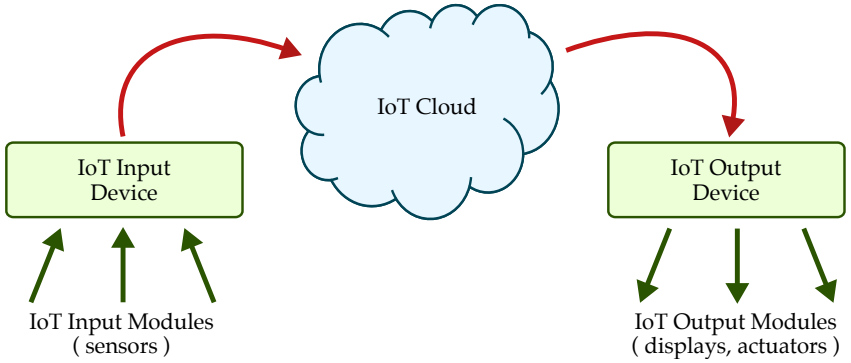
CURIE Academy, Summer 2014
Design Project Input/Output Module Descriptions

Christopher Torng
School of Electrical and Computer Engineering
Cornell University

Each IoT design project will involve building an IoT system comprised of an IoT input device, IoT cloud, and IoT output device. The IoT input devices will have various input modules attached that can sense what is going on in the environment (e.g., light, temperature, motion, force, current, pulse, liquid) and be able to upload data into the cloud. Each IoT output device will be able to download data from the could and will have various output modules attached to display data (e.g., LEDs, piezo buzzer, mini-printer) or react in some way (e.g., servo, relay for controlling appliances). The following diagram illustrates the overall approach we will be using in our IoT systems.

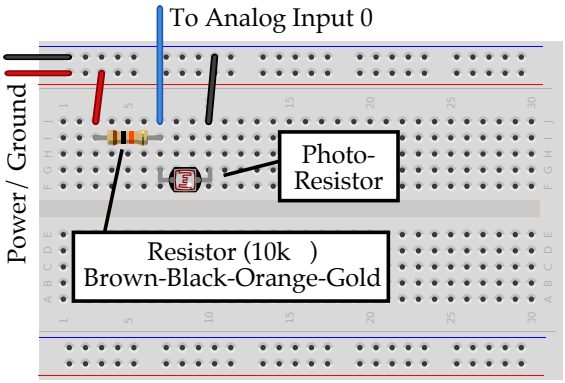


The rest of this document describes each input/output module and gives a simple circuit along with the corresponding code to get you started. You will need to work on your own to determine the best way to integrate the input/output module with the IoT cloud. You can use Lab 3 as an example. The various input and output modules are listed below.

1. Light Input Module

This input module enables your IoT device to sense how much light is reaching the device. The module uses a small photo-resistor which is similar in spirit to the potentiometer you experimented with in Lab 2. For the potentiometer, the resistance varied as you twisted the knob on the robot. For the photo-resistor, the resistance will vary based on how much light reaches the device. The Arduino cannot directly sense resistance, but it can sense an analog input voltage. We will use a simple circuit called a voltage divider so that the voltage across the photo-resistor is proportional to the resistance.

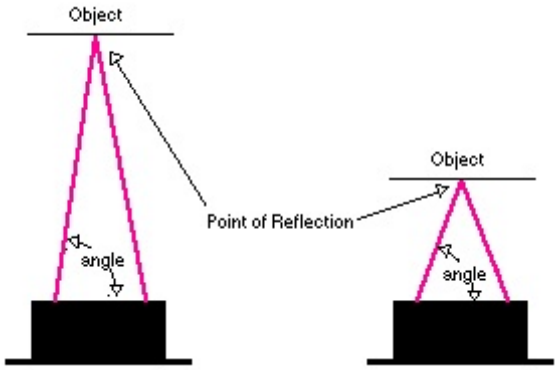
A sample circuit and Arduino code is shown below to get you started. The circuit places a 10 kΩ resistor in series with the photo-resistor so that voltage is divided between the two components. A 10 kΩ resistor has brown-black-orange bands. We use a wire to connect the node in between the resistor and the photo-resistor to an analog input of the Arduino so that the Arduino can read the voltage from one end of the 10 kΩ resistor to ground. The example code will print the analog reading from the photo-resistor on the serial monitor, similar to how we printed the analog reading from the grayscale sensor in Lab 2. After setting up the circuit and programming the Arduino, open the serial monitor and note how the sensor reading varies when the device is covered with your hand or when the device is placed under a bright light. The reading from the sensor should get larger when the device is in a dark environment, and should get smaller when the device is in a bright environment.

