Introduction to Arduino (programming, wiring, and more!)

James Flaten, MN Space Grant Consortium
with Ben Geadelmann, Austin Langford, et al.

University of MN – Twin Cities
Aerospace Engineering and Mechanics Department

Some references we used (there are many):

1. *Beginning Arduino* by Michael McRoberts, 2nd edition

2. http://www.arduinoclassroom.com/index.php/arduino-101

3. http://www.oomlout.com/a/products/ardx/

Goals of this lesson

- Learn what microcontrollers are and things they can do.
- Learn how to use a (solderless) breadboard to wire up sensors and other hardware to an Arduino Uno.
- Learn to open, understand, comment, upload/run, and edit Arduino programs (AKA sketches).
- Learn the difference between input, output, and power pins and also between analog and digital pins.
- Learn to control things like LEDs and write information to the serial monitor (a text window on the computer).
- Learn to read sensor values and log data to an SD card for later retrieval and analysis.
- Learn the basics of using motors
- Learn to expand Arduino functionality with “shields.”
- Put you in a position to implement an Arduino-based computer system for your own projects.

What is a Microcontroller?

- "A microcontroller is a very small computer that has digital electronic devices (peripherals) built into it that helps it control things. These peripherals allow it to sense the world around it and drive the actions of external devices." (Ref. 2)
- It is an "embedded computer system" that continuously repeats software (programming) commands
- Examples: Arduino Uno, Raspberry Pi, etc.

Arduino Uno

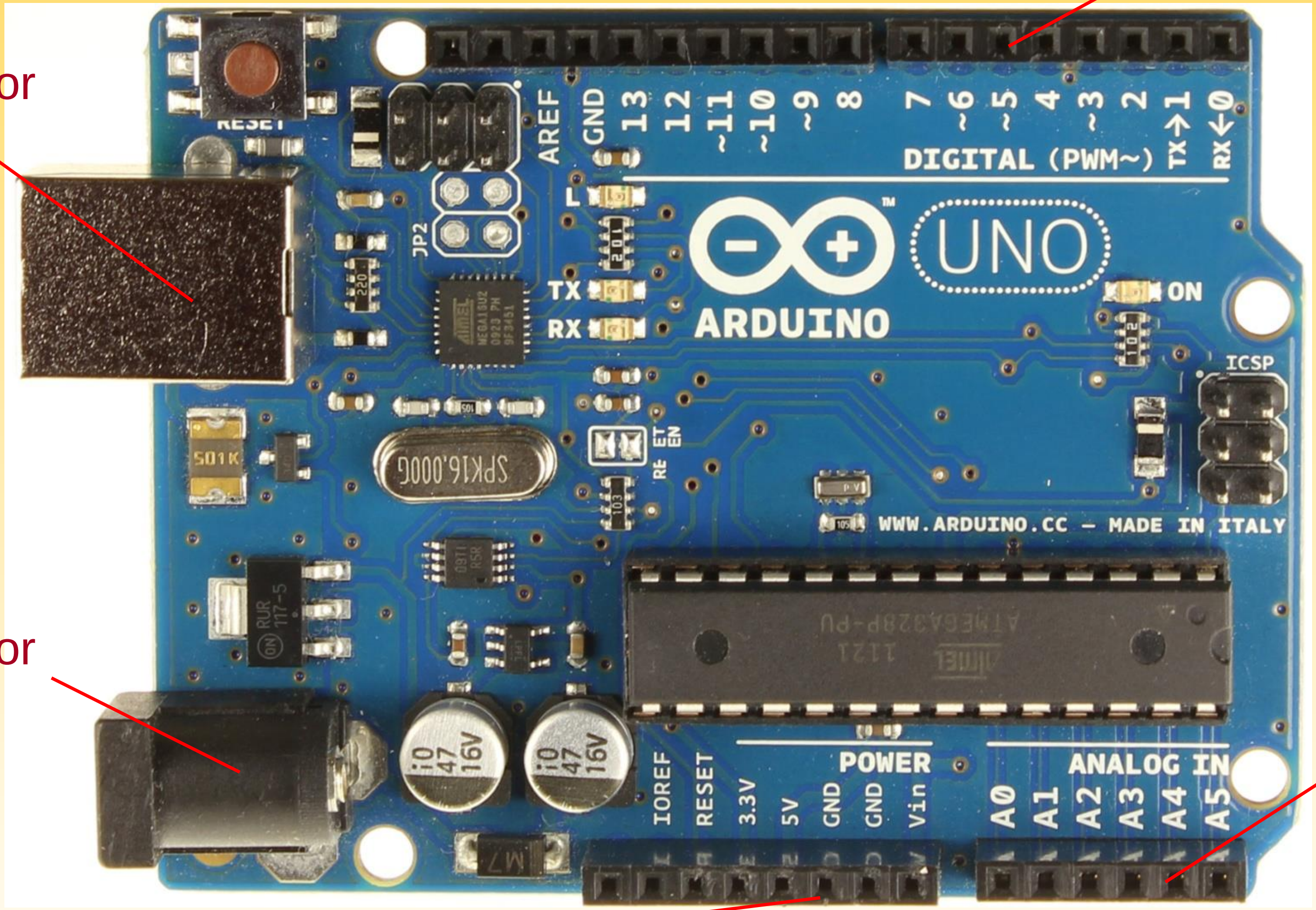
Digital (Input/Output) Pins

USB Connector

Battery Connector

Analog (Input) Pins

Power Pins



Arduino Uno – more details (Ref. 2)

