BASIC

Learn Arduino BASICS

Tutorials

First Edition

201 Starter Kit Series

ARD-02

OSEPP™

www.osepp.com

Table of Contents

Parts 4
Included in this Kit 5
Introduction 9
Requirements 10
Installing the Arduino Environment 11
Downloading Project Code 14
Temperature 16
Reading Temperature 17
Light 22
Sensing Light 23
Automatic Light Switch 29
LEDs 34
LED Serial Switch 35
LED Knight Rider 40
LED Bar Graph 46
Sound 51
Melody 52
Pitch Follower 58
Knock Sensor 63
Ultrasonic 68
Ultrasonic Range Finder 69
Servo Motors 75
Introduction to Servo Motors 76
Servo Motor Control 81
Stepper Motors 86
Introduction to Stepper Motors 87
Stepper Motor Control 91
LCD 96
Using the LCD 97
Project Ideas 102
Door Alarm 103
Stay Connected 110
Support 111

Terms used in this book

Code A collection of human readable statements that can be compiled into machine language.

Sketch What Arduino users refer to as the code that runs on an Arduino board. It can be made up of one or more source files.

Project An encapsulation of all assets used to accomplish a task. In this book, this is typically your Arduino Sketch and your circuit.

Circuit A collection of electrical components which form a closed current loop.

Current A measurement for the flow of electrical charge.

Voltage A measurement for the electrical potential energy difference.

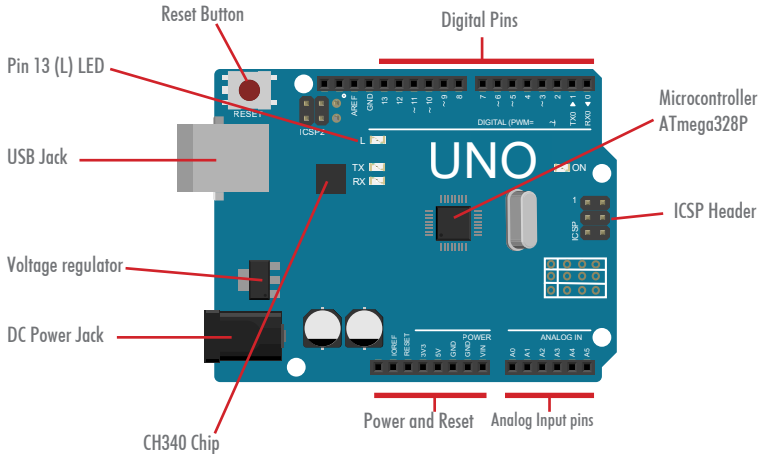
Function A collection of statements which can be invoked. Functions can take any number of arguments and can return data back to the caller.

Method Similar to a function, but is encapsulated by an object.

Parts

Included in this Kit

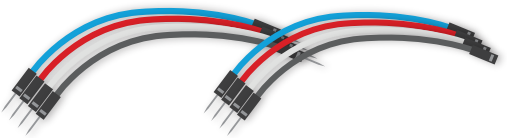
UNO	Microcontroller	ATmega328P
	Clock Speed	16 MHz
	Flash Memory	32 KB
	SRAM	2 KB
	EEPROM	1 KB
	Operating Voltage	5V
	Input Voltage	6-12 V
	Digital I/O Pin Count	14 (including 6 for PWM output)
	Analog Input Pin Count	6
	Other Connections	Mini-USB
		CSP for ATmega2560
		DC power connector
		4-pin latchable Molex connector
	Dimensions	2.95 x 2.13 x 0.61 inches
		(75.0 x 54.0 x 15.5 mm)
	Power Source	USB or external DC power supply



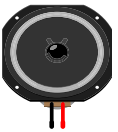
Parts

Jumper Cables 40 M/M

10 F/M

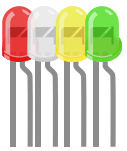


Speakers 1 - 8 ohm, 0.5w

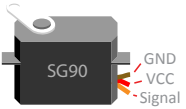


Buzzers 1

LEDs 12 - Red
2 - Green
2 - Blue
2 - Yellow
2 - White

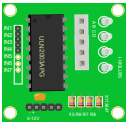


Servos 1 - SG90

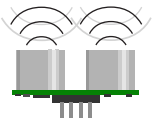


Steppers 1 - 28BYJ-48, 5v

Stepper Driver 1 - ULN2003 Driver

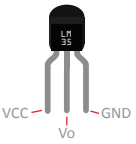


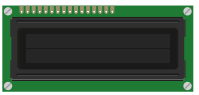
Ultrasonic Range finder 1 - HC-SR04

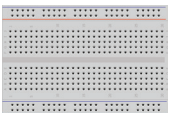


**Images are developed using Fritzing

Potentiometers	2 - 10k ohm, linear
Tact push buttons	2 - Momentary
Temperature sensor	1 - LM35
Photocell	1
LCD	1 - HD44780 compatible, 16x2
Breadboard	1
USB Cable	1 - USB Cable
2-Pin Screw Terminal	2
9v Cable	1





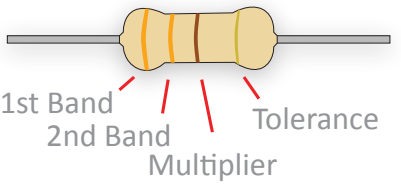


**Images are developed using Fritzing

Parts

Resistors	5 - 1M ohm	Brown/Black/Green/Gold
	5 - 10k ohm	Brown/Black/Orange/Gold
	5 - 1k ohm	Brown/Black/Red/Gold
	10 - 330 ohm	Orange/Orange/Brown/Gold
	5 - 100 ohm	Brown/Black/Brown/Gold

Reading a resistors color band



To distinguish left from right there is a gap between the multiplyer and tolerance bands.

You can easily read a resistors color band to find its value. Simply look up the 1st and 2nd band colors in the table below, find the value from the "Significant figures" column ex) Orange, Orange would be 33. Then take the multiplier band color and look up the value from the multiplier column in the table below. You then multiply the number from the first two bands by the multiplier value to find the resistors resistance. The tolerance band tells you the quality of the resistor, how close the actual resistance is to the recorded resistance.

ex) Orange/Orange/Brown/Gold is 33 x 10^1 = 330 ohms at ±5% tolerance

Color	Significant figures	Multiplier	Tolerance
Black	0	$\times 10^0$	–
Brown	1	$\times 10^1$	±1%
Red	2	$\times 10^2$	±2%
Orange	3	$\times 10^3$	–
Yellow	4	$\times 10^4$	(±5%)
Green	5	$\times 10^5$	±0.5%
Blue	6	$\times 10^6$	±0.25%
Violet	7	$\times 10^7$	±0.1%
Gray	8	$\times 10^8$	±0.05% (±10%)
White	9	$\times 10^9$	–
Gold	–	$\times 10^{-1}$	±5%
Silver	–	$\times 10^{-2}$	±10%
None	–	–	±20%

Introduction

This section includes:

- Requirements
- Installing the Arduino Environment
- Downloading Project Code

Requirements

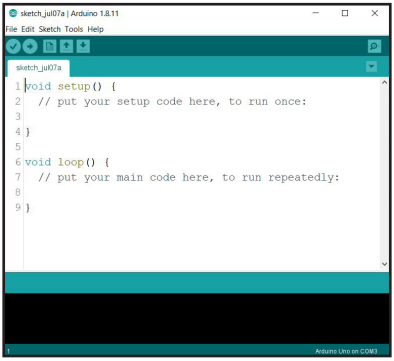
This kit requires no prior electronic knowledge. We've included easy to follow breadboard diagrams for every tutorial to guide you through wiring the individual components.

This book only supports Microsoft Windows, however, Arduino is compatible with Windows, Mac OS, and Linux. If you're comfortable using the Arduino environment on your platform don't worry if you're not using Windows.

You're holding the 201 kit, we also offer the 101 Starter kit which is more of an introductory kit. If you skipped the 101 don't worry. You've got everything you need to do the tutorials in this booklet.

This booklet covers some more advanced topics than the 101, but can stand on its own.

Installing the Arduino Environment



The Arduino Environment is the collection of software used to compile, and load your projects onto your Arduino.

If you haven't already installed the Arduino Environment, here's how to do it on the Microsoft Windows operating system.

Step I

Download the latest Arduino Environment Windows Installer here:

http://arduino.cc/en/main/software

Step II

Connect your Arduino board to your PC.

Step III

Test your Arduino Environment by attempting to upload a sketch.

1. Open the Arduino Environment found in your program files
2. Load up the blinking LED example sketch; in the File menu go to.

File » Examples » 01.Basics » Blink

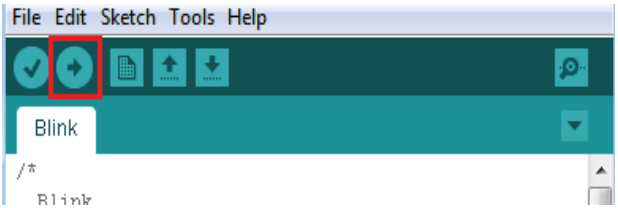
3. Select your board, in the File menu go to:

Tools » Boards » Arduino Uno

4. Select the virtual COM port your Arduino has been assigned; in the File menu go to:

Tools » Port » COM X

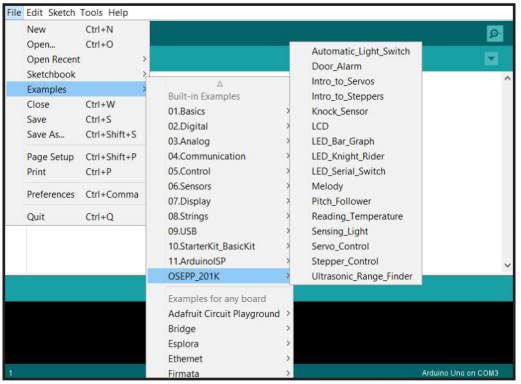
5. Upload the sketch by clicking the upload button



6. Verify that the LED is blinking on your UNO board.

If you're unable to get the LED to blink look for further help online. Check the last page of this book for OSEPP support contact information. We're always happy to help in any way we can!

Downloading Project Code



Please take the time to download the project code and install it into your Arduino environment. This will give you easy access to all the tutorials in this booklet.

Step I

Download our zip file here:

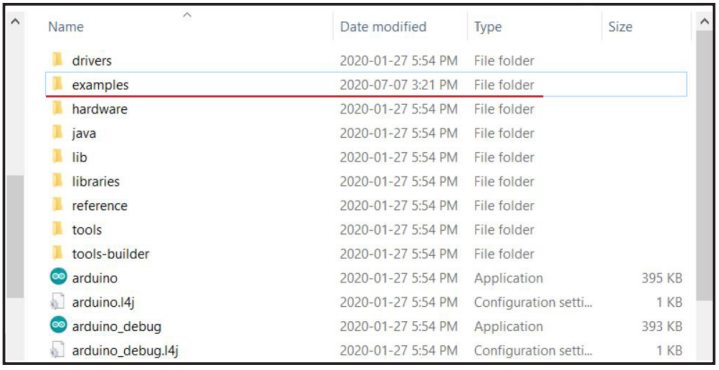
http://osepp.com/files/osepp_201k.zip

Step II

Unzip the osepp_201 Zip file.

Step III

Open the Arduino environment and drag osepp_201k file into the "examples" folder. This is the folder where all of your Arduino example sketches live.



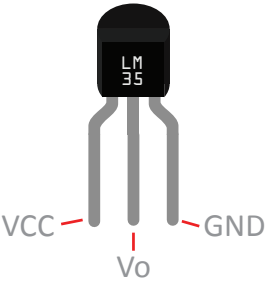
All the project code from this tutorial book has now been added to your examples menu.

How to load the sketch for each tutorial?
To access all the project code in this booklet; in the File menu go to:

File » Examples » OSEPP_201K

Temperature

New Part:



LM35 Temperature Sensor

In this section we learn how to read the temperature.

Tutorials include:

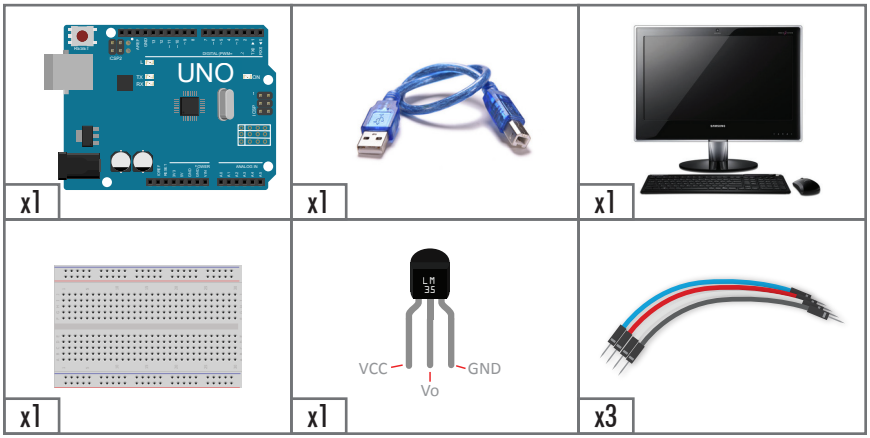
1. Using the included LM35 temperature sensor.

Reading Temperature

In this tutorial we will learn how to read the temperature in Celsius using the LM35 temperature sensor and the UNO's Analog-Digital-Converter (ADC).

I - Things you will need

You will need an UNO, USB cable, a computer, a breadboard, a LM35 temperature sensor, and 3 jumper cables.



II – Background information

The LM35 temperature sensor is a simple device which when powered will output 0.01V per degree Centigrade on the Vo pin. We can use the ADC on the UNO to measure that voltage. The ADC on your UNO will read any voltage from GND to VCC (5v) and convert it to a digital number. This number is a 10bit value which is proportional to the voltage applied. At 0v it will read 0, and at VCC it will read 1023.

We need to do some math to convert the ADC reading to degrees Centigrade.
Since the ADC reading is proportional to the voltage applied we can get the voltage on the ADC pin from the ADC value read like this:

$$V=(ADC/1023)*VCC \qquad \textbf{EQ1}$$

Where ADC is the ADC value read, VCC is the supply voltage (5v), and V is the voltage applied to the ADC pin.

Now that we have the voltage read from the Vo pin on our LM35 temperature sensor we need to convert that voltage to degrees Centigrade. The LM35 will output 0.01V per degree Centigrade, let's express that like this:

$$dC=V/(0.01) \text{ OR } dC=V*100 \qquad \textbf{EQ2}$$

Where V is the voltage from on the Vo pin of the LM35 sensor, dC is the degrees in Centigrade.

If we combine the two equations by substituting V from the EQ2 for EQ1 we get:

$$dC=(ADC/1023)*VCC*100 \qquad \textbf{EQ3}$$

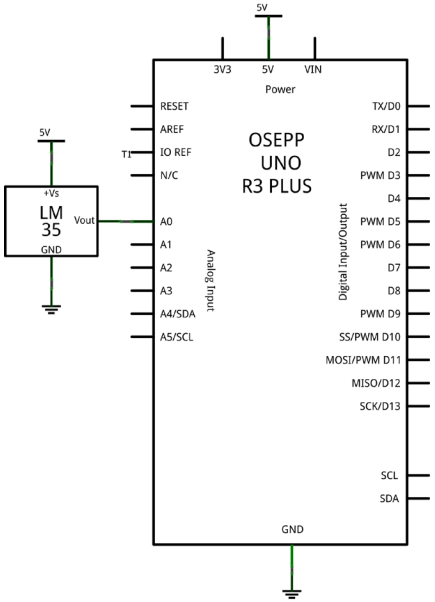
To simplify the equation a little bit we can input 5 for VCC and multiply it by 100:

$$dC = (ADC/1023)*500 \quad \textbf{EQ4}$$

Where ADC is the ADC value read, and dC is the degrees in Centigrade.

III – Schematic Diagram

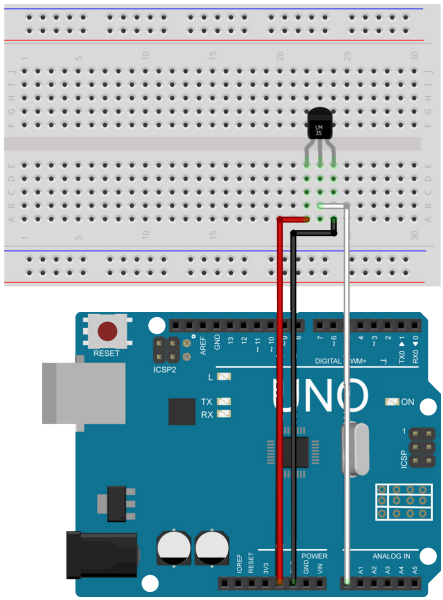
On the left of the schematic you'll see the LM35 temperature sensor. We're going to connect it to 5v, gnd, and A0 on the UNO.



**Images are developed using Fritzing

IV – Wiring the Breadboard

Let's wire it up. Take those jumper cables and connect them to the LM35 sensor using your breadboard as shown here.