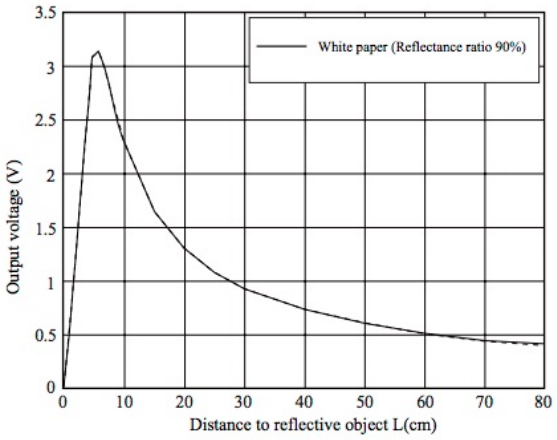



```
1  int pin_light = A0;
2
3  void setup() {
4    Serial.begin(9600);
5  }
6
7  void loop() {
8    int light_level = analogRead( pin_light );
9    Serial.println( light_level );
10   delay(1000);
11 }
```

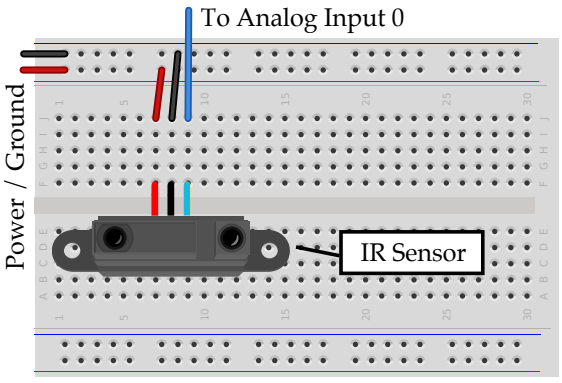
2. Infrared (IR) Input Module

This input module enables your IoT device to sense the distance to an object placed 10–80 cm in front of it, using the same Sharp GP2Y0A21 Distance Sensor you possibly experimented with in Lab 2. The sensor works by bouncing infrared (IR) light off objects and sensing the angle at which it returns – a process called *triangulation*. The figure below to the left shows IR light bounced off an object placed at two different distances. Notice how knowing the angle of reflected light tells you the object's distance. The output of the sensor is a voltage, corresponding to the distance derived from the angle of reflected light. Moving an object to different distances changes the output voltage as shown in the figure below to the right. Can you see why the range of this device is chosen to be 10–80 cm?





A sample circuit and Arduino code is shown below to get you started. There is no need for any extra components; we directly connect the blue wire from the IR sensor to an analog input on the Arduino. The example code will print the analog reading from the IR sensor on the serial monitor, similar to how we printed the analog reading from the grayscale sensor in Lab 2. After setting up the circuit and programming the Arduino, open the serial monitor and experiment with placing objects (e.g., a book) at various distances away from the front of the sensor. The reading from the sensor should be larger for objects farther away from the sensor and smaller for objects closer to the sensor.




```
int pin_ir = A0;

void setup() {
  Serial.begin(9600);
  pinMode( pin_ir, INPUT );
}

void loop() {
  int distance = analogRead( pin_ir );
  Serial.println( distance );
  delay(1000);
}
```

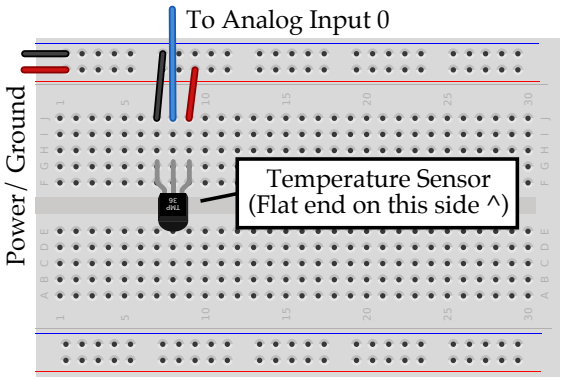
Questions: (1) What happens if the object is further than 80 cm (max range) from the sensor? (2) What happens if the object is closer than 10 cm (min range) from the sensor? (3) The sensor is not outputting distance in any particular unit. How can we convert the readings into centimeters?

