



Dr. Rajesh Panicker

Acknowledgements:

- www.Arduino.cc
- www.sparkfun.com
- Some slides from Arduino introduction slides by Linz Craig, Nick Poole, Prashanta Aryal, Theo Simpson, Tai Johnson, and Eli Santistevan
- Most Arduino-based circuit images are created using <https://www.tinkercad.com>
(create an online account to simulate all the examples before trying on real hardware!)
- Some images created using Fritzing
- Some images are copyrighted by their respective owners

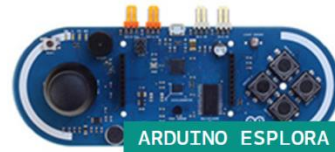
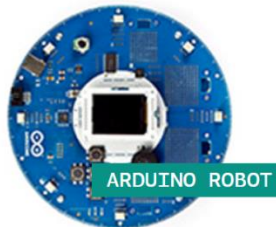
What is Arduino?

- Arduino is an open-source electronics platform based on easy-to-use hardware and software
- Arduino boards are able to read inputs - light on a sensor, a finger on a button, or a Twitter message - and turn it into an output - activating a motor, turning on an LED, publishing something online
- You can tell your board what to do by sending a set of instructions to the microcontroller on the board. To do so you use the Arduino programming language (C++), and the Arduino Software (IDE)
- A worldwide community of makers - students, hobbyists, artists, programmers, and professionals - has gathered around this open-source platform, their contributions have added up to an incredible amount of accessible knowledge that can be of great help to novices and experts alike

Why Arduino?

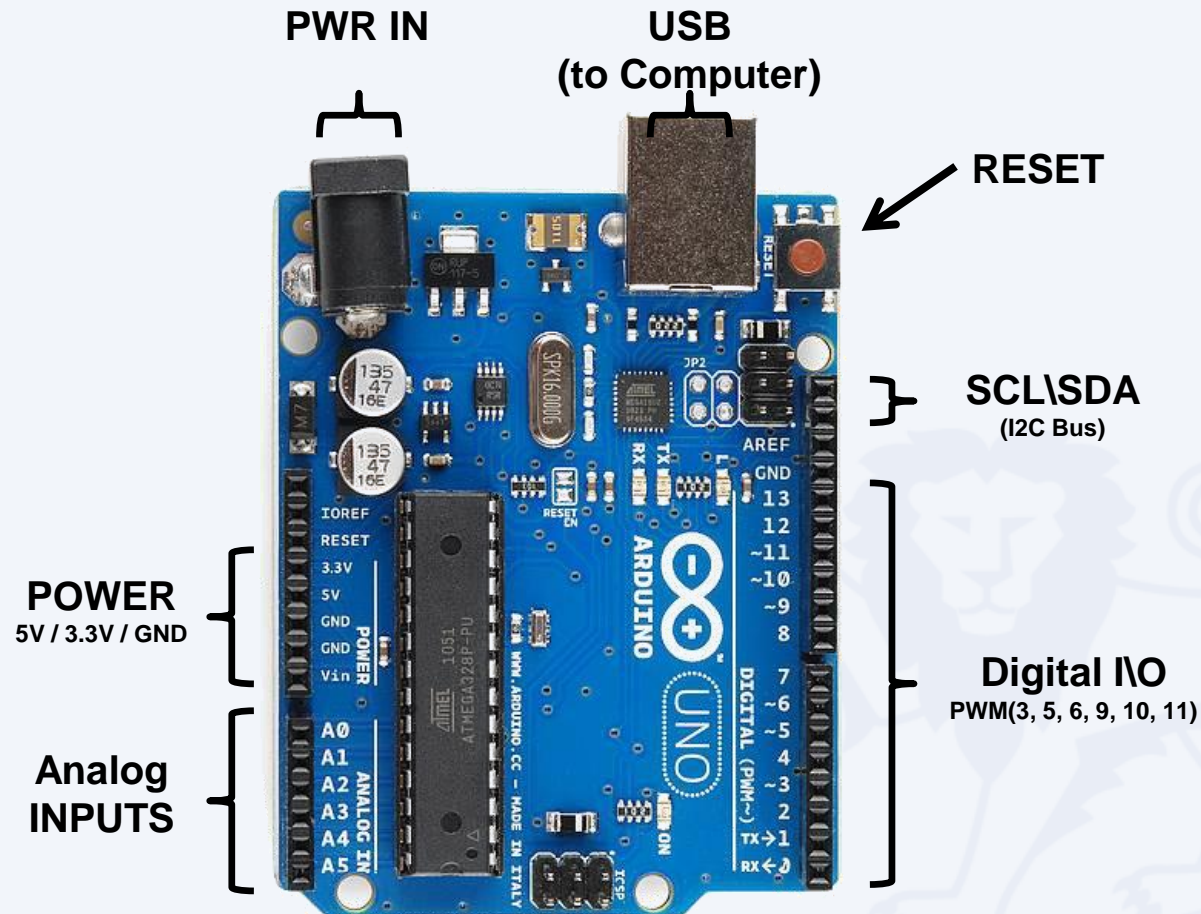
- Over the years Arduino has been the brain of thousands of projects, from everyday objects to complex scientific instruments
- Inexpensive
- Cross-platform (IDE works on Windows, Mac and Linux)
- Simple, clear programming environment
- Open-source hardware empowering users to build them independently and eventually adapt them to their particular needs
- Software growing through the contributions of users worldwide

Arduino Boards (the “Brain”)



<https://www.arduino.cc/en/Main/Products>

Arduino Uno (most popular)



Input vs. Output

Referenced from the perspective of the Arduino Board

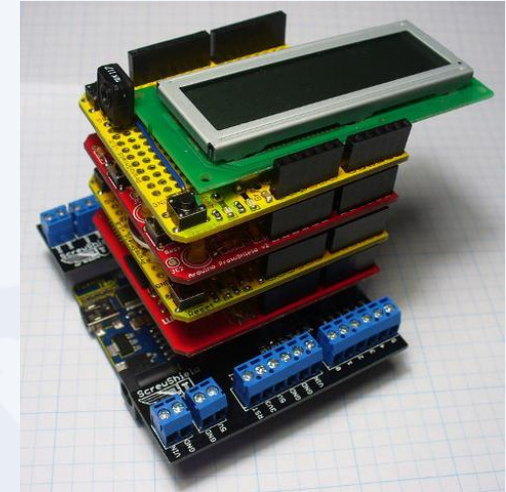
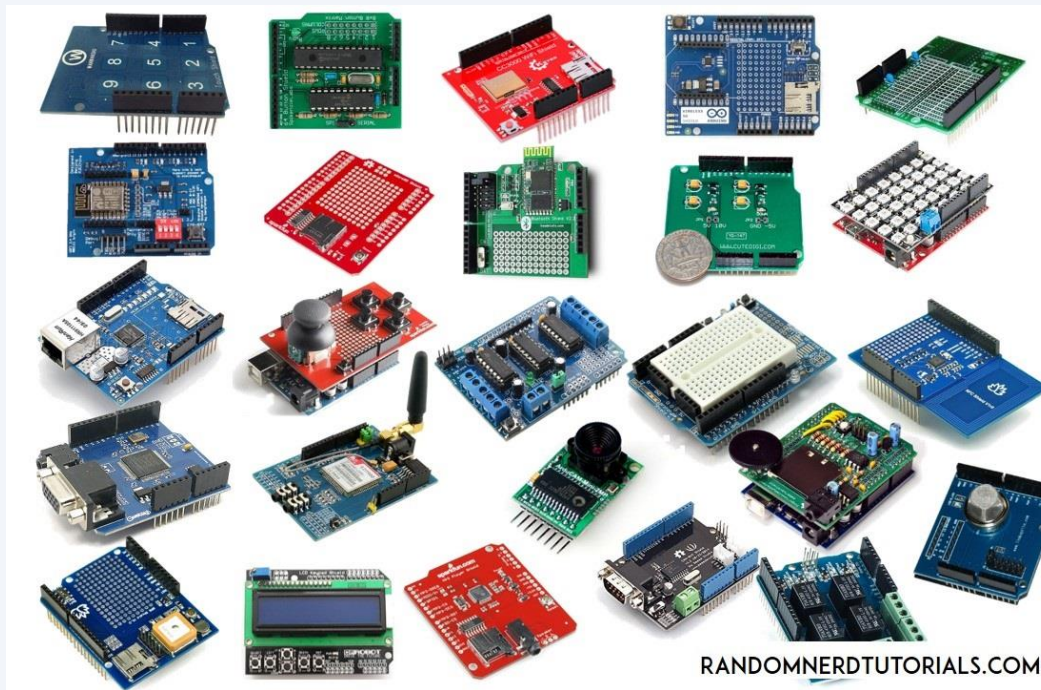
Inputs is a signal / information going into the board

Output is any signal exiting the board



- Almost all systems that use physical computing will have some form of output
- A device which can provide input(s) is called an input device, usually referred to as sensors. Ex: Light sensors (LDRs), Accelerometers, Push buttons
- Output devices are usually referred to as actuators Ex: Motors, LEDs

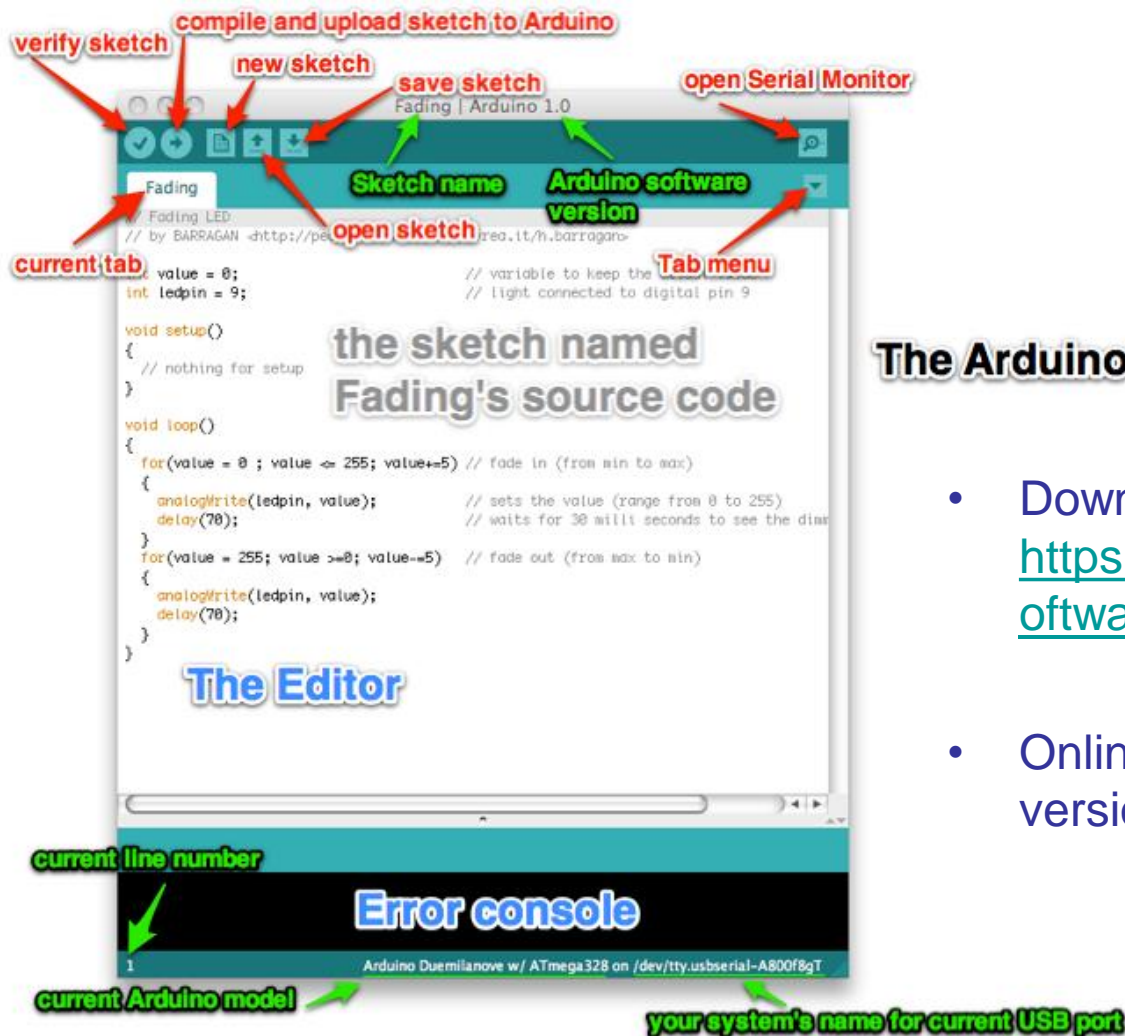
Shields (the “Body Parts”)



Stacked
shields

- Shields provide an easy way to interface sensor and actuators with the Arduino – avoids having to wire them up manually
- Shields can be stacked (terms and conditions apply!)

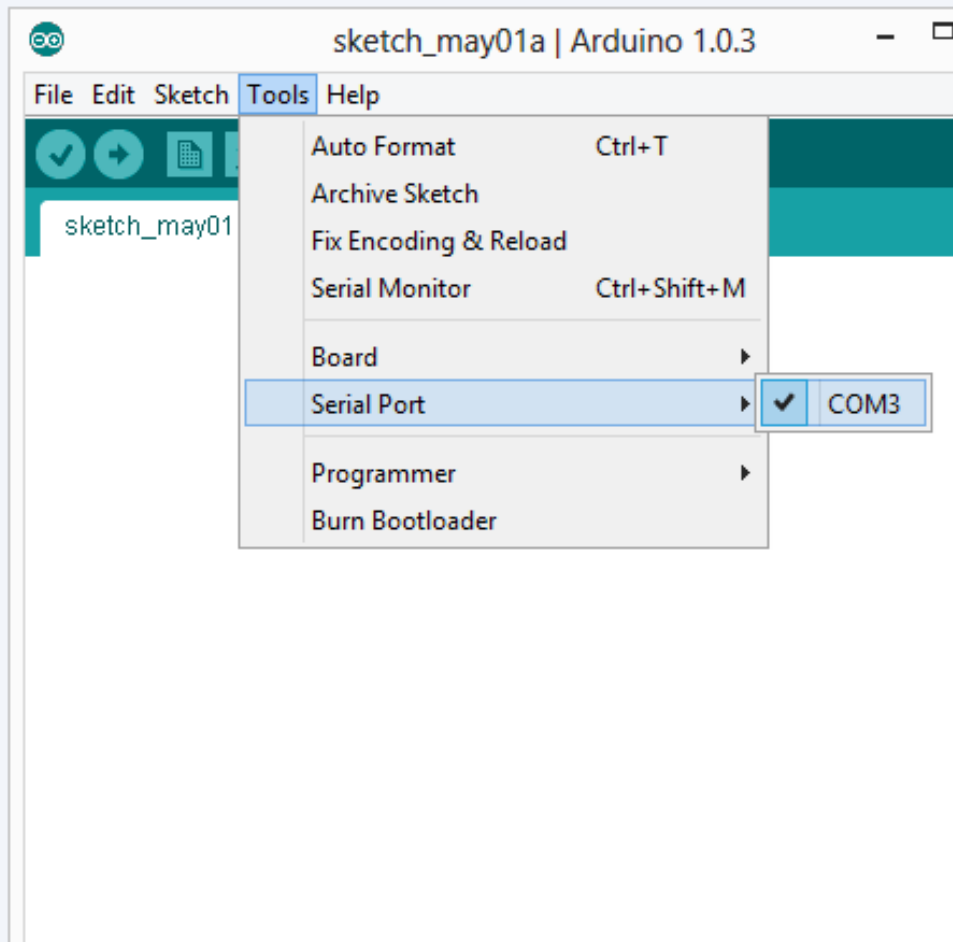
Arduino IDE



The Arduino IDE

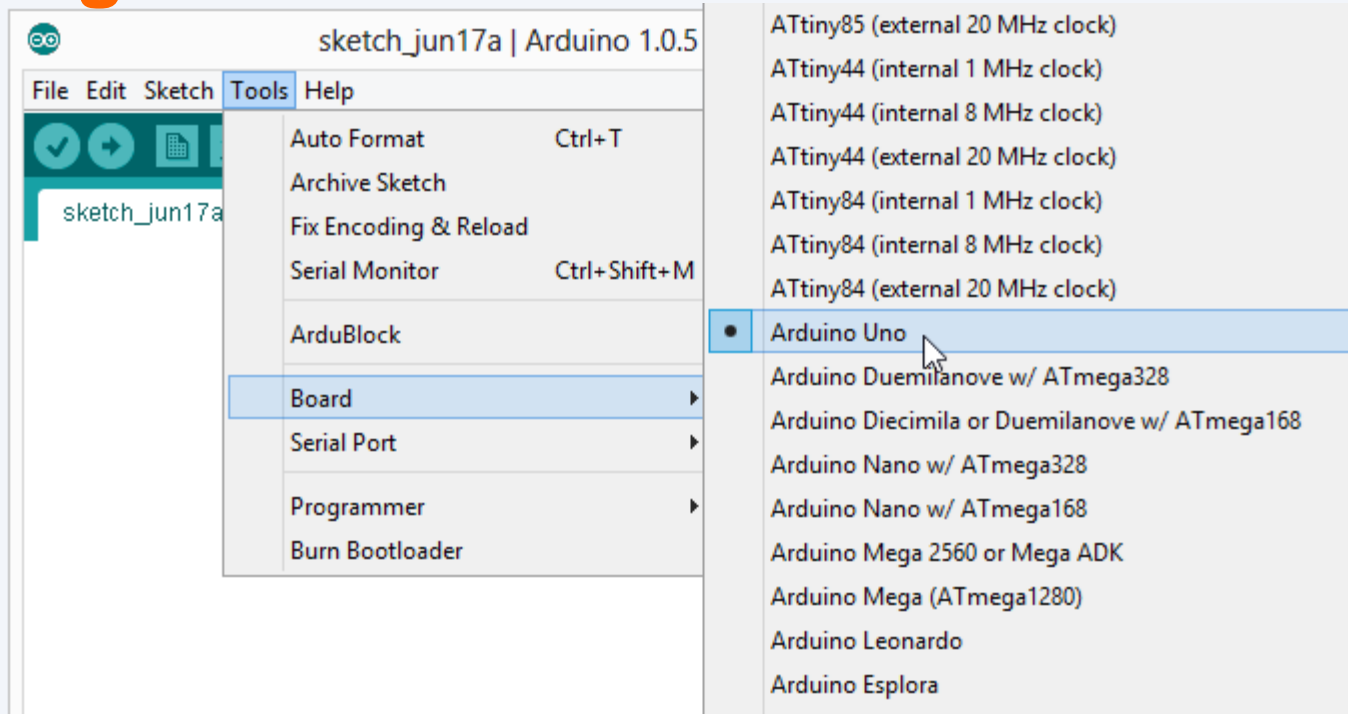
- Download from <https://www.arduino.cc/en/Main/Software>
- Online version available (but local version is recommended)

Settings: Tools → Serial Port



- Your computer communicates to the Arduino via a serial port → through a USB-Serial adapter
- Check to make sure that the drivers are properly installed

Settings: Tools → Board



- Next, double-check that the proper board is selected under the Tools→Board menu

Comments, Statements

- Comments are for you – the programmer or any other human who might read your code
- Comments are *not run* on the Arduino board

```
// single line comment  
/*  
    multi-line  
    comments  
*/
```

- Statements are compiled into an 'executable' of 1s and 0s, which are run on the Arduino.