**Images are developed using Fritzing

V – Writing the Sketch

Start a new project and load up the following sketch. If you imported our library (refer to page 14) you'll be able to load it up in the Arduino environment by going:

File » Examples » OSEPP_201K » Reading_Temperature

This sketch will read the temperature from the LM35 sensor and print it out over serial.

```
/*
 * Tutorial 1: Reading Temperature
 *
 * Read the temperature in celcius over serial.
 *
 *
 * To see this sketch in action open up the serial
 * monitor. Clamp your fingers over the LM35
 * sensor. The temperature will rise and then fall
 * as you remove your fingers.
 *
 * The circuit:
 * - LM35 to 5v, GND, and Vo to analog 0
 *
 */

// the output pin of the LM35 temperature sensor
int lm35Pin = A0;

void setup()
{
    // set up serial at 9600 baud
    Serial.begin(9600);
}

void loop()
{
    int analogValue;
    float temperature;

    // read our temperature sensor
    analogValue = analogRead(lm35Pin);

    // convert the 10bit analog value to celcius
    temperature = float(analogValue) / 1023;
    temperature = temperature * 500;

    // print the temperature over serial
    Serial.print("Temp: ");
    Serial.print(temperature);
    Serial.println("C");

    // wait 1s before reading the temperature again
    delay(1000);
}
```

Code Hint: On line 35 we need to convert the variable analogValue to a float before we do division. If you don't convert it to a float the program will do integer division and we will end up with a whole number (0 or 1) instead of a decimal number from 0 to 1.

To learn more about data types check out the Arduino Reference pages.

Once loaded into your UNO open up the serial monitor; in the file menu, go:

Tools » Serial Monitor

You will see the temperature printed out every second in the serial monitor window.

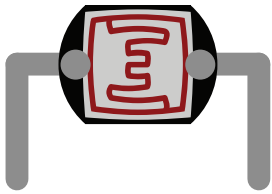
VI - Review

In this tutorial we've learnt:

- How to read values from the UNOs ADC
- How to convert the ADC value read to the actual voltage applied to the analog pin on your UNO
- How to convert the voltage from the LM35 to degrees Centigrade

Light

New Part:



Photocell

In this section we will learn how to measure light.

Tutorials include:

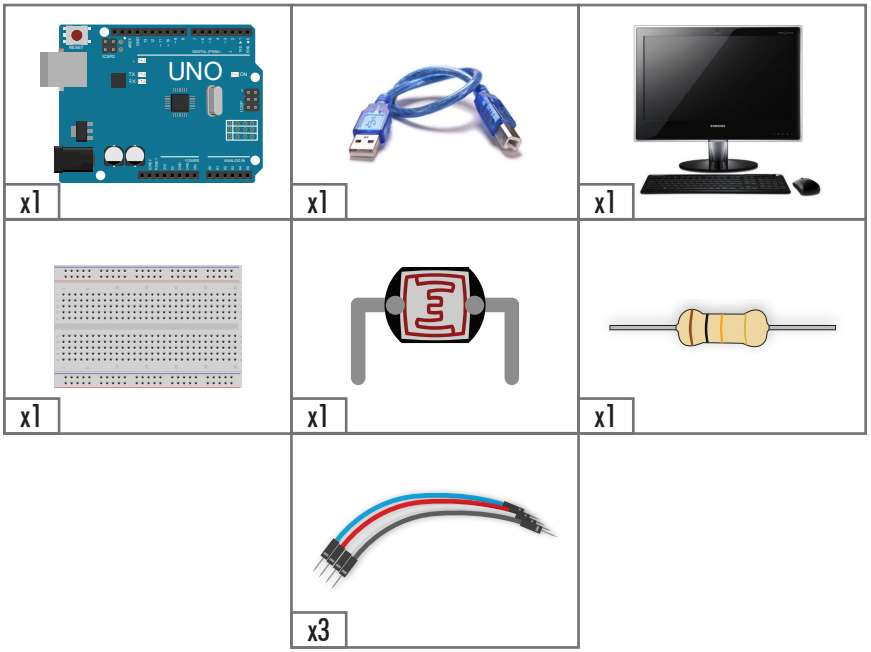
1. Sensing Light
2. Automatic Light Switch

Sensing Light

In this tutorial we will learn how to read the amount of light hitting our photocell. We learn about the programming flow-of-control feature called the Switch. Our sketch will read the photocell and send four brightness states over serial.

I - Things you will need

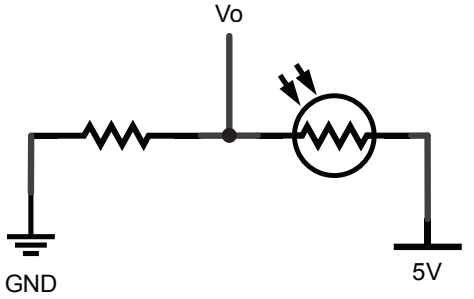
You will need an UNO, USB cable, a computer, a breadboard, a photocell, a 10k resistor, and 3 jumper cables.



Light

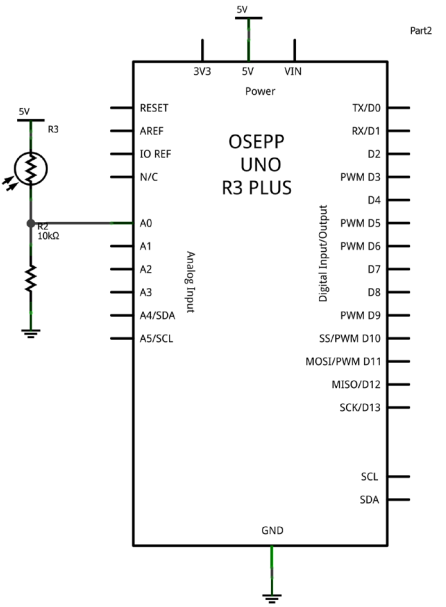
II - Background information

A photocell behaves like a variable resistor. The more light applied to the photocell, the less resistance it has. Since the analog pins on the UNO reads voltages, not resistance, we need a circuit to convert the photocell's resistance into a voltage we can read. To do this we can make a voltage divider by adding a fixed resistor in series with the photocell.



III – Schematic Diagram

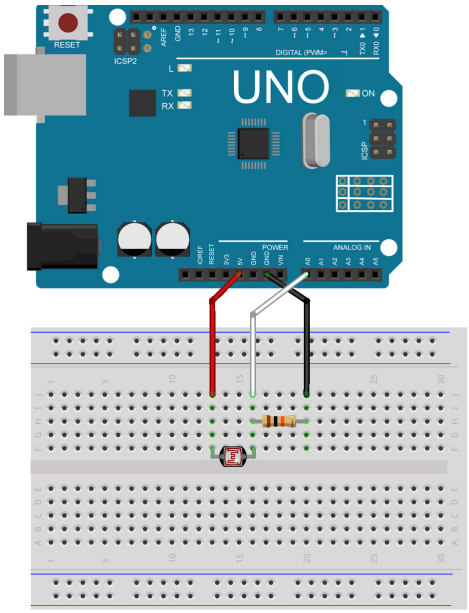
Here's the schematic diagram. We will connect a 10k resistor in series with the photocell, and wire the center of the divider to the UNOs A0 pin.



**Images are developed using Fritzing

IV – Wiring the Breadboard

Let's wire up the breadboard.



**Images are developed using Fritzing

V – Writing the Sketch

Start a new project and load up the following sketch. If you imported our library (refer to page 14) you'll be able to load it up in the Arduino environment by going:

File » Examples » OSEPP_201K » Sensing_Light

This sketch will read brightness and will print out dark, dim, medium, or bright over serial.

```
/*
 * Tutorial 2a: Sensing Light
 *
 * Measure brightness using a photocell over serial.
 *
 *
 * To see this sketch in action, put the board and sensor in a well-lit
 * room, open the serial monitor, and and move your hand gradually
 * down over the sensor.
 *
 * The circuit:
 * - photoresistor from analog in 0 to +5V
 * - 10K resistor from analog in 0 to ground
```

Light

```
 *
 *
 * created 1 Jul 2009
 * modified 9 Apr 2012
 * by Tom Igoe
 * modified 13 August 2013
 * by Blaise Jarrett
 *
 * This example code is in the public domain.
 *
 * Derivative work from:
 * http://www.arduino.cc/en/Tutorial/SwitchCase
 *
 */

// these constants won't change. They are the
// lowest and highest readings you get from your sensor:
//
// sensor minimum, discovered through experiment
const int sensorMin = 0;
// sensor maximum, discovered through experiment
const int sensorMax = 800;

// the photocell voltage divider pin
int photocellPin = A0;

void setup()
{
    // set up serial at 9600 baud
    Serial.begin(9600);
}

void loop()
{
    int analogValue;
    int range;

    // read our photocell
    analogValue = analogRead(photocellPin);
    // map the sensor range to a range of four options
    range = map(analogValue, sensorMin, sensorMax, 0, 3);

    // do something different depending on the
    // range value
    switch (range)
    {
        // your hand is on the sensor
        case 0:
            Serial.println("dark");
            break;
        // your hand is close to the sensor
        case 1:
            Serial.println("dim");
            break;
        // your hand is a few inches from the sensor
        case 2:
            Serial.println("medium");
            break;
        // your hand is nowhere near the sensor
        case 3:
            Serial.println("bright");
            break;
    }

    // wait 0.25s before reading the photocell again
    delay(250);
}
```

Once loaded into your UNO open up the serial monitor; in the file menu, go:

Tools » Serial Monitor

Move your hand over the photocell and observe what gets printed out in the serial window.

VI – Switch and Map Explained

The Switch Statement:

Like if statements, switch...case controls the flow of programs by allowing programmers to specify different code that should be executed in various conditions. In particular, a switch statement compares the value of a variable to the values specified in case statements. When a case statement is found whose value matches that of the variable, the code in that case statement is run.

The break keyword exits the switch statement, and is typically used at the end of each case. Without a break statement, the switch statement will continue executing the following expressions ("falling-through") until a break, or the end of the switch statement is reached.

Alternatively we could have written this sketch with four if/else statements, but a switch is a little bit cleaner.

The map function explained:

"map(value, fromLow, fromHigh, toLow, toHigh)"
Re-maps a number from one range to another. That is, a value of fromLow would get mapped to toLow, a value of fromHigh to toHigh, values in-between to values in-between, etc.

VII - Review

In this tutorial we've learnt:

- How to read a photocell
- How to use the map function

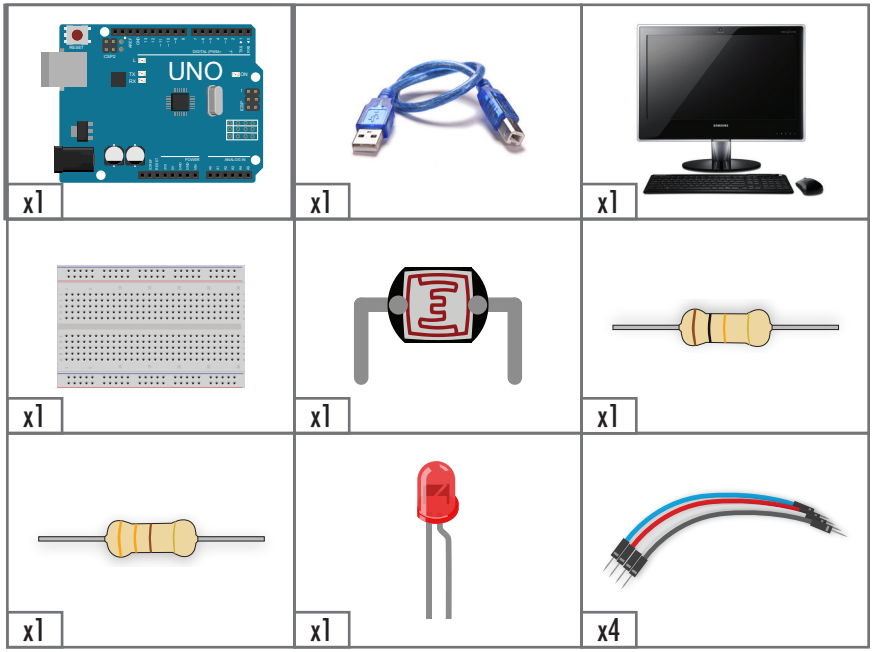
Automatic Light Switch

In this tutorial we will learn how to automatically turn an LED on when it's dark by reading the amount of light hitting a photocell.

This is an extension on the previous tutorial. Refer to the previous tutorial to learn more about the photocell.

I - Things you will need

You will need an UNO, USB cable, a computer, a breadboard, a photocell, a 10k resistor, a 330ohm resistor, an LED, and 4 jumper cables.



Light

II - Background information

We're going to use what we've learned in the previous tutorial to determine if it's dark, but this time we're going to use that information to toggle an LED instead of printing it out over serial.

The UNO has 14 digital pins that can be configured as input or output. We can use a digital pin in output mode to drive an LED. In output mode, digitalWrite(LED, LOW) will set the pin to 0v, and digitalWrite(LED, HIGH) will set the pin to VCC(5v).

A resistor is needed when using an LED to limit the current flow. Typical LEDs are rated for up to 20mA of forward current and have a forward voltage drop of 2V. We can use the following formula to find the minimum resistor value for our LED:

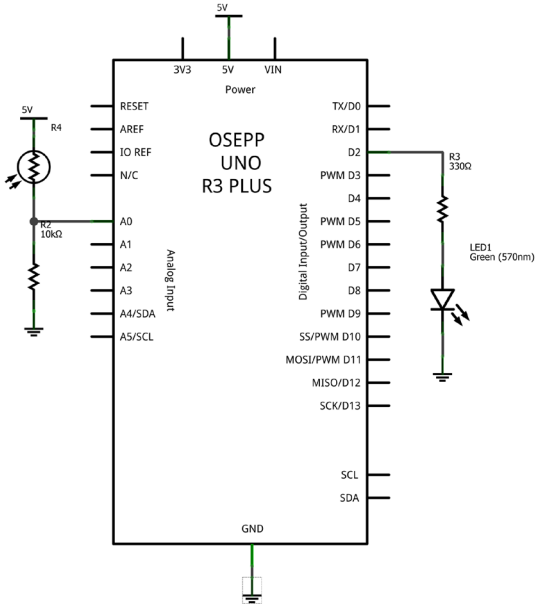
$$R = (VCC - Vf)/If$$

Where VCC is the supply voltage (5v on the UNO), Vf is the LEDs forward voltage drop, and If is the LEDs forward voltage current.

If you solve that equation using 5v for VCC, 2v for Vf, and 0.020A (20mA) for If you'll get a minimum resistor value of 150ohms. We're going to use a 330ohm resistor in this tutorial.

III – Schematic Diagram

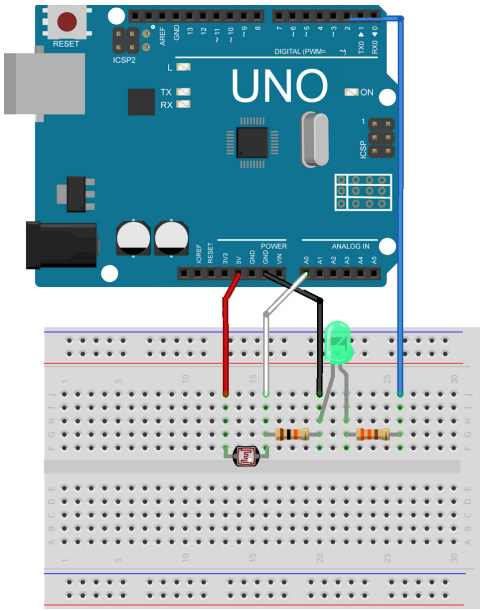
Here's the schematic diagram. We will connect a 10k resistor in series with the photocell, and wire the center of the divider to the UNOs A0 pin.



**Images are developed using Fritzing

IV – Wiring the Breadboard

Let's wire up the breadboard. Take the board you made in the previous tutorial and add an LED and resistor to it.



**Images are developed using Fritzing

V – Writing the Sketch

Start a new project and load up the following sketch. If you imported our library (refer to page 14) you'll be able to load it up in the Arduino environment by going:

File » Examples » OSEPP_201K » Automatic_Light_Switch

This sketch will automatically turn on an LED when it gets dark.

```
/*
 * Tutorial 2b: Automatic Light Switch
 *
 * Automatically turns on an LED when it gets dark.
 *
 *
 * To see this sketch in action put the board in a
 * room with little or no sunlight, only lit by your room lights.
 * Turn the room lights on and off. The LED will automatically
 * turn on when its dark and off when its light.
 *
 * The circuit:
 * - photoresistor from analog in 0 to +5V
 * - 10K resistor from analog in 0 to ground
 * - LED connected to digital pin 2 through a 300ohm resistor
 *
 * Author: Blaise Jarrett - OSEPP
 *
 */

// A constant that describes when its dark enough to
// light the LED. A value close to 600 will light the led
// with less darkness. Play with this number.
const int sensorDark = 500;

// the photocell voltage divider pin
int photocellPin = A0;
// the LED pin
int LEDPin = 2;

void setup()
{
    // initialize the LED pin as output
    pinMode(LEDPin, OUTPUT);
}
```

```
36
37 void loop()
38 {
39     int analogValue;
40
41     // read our photocell
42     analogValue = analogRead(photocellPin);
43
44     // The higher the analogValue reading is the darker it is.
45     // If its atleast as dark as our constant "sensorDark"
46     // light the LED
47     if (analogValue < sensorDark)
48     {
49         digitalWrite(LEDPin, HIGH);
50     }
51     // Otherwise turn the LED off
52     else
53     {
54         digitalWrite(LEDPin, LOW);
55     }
56
57     // wait 1ms for better quality sensor readings
58     delay(1);
59 }
```

Try playing with the sensorDark constant to adjust the brightness that triggers when the LED turns on.