3. Temperature Input Module

This input module enables your IoT device to sense the temperature of the environment using an easy-to-use temperature sensor. This may be useful in environmental control systems, thermal protection, industrial process control, fire alarms, power system monitors, cpu thermal management, etc. The TMP36 temperature sensor contains an integrated circuit with a single signal output lead. This signal's voltage rises and falls, depending on the ambient temperature, at 10 mV per degree centigrade (Celsius).

A sample circuit and Arduino code is shown below to get you started. There is no need for any extra components; we directly connect the middle lead on the temperature sensor to an analog input on the Arduino. Make sure the sensor's "flat end" is facing the correct direction (shown in the diagram). If you point the flat end of the sensor to your left, the top lead connects to power (the red wire), the middle lead is the signal, and the bottom lead connects to ground (the black wire).

Since the analog reading from the temperature sensor is a digital value between 0 and 1023, the example code scales this value to a range between 0–5 V, converts voltage to temperature, and then prints the temperature in centigrade (Celsius) on the serial monitor, similar to how we printed the analog reading from the grayscale sensor in Lab 2. After setting up the circuit and programming the Arduino, open the serial monitor and check the reported room temperature in degrees centigrade (Celsius). Does the reading sound right? Experiment with the sensor by changing the ambient temperature (e.g., with a hair dryer) and see how warm or cool you can make it!

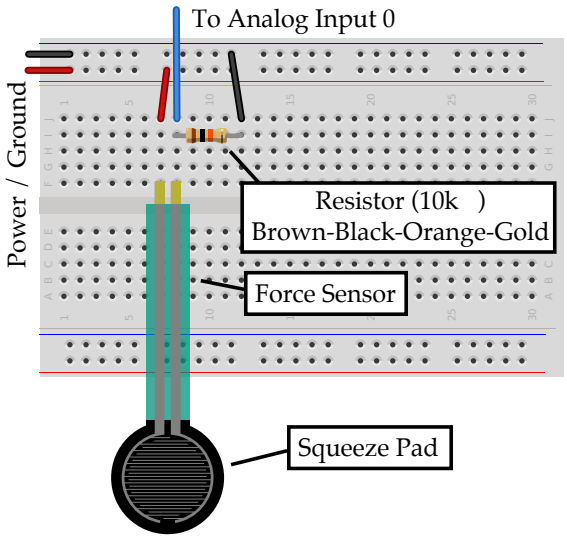



```
int pin_temperature = A0;

void setup() {
  Serial.begin(9600);
  pinMode( pin_temperature, INPUT );
}

void loop() {

  // Read temperature sensor and convert from
  // a 0 to 1023 digital range to 0 to 5 volts.

  float voltage =
    analogRead( pin_temperature ) * .004882814;

  // Convert from 10 mV per degree with 500
  // mV offset to degrees Celsius (i.e.,
  // temperature = (voltage-500mV) * 100).

  float temperature = (voltage - .5) * 100;

  Serial.println( temperature );
  delay(1000);
}
```

4. Force Input Module

This input module enables your IoT device to sense the amount of force exerted on the surface of a squeeze pad. This may be useful in medical devices, physical therapy, stress relief technology, virtual motion, computer peripherals, touch-sensitive devices, etc. The sensor is a variable resistor whose resistance depends on how much force is exerted on the squeeze pad (e.g., squeezing it between two fingers); the resistance is high (infinite) when there is no force and low when the force is high. The Arduino cannot directly sense resistance, but it can sense an analog input voltage. We will use a simple circuit called a voltage divider so that the voltage across the force sensor is proportional to the resistance.

A sample circuit and Arduino code is shown below to get you started. The circuit places a 10 kΩ resistor in series with the force sensor so that voltage is divided between the two components. A 10 kΩ resistor has brown-black-orange bands. We use a wire to connect the node in between the resistor and the force sensor to an analog input of the Arduino so that the Arduino can read the voltage from one end of the 10 kΩ resistor to ground. The example code will print the analog reading from the force sensor on the serial monitor, similar to how we printed the analog reading from the grayscale sensor in Lab 2. After setting up the circuit and programming the Arduino, open the serial monitor and check how much force the sensor is detecting. Then try squeezing the squeeze pad between two fingers. Does the reading increase? Can you explain this behavior?

