can probably work out, all the void loop() bit really does is cycle around checking each pin in turn to see if it equals 'LOW'. If it does then the tone for that note is sent to the buzzer.

Also, in case you are wondering ... we are testing for LOW and not HIGH because the INPUT\_PULLUP mode inverts the behaviour of the pins. When the spoon touches a key it completes a circuit to GND which is LOW. Otherwise the pins are 'pulled' HIGH because of the INPUT\_PULLUP.



## 8/ Let's add a 'wah wah' effect!

Now that you can play standard notes with your piano, let's use the potentiometer to add a 'wah wah' effect.

Replace the program loop code you entered previously with this modified block that includes a command to read a value from the pot and add that value to the frequency of each note:

```
// Program loop.
void loop()
{
  wahWah = analogRead(potPin);
  if(digitalRead(A0) == LOW) tone(buzzerPin, NOTE_C4 + wahWah, 100);
  if(digitalRead(A1) == LOW) tone(buzzerPin, NOTE_D4 + wahWah, 100);
  if(digitalRead(A2) == LOW) tone(buzzerPin, NOTE_E4 + wahWah, 100);
  if(digitalRead(A3) == LOW) tone(buzzerPin, NOTE_F4 + wahWah, 100);
  if(digitalRead(2) == LOW) tone(buzzerPin, NOTE_G4 + wahWah, 100);
  if(digitalRead(4) == LOW) tone(buzzerPin, NOTE_A4 + wahWah, 100);
  if(digitalRead(5) == LOW) tone(buzzerPin, NOTE_B4 + wahWah, 100);
  if(digitalRead(6) == LOW) tone(buzzerPin, NOTE_C5 + wahWah, 100);
}
```

➔ 9/ Upload again. This time while a note is playing, move the potentiometer back and forth to create a fantastic 'wah wah' effect. Rock.On.With.It!



### GET.(ON).WITH.MORE

- Try adding some code to include the on-board LEDs in your piano's effects.
- Include the serial monitor so that you can always see the setting of the pot.
- Use the pushbutton as a black key so that holding it down in combination with the spoon plays the sharp # value for C, D, F, G, A.

## What.Next?

### Add components

There is a whole world of components that can be connected and controlled from your ThinkerShield:

- temperature and moisture sensors
- tilt switches, reed switches and relays
- motors and steppers
- LED ribbons, RGB LEDs
- digital displays
- lights, speakers
- home automation devices.

Visit your local or online store and see what you can find.

### Challenge yourself

If you can think it, there's probably a way to code it!

The Arduino programming language is powerful and flexible and for the most part, pretty easy to learn.

So challenge yourself, give yourself a little project you would like to try — a toy, a game or an automated piece of art and just start entering some code.

When you get stuck, or want to know more, just search the web — chances are the answer is out there and easy enough to find!

### Arduino Sketchbook and Examples

Under the 'File' menu of the Arduino software you will find the 'Sketchbook' and 'Examples' sub menus. You will be able to run lots of these programs with your ThinkerShield. Just remember to set the pin numbers at the start of the programs to match the pin numbers of the ThinkerShield's on-board components.

### Sites

The following websites are great sources of Arduino based code, projects and components:

<http://playground.arduino.cc/Projects/Ideas>

[www.instructables.com/id/Arduino-Projects/](http://www.instructables.com/id/Arduino-Projects/)