VII - Review

In this tutorial we've learnt:

- How to toggle an LED

LEDs

In this section we learn how to control LEDs over serial and build a couple of LED displays.

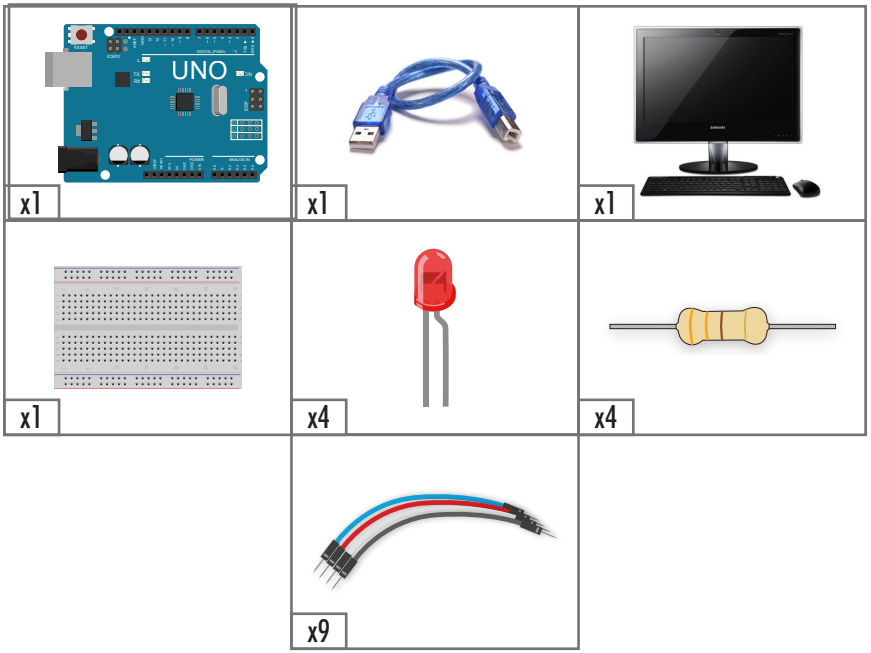
Tutorials include:
1. LED Serial Switch
2. LED Knight Rider
3. LED Bar Graph

LED Serial Switch

In this tutorial we will learn how to control four LEDs using serial. We're going to do this using the switch statement.

I - Things you will need

You will need an UNO, USB cable, a computer, a breadboard, 4 LEDs (red, green, yellow, white), 4 330ohm resistors, and 9 jumper cables.



LEDs

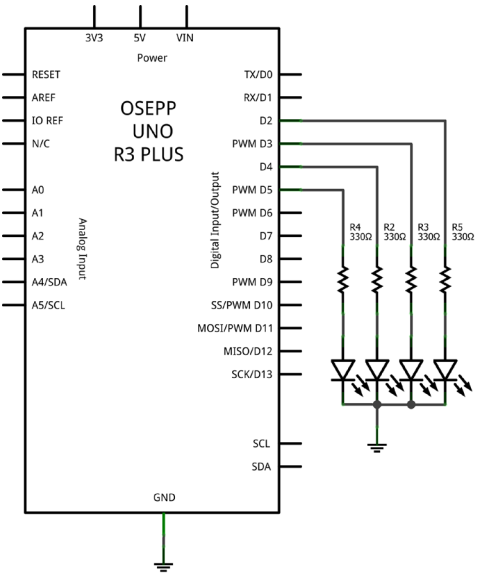
II - Background information

We used serial already in the "Sensing Light" and "Reading Temperature" tutorials. In those tutorials we printed data out over serial. This time we're going to take some user input over serial instead.

We're going to control four LEDs over serial by sending characters to toggle each LED. To read a character from serial we use the method "Serial.read()".

III – Schematic Diagram

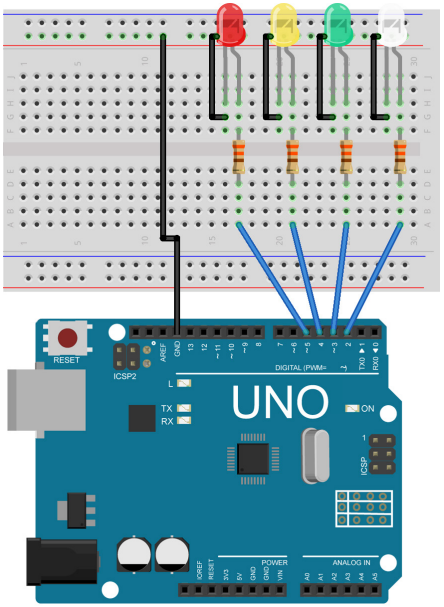
Here's the schematic diagram.



**Images are developed using Fritzing

IV – Wiring the Breadboard

Let's wire up the breadboard.



**Images are developed using Fritzing

V – Writing the Sketch

Start a new project and load up the following sketch. If you imported our library (refer to page 14) you'll be able to load it up in the Arduino environment by going:

File » Examples » OSEPP_201K » LED_Serial_Switch

```
/*
 * Tutorial 3a: LED Switch
 *
 * Toggle LEDs over serial using the Switch case.
 *
 *
 * To see this sketch in action open the serial monitor
 * and send these characters to toggle the corresponding LED:
 * - r = Red
 * - y = Yellow
 * - g = Green
 * - w = white
 * Send any other character to turn all four LEDs off.
 *
 * The circuit:
 * - 4 LEDs attached to digital pins 2 through 5 with 330ohm resistors
 *
 * Author: Blaise Jarrett
 *
 */
```

LEDs

```
// The LEDs are connected to these pins
int LEDRedPin = 5;
int LEDYellowPin = 4;
int LEDGreenPin = 3;
int LEDWhitePin = 2;

void setup()
{
    // set up serial at 9600 baud
    Serial.begin(9600);

    // set all four LED pins to output mode
    pinMode(LEDRedPin, OUTPUT);
    pinMode(LEDYellowPin, OUTPUT);
    pinMode(LEDGreenPin, OUTPUT);
    pinMode(LEDWhitePin, OUTPUT);
}

void toggleLED(int LEDPin)
{
    // toggle the LED on the pin passed as an argument
    digitalWrite(LEDPin, !digitalRead(LEDPin));
}

void loop()
{
    if (Serial.available() > 0)
    {
        // read a single character over serial
        int inByte = Serial.read();

        // do something different for each character
        switch (inByte)
        {
            // if we receive r, y, g, or w
            // toggle the respective LED using our function
            case 'r':
                toggleLED(LEDRedPin);
                break;
            case 'y':
                toggleLED(LEDYellowPin);
                break;
            case 'g':
                toggleLED(LEDGreenPin);
                break;
            case 'w':
                toggleLED(LEDWhitePin);
                break;
            default:
                // if we receive any other character turn all
                // the LEDs off
                digitalWrite(LEDRedPin, LOW);
                digitalWrite(LEDYellowPin, LOW);
                digitalWrite(LEDGreenPin, LOW);
                digitalWrite(LEDWhitePin, LOW);
                break;
        }
    }
}
```

Once loaded into your UNO open up the serial monitor; in the file menu, go:

Tools » Serial Monitor

Send these characters one at a time to toggle each LED:

- r - Red
- y - Yellow
- g - Green
- w - white

VII - Review

In this tutorial we've learnt:

- How to control our program with our computer using serial
- About the default case of the switch statement

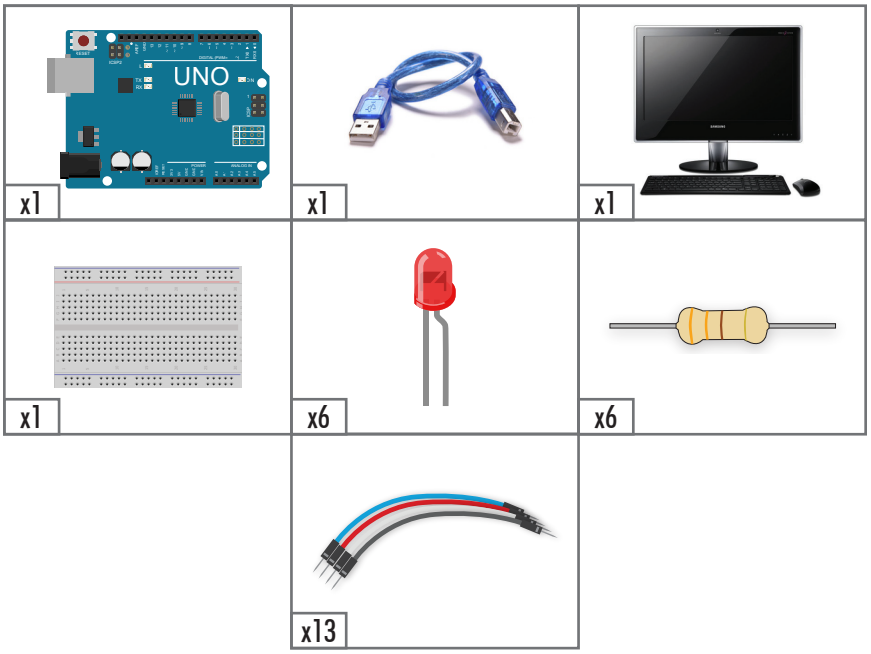
LED Knight Rider

In this tutorial we will learn about the For Loop.

We call this tutorial "Knight Rider" in memory of a TV-series from the 80's where David Hasselhoff had an AI machine named KITT driving his Pontiac. The car had been augmented with plenty of LEDs in all possible sizes performing flashy effects. In particular, it had a display that scanned back and forth across a line, as shown in this exciting fight between KITT and KARR. This tutorial will show you how to duplicate the KITT display.

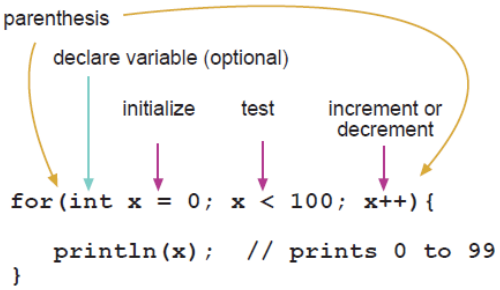
I - Things you will need

You will need an UNO, a mini USB cable, a computer, a breadboard, 6 LEDs of any color, 6 330ohm resistors, and 13 jumper cables.



II - The For loop explained

The for loop is used to repeat a block of statements enclosed in curly braces. An increment counter is usually used to increment and terminate the loop. The for loop is useful for any repetitive operation, and is often used in combination with arrays to operate on collections of data/pins.



The initialization happens first. Then each time through the loop, the condition is tested; if it's true, the statement block, and the increment is executed, then the condition is tested again. When the condition becomes false, the loop ends.

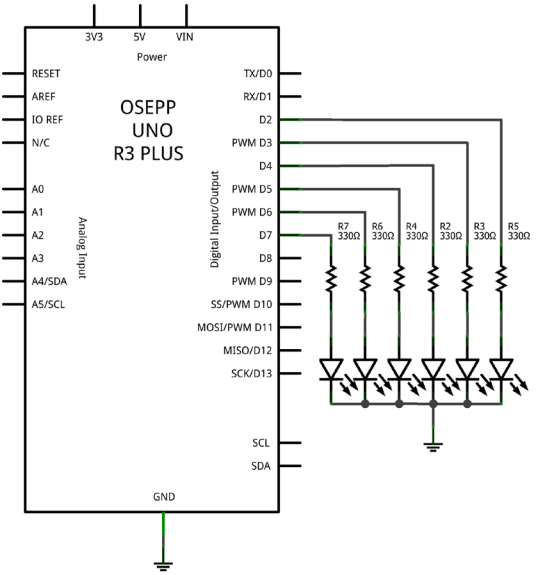
III - Background information

There's nothing new here, we're going to control the LEDs the same way we've done previously. But this time we're going to do it using a for loop.

LEDs

IV – Schematic Diagram

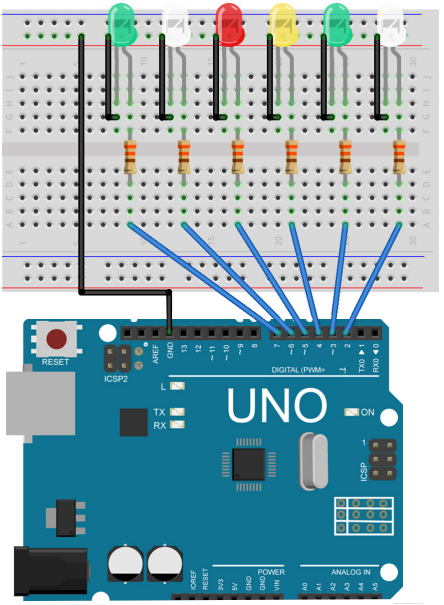
Here's the schematic diagram.



**Images are developed using Fritzing

V – Wiring the Breadboard

Let's wire up the breadboard. If you did the previous tutorial just add two more LEDs and two more resistors.



**Images are developed using Fritzing

VI – Writing the Sketch

Start a new project and load up the following sketch. If you imported our library (refer to page 14) you'll be able to load it up in the Arduino environment by going:

File » Examples » OSEPP_201K » LED_Knight_Rider

```
/*
 * Tutorial 3b: LED Knight Rider
 *
 * Demonstrates the use of a for() loop.
 * Lights multiple LEDs in sequence, then in reverse.
 *
 * The circuit:
 *  - 6 LEDs attached to digital pins 2 through 7 with 330ohm resistors
 *
 * created 2006
 * by David A. Mellis
 * modified 30 Aug 2011
 * by Tom Igoe
 * modified 14 August 2013
 * by Blaise Jarrett
 *
 * This example code is in the public domain.
 *
 * Derivative work from:
 * http://www.arduino.cc/en/Tutorial/ForLoop
 *
 */

// The time in ms each LED stays on for
// experiment with this number, the lower the number
// the faster the LEDs change
int timer = 100;

void setup()
{
    // use a for loop to initialize each pin as an output
    for (int thisPin = 2; thisPin < 8; thisPin++)
    {
        pinMode(thisPin, OUTPUT);
    }
}
```

```
37
38 void loop()
39 {
40     // loop from the lowest pin to the highest
41     for (int thisPin = 2; thisPin < 8; thisPin++)
42     {
43         // turn the pin on
44         digitalWrite(thisPin, HIGH);
45         // wait to turn it off so we can see it
46         delay(timer);
47         // turn the pin off
48         digitalWrite(thisPin, LOW);
49     }
50
51     // loop from the highest pin to the lowest
52     for (int thisPin = 7; thisPin > 1; thisPin--)
53     {
54         // turn the pin on
55         digitalWrite(thisPin, HIGH);
56         // wait to turn it off so we can see it
57         delay(timer);
58         // turn the pin off
59         digitalWrite(thisPin, LOW);
60     }
61 }
```

Once loaded you'll see the LEDs light up in sequence back and forth.
Play with the timer value to adjust the speed of the effect.

VII - Sketch explained

On line 31 we use the for loop to initialize each LED pin as an output. The loop starts at 2, and ends after running 7. The execution flow looks like this:

Initialize thisPin to 2
check if 2 is less than 8? -> run statement with 2 -> increment thisPin
check if 3 is less than 8? -> run statement with 3 -> increment thisPin
....
check if 7 is less than 8? -> run statement with 7 -> increment thisPin
check if 8 is less than 8? No, exit for loop

Line 40 has the same for loop, but line 51 is a bit different. This one works the same way, but in reverse. The execution flow looks like this:

Initialize thisPin to 7
check if 7 is greater than 1? -> run statement with 7 -> decrement thisPin
check if 6 is greater than 1? -> run statement with 6 -> decrement thisPin
...
check if 2 is greater than 1? -> run statement with 2 -> decrement thisPin
check if 1 is greater than 1? No, exit for loop

VI - Review

In this tutorial we've learnt:

- About the for loop and its flow of execution

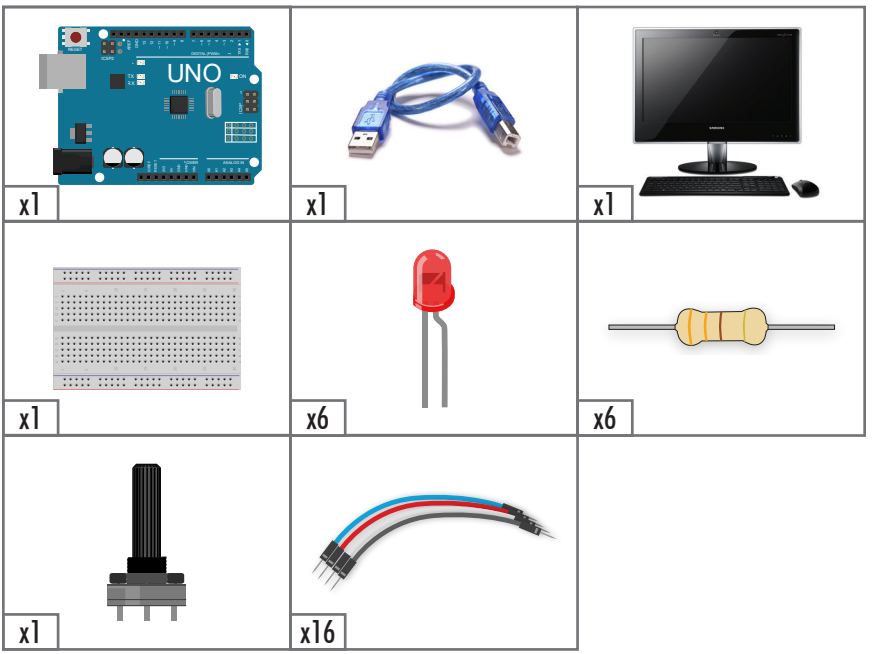
LED Bar Graph

In this tutorial we will learn about Arrays. We can use what we did in the last lesson and add a 10k potentiometer to make a bar graph. We'll use arrays to keep track of the LEDs.

