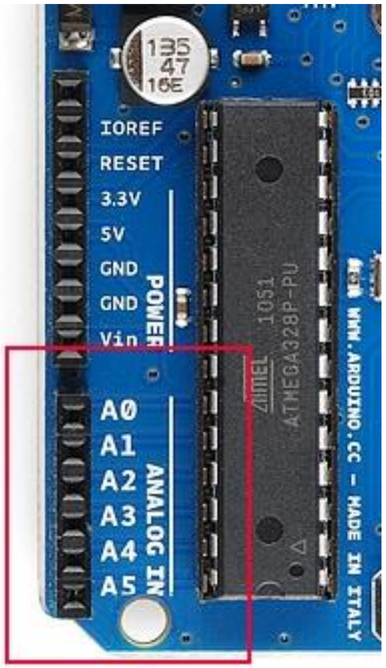
Dr. Hiran Ekanayake

ANALOG INPUT

Analog Input

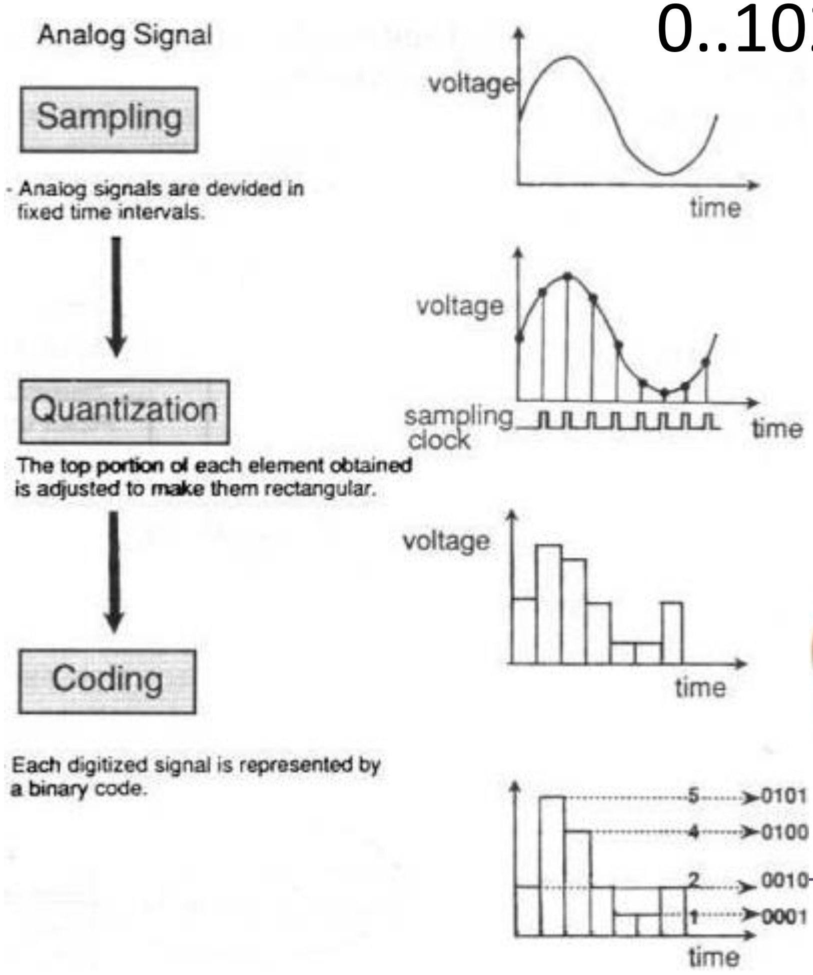
- Which pins can be used for analog input?
- What is an analog input?
- How do Arduino read an analog input?
- How do Arduino represent an analog input?
- What is the accuracy of Arduino's analog input?
- How do ADCs work?
- What are the applications of ADC?



Analog Input

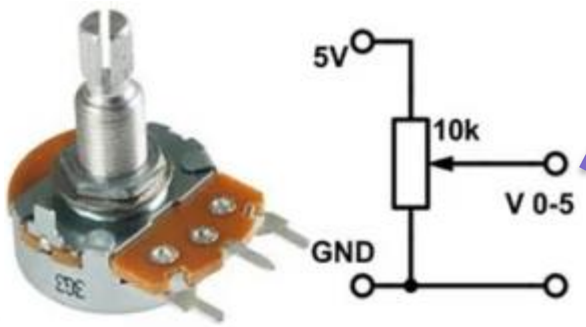
```
int sensorValue = analogRead(A0);
```

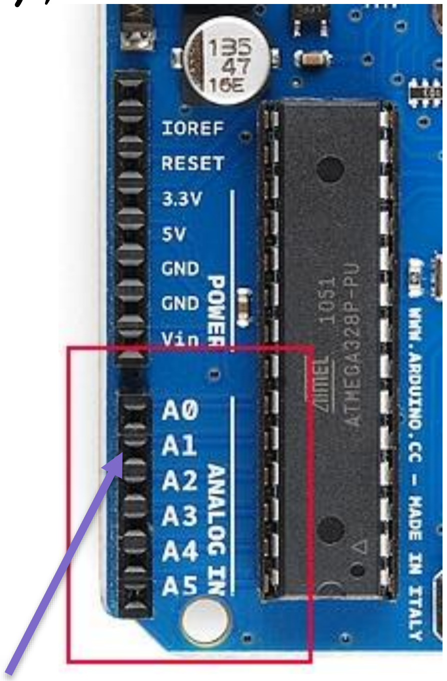
0..1023 ← 0..5V



Analog Signal
Sampling
- Analog signals are devided in fixed time intervals.
Quantization
The top portion of each element obtained is adjusted to make them rectangular.
Coding
Each digitized signal is represented by a binary code.
voltage
time
sampling clock
5
4
2
1
0101
0100
0010
0001