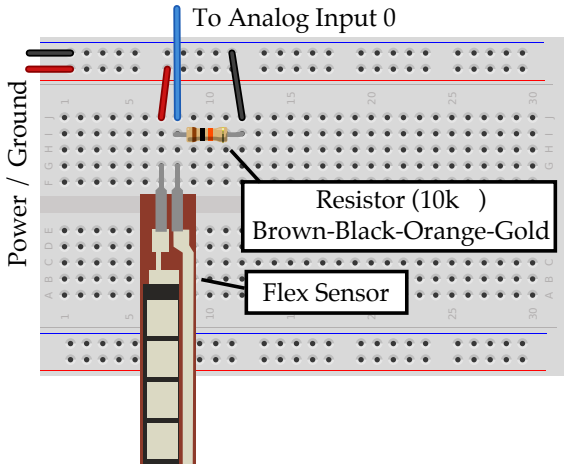



```
1  int pin_force = A0;
2
3  void setup() {
4    Serial.begin(9600);
5    pinMode( pin_force, INPUT );
6  }
7
8  void loop() {
9    int force = analogRead( pin_force );
10
11   Serial.println( force );
12   delay(1000);
13 }
```

5. Flex Input Module

This input module enables your IoT device to sense the flexing of a 6 cm strip. This may be useful in medical devices, physical therapy, stress relief technology, virtual motion, computer peripherals, etc. The flex sensor is a variable resistor whose resistance depends on how much the strip is flexed (i.e., by bending it); the resistance is higher when flexed in one direction and lower when flexed in the other. The Arduino cannot directly sense resistance, but it can sense an analog input voltage. We will use a simple circuit called a voltage divider so that the voltage across the flex sensor is proportional to the resistance.

A sample circuit and Arduino code is shown below to get you started. The circuit places a 10 kΩ resistor in series with the flex sensor so that voltage is divided between the two components. A 10 kΩ resistor has brown-black-orange bands. We use a wire to connect the node in between the resistor and the flex sensor to an analog input of the Arduino so that the Arduino can read the voltage from one end of the 10 kΩ resistor to ground. The example code will print the analog reading from the flex sensor on the serial monitor, similar to how we printed the analog reading from the grayscale sensor in Lab 2. After setting up the circuit and programming the Arduino, open the serial monitor and check what the sensor is reporting regarding how much the strip is flexed. Try flexing the sensor strip in one direction. Does the reading increase or decrease? Try flexing the sensor strip in the other direction. Can you explain this behavior?

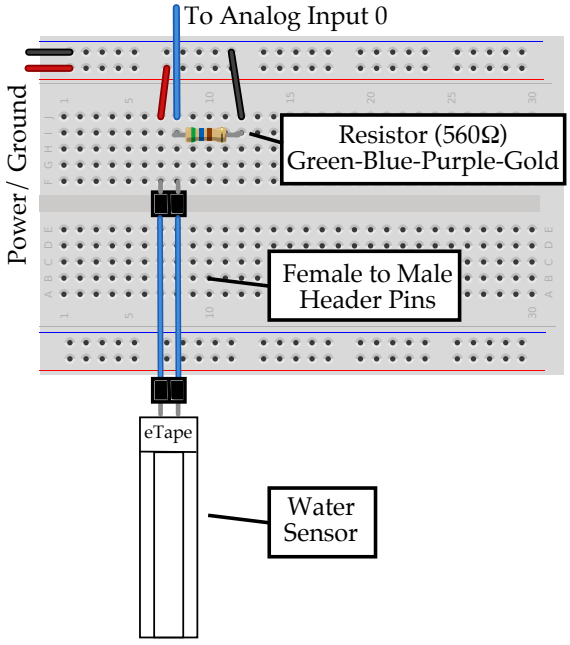



```
int pin_flex = A0;

void setup() {
  Serial.begin(9600);
  pinMode( pin_flex, INPUT );
}

void loop() {
  int flex = analogRead( pin_flex );

  Serial.println( flex );
  delay(1000);
}
```