I - Things you will need

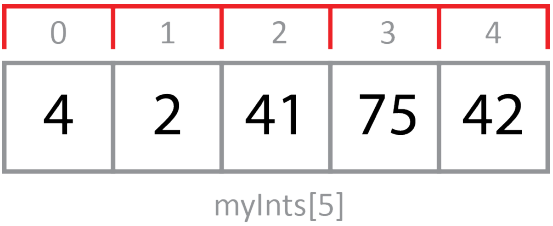
You will need an UNO, USB cable, a computer, a breadboard, 6 LEDs of any color, 6 330ohm resistors, a 10k ohm potentiometer, and 16 jumper cables.

If you did done the last tutorial you'll just need to add the potentiometer.



II - The Array explained

An array is a collection of variables that can be accessed with an index number. Arrays can be multidimensional and get quite complicated. Let's look at a simple, one dimension array.



In the image above each grey box represents an integer. Each number in the red box is the index number used to access the corresponding integer in the grey box below.

An array is declared the same way you would declare any variable "TYPE NAME", but we add square brackets with the size of the array "TYPE NAME[SIZE]".

```
myInts[5];
```

This variable is effectively the same as declaring five separate ints, but with an array we can conveniently access them all with an index number.

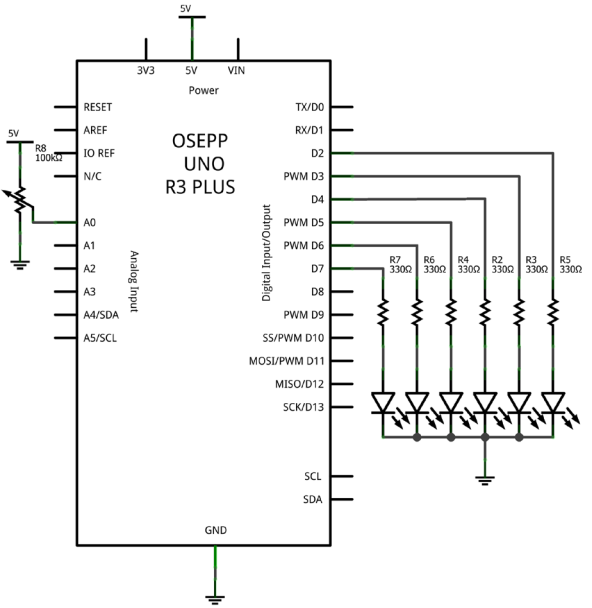
The indexes of arrays are always 0 based. If you have an array of 5 elements, the first index will be 0, and the last index will be 4.

```
myInts[0] // will get you 4 (the first number)
myInts[2] // will get you 41 (the third number)
```

LEDs

III – Schematic Diagram

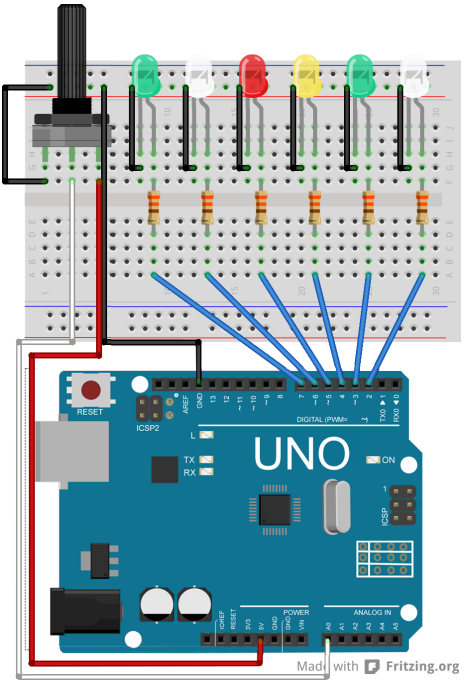
Here's the schematic diagram.



**Images are developed using Fritzing

IV – Wiring the Breadboard

Let's wire up the breadboard. If you did the previous tutorial just add the potentiometer.



**Images are developed using Fritzing

V – Writing the Sketch

Start a new project and load up the following sketch. If you imported our library (refer to page 14) you'll be able to load it up in the Arduino environment by going:

File » Examples » OSEPP_201K » LED_Bar_Graph

This sketch displays the position of your potentiometer on the LEDs as a bar graph.

```
/*
 * Tutorial 3c: LED Knight Rider
 *
 * Turns on a series of LEDs based on the value of an analog sensor.
 * This is a simple way to make a bar graph display. Though this graph
 * uses 6 LEDs, you can use any number by changing the LED count
 * and the pins in the array.
 *
 * The circuit:
 *  - 6 LEDs attached to digital pins 2 through 7 with 330ohm resistors
 *
 * created 4 Sep 2010
 * by Tom Igoe
 * modified 14 August 2013
 * by Blaise Jarrett
 *
 * This example code is in the public domain.
 *
 * Derivative work from:
 * http://www.arduino.cc/en/Tutorial/BarGraph
 *
 */

// the pin that the potentiometer is attached to
int potPin = A0;
// an array of pin numbers to which LEDs are attached
// to add more LEDs just list them here in this array
int ledPins[] = {2, 3, 4, 5, 6, 7};
// the number of LEDs in the bar graph
int ledCount = sizeof(ledPins) / sizeof(ledPins[0]);

void setup()
{
    // use a for loop to initialize each pin as an output
    for (int thisLed = 0; thisLed < ledCount; thisLed++)
    {
        pinMode(ledPins[thisLed], OUTPUT);
    }
}

```

```
41 void loop()
42 {
43     // read the potentiometer
44     int potReading = analogRead(potPin);
45     // map the result to a range from 0 to the number of LEDs
46     int ledLevel = map(potReading, 0, 1023, 0, ledCount);
47
48     // loop over the LED array:
49     for (int thisLed = 0; thisLed < ledCount; thisLed++)
50     {
51         // if the array element's index is less than ledLevel
52         // turn the pin for this element on
53         if (thisLed < ledLevel)
54         {
55             digitalWrite(ledPins[thisLed], HIGH);
56         }
57         // turn off all pins higher than the ledLevel
58         else
59         {
60             digitalWrite(ledPins[thisLed], LOW);
61         }
62     }
63 }
```

Once loaded the LEDs will light up as a bar graph controlled by the position of the potentiometer. Play with the potentiometer to see the bar graph change.

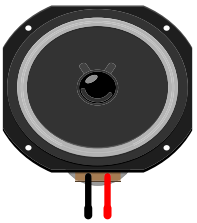
VI - Review

In this tutorial we've learnt:

- About single dimension arrays

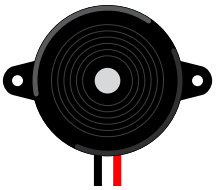
Sound

New Part:



Speaker

New Part:



Buzzer

In this section we learn how to play tones and sense knocks using the speaker and buzzer.

Tutorials include:

1. Melody
2. Pitch Follower
3. Knock Sensor

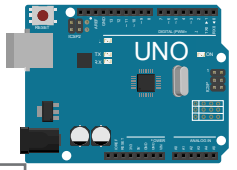
Sound

Melody

In this tutorial we will learn how to play a melody with the UNO and a speaker.

I - Things you will need

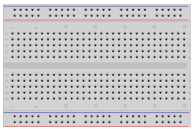
You will need an UNO, USB cable, a computer, a breadboard, a 100 ohm resistor, a speaker, a 2-pin screw terminal, and 2 jumper cables.

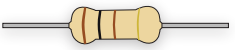


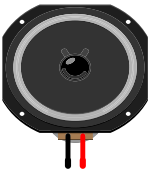
x1


x1


x1


x1


x1


x1


x1


x2

II - Background information

With the Tone() function we can play easily play a tone through a speaker connected to one of the digital pins. The tone function looks like this:

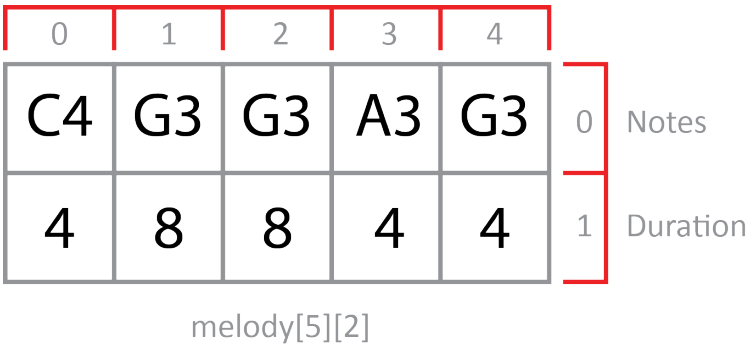
```
tone(pin, frequency)
tone(pin, frequency, duration)
```

Simply call tone with the pin number of the connected speaker, the frequency of the tone, and optionally the duration of the tone to be played. If you don't call tone() with the duration, the tone will play until you call noTone().

In this example we're not going to use noTone() and we're going to pass the duration of our notes as an argument.

III - The Array explained

We're going to use a two dimensional array in this tutorial to store our list of notes for our melody.



Sound

In the previous lesson we used a single dimension array, it was declared like so:
"TYPE NAME[SIZE]"

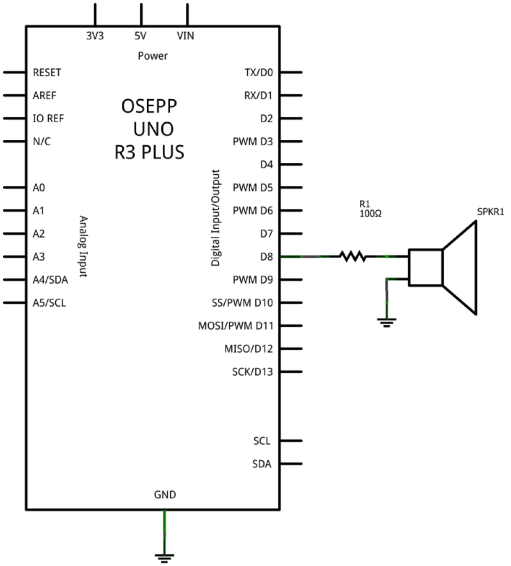
We can add another dimension to our array by simply adding more square brackets like so:
"TYPE NAME[SIZE][SIZE]"

You could describe our above array as having 5 elements of size 2.
Just like the single dimension array we can access each element by placing a 0 based index number in the square brackets after the array name. Refer to the array diagram for these two examples:

```
Melody[4][0] // will get you G3
Melody[3][1] // will get you 4
```

IV – Schematic Diagram

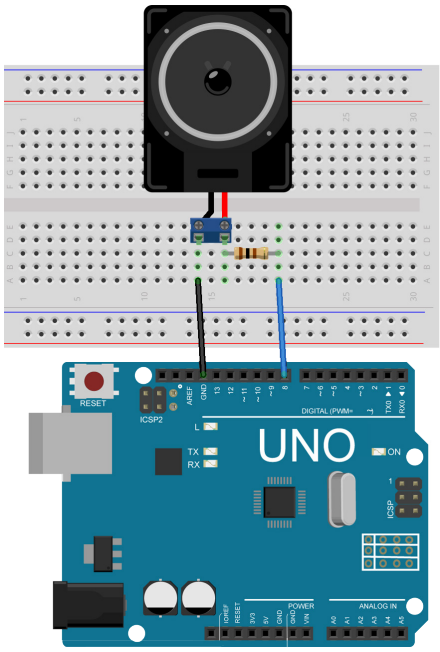
Here's the schematic diagram.



**Images are developed using Fritzing

V – Wiring the Breadboard

Let's wire up the breadboard. Don't worry about the direction of the speaker, either way will work.



**Images are developed using Fritzing

V – Writing the Sketch

Start a new project and load up the following sketch. If you imported our library (refer to page 14) you'll be able to load it up in the Arduino environment by going:

File » Examples » OSEPP_201K » Melody

This sketch plays a melody stored in our two dimensional array "melody".

Sound

```
/*
 * Tutorial 4a: Melody
 *
 * Plays a melody through a speaker.
 *
 * The circuit:
 *  - 8-ohm speaker conected to digital pin 8 through a 100 ohm resistor
 *
 * created 21 Jan 2010
 * modified 30 Aug 2011
 * by Tom Igoe
 * modified 14 August 2013
 * by Blaise Jarrett
 *
 * This example code is in the public domain.
 *
 * Derivative work from:
 * http://arduino.cc/en/Tutorial/Tone
 *
 */

// include our list of note pitches
#include "pitches.h"

// the pin the speaker is attached to
int speakerPin = 8;

// the notes in our melody and their duration in fractions of a second
// e.g. quarter note = 4, eighth note = 8, etc.
const int melody[][2] =
{
    {NOTE_C4, 4},
    {NOTE_G3, 8},
    {NOTE_G3, 8},
    {NOTE_A3, 4},
    {NOTE_G3, 4},
    {NOTE_BLANK, 4},
    {NOTE_B3, 4},
    {NOTE_C4, 4}
};

void setup()
{
    // figure out the number of notes in our melody
    int numberOfNotes = sizeof(melody) / sizeof(melody[0]);

    // iterate over the notes of the melody
    for (int thisNote = 0; thisNote < numberOfNotes; thisNote++)
    {
        // grab our note and note duration from our array
        int thisNoteTone = melody[thisNote][0];
        int thisNoteDuration = melody[thisNote][1];

        // to calculate the note duration in ms
        int noteDurationMS = 1000 / thisNoteDuration;

        // play the note
        tone(speakerPin, thisNoteTone, noteDurationMS);

        // to distinguish the notes, set a minimum time between them.
        // the note's duration + 30% seems to work well:
        delay(noteDurationMS * 1.30);
    }
}

void loop()
{
    // no need to repeat the melody.
    // do nothing
}
```

Play around with the two dimensional array "melody" to play a melody of your own. Look in the file pitches.h for more notes.

Code Hint: Because we declared and initialized our array "melody" (line 30). We don't have to give it the length of the array in the first square brackets. The compiler will figure it out for us based off how many entries we have included.

Code Hint: NOTE_C4 isn't an integer is it? Look in the file pitches.h, you'll see we've "defined" NOTE_C4 as 262. When our sketch is compiled the text NOTE_C4 is replaced with the number 262.

VI - Review

In this tutorial we've learnt:

- About two dimension arrays
- How to use the Tone() function to play a melody

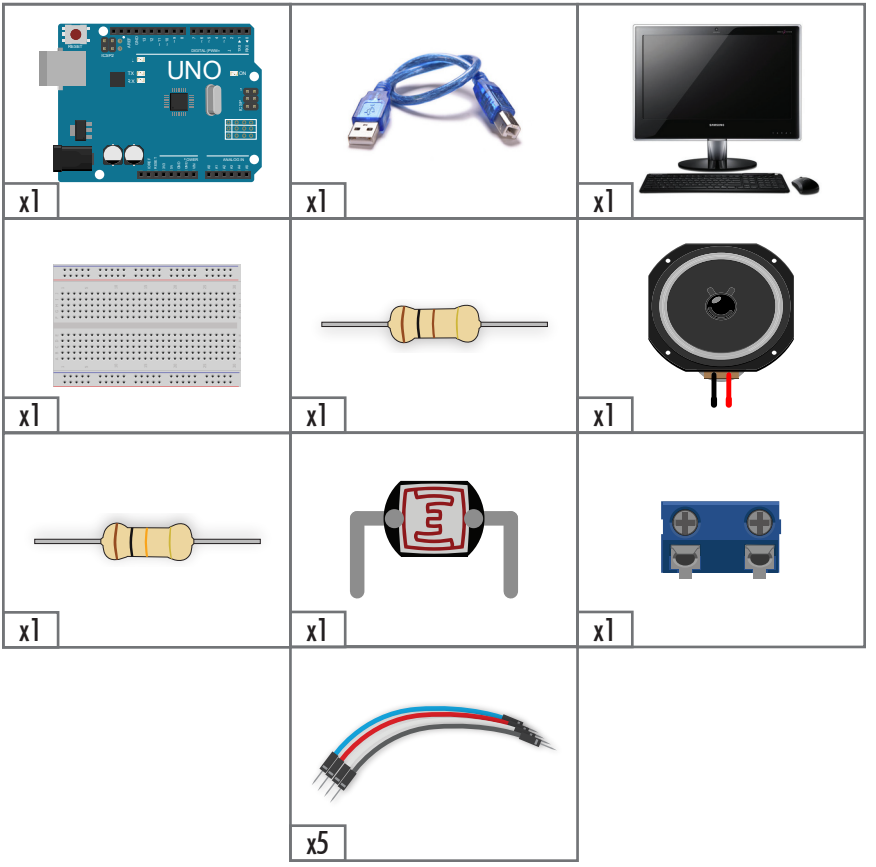
Sound

Pitch Follower

In this tutorial we will learn how to play a varying pitch determined by the light hitting your photocell. Cover the light on the photocell and the pitch played will go lower, shine more light and it will go higher.