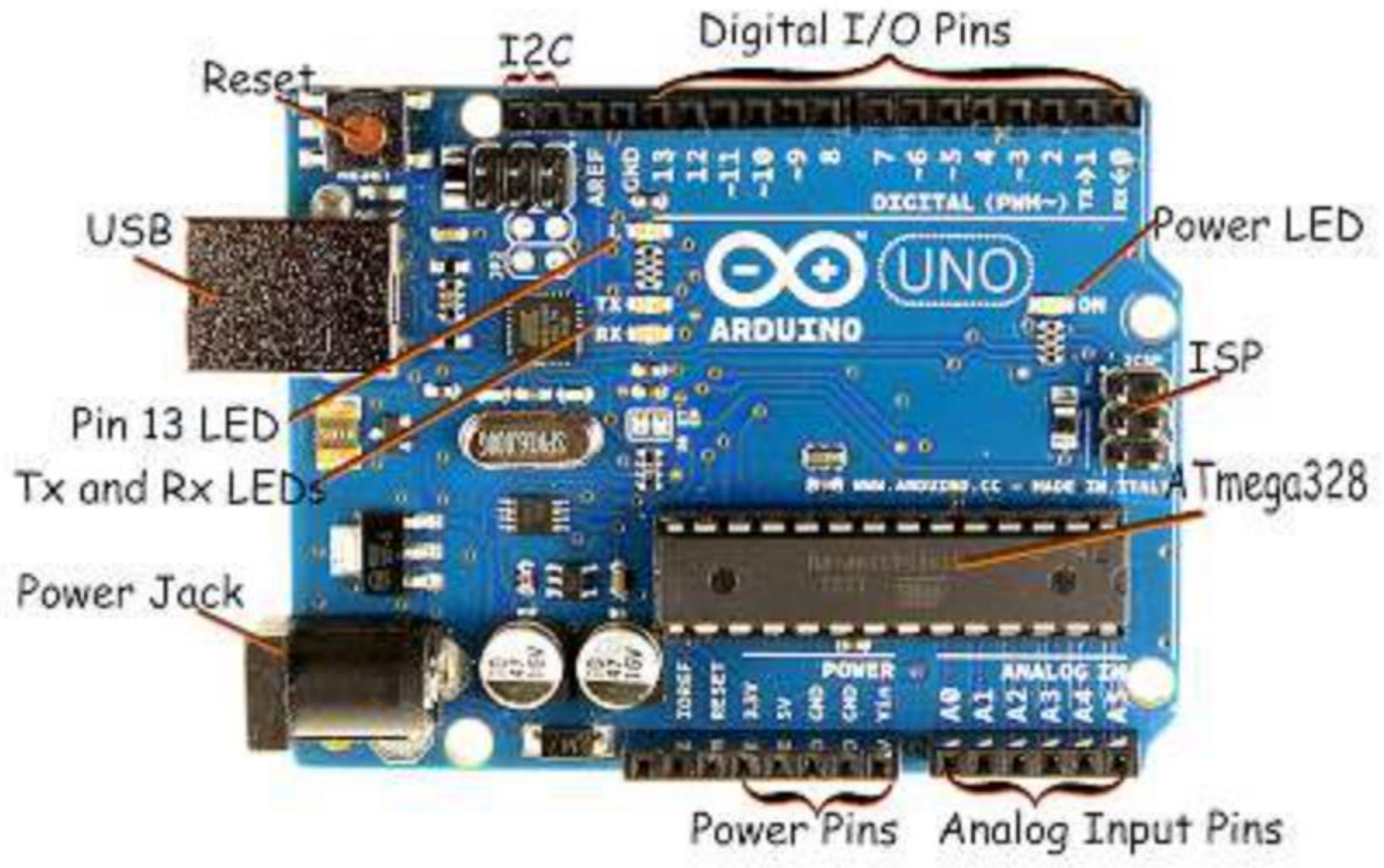
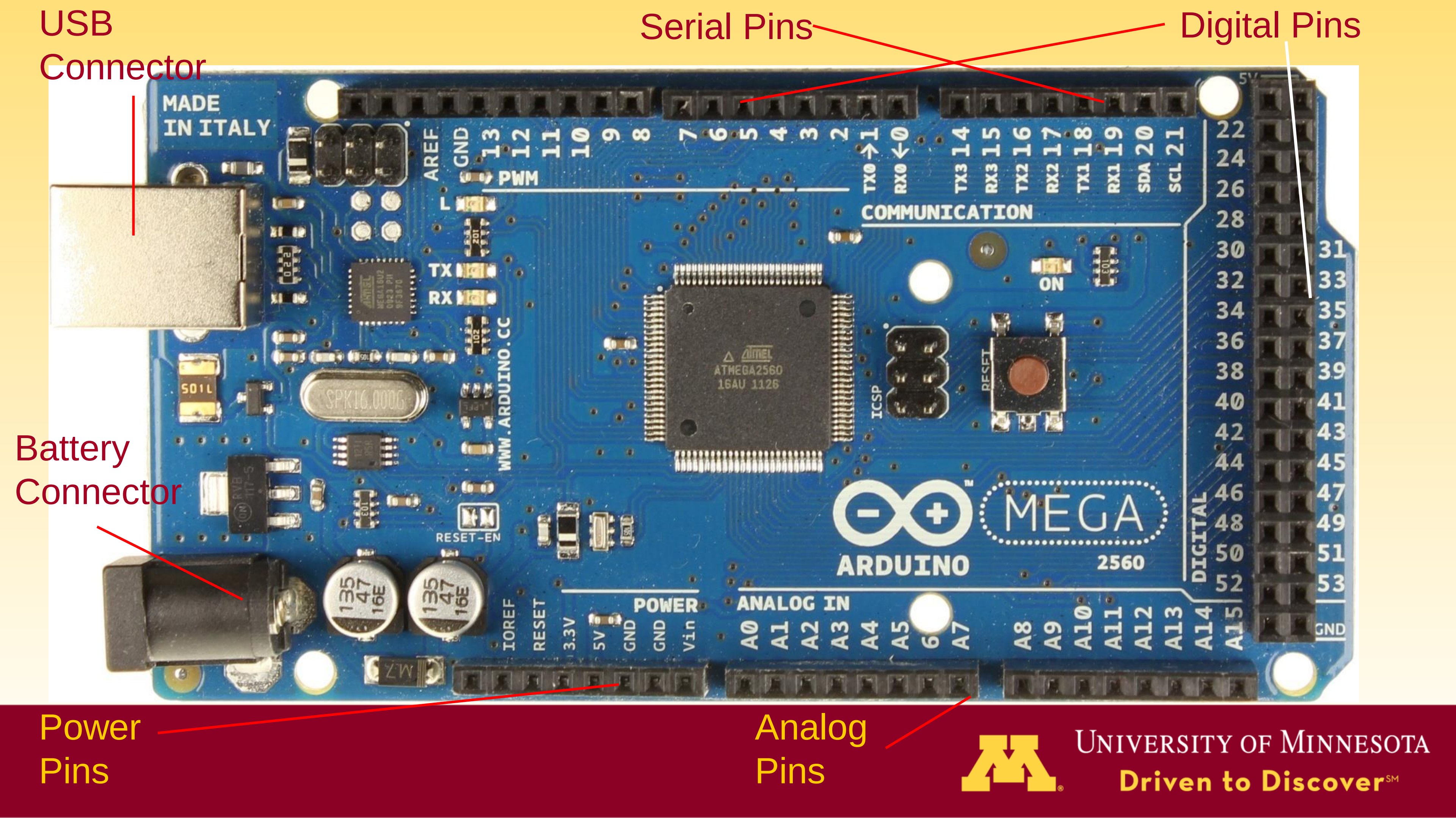


Figure 1: The Arduino Uno R3

Arduino Mega – Larger, more pins (we won't be using this today)



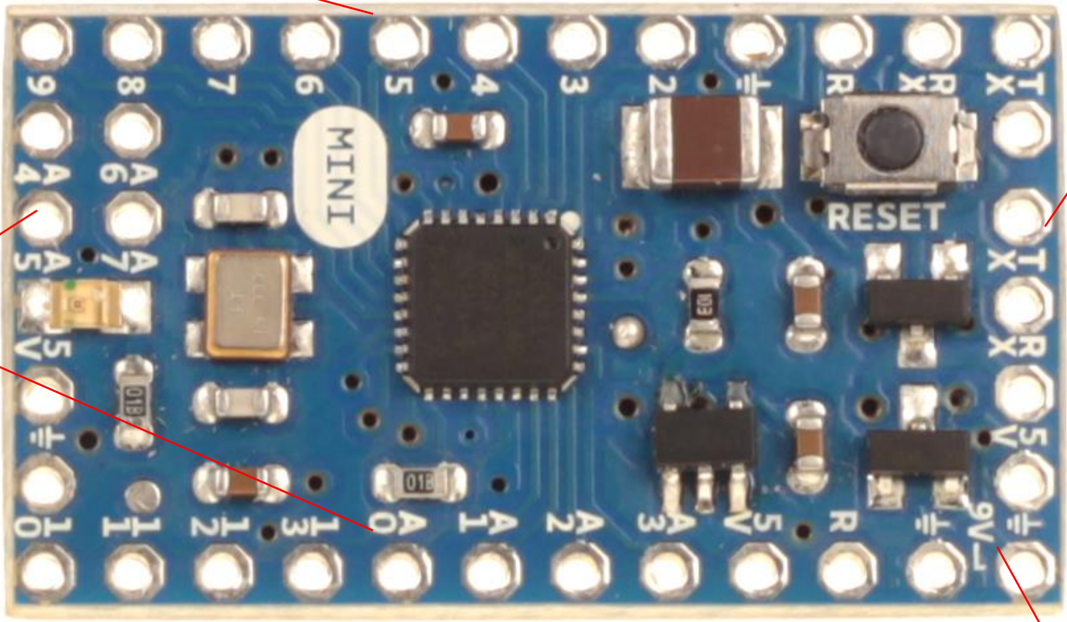
Arduino Mini – Smaller and lighter; (more appropriate for some applications)

Digital Pins

Serial Connectors

Analog Pins

Power Pins



(Solderless) Breadboard (Ref. 2)

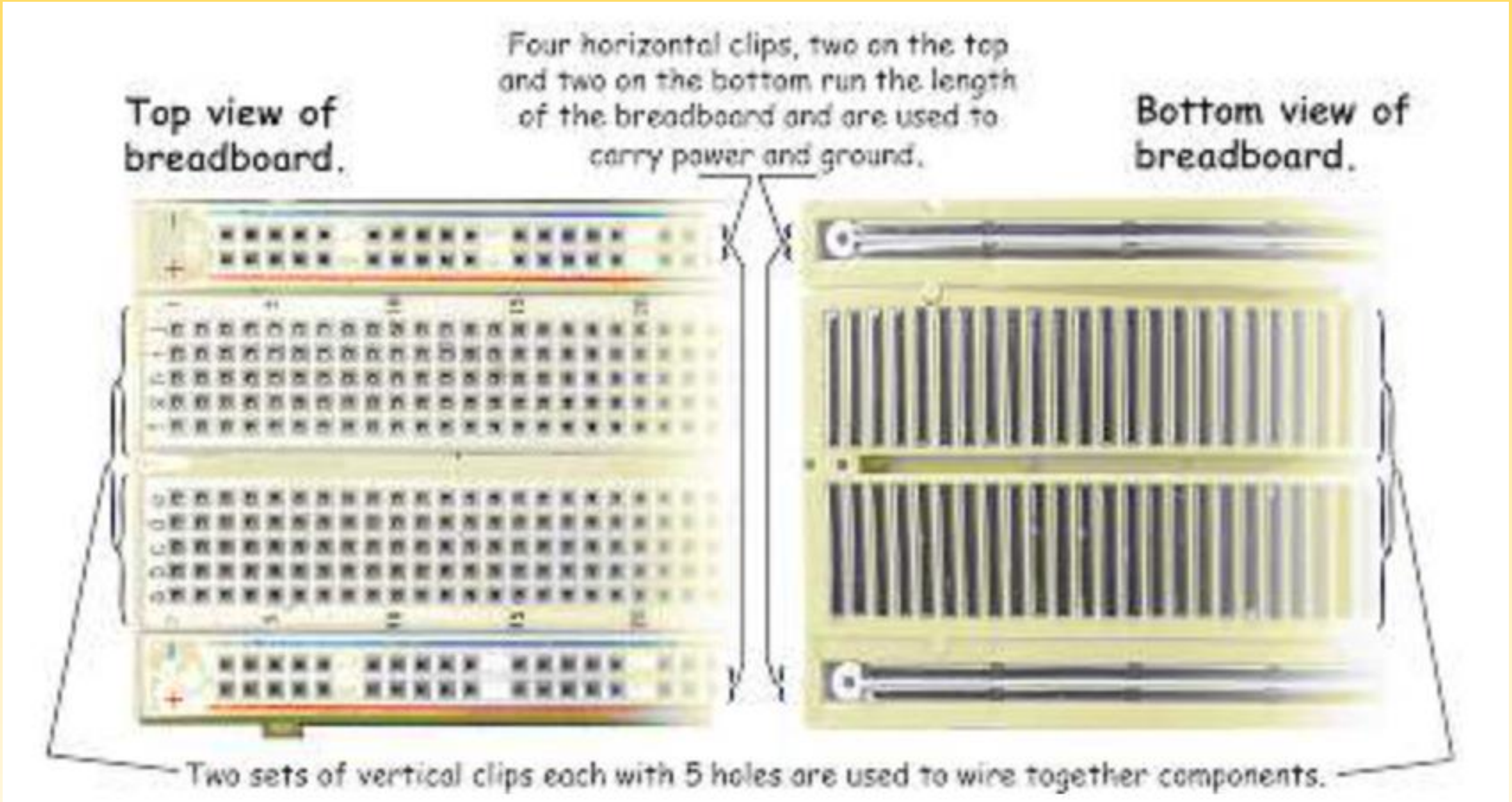

Top view of breadboard.
Four horizontal clips, two on the top and two on the bottom run the length of the breadboard and are used to carry power and ground.
Bottom view of breadboard.
Two sets of vertical clips each with 5 holes are used to wire together components.