6. Water Input Module

This input module enables your IoT device to sense the level of water in the environment using a water sensor called eTape. Water level sensing can play an important role in environmental monitoring systems, disaster warning, cooling systems, water purification, medical equipment care, etc. This tape can easily be secured to a street pole, hung on the wall of a room, or attached to the side of a water container or other similar environments. The tape is a variable resistor. When immersed in water, the sensor tape is compressed from both sides by hydrostatic pressure, and the tape's resistance varies as the water level rises and falls. The resistance and water level are inversely proportional: the lower the water level, the higher the output resistance; the higher the water level, the lower the output resistance. The Arduino cannot directly sense resistance, but it can sense an analog input voltage. We will use a simple circuit called a voltage divider so that the voltage across the water sensor is proportional to the resistance.

A sample circuit and Arduino code is shown below to get you started. The circuit places a 560 Ω resistor in series with the water sensor so that voltage is divided between the two components. A 560 Ω resistor has green-blue-purple bands. We use a wire to connect the node in between the resistor and the water sensor to an analog input of the Arduino so that the Arduino can read the voltage from one end of the 560 Ω resistor to ground. Use long wires to connect the water sensor to the breadboard so that you can freely move the sensor away from the Arduino and breadboard (which should not get wet!). The example code will print the analog reading from the water level sensor on the serial monitor, similar to how we printed the analog reading from the grayscale sensor in Lab 2. After setting up the circuit and programming the Arduino, open the serial monitor and check the value of the water level sensor when dipped in a container of water. Be careful not to get the top electrical leads of the sensor wet. Try dipping the water level sensor further into the water. What happens to the reading and why do you think this happens?

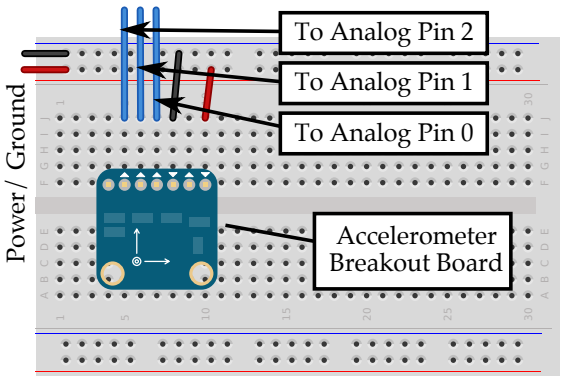



```
int pin_water = A0;

void setup() {
  Serial.begin(9600);
  pinMode( pin_water, INPUT );
}

void loop() {
  int water_level = analogRead( pin_water )

  Serial.println( water_level );
  delay(1000);
}
```

7. Accelerometer Input Module

This input module enables your IoT device to sense physical acceleration – perfect for applications requiring tilt-sensing, motion, shock, vibration, etc. The ADXL335 3-axis accelerometer you will use is a micro-electro-mechanical system (MEMS) device: a device that combines tightly-coupled electrical and mechanical systems on a very small scale (micro-meters!). The accelerometer works by measuring the capacitance between fixed plates and a tiny structure suspended in mid-air by tiny springs. As the device experiences acceleration in the X, Y, and/or Z axes, the suspended structure deflects, changing the capacitance along each axis. Each capacitance converts to an output voltage, proportional to the acceleration on that axis, which can then be sensed by the Arduino.