Statements end with a ;

```
c = a+b; // c is assigned the sum of a, b
```

Variables

- Variables are containers to store intermediate data
- In C language, a variable needs to have a type, which specifies the nature and the size of data it holds
- Some of the standard data types are
 - **int** : 16-bit integers in the range -32,768 to 32,767
 - **unsigned int** : 16-bit positive integers (incl 0), 0 to 65535
 - **char** : 8-bit integers from -128 to 127. Characters such as '0', 'A', 'a' etc. are represented using ASCII (48, 65, 97 respectively - read up more!). BYTE is the same as unsigned char
 - **float** : for floating point numbers such as 2.25, -5.875 (32-bits)
 - Types can also be user defined (through **classes**)
- Variables can be declared with or without initialization
 - **int** a; // declaration
 - **int** a = 10; // declaration and initialization

Selection, Iteration

- A program which executes all the statements in sequence is probably not very interesting
- Sometimes, a part of a program must be executed based on a certain condition (selection)

```
if (a>0) {  
    c = a+b;  
    //executed when a>0  
}  
else{  
    c = a-b;  
    //executed when a<0 or a==0  
}
```
- Sometimes, a part of a program must be executed repeatedly until a certain condition is satisfied (iteration / loop)

```
while (a>0) {  
    a = a-b;  
    /* executed over and over  
    until a<0 or a==0 */  
}
```


Functions

- A function is a segment of code which can be invoked from different parts of the program in a parameterized manner
- A function involves 3 aspects

- Declaration (typically done in header files)

```
int add(int, int)
```

- Definition (typically done by the library vendor)

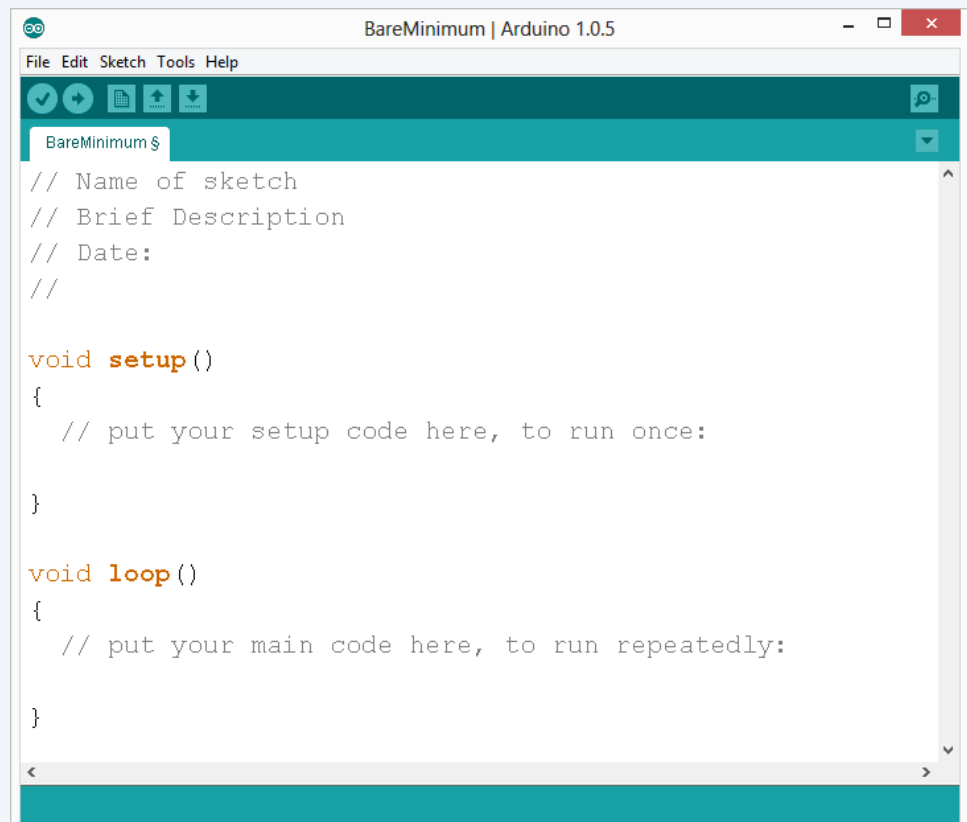
```
int add(int x, int y){ // x, y are function parameters  
                        //of the function 'add'  
  
    return x+y;  
}
```

- Usage (usually, we need to do only this)

```
{  
    ... //other statements  
    c = add(a,b);  
    /* causes add() to be called with a, b passed as arguments.  
       The value returned is assigned to c */  
    ... //other statements  
}
```

Arduino Program Structure

- Setup and loop are functions called automatically (unlike all other functions needs to be called explicitly)
- `setup()` is called *once*, when you release the reset button / plug in your Arduino / upload a new code onto the board
- `loop()` is called *repeatedly*; anything inside the `loop()` gets executed over and over as long as the system remains powered on

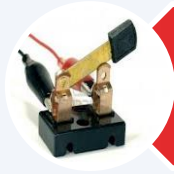


```
BareMinimum | Arduino 1.0.5
File Edit Sketch Tools Help
BareMinimum $
// Name of sketch
// Brief Description
// Date:
//

void setup()
{
    // put your setup code here, to run once:
}

void loop()
{
    // put your main code here, to run repeatedly:
}
```

BIG 6 CONCEPTS



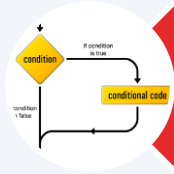
`digitalWrite()`



`analogWrite()`



`digitalRead()`



`if()` statements



`analogRead()`



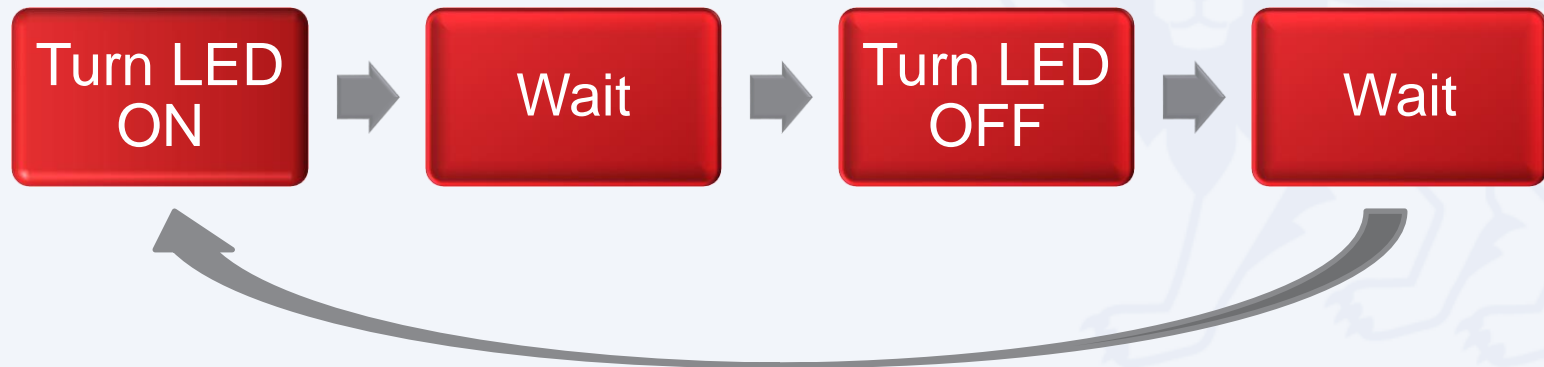
Serial communication

Let's Get Started..

Blinky

“Hello World” of Physical Computing

how do we implement this?



Digital Output



Three commands to know...

```
pinMode(pin, INPUT/OUTPUT) ;
```

```
ex: pinMode(13, OUTPUT) ;
```

```
digitalWrite(pin, HIGH/LOW) ;
```

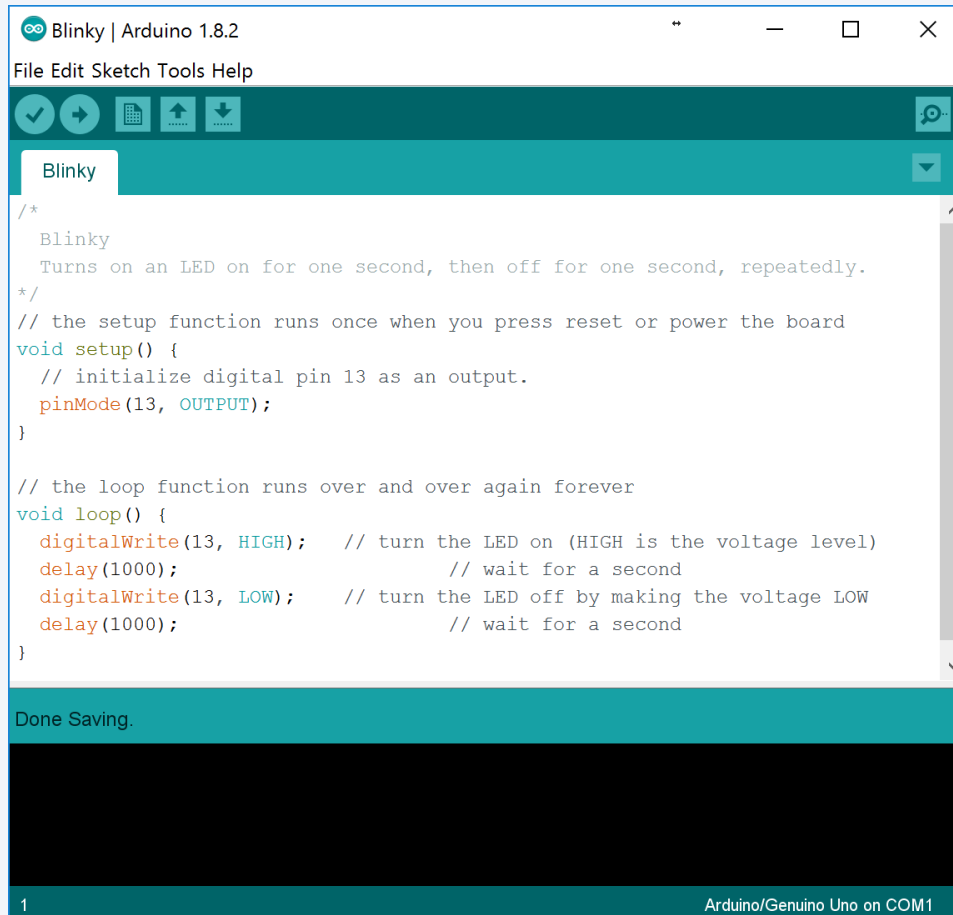
```
ex: digitalWrite(13, HIGH) ;
```

```
delay(time_ms) ;
```

```
ex: delay(2500) ; // delay of 2.5 sec.
```

```
// NOTE: -> commands are CASE-sensitive
```

Blink



```
/*
 * Blinky
 * Turns on an LED on for one second, then off for one second, repeatedly.
 */
// the setup function runs once when you press reset or power the board
void setup() {
  // initialize digital pin 13 as an output.
  pinMode(13, OUTPUT);
}

// the loop function runs over and over again forever
void loop() {
  digitalWrite(13, HIGH); // turn the LED on (HIGH is the voltage level)
  delay(1000);             // wait for a second
  digitalWrite(13, LOW);  // turn the LED off by making the voltage LOW
  delay(1000);             // wait for a second
}
```

Type this code, click
“Upload” and
observe the LED
close to pin 13

You have just
completed your first
Arduino program!

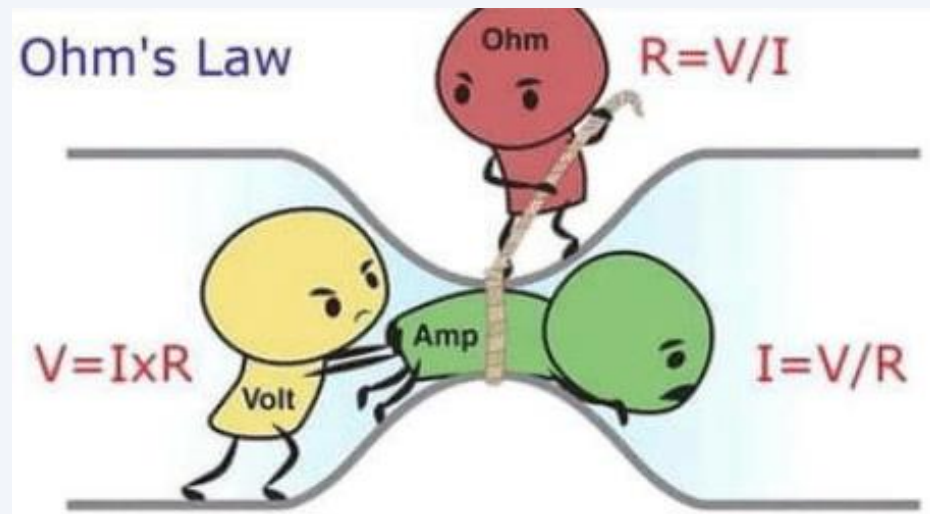
Electricity : Basic Concept Review

- **Ohm's Law**
- **Voltage**
- **Current**
- **Resistance**
- **Using a Multi-meter**



Ohm's Law

- Ohm's Law describes the relationship between the Voltage (**V**), Current (**I**) and Resistance (**R**) of a circuit
- It is stated as $V = I.R$
- $I = V/R$ and $R = V/I$ naturally follows



Electrical Properties

Voltage V

- The amount of potential energy in a circuit.
- Units: Volts (V)

Current I

- The rate of charge flow in a circuit.
- Units: Amperes (A)

Resistance R

- Opposition to charge flow.
- Units: Ohms (Ω)

