

Arduino Programming and Building Lab

Objective

To utilize the Arduino IDE Software platform to write simple control logic commands. This code will be used to control a variety of inputs and outputs connected to an Arduino Uno microcontroller. This lab will also expose students to simple control codes, and online resources available for the Arduino software.

Background

In conjunction with the different models of microcontrollers, the Arduino Company provides an open-source Arduino Integrated Development Environment (IDE). This programming environment is written in Java and has been designed to introduce programming to an audience that is less familiar with coding. Programs written in this environment are called ‘sketches’, and the language used is a simple one with support for C and C++ programming languages.

A basic Arduino sketch consists of at least two functions: a ‘setup’ function and a ‘loop’ function. The setup function executes once only at the start of the program execution. It performs any actions that are initially required to run the rest of the program, such as initializing any external components and setting the communication frequency between the Arduino board and PC. The loop function acts as the programs driver and runs in a continuous loop (i.e. “loop forever”). This loop specifies the order of operations that the microcontroller will perform on an ongoing basis.

Arduino boards, as well as many Arduino-compatible external components, contain hardware for serial communication. This form of communication can send data a high rate, but transmits data bits serially (i.e. one after another, a single bit at a time). This allows the components of the system (or a PC) to communicate larger and more complex sequences of data much more easily, with less demands on physical input or output pins.

Because of Arduino’s open-source nature, third party companies, as well as end users are free to take the software and tailor it to their individual needs. This means that for almost any application, sketches are available for download online. Companies making external devices (such as a motor shield or sensors) for use with Arduino boards often have example sketches and sketch libraries available for specific use for their products. These resources can be used to create a custom sketch that can use many different peripherals at once, by combining aspects of many

different example sketches. Before writing your own code, like we will be doing here, it is worth looking online to determine if something similar is available online.

Downloads

This lab utilizes Arduino IDE libraries that do not come standard with the Arduino IDE environment. In order for these programs to function properly, these libraries must be downloaded and inserted into the libraries folder within the system files. A list of the required libraries can be found in the troubleshooting section. These should already be installed on the lab computers, but may have to be installed if a student is using their personal computer or if the lab PCs have been corrupted or altered (e.g. by other students).

The Arduino IDE can be downloaded from here: <https://www.arduino.cc/en/Main/Software>

The libraries used in this lab are:

- One wire: https://github.com/adafruit/MAX31850_OneWire
- Dallas Temperature:
<https://github.com/milesburton/Arduino-Temperature-Control-Library>

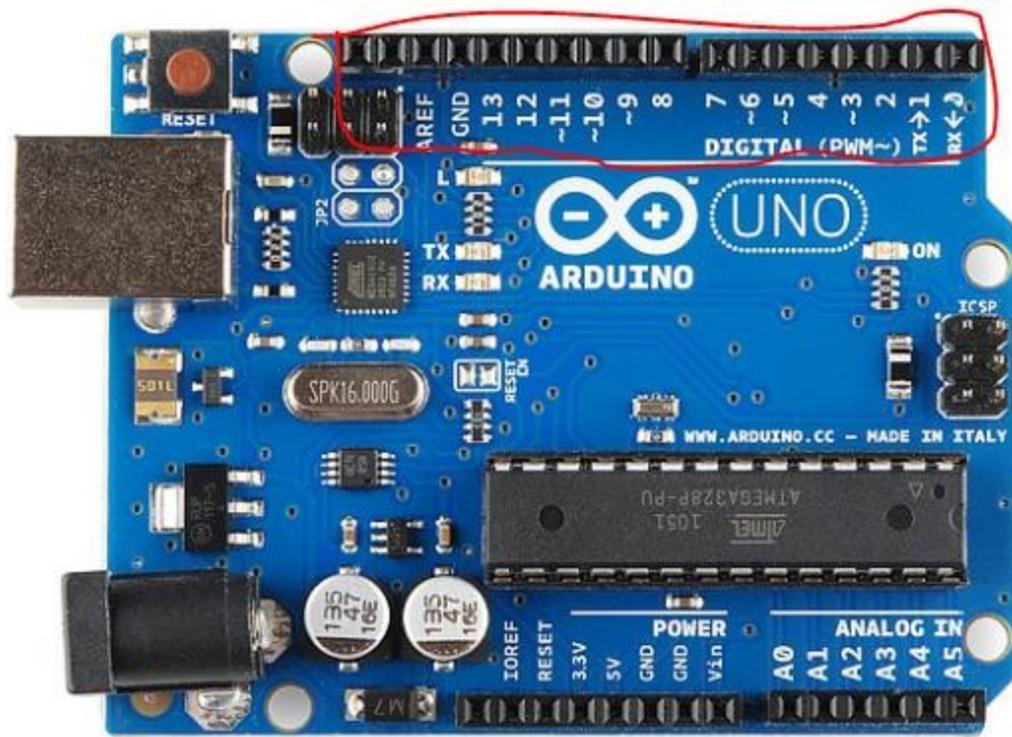
To install the required libraries, unzip the downloaded folders to a single folder, then move that folder to the Arduino libraries folder in the system files folder. Restart the Arduino IDE for the libraries to appear. For most windows users, the default pathway will be Program Files (x86)>Arduino>libraries, but this can change with different operating systems, different versions of windows, or if the Arduino IDE was installed to a different location on the computer.

Adafruit.com has a useful tutorial on installing libraries that can be found here: <https://learn.adafruit.com/adafruit-all-about-arduino-libraries-install-use>

Arduino Board

Arduino senses the environment by receiving inputs from many different types of sensors, and can also affect its surrounding environment with outputs that control lights, motors, heaters, or other actuators. These input and output signals are made available on a digital pins header (see diagram below).

