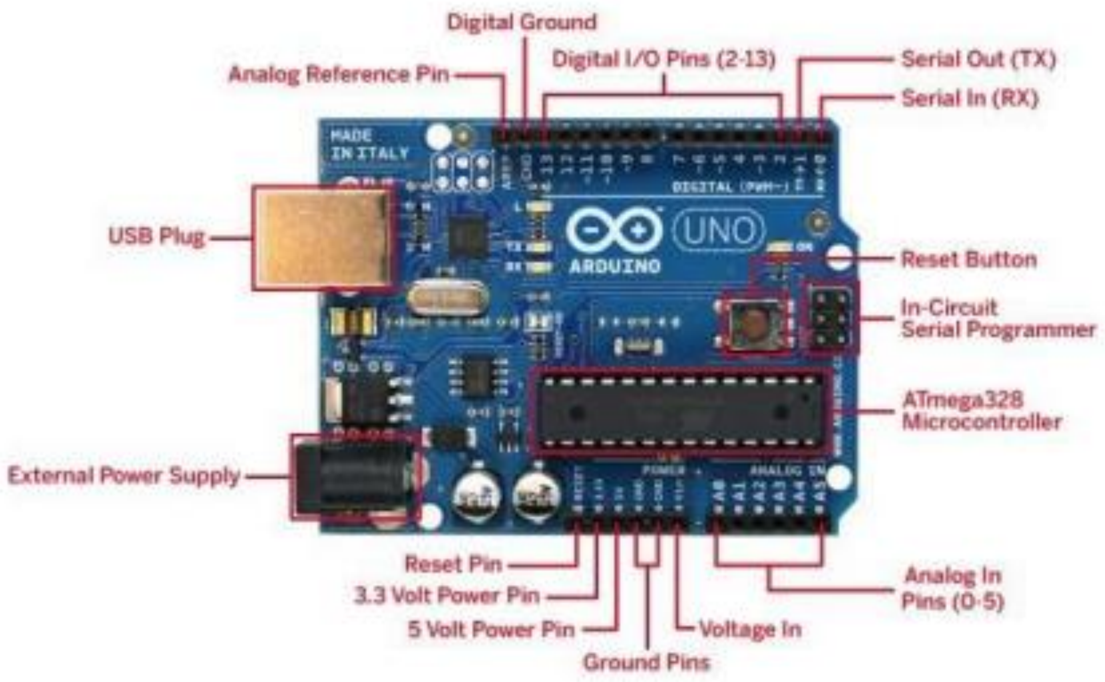
ARDUINO MANUAL //

INTRODUCTION

This manual's purpose is to help guide you through any Arduino projects you would like to accomplish. There are many detailed aspects of the Arduino that a user must be aware of. This manual will be your best friend to help you get started! Please do read it carefully as it will provide you with tips, tricks, How-To's, and general safety precautions so you don't *kill* your Arduino!

THE ARDUINO BOARD

The Arduino Board at first glance may look very confusing. There are many different components to the board and every single one has a purpose. It is very important to know these details when building a project as simple mistakes can cause a project to fail or cause the Arduino to break. Here is the board layout of an Arduino (your Arduino may look slightly different):



For the majority of projects, only the Digital I/O pins, Analog pins, USB plug, 5-volt Pins, and ground pins will be used. The rest of the pins are used for specialty purposes which will are not required for many projects.

The following will discuss the details of what each component does on the Arduino.

ANALOG AND DIGITAL PINS

Everything on the Arduino board uses electricity. The tiny logic gates and structures on the board use binary: a value is either 1 or 0. For a computer to know if a value is 1 or 0, an electrical signal at a non-zero voltage is supplied to represent a value of 1. In contrast, a signal that is zero-voltage reads a value of 0. With all these signals, computers can make computations using logic gates. Arduino has two primary input/output (I/O) types: digital and analog. Both I/O types use the premise of supplying voltage to send or receive signals in. They do differ and it is quite important to know the two differences.

Analog pins: these pins can read or supply a multitude of values ranging from 0 to 1023, corresponding to input voltages from 0 to 5V.

Digital pins: these pins only have two values: on (HIGH) or off (LOW) where HIGH is a 5V signal and LOW is a 0V signal (ground).