A sample circuit and Arduino code is shown below to get you started. The example code will print calibrated accelerometer readings for each axis on the serial monitor, similar to how we printed the analog reading from the grayscale sensor in Lab 2. After setting up the circuit and programming the Arduino, open the serial monitor, place your device on the table, and check the values the accelerometer is sensing. These should be close to (x = 0, y = 0, z = 100). Then pick up the device and tilt it or shake it. See how the values change along the three axes!




```
int pin_x_accel = A2;
int pin_y_accel = A1;
int pin_z_accel = A0;

// Calibration

int x_min = 267; int x_max = 399;
int y_min = 272; int y_max = 404;
int z_min = 274; int z_max = 412;

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

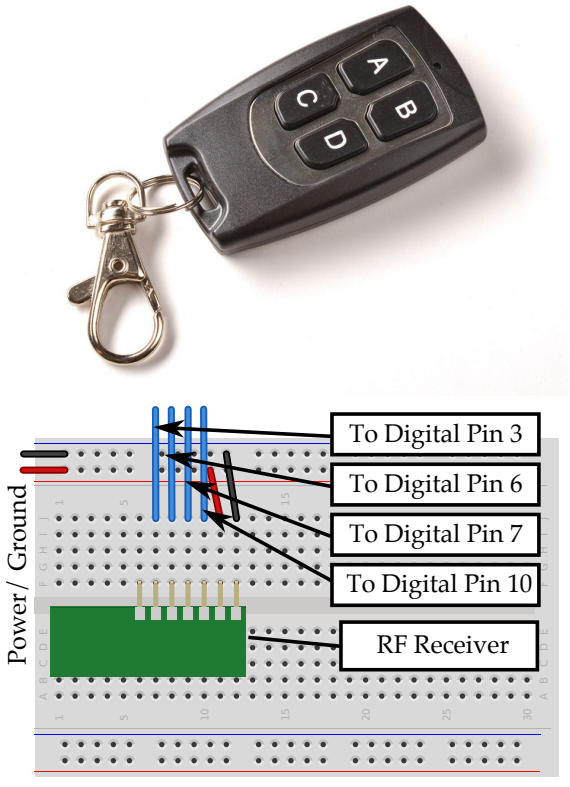
```
void loop() {
  // Read each axis.

  int x_raw = analogRead( pin_x_accel );
  int y_raw = analogRead( pin_y_accel );
  int z_raw = analogRead( pin_z_accel );

  // Scale measurements to calibrated minimum
  // and maximum.

  int x_scaled =
    map( x_raw, x_min, x_max, -100, 100 );
  int y_scaled =
    map( y_raw, y_min, y_max, -100, 100 );
  int z_scaled =
    map( z_raw, z_min, z_max, -100, 100 );

  // Report scaled measurements.

  Serial.print("  x: "); Serial.print( x_scaled );
  Serial.print("; y: "); Serial.print( y_scaled );
  Serial.print("; z: "); Serial.print( z_scaled );
  Serial.println();

  // Wait 500ms before next reading.

  delay(500);
}
```

8. RF Input Module

This input module enables your IoT device to send and receive radio frequency (RF) signals wirelessly in a 25 ft radius. RF control devices are useful in many applications – e.g., car security systems, home security and automation systems, garage door controllers, remote control fan, remote control toys, remote control industrial use. In an RF-based control system, an RF transmitter emits radio waves in the range of 3 kHz to 300 GHz, and an RF receiver receives the signals and acts on them. In this module, you will use a 4-button keyfob RF remote to send 315 MHz signals to an RF receiver connected to your Arduino, which can then act on the signal.

A sample circuit is shown below to get you started. The RF receiver module has seven pins. D0, D1, D2, and D3 are signal pins corresponding to the four buttons on the keyfob RF remote; VT is a special pin that is HIGH if any button is pushed and LOW if none are pushed. The 4-button keyfob RF remote requires no assembly or connectivity. Pressing any of the four buttons transmits a 315 MHz radio wave command that can be picked up by the RF receiver.

The example Arduino code shown below will read which button you have pressed on the remote, and then print the output on the serial monitor every 100 ms. After setting up the circuit and programming the Arduino, open the serial monitor and try pressing each button on the RF remote. You should see each letter detected by the Arduino displayed on the serial monitor. Try to see how far away you can move with the RF remote while still successfully receiving the signal!




```
int pin_RF_A = 3;
int pin_RF_B = 6;
int pin_RF_C = 7;
int pin_RF_D = 10;

void setup() {
  pinMode( pin_RF_A, INPUT );
  pinMode( pin_RF_B, INPUT );
  pinMode( pin_RF_C, INPUT );
  pinMode( pin_RF_D, INPUT );
  Serial.begin(9600);
}

void loop() {
  // Read RF input on each pin
  int A = digitalRead( pin_RF_A );
  int B = digitalRead( pin_RF_B );
  int C = digitalRead( pin_RF_C );
  int D = digitalRead( pin_RF_D );

  // Print the letter if the input is detected.
  if (A) Serial.print("A"); else Serial.print("-");
  if (B) Serial.print("B"); else Serial.print("-");
  if (C) Serial.print("C"); else Serial.print("-");
  if (D) Serial.print("D"); else Serial.print("-");
  Serial.println(); // make a new line

