Introduction to Arduino
Programming, Wiring, Sensors
University of MN -- Twin Cities
Aerospace Engineering and Mechanics Department
Goals of this lesson

- Learn what microcontrollers are and what they can do.
- Learn to wire a solderless breadboard.
- Learn to open, interpret and interact with Arduino programs.
- Learn to control electronic components such as LEDs and motors and write information to the serial monitor.
- Learn to expand Arduino functionality with shields.
- Put you in a position to implement your own Arduino programs.
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 - components

Figure 1: The Arduino Uno R3
Solderless Breadboard

Figure 13: Top and bottom of breadboard
Solderless breadboard interior

Figure 14: Breadboard bottom with clips pulled out
Proto shield and “tiny” breadboard

- We will not use the proto shield in this demonstration but they can be seen in the kit.

- Notice that the “tiny” solderless breadboard (ours are white) has no power rails and very limited real-estate. Use it carefully!
Intro to software


- Arduino programs or “sketches” are written in a programming language that is essentially an extension of C++.

- Every sketch must have a `setup()` function, which is executed once, followed by a `loop()` function that is executed repeatedly until power is cut. (For those familiar with C++, these replace the `main()` function.)

- It is recommended that you add comments to this code to make it easier to understand in the future or for others reading your code. This can be done by adding `//` after any line of code and writing your commentary.

- Many sensors come with prewritten software - look online for sample code, libraries (of functions), and tutorials.
Parts of the IDE main screen

- 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

Text area for writing/editing sketches.

Error messages and other feedback show up here.
Bare minimum - sketch organization

```cpp
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
}
```
Practice Sketches

- The following activities use several pre-built sketches.

- To get them, go to [https://github.com/MNSGC-Ballooning/Arduino-Training](https://github.com/MNSGC-Ballooning/Arduino-Training)

- Click the green “Clone or Download” button on the right-hand side, then click “Download ZIP”

- Extract the contents to any folder you like. If unsure, “.../Documents/Arduino” is a good choice. Just remember where the sketches are saved.
Activity 1 - making an on-board and external LED blink

- Paying attention to the connected rows and rails on the breadboard, place an LED on the breadboard. (be careful not to put both legs into a single connected rail.

- 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.
External LED circuit

Your circuit should look similar to this, but with the specified pins. This diagram is called a “breadboard view”
Circuit diagram (aka schematic)

You can also use a more technical circuit representation, called a circuit diagram or schematic.
Note the utility of having both 0V (AKA GND) and 5V “rails”. It is important to note that the tiny breadboards do not have rails.
Activity 1 - continuation

- 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 COM 3)

- Load the “BasicBlink” sketch from the Arduino-Training folder downloaded earlier

- Look at this 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 - the sketch will stay in the Uno memory until overwritten or erased.

- Digital HIGH is 5V and LOW is 0V (gnd)

- Analyze the code, see if you can connect each line with its function. Alter the blinking speed of the LED from 1000 milliseconds to 200 milliseconds.
Activity 1A - write some code

- Using Activity 1 as a guide, now wire an additional LED/resistor pair to another digital pin on the Arduino.

- Open a new sketch, and find a way to make the two LEDs blink out of phase (ie when one turns on, the other should turn off). Note: you will have to save your sketch before you run it. Name it whatever you like.

- Now, modify your sketch to make second LED blink twice as fast as the first. Hint: you’ll have to think carefully about how you use the delay() function.
About Motors

- There are several types of motors:
  - **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 degrees) in precise degree increments. Uses Servo library in Arduino. They have 3+ pins and are controlled by pulse-width modulation ("~" pins)
  - **Continuous rotation servo** - can go all the way around continuously. Interprets PWM value as speed & direction.
  - **Stepper Motors** - Servo capable of full rotation in small steps. They use the Stepper library in Arduino and have 3+ pins.
Activity 2: Using a Servo

- With the Arduino Uno unplugged, wire the servo according to the chart on the next slide.

- Load the Sweep sketch

- 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 with servos outside this kit, some use a lot of power and may need an external power source.

- As can be seen from the comments, your servo is initially running from 0 to 180 degrees in steps of 1 degree, each with a duration of 15ms. Try to change this code so that the servo moves between 0 and 90 degrees with 5 degree steps that have a duration of 50ms.
The servo has a plug attached to its ribbon cable. This is so that you can extend the cable using plugs and/or another cable. It also allows it to plug into specific plugs built into some shields as can be seen on the next slide.