Both sets of pins are very useful for certain tasks. Digital pins are great for turning sensors or lights on or off. Analog pins are great for taking in detailed data that has a range of values like sensors that measure light intensity or soil moisture content. Certain sensors will require you to use the analog pins or the digital pins. **Mixing the two up will potentially break the sensor or Arduino board!**

RESET BUTTON & RESET PIN

The reset button and reset pin seem unhelpful at first but they will be quite useful for many projects. In short, the reset button or reset pin will reset the code on the Arduino to start from the beginning. This will be extremely helpful when you want to run your project again once it finished. If your code has finished running, the Arduino will stop. Similarly, if you want to reset your code mid-way through it running because you already *know* it messed up, you can use the reset feature to start over again.

For almost all projects, just using the reset button will suffice.

Note: The reset button is not always in the location, e.g. sometimes it is next to the USB plug

SHORT CIRCUITS & VOLTAGE REQUIRED TO RUN ARDUINO

As mentioned before, the Arduino uses voltage values to compute code with binary. Just like how we can get extremely hurt by touching voltage that is very high, the Arduino can get extremely hurt if supplied with a voltage higher than its working rating. As well, you can damage the Arduino if you create a short circuit on the board.

Short Circuit: This occurs when you create a circuit with NO load in the circuit

This means that the voltage in the circuit has nowhere to be used. A load "uses" the voltage in a circuit. When there is no load, the circuit will heat up and destroy the components on the Arduino. An example of this is if you connect an I/O pin directly to the ground pin with a wire.

Warning: DO NOT do this – it WILL kill your Arduino.

Other ways to damage or kill your Arduino include:

- Shorting I/O pins to other I/O pins
- Applying a voltage greater than 5V to pins
- Shorting voltage supply pins to ground
- Getting water on the Arduino (which in turn shorts the components on the board)

The operating voltage of the Arduino varies from 3.3V to 5V. On the board, there are two pins called 3.3V Power Pin and 5V Power Pin. Each pin supplies a voltage of the described value. Some sensors will need to be supplied with a certain voltage and these pins can do so. Remember to always double check what voltage is being supplied where. **Students every year accidently break Arduino boards because of simple mistakes that could have been prevented.**

POWERING THE ARDUINO

The Arduino, like many electronic devices, needs to be powered. When plugged into the computer, the USB supplies the power to the board. When you want to power the board without the USB, there is an external power supply input that uses an AC-DC power converter that inserts into the barrel adapter. As well, there is a V_{in} pin that can accept 5V but we highly recommend you do not use this without a proper voltage regulator. DO NOT plug in a voltage greater than 5V into the V_{in} pin. As mentioned, do not use this pin without a voltage regulator as well.

The Arduino can run from 5V to 12V. The recommended operating voltage is 7V-9V. If you are using the barrel adapter, make sure it is running on 7V or 9V.

FINAL REMARKS

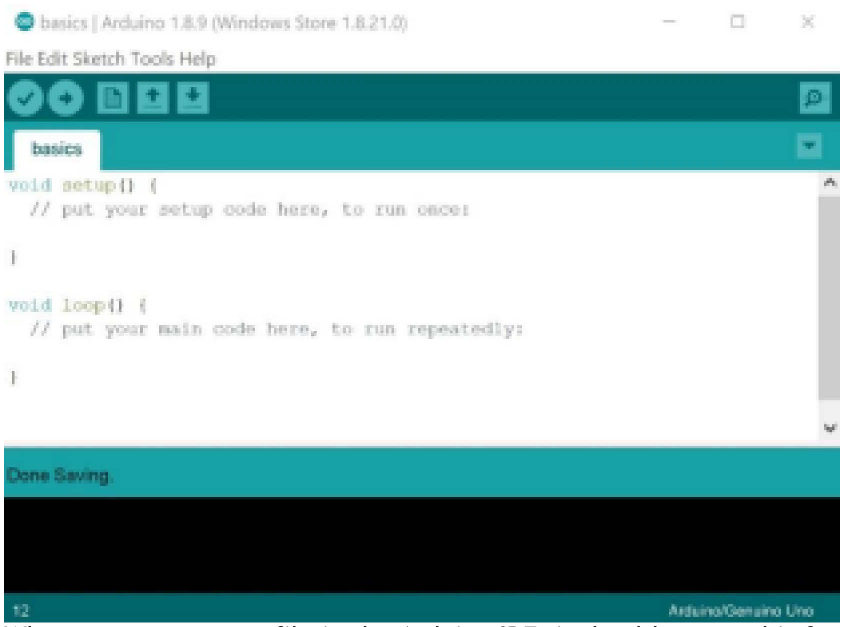
When wiring up all your sensors and components, double check where your wires go and what voltage is being supplied. Refer to the sensor manual (found online or by asking the facilitators) at any point to confirm your wiring layout.