Figure 13: Top and bottom of breadboard

Breadboard Innards (Ref. 2)

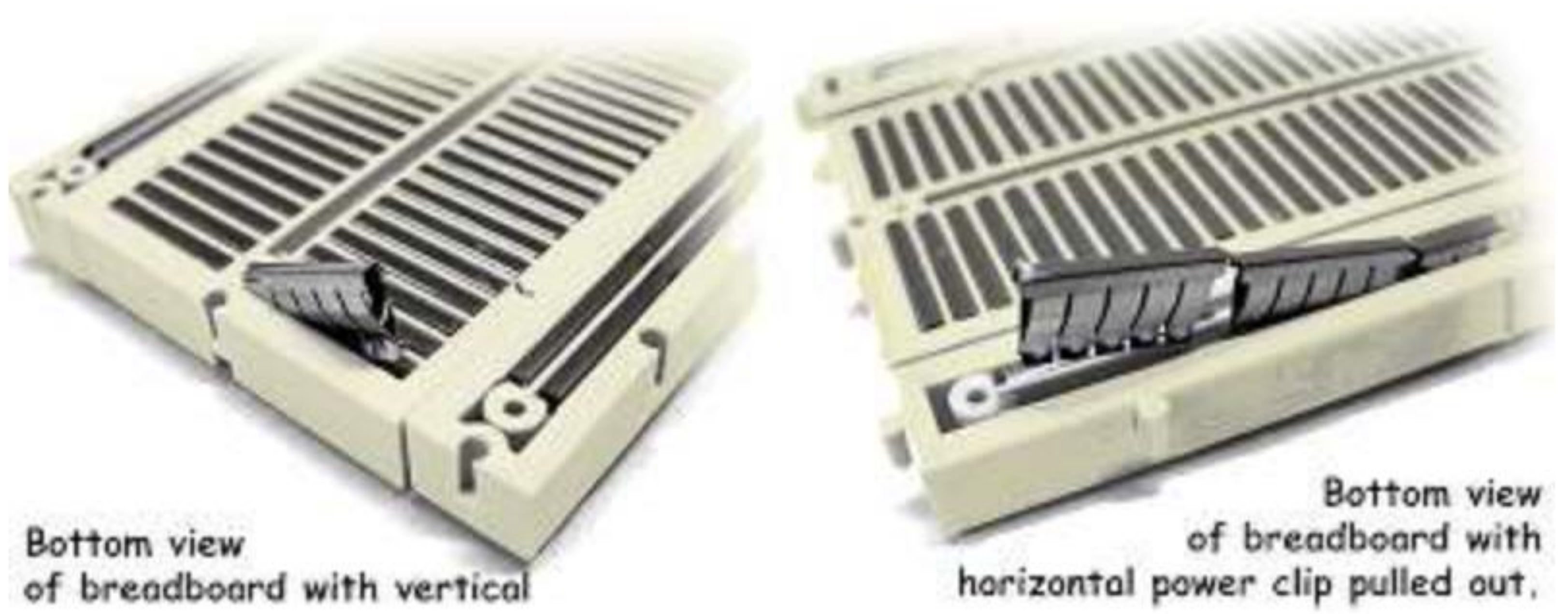

Bottom view
of breadboard with vertical
Bottom view
of breadboard with
horizontal power clip pulled out.

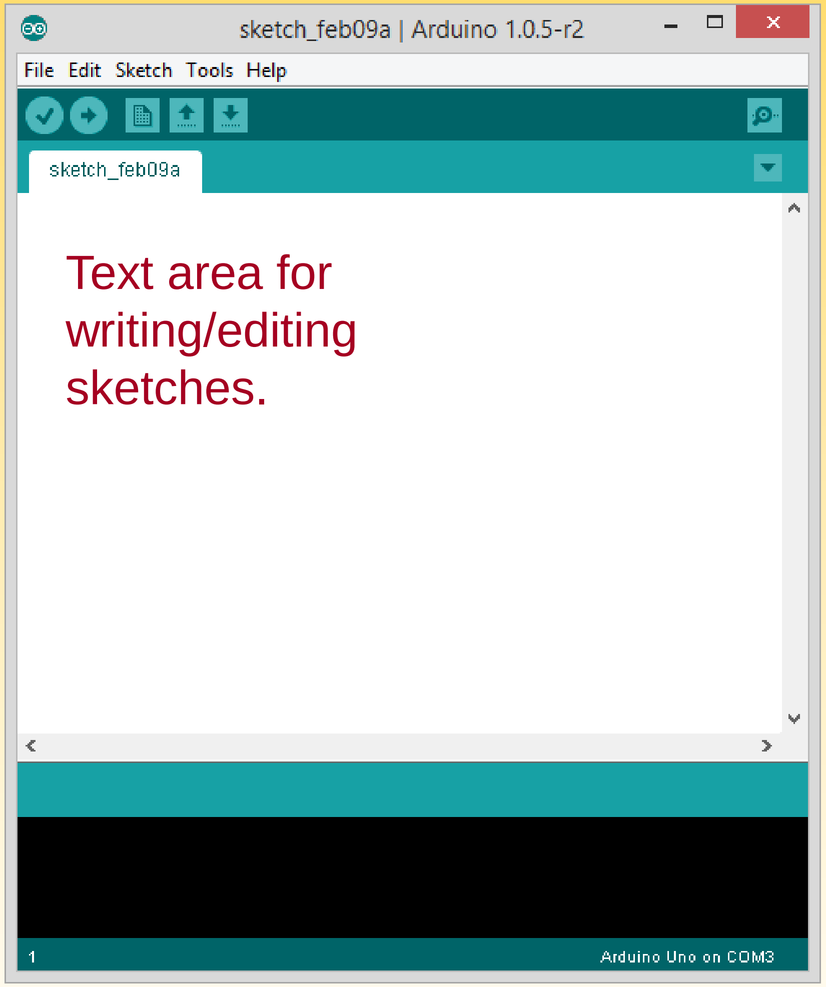
Figure 14: Breadboard bottom with clips pulled out

Insert 22-gauge solid wire jumpers and component leads into breadboard clips to make electrical connections without soldering. Use the edge “rails” for power (5V) and ground.

Introduction to Software

- Arduino microcontrollers are programmed using the Arduino IDE (Integrated Development Environment)
  - Can be downloaded for free from http://arduino.cc/en/Main/Software
- Arduino programs, called "sketches", are written in a programming language similar to C and C++
- Every sketch must have a `setup()` function (executed just once) followed by a `loop()` function (potentially executed many times); add "comments" to code to make it easier to read (technically optional, but actually required (by me))
- Many sensors and other hardware devices come with prewritten software – look on-line for sample code, libraries (of functions), and tutorials

Parts of the IDE main screen



Text area for writing/editing sketches.