$$ADC = \frac{V_{IN} \times 1024}{V_{REF}}$$

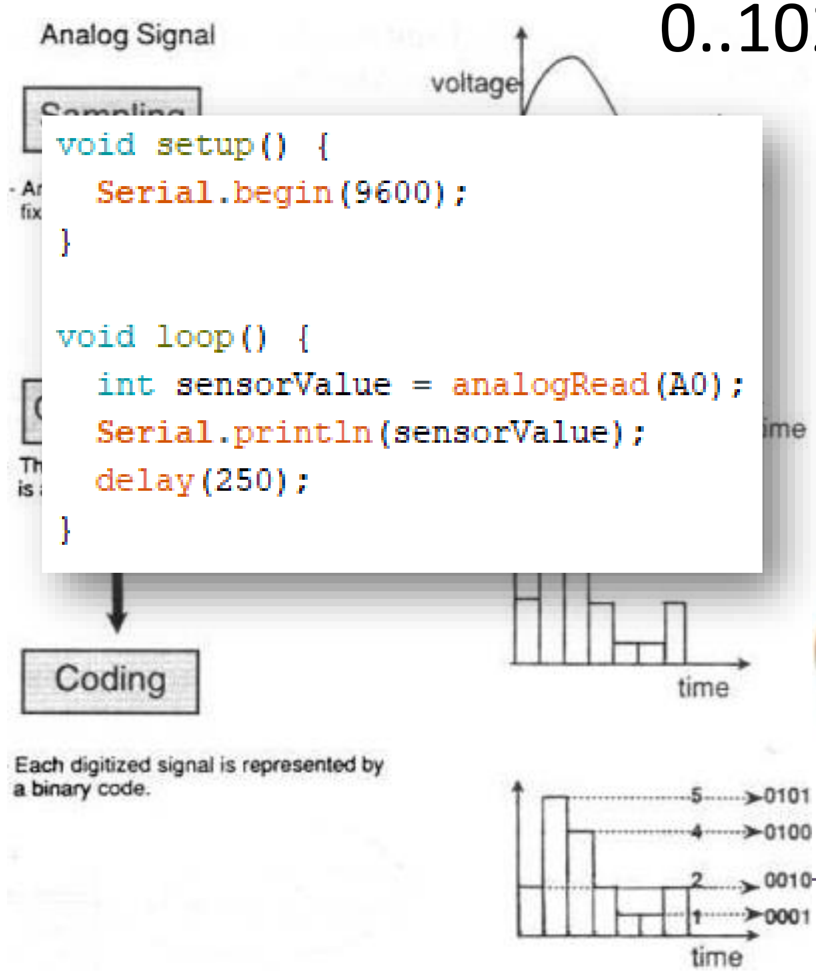




Analog Input

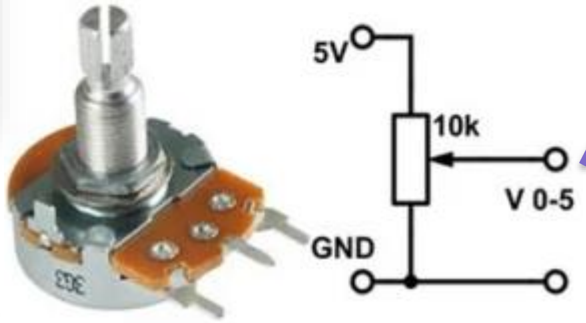
`int sensorValue = analogRead(A0);`

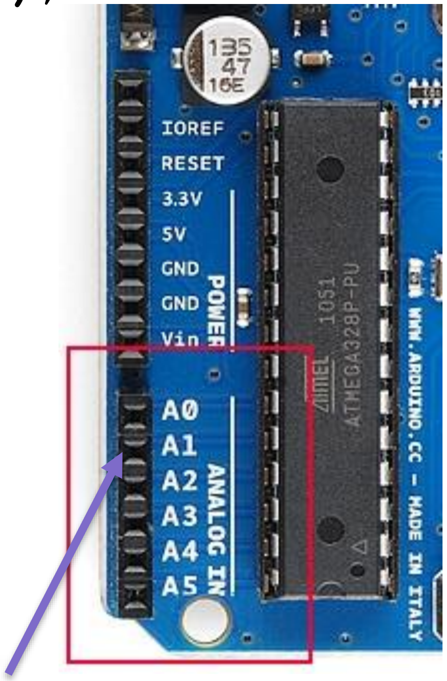
0..1023 ⟵ 0..5V



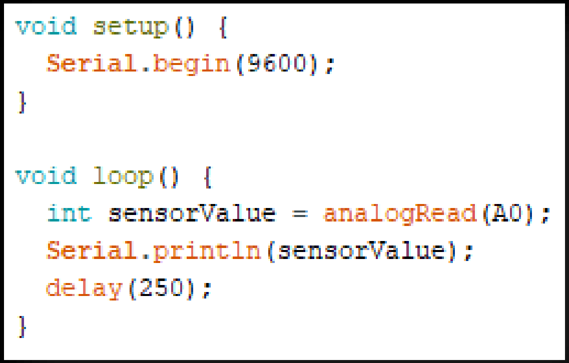
```
void setup() {
  Serial.begin(9600);
}

void loop() {
  int sensorValue = analogRead(A0);
  Serial.println(sensorValue);
  delay(250);
}
```

$$ADC = \frac{V_{IN} \times 1024}{V_{REF}}$$





Analog Input

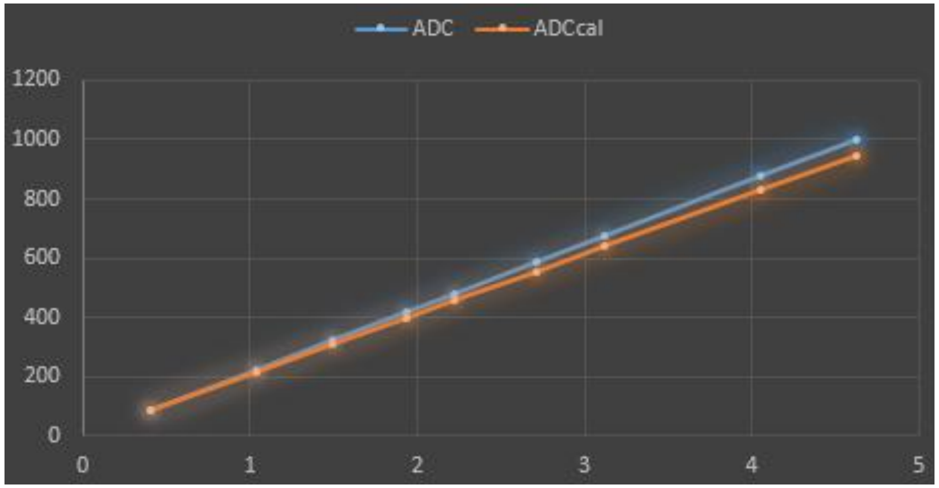


```
void setup() {
  Serial.begin(9600);
}

void loop() {
  int sensorValue = analogRead(A0);
  Serial.println(sensorValue);
  delay(250);
}
```

Vin	ADC	ADCcal	Vin	ADC	ADCcal
0.409	86		2.71	585	
1.035	222		3.12	674	
1.495	321		4.05	875	
1.939	418		4.63	1000	
2.22	480				

$$ADC = \frac{V_{IN} \times 1024}{V_{REF}}$$

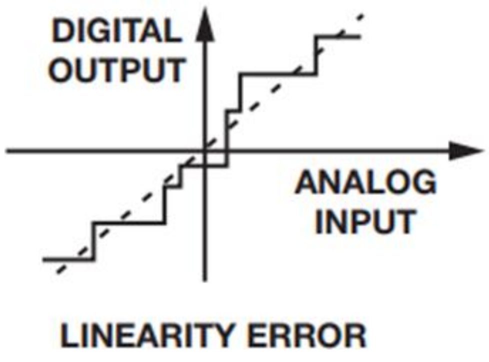
Analog Input

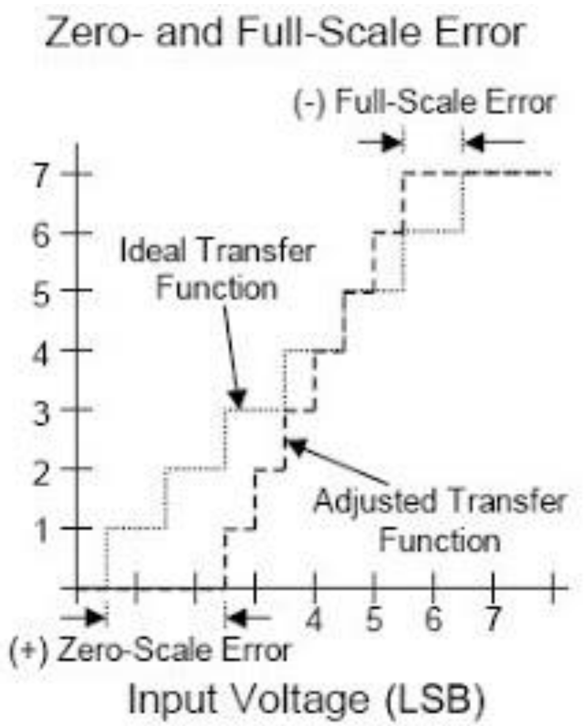


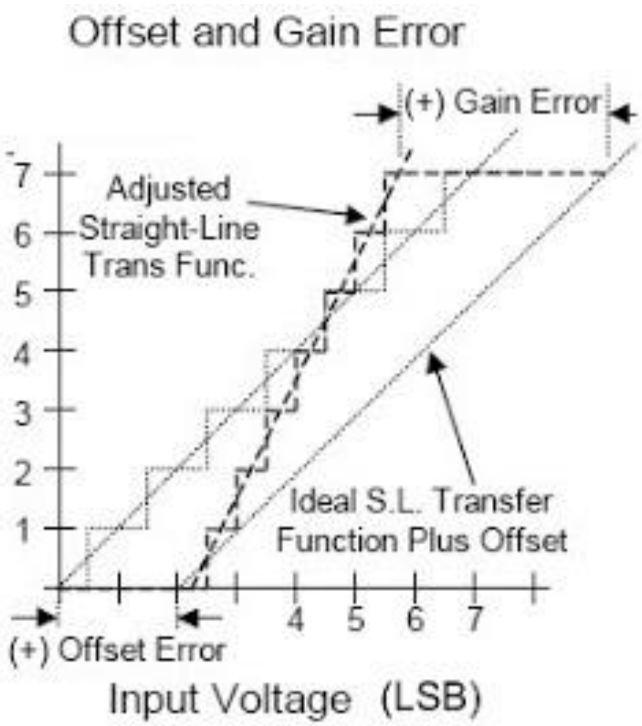
Vin	ADC	ADCcal	Vin	ADC	ADCcal
0.409	86	84	2.71	585	555
1.035	222	212	3.12	674	639
1.495	321	306	4.05	875	829
1.939	418	397	4.63	1000	948
2.22	480	455			