PROGRAMMING AND CODING STRUCTURES

In order to make the Arduino do anything, it must be programmed. Programming the Arduino is done in the Arduino IDE software (located on your computer). Basic programming requirements are needed to program the Arduino. For basic programs, an outline of the code structure will be demonstrated in this document. The Arduino uses a version of the programming language C/C++, with special lines of code to run the Arduino.

ABSOLUTE VERY BASICS

To run code on the Arduino, a specific code structure is required. This is the main loop that your code will reside in. The Arduino code will not compile if it does not follow this framework. The framework will look like this:


basics | Arduino 1.8.9 (Windows Store 1.8.21.0)
File Edit Sketch Tools Help
basics

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

}

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

}
```

Done Saving.

When you open a new file in the Arduino IDE, it should create this framework. If it does not, just enter it in exactly as shown.

Any code inside the void setup() function is run only once. Usually you initialize pin locations and other one-time setup requirements which will be explained further on.

Any code inside the void loop() function, will repeat indefinitely until the code has a stop function or the power to the Arduino is cut off. As well, pressing the reset button will restart the Arduino code.

There are other functions that Arduino can understand but they are more advanced and will be specific for a sensor. Most, if not all, of our sensors will only need these two basic starting functions.

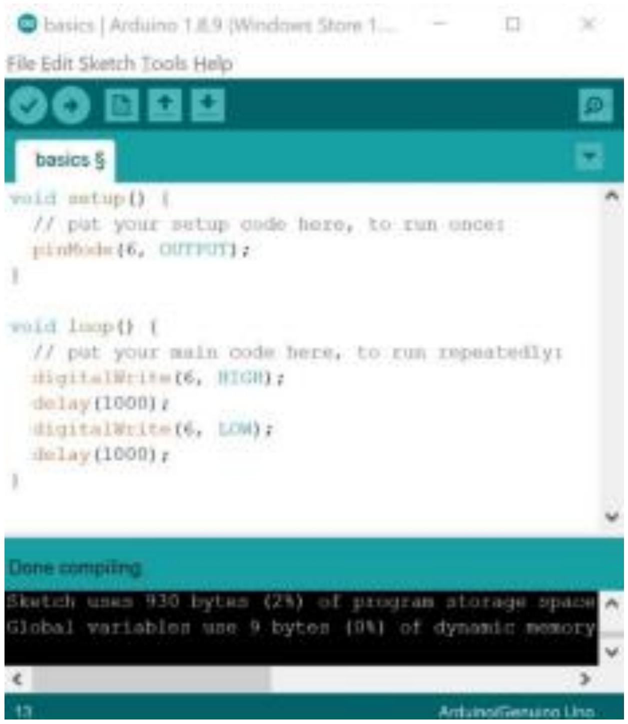
INITIALIZING ANALOG AND DIGITAL PINS

In order to send out a signal, the pins must be initialized. This means the board knows to use these pins in a certain way. The code to initialize a pin will be different from digital to analog yet in both cases, the code must match what is shown here.

Digital Pins:

In this section, we will be programming the Arduino to turn on and off an LED every 1 second.

1) In void setup(), write pinMode(6, OUTPUT); to initialize pin 6 as an output pin.
2) In void loop(), write digitalWrite(6, HIGH); to set pin 6 on high (pin 6 is on)
3) Write delay(1000); to have a 1000ms delay (1000ms = 1 second)
4) Write digitalWrite(6, LOW); to set pin 6 on low (pin 6 is off)
5) Finally, write delay(1000); to have a 1000ms delay


basics | Arduino 1.8.9 (Windows Store 1...
File Edit Sketch Tools Help
basics §
Done compiling
13
Arduino/Genuino Uno