Name of current sketch

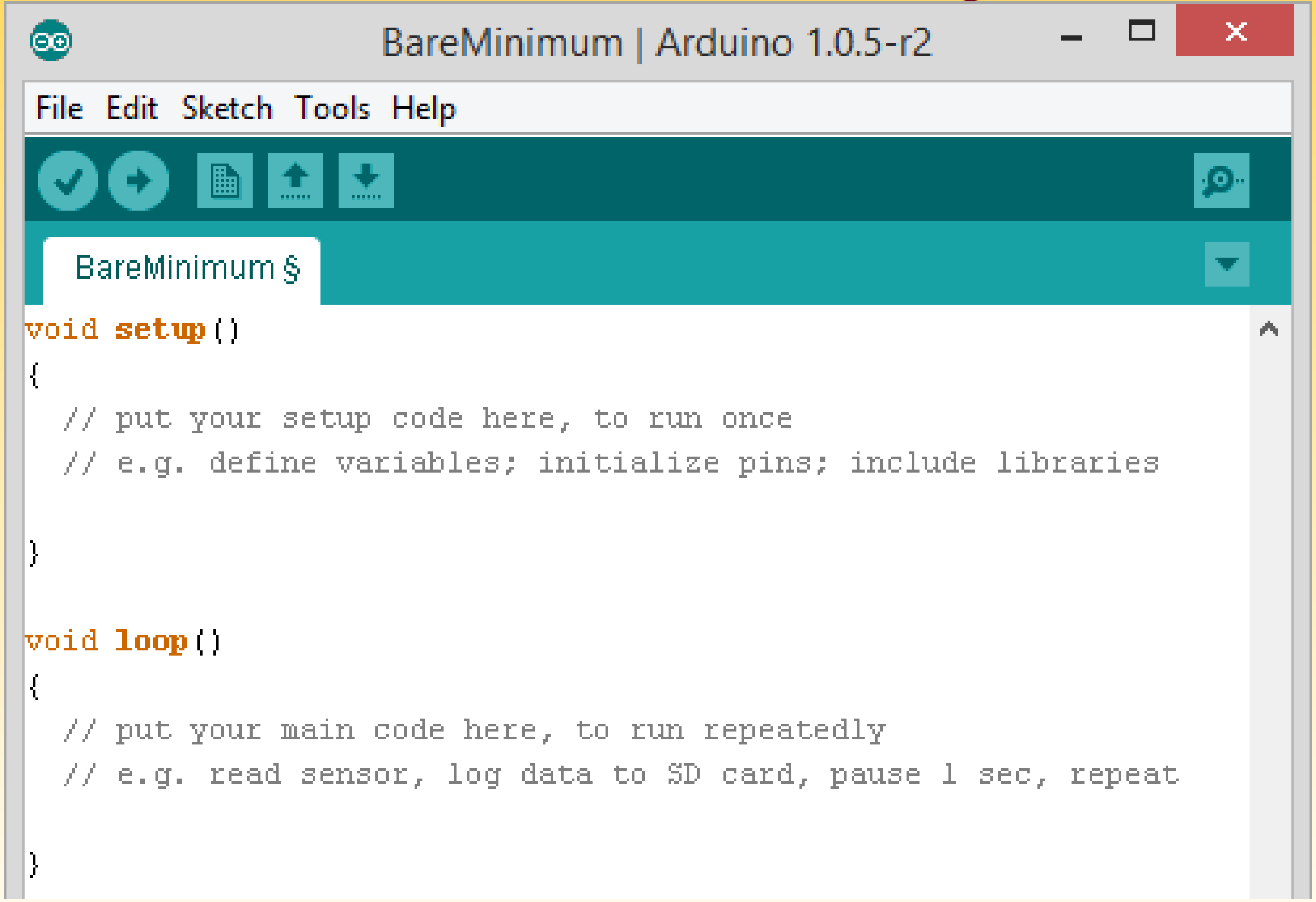
Main menus

Action buttons/icons

- Verify (AKA compile)
- Upload (send to Arduino)
- Start a new sketch
- Open a sketch (from a file)
- Save current sketch (to a file)
- Open Serial Monitor window

Error messages and other feedback show up here.

BareMinimum – sketch organization



```
void setup()
{
  // put your setup code here, to run once
  // e.g. define variables; initialize pins; include libraries

}

void loop()
{
  // put your main code here, to run repeatedly
  // e.g. read sensor, log data to SD card, pause 1 sec, repeat

}
```

Activity 1 – making an on-board and external LED blink

- Place an LED on the breadboard (don't put both legs into a single (connected) column)
- Wire the LED's negative (shorter) lead to a 560 Ohm "safety resistor" then wire the other end of the resistor to ground (GND)
- Wire the LED's positive (longer) lead to digital Pin 13 on the Arduino
- Plug in the Arduino with the USB cable and run the Arduino IDE software on the computer
- Under `Tools:Board` make sure `Arduino Uno` is selected
- Under `Tools:Serial Port` select the correct `COM` port (e.g. `COM3`)
- Under `File:Examples:01Basics` select the `Blink` sketch
- Look at the code. Note `delay(1000)` waits 1000 millisec = 1 second
- Verify (AKA Compile) the code ✓, then Upload it → to the Uno – notice that it runs immediately (and will re-run if you power cycle) – the sketch will stay in the Uno memory until overwritten or erased
- Discuss how the sketch works (in general terms: `setup()`, `loop()`, liberal use of comments); digital HIGH is `5V` and LOW is `0V` (ground)
- This is an example of "control (output) using digital pin 13"

About Motors

- There are several different types
  - – Standard DC motor - input current for full continuous rotation. No special pins or wiring.
  - – Standard servomotor (AKA servos) - Motor capable of limited rotation (generally 180°) in precise degree increments. Uses `Servo` library in Arduino. Have 3+ pins. Controlled by pulse-width modulation ("~" pins)
  - – Continuous rotation servo – can go all the way around continuously. Interprets PWM value as speed & dir.
  - – Stepper Motors - Servo capable of full rotation in small steps. Uses `Stepper` library in Arduino. Have 3+ pins
  - – We are using a standard servo in this lesson.

Activity 2: Using a Servo

- With the Arduino Uno unplugged, wire up the servo according to the chart on the next slide
- Load the sketch `Servo.ino`
- Take a look at the code and note the commands used to drive the servo. The comments should help you understand what each does.
- Be careful – Some servos use a lot of power and may need an external power source.

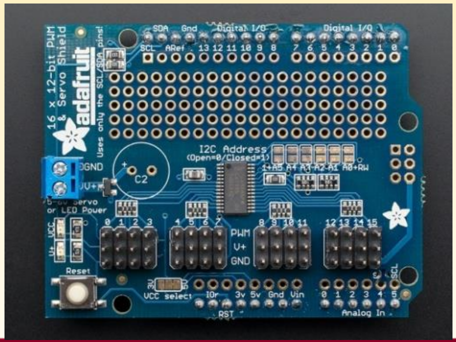
Servo

Pin	Description	Attached To
Brown	Ground	GND
Red	Power (5V)	5V
Orange	Control Lead	D3

Note that the servo has a plug attached to its ribbon cable. This is so that we can more easily extend the cable using plugs and more ribbon cable. It also allows it to plug into specific plugs built into some shields. Several companies make a few different **Motor Shields,** which are shields specifically designed to drive servos, motors, and stepper motors. These usually support 2 to 4 motors, although Adafruit has one that will control 16. They generally have plugs built into the shield for the motors and often drive them through some sort of a serial connection (I2C or SPI is common).

Right: Adafruit Motor Shield

http://www.adafruit.com/products/1411



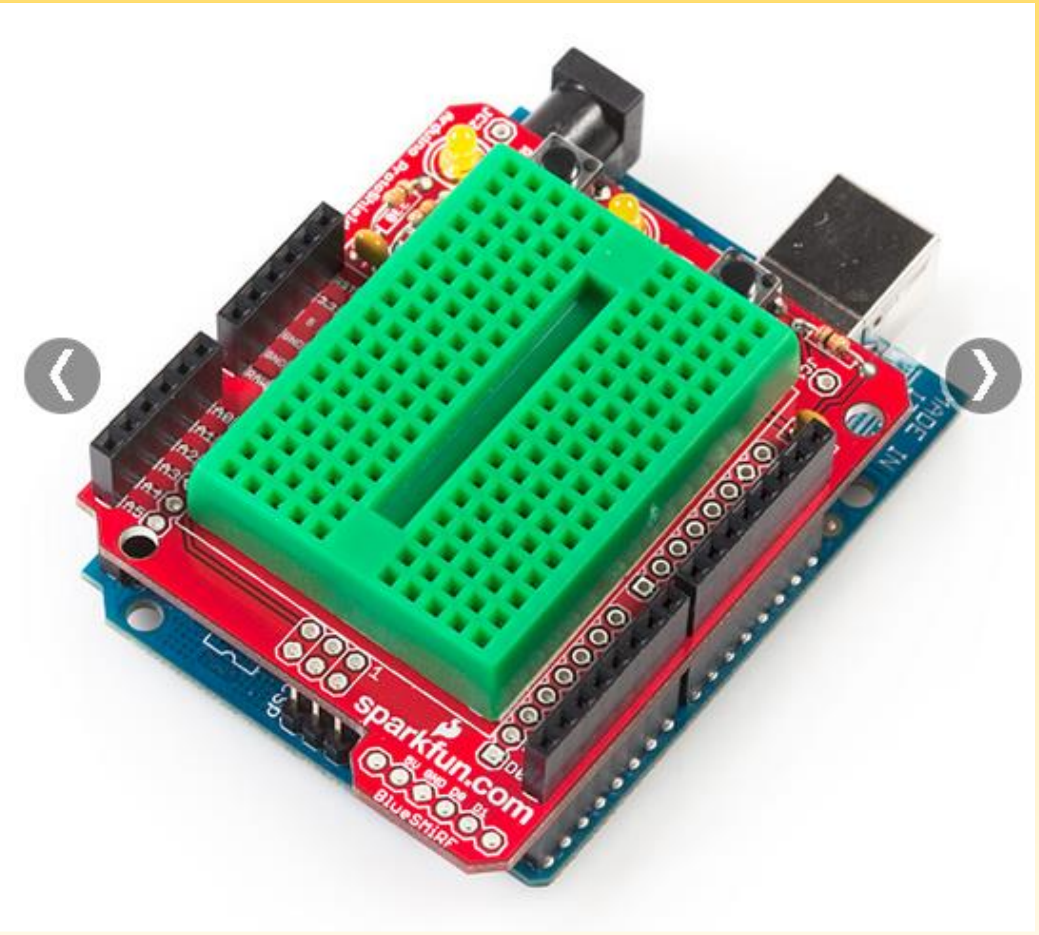
To Sweep or Not to Sweep

- In many applications it might not be necessary to use a “sweep” sketch, where the servo sweeps between angles in incremental small steps.
- Another way may be to set the servo to just “open” (go to one angle) and “close” (go to another angle). This might be easier to code than “sweep.”