$$V = I R$$

$$V = I R$$

Current Flow Analogy



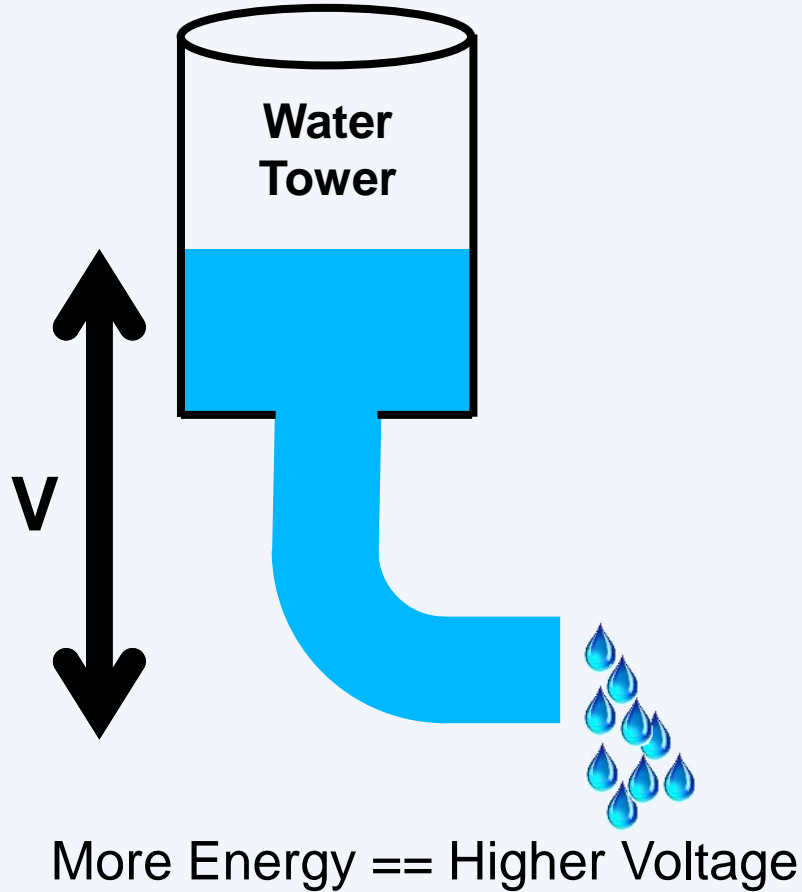
High Current



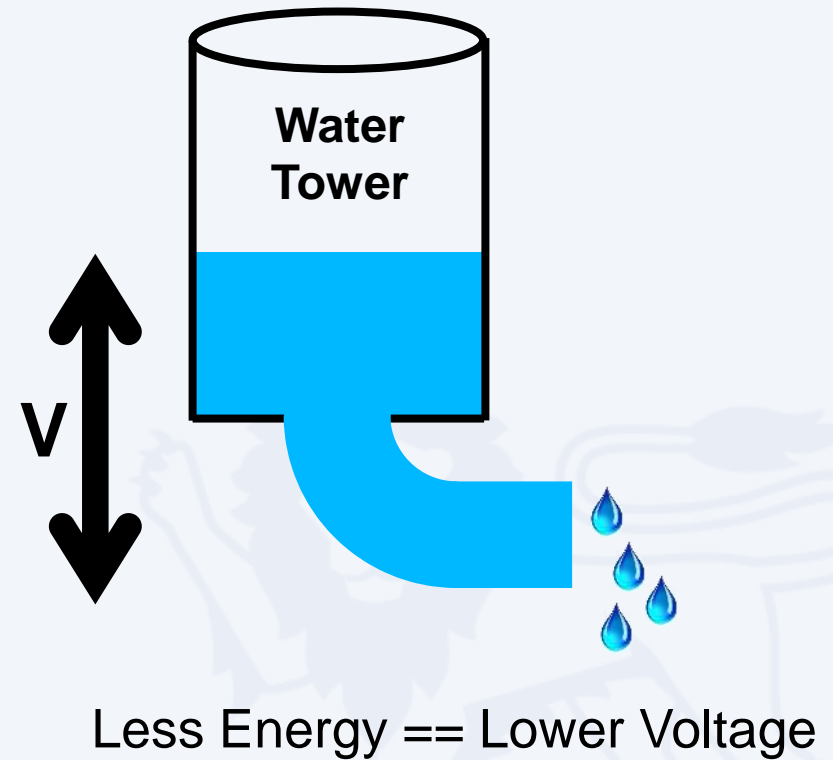
Low Current

$$V = I R$$

Voltage Analogy



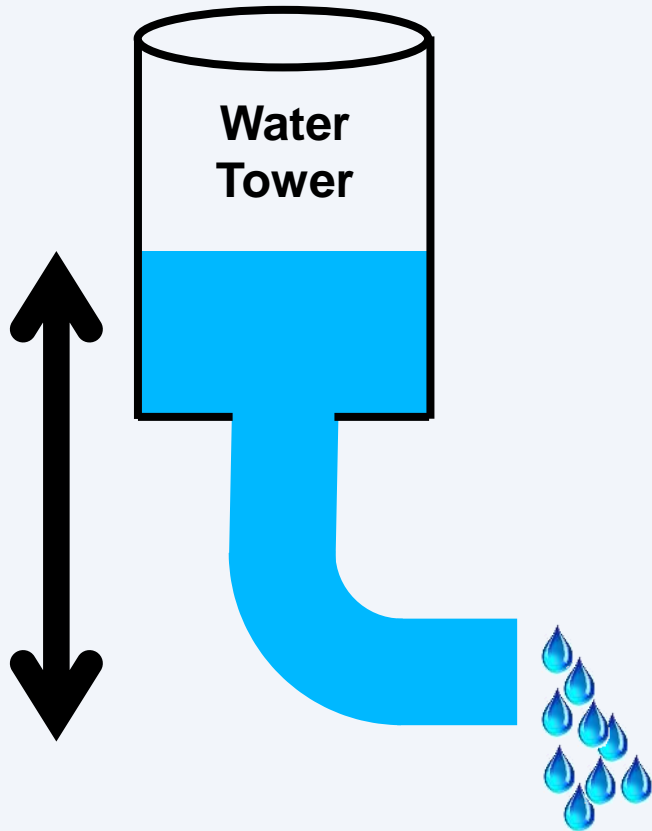
$$V = I R$$



$$V = I R$$

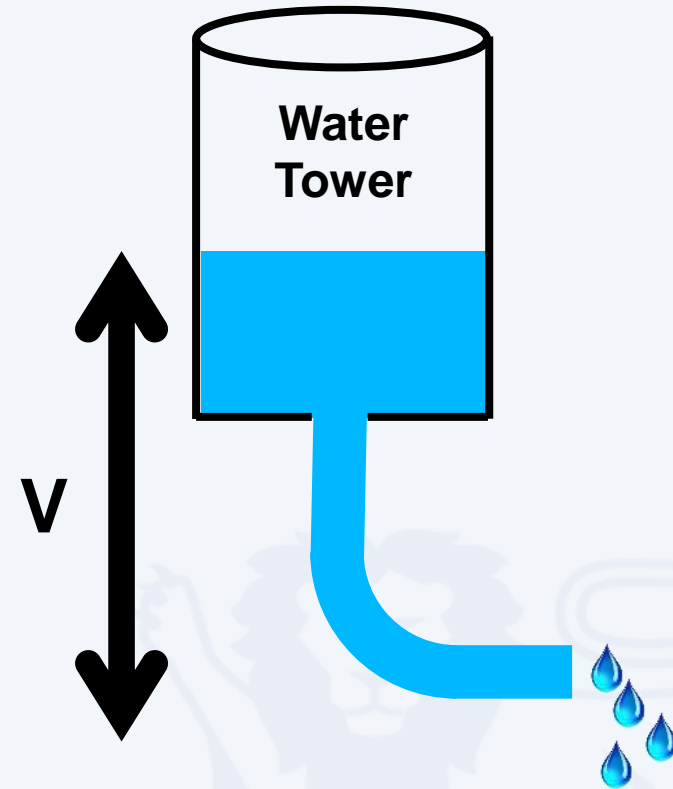
$$V = I R$$

Resistance Analogy



Big Pipe == Lower Resistance

$$V = I R$$



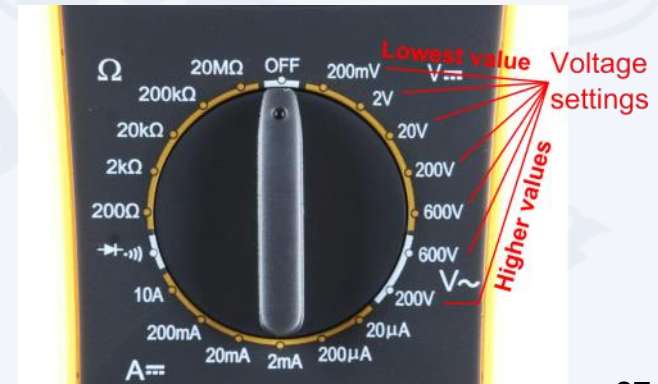
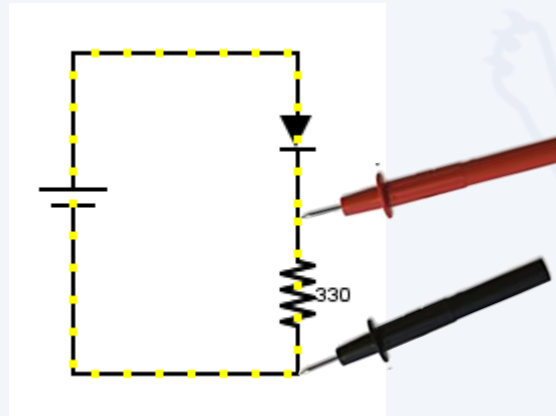
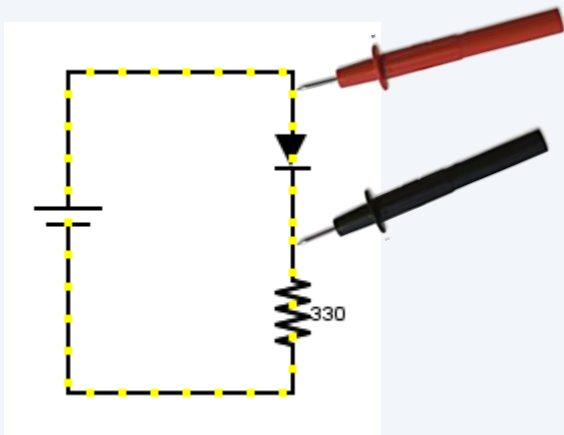
Small Pipe == Higher Resistance

$$V = I R$$



Measuring Electricity – Voltage

Voltage is a measure of potential electrical energy.
A voltage is also called a potential difference – it is measured between two points in a circuit – across a device

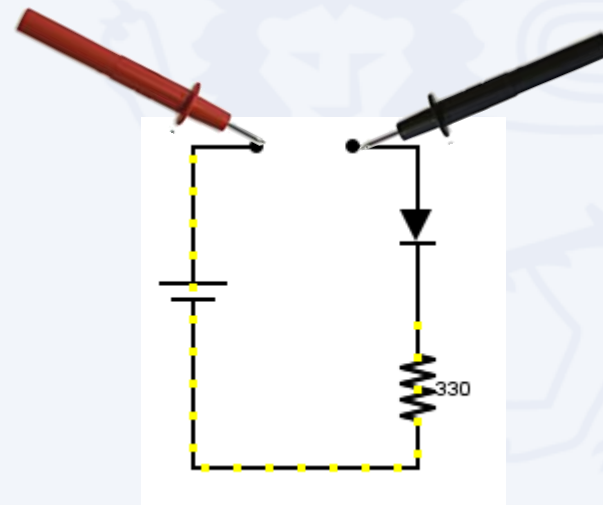
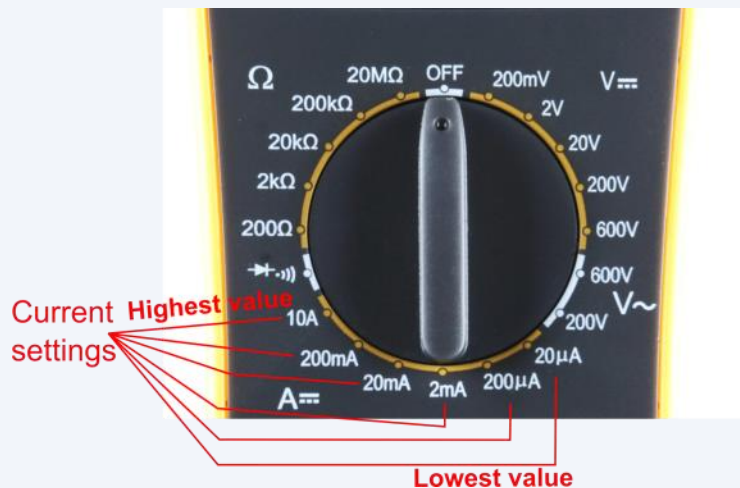




Measuring Electricity -- Current

Current is the measure of the rate of charge flow. For Electrical Engineers – we consider this to be the movement of electrons

In order to measure this – you must break the circuit or insert the meter in-line (series)



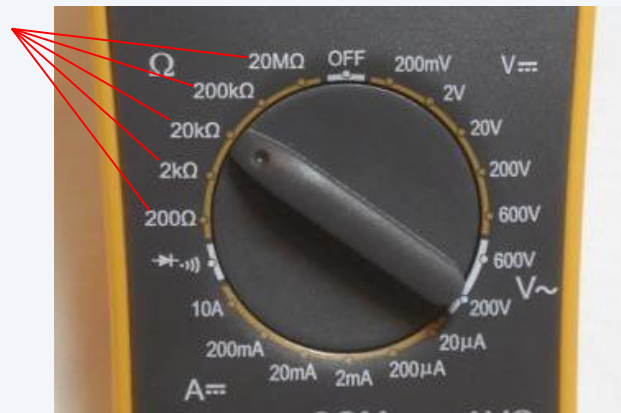


Measuring Electricity -- Resistance

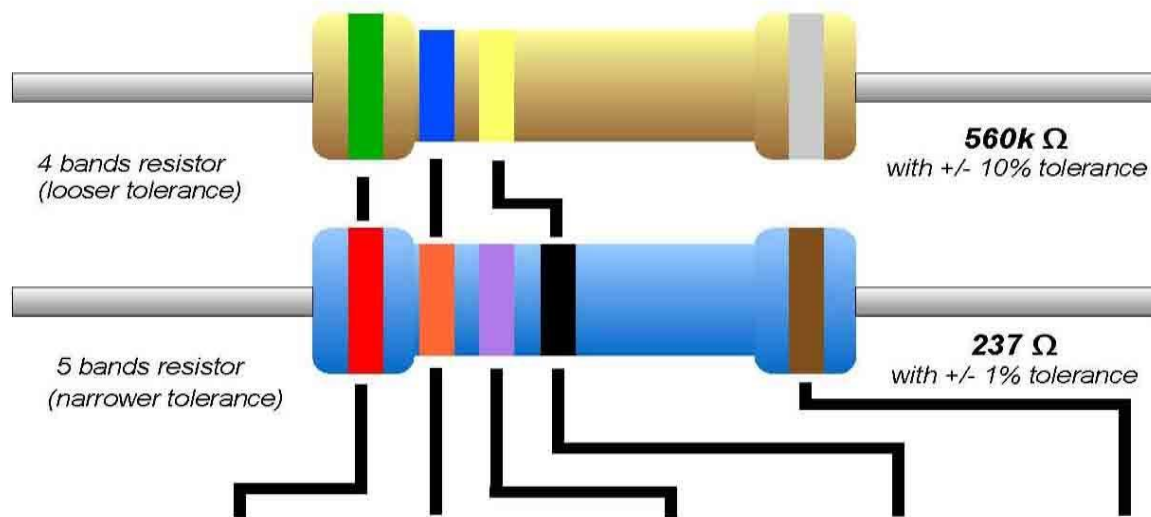
Resistance is the measure of how much opposition to current flow is in a circuit

Components should be removed entirely from the circuit to measure resistance. Note the settings on the multi-meter. Make sure that you are set for the appropriate range

Resistance
settings



Resistor Color Code

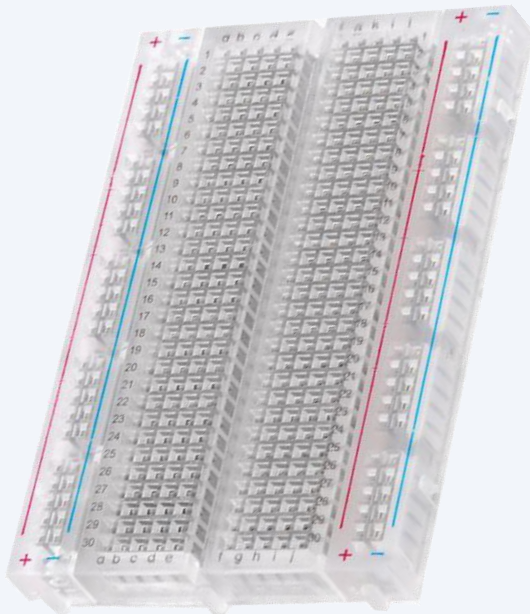


Color	1 st Band	2 nd Band	3 rd Band	Multiplier	Tolerance
Black	0	0	0	x 1 Ω	
Brown	1	1	1	x 10 Ω	+/- 1%
Red	2	2	2	x 100 Ω	+/- 2%
Orange	3	3	3	x 1K Ω	
Yellow	4	4	4	x 10K Ω	
Green	5	5	5	x 100K Ω	+/- 5%
Blue	6	6	6	x 1M Ω	+/- .25%
Violet	7	7	7	x 10M Ω	+/- .1%
Grey	8	8	8		+/- .05%
White	9	9	9		
Gold				x .1 Ω	+/- 5%
Silver				x .01 Ω	+/- 10%

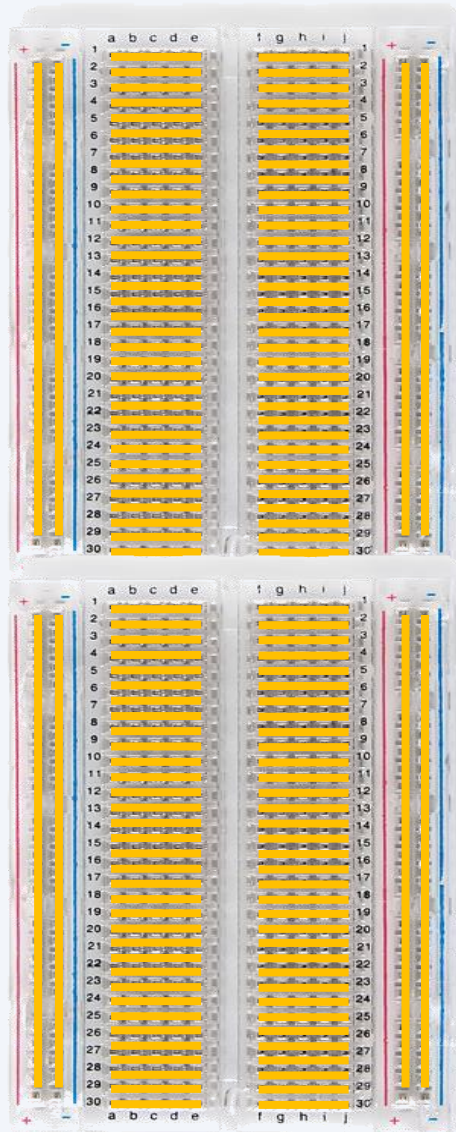
Prototyping Circuits : Breadboard

One of the most useful tools in an engineer or Maker's toolkit

- A breadboard is easier than soldering
- A lot of those little holes are connected, which ones?