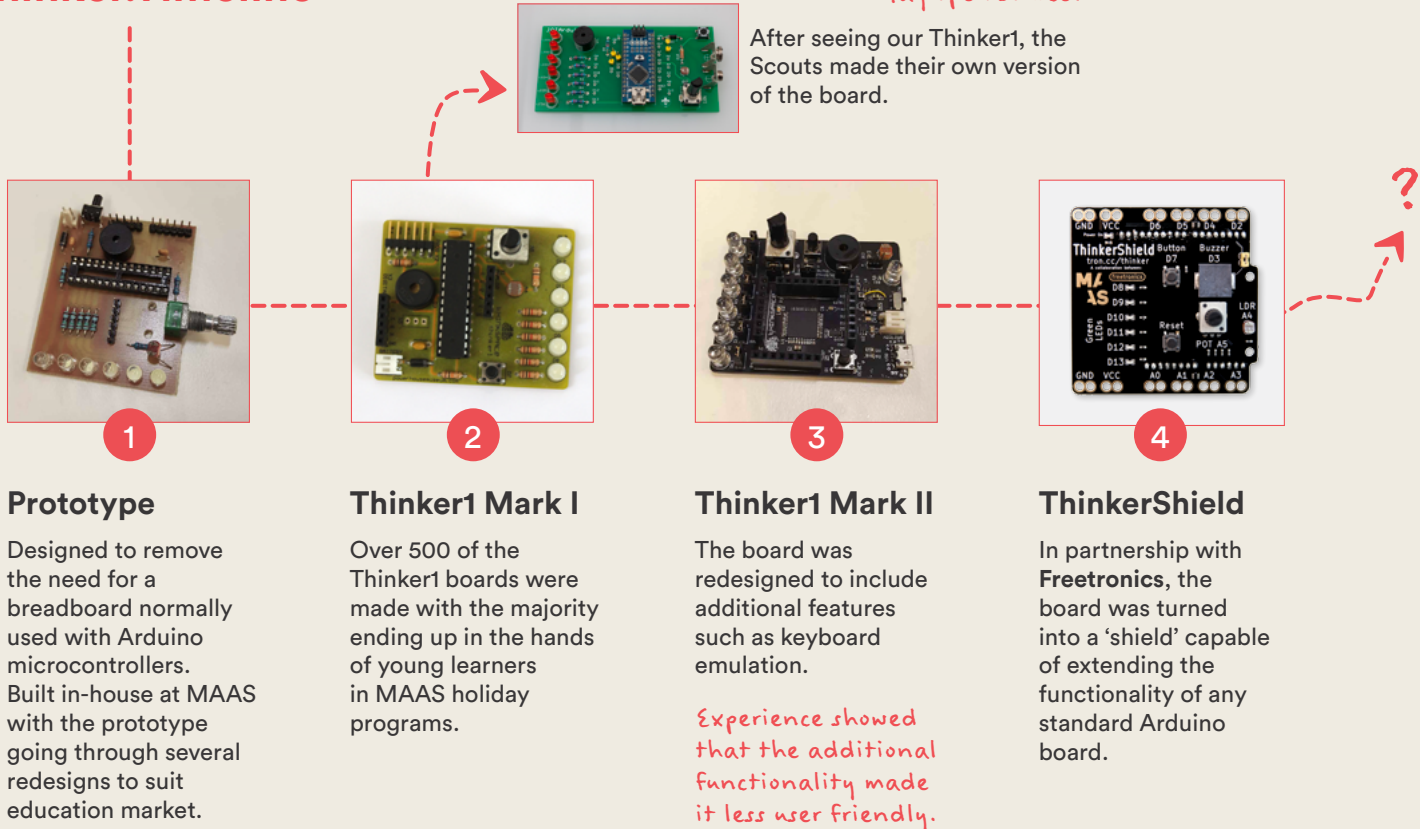
[www.freetrronics.com.au](http://www.freetrronics.com.au)

[www.stackoverflow.com](http://www.stackoverflow.com)



Or, the awesome, hilarious, sometimes tragic, but ultimately life-affirming story of the Thinkershield!

## Thinker.Timeline



### Prototype

Designed to remove the need for a breadboard normally used with Arduino microcontrollers. Built in-house at MAAS with the prototype going through several redesigns to suit education market.

### Thinker1 Mark I

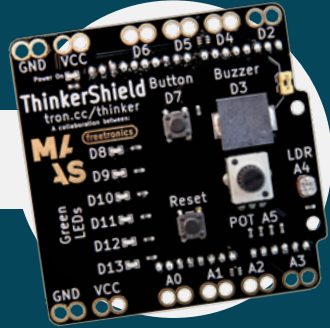
Over 500 of the Thinker1 boards were made with the majority ending up in the hands of young learners in MAAS holiday programs.

### Thinker1 Mark II

The board was redesigned to include additional features such as keyboard emulation.

### Thinkershield

In partnership with **Freetronics**, the board was turned into a 'shield' capable of extending the functionality of any standard Arduino board.



The ThinkerShield for Arduino makes it easy for anyone to get started with programming and controlling things with their computer in minutes! No need for any wiring or soldering or program knowledge.

Even if you have never seen a computer program before, we guarantee you will be making things flash, buzz, beep and respond in no time.

So don't just watch what's going on in the world of electronics and computing. Take a step to start understanding it. Grab a ThinkerShield and an Arduino, turn to the first activity in this book, and **get.on.with.it!**