Notes about Libraries:

- Libraries take large amounts of code and refine it into a few simple commands we can use. Ex: `datalog.print` will print something to a file.
- This actually takes a lot of code, but by putting it in a library we can ignore the basic code and focus on the useful commands and functions
- Usually ends in a .h (header file)
- Used in our programs by the `#include` command: `#include "SD.h"`
- Almost everything that is not analog (digital sensors, servos, etc) use a library of some sorts (Many use the same library - i.e. `Wire`)
- You can install more libraries into your IDE. Look under `Sketch:Import Library` to see which ones you have.
- Ex. `SD.h, SoftwareSerial.h, Wire.h, OneWire.h, RTClib.h`

Proto shield & “tiny” breadboard



- We are not using the proto shield today, but they can be seen in your kit.
- Notice that this “tiny” solderless breadboard (ours are white) has no power rails and very limited real-estate. Use it wisely!

A few words on sensors

- The rest of the lesson will focus on how to use various sensors to make measurements and how to “log” (record) those values to an SD card.
- Sensors can be broken into two major categories:
- Analog sensors
  - – Based on changing the voltage of the sensor
  - – Can only use analog inputs (Arduino pins A0, A1, etc)
  - – Generally easier to program and use
  - – Not as accurate & easily interfered with by other electronics (noise in voltage)
  - – Readings can be influenced by cable length – **keep connection cables short!**
- Digital Sensors:
  - – Usually use digital pins
  - – Can use more advanced forms of communication to let multiple sensors share the same pins (sometimes even using analog pins)
  - – Generally more difficult to program and wire up; often need libraries and significantly more code for each type of sensor used
  - – Most advanced sensors (GPS, IMU, etc) are digital – this is the best way to pass more data between sensor and microcontroller quickly & efficiently

About Digital Sensors:

- Digital sensors used with Arduinos are powered using either 3.3 volts or 5 volts – both are available on the Arduino
- Always check to be sure you are providing the correct voltage. If you send a 3.3V sensor 5V it is very easy to blow it out.
- Watching for pin conflicts and voltage discrepancies is par for the course when using microcontrollers. If you aren’t careful you can get bad data, fry components, or possibly even damage the microcontroller itself.
- Aside: Also watch out for components that look similar (e.g. the analog temp sensor and the digital temp sensor). And remember that most components require a specific orientations (i.e. it matters which leg gets attached to which pin); resistors and some types of capacitors are an exception and can get plugged in either way

Activity 3 – Wiring and logging sensor data

Shields and sensors involved:

- Micro-SD or full-size-SD card shield
- Data LED indicator (tells whether the SD card is logging data (steady flash) or not (multi-flash (indicating an error))
- Analog temperature sensor
- Digital 3-axis magnetometer sensor

Micro-SD or SD Shield and Data LED Indicator

- Plug the micro-SD or SD shield directly onto an unpowered Arduino Uno. All the legs must go straight into the header – don’t bend any.
- Wire an LED using the tiny breadboard:

 - Positive (long) leg wired to a safety resistor. Other end of resistor connects to digital pin 5.

 - Negative (short) leg wired to ground (GND).

(Aside – This has the resistor on the opposite leg from last time. It is important to get the orientation of the LED itself right, but not the resistor and it doesn’t matter which leg the safety resistor goes on as long as it is in series with the LED to limit current flow through the LED.

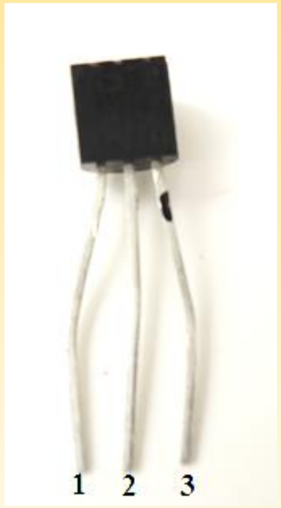
Analog Temperature Sensor

- With the Arduino Uno unpowered, wire the `TMP 36` sensor using the table below

Pin	Function	Attached To
1	`5V` (Power)	`5V` pin
2	Analog Output	Pin `A2`
3	Ground	`GND` pin

- This 3-pin sensor has `TMP` written on it. This is called a `TO-92` package.
- Make sure the flat side of the sensor has “TMP” written on it!
- **Check the orientation!**

Note: The photo shows flat side facing you.



Digital 3-Axis Magnetometer

- With the Arduino unplugged, add the Digital 3-Axis Magnetometer `MAG 3110` to your breadboard using the table below:

Pin	Function	Attached To
VCC	Power Input (3.3V)	3.3V
GND	Ground	GND
SDA	Serial Data Line (I2C Bus)	A4
SCL	Serial Clock Line (I2C Bus)	A5
INT	Interrupt Pin	Not Connected

- Aside: a magnetometer can be combined with an accelerometer (and maybe gyros) to make an IMU (Inertial Measurement Unit)
  - Not perfect - a magnetometer can be interfered with by magnets, metals, and other electronic devices (good IMUs can account for such interference, to some degree)

Logging the Data

- Open the sketch `Temp_Mag_Log.ino`
- As you can see in the code setup, data logging can be a bit complex.
  - Most of the time you can directly copy the code from the setup loop used for the SD card
- A few details to keep in mind:
  - Must do all reading/writing of data in the main loop.
  - We included a special piece of code to write a new file each time the Arduino is powered. Without this, it would overwrite our old data
  - Pins can change depending on the SD card reader you use. The Sparkfun Micro-SD Shield uses D8 and D10-13. The Adafruit full-size-SD-card shield uses D10, not D8, for “chip select.”

Datalogging (To SD Card)

- Load the sketch `Temp_Mag_Log.ino` onto the Arduino
- Open the Serial Monitor window

  - Important note! The serial monitor normally operates at `9600` baud but this sketch runs it at `115200` baud so you will need to open the serial monitor and change the baud rate using the drop-down menu at the bottom right hand corner.

- What is shown in the serial window? How can you verify the data recorded is being saved onto the SD card?
- After you have run the Arduino for a bit, unplug it, disconnect the SD card, and read the SD card on your computer. How does the data look?

What is always useful…

- How can you analyze raw data off the SD card? You may not remember when/where it was taken so it is useful to add a “time stamp” and/or a “GPS (location) stamp” to data as it is saved.
- A RTC (real time clock) comes in certain SD shields or breakout boards – use to put a time stamp on data.
- GPS can also come as a separate shield or a breakout board on its own – use to put a location stamp on data.
- Again, when logging sensor data, it’s useful to tag the data to help remember exactly where it came from.

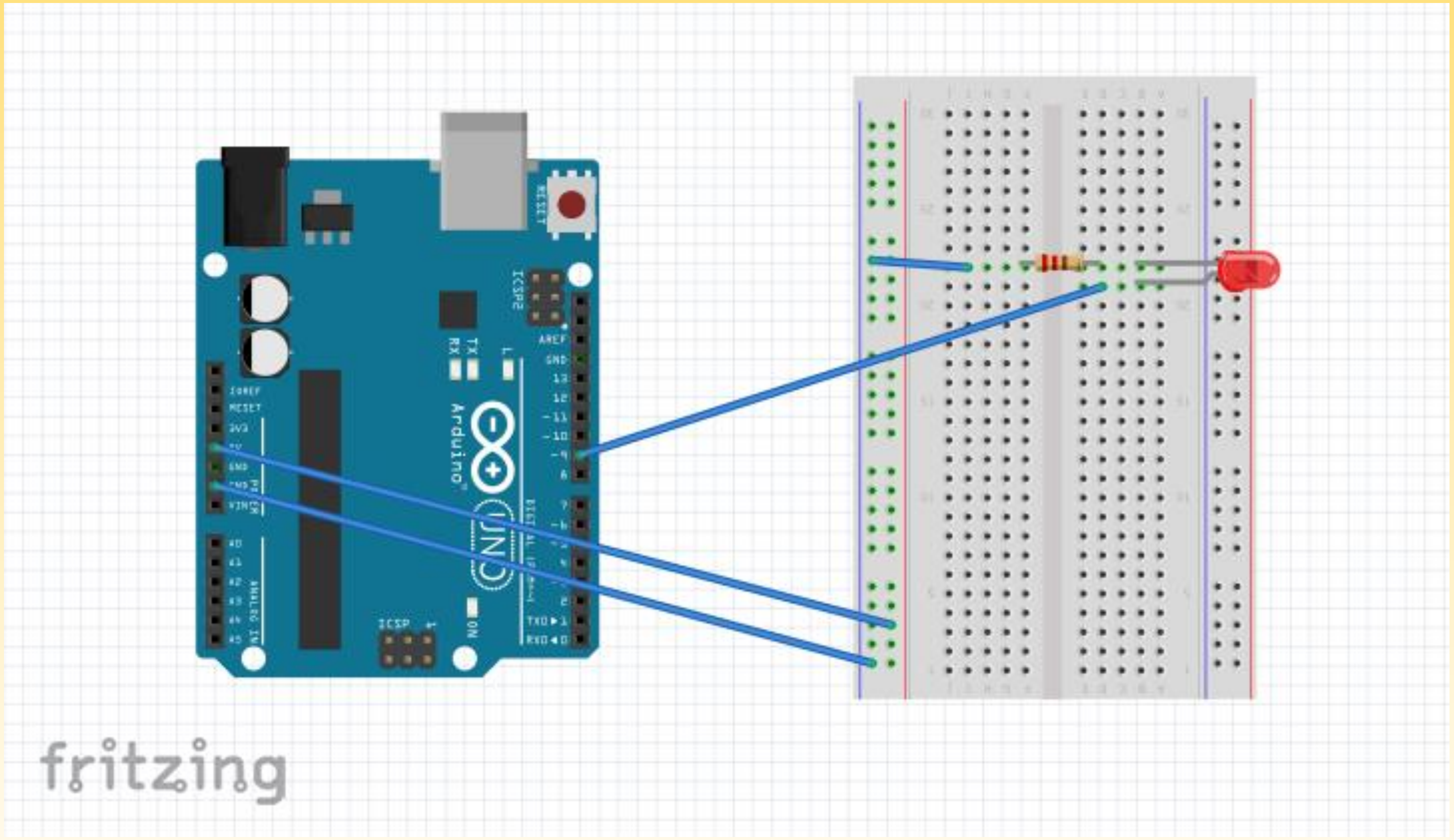
Notes on writing to the “Serial Monitor”