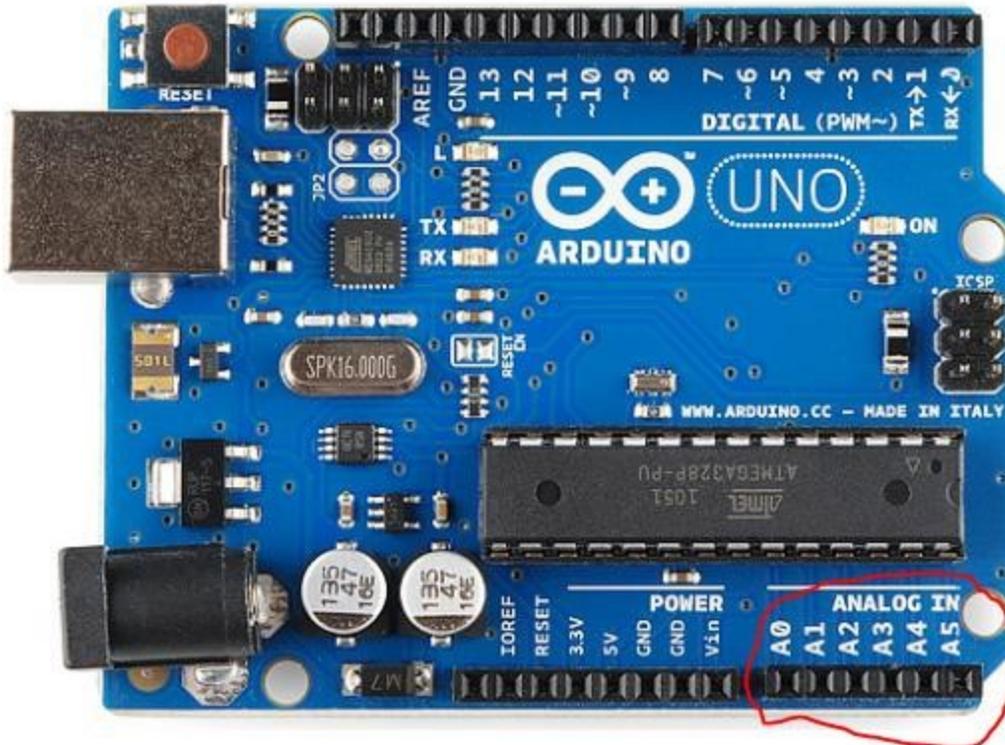
Digital Pins



- These pins can be configured as digital INPUTS and used to send signals into the Arduino processor (e.g. to read sensed logic voltage levels). Digital signals are normally binary and only have two distinct voltage values (0V or ‘low’ or 5V or ‘high’).
- The pins can also be configured to act as an OUTPUT to drive digital ‘high’ or ‘low’ logic voltages from the Arduino processor (e.g. to turn things on or off).
- **Pin13 (LED):** There is a built-in LED pre-connected to digital pin 13. When the value of the pin is driven HIGH by the processor, the LED on the board is illuminated, when the pin is LOW, it's turned off. This can be used as a status indicator when programs are running.

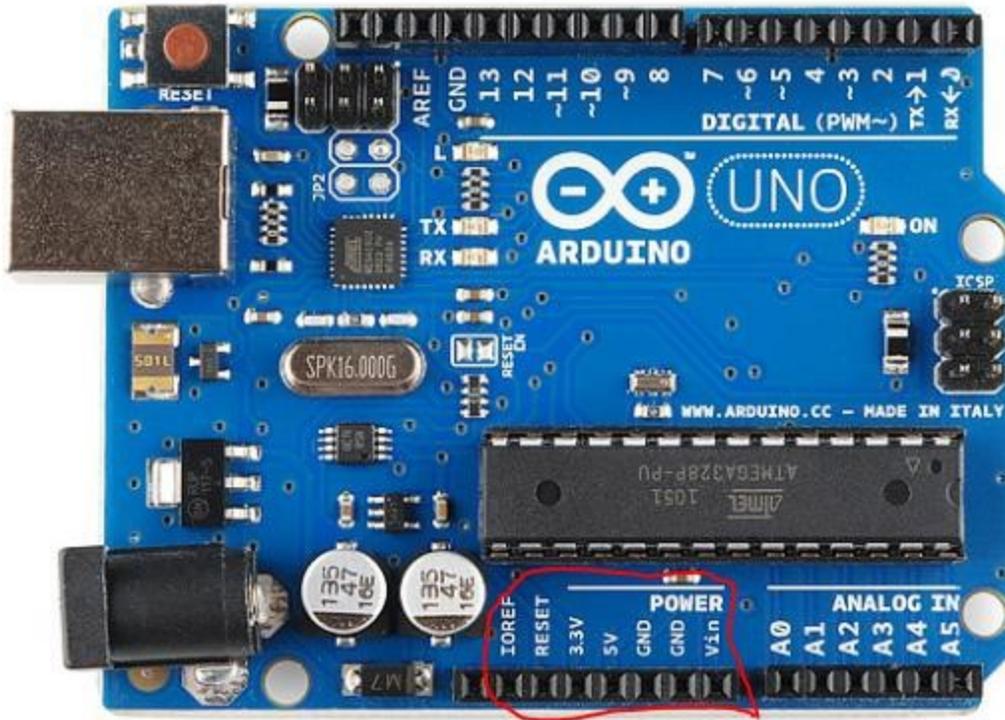
There are many types of real-world signals which are not binary digital ones (i.e. they have more voltage levels than just ‘high’ or ‘low’). These kinds of signals can have a continuous range of voltage values and are termed analog signals. These signals need to be converted into digital representations which can be used inside the software-executing portion of the Arduino processor (which only uses binary digital signals). Analog signal inputs can be accepted for conversion into digital via the Analog Pins header (see diagram below). The Analog to Digital (A/D or “A to D”) conversion is done inside the processor with specialized circuitry.

Analog Pins



To help power simple external circuitry, the Arduino also provides a few common digital voltage output values (e.g. 3.3V and 5V) as well as a 0V or reference voltage (Ground or ‘GND’). These pins are located next to the Analog pins header (see diagram below). The ‘Vin’ pin is a diode-drop voltage away from the input voltage being supplied to power the Arduino board and generate the other voltages.