Solderless Breadboard



Each row (horiz.) of 5 holes are connected

Vertical columns – called power bus are connected vertically

Longer breadboards have the vertical connections broken in the middle – think of them as 2 separate small breadboards

Using Breadboard

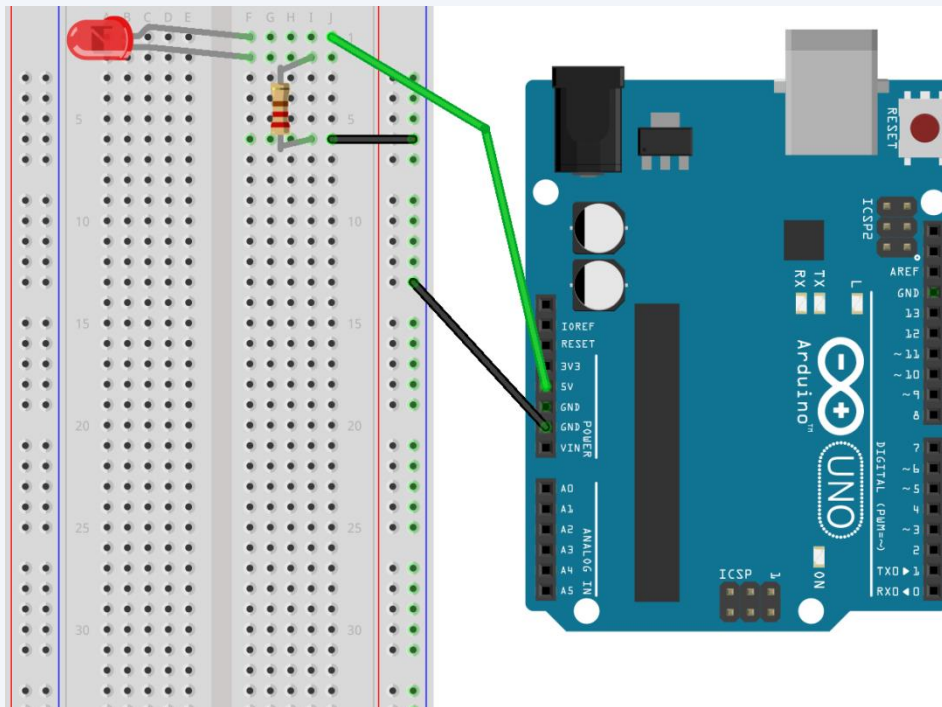
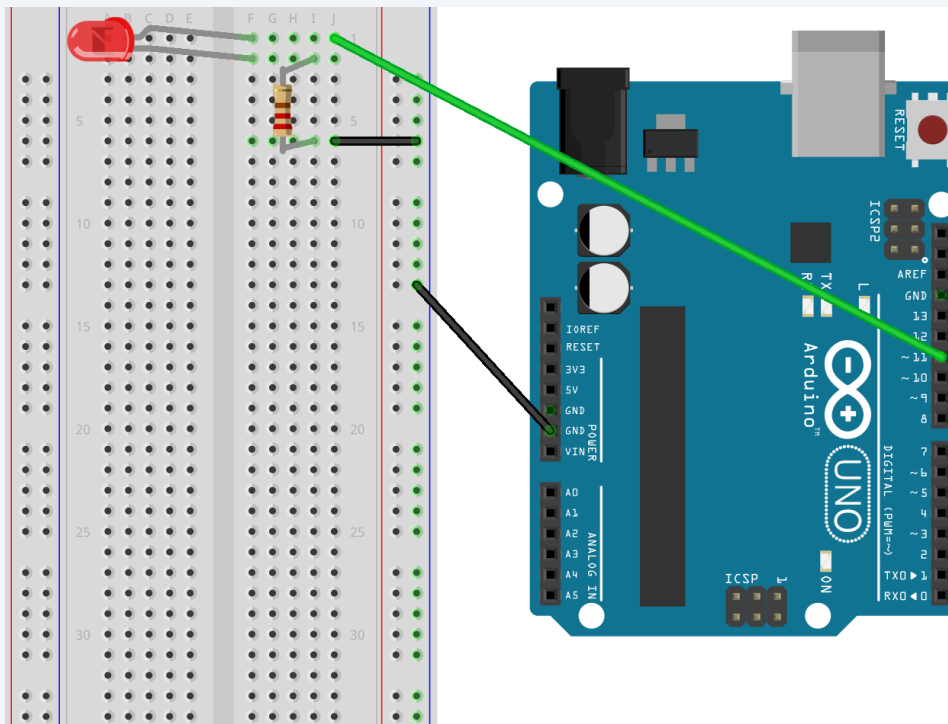


Image created in Fritzing

Use the breadboard to wire up a single LED with a 330 Ohm Resistor (Orange-Orange-Brown)

Note: the longer leg on the LED is the positive leg (green wire, connect to 5V) and the shorter leg is the negative (black wire, connect to 0V)

Blinking the LED



Move the green wire from the power pin to pin 13 on the Arduino board without changing the program

Try changing the connection from pin 13 to pin 11 (as shown in the image). How should your program be modified?

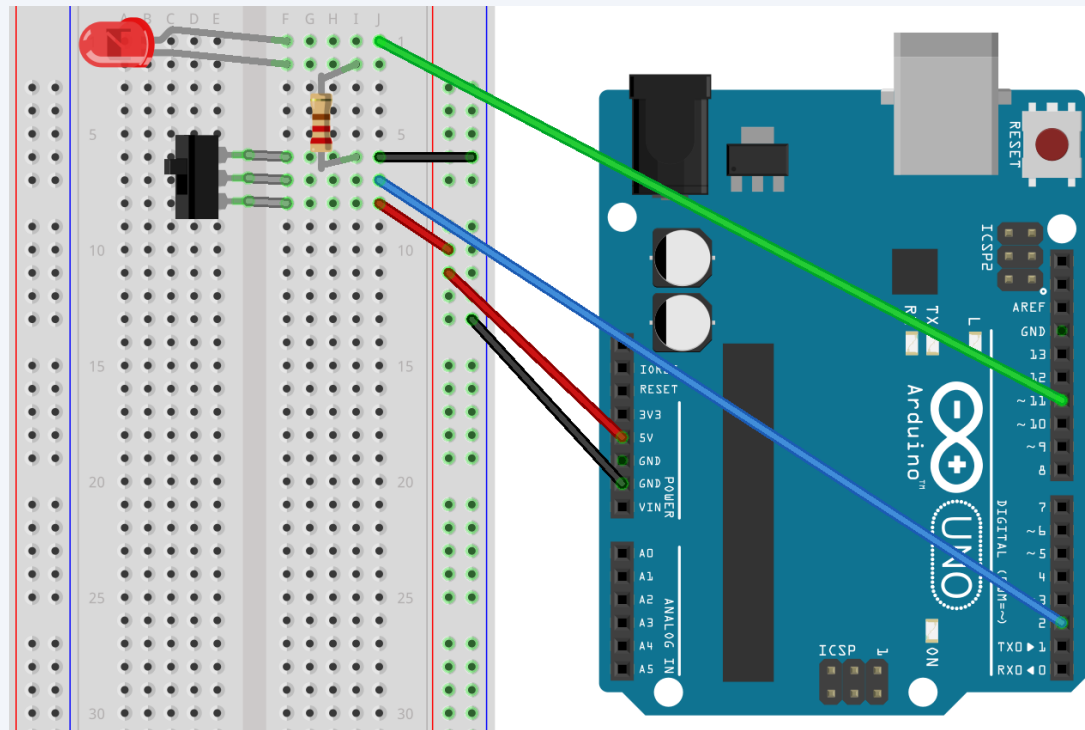
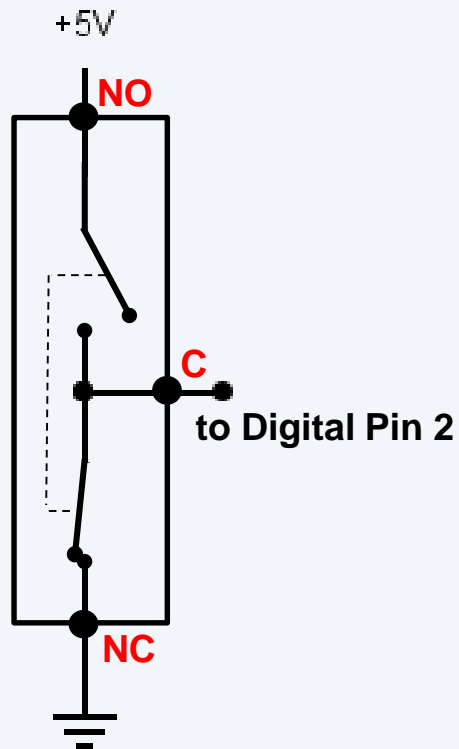
Digital Input



Digital Sensors

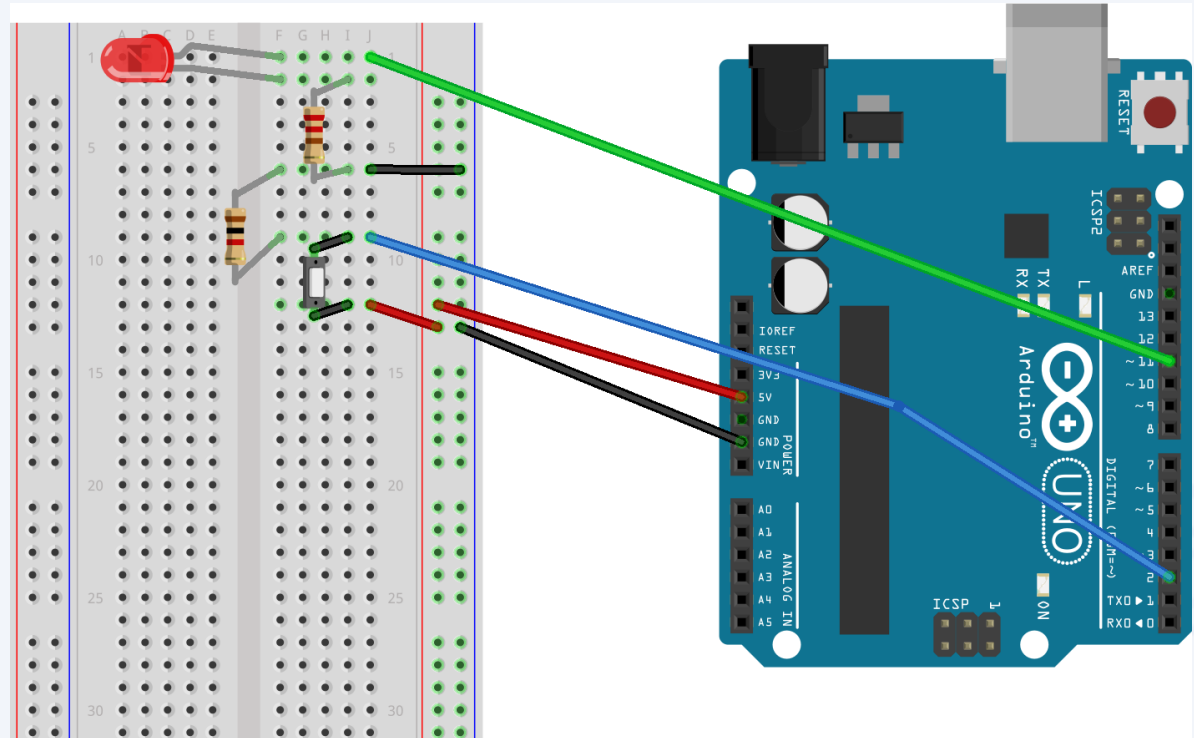
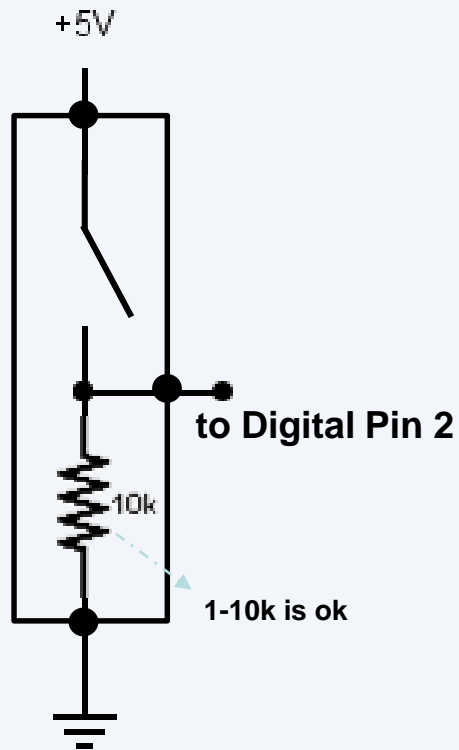
- Digital sensors are (more) straight forward (than Analog)
- No matter what the sensor there are only two settings:
On and Off
- Signal is always either HIGH (On) or LOW (Off)
- Voltage signal for HIGH will be 5V (more or less) on Arduino Uno. Other Arduinos could use different voltages!
- Voltage signal for LOW will be 0V on most systems

Digital Input – Switch (double throw)



NO = normally open; C = common; NC = normally closed.

Digital Input – Switch (single throw)



Digital Input

- Connect digital input to your Arduino using Pins # 0 – 13 (Avoid pins # 0 & 1 though as they are used for *Serial* later, and pin #11 and 13 as we are already using it)

- Digital Input needs a `pinMode` command:

```
pinMode (pinNumber, INPUT);
```

Make sure to use ALL CAPS for INPUT

- To get a digital reading:

```
int buttonState = digitalRead (pinNumber);
```

- Digital Input values are only HIGH (On) or LOW (Off)

We declare a
variable as an
integer.

We set it equal to the function
`digitalRead(pushButton)`

The function `digitalRead()` will return
the value 1 or 0, depending on whether
the button is being pressed or not
being pressed.

```
int buttonState = digitalRead(pushButton);
```

We name it
`buttonState`

The value 1 or 0 will be saved in
the variable `buttonState`.

Recall that the `pushButton`
variable stores the number 2

Conditional Statements `if ()`

```
void loop()  
{  
    int buttonState = digitalRead(2);  
    if(buttonState == HIGH)  
    {    // do something  
    }  
    else  
    {    // do something else  
    }  
}
```

Exercise 1

Modify your blinky program such that it blinks only when the switch is turned ON

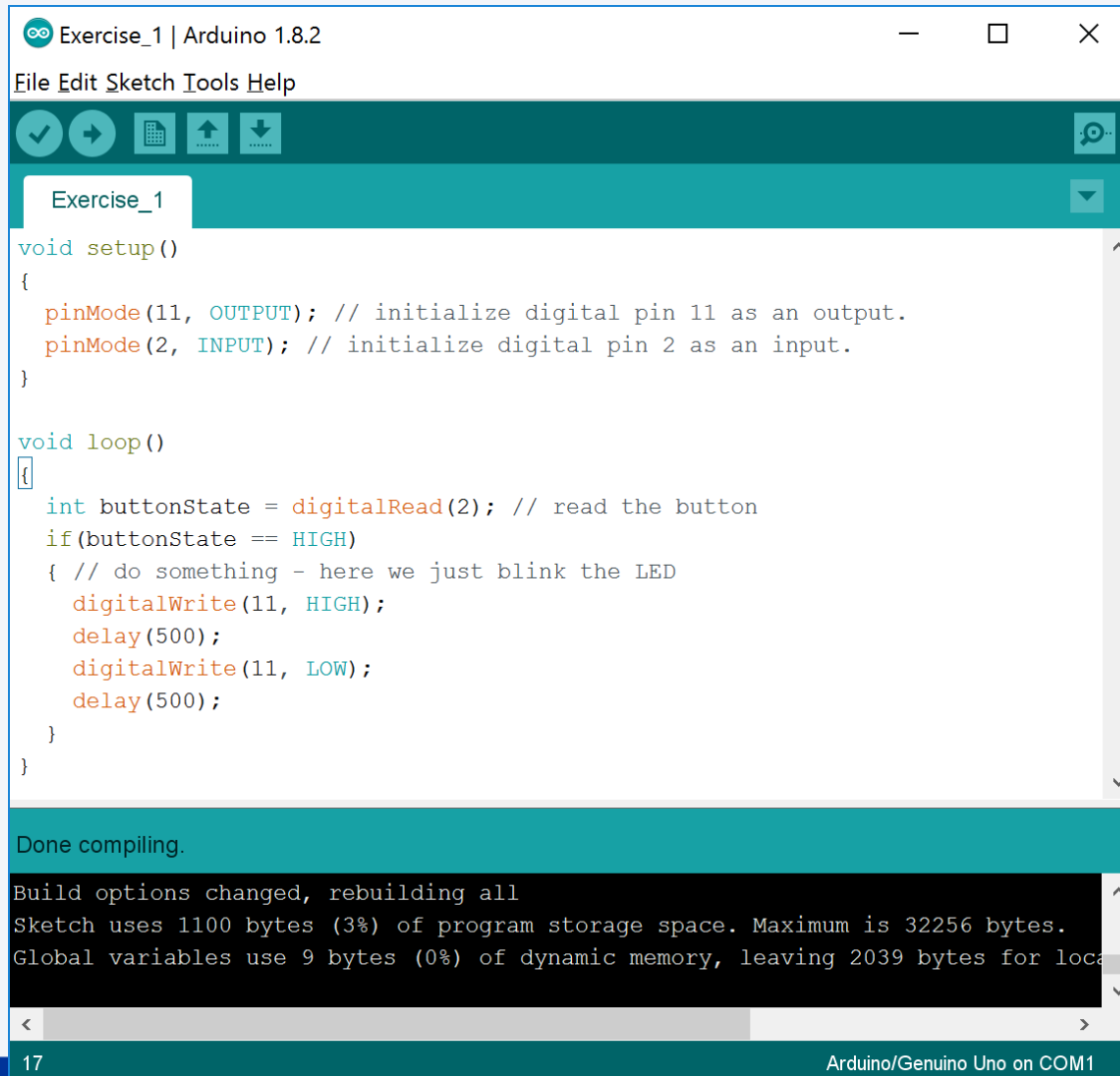
Hint :

In `setup()`, `pinMode(2, INPUT)` ; should be inserted

Place the code for blinking the LED below

`// do something in the previous slide`

Exercise 1 Solution



```
Exercise_1 | Arduino 1.8.2
File Edit Sketch Tools Help

Exercise_1

void setup()
{
  pinMode(11, OUTPUT); // initialize digital pin 11 as an output.
  pinMode(2, INPUT); // initialize digital pin 2 as an input.
}

void loop()
{
  int buttonState = digitalRead(2); // read the button
  if(buttonState == HIGH)
  { // do something - here we just blink the LED
    digitalWrite(11, HIGH);
    delay(500);
    digitalWrite(11, LOW);
    delay(500);
  }
}

Done compiling.

Build options changed, rebuilding all
Sketch uses 1100 bytes (3%) of program storage space. Maximum is 32256 bytes.
Global variables use 9 bytes (0%) of dynamic memory, leaving 2039 bytes for local variables.

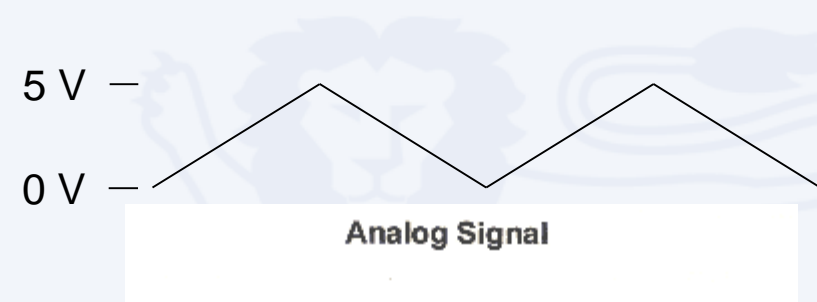
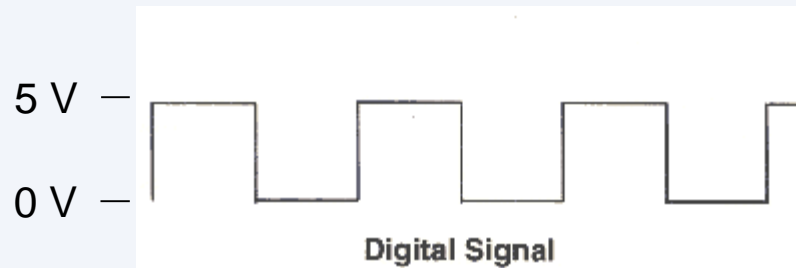
17 Arduino/Genuino Uno on COM1
```


Analog Output



Analog vs. Digital

- Arduinos are digital devices – ON or OFF. Also called discrete
- Analog signals are anything that can be a full range of values. Examples?

