GET.(ON).WITH.IT

Quick-start guide for the ThinkerShield. Attach it to your Arduino and get on with it!
All you need:

**ThinkerShield board**
A quickstart 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 them from any electronics shop for just a few dollars.*
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!
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

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:


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 your computer

Carefully align all your ThinkerShield’s leg pins with the sockets on your Arduino (as shown on the opposite page) and press them together.

Connect your Arduino to your computer using the appropriate USB connector such as a USB B-Type cable or micro USB cable (as per your board manufacturer’s specifications).
4. Select the Serial Port and Board

To communicate with your ThinkerShield (via the Arduino board) the Arduino software needs to know which communication port is being used and what type of Arduino board you have. The board type will be stated in the manufacturer’s specifications (such as Arduino UNO).

The Serial Port on Windows will likely be COM3, COM4 or COM5. On a Mac it will be something like: /dev/cu.usbserial-1B1.

If can’t easily see which port is being used try:

- using System Profiler (on Mac) or Device Manager from the Control Panel (on Windows) to identify which port the board is using.
- look at the list of available serial ports (in the Tools menu), then unplug your board and re-open the menu. The entry from the list that disappears is the correct Serial Port for your board
- checking troubleshooting on the Arduino site.

Once your board is connected, both the Serial Port and Board selections can be made from the Arduino ‘Tools’ menu.
5. Download ThinkerShield activity 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 activity files from our site:

ma.as/thinkershieldresources

Copy the files to a convenient location on your computer.

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
source: Arduino site
6 green Light-emitting diodes (LEDs) for making light

USB port (on Arduino board) to connect to your computer

external power socket (on Arduino board) to power board when not connected to a computer

push button a simple momentary switch

6 green Light-emitting diodes (LEDs) for making light

Vcc (+5 volts)

GND (0 volts)

external digital pin connections (D6, D5, D4, D2) for connecting additional components (+5 volts)

detachable buzzer jumper pull it off to silence the buzzer

piezo buzzer for making sound and music

potentiometer a controllable variable resistor

light dependent resistor for sensing light

external analog pin connections (A0, A1, A2, A3) for connecting additional sensors (+5 volts)

light dependent resistor for sensing light

reset button press to reset and rerun the currently loaded program

external digital pin connections (D6, D5, D4, D2) for connecting additional components (+5 volts)

detachable buzzer jumper pull it off to silence the buzzer

piezo buzzer for making sound and music

potentiometer a controllable variable resistor

light dependent resistor for sensing light

external analog pin connections (A0, A1, A2, A3) for connecting additional sensors (+5 volts)

light dependent resistor for sensing light

reset button press to reset and rerun the currently loaded program

external digital pin connections (D6, D5, D4, D2) for connecting additional components (+5 volts)

detachable buzzer jumper pull it off to silence the buzzer

piezo buzzer for making sound and music

potentiometer a controllable variable resistor

light dependent resistor for sensing light

external analog pin connections (A0, A1, A2, A3) for connecting additional sensors (+5 volts)
**In.A.Blink**

Start by flashing a LED

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!

**Ingredients**

- ThinkerShield
- Arduino
- USB cable

1. Connect your ThinkerShield to your Arduino and then to your computer with the USB cable.
2. Open a very simple Arduino program that will make the LED on pin D13 flash on and off.
3. Upload the program to your ThinkerShield.
4. Check that everything works and start experimenting in a blink!

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.
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!

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 1-3 again.
- Go to: www.arduino.cc click on Forum and look for troubleshooting.
What’s in an Arduino program?

**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 /* */

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

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

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

**DEBUGGER AREA:** Any errors will be displayed here. The debugger will become your best friend.

**STATUS BAR:** Displays the current board type and which COM port is being used.

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

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

**KEYWORDS:** These words have particular meaning in the Arduino language. They always appear in orange.
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.
Now let’s start talking to your ThinkerShield and take control of those LEDs so we 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.

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

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

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

Wait one second.

The waiting time is entered as milliseconds (thousandths of a second). So delay(1000) tells the program to wait one second.