Power Pins



Arduino Software

You can tell your Arduino what to do by writing code in the Arduino programming language and using the Arduino development environment. Some important facts for the lab are given below and are intended to help new programmers (this is probably less useful for more experienced Arduino programmers). If you need more help, ask the Teaching Assistant for clarification or use the internet to try and find out the answers to your questions:

- **Comment bars:** “/* */” and “//”: These allow you to write commentaries in your code. Everything you write in between “/* */” or after “//” will be a comment and will not affect the code.
- **Semicolon “;”:** This character ends a program statement and lets the compiler know ‘the end of the current line/statement’. A compiler is a computer program that transforms source code written in a programming language into another computer language, usually one that can be executed more easily on a real processor.
- **“setup” function:** The setup function runs only once and performs any actions that are initially required to run the rest of the program, such as initializing any peripheral

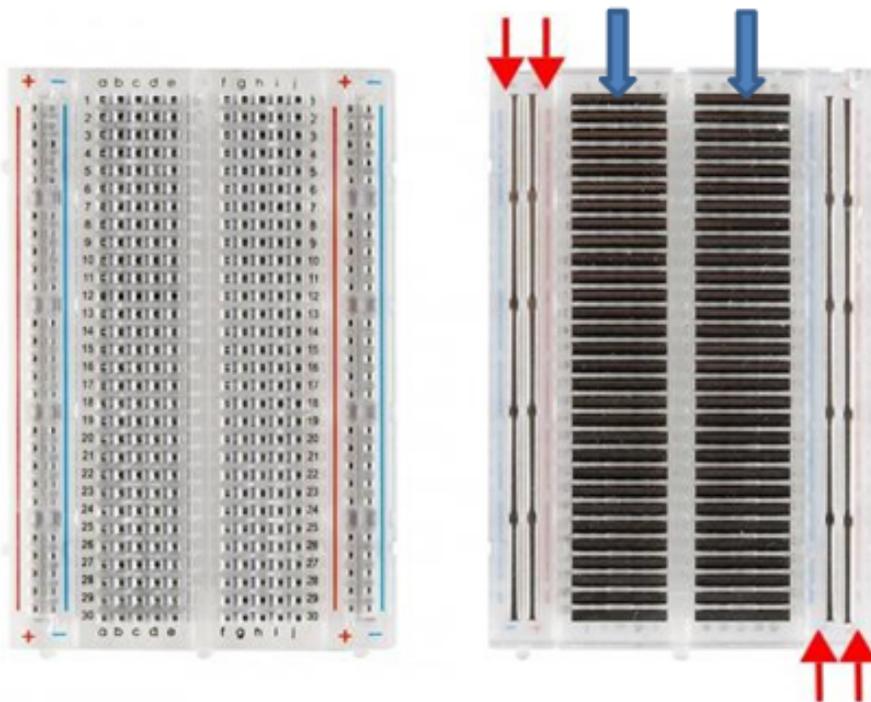
components and setting the communication frequency between the Arduino board and PC. <https://www.arduino.cc/en/Reference/Setup>

- **“loop” function:** The loop function acts as the programs driver; it runs in a continuous loop, and specifies the order of operations that the microcontroller will perform. <https://www.arduino.cc/en/Reference/Loop> . Execution starts at the top, goes through the contents of the loop and then starts executing from the top again. This procedure is repeated forever.
- **Brackets “{ }”** Start and ends a function or is used to group a set of different statements together, making them effectively a single long sequential statement, that is executed as a ‘glob’.
- **pinMode():** Configures the specified pin to behave either as an INPUT or an OUTPUT, see <https://www.arduino.cc/en/Reference/PinMode> for more information.
 - Syntax: pinMode(pin, mode)
- **digitalWrite():** Writes a HIGH or a LOW value to a digital pin.
 - If the pin has been configured as an OUTPUT with pinMode(), its voltage will be set to the corresponding value: 5V (or 3.3V on 3.3V boards) for HIGH, 0V (ground) for LOW, see <https://www.arduino.cc/en/Reference/DigitalWrite> for more information.
 - Syntax: digitalWrite(pin, value)
- **Delay:** Pauses the program for the amount of time (in milliseconds) specified as the parameter.
 - Syntax: delay(ms) <https://www.arduino.cc/en/Reference/Delay>
- **Variable:** a variable is a storage location that is paired with an associated symbolic name (an identifier), which contains some known or unknown quantity of information referred to as the value of that variable.
- In this lab we are going to use two types of variable value types: “int” and “float”.
 - Int : Allows you to store integer (i.e. counting whole) numbers. <https://www.arduino.cc/en/Reference/Int>.
 - Syntax: int variable = value;
 - Float : Store floating-point numbers, a number that has a decimal point and an exponent. The result is that these kinds of variables can have a much larger range than is possible for integers, but they will have some roundup errors because they are implemented with finite precision (i.e. only a certain number of decimal digits can be used). See <https://www.arduino.cc/en/Reference/Float> for more information.
 - Syntax: float variable = value;
- **Serial.begin():** This starts off the serial communications interface hardware. You will need this, if you want to print anything using the serial monitor.

- For more information, see <https://www.arduino.cc/en/Serial/Begin> for more information.
- Syntax: `Serial.begin(9600)` sets the serial interface to run at a rate of 9600 serial bits/s
- **Serial.print():** Prints data to the serial port as human-readable text, see <https://www.arduino.cc/en/Serial/Print> for more information.
 - Syntax: `Serial.print(value)`
- **“if” statement:** See <https://www.arduino.cc/en/Reference/If> : This lets you conditionally execute the following statements, depending on a tested condition.

Breadboard

A breadboard is used to prototype a temporary circuit. You can build, test and analyze a circuit without any permanent connections. It is made up of terminal strips and power rails. The **terminal strips** are used to hold any number of components in place and make electrical connections in a horizontal row. The **power rails** are the long vertical strips and are used to facilitate power (+) and ground (-) connections by placing them all in one column. For some general background on how such a breadboard actually works, take a look at the YouTube video about breadboarding at: <https://www.youtube.com/watch?v=6WReFkfrUIk> .