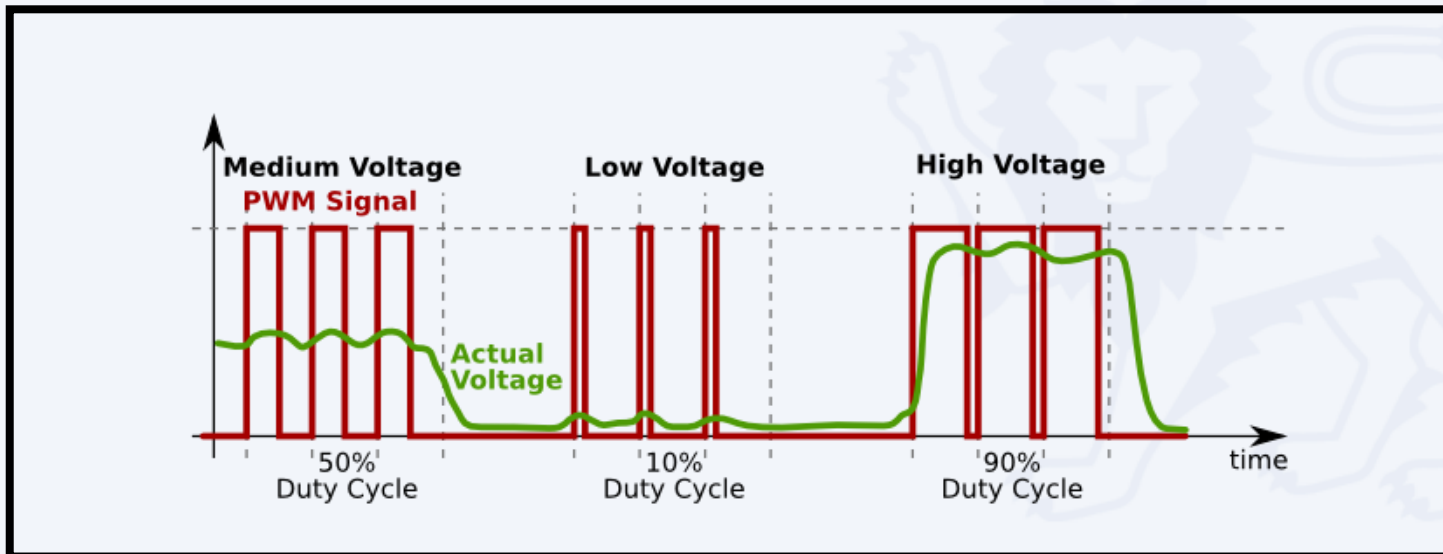


- How do we generate the effect of *analog* using *digital*?

Analog vs. Digital

- To create (mimic) an analog signal, the Arduino uses a technique called Pulse Width Modulation (PWM). By varying the duty cycle, we can mimic an “average” analog voltage

Pulse Width Modulation (PWM)



analogWrite()

```
analogWrite(pin, val);
```

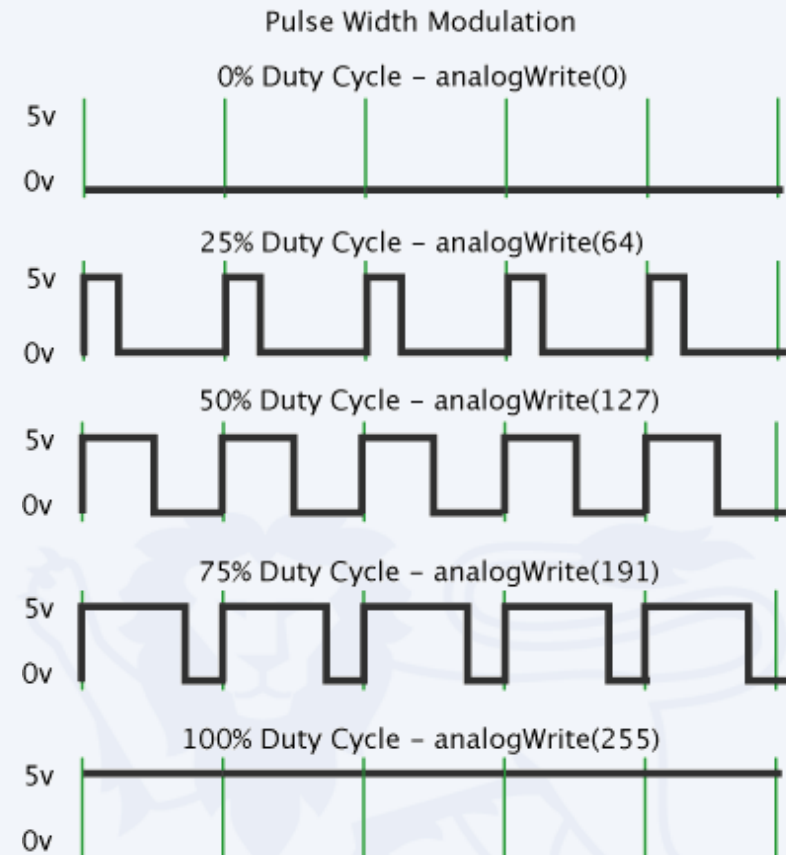
`pin` – refers to the OUTPUT
pin (limited to pins 3, 5, 6, 9, 10,
11.) – denoted by a ~ symbol

`val` – 8 bit value (0 – 255).

0 => 0V | 255 => 5V

Note : You do not need to call `pinMode()` to set the pin as an output before calling `analogWrite()`

The `analogWrite` function has nothing to do with the analog pins or the `analogRead` function



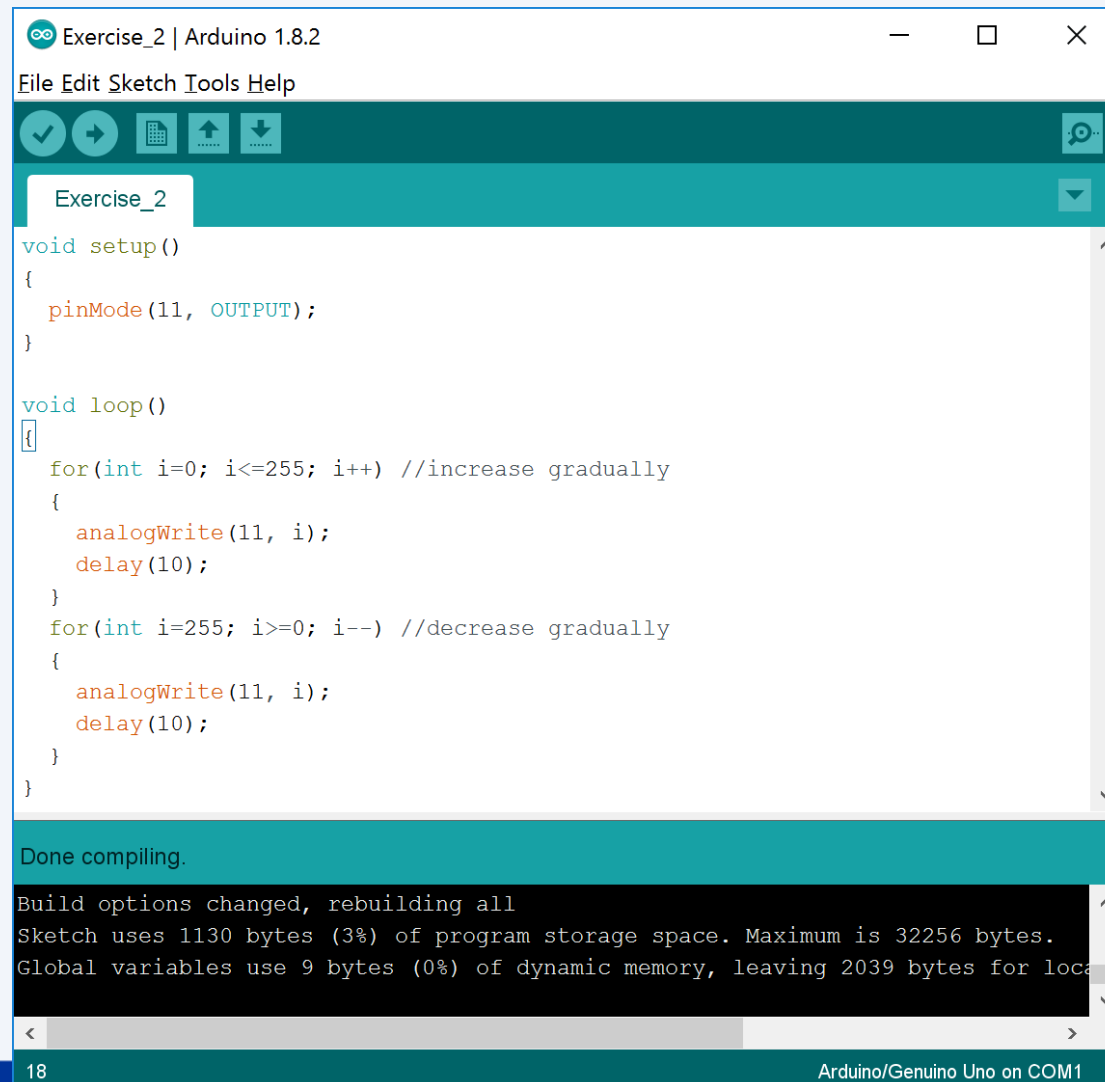
Exercise 2

- Create a program such that the LED brightness gradually increases from 0 to 255, and then goes abruptly to 0

Hints :

- Use pin 11. If you are already having the LED connected to pin 11, you need not change any connection. Why can't you use pin 13?
- You will have to use a for loop. Lookup for in <https://www.arduino.cc/en/Reference>
- The delay should be around 5-10 milliseconds
- Can you modify your program to decrease the brightness gradually from 255 to 0 instead of an abrupt change?

Exercise 2 Solution



```
Exercise_2 | Arduino 1.8.2
File Edit Sketch Tools Help

void setup()
{
  pinMode(11, OUTPUT);
}

void loop()
{
  for(int i=0; i<=255; i++) //increase gradually
  {
    analogWrite(11, i);
    delay(10);
  }
  for(int i=255; i>=0; i--) //decrease gradually
  {
    analogWrite(11, i);
    delay(10);
  }
}

Done compiling.

Build options changed, rebuilding all
Sketch uses 1130 bytes (3%) of program storage space. Maximum is 32256 bytes.
Global variables use 9 bytes (0%) of dynamic memory, leaving 2039 bytes for local variables.

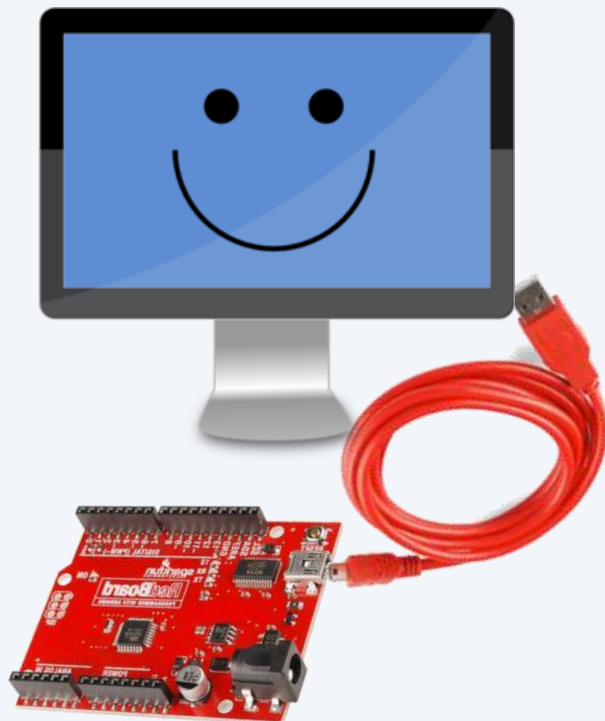
18 Arduino/Genuino Uno on COM1
```

Serial Communication



Using Serial Communication

Method used to transfer data between two devices.



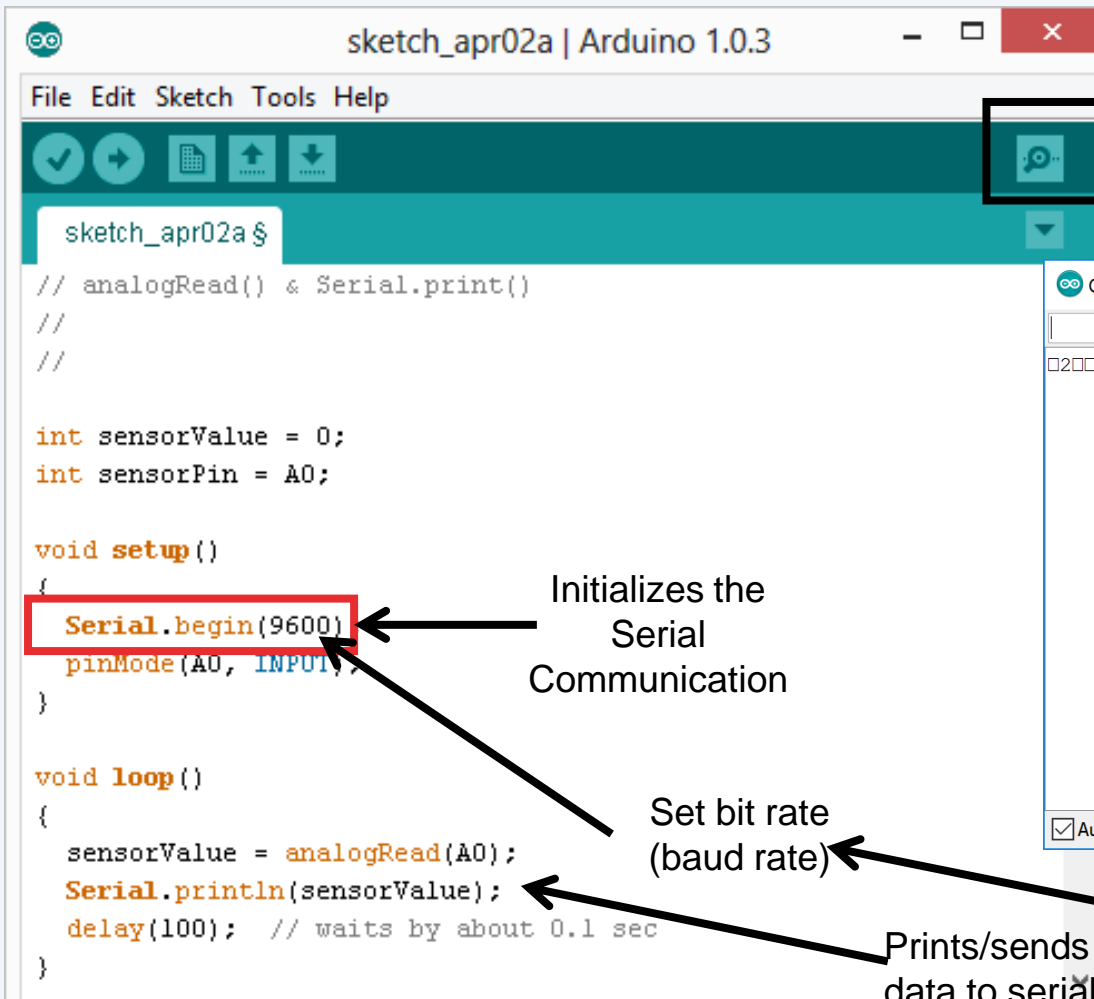
Data passes between the computer and Arduino through the USB cable. Data is transmitted as zeros ('0') and ones ('1') sequentially.



Arduino dedicates Digital I/O pin # 0 to receiving and Digital I/O pin #1 to transmit

For this reason, we typically do not use Digital I/O 0 or 1 for anything in our designs.