Loop around forever.
LET’S.GET.(ON).WITH.IT

1/ Open the file `ts_LED_Getting_Flashy.ino`

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

```c
//Setup LED variables
// the setup routine runs once when you press reset!
void setup() {
    int ledPin = 12;
}
```

and insert the following line of code:

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

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

```c
// set the LED pin as an output:
,
```

and insert the following line of code:

```c
digitalWrite(ledPin, HIGH); //LED on
```

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

```c
//turn the LED on and off
,
```

and add the following 4 lines of code:

```c
delay(1000); //wait 1 sec
digitalWrite(ledPin, LOW); //LED off
delay(1000); //wait 1 sec
```

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

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

This gets the LED on pin 12 ready for flashing.

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

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!
Your ThinkerShield has 6 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!

**Even.Flashier**
Controlling multiple LEDs at the same time

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 digitalWrites to send each LED high or low according the on/off pattern you want to create.

Get flashier and flashier and flashier!
LET'S GET (ON) WITH IT

1/ Open the file `ts_LED_Even_Flashier.ino`

2/ Look for the comment //Setup LED variables

```cpp
// Setup LED Variables
int ledPin = 13;
```

and insert the following code:

```cpp
int led2Pin = 8;
```

3/ Find the comment //set the LED pin as an output

```cpp
// Set LED pins as digital outputs
pinMode(ledPin, OUTPUT);
```

and insert the following code:

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

4/ Look for the comment //turn the LEDs on and off

```cpp
// turn the LEDs on and off
```

and add the following code:

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

5/ Upload your program. The LEDs on pins 13 and 8 should flash on and off every half second.
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.

**Ingredients**

- Component: potentiometer

**Pot.Basics**

Reading the potentiometer

- Create a variable to hold the potPin value.
- Send the value to the Serial Monitor.

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

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

Analog devices like potentiometers (or ‘pots’) are often used as sliders on lights.

---

**Component: potentiometer**
1. Open the file `ts_Pot_Basics.ino`

2. Find the comment `//define pin` and insert the following code:
   ```
   int potPin = A4;
   ```

3. Find the comment `//turn the serial port on` and insert the following code:
   ```
   Serial.begin(115200);
   ```

4. Locate the comment `//read the input pin` and insert the following code:
   ```
   potValue = analogRead(potPin);
   ```

5. Lastly, find the comment `//print the contents of ...` and insert this line of code:
   ```
   Serial.println(potValue);
   ```

6. Upload your program. Open the Serial Monitor by clicking the magnifying glass icon. Turn the pot back and forth and see how the values change!

---

The Serial Monitor lets you see the values coming from your components.

Create a variable for analog pot pin A4.

Activate the SERIAL MONITOR so you can use it to see values on your computer screen.

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

This line prints the value to the Serial Monitor so you can see it.

Not working?
- Check your code for missing `;` or `(` or `)` or `{` or `}`
- Make sure `potPin = 4` (not some other number).
- Do the baud rates in the code and on the Serial Monitor match?
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.

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.

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

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

Ingredients

- Potentiometer on analog pin A4
- Green LEDs on digital pins D8-D13
LET'S.GET.(ON).WITH.IT

1/ Open the file `ts_Pot_Light_Dimmer.ino`

2/ Look for the comment `//define pins`

```c
int ledPin = 9;
```

This creates a variable for digital LED pin 9.

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

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

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

4/ Find the comment `//write the value to the LED`

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

This will send our value to our selected LED pin.

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 actually using something called Pulse Width Modulation (or PWM) to send varying amounts of power to the LED which changes it’s 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.
Let’s decode the Light Dimmer Code

```c
int ledPin = 9;

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

// ThinkerShield has a potentiometer connected to analog pin A4.
// 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 = A4;
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);
}

// void loop()
{
  // The read() function reads an analog value from an analog pin.
  value = analogRead(potPin);

  // To make a value greater than zero, use the line:
  value = value > 200 ? value : 0;

  // Change the brightness of the LED using the value read from the pot.
  analogWrite(ledPin, value);
}

// The following code demonstrates how to loop through an array.

int main()
{
  // We loop through an array of integers.
  int arr[] = {1, 2, 3, 4, 5};

  // Loop through the array.
  for (int i = 0; i < 5; i++)
  {
    // Print the array value.
    Serial.println(arr[i]);
  }

  return 0;
}

*/
```

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.