Apparatus and Equipment Overview

- The equipment that will be used in this lab includes:
- 1 x Arduino Uno R3 Microcontroller
- 1 x USB 2.0 Type A Plug to USB 2.0 Type B Plug Cable
- 1 x Lab or Personal Computer (running Windows, Macintosh, or Linux OS)
- Arduino IDE Software
- Corresponding Arduino IDE Libraries (outlined below)
- 1 x 2 channel 5V relay
- 1 x Temperature sensor DS18B20
- 1 x 4.7K ohm resistor
- 1 x Half breadboard
- 1 x 12v wall plug
- 4 x Male-female jumper wire
- 7 x Male-male jumper wire
- 1 x Heating pad 5x10cm
- 1 x 12v fan
- 1 x Aluminum block

Pre-Lab Preparation

Before arriving to the lab, students should review the lab manual and familiarize themselves with the lab setup and procedures. Students may use their own computer for this lab if they wish (the software is free), provided that they have downloaded and installed the Arduino IDE, as well as the required libraries outlined in downloads section. Reviewing the Arduino IDE syntax, as well as trying to write some of the programs beforehand is also encouraged. A useful list of commands used in the Arduino IDE can be found here:

<https://www.arduino.cc/en/Reference/HomePage>.

All of the lab PCs have the software already installed.

Pre-Lab Questions

What is the programming language used for Arduino?

What are the 2 main sections in an Arduino program?

What are the 3 main sections on the Arduino Uno board?

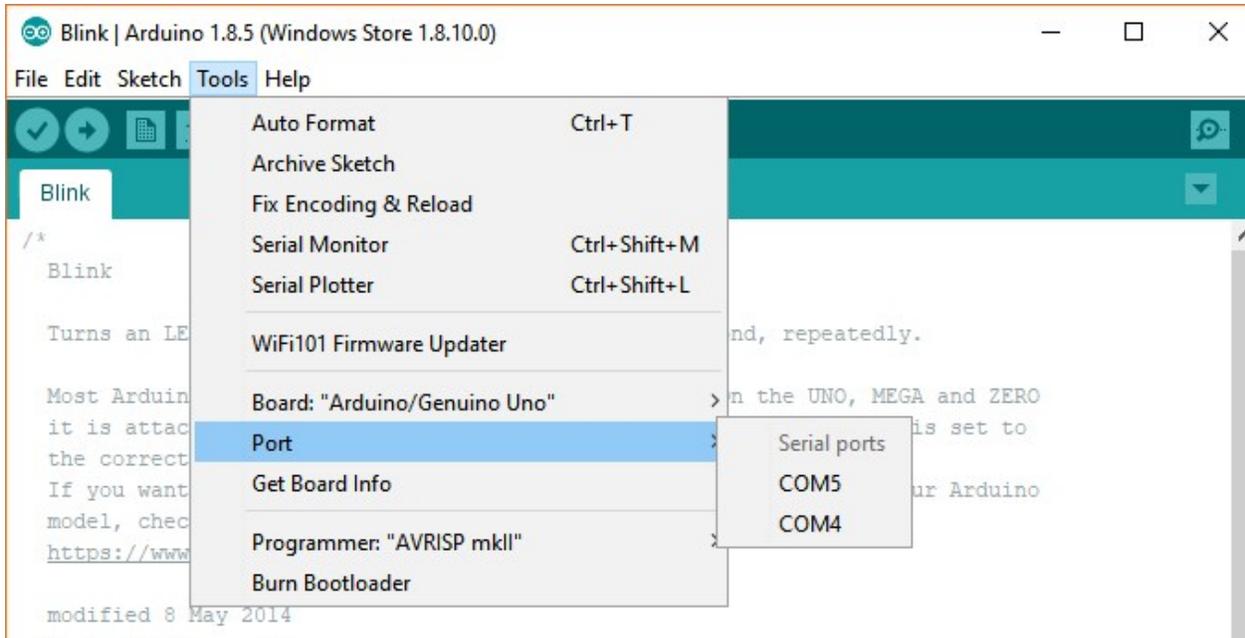
What is the difference between terminal strips and power rails on a breadboard?

What does electrical polarity mean?

Part A – LED Blink Program

The next part of this lab reviews the basic method of connecting and programming an Arduino microcontroller. It utilizes a sample sketch included in one of the Arduino IDE's default libraries, and is primarily used to verify that the microcontroller and the connection are both functioning properly and gives a basic idea of how to run a very simple Arduino sketch program.

1. Connect the Arduino board to an open USB port on the computer with the USB cable. The Arduino board LEDs should light up, as power is drawn through the USB connection.
2. Launch the Arduino IDE Software, and pick 'Arduino/Genuino Uno' from the dropdown menu in Tools>Board. Back in the 'Tools' tab, select 'Port' and choose the serial port that the Arduino is connected to. If the correct serial port is not apparent, unplug the Arduino, and see which port disappears. Re-connect the board, and select that port. Remember, if you don't select the correct serial port, then you won't be able to see any printed output.



3. Open File>Examples>01.Basics>Blink. This will open a new sketch that looks like the one below.

```

modified 2 Sep 2016
by Arturo Guadalupi

modified 8 Sep 2016
by Colby Newman
*/

// the setup function runs once when you press reset or power the board
void setup() {
  // initialize digital pin LED_BUILTIN as an output.
  pinMode(LED_BUILTIN, OUTPUT);
}

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

```

4. Now verify and upload the sketch to the Arduino board. To verify, click the check mark in the upper left hand corner. The Arduino IDE will try to compile the sketch (without uploading to the board), and warn of any syntax errors in the programming. Once this is

complete, press the upload button (arrow beside the check mark). The sketch will recompile, and upload to the board.



5. Once the upload is complete, the program will automatically run on the microcontroller. In this case, the LED labelled 'L' on the Arduino should flash slowly.

How fast does it flash (i.e. what is the flashing rate and duty cycle)?