<table>
<thead>
<tr>
<th>Pin</th>
<th>Description</th>
<th>Attached To</th>
</tr>
</thead>
<tbody>
<tr>
<td>Brown</td>
<td>Ground</td>
<td>GND</td>
</tr>
<tr>
<td>Red</td>
<td>Power (5V)</td>
<td>5V</td>
</tr>
<tr>
<td>Orange</td>
<td>Control Lead</td>
<td>D3</td>
</tr>
</tbody>
</table>
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 as we do in the previous exercise, where the servo sweeps between angles in incremental small steps.

- Another way may be to set the servo to just “open” (go to a specified angle) and “close” (go to a different specified angle). This might be easier to code than the “sweep” motion.
Activity 2A: Defining Functions

- Like any other language, Arduino allows the user to write their own functions, enabling large blocks of code to be reused easily.

- Also allow basic actions in a program to be easily labelled, like opening a servo.

- Open a new sketch, and copy the everything from Sweep up to the loop.

- Define two new functions after the loop: void open() and void close().

- Have the open function set the servo to 135 degrees, and close to 45.

- Now, make the servo switch between open and closed in the main loop, pausing for a second in each state.
Notes about Libraries

- Libraries take large amounts of code and refine it into a few simple commands that we can use conveniently. Ex: `myservo.write()` will set the servo to a position.

- This `myservo.write()` function actually takes up a lot of code, but by putting that function in a library we can ignore this basic code and focus on other useful commands and functions.

- They usually end in a `.h` (header file)

- Libraries can be used in our programs with the `#include “Filename.h”` command.

- Almost any component that is not analog (digital sensors, servos, etc) uses a library of some sort.

- You can install more libraries into your IDE. Look under Sketch: Import Library to
A few words on sensors

- The rest of the lesson will focus on how to use various sensors to make measurements and how to record those values onto an SD card.

- Sensors are broken into two major categories: analog and digital
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, and collected data is easily interfered with by other electronics (noise in voltage)
- Readings can be influenced by cable length - *keep connection cables short!*
Typical wiring of a powered analog sensor
Activity 3 - analog (air) pressure sensor

- With the Arduino unpowered, wire the 0-15 psi pressure sensor as follows.

- Load the sketch AnalogPressure; study code then run, test, and discuss.

- You’ll need to open the serial monitor to view the data. Set the baud rate to 115200.

<table>
<thead>
<tr>
<th>Pin</th>
<th>Function</th>
<th>Attached To</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Nothing</td>
<td>Not connected</td>
</tr>
<tr>
<td>2</td>
<td>5V (Power)</td>
<td>5V pin</td>
</tr>
<tr>
<td>3</td>
<td>Analog Output</td>
<td>Pin A0</td>
</tr>
<tr>
<td>4</td>
<td>Ground</td>
<td>GND pin</td>
</tr>
</tbody>
</table>

- This 4-pin pressure sensor is the Honeywell SSCSANN015PAAA5 and is called a SIP-AN package.
Activity 4 - analog relative humidity sensor

- With the Arduino Uno unpowered, wire the HIH4030 relative humidity sensors using the table below (leave other sensors in place)

- Load the sketch AnalogHumidity; study code, then run, test, and discuss.

<table>
<thead>
<tr>
<th>Pin</th>
<th>Function</th>
<th>Attached To</th>
</tr>
</thead>
<tbody>
<tr>
<td>GND</td>
<td>Ground</td>
<td>GND pin</td>
</tr>
<tr>
<td>OUT</td>
<td>Analog Data Output</td>
<td>Pin A1</td>
</tr>
<tr>
<td>5V</td>
<td>5V (Power)</td>
<td>5V pin</td>
</tr>
</tbody>
</table>

- 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.
Writing to the Serial Monitor

- Before setup in the coding window, the variable being recorded in a serial monitor (or SD card, if used) 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)`. **Important:** 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 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 opened.)
Activity 4A - Multiple Analog Sensors

- Currently, both the Pressure and Humidity sensors should still be plugged into the Arduino. If not, rewire them as described in Activities 3 and 4.

- Open the sketches for each activity, as well as a new sketch.

- Using the previous two sketches as a guide, attempt to write a sketch that reads and prints data from both sensors once every second.

- Hint: think carefully about which lines of code are necessary for each sensor individually, and which are needed only once for the sketch as a whole.
Digital Sensors

- Usually use digital pins (some analog pins can also be used as digital).