Serial Monitor & analogRead()



```

sketch_apr02a | Arduino 1.0.3
File Edit Sketch Tools Help

sketch_apr02a $

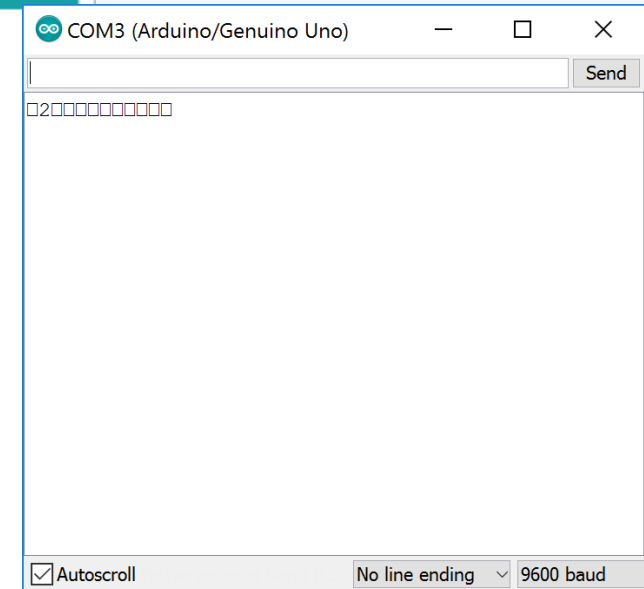
// analogRead() & Serial.print()
//
//

int sensorValue = 0;
int sensorPin = A0;

void setup()
{
  Serial.begin(9600);
  pinMode(A0, INPUT);
}

void loop()
{
  sensorValue = analogRead(A0);
  Serial.println(sensorValue);
  delay(100); // waits by about 0.1 sec
}
  
```

Opens up a
Serial Terminal
Window



Initializes the
Serial
Communication

Set bit rate
(baud rate)

Prints/sends
data to serial

Should be set to
be the same

Sending a Message

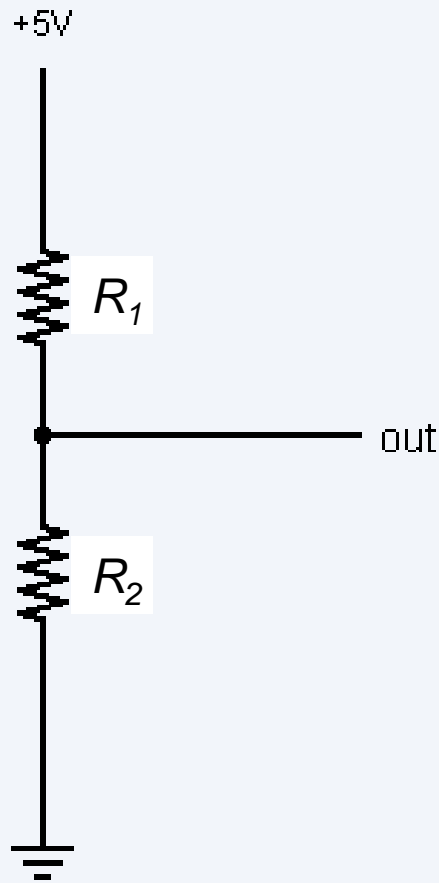
```
void setup()  
{  
  // initialize serial:  
  Serial.begin(9600);  
}  
void loop ( )  
{  
  Serial.print("Hands on") ;  
  Serial.print("Learning") ;  
  Serial.println("is Fun!!!") ;  
}  
  
/* print does not terminate the line.  
println terminates the line such that  
anything printed after that will be in a new line */
```

If you see strange looking characters in the serial monitor, your bit / baud rate set in the Serial Monitor most likely does not match the one used in your Arduino sketch

Analog Input



Voltage Divider

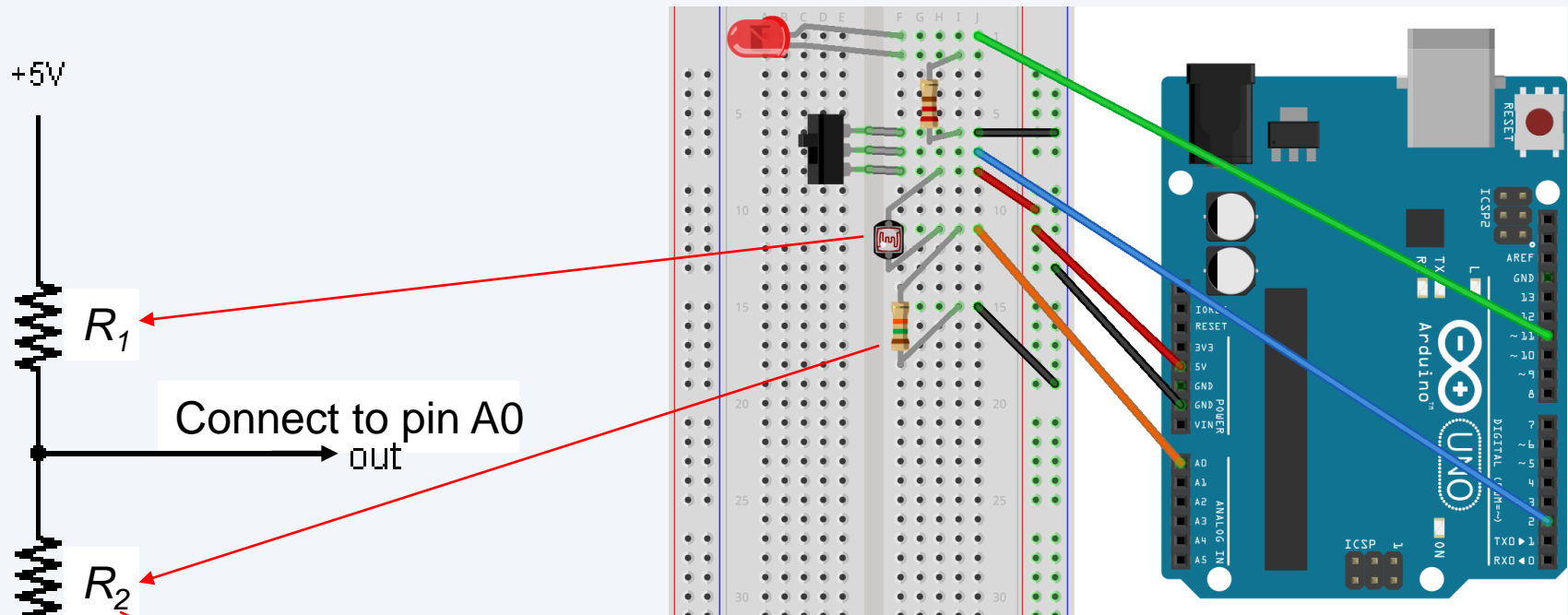


$$V_{R1} = V_{CC} \cdot \left(\frac{R_1}{R_{Total}} \right)$$

$$V_{R2} = V_{CC} \cdot \left(\frac{R_2}{R_{Total}} \right)$$

$$R_{Total} = R_1 + R_2$$

Light Dependent Resistor (LDR)



You have two options. See slide 58 for setting the threshold in your program depending on the option you chose.

Option 1 : Use a 10K resistor.

Option 2: Cover the LDR completely. Measure the resistance (R_{dark}) using multimeter. Expose the LDR to light. Measure the resistance (R_{bright}) using multimeter. Choose $R_2 = (R_{\text{dark}} + R_{\text{bright}})/2$

analogRead()

Arduino uses a 10-bit A/D Converter:

this means that you get input values from 0 to 1023

$0\text{ V} \rightarrow 0$

$5\text{ V} \rightarrow 1023$

Ex:

```
int sensorValue = analogRead(A0);
```

Exercise 3

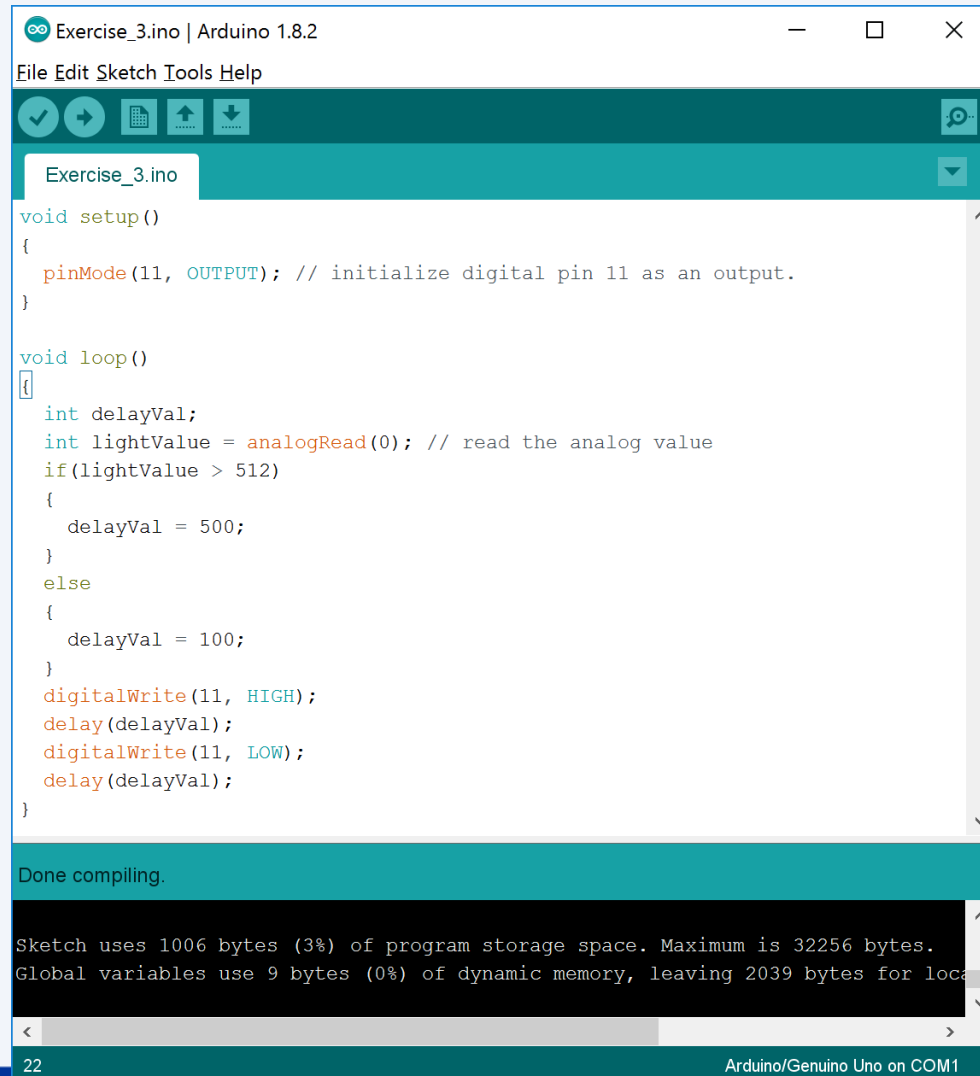
- Modify your blinky program such that it blinks faster when the LDR is covered, and slower when it is exposed to light

Hint :

Check whether the sensorValue is greater than a certain threshold using an 'if' condition

- The threshold can be set to 512 (approx. half of the maximum, which is 1023) if you chose Option 2 in slide 56
- If you chose Option 1, you need to find the threshold experimentally. Use the program in slide 52 to print the reading 1) when the LDR is fully exposed to light and 2) when the LDR is fully covered. The mean of the two can be used as the threshold

Exercise 3 Solution



```
Exercise_3.ino | Arduino 1.8.2
File Edit Sketch Tools Help

Exercise_3.ino

void setup()
{
  pinMode(11, OUTPUT); // initialize digital pin 11 as an output.
}

void loop()
{
  int delayVal;
  int lightValue = analogRead(0); // read the analog value
  if(lightValue > 512)
  {
    delayVal = 500;
  }
  else
  {
    delayVal = 100;
  }
  digitalWrite(11, HIGH);
  delay(delayVal);
  digitalWrite(11, LOW);
  delay(delayVal);
}

Done compiling.

Sketch uses 1006 bytes (3%) of program storage space. Maximum is 32256 bytes.
Global variables use 9 bytes (0%) of dynamic memory, leaving 2039 bytes for local variables.

22 Arduino/Genuino Uno on COM1
```


Exercise 4

- Modify your program such that
- When the switch is off
 - The message OFF is repeatedly printed via Serial, and the system will not do anything else
- When the switch is on
 - The LED will gradually turn on or off at a speed which is related to the amount of light falling on the LDR
 - The message ON is repeatedly printed via Serial
- Use a Serial bit/ baud rate of 115200 instead of 9600 for the above

Adding Contributed Libraries

- The beauty of Arduino platform is that there are libraries available for most peripherals. All you need to do is to Google
- To add a .zip file, Sketch > Include Library > Add .ZIP Library > select the .zip file
- Alternatively (if the library is unzipped), copy the folder to Documents\Arduino\libraries

After including the library, the path has to be exactly as shown below
Documents\Arduino\libraries\<lib_name>\<lib_name>.h/.cpp files

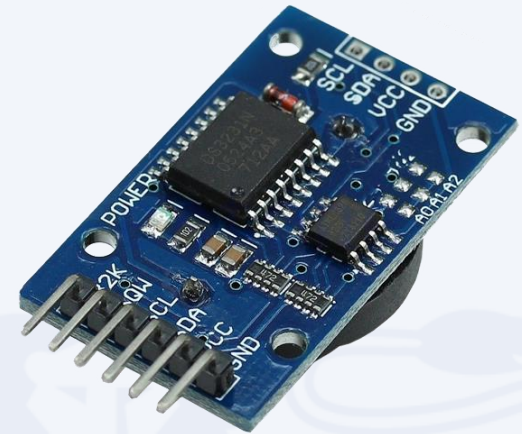
- Restart the Arduino IDE after adding the library
- You can see that the library at Sketch > Include Library
- Many libraries come with example programs, which can be your starting point. The example programs can be found at File > Examples > Examples from contributed libraries

Real-time Clock LCD Module Humidity Sensor



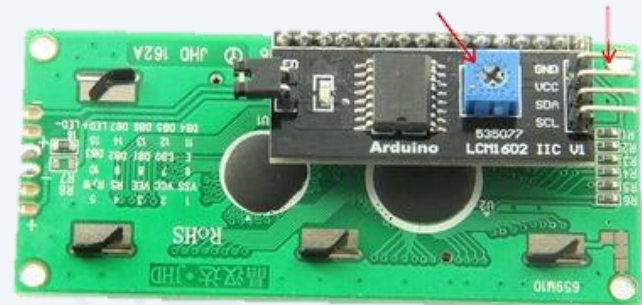
DS3232 REAL-TIME CLOCK

- Maintains time even when Arduino is not powered
- Used in computers to maintain time
- Time can be set from the Arduino
- Uses I2C protocol
- 4 connections
 - SDA (Serial Data) – Connect to SDA pin of Arduino (2 pins away from pin 13)
 - SCL (Serial Clock) – Connect to SCL pin of Arduino (3 pins away from pin 13)
 - VCC – 5V of Arduino
 - GND – GND of Arduino
- You can use the SetSerial program to set the time
 - Send 2019,10,19,16,10,00 from serial monitor to set the time to 4:10pm on 19/10/2019



I2C 16x2 CHARACTER LCD MODULE

- 16x2 display with blue backlight
- Contrast need to be adjust using a screw-driver
- Uses I2C protocol
- 4 connections
 - SDA – SDA pin of Arduino
 - SCL – SCL pin of Arduino
 - VCC – 5V of Arduino
 - GND – GND of Arduino
- Some modules may use a different address – use the I2Scanner program can find out
- Can be connected in parallel to the DS3232 RTC and be operated simultaneously (the two are distinguished by their addresses)



DHT 11 HUMIDITY SENSOR

- Gives temperature and humidity readings
- Uses a custom (non-standard) protocol
- Data can be up to 2 seconds old
- 3 connections
 - DATA – Serial Data. Connect to any digital pin of Arduino
 - The supplied program is meant to use pin 12
 - Change `#define DHTPIN <pin_number>` if using any other pin
 - VCC – 5V of Arduino
 - GND – GND of Arduino

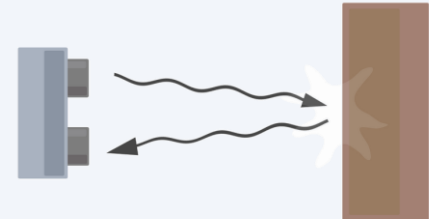


Ultrasonic Sensor Motors etc.