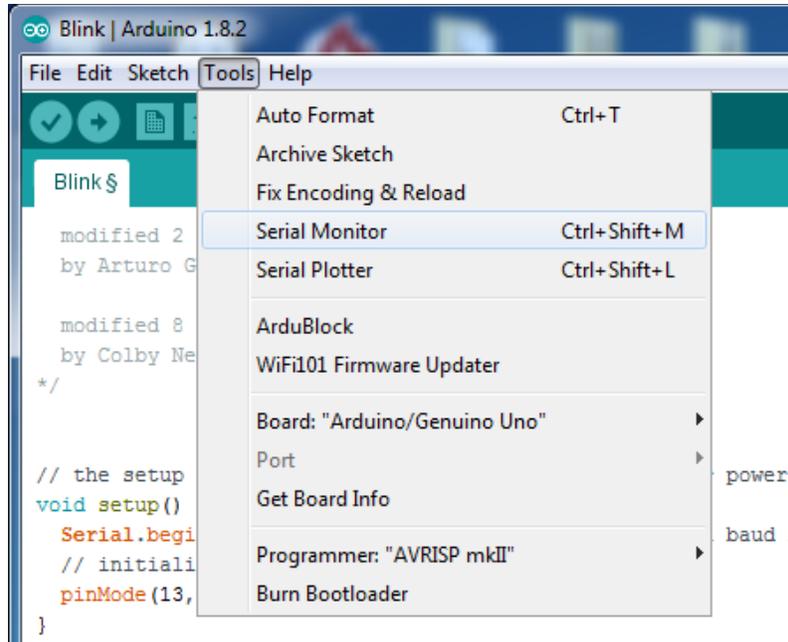
Try to change the value in your delay. Change it to 5000 and to 300, verify and upload the code. What happens to the "L" on the Arduino Board in the two cases?

6. Change the variable LED_BUILTIN to 13, they represent the same pin on the Arduino. Print the value of this variable in the serial monitor by adding Serial.begin(9600); and Serial.print(LED_BUILTIN); like in the following picture.
7. Upload your new code.

```
// the setup function runs once when you press reset or power the board
void setup() {
  Serial.begin(9600); //start a serial connection for a baud rate of 9600
  // initialize digital pin LED_BUILTIN as an output.
  pinMode(13, OUTPUT);
}

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

8. Open the serial monitor to see what is being printed, select Tools>Serial Monitor.



9. Replace `Serial.print(LED_BUILTIN)` by `Serial.println(LED_BUILTIN)` and then by `Serial.println("LED_BUILTIN")`. Verify and upload your code. What happened in those two cases?

Part B – Temperature Sensor Reading

This part of the lab uses a simple breadboard circuit to hook up the various components that are required for the lab (see the diagram below).

Note that the neater your circuit is, the easier it will be to debug it, if/when there are problems and the less likely that wires will come loose, etc. In general, you want to have your wires sit flush with the plastic surface of the breadboard, so that you can see all of the connections that you are making and so that people trying to help you can see them too! Using color-coded wires (e.g. red for +5V and black for ground) is another common practice to help simplify debugging. While the circuit will work with any color of wire, following such conventions will make your circuit 'better', much like the improvements that you get in usability, when including comments in your computer programs.

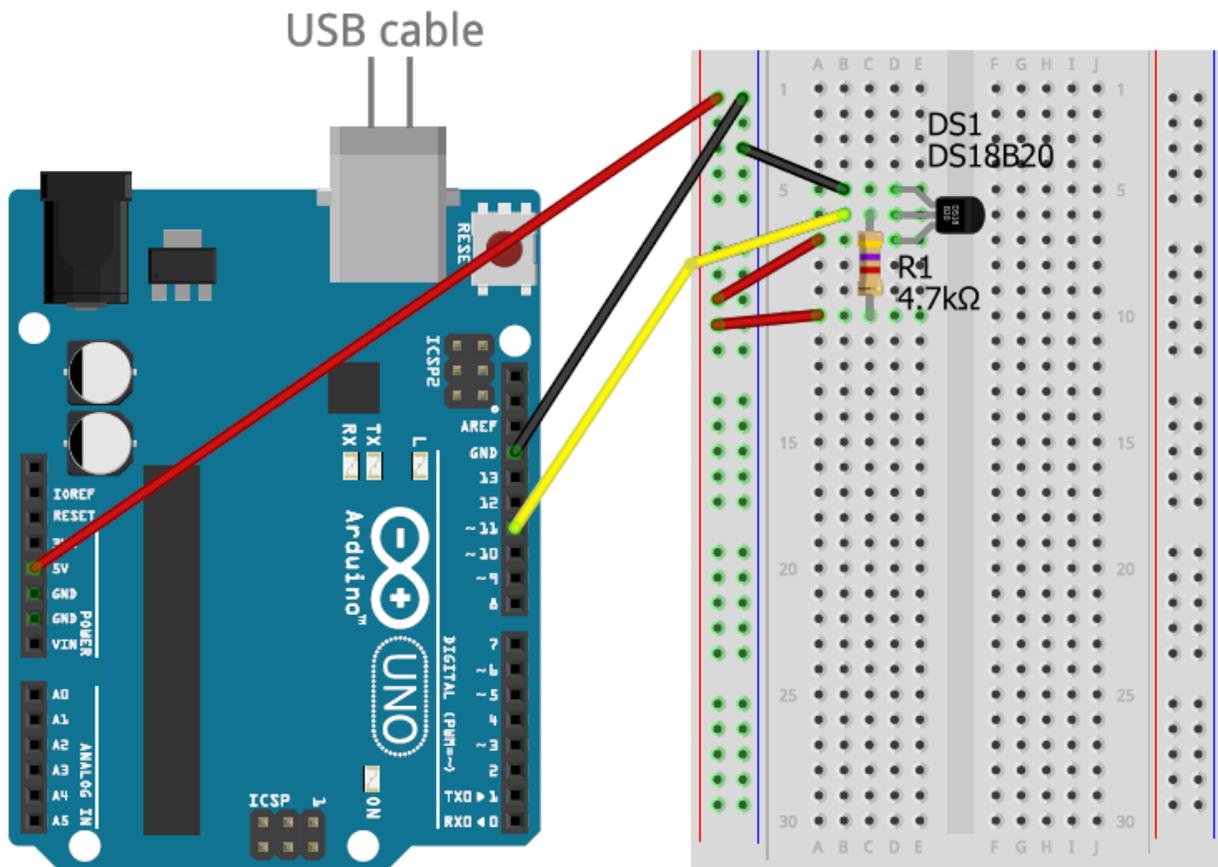
This program will read the temperature of the environment using a DS18B20 temperature sensor. The temperature will be printed on the serial monitor of the Arduino IDE. The temperature

sensor has an operating temperature range of -55°C to $+125^{\circ}\text{C}$ and is accurate to $\pm 0.5^{\circ}\text{C}$ over the range of -10°C to $+85^{\circ}\text{C}$.

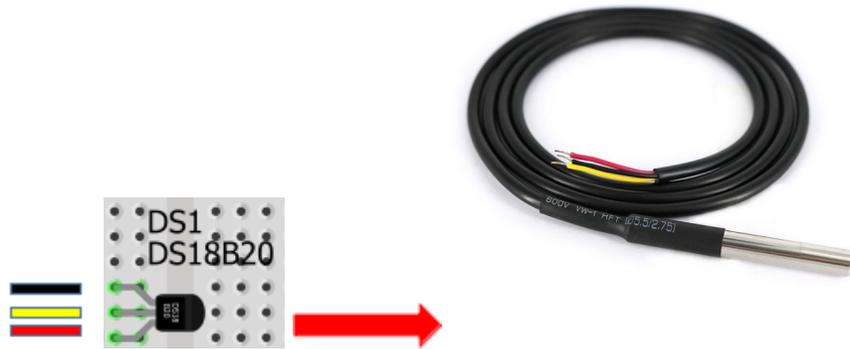
1. Open an example sketch in the Arduino IDE (File>Examples>MAX31850 DallasTemp>Simple). Make sure that the correct microcontroller and communication port is still selected by checking the bottom right of the window. You can follow the meaning of the commands by reading the comments (grey words) in the code.



2. Connect the DS18S20 temperature sensor to the Arduino board by using the wiring diagram shown in the figure below.



*Note: the above temperature sensor is the symbol but the following picture is what it actually looks like. Using the image below as a guide we can connect the temperature sensor. The **red** wire must connect to the 5V power source, **yellow** will connect to the digital I/O port (#11), and finally **black** will connect to the common ground.



The temperature sensor needs to be installed securely inside the aluminum block.

3. Make sure that the 'OneWire' and 'DallasTemperature' libraries are in orange at the top of the sketch, meaning they are recognized and properly installed. The link to download the libraries and how to install them can be found at the top of this manual.