- Can use more advanced forms of communication to let multiple sensors share the same 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 and efficiently.
Digital Sensors - continued

- Digital sensors are powered with either 5V or 3.3V

- 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.

- Be careful to watch for pin conflicts and voltage discrepancies when using microcontrollers. If you aren’t careful you can get bad data, fry components, or possibly damage the microcontroller.

- Watch out for components that look similar! Be sure to verify that you are always using the intended sensor.
Libraries for Activities 5 onward

- The following activities use libraries that are NOT included in the standard Arduino download.
- If any #include statement causes an error of “No such file or directory”, that means the library in question has not been downloaded.
- To correct this, any missing libraries will need to be added manually.
- Use the following GitHub links to find the necessary libraries:
  - ADXL345: https://github.com/sparkfun/SparkFun_ADXL345_Arduino_LIBRARY
  - MPL3115A2: https://github.com/sparkfun/MPL3115A2_Breakout
  - MAG3110: https://github.com/sparkfun/SparkFun_MAG3110_Breakout_Board_Arduino_Library
  - OneWire: https://github.com/PaulStoffregen/OneWire
  - DallasTemperature: https://github.com/milesburton/Arduino-Temperature-Control-Library
  - TinyGPS: https://github.com/mikalhart/TinyGPS
- For each, click on the green “Clone or download” button, then click “Download ZIP”
- In the Arduino IDE, go to the “Sketch” dropdown menu, select “Include Library”, “Add .ZIP Library…”, and select the downloaded zip folder to add it to the IDE.
- This extracts the contents to the .../Documents/Arduino/libraries folder, so the zip can be safely deleted once this is done.
Activity 5 - 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 pins D10-D13 for SPI 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).

- Check the baud rate on the Serial monitor

- Plug in the Arduino Uno and load DigitalAccel

- Study the code (notice how compact it is thanks to the library); run it (open serial monitor); test all 3 axes by moving the sensor around, discuss.
# Digital 3-Axis Accelerometer (breakout board ADXL345)

<table>
<thead>
<tr>
<th>Pin</th>
<th>Description</th>
<th>Attached To</th>
</tr>
</thead>
<tbody>
<tr>
<td>GND</td>
<td>Ground</td>
<td>GND</td>
</tr>
<tr>
<td>VCC</td>
<td>Power input (3.3V)</td>
<td>3.3V</td>
</tr>
<tr>
<td>CS</td>
<td>Chip Select</td>
<td>D10</td>
</tr>
<tr>
<td>INT 1</td>
<td>Interrupt 1 Output</td>
<td>N/A</td>
</tr>
<tr>
<td>INT 2</td>
<td>Interrupt 2 Output</td>
<td>N/A</td>
</tr>
<tr>
<td>SDO</td>
<td>Serial Data Output (SC0)</td>
<td>D12</td>
</tr>
<tr>
<td>SDA</td>
<td>Serial Data (SDA)</td>
<td>D11</td>
</tr>
<tr>
<td>SCL</td>
<td>Serial Communications Clock (SCL)</td>
<td>D13</td>
</tr>
</tbody>
</table>
Activity 6 - 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 V I²C device.

<table>
<thead>
<tr>
<th>Pin</th>
<th>Description</th>
<th>Attached To</th>
</tr>
</thead>
<tbody>
<tr>
<td>INT2</td>
<td>Pressure interrupt 2</td>
<td>Not connected</td>
</tr>
<tr>
<td>INT1</td>
<td>Pressure interrupt 1</td>
<td>Not connected</td>
</tr>
<tr>
<td>SDA</td>
<td>Serial Data</td>
<td>A4</td>
</tr>
<tr>
<td>SCL</td>
<td>Serial Communications (Clock)</td>
<td>A5</td>
</tr>
<tr>
<td>VCC</td>
<td>Power input (3.3V)</td>
<td>3.3V</td>
</tr>
<tr>
<td>GND</td>
<td>Ground</td>
<td>GND</td>
</tr>
</tbody>
</table>

- Look for pin conflicts, Hopefully there won’t be any (but always look).
- Plug in the Arduino Uno and load the sketch DigitalPressure.
- Study the code. Run it in the serial monitor, test, and discuss.
Activity 7 - 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 into an unpowered arduino Uno. All legs must go straight into the header - do not bend any.