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.
  pinMode(ledPin, OUTPUT);

  // The potPin will default to input as we need it to be.
}
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'.

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

```
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 254 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);
}
```

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

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).

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.
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 2 LED pins as outputs. We picked pins 8 and 13.

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 2 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 A4 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**

- Potentiometer on analog pin A4
- Light-emitting diodes (LEDs) on digital pins D8, D13

**ThinkerShield**

- Button
- D7
- Buzzer
- D8
- D9
- D10
- D11
- D12
- D13

**DOT LED Bouncer**

**Component: potentiometer**
LET’S.GET.(ON).WITH.IT

1/ Open the file `ts_Pot_LED_Bouncer.ino`

2/ In the `void setup` section, find the comment
   // initialize the LED pins as digital outputs
   pinMode(led1Pin, OUTPUT);
   pinMode(led2Pin, OUTPUT);
   
3/ Find the comment //read the input pin
   value = analogRead(potPin);
   
4/ Find the comment //use if statement to turn LEDs on and off
   if(value == 1023)
   {
     digitalWrite(led1Pin, HIGH);
     digitalWrite(led2Pin, LOW);
   }
   else if(value == 0)
   {
     digitalWrite(led1Pin, LOW);
     digitalWrite(led2Pin, HIGH);
   }

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);
}
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.

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.

Turn on the Serial Port with Serial.begin so we can see the values from the sensor.

Send the value to the Serial port.

Components:
- LDR
- ThinkerShield
- LEDs
- POT

Ingredients:
- Laptop
- Breadboard
- Jumper Wires
- Sunlight

Light dependent resistor (LDR) on analog pin A5
LET’S.GET.(ON).WITH.IT

1/ Open the file `ts_LDR_Basics.ino`

2/ Find the comment `//turn the serial port on`

```cpp
//turn the serial port on

Serial.begin(115200);
```

and insert the following code:

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

This will activate the SERIAL MONITOR so you can use it to read values.

3/ Find `//read the input pin`

```cpp
// read the input pin

// print the contents of the variable to the serial monitor

value = analogRead(LDRPin);
```

and insert the following code:

```cpp
value = analogRead(LDRPin);
```

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

4/ Lastly, locate the comment `//print the contents of the variable to the serial monitor`

```cpp
// print the contents of the variable to the serial monitor

// short delay before reading again

delay(25);
```

and insert this line of code:

```cpp
Serial.println(value);
```

This will print the value to the Serial Monitor so you can see it.

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.
**LDR.Night.Light**

Make an automatic night light

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.

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

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

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

**Ingredients**

- Light-dependent resistor (LDR) on analog pin A5
- Light-emitting diodes (LEDs) on digital pins D8-D13
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

3/ Try to find the highest and lowest values that the light sensor will produce in the place you are in by exposing it to different amounts of light.

4/ Look for the comment `//map the values`

   ```
   Serial.println(LDRvalue);
   //map the values
   //use conditional statement to turn LEDs on and off
   ```

   and insert this line of code:

   ```
   value = map(LDRvalue, 97, 330, 0, 1);
   ```

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 8 lines of code:

   ```
   if (value == 1)
   {
     digitalWrite(ledPin, HIGH);
   }
   else
   {
     digitalWrite(ledPin, LOW);
   }
   ```

6/ Upload and start scaring some roaches!

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

Learn to read the state of the ThinkerShield’s push button and use it to switch a LED on and off — while the button is held down the LED will come on.

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.

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

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

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

Ingredients

Push button on digital pin D7

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

Push.Button.Basics

Component: push button

Learn to read the state of the ThinkerShield’s push button and use it to switch a LED on and off — while the button is held down the LED will come on.
1/ Open the file ts_PushButton_Basics.ino

2/ Find the comment //variables and insert the following line of code:
   ```
   boolean buttonState = false;
   ```

3/ Look for the comment //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 and insert this code:
   ```
   buttonState = digitalRead(buttonPin);
   //turn the LED on and off
   if (buttonState == true)
     digitalWrite(ledPin, HIGH);
   else
     digitalWrite(ledPin, LOW);
   ```

5/ Upload and give it a try!