$$ADC = \frac{V_{IN} \times 1024}{V_{REF}}$$

ADC Conversion Errors

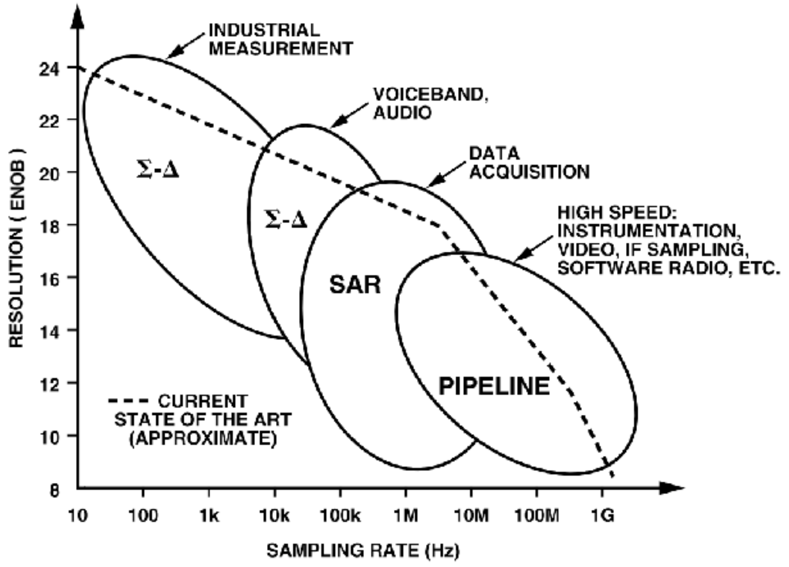


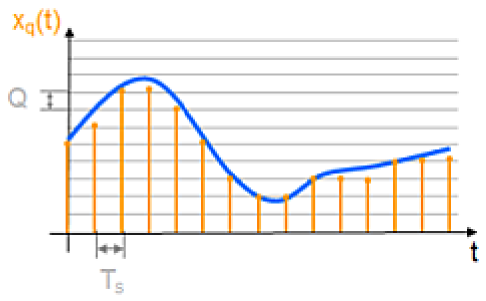


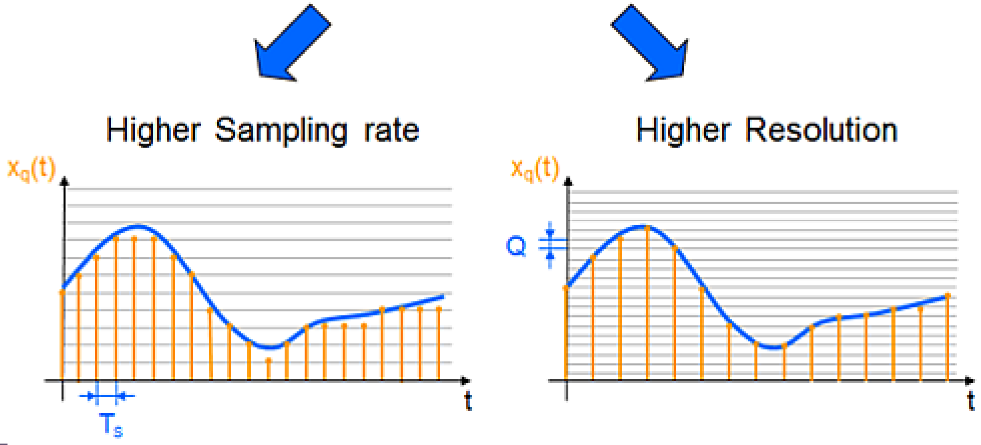


http://www.atx7006.com/articles/static_analysis/adc_parameters

ADC Accuracy







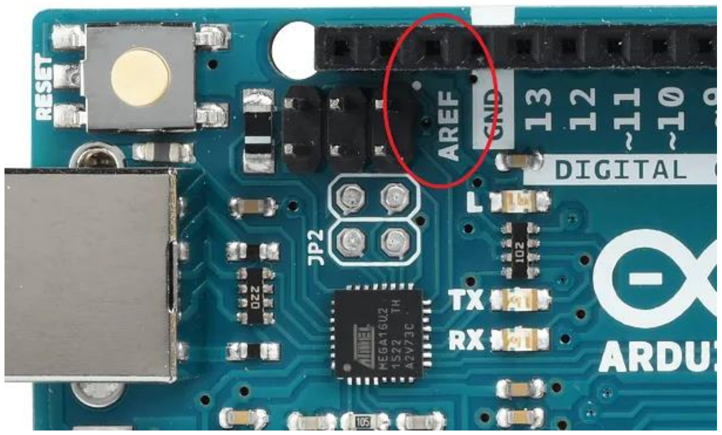
10-bit resolution
0.5 LSB integral non-linearity
±2 LSB absolute accuracy
65 to 260μs conversion time
Up to 15kSPS
6 multiplexed single ended input channels
2 additional multiplexed single ended input channels
Temperature sensor input channel
Optional left adjustment for ADC result readout
0 to V_{CC} ADC input voltage range
Selectable 1.1V ADC reference voltage

ATmega328P

ADS1299: Delta-Sigma ADC, Sim.S, 8xCh, 24 bit, 16 kSPS

AREF: Analog Reference

- What is the purpose of the AREF pin?
- How do you use the AREF pin?

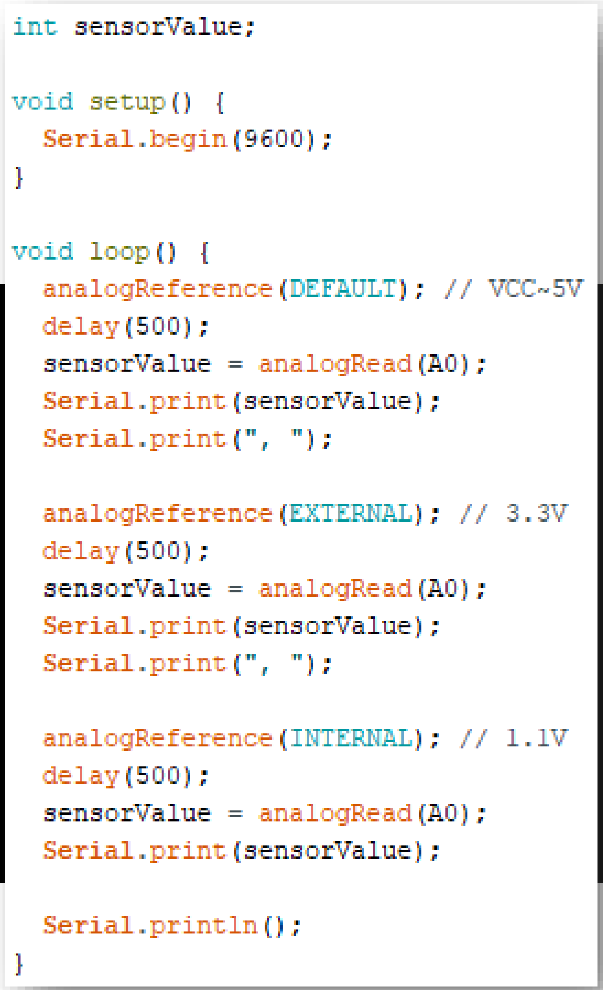


AREF: Analog Reference

- By default, AREF=VCC
- If configured with `analogReference(type)`,
  - DEFAULT: VCC(5V) on Uno
  - INTERNAL: a built-in reference of 1.1V in ATmega328P
  - INTERNAL1V1: a built-in reference of 1.1V (Mega only)
  - INTERNAL2V56: a built-in reference 2.56V (Mega only)
  - EXTERNAL: through a voltage (0-5V) applied to the AREF pin

https://www.arduino.cc/reference/en/language/functions/analog-io/analogreference/

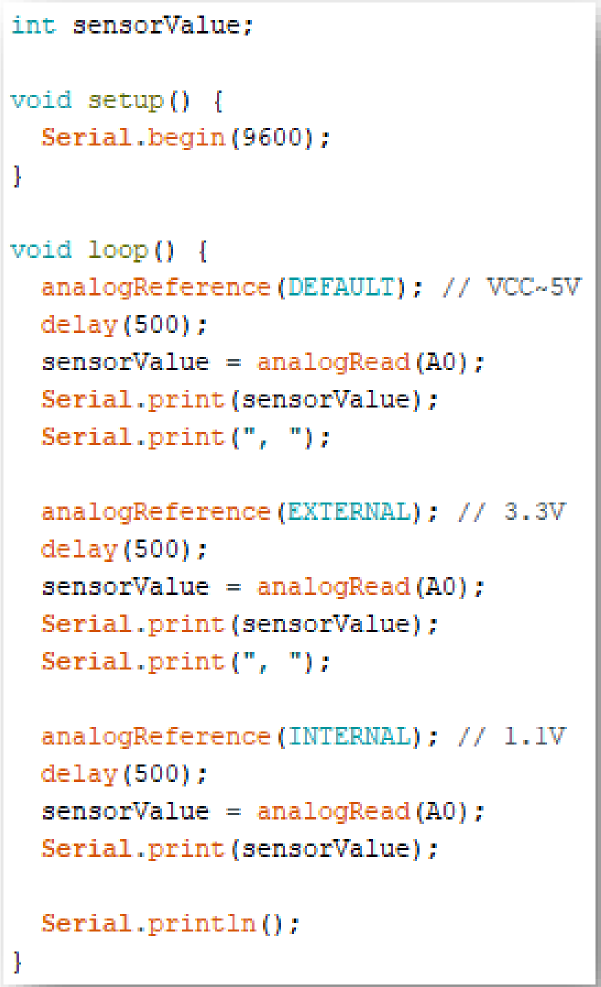
AREF: Analog Reference



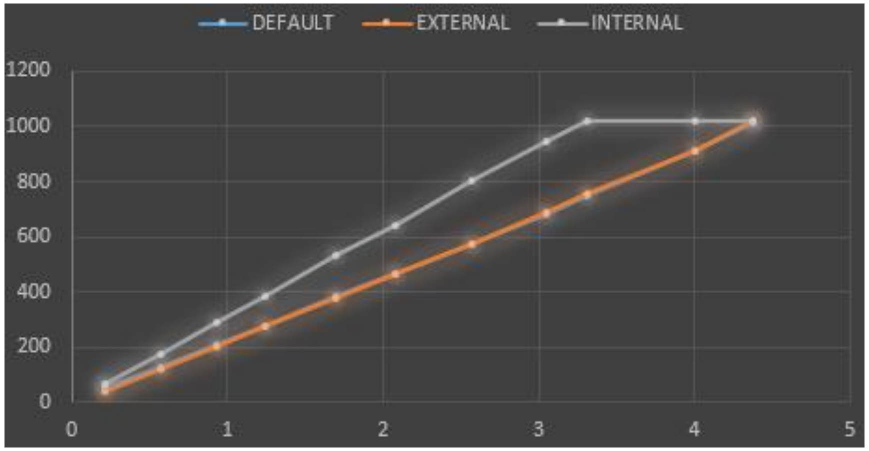
```
int sensorValue;

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

void loop() {
  analogReference(DEFAULT); // VCC~5V
  delay(500);
  sensorValue = analogRead(A0);
  Serial.print(sensorValue);
  Serial.print(", ");

  analogReference(EXTERNAL); // 3.3V
  delay(500);
  sensorValue = analogRead(A0);
  Serial.print(sensorValue);
  Serial.print(", ");

  analogReference(INTERNAL); // 1.1V
  delay(500);
  sensorValue = analogRead(A0);
  Serial.print(sensorValue);

  Serial.println();
}
```