I - Things you will need

You will need an Uno, USB cable, a computer, a breadboard, a 100 ohm resistor, a speaker, a 10k ohm resistor, a photocell, a 2-pin screw terminal, and 5 jumper cables.

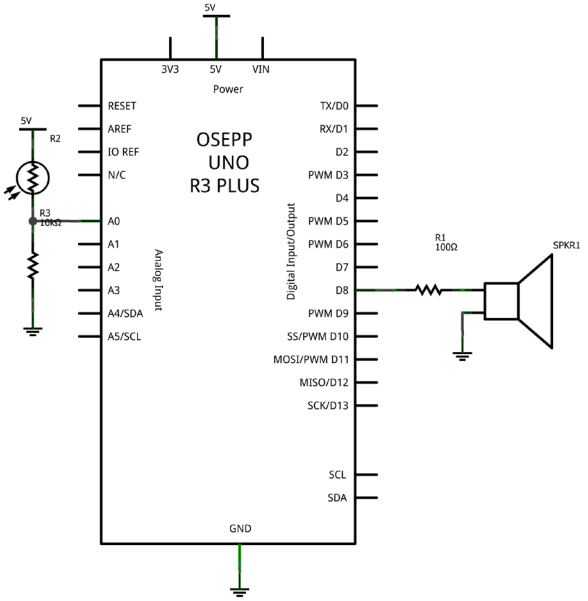


II - Background information

There isn't much we haven't seen here. We're combining our last tutorial on the Tone() function with the "Sensing Light" tutorial from earlier in this book.

III – Schematic Diagram

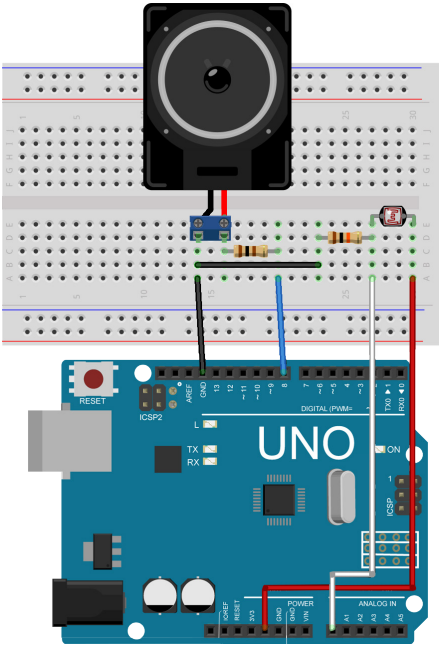
Here's the schematic diagram.



**Images are developed using Fritzing

IV – Wiring the Breadboard

Let's wire up the breadboard. If you've already done the previous tutorial you'll just need to add the photocell and resistor.



**Images are developed using Fritzing

V – Writing the Sketch

Start a new project and load up the following sketch. If you imported our library (refer to page 14) you'll be able to load it up in the Arduino environment by going:

File » Examples » OSEPP_201K » Pitch_Follower

This sketch plays a varying pitch controlled by the light hitting the photocell. Move your hand around the photocell, or even try shining a flashlight onto the photocell.

```
/*
 * Tutorial 4b: Pitch Follower
 *
 * Plays a varying pitch through the speaker. The pitch follows the light
 * present on the photocell. Cover the light and the pitch will go lower,
 * shine more light and it will go higher.
 *
 * The circuit:
 * - 8-ohm speaker conected to digital pin 8 through a 100 ohm resistor
 * - photoresistor from analog in 0 to +5V
 * - 10K resistor from analog in 0 to ground
 *
 * created 21 Jan 2010
 * modified 31 May 2012
 * by Tom Igoe, with suggestion from Michael Flynn
 * modified 14 August 2013
 * by Blaise Jarrett
 *
 * This example code is in the public domain.
 *
 * Derivative work from:
 * http://arduino.cc/en/Tutorial/Tone2
 *
 */

// the pin the speaker is attached to
int speakerPin = 8;
// the photocell voltage divider pin
int photocellPin = A0;

// sensor minimum, discovered through experiment
const int sensorMin = 0;
// sensor maximum, discovered through experiment
const int sensorMax = 600;
// the lowest pitch possible
const int lowestPitch = 150;
// the highest pitch possible
const int highestPitch = 1000;

void setup()
{
    // nothing to do here
}

void loop()
{
    int analogValue;
    int pitch;

    // read our photocell
    analogValue = analogRead(photocellPin);

    // map the analog input range
    // to the output pitch range
    // play with the constants up above
    pitch = map(analogValue, sensorMin,
                sensorMax, lowestPitch, highestPitch);

    // play the pitch
    tone(speakerPin, pitch);
}
```

Sound

Try playing around with constants at the top of the pitch to change the light response.

VI - Review

In this tutorial we've learnt:

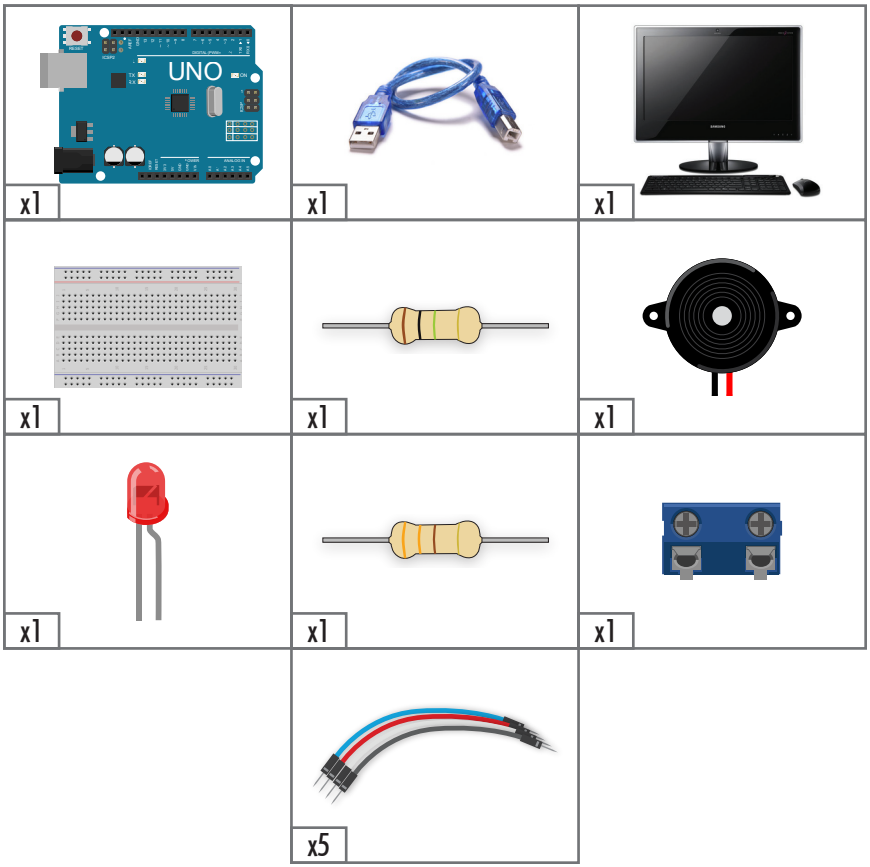
- How to map light input from our photocell to a pitch played through a speaker

Knock Sensor

This tutorial shows you how to use a Piezo element to detect vibration, in this case, a knock on a door, table, or other solid surface.

I - Things you will need

You will need an Uno, USB cable, a computer, a breadboard, a 1 Mohm resistor, a piezo buzzer, a 330 ohm resistor, an LED, a 2-pin screw terminal, and 5 jumper cables.



Sound

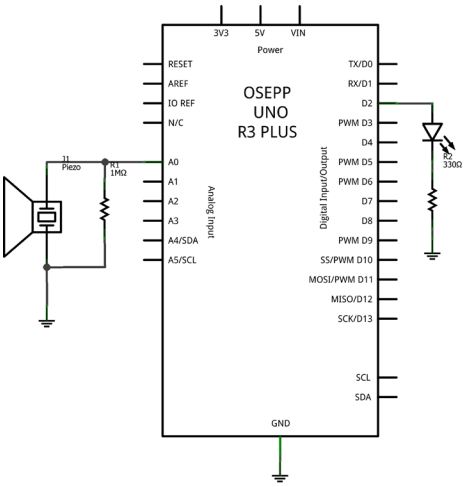
II - Background information

A piezo is an electronic device that generates a voltage when it's physically deformed by a vibration, sound wave, or mechanical strain. Similarly, when you put a voltage across a piezo, it vibrates and creates a tone. Piezos can be used both to play tones and to detect tones.

We can read the output of the Piezo using the analog ports with AnalogRead, just like we've done in previous lessons with other analog sensors. If the sensors output is stronger than a certain threshold we will print "Knock!" on the serial port and light up the LED.

III – Schematic Diagram

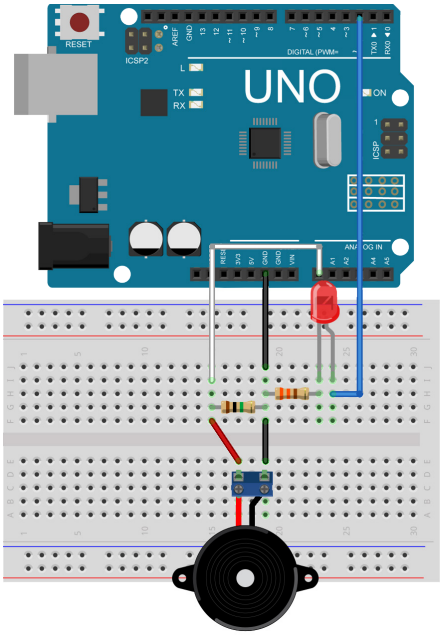
Here's the schematic diagram.



**Images are developed using Fritzing

IV – Wiring the Breadboard

Let's wire up the breadboard.



**Images are developed using Fritzing

V – Writing the Sketch

Start a new project and load up the following sketch. If you imported our library (refer to page 14) you'll be able to load it up in the Arduino environment by going:

File » Examples » OSEPP_201K » Knock_Sensor

This sketch lights the LED and prints "Knock!" to your serial monitor when the piezo buzzer detects a knock. Try placing your board on your desk and giving your desk a whack.

Sound

```
/*
 * Tutorial 4c: Knock Sensor
 *
 * Prints "Knock!" to serial when the piezo buzzer
 * detects a knock. Try placing your board on your desk
 * and giving your desk a whack.
 *
 * The circuit:
 * + connection of the piezo attached to analog in 0
 * - connection of the piezo attached to ground
 * 1 megohm resistor attached from analog in 0 to ground
 *
 * created 25 Mar 2007
 * by David Cuartielles <http://www.0j0.org>
 * modified 30 Aug 2011
 * by Tom Igoe
 * modified 14 August 2013
 * by Blaise Jarrett
 *
 * This example code is in the public domain.
 *
 * Derivative work from:
 * http://www.arduino.cc/en/Tutorial/Knock
 *
 */

// the piezo is connected to analog pin 0
int piezoPin = A0;
// the pin our LED is connected to
int ledPin = 2;
// threshold value to decide when the detected sound is a knock or not
const int threshold = 10;

void setup()
{
    // use the serial port
    Serial.begin(9600);

    // set up our LED
    pinMode(ledPin, OUTPUT);
}

void loop()
{
    int analogValue;

    // read the sensor and store it in the variable sensorReading:
    analogValue = analogRead(piezoPin);

    // if the sensor reading is greater than the threshold:
    if (analogValue > threshold)
    {
        // send the string "Knock!" back to the computer, followed by newline
        Serial.println("Knock!");

        // turn on our LED
        digitalWrite(ledPin, HIGH);
        // we have to wait before turning it off so you can see it light up
        delay(500);
        digitalWrite(ledPin, LOW);
    }

    // delay to avoid overloading the serial port buffer
    delay(100);
}
```

Once loaded into your Uno open up the serial monitor; in the file menu, go:

Tools » Serial Monitor

Knock on your desk, or tap the piezo and you will see "Knock!" printed out in the serial monitor window.

Try adjusting the "threshold" variable to change the sensitivity.

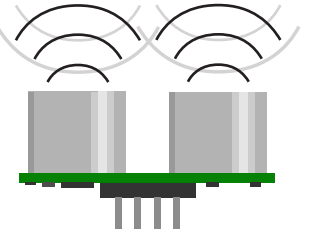
VI - Review

In this tutorial we've learnt:

- How to use a Piezo to detect a knock

Ultrasonic

New Part:



Ultrasonic Range Finder

In this section we learn how to use the Ultrasonic Range Finder.

Tutorials include:

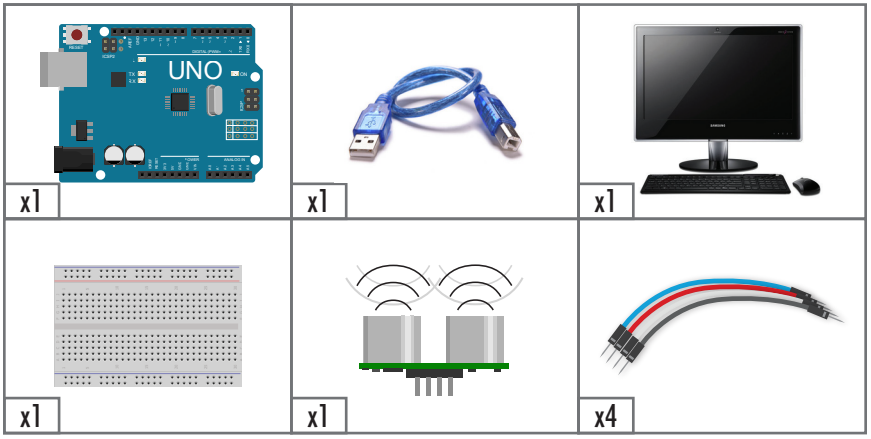
1. Ultrasonic Range Finder

Ultrasonic Range Finder

This tutorial shows you how to use the Ultrasonic Range Finder to measure distance.

I - Things you will need

You will need an Uno, USB cable, a computer, a breadboard, a HC-SR04 Ultrasonic range finder, and 4 jumper cables.



Ultrasonic

II - Background information

The HC-SR04 Ultrasonic range finder detects the distance of the closest object in front of the sensor (from 2 cm up to 3m). It works by sending out a burst of ultrasound and listening for the echo when it bounces off of an object. The Arduino board sends a short pulse to trigger the detection on the "Trig" pin, and then listens for a pulse in the "Echo" pin using the pulseIn() function. The duration of this echo pulse is equal to the time taken by the ultrasound to travel to the object and back to the sensor. Using the speed of sound, this time can be converted to distance.

To get the distance in centimeters from the pulse length we're going to have to do some math:

The speed of sound travels at 340.29 m/s. Let's convert that to centimeters:

340.29m/s x 100cm/m = 34,029cm/s

We now know how far the speed of sound will travel in centimeters in one second. We want to know how long it will take sound to travel one centimeter. We need the inverse.

1/34029 = 0.000029387 s/cm

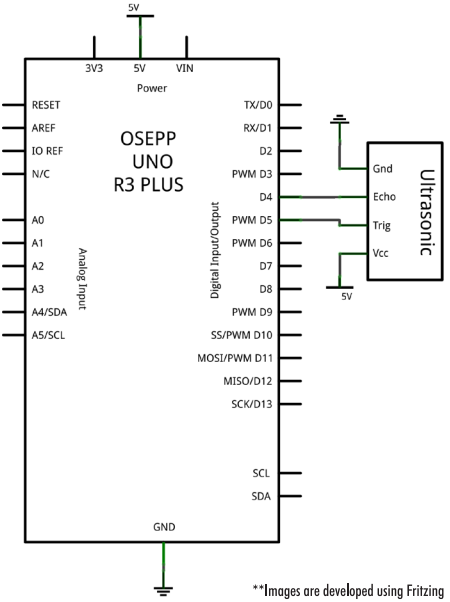
We're going to work in microseconds, not seconds, lets convert that to microseconds:

0.000029387 s/cm * 1,000,000micros/s = 29.387 micros/cm

Every 29.387 microseconds sound will travel one centimeter. We'll use this information in our sketch to calculate distance from time measured.

