Lab 1: Intro to Arduino environment, and actuators and sensors

This lab will familiarize you with the Arduino environment and using Git. If you are already familiar, this will be super easy. Let teammates who aren't familiar with arduino/git do it first. Help each other out.

This lab will work on an arduino nano but you'll need to connect an external LED. Arduino nanos, LEDs, breadboards, and resistors are available in the Maker Space vending machine. You can get everything for under $5 with bevo bucks.

Github Instructions

1. Create a github account and log in. Use your utexas email so you can get the free github education benefits.
2. Accept the assignment link from the canvas announcement. Link your github account to your EID.
3. If you team doesn't exist yet, create it. Please name your team Team_# with your team number from canvas. You can then create a repo. Check the box that says initialize with a README. This will create a repo for your team.
4. Install git on your computer following the instructions here: https://git-scm.com/book/en/v2/Getting-Started-Installing-Git
5. Once you have git installed, you can clone your repo that you created in step 3 to your computer. Use git bash, linux command line, or mac command line to clone the repo wherever you want. I recommend making a folder to put all your repos for this class. Create a branch with your name and start coding.

Git can be a little tricky to get the hang of, but it is extremely useful and will make your life a lot easier going forward, especially for group projects. I recommend reading this article: https://rachelcarmena.github.io/2018/12/12/how-to-teach-git.html
Everything is googleable. The most important commands are clone, add, status, commit, push, and pull. I've uploaded a short cheat sheet to canvas. If you have questions, come talk to me in office hours.

I also recommend setting up an SSH key with your github account so you don't have type in your password all the time. Follow these instructions:
https://help.github.com/articles/generating-a-new-ssh-key-and-adding-it-to-the-ssh-agent/

Lab Procedure

1. Go to https://www.arduino.cc/en/Main/Software to download and install the Arduino IDE. When you open the IDE you'll see a blank sketch with a setup function and a loop function. *Recall from class, why do embedded systems need a setup and a loop?*

2. Let's program our board with this empty project. First we need to tell the IDE which board we are using. Go to Tools > Board: ... and select "Arduino Genuino Mega or Mega 2560". Also go to Tools > Ports and select the COM port for your arduino. Now look under File and see a check mark and an arrow. The checkmark is to compile your code, and the arrow is to program the board. If the code is not compiled, clicking the arrow will first compile the code, then program the board. *Think back to EE306; why does our code need to be compiled?*

3. Now let's add some hardware. Your board has an onboard LED we'll start with. Go to File > Examples > Basics > Blink. Read through the code for understanding, then program the board. You should see the LED blinking.

```
void timedBlink (int interval){
  digitalWrite(LED_BUILTIN, HIGH);   // turn the LED on (HIGH is the voltage level)
  delay(interval*1000);                    // wait for the interval
  digitalWrite(LED_BUILTIN, LOW);   // turn the LED off by making the voltage LOW
  delay(1000);                              // wait for a second
}
```

4. Now create a new sketch inside your repo that you cloned. Make sure you're on your own branch. After creating the sketch, use git add * to add the sketch folder and files. Check that they're being tracked using git status. Copy the setup code from the blink example. Put the code above into the space between setup and loop. Use this code to obtain the same result as before. You'll need to change the code in the loop to call this new function "timedBlink", and you'll need to pass the function a number for the parameter "interval". Hint: you only need 1 line of code in the loop. After you get that working, edit the loop again so that the board blinks with this sequence: 1 second on, 1 off, 2 on, 1 off, 3 on, 1 off, repeat.

```
void dimmer(int freq, int duty){
  int period, onTime, offTime;
  period = 1000 / freq;            // delay() is in milliseconds
  onTime = period * duty / 100;      // how to pretend duty is a fraction
  offTime = period - onTime;        // all time is accounted for (none lost to rounding)
  digitalWrite(LED_BUILTIN, HIGH);   // turn the LED on (HIGH is the voltage level)
  delay(onTime);                              // wait for the interval
  digitalWrite(LED_BUILTIN, LOW);    // turn the LED off by making the voltage LOW
  delay(offTime);                             // wait for a 10ms
}
```

5. Create another sketch like in step 4. Add it to your git repo. Use the code above to make the LED fade in and out. Use a loop of count up and down that calls the dimmer code every iteration with a new duty value. You do not need to change my code, just call dimmer from your loop. Now, reduce the frequency (currently 100Hz) in 10Hz increments (Now you need to change my code). Observe the unwanted behavior. *What*

is going wrong? Brainstorm some solutions. Dimmers exist in the real world. What is their solution?

6. When you're done, push your branch to the github repo. Hint: git add *, git commit -am "your message here", git push --set-upstream origin <your branch name>

Deliverables:

1. Questions - answer these in your git repo readme:
    a. Why do embedded systems need a setup and a loop?
    b. Why does our code need to be compiled?
    c. When lowering the frequency in step four, what is going wrong? Brainstorm some solutions. Dimmers exist in the real world. What is their solution?
2. Two Arduino Sketches
    a. Part 3's blinking sequence
    b. Part 4 with the fading dimmer code