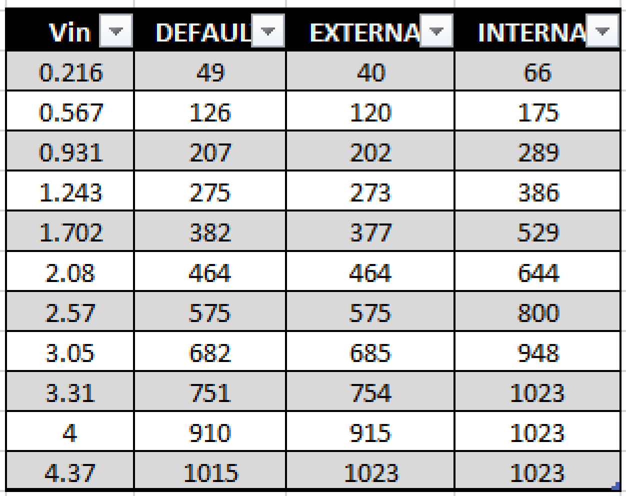
AREF = 3.3V

AREF: Analog Reference



```
int sensorValue;

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

void loop() {
  analogReference(DEFAULT); // VCC~5V
  delay(500);
  sensorValue = analogRead(A0);
  Serial.print(sensorValue);
  Serial.print(", ");

  analogReference(EXTERNAL); // 3.3V
  delay(500);
  sensorValue = analogRead(A0);
  Serial.print(sensorValue);
  Serial.print(", ");

  analogReference(INTERNAL); // 1.1V
  delay(500);
  sensorValue = analogRead(A0);
  Serial.print(sensorValue);

  Serial.println();
}
```





Vin	DEFAUL	EXTERNA	INTERNA
0.216	49	40	66
0.567	126	120	175
0.931	207	202	289
1.243	275	273	386
1.702	382	377	529
2.08	464	464	644
2.57	575	575	800
3.05	682	685	948
3.31	751	754	1023
4	910	915	1023
4.37	1015	1023	1023

Practical Task

- Repeat the analog input experiment by taking at least 20 readings.
  - Calculate different types of errors such as offset, gain, etc.
  - The INTERNAL and EXTERNAL analog references in the second experiment did not give the expected results. Discuss the possible reasons.

Dr. Hiran Ekanayake

ADC APPLICATION: VOLTMETER

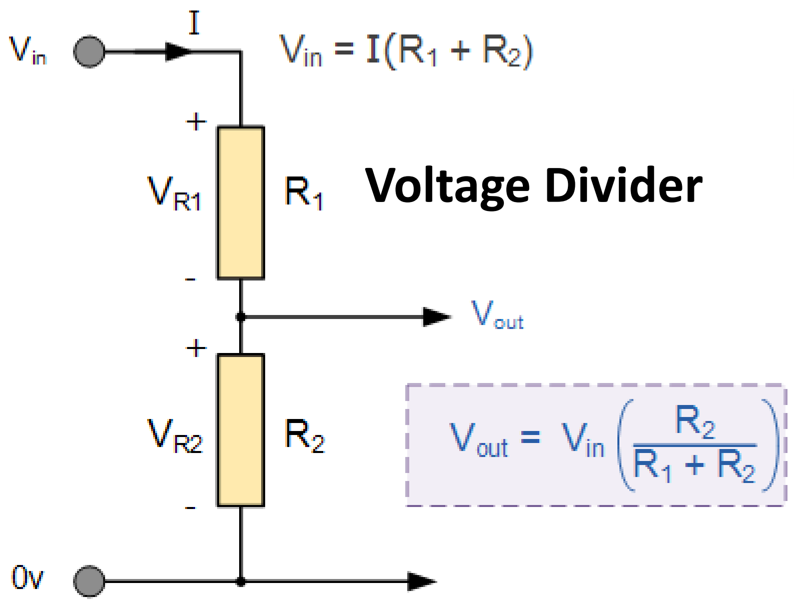
ADC Application: Voltmeter

- Is it possible to use the analog input to measure an unknown voltage?
- Is it possible to measure voltages greater than 5V?

ADC Application: Voltmeter

$$ADC = \frac{V_{IN} \times 1024}{V_{REF}}$$

$$V_{IN} = \frac{ADC * V_{REF}}{1024}$$



Voltage Divider

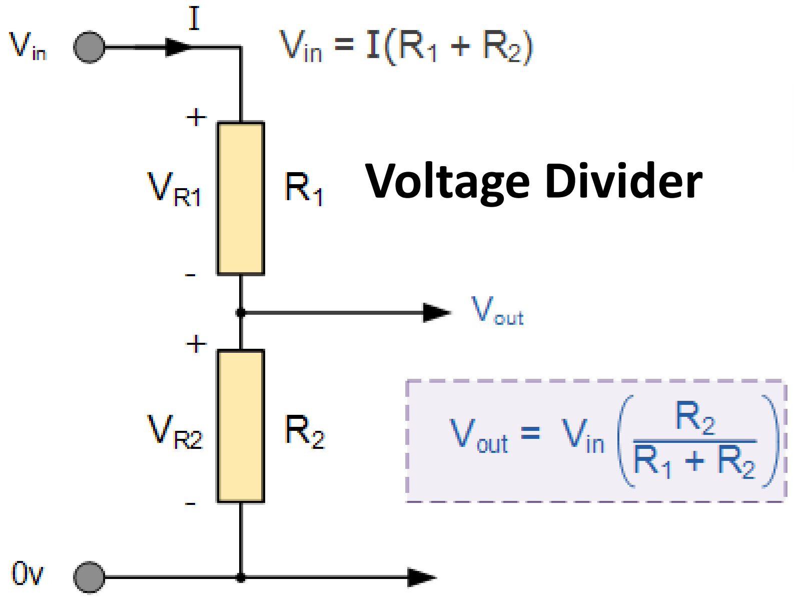
$$V_{out} = V_{in}\left(\frac{R_2}{R_1 + R_2}\right)$$

$$ADC = \frac{V_{IN} * \left(\frac{R_2}{R_1 + R_2}\right) * 1024}{V_{REF}}$$

$$V_{IN} = \left(\frac{R_1 + R_2}{R_2}\right) * \frac{ADC * V_{REF}}{1024}$$

ADC Application: Voltmeter 0-10V

$$ADC = \frac{V_{IN} \times 1024}{V_{REF}} \Rightarrow V_{IN} = \frac{ADC * V_{REF}}{1024}$$



Voltage Divider

$$V_{out} = V_{in}\left(\frac{R_2}{R_1 + R_2}\right)$$

$$\Rightarrow ADC = \frac{V_{IN} * \left(\frac{R_2}{R_1 + R_2}\right) * 1024}{V_{REF}}$$

$$V_{IN} = \left(\frac{R_1 + R_2}{R_2}\right) * \frac{ADC * V_{REF}}{1024}$$

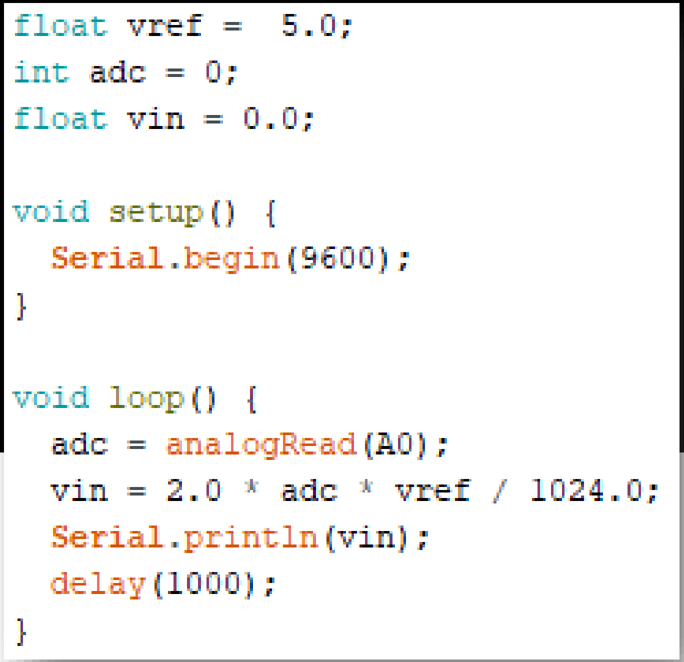
$$\frac{V_{OUT}}{V_{IN}} = \frac{R_2}{R_1 + R_2}$$

$$\frac{5}{10} = \frac{R_2}{R_1 + R_2} \Rightarrow R_1 = R_2$$

Use like 10k resistors

What aspects should be considered when you decide values for the resistors?

ADC Application: Voltmeter 0-10V



```
float vref =  5.0;
int adc = 0;
float vin = 0.0;

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

void loop() {
  adc = analogRead(A0);
  vin = 2.0 * adc * vref / 1024.0;
  Serial.println(vin);
  delay(1000);
}
```

Issues

- How do you improve the accuracy of the voltmeter?
  - VREF value
  - R1 & R2 values
  - Smoothing
- How do you protect the Arduino from measuring voltages beyond the acceptable range?