- Before `setup`, the variable being recorded in a serial monitor or SD card is declared
- In `setup`, the serial communications need to be started, with the baud rate (char per second) given in parentheses like this: `Serial.begin(9600).` **When you open the serial monitor window make sure the baud rate (in the lower right hand corner) matches the baud rate called for in the sketch – pull-down to change it if need be.**
- At the end of the loop, the command `Serial.print();` is used to print the data to the serial monitor.
- Every time the serial monitor is reopened, it is “refreshed” and starts data over (e.g. if you were counting, it would always start at “1” when you open it)

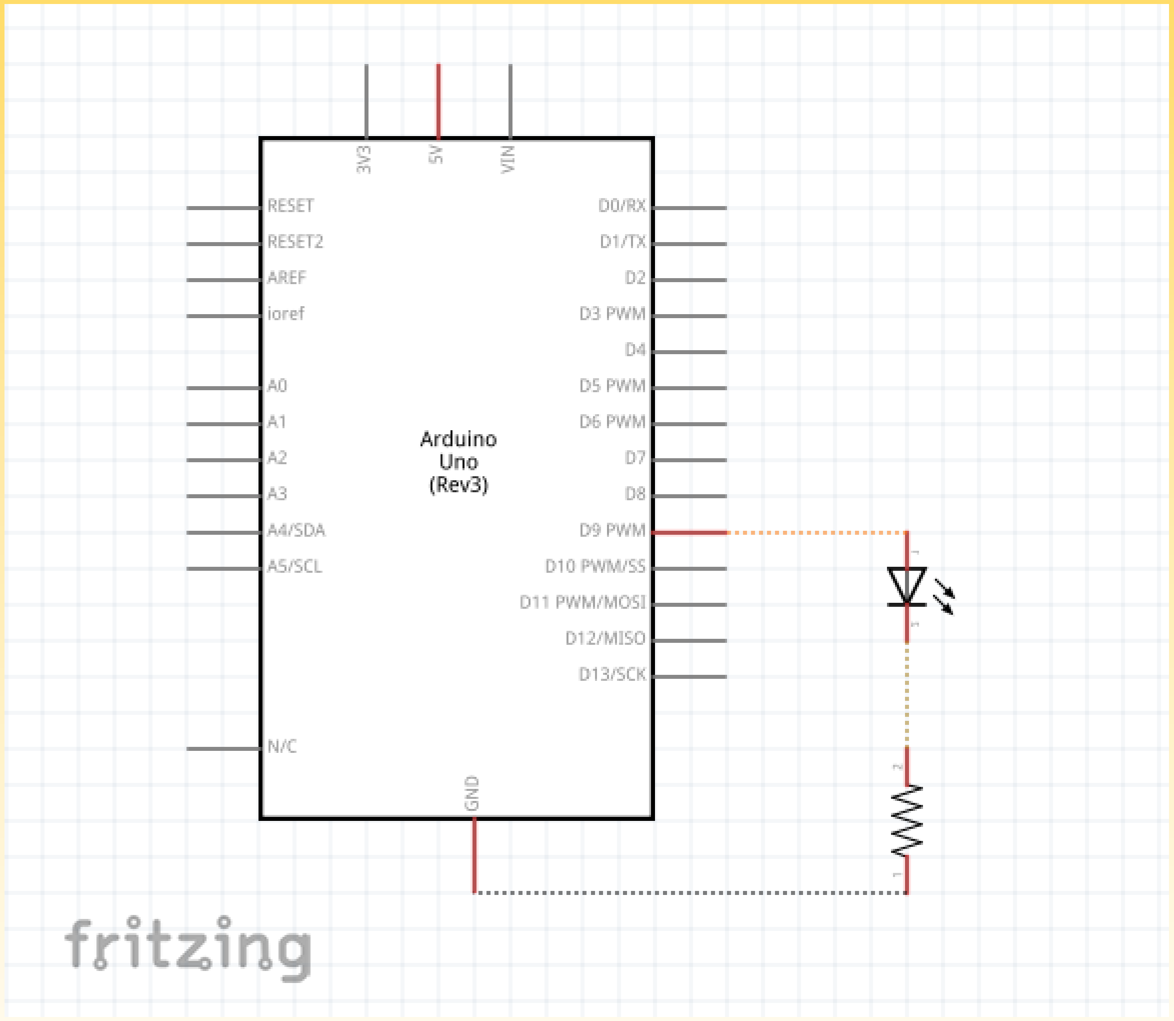
Notes

- Flashing LEDs and/or writing information to the serial monitor (window) lets the sketch tell observers what it is up to – the latter only works when you are attached to a computer with a screen, so the former method might be more useful for field/flight implementation
- Digital pins can be set to `OUTPUT` mode, after which you can send them digital values (just `HIGH` or `LOW`) with `digitalWrite` or else more-continuous analog values (from `0` to `255`) with `analogWrite` – the latter still sets them just `HIGH` or `LOW`, but does so only some fraction of the time which makes some devices think the output voltage is somewhere between `HIGH` (`5` Volts) and `LOW` (`0` Volts)
- “fritzing” software (free download) for drawing circuits: “breadboard view” (fairly realistic) vs “circuit diagram” view (AKA “schematic”)

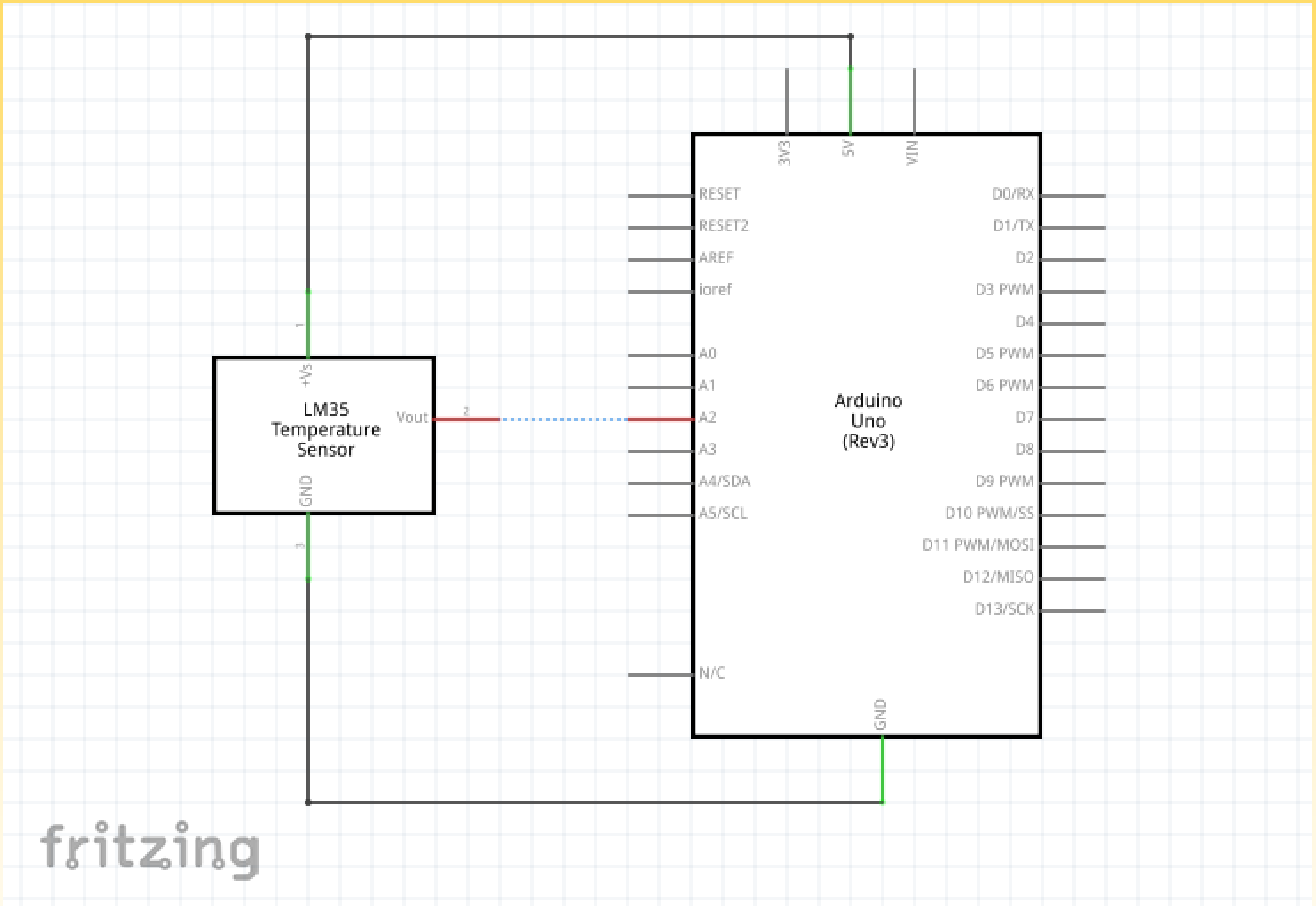
A “breadboard view” looks fairly realistic