III – Schematic Diagram

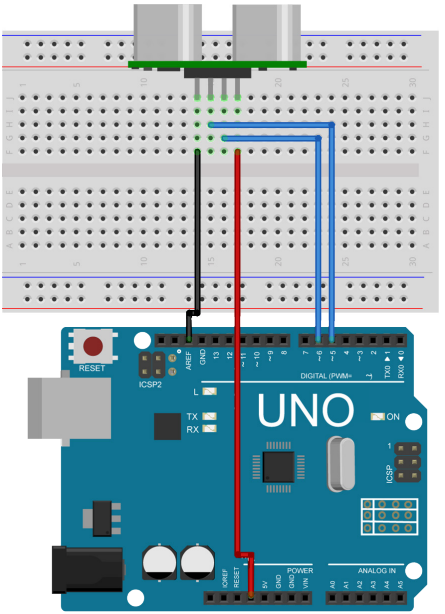
Here's the schematic diagram.



**Images are developed using Fritzing

IV – Wiring the Breadboard

Let's wire up the breadboard.



**Images are developed using Fritzing

V – Writing the Sketch

Start a new project and load up the following sketch. If you imported our library (refer to page 14) you'll be able to load it up in the Arduino environment by going:

File » Examples » OSEPP_201K » Ultrasonic_Range_Finder

This sketch prints the distance in centimeters to the closest object in front of the ultrasonic range finder.

```
/*
 * Tutorial 5: Ultrasonic Range Finder
 *
 * Prints the distance measured from the
 * range finder over serial. Place the board on your desk
 * and open up the serial monitor. Move an object infront of the
 * ultrasonic sensor and you'll see the distance to the object
 * printed out over serial.
 *
 * The circuit:
 * - 5v, ground connected to Ultrasonic sensor
 * - digital pin 4 conneceted to Ultrasonic sensor echo pin
 * - digital pin 5 conneceted to Ultrasonic sensor trig pin
 *
 * created 3 Nov 2008
 * by David A. Mellis
 * modified 30 Aug 2011
 * by Tom Igoe
 * modified 14 August 2013
 * by Blaise Jarrett
 *
 * This example code is in the public domain.
 *
 * Derivative work from:
 * http://www.arduino.cc/en/Tutorial/Ping
 *
 */

// the pins connected to the Ultrasonic sensor
int echoPin = 4;
int trigPin = 5;

void setup()
{
    // set up serial
    Serial.begin(9600);
    // set the pinmode on our ultrasonic echo, and tric pins
    pinMode(echoPin, INPUT);
    pinMode(trigPin, OUTPUT);
}

```

```
42 void loop()
43 {
44     float distanceCentimeters;
45     int pulseLenMicroseconds;
46
47     // bit-bang a small square wave
48     // on the trig pin to start the range
49     // finder
50     digitalWrite(trigPin, LOW);
51     delayMicroseconds(20);
52     digitalWrite(trigPin, HIGH);
53     delayMicroseconds(100);
54     digitalWrite(trigPin, LOW);
55
56     // measure the pulse length from the echo pin
57     pulseLenMicroseconds = pulseIn(echoPin, HIGH);
58
59     // calculate the distance using the speed of sound
60     distanceCentimeters = pulseLenMicroseconds / 29.387 / 2;
61
62     // print it out over serial
63     Serial.println(distanceCentimeters);
64
65     delay(100);
66 }
```

Code Hint: On line 60 why don't we have to convert pulseLen to a float like we did in the first tutorial? Only one of the division operands have to be a floating point type to enforce floating point division. If we used 29 instead of 29.387 it would do integer division and we would end up with a whole number.

Code Hint: On line 60 why do we divide by 2? The sound travels twice the distance with the ultrasonic range finder. It has to go to the object, and then back again.

Once loaded into your Uno open up the serial monitor; in the file menu, go:

Tools » Serial Monitor

Place the board on your desk and open up the serial monitor. Move an object in front of the ultrasonic sensor and you'll see the distance to the object printed out over serial.

VI - Review

In this tutorial we've learnt:

- How to use the Ultrasonic Range Finder

Servo Motors

New Part:

SG90 Servo Motor

In this section we learn the basics of how to use servo motors.

Tutorials include:

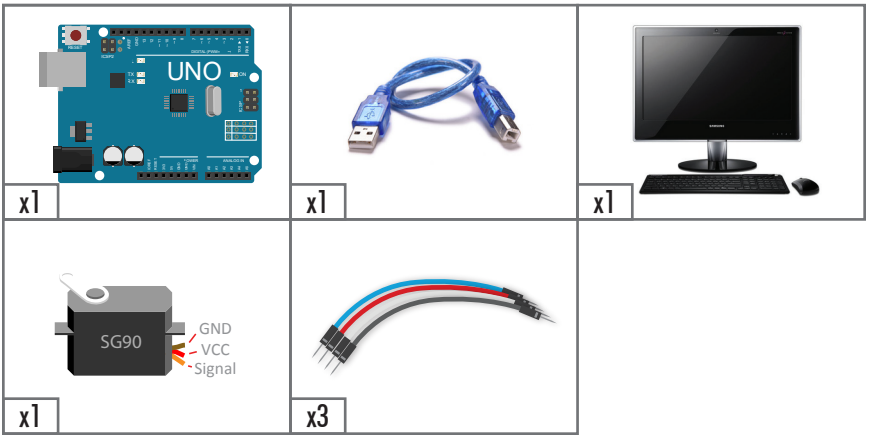
1. Introduction to Servo Motors
2. Servo Motor Control

Introduction to Servo Motors

This tutorial shows you the basics of how to use a servo motor. We're going to write a program to sweep our servo motor back and forth across 180 degrees.

I - Things you will need

You will need an Uno, USB cable, a computer, a SG90 servo motor, and 3 jumper cables.



II - Background information

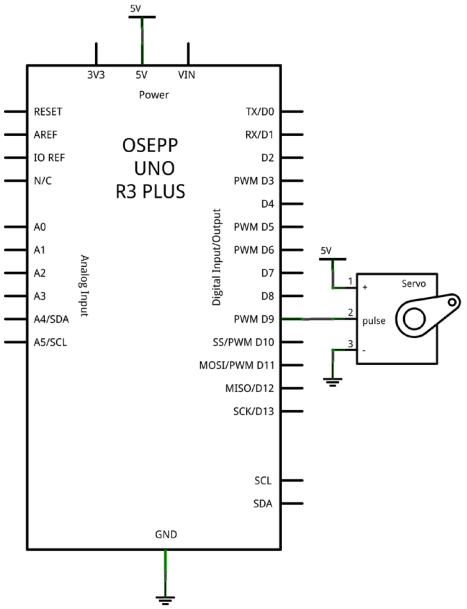
Servo motors are handy little motors that have limited, but accurate motion. They're typically used for applications that require precise, small movement (ex RC car steering).

Typical servo motors only allow you to control the angular position of the motor. They typically rotate 180 degrees, but not always. The included SG90 servo motor rotates 180 degrees.

Servo motors have three wires: power, ground, and signal. The power wire is typically red, and should be connected to the 5V pin on the Arduino board. The ground wire is typically black or brown and should be connected to a ground pin on the Arduino board. The signal pin is typically yellow, orange or white and should be connected to one of the digital pins on the Arduino board.

III – Schematic Diagram

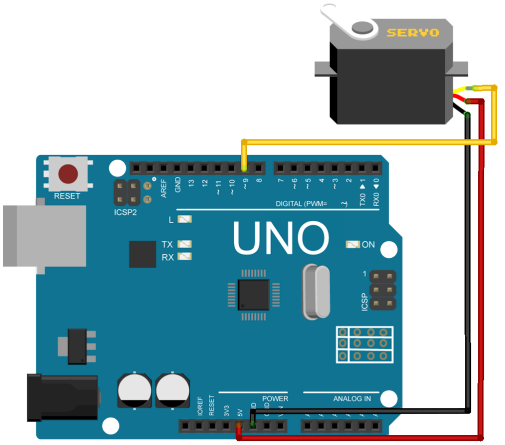
Here's the schematic diagram.



**Images are developed using Fritzing

IV – Wiring the Motor

Let's wire up the Motor. We don't need a breadboard for this tutorial. Connect the motor directly to your Uno using three jumper cables.



**Images are developed using Fritzing

V – Writing the Sketch

Start a new project and load up the following sketch. If you imported our library (refer to page 14) you'll be able to load it up in the Arduino environment by going:

File » Examples » OSEPP_201K » Intro_to_Servos

This sketch rotates the SG90 servo motor back and forth 180 degrees.

```
/*
 * Tutorial 6a: Introduction to Servo Motors
 *
 * Simply rotates your server from 0 to 180 degrees and back.
 *
 * The circuit:
 * - Brown pin to ground
 * - Red pin to 5v
 * - Orange pin to digital pin 9
 *
 * by BARRAGAN <http://barraganstudio.com>
 * modified 14 August 2013
 * by Blaise Jarrett
 *
 * This example code is in the public domain.
 *
 * Derivative work from:
 * http://arduino.cc/en/Tutorial/Sweep
 *
 */

#include <Servo.h>

// the Orange pin is connected to digital pin 9
int servoPin = 9;

// create servo object to control our servo
// a maximum of eight servo objects can be created
Servo myServo;

void setup()
{
    // attaches the servo on pin 9 to the servo object
    myServo.attach(servoPin);
}

void loop()
{
    // move the servo to degree 0
    myServo.write(0);

    // wait for it to move
    delay(1000);

    // move the servo to degree 180
    myServo.write(180);

    // wait for it to move
    delay(1000);
}
```

On line 40 and 46 we simply tell our servo motor what degree to move to. We only have control over the angular position of the servo, we can't control how fast it moves.

VI - Review

In this tutorial we've learnt:

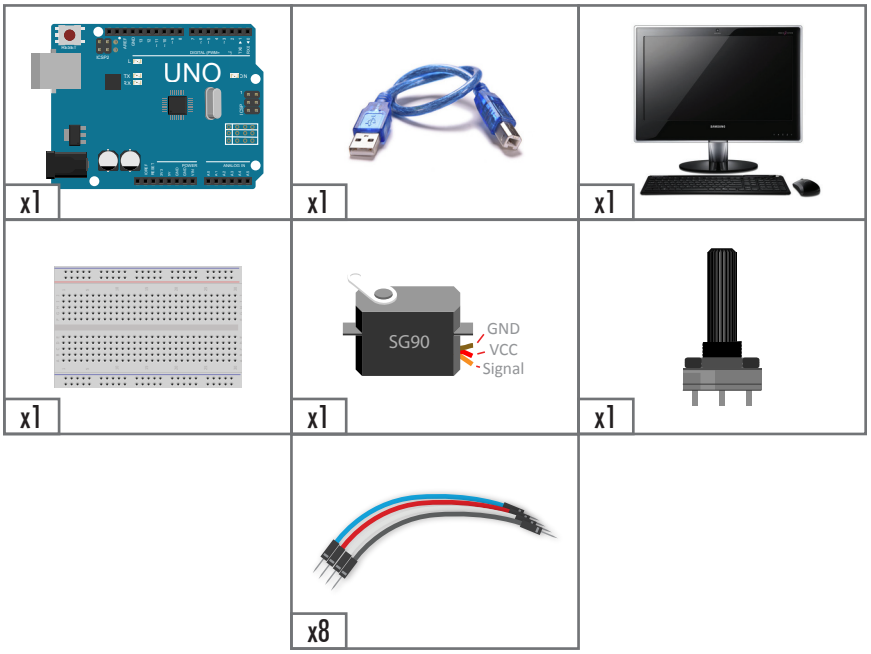
- How to use a servo motor

Servo Motor Control

This tutorial extends on the previous one by adding a potentiometer. This time you'll be able to control the position of the servo motor with the potentiometer.

I - Things you will need

You will need an Uno, USB cable, a computer, a breadboard, a SG90 servo motor, a 10k ohm potentiometer, and 8 jumper cables.



Servo Motors

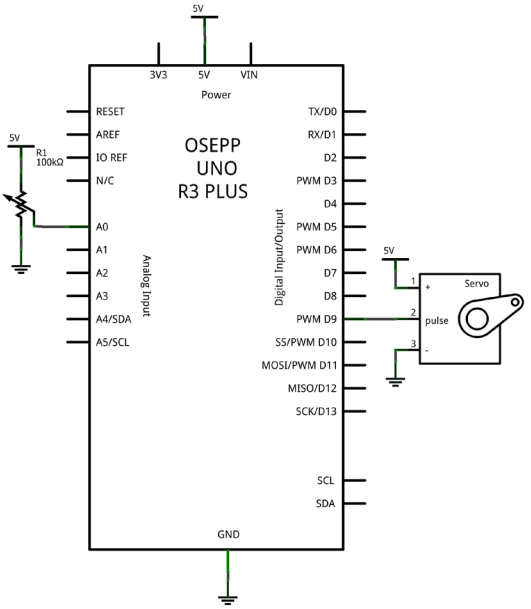
II - Background information

We're not doing much new here. We've used the potentiometer in previous tutorials for analog input. We're going to do the same here, but this time use it to control the servo motor.

Check the previous tutorial for more information on servo motors.

III – Schematic Diagram

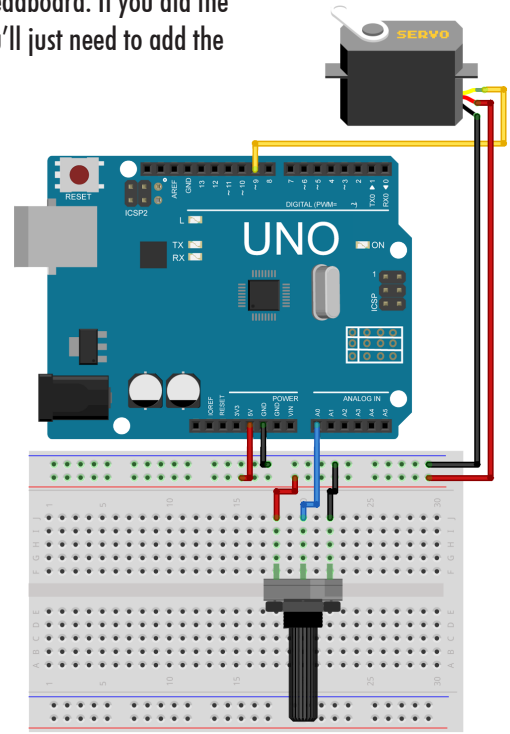
Here's the schematic diagram.



**Images are developed using Fritzing

IV – Wiring the Breadboard

Let's wire up the breadboard. If you did the previous tutorial you'll just need to add the potentiometer.



**Images are developed using Fritzing

V – Writing the Sketch

Start a new project and load up the following sketch. If you imported our library (refer to page 14) you'll be able to load it up in the Arduino environment by going:

File » Examples » OSEPP_201K » Servo_Control

This sketch will rotate the servo motor following the potentiometer.

Servo Motors

```
/*
 * Tutorial 6b: Servo Control
 *
 * Control the position of your servo using
 * a potentiometer.
 *
 * The circuit:
 * - Servo Brown pin to ground
 * - Servo Red pin to 5v
 * - Servo Orange pin to digital pin 9
 * - Potentiometer connected to 5v, gnd as a voltage divider
 * - Potentiometer wiper connected to Analog pin 0
 *
 * by Michal Rinott <http://people.interaction-ivrea.it/m.rinott>
 * modified 14 August 2013
 * by Blaise Jarrett
 *
 * This example code is in the public domain.
 *
 * Derivative work from:
 * http://arduino.cc/en/Tutorial/Knob
 *
 */

#include <Servo.h>

// the Orange pin is connected to digital pin 9
int servoPin = 9;
// analog pin used to connect the potentiometer
int potPin = A0;

// create servo object to control our servo
// a maximum of eight servo objects can be created
Servo myServo;

void setup()
{
    // attaches the servo on pin 9 to the servo object
    myServo.attach(servoPin);
}

void loop()
{
    int analogValue;
    int position;

    // reads the value of the potentiometer (value between 0 and 1023)
    analogValue = analogRead(potPin);

    // scale it to use it with the servo (value between 0 and 180)
    position = map(analogValue, 0, 1023, 0, 179);

    // sets the servo position according to the scaled value
    myServo.write(position);

    // waits for the servo to get there
    delay(15);
}
```

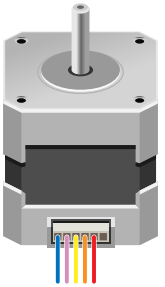
VI - Review

In this tutorial we've learnt:

- How to use a servo motor with a potentiometer

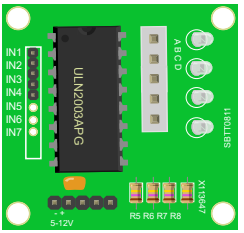
Stepper Motors

New Part:



28BYJ-48 Stepper Motor

New Part:



Stepper Motor Driver

In this section we learn the basics of how to use stepper motors.