- Wire an LED using the tiny breadboard:
  - Positive (long) leg wired to a safety resistor with the other end of the resistor connected to the digital 5 pin.
  - Negative (short) leg wired to ground (GND)
  - (This setup has the resistor on the opposite leg from previously. It is important to get the orientation of 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

<table>
<thead>
<tr>
<th>Pin</th>
<th>Description</th>
<th>Attached To</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>5V (Power)</td>
<td>5V pin</td>
</tr>
<tr>
<td>2</td>
<td>Analog Output</td>
<td>Pin A2</td>
</tr>
<tr>
<td>3</td>
<td>Ground</td>
<td>GND pin</td>
</tr>
</tbody>
</table>

- This 3-pin sensor has TMP written on it. This is called a T0-92 package.

- Check the orientation and make sure the flat side of the sensor has “TMP” written on it. There are different sensors that look exactly alike.
With the arduino unplugged, add the digital 3-Axis magnetometer MAG 3110 to your breadboard using the table to the left.

- A magnetometer can be combined with an accelerometer and gyro to make an IMU (Inertial measurement Unit). This is 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 Data

- Open the sketch TempMagLogger.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.
  - You must include a special piece of code to write a new file each time the Arduino is powered. Without this, it would add data to the same file every time, making it nearly impossible to tell which data was taken in a given run
  - Pins can change depending on the SD card reader you use. The Sparkfun Micro-SD shield uses D8 and D11. The Adafruit full size SD card shield uses D10, not D8 and D11.
Datalogging (to SD card)

- Load the sketch TempMagLogger.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.

- Analyze what you can see in the serial window, what can you make of it?

- The code that you are using is logged onto the SD card in a new document every time the program is restarted. To view this data, put the card into the computer and open the latest document to view this data.
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) which 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 (5V) and LOW (0V).

- “Fritzing” software (free download) for drawing circuits: “breadboard view” (fairly realistic) vs “circuit diagram” view (AKA “schematic”)
Activity 8 - 2 digital temp sensors, SD card pt. 1

- With the Arduino Uno unplugged, add one Dallas one-wire digital temp sensor on the breadboard and a second one on the end of a 3-wire cable (if not provided then make your own out of jumper wires as shown on the following slide).

- Notice that both sensors need ground and power (5V) and that they are polled through a single digital pin (D2). Also notice that a 4.7 Ohm “pull-up” resistor is required between pin D2 and the 5V line so that it stays HIGH when not in use.

- Don’t forget to download the libraries for these sensors - you’ll need both OneWire and DallasTemperature
Wiring for 2 digital temp sensors

- It is not shown, but the tiny breadboard may actually be mounted on a protoshield on top of the Arduino Uno.

- Use this wiring color convention: red for power, black for ground, green for data.
Activity 8 - 2 digital temp sensors, SD card pt. 2

- Put an SD card into your SD card shield.

- Be sure the code has the correct #define line uncommented - 8 if using the Sparkfun shield, 10 if using Adafruit

- Plug in the Arduino Uno and load the sketch SDTempBus, 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.
Activity 8 - 2 digital temp sensors, SD card pt. 3

- 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 temp - also try removing the SD card afterward 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 analog cable lengths and use digital sensors when off-board (if possible).
What could be useful...

- You may not remember when/where it was taken so it is useful to add a timestamp and/or a GPS location stamp to data as it is saved.

- An RTC (real time clock) comes in certain SD shields or breakout boards - you can use this to put a timestamp on data.

- GPS can also come as a separate shield or a breakout board on its own - you can use this to put a location stamp on data.

- Remember: measurements are only useful with context. Some method of identifying where or when each was taken is critical.
Activity 9 - GPS, external power pt. 1

- With the Arduino unplugged, add the GPS module to the upper right hand corner of your breadboard using the table on the next slide.

- Put a red LED on pin 3 and a green LED (it might be faint) on pin 4. Use safety resistors on both, of course.

- Plug in the USB cable and load the sketch GPSLogger. Upload the sketch to the Arduino Uno (watch out for chip select and baud rate!). Open the serial monitor and watch it run. Notice the red LED gives error states (see code) and the green LED flashes every time the unit tries to log a data point.