Ultrasonic Distance Sensor



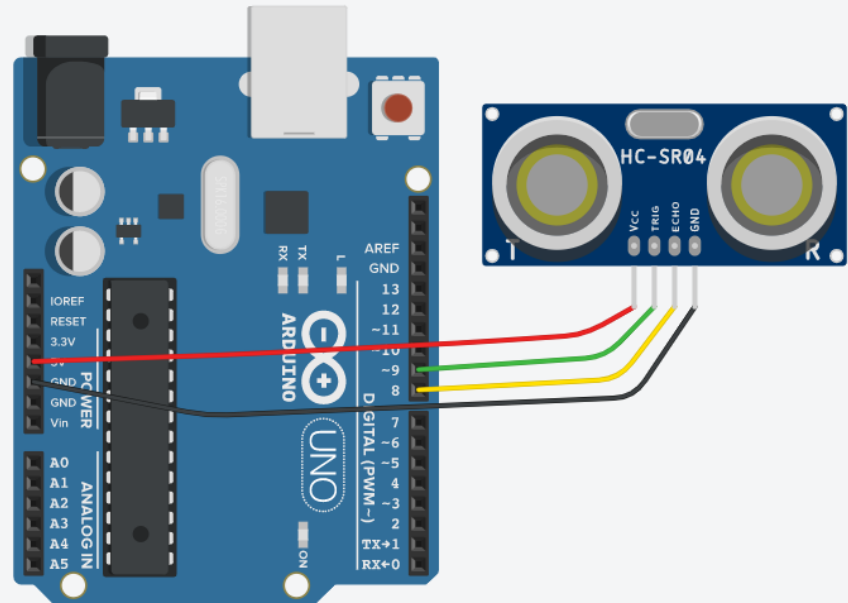
- Ultrasonic sensors use the same principle bats (flying mammals) use to detect obstacles – by sending ultrasonic signals and analysing the reflection / echo
- The time between sending a signal (a trigger pulse of width 10 μ s) and receiving an echo is used to calculate the distance from an object
- HC-SR04 needs +5V DC, works for obstacles at an angle $<15^\circ$, for distances between 2cm – 400 cm
- Practically, 3-300 cm is fairly accurate (up to 0.3cm). If it is too close or too far, no meaningful echo will be received, and the reading might show a high / inaccurate distance
- Distance = (time taken from trigger to echo/2) x speed of sound
 - Speed of sound is about 345 m/s at 23°C

Ultrasonic Distance Sensor

```
long readUltrasonicDistance(int triggerPin, int echoPin)
{
    pinMode(triggerPin, OUTPUT); // Clear the trigger
    digitalWrite(triggerPin, LOW);
    delayMicroseconds(2);
    // Sets the trigger pin to HIGH state for 10 microseconds
    digitalWrite(triggerPin, HIGH);
    delayMicroseconds(10);
    digitalWrite(triggerPin, LOW);
    pinMode(echoPin, INPUT);
    // Reads the echo pin, and returns the
    // sound wave travel time in microseconds
    return pulseIn(echoPin, HIGH);
}

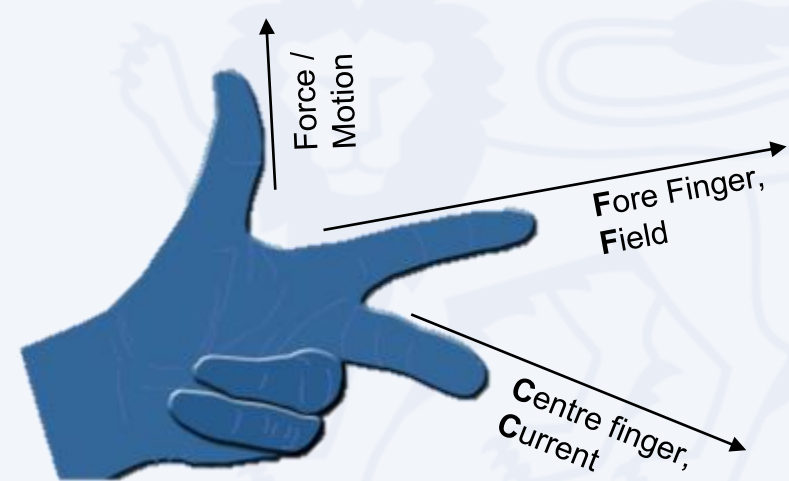
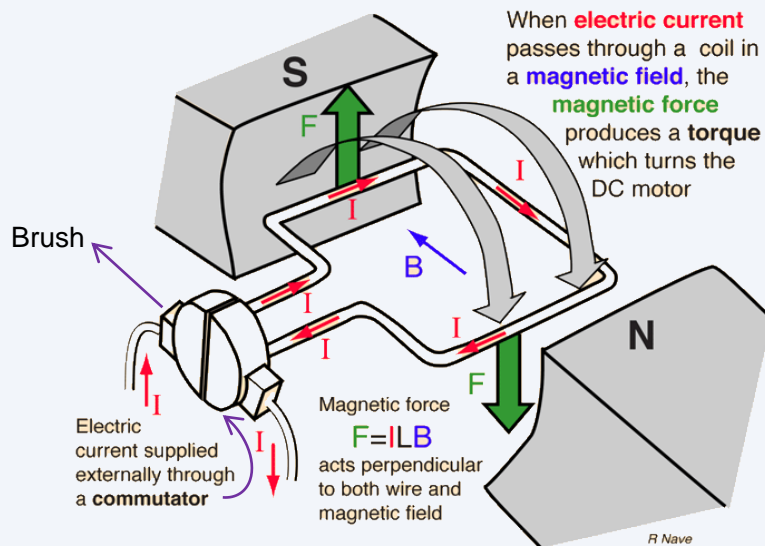
void setup()
{
    Serial.begin(9600);
}

void loop()
{
    Serial.println(0.01723 * readUltrasonicDistance(9, 8));
    // try to make sense of this formula!
    delay(10);
}
```

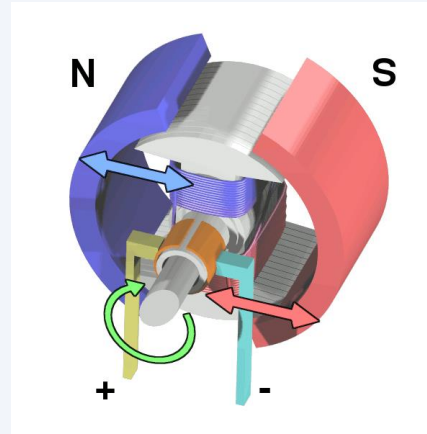


DC Motors : Principle

- If an electric charge moves, it generates its own magnetic field, which interacts with the original field (from permanent magnets)
 - Hence, a magnetic field exerts a force on a moving electric charge
- A DC motor uses a permanent magnet to exert a force on a current-carrying coil of wire
- The direction in which the coil of wire moves can be found using Fleming's left hand rule



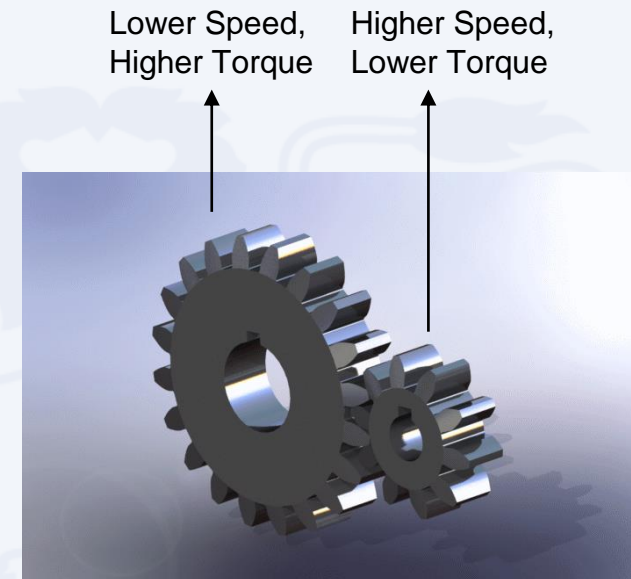
DC Motors



- Easy DIY DC Motor
<https://www.youtube.com/watch?v=DsZCW34LktU>
- DC motors are used in applications which require good energy efficiency, but only imprecise control. Control, if necessary, has to be achieved through some external control mechanism
- Used in fans, drills, traction etc.

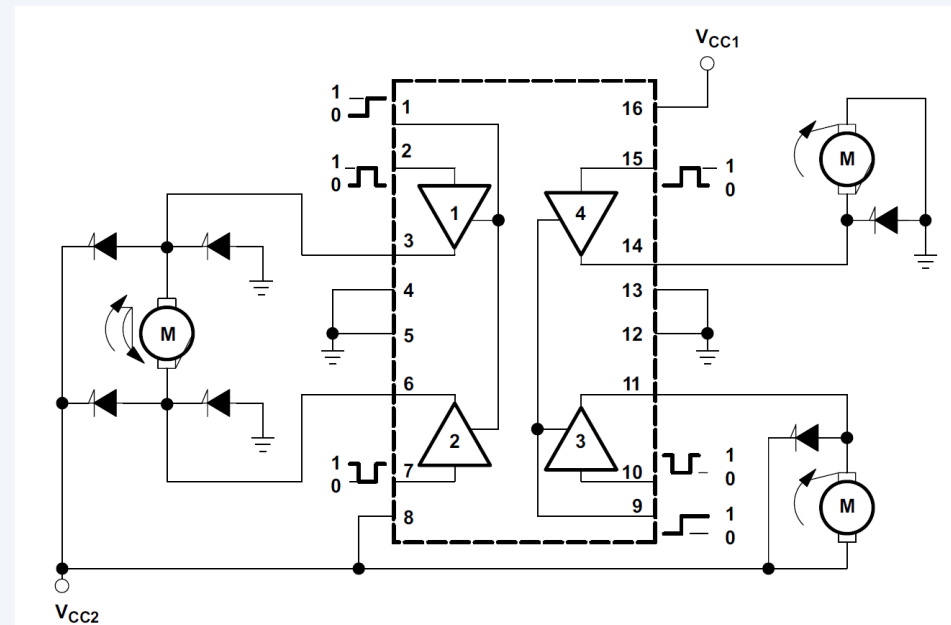
Torque and Speed

- The equivalent of force for rotation is called Torque
- Higher torque applied to the wheel = better ability to overcome obstacles / friction, climb up a slope etc.
- Usual DC motors run at very high rpm, but produces very low torque – typically not enough to run a robot
- We use gears to increase the torque, but this comes at the expense of speed
 - Torque x rotational speed = power drawn by the motor
 - Power drawn by the motor depends on the a number of factors, the most important being the duty cycle (in PWM)
 - Duty cycle can be controlled using `analogWrite()`



Why Motor Driver?

- An Arduino pin can deliver only ~40mA current
- DC motors like the one we use draw several 100s of mA when loaded
- We need a switch/ current amplifier to drive the motor based on the signals from the Arduino
- Arduino -> Driver -> Motor

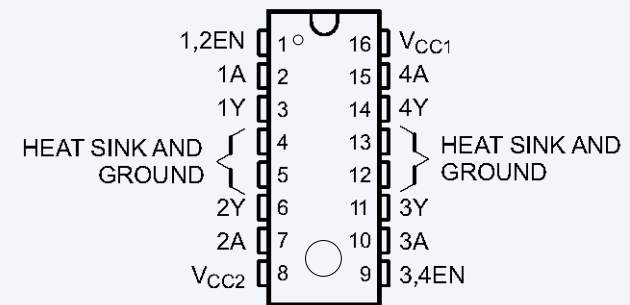


L293D is a popular motor driver
A motor driver allowing bidirectional drive is called an *H-bridge*

Note : L293D has diodes are already connected inside the chip and need not be connected externally

L293D Motor Driver

- L293D has 4 switches. This can be used to drive 4 motors unidirectionally or 2 motors bidirectionally
 - Note that the enables for inputs 1 & 2 are combined (you probably won't need it, but enable should be high for the PWM signal from Arduino to have any effect). Ditto for enables of inputs 3 & 4



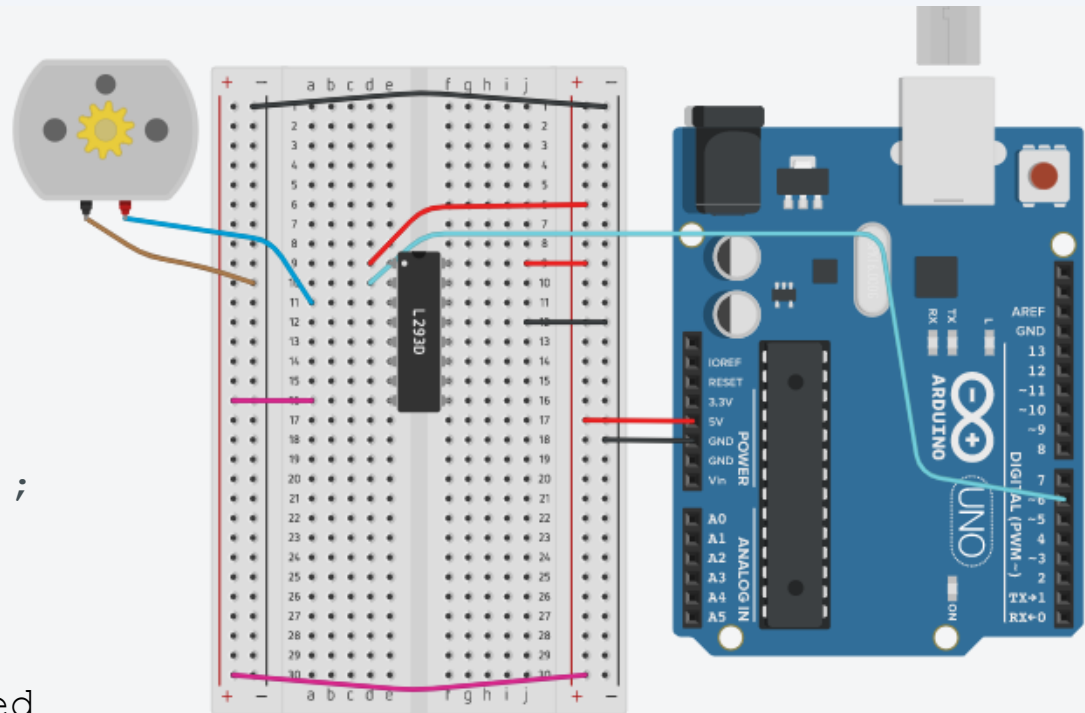
Pin Number	Pin Function	Description
1	Enable 1,2	This pin enables the input pin Input 1(2) and Input 2(7). Usually connected to 5V of Arduino; can also be controlled from your program by connecting it to an Arduino digital output pin.
2	Input 1	PWM control for Output 1 pin. From an Arduino pin supporting analogWrite()
3	Output 1	Connected to one end of Motor 1
4	Ground	Ground pins are connected to ground of circuit (0V)
8	Vcc2 (Vs)	Connected to the power source for running motors (4.5V to 36V)
16	Vcc1 (Vss)	Connected to +5V of Arduino to enable IC function

Unidirectional Control

```
void setup()
{
  pinMode(6, OUTPUT);
}

void loop()
{
  analogWrite(6, speed);
  delay(10);
}

/* speed is 0 to 255
depending on the desired
speed / power */
```

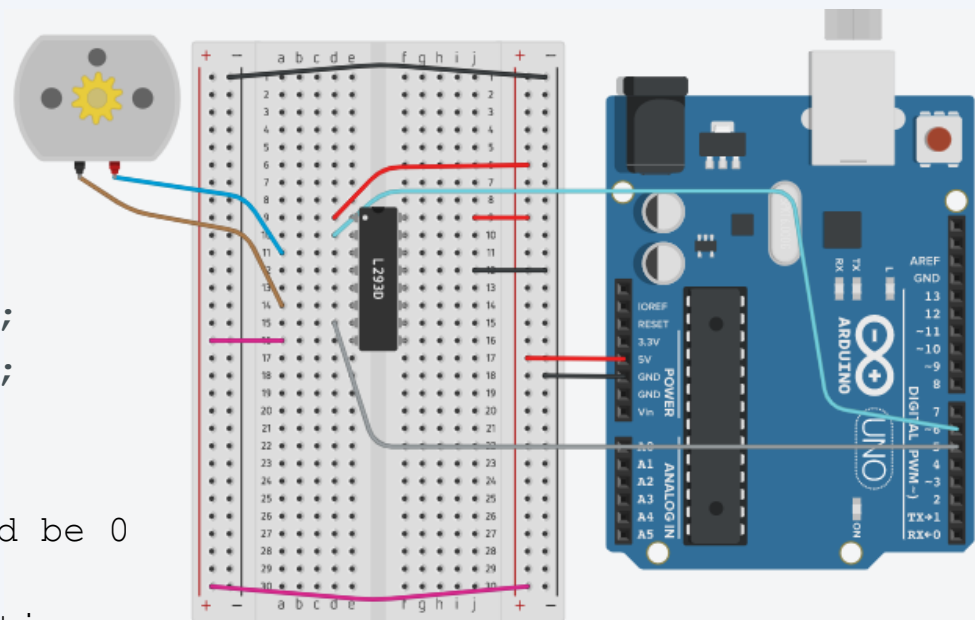


Bidirectional Control

```
void setup()
{
    pinMode(5, OUTPUT);
    pinMode(6, OUTPUT);
}

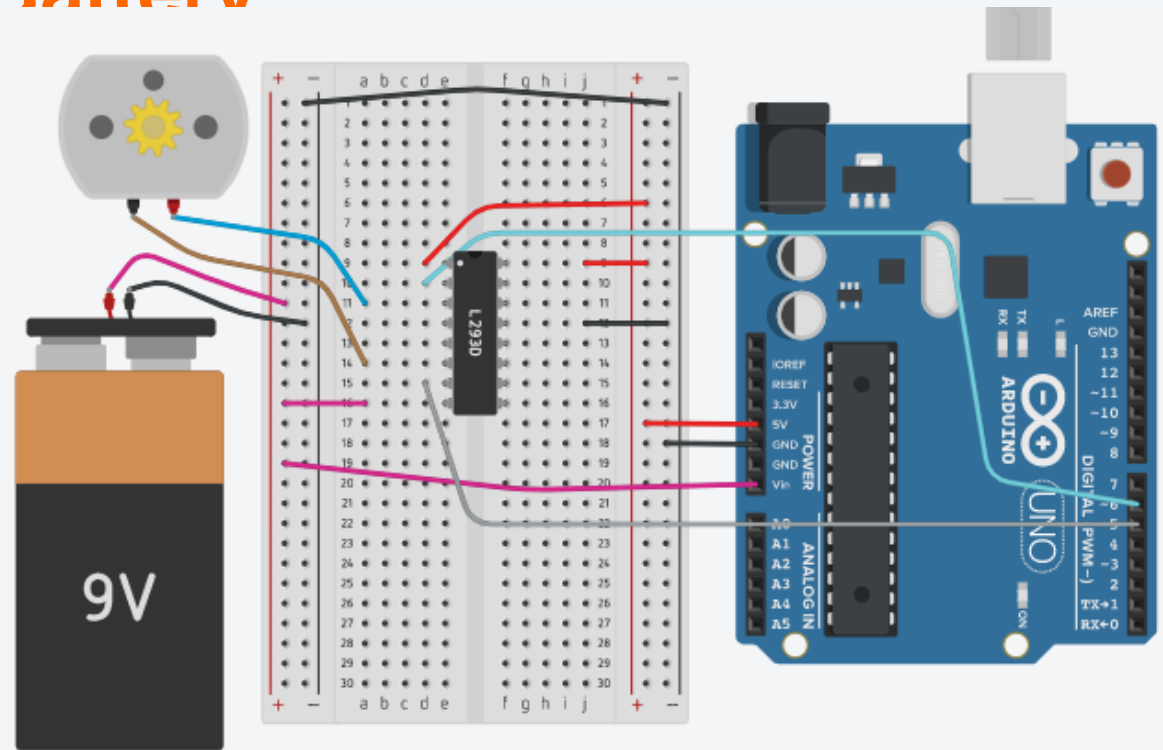
void loop()
{
    analogWrite(5, dir2speed);
    analogWrite(6, dir1speed);
    delay(10);
}

/*dir1speed and dir2speed should be 0
to 255. Effective speed will be
(dir1speed-dir2speed). For rotation
in direction 1, dir1speed is set to
the desired speed and dir2speed is
set to 0 (and vice versa) */
```