Practical Task

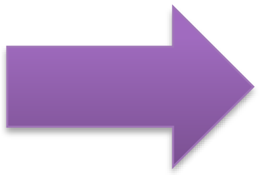
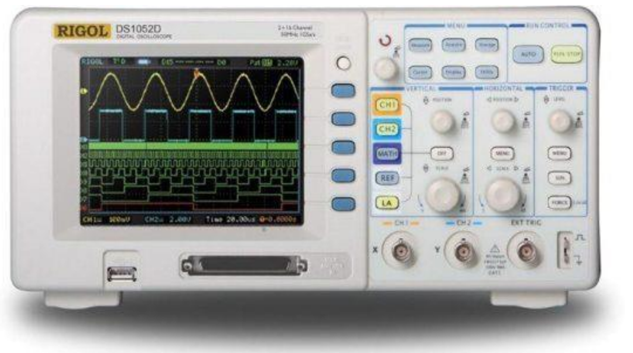
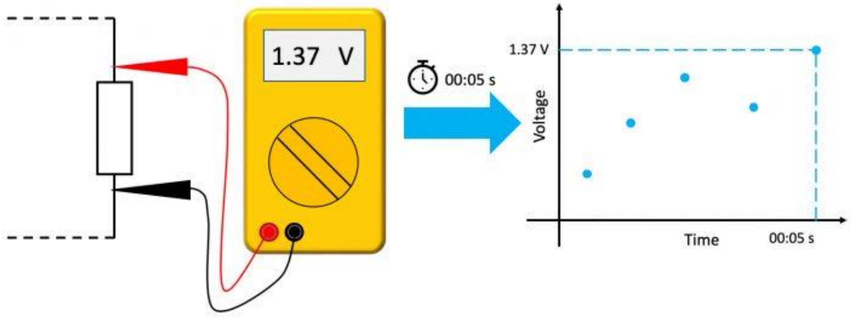
- Design an Arduino Uno based voltmeter which can measure voltages in the range of 0-20V.
- Discuss how you would resolve the issues discussed earlier to improve the accuracy of the measurements.

Dr. Hiran Ekanayake

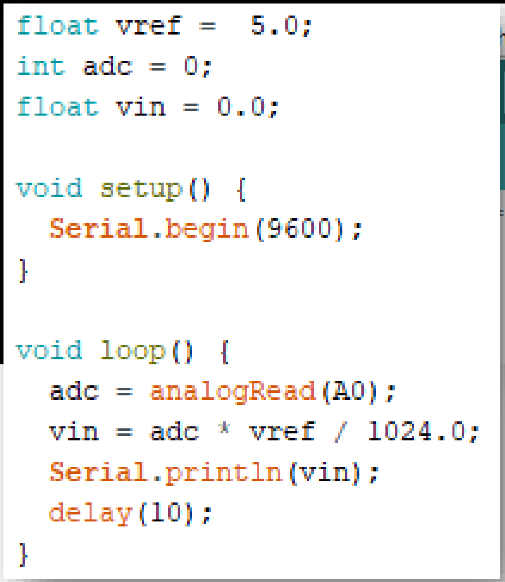
ADC APPLICATION: OSCILLOSCOPE

ADC Application: Oscilloscope

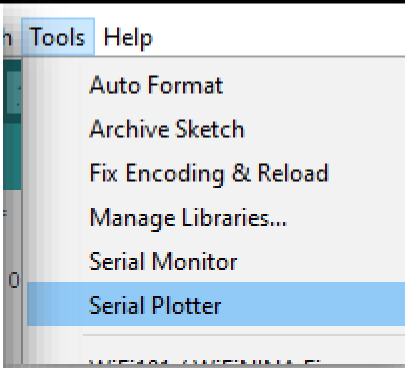
- What is the difference between a voltmeter and an oscilloscope?

Data Visualization



```
float vref =  5.0;
int adc = 0;
float vin = 0.0;

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

void loop() {
  adc = analogRead(A0);
  vin = adc * vref / 1024.0;
  Serial.println(vin);
  delay(10);
}
```

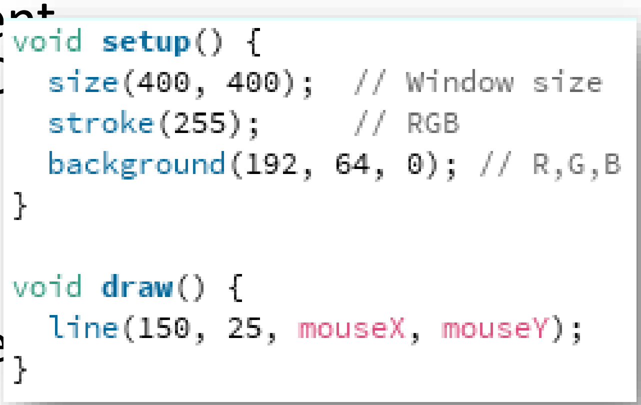


Data Visualization

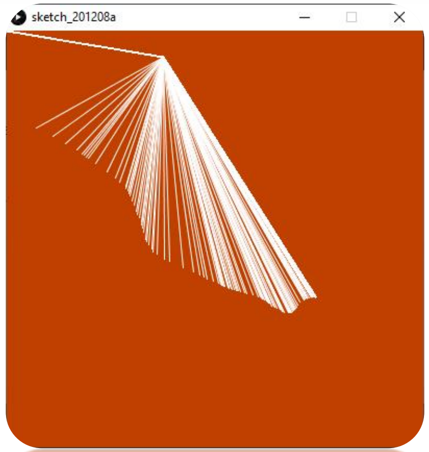
- Are there any programmable tools that can be used to visualize the data coming from Arduino?

Processing @ https://processing.org/

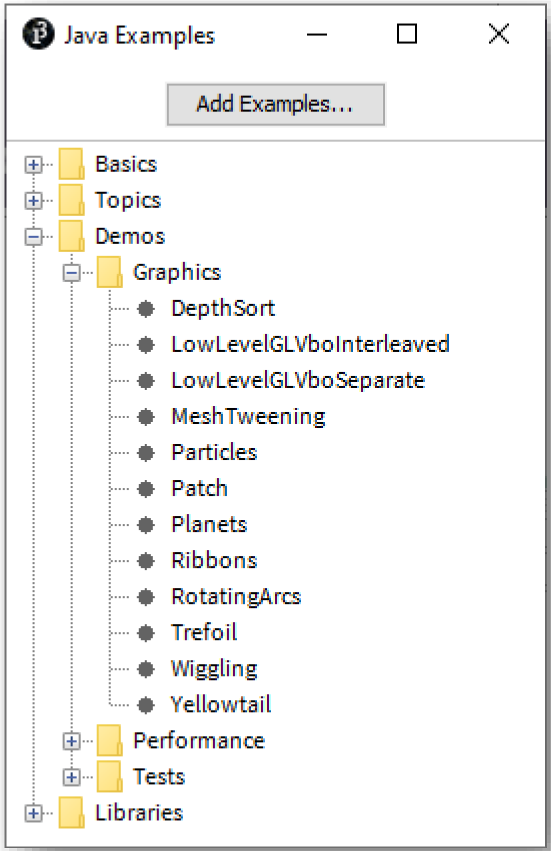
- An opensource simple programming environme for developing visually oriented applications (2D 3D, PDF, SVG, etc.)
- Available for Linux, Mac OS X, and Windows
- It is based on Java
- A Processing program is called a sketch. Sketche are stored in the sketchbook
- The Processing equivalent of a "Hello World" program is simply to draw a line: line(15, 25, 70, 90)
- Interactive programs in Processing are drawn as a series of frames, which can be create by adding functions titled setup() and draw()



```
void setup() {
  size(400, 400);   // Window size
  stroke(255);      // RGB
  background(192, 64, 0); // R,G,B
}

void draw() {
  line(150, 25, mouseX, mouseY);
}
```



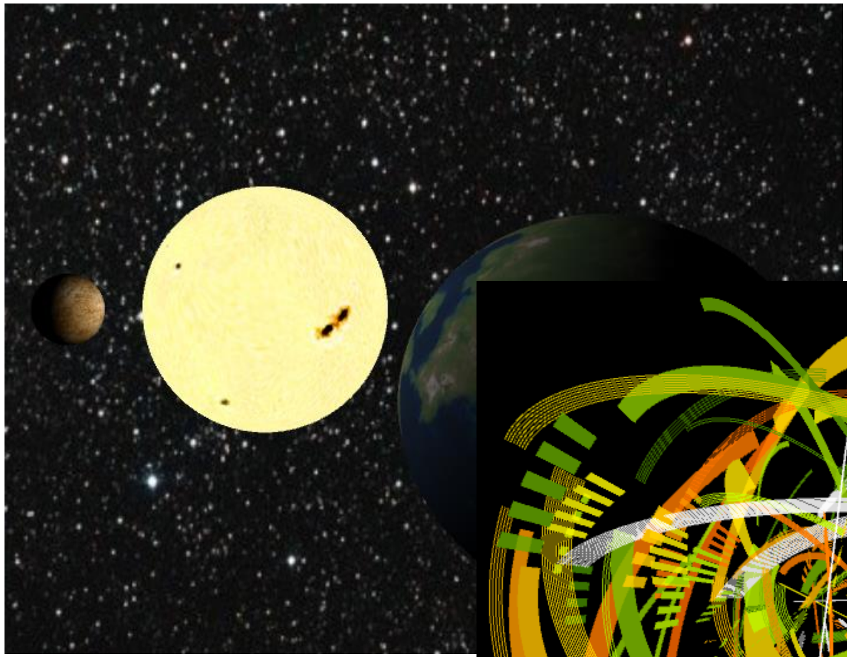
Processing Examples

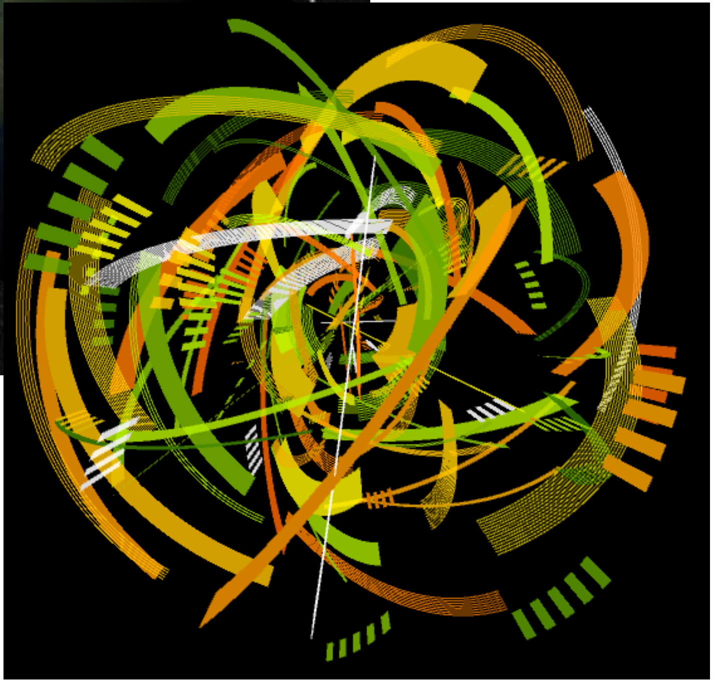


Java Examples

Add Examples...