```
Simple §  
#include <OneWire.h>  
#include <DallasTemperature.h>
```

4. Change the value of the ONE_WIRE_BUS variable to 11 since that is the pin where our sensor is connected to the Arduino.

```
#include <DallasTemperature.h>  
  
// Data wire is plugged into port 11 on the Arduino  
#define ONE_WIRE_BUS 11
```

5. At the end of the loop, add a line to print to the serial monitor to indicate the units of our value and a delay so we measure the temperature every 0.5 seconds, like in the following picture.

```
Serial.print("Temperature for the device 1 (index 0) is: ");  
Serial.println(sensors.getTempCByIndex(0));  
Serial.println("C");  
delay(500); // wait 500 milliseconds  
}
```

6. Connect the Arduino board to the computer by using a USB cable and upload the sample code to the Arduino controller. Open the serial monitor (Tools>Serial Monitor) to see if you are reading the temperature correctly.

7. Unplug the USB cable when you are done.

****Do not disconnect the temperature sensor****

Part C – Temperature controller

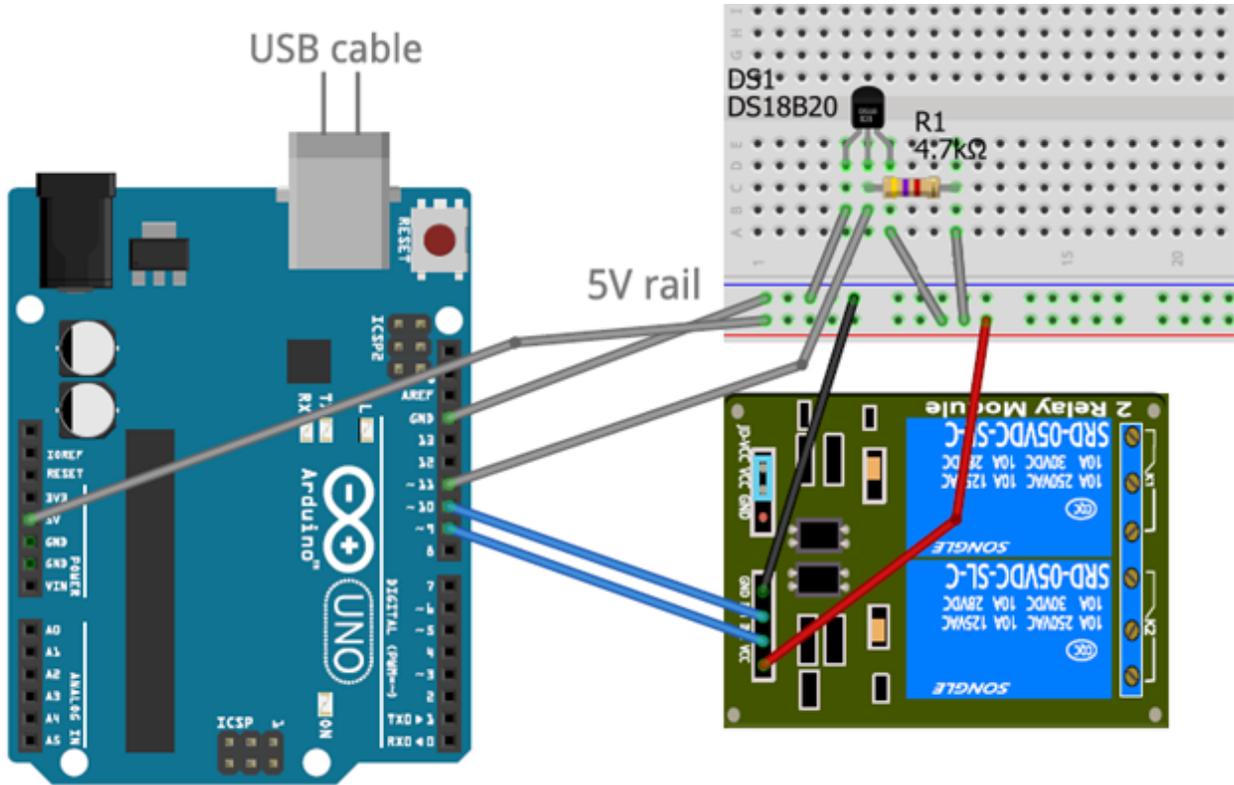
In this part of the lab, you will have to create a program to control the temperature of a piece of material (aluminum) using a heating pad (heater) and a fan (cooler). Your code must heat and cool the material until the temperature reaches the desired value. **You should use the temperature sensor code of the previous part** to read the temperature so you can see how the temperature of your material is changing in real time. The first value that you will have to reach is 35°C. You must record the time that it takes for the aluminum block to reach the desired value of the temperature and plot a graph of temperature vs time. The second value that you will have to reach is 25°C (requiring you to cool the block from 35°C to 25°C). You must also record the time that it takes to reach this temperature and plot another graph of this, temperature vs time. In the end there should be one graph showing the upward trend (heating) and downward trend (cooling) plotted temperature vs. time.

At the end of the Lab you should be able to plot a graph using Microsoft Excel.