Tutorials include:

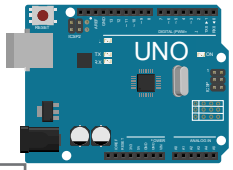
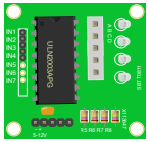
1. Introduction to Steppers
2. Stepper Control

Introduction to Stepper Motors

This tutorial shows you the basics of how to use a Stepper motor. We're going to write a program to have our stepper motor rotate one revolution in forward, and then one revolution back.

I - Things you will need

You will need an Uno, USB cable, a computer, a 28BYJ-48 stepper motor and stepper driver, and 6 F/M jumper cables.

 x1	 x1	 x1
 x1	 x1	 x6

Stepper Motors

II - Background information

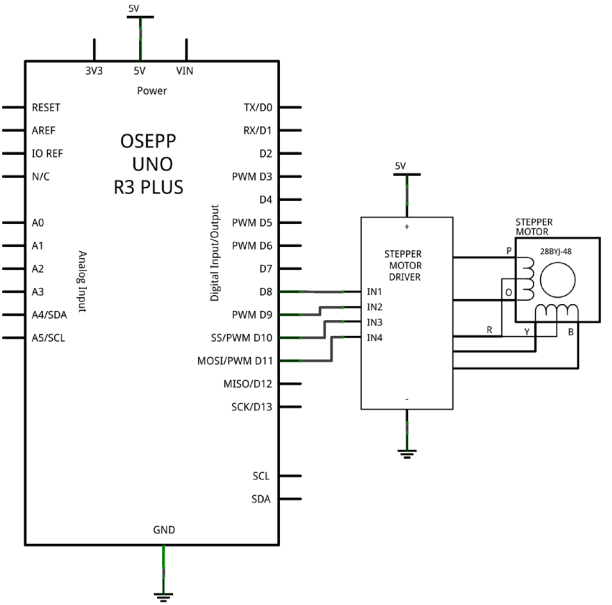
Stepper motors, due to their unique design, can be controlled to a high degree of accuracy without any feedback mechanisms. The shaft of a stepper, mounted with a series of magnets, is controlled by a series of electromagnetic coils that are charged positively and negatively in a specific sequence, precisely moving it forward or backward in small "steps".

Unlike our servo this stepper can rotate continuously. We have control over velocity, and angular position.

We're using a Unipolar stepper motor. You can see the coil wiring in the schematic below.

III – Schematic Diagram

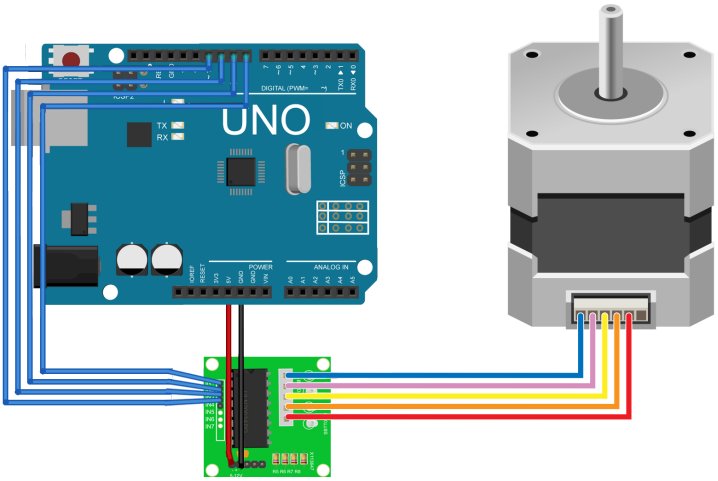
Here's the schematic diagram.



**Images are developed using Fritzing

IV – Wiring the Motor

Let's wire up the Motor. We don't need a breadboard for this tutorial, connect the motor to the driver board and then connect the driver directly to your Uno.



**Images are developed using Fritzing

V – Writing the Sketch

Start a new project and load up the following sketch. If you imported our library (refer to page 14) you'll be able to load it up in the Arduino environment by going:

File » Examples » OSEPP_201K » Intro_to_Steppers

This sketch will rotate the stepper motor one revolution forward and then one revolution backwards.

Stepper Motors

```
/*
 * Tutorial 7a: Introduction to Stepper Motors
 *
 * Simply rotates your stepper from 360 degrees forward
 * and back.
 *
 * The circuit:
 * - Stepper driver powered with 5v, GND
 * - D8-D11 connected to IN1-IN4 on the stepper driver
 *
 * by Blaise Jarrett
 *
 * This example code is in the public domain.
 *
 *
 */

#include <Stepper.h>

int stepIN1Pin = 8;
int stepIN2Pin = 9;
int stepIN3Pin = 10;
int stepIN4Pin = 11;

int stepsPerRevolution = 2048;

Stepper myStepper(stepsPerRevolution,
                  stepIN1Pin, stepIN3Pin,
                  stepIN2Pin, stepIN4Pin);

void setup()
{
    // set the RPM
    myStepper.setSpeed(6);
}

void loop()
{
    // step one revolution in one direction
    myStepper.step(stepsPerRevolution);
    // wait a second
    delay(1000);

    // step one revolution in the other direction
    myStepper.step(-stepsPerRevolution);
    // wait a second
    delay(1000);
}
```

VI - Review

In this tutorial we've learnt:

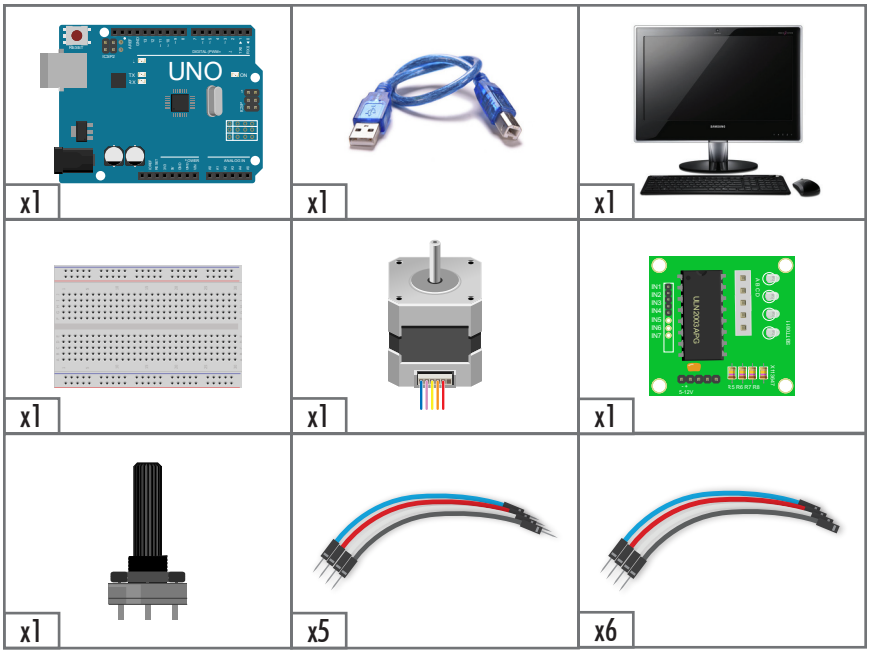
- How to use a stepper motor

Stepper Motor Control

Like the tutorial “Servo Motor Control” this tutorial extends on the Stepper Introduction by adding some user control with a potentiometer.

I - Things you will need

You will need an Uno, USB cable, a computer, a breadboard, a 28BYJ-48 stepper motor and stepper driver, a 10k ohm potentiometer, 5 M/M jumper cables, and 6 F/M jumper cables.



Stepper Motors

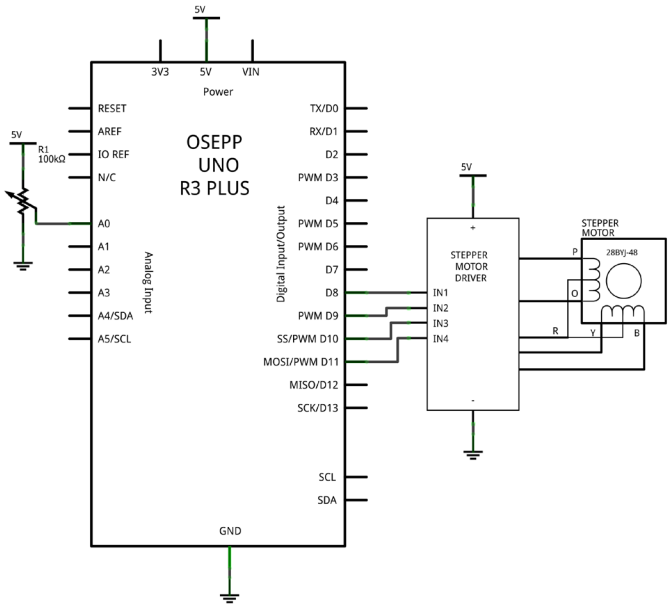
II - Background information

We're not doing much new here. We've used the potentiometer in previous tutorials for analog input. We're going to do the same here, but this time use it to control the stepper motor.

Check the previous tutorial for more information on stepper motors.

III – Schematic Diagram

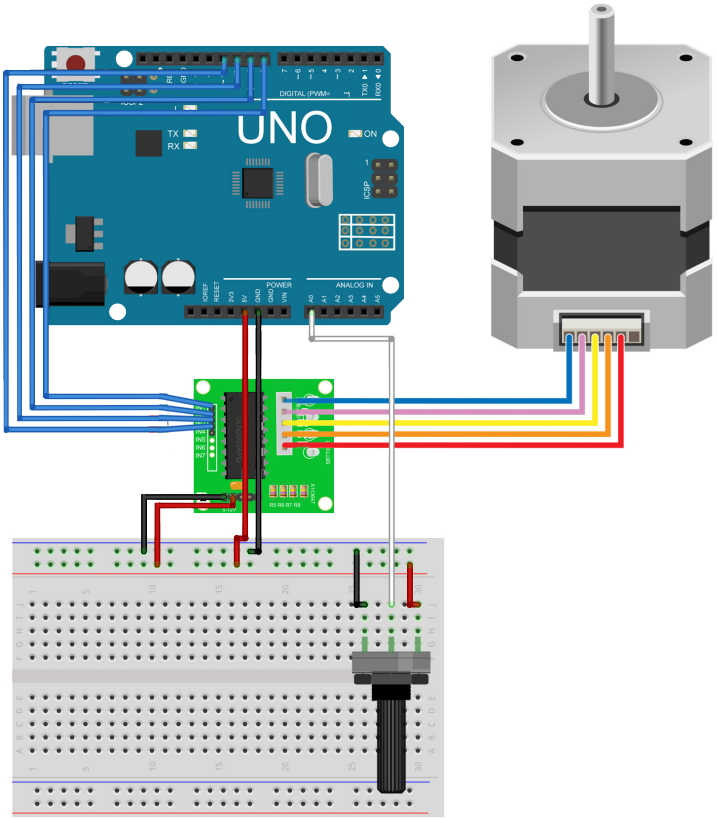
Here's the schematic diagram.



**Images are developed using Fritzing

IV – Wiring the Breadboard

Let's wire up the breadboard. If you did the previous tutorial you'll just need to add the potentiometer.



**Images are developed using Fritzing

V – Writing the Sketch

Start a new project and load up the following sketch. If you imported our library (refer to page 14) you'll be able to load it up in the Arduino environment by going:

File » Examples » OSEPP_201K » Stepper_Control

This sketch will rotate the stepper motor following the potentiometer.

Stepper Motors

```
/*
 * Tutorial 7b: Stepper Motor Control
 *
 * Stepper motor follows the turns of your potentiometer.
 *
 * The circuit:
 * - Stepper driver powered with 5v, GND
 * - D8-D11 connected to IN1-IN4 on the stepper driver
 * - Potentiometer connected to 5v, gnd as a voltage divider
 * - Potentiometer wiper connected to Analog pin 0
 *
 * modified 14 August 2013
 * by Blaise Jarrett
 *
 * This example code is in the public domain.
 *
 * Derivative work from:
 * http://arduino.cc/en/Tutorial/MotorKnob
 *
 */

#include <Stepper.h>

// the pins we've connected our stepper driver to
int stepIN1Pin = 8;
int stepIN2Pin = 9;
int stepIN3Pin = 10;
int stepIN4Pin = 11;
// analog pin used to connect the potentiometer
int potPin = A0;

// steps required for one full revolution
int stepsPerRevolution = 2048;

// make our stepper object
Stepper myStepper(stepsPerRevolution,
                  stepIN1Pin, stepIN3Pin,
                  stepIN2Pin, stepIN4Pin);

// the previous reading from the analog input
int previous = 0;

void setup()
{
    // set the speed of the motor to 10 RPMs
    myStepper.setSpeed(10);
}

void loop()
{
    int analogValue;
    int position;

    // reads the value of the potentiometer (value between 0 and 1023)
    analogValue = analogRead(potPin);

    // scale it to use it with the stepper
    position = map(analogValue, 0, 1023, 0, stepsPerRevolution * 0.6);

    // Unlike the servo motor, the stepper isn't controled
    // with an absolute position, we need a relative one.

    // move a number of steps equal to the change in the
    // sensor reading
    myStepper.step(position - previous);

    // remember the previous value of the sensor
    previous = position;
}
```

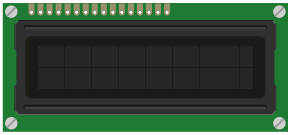
VI - Review

In this tutorial we've learnt:

- How to use a stepper motor with a potentiometer

LCD

New Part:



HD44780 comp. 16x2 LCD

In this section we learn the basics of how to use the HD44780 compatible 16x2 character Liquid Crystal Display.

Tutorials include:

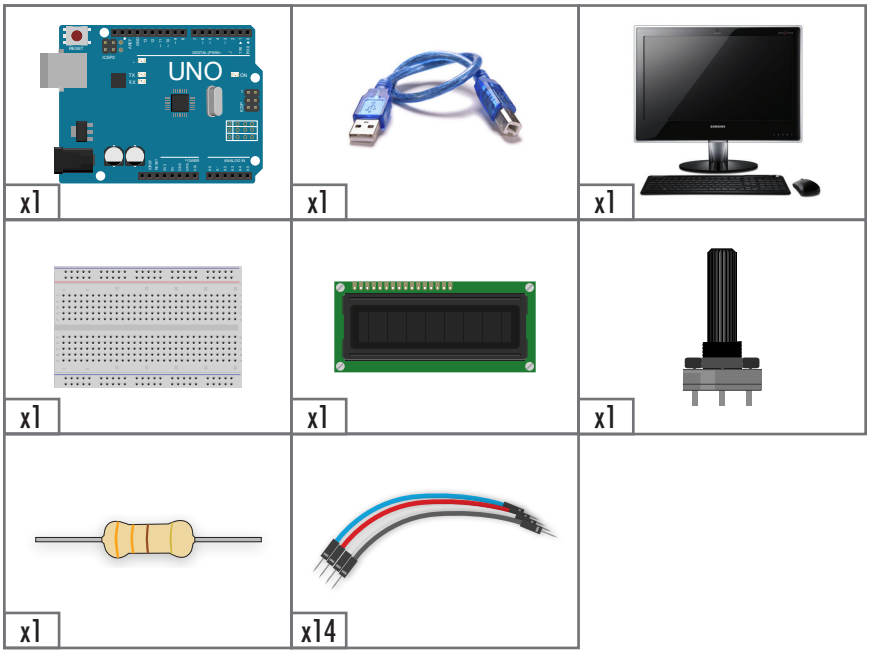
1. Using the LCD

Using the LCD

This tutorial shows you the basics of how to use the included HD44780 compatible Liquid Crystal Display. The display has 16 horizontal characters and 2 lines. With the Arduino LiquidCrystal library we can easily write characters to the display using the library's print function.

I - Things you will need

You will need an Uno, USB cable, a computer, a breadboard, a 16x2 LCD, a 10k ohm potentiometer, a 330 ohm resistor, and 14 jumper cables.



LCD

II - Background information

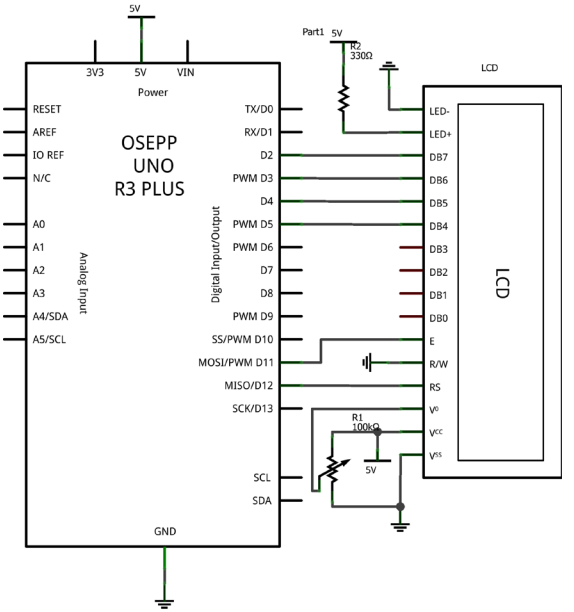
HD44780 compatible Liquid Crystal Displays are very common, and for good reason, they're very easy to use. They use a parallel interface with 8 data lines, and 3 control lines. You can operate the display in either 8-bit or 4-bit mode (by using half the data lines). We're going to wire it up in 4-bit mode.