- Basics
- Topics
- Demos
  - Graphics
    - DepthSort
    - LowLevelGLVboInterleaved
    - LowLevelGLVboSeparate
    - MeshTweening
    - Particles
    - Patch
    - Planets
    - Ribbons
    - RotatingArcs
    - Trefoil
    - Wiggling
    - Yellowtail
  - Performance
  - Tests
- Libraries

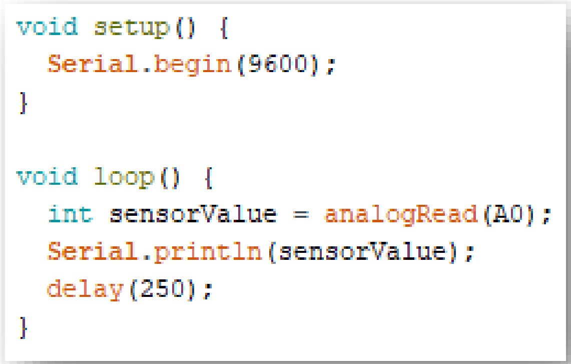




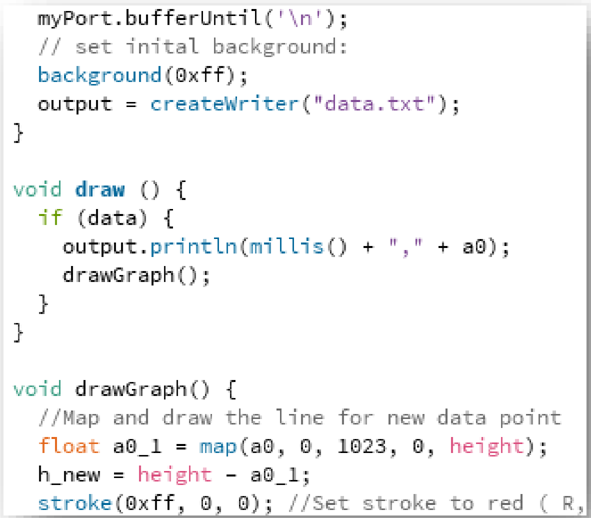
Arduino + Processing

- The IDEs for Processing and Arduino are almost identical
- The Arduino language (based on Wiring) is implemented using C/C++, and therefore has some differences from the Processing language, which is based on Java
  - E.g., Arduino: `Serial.println("hello world");` Processing: `println("hello world");`
  - See https://www.arduino.cc/en/reference/comparison
- Processing is useful when those other computers want to "talk" with an Arduino, for instance to display or save some data collected by the Arduino
- An Arduino board can be directly controlled from Processing without writing code for the Arduino
  - See http://playground.arduino.cc/Interfacing/Processing

Arduino + Processing



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

void loop() {
  int sensorValue = analogRead(A0);
  Serial.println(sensorValue);
  delay(250);
}
```

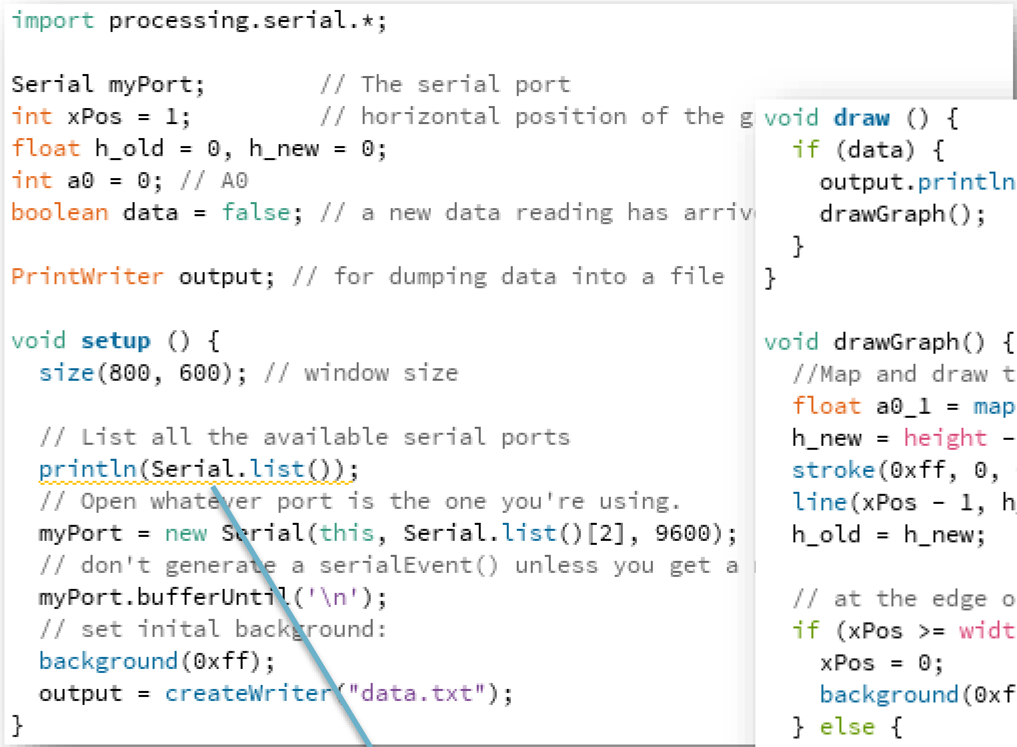


```
  myPort.bufferUntil('\n');
  // set inital background:
  background(0xff);
  output = createWriter("data.txt");
}

void draw () {
  if (data) {
    output.println(millis() + "," + a0);
    drawGraph();
  }
}

void drawGraph() {
  //Map and draw the line for new data point
  float a0_1 = map(a0, 0, 1023, 0, height);
  h_new = height - a0_1;
  stroke(0xff, 0, 0); //Set stroke to red ( R,
```

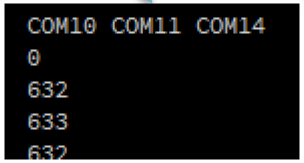
Arduino captures analog data and send to computer through a serial COM port

Processing reads data and displays + save data in a file

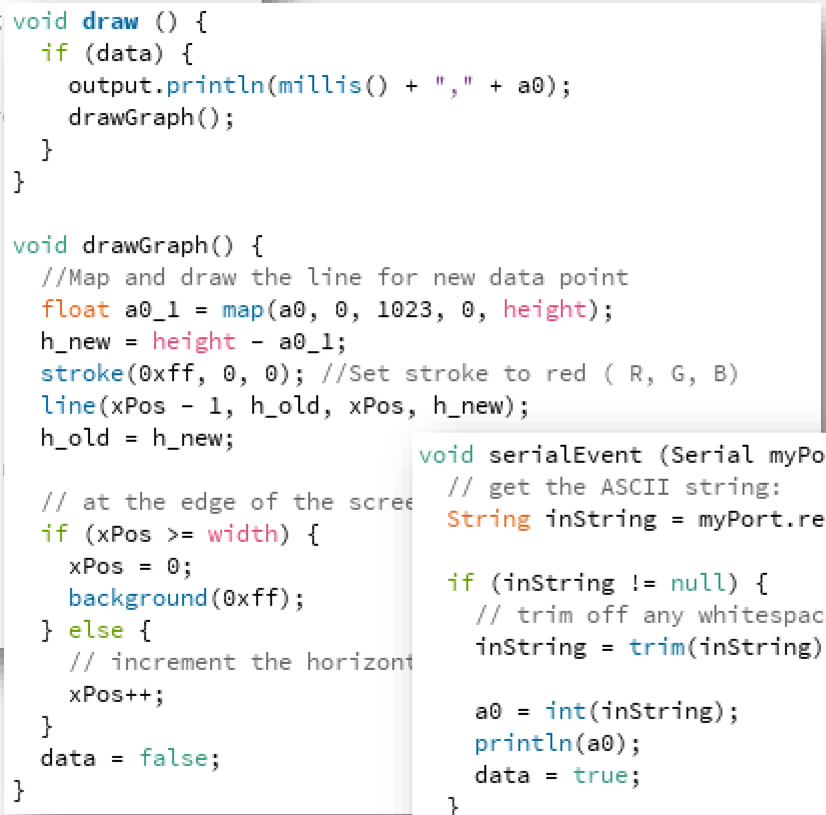
Arduino + Processing



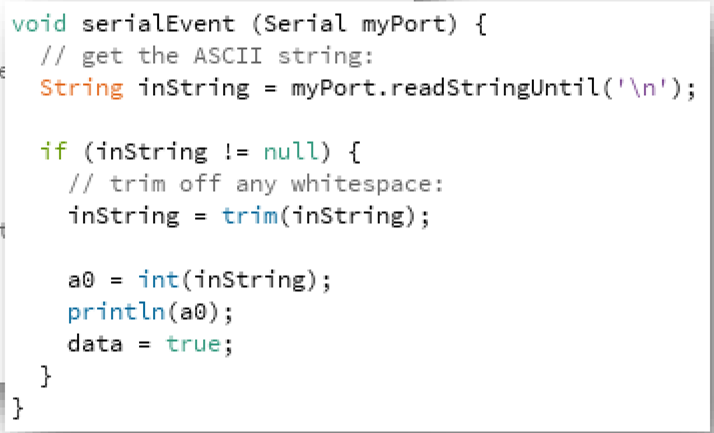
```
import processing.serial.*;