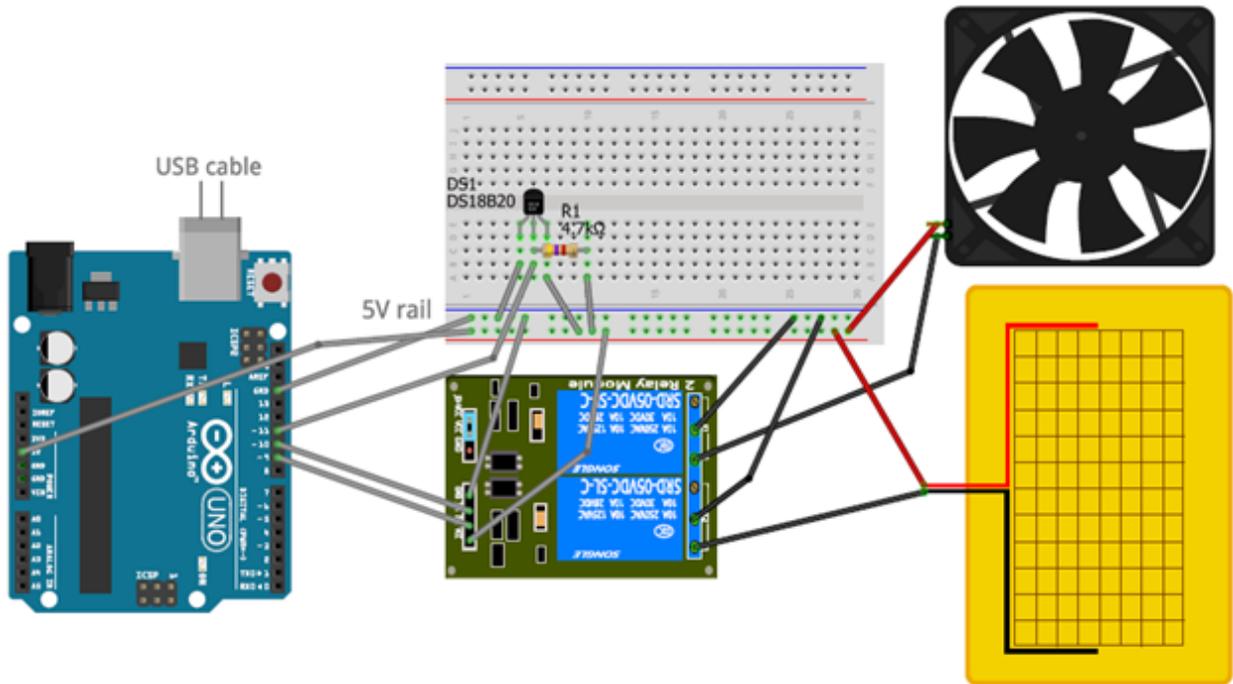
To make the rest of the circuit you will add a fan, a heating pad and a relay to your existing sensor circuit. The relay acts like a switch and is controlled by the Arduino. You will connect the fan and heating pad to separate channels on the relay so they are controlled independently.

Start by connecting the relay to the Arduino. The grey wires in the picture represent the ones that were already there from the previous temperature sensor circuit, you can ignore them.

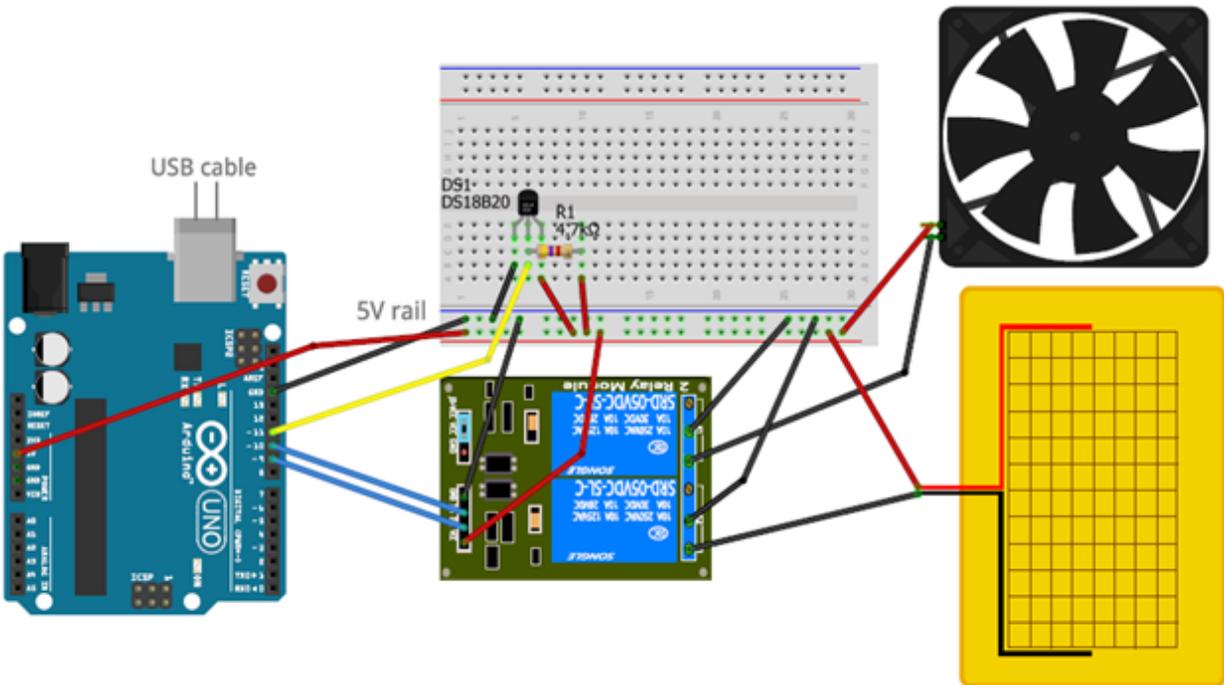
- Relay GND to GND rail
- Relay VCC to 5V rail
- Relay IN1 to Arduino 10
- Relay IN2 to Arduino 9



Next attach the fan and heating pad to a channel on the relay, choosing the middle and left pins. This is so the circuit will be normally open (NO). The relay is low-level trigger which means a low signal from the Arduino will close the circuit and cause the fan or heating pad to turn on. These components can work at 5V but will not be as effective as at 12V, therefore the heating pad will not be very hot and the fan may have a hard time to start turning.



Here is the full circuit diagram without any grey wires:



1. For this section, open the template for the code from Brightspace and replace the `***` in the code with the appropriate values, based on what you learned in the previous sections.
 - a. Initialize all the variables that you are going to need.
 - b. Define the OUTPUT pins that you will use for the heater and the cooler.
 - c. Write the conditional statement that will control the temperature.

What is the difference between an “if” statement and an “if ... else” type of statement?

- d. Control the fan and heating pad with digital commands from the Arduino, you will notice that the relay will be activated (closed circuit) with a LOW signal.
- e. Record the time until the materials reach the desired temperature (write software to do so!).

Explain how you will measure time with your code?

- f. Print in the serial monitor how the value of the temperature is changing and the time when the data has been recorded (i.e. you will need to be able to measure the time, so make sure that you have answered the question above).