---

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.

---

This conditional statement will switch the LED on if the buttonState variable is true and keep it off if it’s not.
**Get.A.Toggle.On**

Make the push button into a toggle switch

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!

- **Ingredients**
  - Light-emitting diodes (LEDs) on digital pins D8-D13
  - Push button on digital pin D7

- **Wow! My debouncing code really works!**

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.
1/ Open the file `ts_PushButton_GetAToggleOn.ino`

2/ Find the comment `// boolean (or true/false) variables` and insert these 2 lines of code:
   ```
   boolean lastButton = LOW;
   boolean ledOn = false;
   ```

3/ In the void loop section, look for the comment `// read button value` and insert this conditional code:
   ```
   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.
Learn to make sounds with the buzzer

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 3, 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.

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

Buzzer or piezo speaker on digital pin D3
1/ Open the file ts_Buzzer_Basics.ino

2/ Define the 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.

   ```
   //play a tone with the buzzer
   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.

   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 black '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.

**Ingredients**

- Buzzer or piezo speaker on digital pin D3
- Potentiometer on analog pin A4

**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 30~5000 Hz.

Use tone() to write the frequency value (freq) to the buzzer and listen to the results.
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!
Play.A.Song

Make the buzzer make music

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

Ingredients

- 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!
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.

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:

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

Here's the melody array from our code:

```cpp
NOTE_G3 262
NOTE_D4 294
NOTE_E4 330
NOTE_F4 349
NOTE_G4 392
NOTE_A4 440
NOTE_B4 494
NOTE_C5 523
```

We are using a useful technique that lets 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.

We are using a useful technique that lets 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.

```cpp
#define NOTE_G3 196
```
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.

Ingredients

- LEDs on pins D8-D13
- Pot on pin A4
- Push button on pin D7
- Buzzer on pin D3

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.

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

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.

Randomise the random number generator, set all pinModes are required and start the Serial Monitor.

Scroll the LEDs to the random dice number position.

Flash the chosen LED and make a buzz. Leave the LED on.

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

wait for a button press

return to main loop

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.

Randomise the random number generator, set all pinModes are required and start the Serial Monitor.

Flash the chosen LED and make a buzz. Leave the LED on.

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

wait for a button press

return to main loop

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.

Randomise the random number generator, set all pinModes are required and start the Serial Monitor.

Flash the chosen LED and make a buzz. Leave the LED on.

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

wait for a button press

return to main loop

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.

Randomise the random number generator, set all pinModes are required and start the Serial Monitor.

Flash the chosen LED and make a buzz. Leave the LED on.

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

wait for a button press

return to main loop

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.

Randomise the random number generator, set all pinModes are required and start the Serial Monitor.

Flash the chosen LED and make a buzz. Leave the LED on.

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

wait for a button press

return to main loop

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.

Randomise the random number generator, set all pinModes are required and start the Serial Monitor.

Flash the chosen LED and make a buzz. Leave the LED on.

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

wait for a button press

return to main loop

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.
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:

```cpp
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:

```cpp
// 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:

```cpp
// 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.

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!

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!
5/ Move down into the void loop section, locate the comment //read in the button and pot values and carefully insert these 2 lines:

```c
// 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:

```c
// 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:

```c
//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.
// 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);
}
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.

Initialise the Serial Port and set all LED pins and the buzzer pin as OUTPUTS.

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.

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

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

Delay for a time based on the light value.

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:

```cpp
//define pins
int LDRPin = A5;
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:

```cpp
//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 6 lines:

```cpp
//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:

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

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

Add the following line of code:

```cpp
// map the LDR value 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 1 and all 6 LEDs according to the mapped value.

Carefully add the following lines of code:

```cpp
// 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);
}
```

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);
}

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);

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.

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

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?

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

Define variables for all 6 LED pins and other required values.

Create an array to hold all the 0s and 1s that make up the letters in the word to be displayed.