Serial myPort;         // The serial port
int xPos = 1;          // horizontal position of the g
float h_old = 0, h_new = 0;
int a0 = 0; // A0
boolean data = false; // a new data reading has arriv

PrintWriter output; // for dumping data into a file

void setup () {
  size(800, 600); // window size

  // List all the available serial ports
  println(Serial.list());
  // Open whatever port is the one you're using.
  myPort = new Serial(this, Serial.list()[2], 9600);
  // don't generate a serialEvent() unless you get a
  myPort.bufferUntil('\n');
  // set inital background:
  background(0xff);
  output = createWriter("data.txt");
}
```



```
COM10 COM11 COM14
0
632
633
632
```

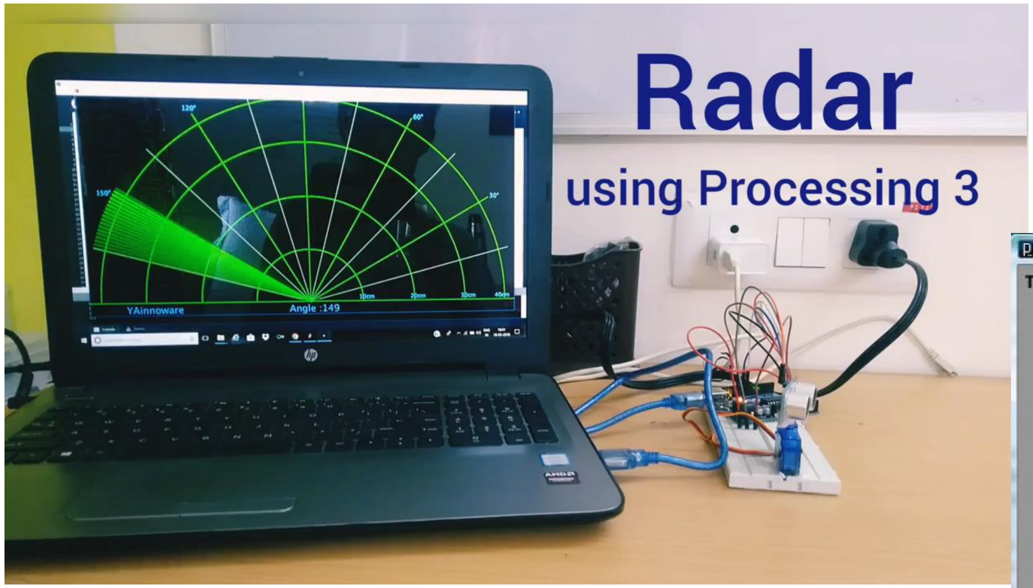


```
void draw () {
  if (data) {
    output.println(millis() + "," + a0);
    drawGraph();
  }
}

void drawGraph() {
  //Map and draw the line for new data point
  float a0_1 = map(a0, 0, 1023, 0, height);
  h_new = height - a0_1;
  stroke(0xff, 0, 0); //Set stroke to red ( R, G, B)
  line(xPos - 1, h_old, xPos, h_new);
  h_old = h_new;

  // at the edge of the scree
  if (xPos >= width) {
    xPos = 0;
    background(0xff);
  } else {
    // increment the horizont
    xPos++;
  }
  data = false;
}
```

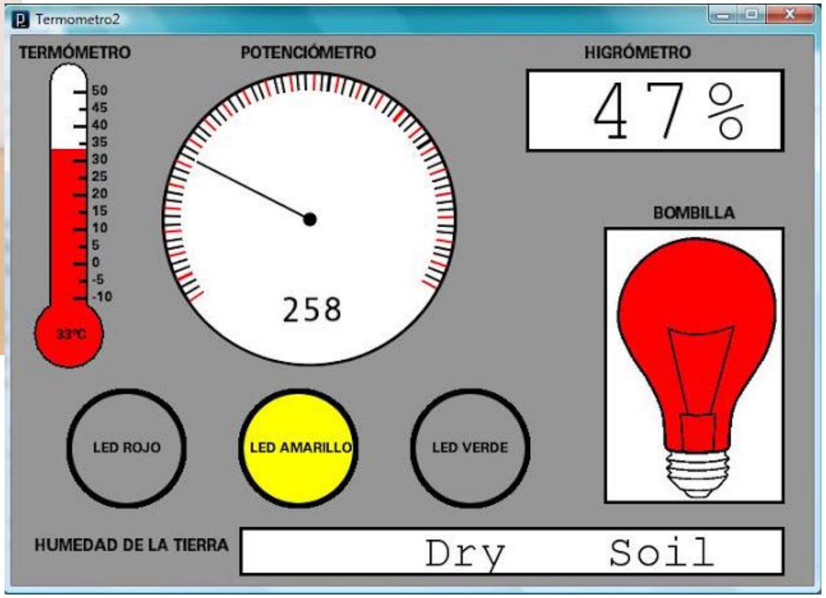


```
void serialEvent (Serial myPort) {
  // get the ASCII string:
  String inString = myPort.readStringUntil('\n');

  if (inString != null) {
    // trim off any whitespace:
    inString = trim(inString);