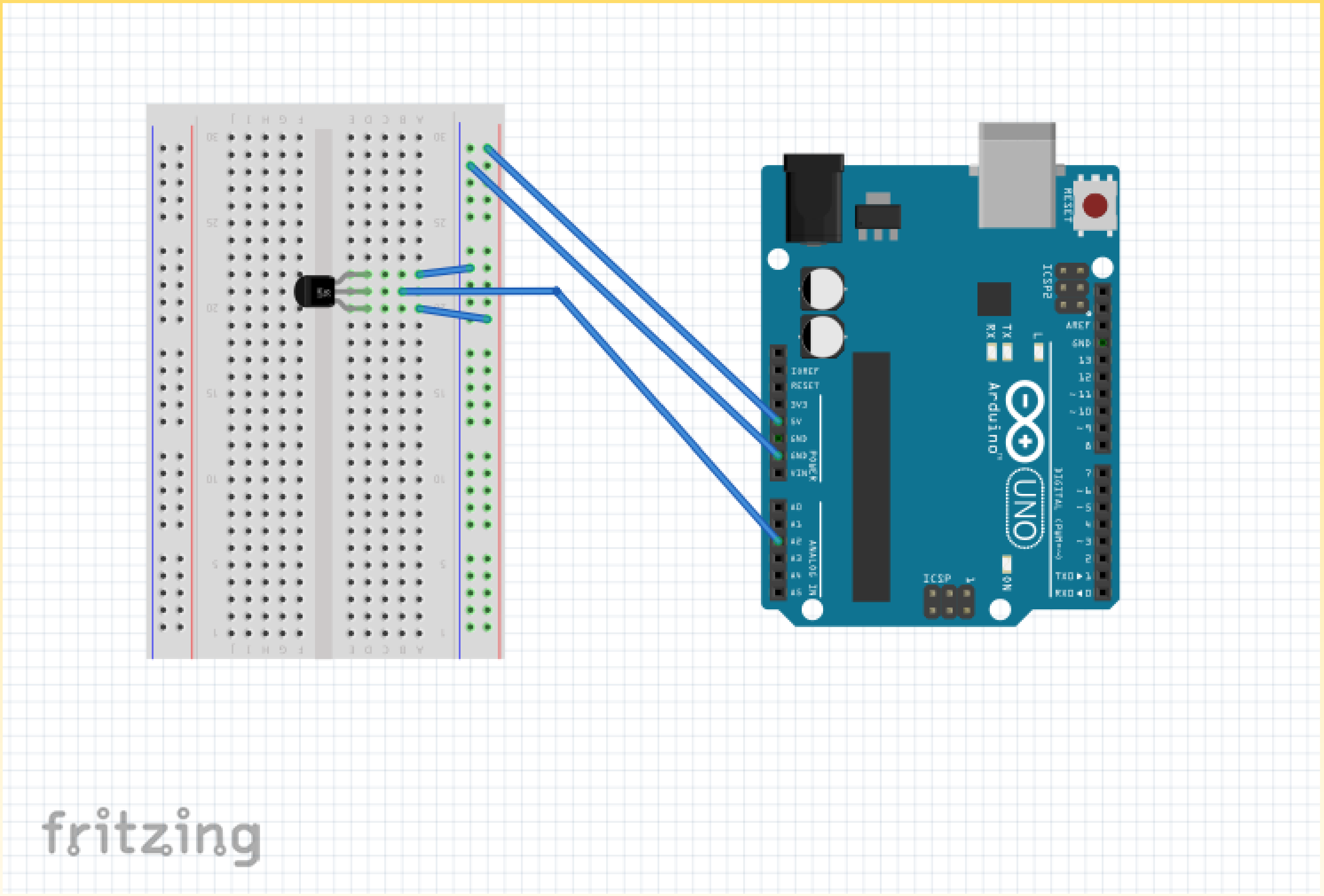
A circuit diagram (AKA schematic)



Typical wiring of a powered analog sensor



Note the utility of having both 0V (AKA GND) and 5V “rails” (but “tiny” breadboards don’t have rails)



Next Steps

- This has only been a brief intro, things can get a lot more complicated!
- Look online for help with advanced projects, weird things (pin conflicts, voltage issues, software bugs, etc) can happen pretty easily
- Also keep in mind how much power you are drawing- you can drain individual 9V batteries quickly- use more in parallel to provide enough current for a long run time
- Research and think through projects before building. A good plan can solve many problems before they occur

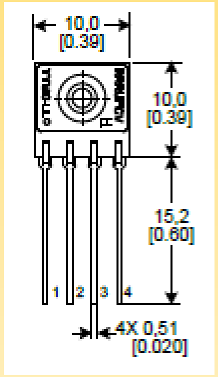
Good further resources

- https://learn.adafruit.com
  - Adafruit makes many shields and sensors, and they have tutorials for almost everything they carry
- http://www.arduinoclassroom.com/index.php/arduino-101
  - Arduino Classroom is currently doing an intro series on Arduinos. Check it for updates and more topics in the future
- http://playground.arduino.cc/
  - Arduino playground is the wiki run by the Arduino company for its products. There is a lot of helpful information on almost everything imaginable here.

Activity 4 – analog (air-)pressure sensor

- With the Arduino Uno unpowered, wire the 0-15 psi pressure sensor using the table below (you can leave the TMP 36 sensor in place)
- Load the sketch `Analog_Pressure`; study code; run; test; discuss

Pin	Function	Attached To
1	Nothing	Not connected
2	`5V` (Power)	`5V` pin
3	Analog Output	Pin `A0`
4	Ground	`GND` pin



- This 4-pin pressure sensor is the Honeywell `SSCSANN015PAAA5` and is called a `SIP-AN` package.

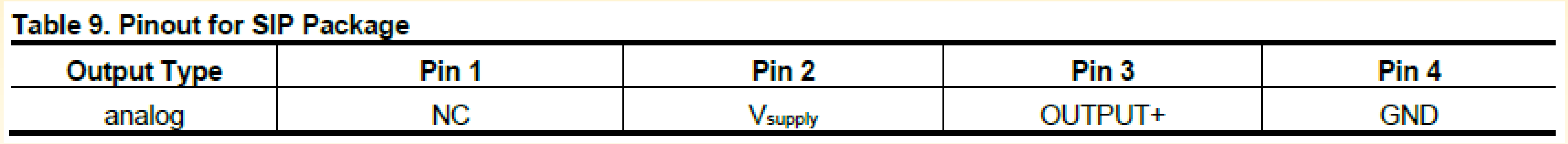


Table 9. Pinout for SIP Package

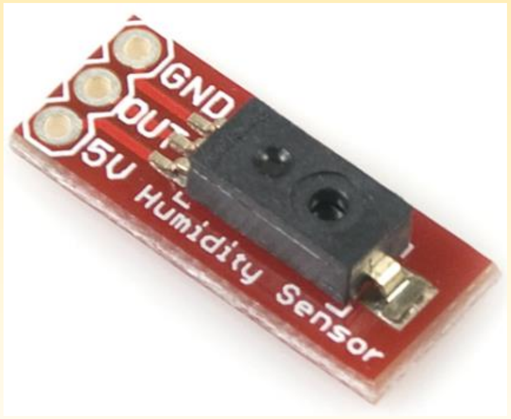
Output Type	Pin 1	Pin 2	Pin 3	Pin 4
analog	NC	V_{supply}	OUTPUT+	GND

Activity 5 – analog relative humidity sensor

- With the Arduino Uno unpowered, wire the `HIH4030` relative humidity sensor using the table below (leave other sensors in place)
- Load the sketch `Analog_Humidity`; study code; run; test; discuss

Pin	Function	Attached To
`GND`	Ground	GND pin
`OUT`	Analog Data Output	Pin `A1`
`5V`	`5V` (Power)	`5V` pin