```
void setup() {
  // put your setup code here, to run once:
  pinMode(6, OUTPUT);
}

void loop() {
  // put your main code here, to run repeatedly:
  digitalWrite(6, HIGH);
  delay(1000);
  digitalWrite(6, LOW);
  delay(1000);
}
```

```
Sketch uses 930 bytes (2%) of program storage space
Global variables use 9 bytes (0%) of dynamic memory
```

Note: Pay careful attention to capital letters. If the "W" in digitalWrite is not capitalized, the code will not work.

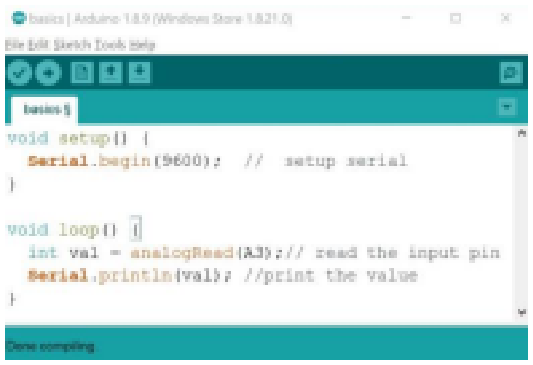
If you follow this code correctly, pin 6 will provide a 5V signal for 1 second then provide a 0V signal for 1 second. Once you hook up an LED with a resistor from pin 6 to the ground pin, you will see the LED blink.

INPUT for pinMode(pin, INPUT); is used for much more advanced topics that utilize pull-up resistors in your circuitry. For now, don't think of this as an option.

Analog Pins:

Analog pins are different from digital pins because you do not need to initialize them with pinMode. As well, you can read variable data with analog pins. One thing that is new will be the serial monitor. This is where you can display data from the Arduino.

The following code will read in an analog value which is an integer than ranges from 0 to 1023. Your sensor attached to this pin will be providing the integer values. Pins for analog values are noted as A1, A2, A3, and so on for Pin 1, 2, 3, and so on.


basics | Arduino 1.8.9 (Windows Store 1.8.21.0)
File Edit Sketch Tools Help
basics §
Done compiling

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

void loop() {
  int val = analogRead(A3);// read the input pin
  Serial.println(val); //print the value
}
```

To start this code, do the following:

1) In void setup(), write Serial.begin(9600);
2) In void loop(), write int val = analogRead(A3);
3) Next, write Serial.println(val);

In order to access the serial monitor, we must initialize it first. By doing Serial.begin(9600), we let the Arduino know we are using it now. Technically, the value 9600 can surprisingly be anything! It is just the baud rate of transfer. However, 9600 is the ideal value to use. Sometimes, the serial port will not open when you run the program.

Accessing the serial monitor: click Tools → Serial Monitor

One final note: This serial monitor only works when the USB is plugged into the Arduino. If an external power is supplied, there is no serial monitor to display values to.

TX/RX Lights:

You will notice two little LED lights on your board that say TX and RX. When you are sending a message to the Arduino (sending code to the Arduino), the RX light will flash. RX stands for Receiving and TX stands for transmitting. Thus, when your code is displaying to the serial monitor, the TX light will flash indicating it is transmitting data. If you do not see this occur when it should be, something is wrong - likely your code!

FINAL REMARKS

Digital and analog pins form the basis of any project with Arduino. Understanding the very important details and differences of the two is key for a successful project. For more advanced students who understand coding quite well, it is easily to implement while loops or if-statements to further improve your project.

FINAL STATEMENT

With these skills in hand, there are endless possibilities to create your ideas with an Arduino. This is only a small sliver of topics that can be discussed on Arduino properties. To better understand your sensors, simply Google the sensor name and the data sheet for it will appear. This will help with the wiring and coding for the sensor. Good luck!

Author: Jason Rock (7 August 2019); modifications by David Bailey (6 March 2021)