    a0 = int(inString);
    println(a0);
    data = true;
  }
}
```

Arduino + Processing Projects



https://create.arduino.cc/projecthub/Yug_Ajmera/radar-sonar-using-processing-3-7302c6



https://maker.wiznet.io/2016/05/31/send-data-sensor-from-arduino-to-processing/

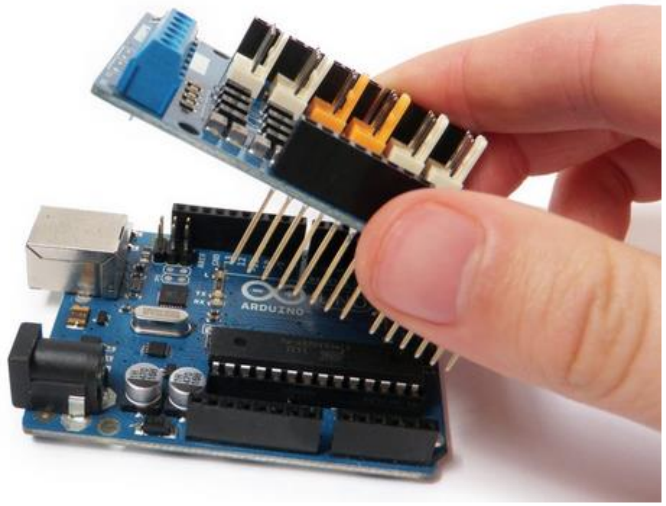
Exercise

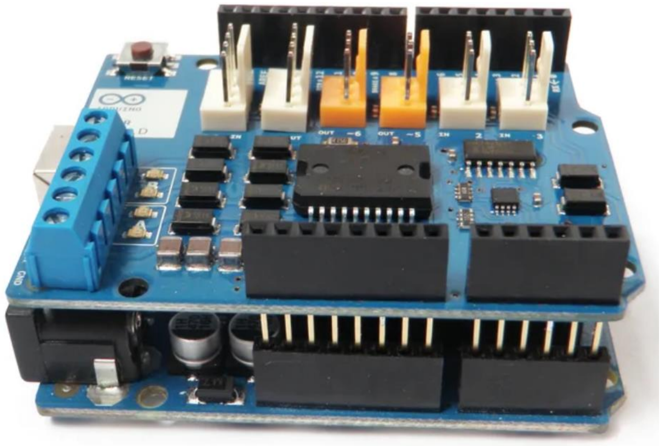
- Briefly describe the differences (purpose, technical, uses, cost, etc.) between the following instruments,
  - Analog multimeter
  - Digital multimeter
  - Analog oscilloscope
  - Digital oscilloscope
  - PC oscilloscope
  - Function generator
  - Logic analyzer
- What are “Periodic Waves”? What are the measurable attributes of periodic waves?
- How do you generate square waves and sine waves using Arduino Uno?

ARDUINO MODULES & SHIELDS

What is an Arduino Shield?

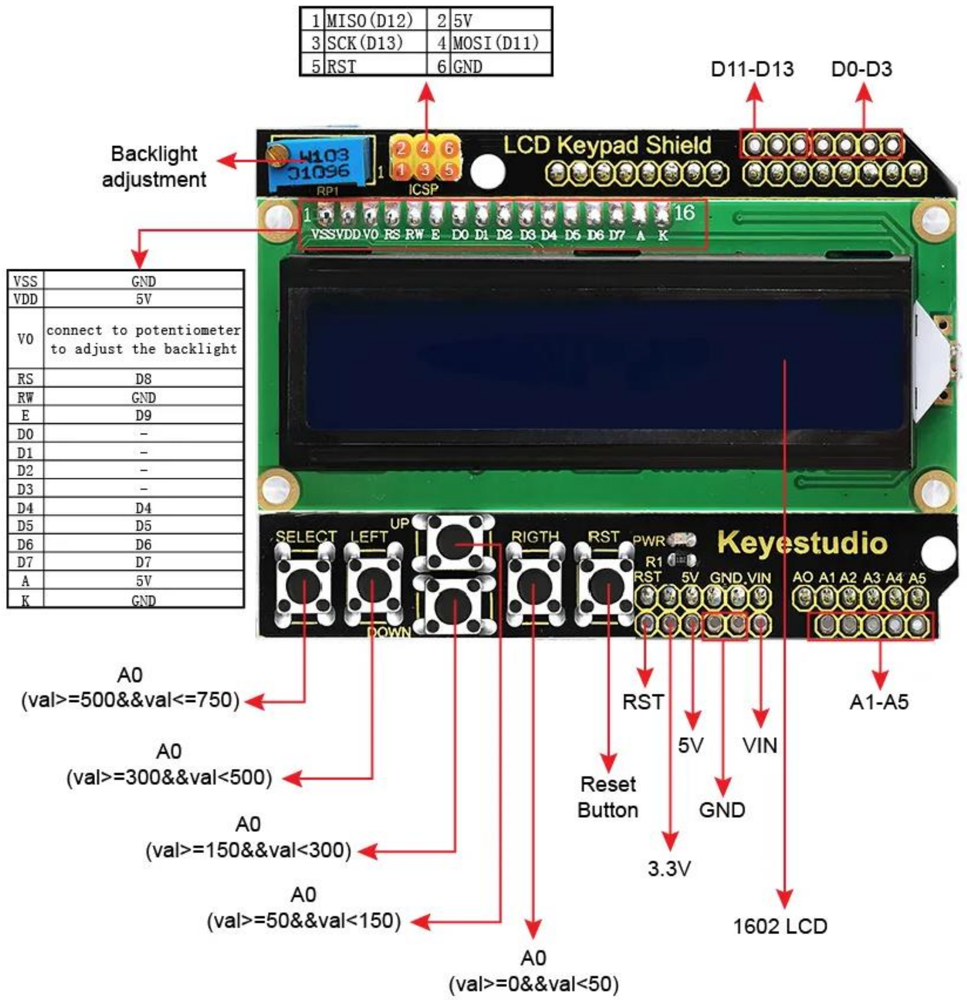
- An Arduino Shield is a hardware expansion board that is designed to fit directly on top of an Arduino microcontroller board, adding new features and functionality to the base system without requiring complex wiring. These shields are modular, stackable, and plug-and-play, making it easy to extend the capabilities of an Arduino without needing a breadboard or additional external components.





1602 16x2 LCD Keypad Shield

- The 1602 16x2 LCD Keypad Shield is a great addition to the Arduino, allowing you to display information on a 16-character by 2-line LCD and interact with your projects using built-in buttons. This shield offers a variety of exciting experiments and projects to get hands-on experience with both output (the LCD display) and input (the keypad buttons).



1	MISO(D12)	2	5V
3	SCK(D13)	4	MOSI(D11)
5	RST	6	GND

VSS	GND
VDD	5V
V0	connect to potentiometer to adjust the backlight
RS	D8
RW	GND
E	D9
D0	–
D1	–
D2	–
D3	–
D4	D4
D5	D5
D6	D6
D7	D7
A	5V
K	GND

RJ
UNO
ARDUINO
LCD1602 Keypad

Feature and Functionality

- Display (1602 16x2 LCD)
- 5 navigation buttons: Up, Down, Left, Right, and Select, along with a reset button
- Pin Usage: The shield typically uses digital pins 4, 5, 6, 7, 8, 9, and 10 for controlling the LCD. The analog pin A0 is used for reading the keypad inputs.
- Need the LiquidCrystal library

Right:	~0
Up:	~144
Down:	~329
Left:	~505
Select:	~742
No button:	1023

A Simple Program for the 1602 16x2 LCD Keypad Shield

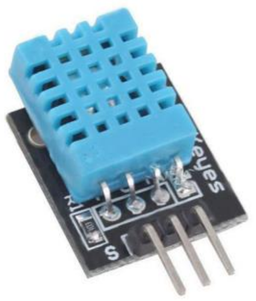
```
#include <LiquidCrystal.h>  // Include the LCD library

// Initialize the library with the numbers of the interface pins
LiquidCrystal lcd(8, 9, 4, 5, 6, 7);

const int buttonPin = A0;  // Keypad is connected to Analog Pin A0
int buttonValue = 0;       // Variable to store button value

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!");
  // Allow time for message to display
  delay(2000);

  // Clear the screen
  lcd.clear();
}
```

1602 16x2 LCD

```
void loop() {
  // Read the button input
  buttonValue = analogRead(buttonPin);

  lcd.setCursor(0, 0);  // Set cursor to the first column, first row

  // Detect which button is pressed based on the analog value
  if (buttonValue < 50) {
    lcd.print("Right ");  // Button "Right" is pressed
  }
  else if (buttonValue < 250) {
    lcd.print("Up    ");  // Button "Up" is pressed
  }
  else if (buttonValue < 450) {
    lcd.print("Down  ");  // Button "Down" is pressed
  }
  else if (buttonValue < 650) {
    lcd.print("Left  ");  // Button "Left" is pressed
  }
  else if (buttonValue < 850) {
    lcd.print("Select");  // Button "Select" is pressed
  }
  else {
    lcd.print("No Press");  // No button pressed
  }

  lcd.setCursor(0, 1);  // Move to the second row
  lcd.print("Analog: ");
  lcd.print(buttonValue);  // Display the raw analog value

  delay(200);  // Small delay to debounce the button
}
```

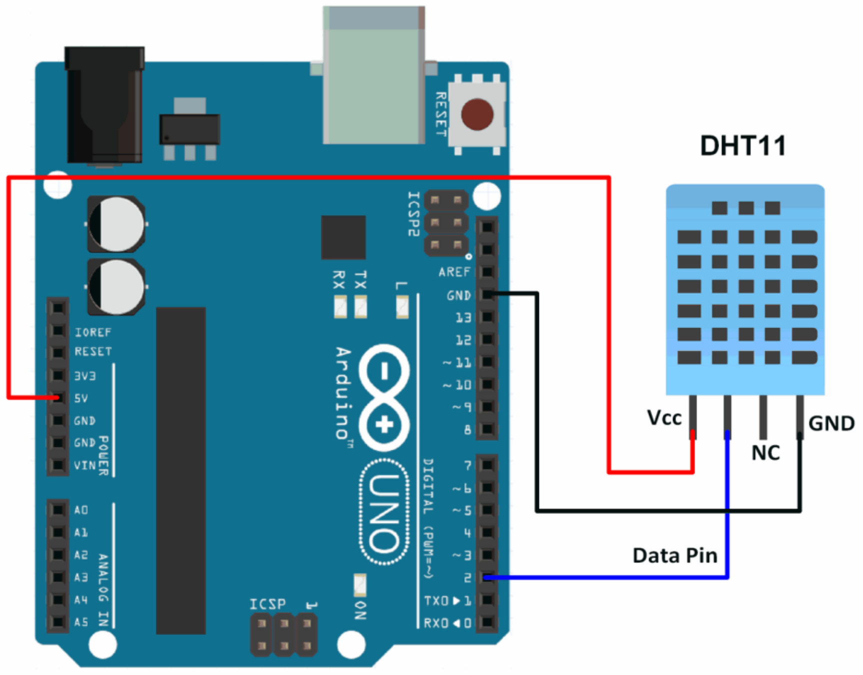
DHT11 Temperature and Relative Humidity Sensor Module

- The DHT11 Temperature and Relative Humidity Sensor Module is a widely used sensor for measuring temperature and humidity in various DIY electronics projects, especially with Arduino. It provides an easy and reliable way to collect environmental data and is commonly used in applications like weather stations, home automation systems, and greenhouse monitoring.



DHT11 Temperature and Relative Humidity Sensor Module

- Go to Sketch > Include Library > Manage Libraries... and search for "DHT".
- Install the DHT Sensor Library by Adafruit.



```
#include "DHT.h"  // Include the DHT library

#define DHTPIN 2  // Pin connected to the DHT sensor
#define DHTTYPE DHT11  // DHT11 sensor

DHT dht(DHTPIN, DHTTYPE);  // Create a DHT object

void setup() {
  Serial.begin(9600);
  dht.begin();  // Initialize the sensor
}

void loop() {
  // Wait a few seconds between measurements
  delay(2000);

  // Read the temperature and humidity
  float humidity = dht.readHumidity();
  float temperature = dht.readTemperature();

  // Check if any reads failed and exit early
  if (isnan(humidity) || isnan(temperature)) {
    Serial.println("Failed to read from DHT sensor!");
    return;
  }

  // Print the results to the Serial Monitor
  Serial.print("Humidity: ");
  Serial.print(humidity);
  Serial.print(" %\t");
  Serial.print("Temperature: ");
  Serial.print(temperature);
  Serial.println(" °C");
}
```