Set pinMode for all LED pins as outputs. Initialise other values.

Jump to printWord function to print the word to the LEDs.

Loop around forever.

Short delay

See how this is done on the next page.

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’.

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

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 shapes, letters and symbols you can come up with.
LED.magic.sign (cont)

**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.

Our **printWord** function does all the hard work! The variable, **sizeWord**, is kind of like the number of "pixels" that will make up the phrase. As each character is 6 "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.
Learn to use your ThinkerShield’s external pin connections to control components connected by clips and wires.

**Ingredients**

- LED
- 220 ohm resistor
- 3 x wires
- +5 volts

**Steps**

1. **Set the pin mode to output.**
2. **Do a digitalWrite to send the pin HIGH and turn the LED on.**
3. **Wait one second.**
4. **Use digitalWrite to turn the LED off.**
5. **Wait one second.**
6. **Loop around forever.**

---

**Diagram:**

- External connectors GND, D5
- 220 Ω resistor
- +5 volts
- LED
- GND

---

**Connect.A.LED**

Move beyond the shield and flash an external LED.
**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:

```cpp
// 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);
}
```

5/ Upload your program. If all goes well, the LED should flash.

**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 3 external LEDs in a traffic light sequence

Connect 3 LEDs to the ThinkerShield’s external connectors and program them to change just like real traffic lights!

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

DigitalWrite to turn on the first LED.

Wait a moment.

DigitalWrite to turn it off.

Repeat for each LED.

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

Ingredients

- LED
- Resistors (3 x)
- Wires (9 x)
- Cardboard

Define all required LED pins: D2, D4, D5 and set them all as OUTPUTs.
LET'S GET (ON) WITH IT

Build it

1/ Put all 3 LEDs into the holes in the cardboard stand (you might want to hold the stand in place with a piece of tape).

2/ Using 3 wires and alligator clips, connect the short leg of each of the LEDs to one end of a resistor (that's 3 resistors, one resistor for each LED).

3/ Use 3 more wires to connect the other end of each resistor to GND on the ThinkerShield.

4/ Finally, use another 3 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!

```cpp
//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);
}
```
Use all 8 external pins, the buzzer and the potentiometer to make a fun digital spoon piano, complete with an awesome Wah Wah control!

Define 8 constants to hold our note values (like we did in the Play.A.Song activity) and declare variables.

Set the mode of all 8 digital and analog external pins as INPUT_PULLUP.

Check value of pot for wahWah.

Step through each pin and test for a LOW. If a pin is LOW then send the note value plus the value of wahWah to the buzzer.

Loop around forever.

Ingredients
- Aluminium foil or copper tape
- Metal spoon
- 9 x wires
- Cardboard
- Laptop
- Electronics board
- Green LEDs
- POT
- LDR
- VCC
- GND
- Reset button
- Buzzer
- D8 D9 D10 D11 D12 D13
- A1 A0 A2 A3
- D5 D6 D4 D2
- ThinkerShield
- Digital.Piano
Build it

1/ Cut out 8 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 8 wires and alligator clips, connect the 8 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 8 notes and declare our variables:

```cpp
//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 = A4;
int buzzerPin = 3;
int wahWah = 0;
```

5/ Now, we create our void setup () section. When connecting to external pins like this we need to initialise all 8 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:

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

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);
}
```

Upload and give it a try! Touch each key with the spoon to play each note.
8/ Let’s add some Wah Wah!

Now that you can play standard notes with your piano, let’s use the potentiometer to add some Wah Wah effects.

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:

```c
// 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).Wi H.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/
- www.freetronics.com.au
- www.stackoverflow.com
Or, the awesome, hilarious, sometimes tragic, but ultimately life-affirming story of the ThinkerShield!

Thinker.Timeline

1. **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.

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

3. **Thinker1 Mark II**
   The board was redesigned to include additional features such as keyboard emulation. Experience showed that the additional functionality made it less user friendly.

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

After seeing our Thinker1, the Scouts made their own version of the board.

Yay open source!
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 your ThinkerShield, an Arduino, turn to the first activity in this book and get.on.with.it!