DC Motor with Battery

- It is better to power the motor directly from the battery rather than the 5V terminal of the Arduino
 - The Arduino 5V pin might not be able to give sufficient current
- Pin 8 of L293D is connected directly to the +ve of the battery.
 - Here, Arduino board is purely a controller for the motor, and does not power it
- The -ve of the battery and GND of the Arduino have to be connected together
- The battery can also power the Arduino, connect the +ve of the battery to Vin



Exercise 5

- Create a program such that the motor rotates clockwise at the maximum speed when the distance from an obstacle is less than 30cm, and rotates anticlockwise at half the maximum speed when the distance from the obstacle is greater than or equal to 30cm

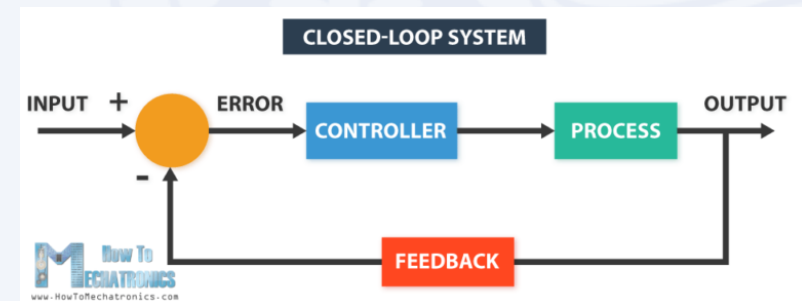
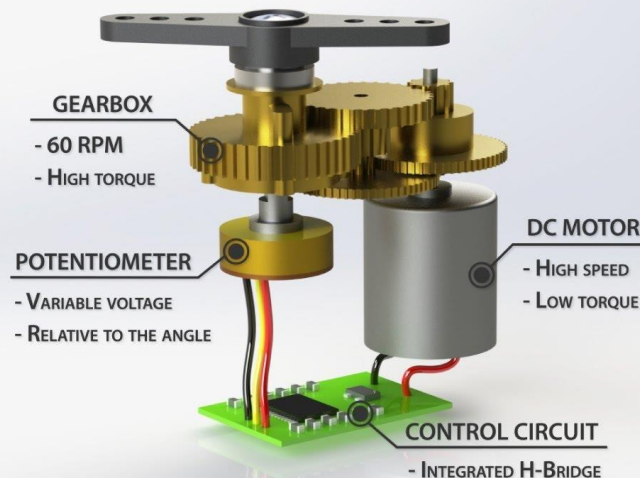
Hints :

- Combine the codes and circuits in slides 3 and 10, and use an 'if' condition
- `analogWrite()` with values 255,0 (`dir1speed,dir2speed`) and 0,128 respectively can be used for clockwise and anticlockwise rotation. The speed is not perfectly linearly related to the value (duty cycle), and that is ok for this exercise
- Easy to do, solution not provided

Servo Motors



- Standard servos are used to maintain a certain angle, not meant for full rotation
- Uses closed loop feedback control
- Servos are used in applications requiring high torque, accurate rotation within a limited angle such as Robotic arms, valve control, rudder control etc.

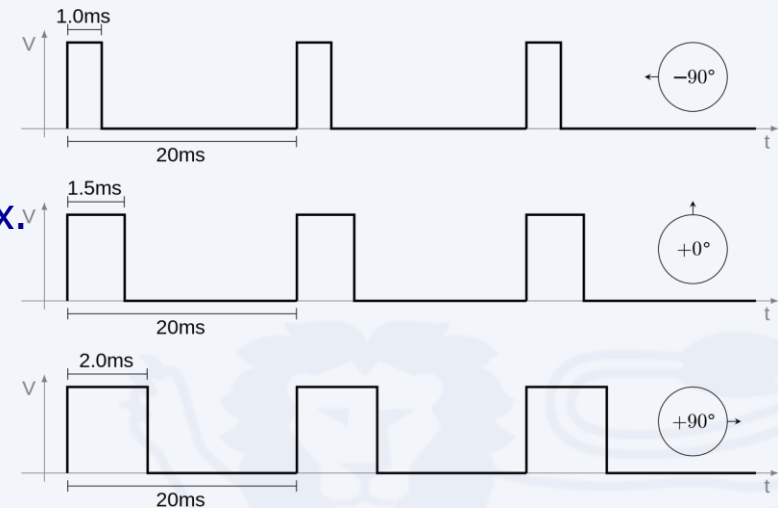


Courtesy : <https://howtomechatronics.com/how-it-works/how-servo-motors-work-how-to-control-servos-using-arduino/>

Servo Motor – SG90

- SG90 is a servo motor which operates based on PWM control signals
- The servo maintains a certain angle (position) based on the width of the pulse fed in through a signal input
- Some technical specifications
 - Weight: 9 g
 - Dimension: 22.2 x 11.8 x 31 mm approx.
 - Stall torque: 1.8 kgf·cm
 - Operating speed: 0.1 s/60 degree
 - Operating voltage: 4.8 V (~5V)

544 – 1500 – 2400
0° – 90° – 180° μs



Note : The pulse width in the image does not correspond to SG90, for illustration of the concept only.
Image courtesy: Stefan Tauner

- PWM frequency = 50Hz
- Pin configuration: **Yellow / Light Orange / White (Signal), Red / Dark Orange (+5V), Brown/Black (Ground)**

Continuous Rotation Servos

- Continuous rotation servos are normal servos modified to perform *open loop speed control* (instead of *closed loop position control*)
 - Rotation speed and direction are controlled through PWM signals (pulse width) for continuous rotation servos, just like how position is controlled for standard servos
 - Effectively, continuous servos are DC motors with integrated motor drivers and reduction gears in a compact, inexpensive package
 - FS90R continuous rotation operating speed: 110RPM (4.8V); 130RPM (6V)
 - Can continuous rotation servos be used to achieve accurate positioning without any additional hardware?

Servo Library

<http://arduino.cc/en/Reference/Servo>

- This library allows an Arduino board to control servo motors.
- Standard servos allow the shaft to be positioned at various angles, usually between 0° and 180°. Continuous rotation servos allow the rotation of the shaft to be set to various speeds
- Any digital pin on UNO can be used, not necessarily those supporting PWM. However, note that using Servo library disables analogWrite() functionality on pins 9 and 10
- **attach(int)** - attach a servo to an I/O pin, e.g., *servo.attach(pin)*, *servo.attach(pin, min, max)*
servo: a variable of type Servo, *pin*: pin number, default values: min = 544 μ s, max = 2400 μ s
- **write(int)** - write a value to the servo to control its shaft accordingly
 Standard servo - set the angle of the shaft
 Continuous rotation servo - set the speed of the servo
 (0: full speed in a direction, 180: full speed in the other, and around 90: no movement)
 e.g., *servo.write(angle)*, angle = 0 to 180
- **detach()** - stop an attached Servo from pulsing its I/O pin

pulse width

Default:

544	–	1500	–	2400	μ s
0°	–	90°	–	180°	

Servo

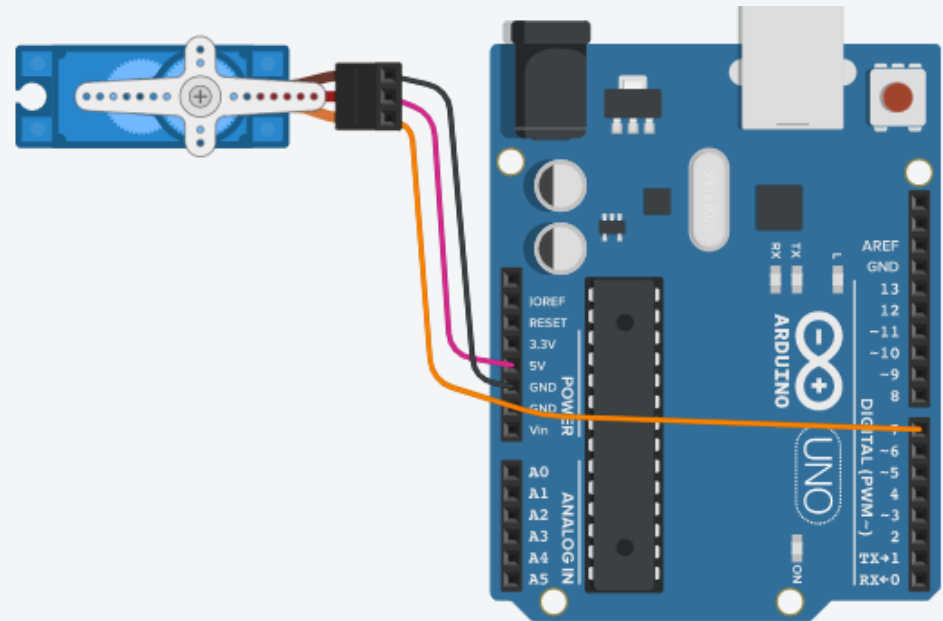
```
#include <Servo.h>

Servo servo_7;

void setup()
{
  servo_7.attach(7);
}

void loop()
{
  servo_7.write(45);
  delay(10);
}

/* servo connected to pin 7
will maintain an angle of 45° */
```



Caution : Do not overload the servo. The servo and your battery / power source could be damaged if servo is overloaded.

Do not power the servo from a 9V battery. Most servos can't take > 6V.

Exercise 6

- Create a program to sweep the servo back and forth between 0° and 180° gradually such that it takes a total of 18 seconds to complete a back and forth sweep

Hint :

- Increment and decrement by 1° , with a 50 milliseconds delay at each step



Exercise 6 Solution

```
#include <Servo.h>

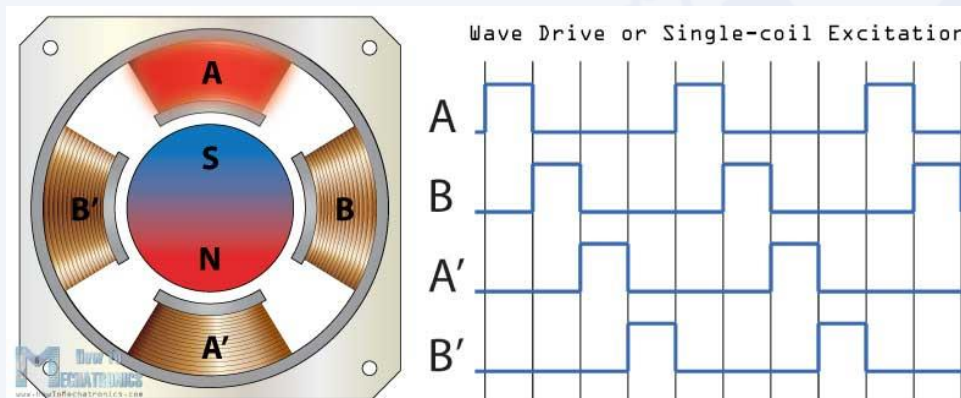
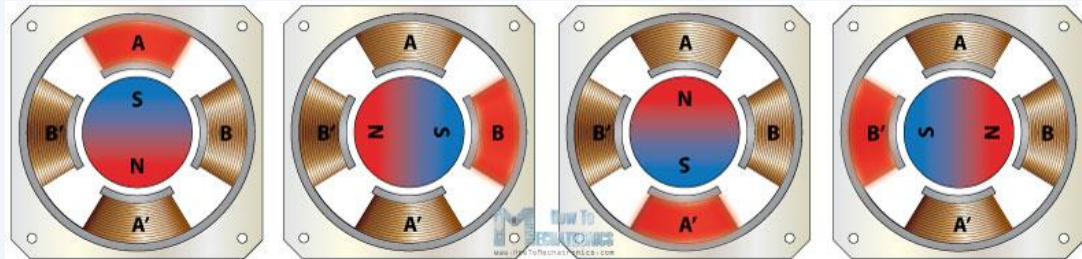
Servo myservo;
int pos = 0;
// variable to store the servo position

void setup()
{
  myservo.attach(7);
}

void loop()
{
  for(pos = 0; pos <= 180; pos++)
  {
    myservo.write(pos);
    delay(50);
  }
  for(pos = 180; pos >= 0; pos--)
  {
    myservo.write(pos);
    delay(50);
  }
}
```

Stepper Motors

- Stepper motors
 - Rotates only by fixed amounts, and controlled by a sequence of digital outputs



Courtesy : <https://howtomechatronics.com/how-it-works/electrical-engineering/stepper-motor/>

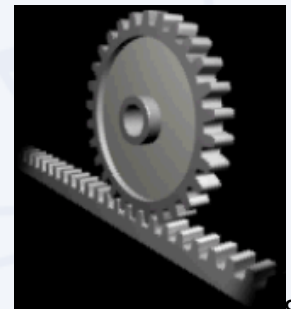
Stepper Motors

- Stepper motors typically (for bipolar steppers) have 4 wires, two each for adjacent coils
- Other, more complicated drive patterns such as full-step drive (two phases on), half-stepping etc. possible
- Step angles of 1.8° , 3.6° , 7.2° etc. are typical
- Open loop control – no feedback-based correction built into the motor unlike servos
- Good for positional control where turn angle is not limited and where the torque required is relatively constant
- Used in printers, 3D printers, *rack and pinion* based actuators

To do : Lookup and go through Arduino Stepper Library

Another good website with animations:

http://www.pcbheaven.com/wikipages/How_Stepper_Motors_Work/



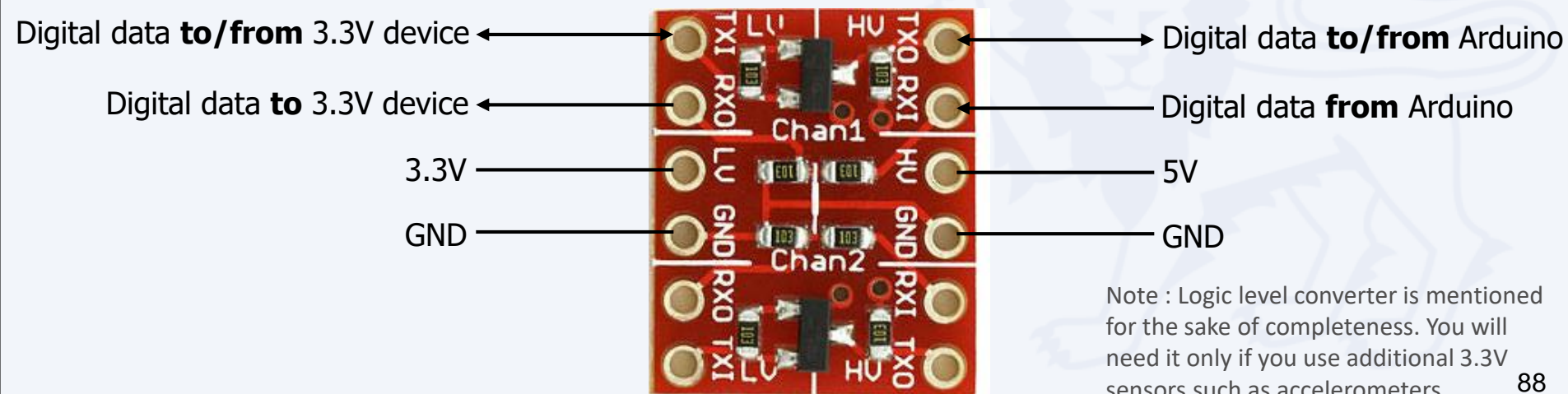
Other type of Motors

- Brushless DC (BLDC) motors
 - Similar to DC motors, but an electronic circuit replaces brush + commutator. Used in electric scooters etc.
- Induction / synchronous motors
 - Used in fans etc, works on AC; the mechanism of operation is somewhat different from that of DC motors

Logic Level Converter

- A level shifter is needed for 5V devices to interface with other lower voltage circuits
- Typically required for interfacing sensors which operate on 3.3V
- For bidirectional systems (signals flow both ways) a level shifter must also be bidirectional
- Level shifters are usually implemented using MOSFETs

Logic Level Converter Board



Arduino vs Raspberry Pi



Arduino UNO



Raspberry Pi

Microcontroller

Suitable for interfacing a fair number of sensors and actuators such as motors, LEDs etc., and to control them with precise (milli/microsecond) timing.

Processes 8 bits at a time @ 16 MHz.

Not suitable for intensive data processing, only basic networking (and that too with additional hardware).

Operates at 5V, fairly robust and doesn't get spoilt that easily.

Typically does not run an operating system; only one program at a time.

Single Board Computer (SBC)

Possible to interface sensors and actuators, though not as good as Arduino in this. Precise timing is harder to achieve.

Processes 32/64 bits at a time @ 700 MHz - 1.5 GHz (depending on model).

Suitable for somewhat intensive data processing such as audio / videos / images, network-intensive applications (has built-in network hardware).

Operates at 3.3V (though it needs a 5V supply), not tolerant of 5V signals on pins and spoils easily if not careful.

Runs a full Linux OS; can be used like a normal computer.

Thank You!!