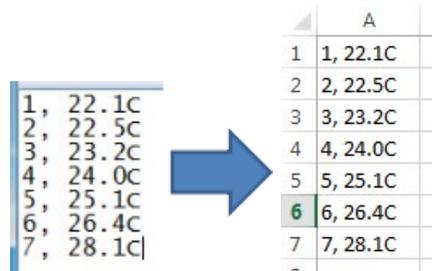
Hint: Don't forget that you can simplify the work of importing data into Excel by using special characters in your formatted output!

- 2. All components need to be grounded and controlled/powered by the Arduino. If you are unsure and don't want to fry electronic components, ask the teaching assistant to help verify your circuit before powering it up!
- 3. Connect the Arduino board to the computer by using a USB cable and upload the sample code to the Arduino controller.

Graphing raw data

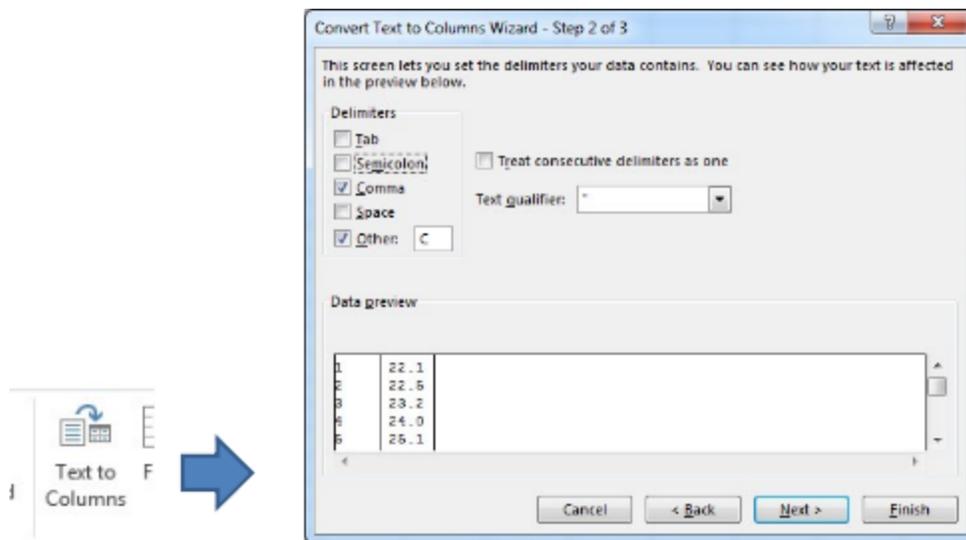
From the Arduino serial monitor you want to get data that can easily be put into a graph and manipulated. Since the graph will plot the different coordinates of temperature vs time you want something that looks like (time, temperature). If you get the following data from the serial monitor, copy it into an excel spreadsheet. Even better, you could use a println statement with a specific format that simplifies the importing process into excel. Then, you just need to capture or cut and paste all of your results into a text file which will be easy to import (see the previous lab on how to import text files into Excel).

Hint: In your println statement, use the appropriate delimiting characters (like commas).



	A
1	1, 22.1C
2	2, 22.5C
3	3, 23.2C
4	4, 24.0C
5	5, 25.1C
6	6, 26.4C
7	7, 28.1C

Now you can divide the data into different columns using the Text to Columns tool. It can be found in the DATA tab. First select all the cells that you want to split and click the button. Then choose Delimited (since we have commas between our values) and Next>. In the next window pick 'Comma' and type C for 'Other'. It will show you a preview of your new columns and you can now click Finish.



Select the new columns again with your data to make a graph. Click on the type of graph you want in the INSERT tab. This will plot all your points. You can now change the chart/axis titles and other options with the '+' beside the graph when you click on it.