- This 3-pin relative humidity sensor is pre-mounted on a "breakout board"
- Notice that we soldered on a 3-pin male header so the breakout board can plug into a (solderless) breadboard



Activity 6 – using a 3-axis accelerometer

- The digital 3-axis accelerometer (breakout board ADXL345) is powered with **3.3 volts, not 5 volts**, and makes use of analog pins **D10-D13** for communication. All 3 outputs (the x, y, and z values) are sent through the serial data line, D11, clocked (i.e. timed) by line D13.
- The digital accelerometer makes use of the SPI library which should already be in your Arduino program.
- Plug in the Arduino Uno and load `Digital_Accelerometer`
- Study the code (it might be long!); run it (open serial monitor); test all 3 axes, discuss

Digital 3-Axis Accelerometer (breakout board ADXL345)

Pin	Description	Attached To
GND	Ground	GND
VCC	Power input (3.3V)	3.3V
CS	Chip Select	D10
INT 1	Interrupt 1 Output	N/A
INT 2	Interrupt 2 Output	N/A
SDO	Serial Data Output (SC0)	D12
SDA	Serial Data (SDA)	D11
SCL	Serial Communications Clock (SCL)	D13

Activity 7 – digital pressure/temp sensor

- With the Arduino Uno unplugged, add the pressure (altitude) / temperature (MPL3115A2) breakout board to your tiny breadboard and wire it as follows. This is a 3.3 volt I^{2}C device.

Pin	Description	Attached To
INT2	Pressure interrupt 2	Not connected
INT1	Pressure interrupt 1	Not connected
SDA	Serial Data	A4
SCL	Serial Communications (Clock)	A5
VCC	Power input (3.3V)	3.3V
GND	Ground	GND

- Look for pin conflicts. Hopefully there won't be any (but always look).
- Plug in the Arduino Uno and Altitude_Pressure_plusTemp_Sensor
- Study the code (it might be long!); run it (open serial monitor); test, discuss

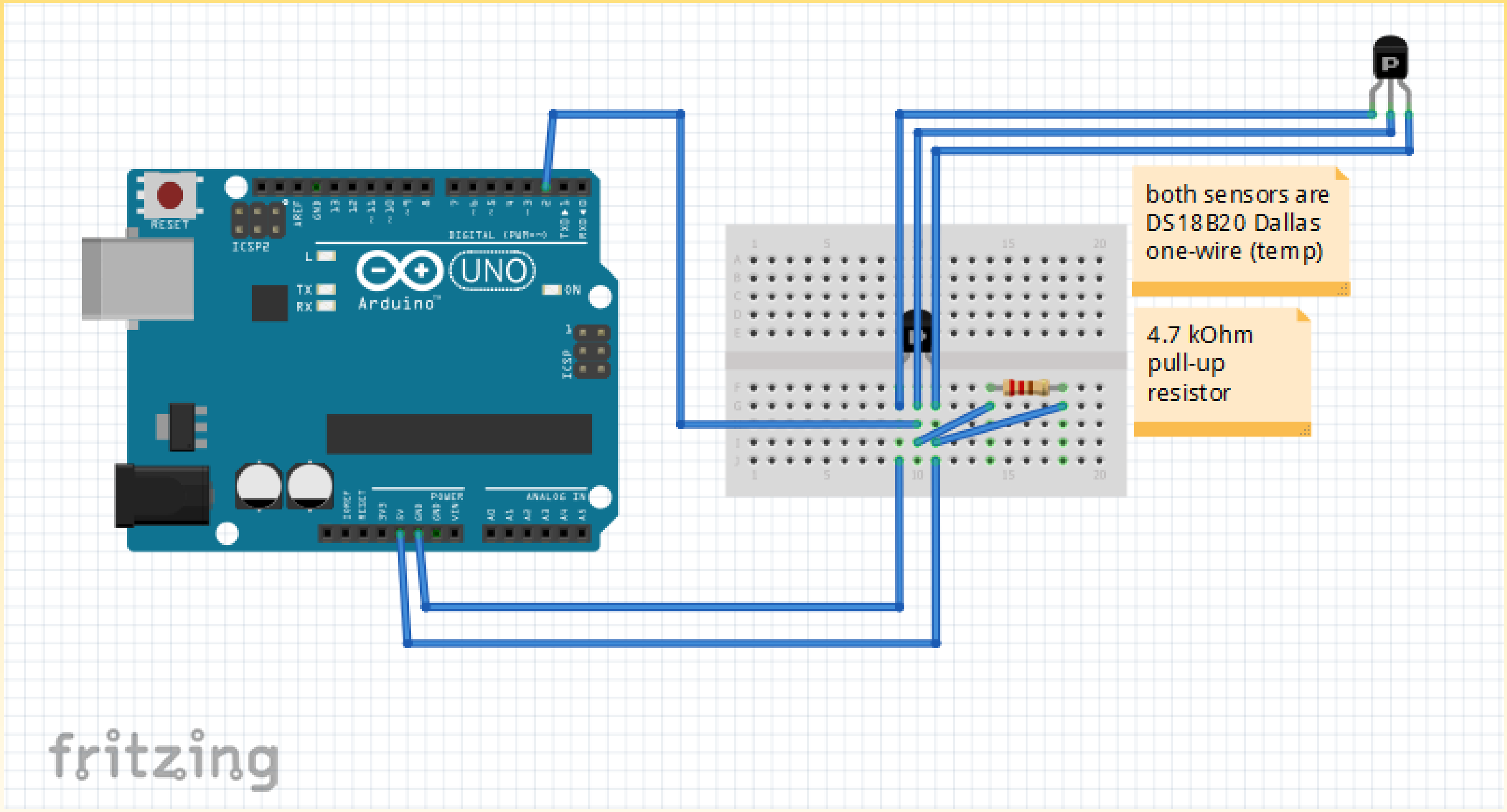
Activity 8 – 2 digital temp sensors, SD card

- With the Arduino Uno unplugged, add one "Dallas one-wire" digital temperature sensor on the breadboard and a second one at the end of a 3-wire cable (if not provided then just make your own with jumper wires), as shown on the following slide. Notice that both sensors need ground and power (5 volts) and that they are polled through a single digital pin (D2). Also notice that a 4.7 kΩ "pull-up" resistor is required between pin D2 and the 5 volt line so that it stays HIGH when not in use.
- Place `OneWire` and `DallasTemperature` into your Arduino library. This activity also uses `Wire` and `SD` (already in your library).
- Put an SD card into your SD card shield. Note – if you are using a SparkFun shield with a microSD card reader it may not have a RTC (the code will still run – it just won't have real time clock data) and be sure the code uses the SparkFun value of "8" rather than the Adafruit value of "10" for the "chip select" variable.
- Plug in the Arduino Uno and load the sketch `SD_RTC_2_Digital_Temp` which will determine the sensor names, poll them, write the results to both the screen and to the SD card (after opening a file), then repeat.
- Study the code (it might be long!); run it (open the serial monitor – confirm baud rate); test it (including touching the sensors one at a time to change their temperature - also try removing the SD card and reading the data directly on your computer). Discuss the utility of saving data this way; discuss the utility of having "off-board" sensors. Note: cable length can impact analog sensor data readings so always minimize cable lengths and use digital sensors when off-board (if possible).

Wiring for 2 digital temperature sensors

(Not shown – the tiny breadboard may actually be mounted on a protoshield on top of the Arduino Uno; optional: if the shield doesn't have a RTC you could add your own (perhaps put that on the upper left side of the breadboard).)

Use this wiring color convention: red for power, black for ground, green for data.



both sensors are DS18B20 Dallas one-wire (temp)
4.7 kOhm pull-up resistor
RESET
ICSP2
DIGITAL (PWM=~)
UNO
Arduino
ON
ICSP
POWER
ANALOG IN
fritzing

Activity 9 – GPS, external power

- With the Arduino unplugged, add the GPS module (to the upper right hand corner of your breadboard) using the table on the next slide.
- Place `TinyGPS` into your Arduino library. This activity also uses `SD`, `SPI`, and `SoftwareSerial` (already in your library).
- Plug in the USB cable and load the sketch `GPS_on_SD`. Upload the sketch to the Arduino Uno (watch out for chip select). Open the serial monitor (confirm baud rate) and watch it run. The GPS probably won't get a lock so the data will probably be uninteresting.
- Unplug the USB cable (AKA the programming cable). Power the Arduino Uno using a 9-volt battery through the power jack instead. You can now run sketches independently from the laptop (but no more serial monitoring!). This mode of operating is critical for applications like quadcopters, ballooning, and rocketry.
- The LED on the GPS gives its lock status. Fast blink means "not locked"; slow blink means locked. You are unlikely to get it to lock indoors so take the package outside until it gets a lock then walk around a bit to collect some interesting data. Come back in and look at the GPS data that was logged to the micro-SD card.

Adafruit Ultimate GPS Breakout

Pin	Function	Attached To
3.3V	Provides a 3.3V Output	Not connected
EN	Enables Sleep Mode	Not connected
VBAT	Allows battery input for RTC	Not connected
FIX	Output at same time as fix LED	Not connected (May want to connect depending on your project)
TX	Transmit	D5
RX	Receive	D6
GND	Ground	GND
VIN	5V Power	5V
PPS	Pulse per second output	Not connected