The built in Arduino Liquid Crystal library will do all hard work for us. The library has a bunch of functions and can be referenced here:

http://arduino.cc/en/Reference/LiquidCrystal

III – Schematic Diagram

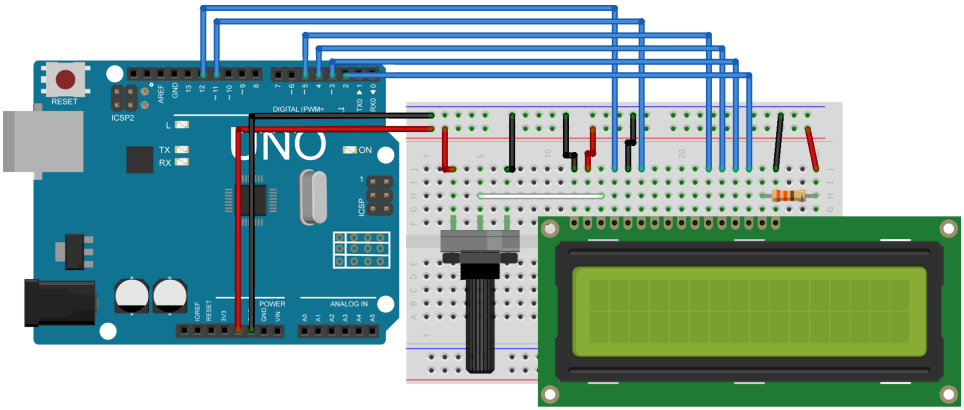
Here's the schematic diagram.



**Images are developed using Fritzing

IV – Wiring the Breadboard

Let's wire up the breadboard.



**Images are developed using Fritzing

V – Writing the Sketch

Start a new project and load up the following sketch. If you imported our library (refer to page 14) you'll be able to load it up in the Arduino environment by going:

File » Examples » OSEPP_201K » LCD

This sketch will print "hello, world!" on the top line, and display a counter on the bottom line. Play with the potentiometer to adjust the contrast until you can see the characters clearly.

```
/*
 * Tutorial 8: Using the LCD
 *
 * Demonstrates the use a 16x2 LCD display.  The LiquidCrystal
 * library works with all LCD displays that are compatible with the
 * Hitachi HD44780 driver. There are many of them out there, and you
 * can usually tell them by the 16-pin interface.
 *
 * Adjust the LCDs contrast with the Potentiometer until you
 * can see the characters on the LCD.
```

LCD

```
 *
 * The circuit:
 * - LCD RS pin to digital pin 12
 * - LCD Enable pin to digital pin 11
 * - LCD D4 pin to digital pin 5
 * - LCD D5 pin to digital pin 4
 * - LCD D6 pin to digital pin 3
 * - LCD D7 pin to digital pin 2
 * - LCD R/W pin to ground
 * - 10K potentiometer divider for LCD pin VO:
 * - 330ohm resistor betweenm LCD pin A and 5v
 * - LCD pin K to ground
 *
 * Library originally added 18 Apr 2008
 * by David A. Mellis
 * library modified 5 Jul 2009
 * by Limor Fried (http://www.ladyada.net)
 * example added 9 Jul 2009
 * by Tom Igoe
 * modified 22 Nov 2010
 * by Tom Igoe
 * modified 14 August 2013
 * by Blaise Jarrett
 *
 * This example code is in the public domain.
 *
 * Derivative work from:
 * http://www.arduino.cc/en/Tutorial/LiquidCrystal
 *
 */

// include the library
#include <LiquidCrystal.h>

// all of our LCD pins
int lcdRSPin = 12;
int lcdEPin = 11;
int lcdD4Pin = 5;
int lcdD5Pin = 4;
int lcdD6Pin = 3;
int lcdD7Pin = 2;

// initialize the library with the numbers of the interface pins
LiquidCrystal lcd(lcdRSPin, lcdEPin,
                  lcdD4Pin, lcdD5Pin, lcdD6Pin, lcdD7Pin);

void setup()
{
    // set up the LCD's number of columns and rows:
    lcd.begin(16, 2);

    // Print a message to the LCD.
    lcd.print("hello, world!");
}

void loop()
{
    // set the cursor to column 0, line 1
    // (note: line 1 is the second row, since counting begins with 0)
    lcd.setCursor(0, 1);

    // print the number of seconds since reset
    lcd.print(millis() / 1000);
}
```

VI - Review

In this tutorial we've learnt:

- How to use the HD44780 compatible LCD

Project Ideas

Now that you've done all the tutorials in this book you know how to:

- Read temperature
- Read brightness
- Control LEDs
- Play sounds
- Read distance
- Use Servos
- Use Motors
- Use the character LCD

You can combine many of these skills to build new things. Here's a few ideas:

- Frequency generator
- Standalone temperature monitor with LCD
- Standalone range finder with LCD
- Door alarm

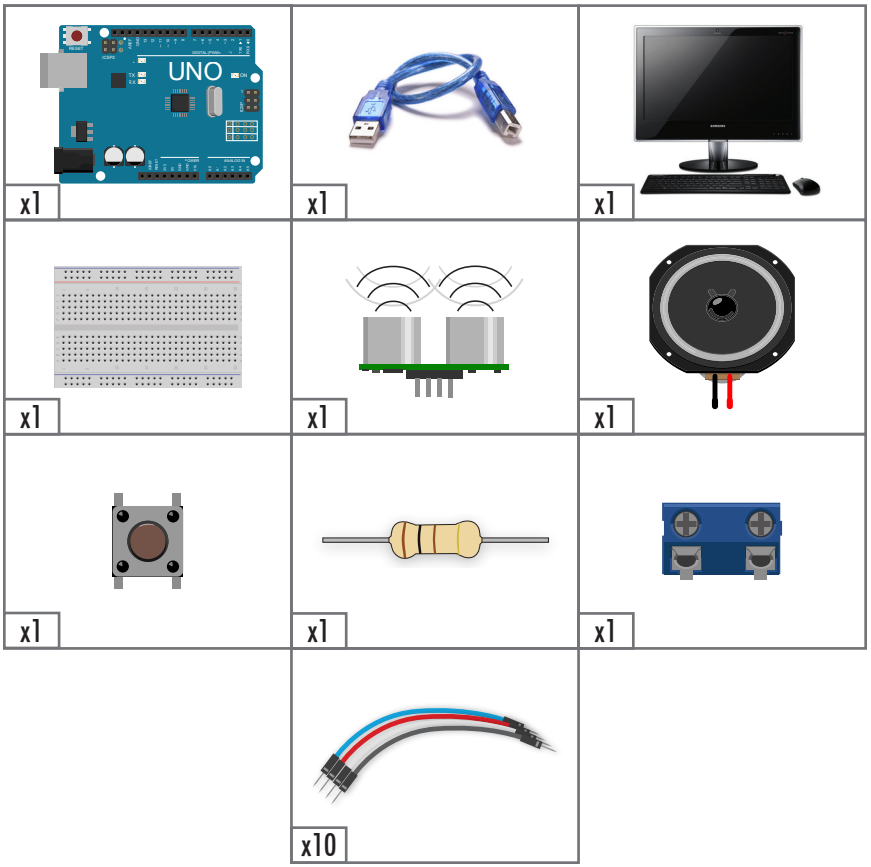
We've included instructions on how to build a door alarm on the next page, but see if you can build it yourself first!

The door alarm is a combination of the Melody and Ultrasonic Range Finder tutorials. The range finder will measure the distance to your door, when that distance changes enough (you've opened the door) we sound the alarm.

Door Alarm

I - Things you will need

You will need an Uno, USB cable, a computer, a breadboard, an Ultrasonic Range Finder, a speaker, a push button, a 100 ohm resistor, a 2-pin screw terminal, and 10 jumper cables.



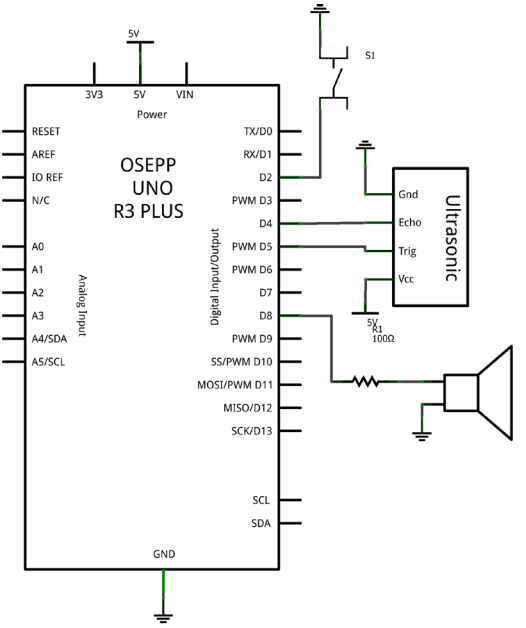
Project Ideas

II - Background information

This tutorial is a combination of the Melody and Ultrasonic Range Finder tutorials. We're going to copy/paste pieces of those tutorials into our sketch and wrap them in functions to make things a little bit neater. After that we'll just add a little bit of logic to detect the change in distance and sound the alarm.

III – Schematic Diagram

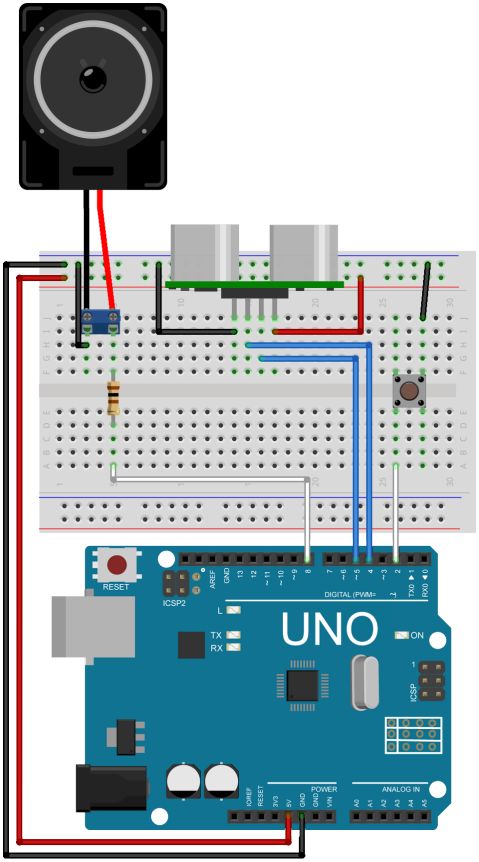
Here's the schematic diagram.



**Images are developed using Fritzing

IV – Wiring the Breadboard

Let's wire up the breadboard.



**Images are developed using Fritzing

V – Writing the Sketch

Start a new project and load up the following sketch. If you imported our library (refer to page 14) you'll be able to load it up in the Arduino environment by going:

File » Examples » OSEPP_201K » Door_Alarm

This sketch will sound an alarm when it measures a change in distance. Place the board about infront of your door, but far enough to stay out of the way when the door opens. Press the push button to record the distance. Open the door and the alarm will sound until the door is closed.

```
/*
 * Tutorial 9: Door Alarm
 *
 * Plays an alarm when the distance to the ultrasonic
 * range finder changes more then a preset tolerance.
 * Place infront of a door and press the push button to set
 * the alarm distance. Opening the door will now sound
 * the alarm, the alarm won't stop until the door is closed.
 *
 * The circuit:
 *  - 8-ohm speaker conected to digital pin 8 through a 100 ohm resistor
 *  - Push button connected to digital pin 2 and ground
 *  - 5v, ground connected to Ultrasonic sensor
 *  - digital pin 4 conneceted to Ultrasonic sensor echo pin
 *  - digital pin 5 conneceted to Ultrasonic sensor trig pin
 *
 * created August 26 2010
 * by Blaise Jarrett
 *
 * This example code is in the public domain.
 *
 * Derivative work from:
 * http://arduino.cc/en/Tutorial/Tone
 * http://www.arduino.cc/en/Tutorial/Ping
 *
 */

// include our list of note pitches
#include "pitches.h"

// the pins connected to the Ultrasonic sensor
int echoPin = 4;
int trigPin = 5;

// the pin connectd to our push button
int buttonPin = 2;

// the pin the speaker is attached to
int speakerPin = 8;

// a distance reading has to change more or less than
// the recorded distance and the tolerance
float distance_to_door_tolerance = 10;

// the distance to our door in CM
// -1 means it has not yet been set
float distance_to_door = -1;

// the notes in our melody and their duration in fractions of a second
// e.g. quarter note = 4, eighth note = 8, etc.
const int melody[][2] =
{
    {NOTE_D7, 4},
    {NOTE_A6, 4},
    {NOTE_BLANK, 4}
};

```

```
58  void play_alarm()
59  {
60      // figure out the number of notes in our melody
61      int numberOfNotes = sizeof(melody) / sizeof(melody[0]);
62
63      // iterate over the notes of the melody
64      for (int thisNote = 0; thisNote < numberOfNotes; thisNote++)
65      {
66          // grab our note and note duration from our array
67          int thisNoteTone = melody[thisNote][0];
68          int thisNoteDuration = melody[thisNote][1];
69
70          // to calculate the note duration in ms
71          int noteDurationMS = 1000 / thisNoteDuration;
72
73          // play the note
74          tone(speakerPin, thisNoteTone, noteDurationMS);
75
76          // to distinguish the notes, set a minimum time between them.
77          // the note's duration + 30% seems to work well:
78          delay(noteDurationMS * 1.30);
79      }
80  }
81
82  float get_distance_cm()
83  {
84      float distanceCentimeters;
85      int pulseLenMicroseconds;
86
87      // bit-bang a small square wave
88      // on the trig pin to start the range
89      // finder
90      digitalWrite(trigPin, LOW);
91      delayMicroseconds(20);
92      digitalWrite(trigPin, HIGH);
93      delayMicroseconds(100);
94      digitalWrite(trigPin, LOW);
95
96      // measure the pulse length from the echo pin
97      pulseLenMicroseconds = pulseIn(echoPin, HIGH);
98
99      // calculate the distance using the speed of sound
100     distanceCentimeters = pulseLenMicroseconds / 29.387 / 2;
101
102     return distanceCentimeters;
103 }
104
105 void setup()
106 {
107     // set the pinmode on our ultrasonic echo, and tric pins
108     pinMode(echoPin, INPUT);
109     pinMode(trigPin, OUTPUT);
110     // set the pinmode for our button to INPUT with pullup
111     pinMode(buttonPin, INPUT_PULLUP);
112 }
113
```

```
114 void loop()
115 {
116     int button;
117     float distance;
118
119     // read the ultrasonic sensor
120     distance = get_distance_cm();
121
122     // if the distance is greater or less then our preset alarm
123     // distance, and we have set a distance, sound the alarm.
124     if (distance > distance_to_door + distance_to_door_tolerance
125         || distance < distance_to_door - distance_to_door_tolerance)
126     {
127         if (distance_to_door != -1)
128             play_alarm();
129     }
130
131     // read our button state
132     button = digitalRead(buttonPin);
133
134     // if the button is pressed, set the distance to the last
135     // distance read
136     if (button == LOW)
137     {
138         distance_to_door = distance;
139     }
140 }
```

Code Hint: On line 111 we use INPUT_PULLUP, what's that all about? A momentary push button has two states, open, and closed. When closed the circuit grounds our digital pin, but when opened nothings connected to our digital pin (this is called "floating"). We can't have the input pin floating, we need it to be either 0v (GND), or 5v (VCC). To get the digital pin to read 5v when the push button isn't pressed you'll need a pull-up resistor. In the 101 kit we use pull-up resistors in the Digital Input tutorial. Why isn't there a pull-up resistor in the schematic? The processor on the Uno has internal resistors available for exactly this purpose, we just have to turn them on! You can do so using INPUT_PULLUP instead of INPUT in the pinMode function call.

Code Hint: On line 124 you may have not seen the OR operator before. The double vertical bar is a Boolean OR operator "||". Likewise, the double ampersand is the AND operator "&&".

VI - Review

In this tutorial we've learnt:

- How to combine the Melody and Ultrasonic Range Finder tutorials to create a door alarm

Stay Connected

Stay connected with us to get the latest product information, free giveaways, and contests. It is also a great way to let us know what's on your mind. We love to hear from our customers about what we are doing right but most importantly, how we can improve our products and services!

https://www.facebook.com/OSEPP.ArduinoCompatible

https://twitter.com/DIY_w_OSEPP

Subscribe by going to our website

Support

Our team of engineers at OSEPP can't guarantee that every product we sell will be perfect; we make mistakes along the way. For that reason, we put a lot of effort into making sure that all complaints, comments, and concerns be dealt with promptly and to the customer's satisfaction!

We strive to provide the "Best after sales support" in the industry.

Here's how you can contact us:

Technical Support support@osepp.com

General Inquiries info@osepp.com

Sales Related sales@osepp.com