- Count the red LED flashes (if any) when you first power up. Error messages are also written to the serial monitor. The GPS probably won’t get a lock indoors, so the data might be boring.
# Adafruit Ultimate GPS Breakout

<table>
<thead>
<tr>
<th>Pin</th>
<th>Function</th>
<th>Attached To</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.3V</td>
<td>Provides a 3.3V Output</td>
<td>Not connected</td>
</tr>
<tr>
<td>EN</td>
<td>Enables Sleep Mode</td>
<td>Not connected</td>
</tr>
<tr>
<td>VBAT</td>
<td>Allows battery input for RTC</td>
<td>Not connected</td>
</tr>
<tr>
<td>FIX</td>
<td>Output at same time as fix LED</td>
<td>Not connected (May want to connect depending on your project)</td>
</tr>
<tr>
<td>TX</td>
<td>Transmit</td>
<td>D5</td>
</tr>
<tr>
<td>RX</td>
<td>Receive</td>
<td>D6</td>
</tr>
<tr>
<td>GND</td>
<td>Ground</td>
<td>GND</td>
</tr>
<tr>
<td>VIN</td>
<td>5V Power</td>
<td>5V</td>
</tr>
<tr>
<td>PPS</td>
<td>Pulse per second output</td>
<td>Not connected</td>
</tr>
</tbody>
</table>
Activity 9 - GPS, external power pt. 2

- Unplug the USB cable (AKA the programming cable). Power the Arduino Uno using a 9V battery through the power jack instead. You can now run sketches independently from the laptop. Now that you do not have access to the serial monitor, the red LED is far more important. Laptop-free operation is critical for applications like ballooning.

- The LED on the GPS itself 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 for a bit (100’s of meters) to collect some interesting data. Come back in and look at the GPS data that is logged to the memory card.
Talking to a payload in flight

- Sometimes it is useful to get data from a payload in real time
- Also useful to give “commands” to payload to control its operation
- Normally this requires the Serial Monitor, but a radio can also be used
- XBees (short range radio modules) are often added for this purpose - allow one device to communicate with another computer or microcontroller nearby (which can in turn relay the data elsewhere)
Activity 10: xBee - pt 1

- This activity requires two Arduinos - pair up with another kit.

- You will also need two xBees: one from either kit, and one “relay” xBee from us - we have fewer of these.

- Make sure both xBees are plugged into breakout boards, then wire the breakouts to the Arduinos using the table on the next slide.

- Wire an LED+resistor to D13 on the Arduino without the relay (the “payload”).

- Load the sketch XBeeBlink onto that Arduino. Notice that the LED will not blink, as the sketch is waiting for an xBee to trigger the flash.

- Disconnect the Arduino from the computer and power it with a 9V battery. Set
XBee breakout wiring

IMPORTANT: make sure the xBee is correctly plugged into the breakout - a pair of white lines at an angle show where the cut corners of the xBee go.

<table>
<thead>
<tr>
<th>Breakout Pin</th>
<th>Function</th>
<th>Attached to</th>
</tr>
</thead>
<tbody>
<tr>
<td>5V</td>
<td>Power</td>
<td>5V</td>
</tr>
<tr>
<td>GND</td>
<td>Ground</td>
<td>GND</td>
</tr>
<tr>
<td>Dout</td>
<td>Data from xBee</td>
<td>D2</td>
</tr>
<tr>
<td>Din</td>
<td>Data to xBee</td>
<td>D3</td>
</tr>
</tbody>
</table>
Activity 10: xBee - pt 2

- To activate the second Arduino as a relay system, ground pin 7 (ie connect D7 directly to GND)

- Reset the Arduino and open the Serial monitor (check baud rate). The phrase “Relay Mode active” should appear. Any data the relay receives from the computer will be relayed to the xBee and vice versa.

- To send data, type into the box above the monitor and press enter.

- The payload will respond to the following four messages (without quotes):
  - “FLIP”: toggles the LED on and off
  - “BLn”: makes the LED blink $n$ times, where $n$ is a positive integer less than 256. Note that each blink takes a full second, and cannot be interrupted...
Next Steps

- This has been only 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

  - (Mostly) complete documentation for arduino language concepts, functions, and common libraries. Very useful for solving general code issues.

- [https://learn.adafruit.com](https://learn.adafruit.com)
  - Adafruit makes many shields and sensors, and they have tutorials for almost everything they carry.

  - Arduino Classroom is currently doing an intro series on Arduinos. Check it for updates and more topics in the future

- [http://playground.arduino.cc/](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.
References

1. *Beginning Arduino* by Michael McRoberts, 2\textsuperscript{nd} edition