  // Loop every 100ms.
  delay(100);
}
```

9. Current Input Module

This input module enables your IoT device to read the AC voltage and current used to power lights, appliances, and everything else in your home. The emonTx Shield you will use is an open-source energy monitoring and control system suitable for domestic and industrial use. The emonTx will allow you to safely measure the voltage and current that a lamp, for example, uses when plugged into the wall.

You will be using an AC-to-AC adapter to measure voltage. Using a transformer, the AC-to-AC adapter reduces the wall voltage down by a factor of about 12, so 110 V becomes 9 V. If the wall voltage changes, you will be able to see that change, reduced by a factor of about 12. The current sensor is also a transformer. By wrapping a ferrite loop around a wire, you can detect the induced magnetic field from an AC current in the wire. This induced field creates a voltage proportional to the current, which we can measure using the Arduino. Power is current times voltage, or $P = IV$, so armed with these two measurements we can calculate the power consumed by a household device.

A sample circuit and Arduino code is shown below to get you started. The emonTx plugs directly into the Arduino, the current sensors plug into the 3.5 mm audio jacks, and the AC-to-AC connector goes into the barrel connector jack. We will share the voltage sensor across all of the current sensors (note that current sensor 4 is disabled, so don't use it). The sample code below will let you calculate power for the devices connected to current sensor 1.

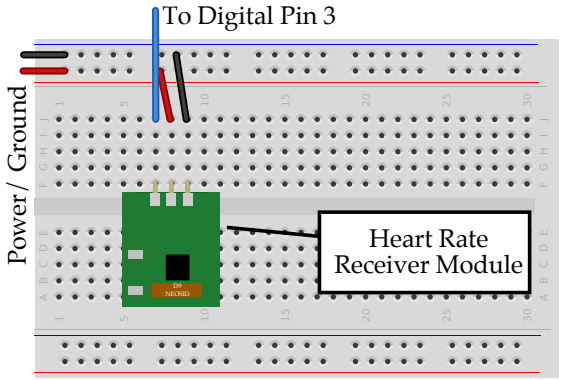
```
#include "EmonLib.h"
EnergyMonitor emon1;

// Include an offset to "zero" the power
float power_offset = 3.9;
```

```
void setup() {
  // Voltage: input pin, calibration, phase_shift
  emon1.voltage(0, 120.0, 1.7);

  // Current: input pin, calibration.
  emon1.current(1, 60.6);

  Serial.begin(9600);
}

void loop() {
  // Calculate all.
  // Params: num_half_wavelengths (crossings), time-out
  emon1.calcVI(20,2000);

  // Extract Real Power into variable
  float realPower = emon1.realPower - power_offset;

  // Extract Vrms into Variable
  float supplyVoltage = emon1.Vrms;

  // Floor the power at 0.
  if (realPower < 0.0)
    realPower = 0.0;

  Serial.print(realPower); Serial.print("W @ ");
  Serial.print(emon1.Vrms); Serial.println("V");
}
```

10. Heart Rate Input Module

This input module enables your IoT device to sense your heart beating. Heart rate monitors are useful for monitoring heart rate in a variety of applications, including exercise and fitness (targetting a healthy heart rate), injury rehabilitation, hospital monitoring systems, and biotelemetry. The Polar T34 Non-Coded Heart Rate Transmitter (the wearable chest strap) monitors and then wirelessly transmits your heart rate data from the chest strap to the Polar Heart Rate Receiver (the small green IC), which can then communicate data to the Arduino. The heart rate monitor works by detecting electrical heart muscle signals as the heart beats. The chest strap can sense these electric signals when the two contacts are moistened and are in direct contact with your skin, and it sends a pulsed wireless signal to the receiver when it detects a beat.

A sample circuit and Arduino code is shown below to get you started. There is no need for any extra components. The example code will print "Heartbeat" on the serial monitor whenever a heart beat is detected. After setting up the circuit and programming the Arduino, you can secure the chest strap to your partner's chest, just below the chest muscles, and buckle the strap together. Moisten the two grooved electrode areas on the back of the chest strap. Check that the wet grooved areas are firmly against the skin and that the word "Polar" is right-side up. The moistened grooved areas activate the transmitter, so make sure to dry the sensor when not in use. After securing the chest strap, open the serial monitor, use your hand to find your own pulse, and verify that "Heartbeat" is printed at the same time your heart is beating.



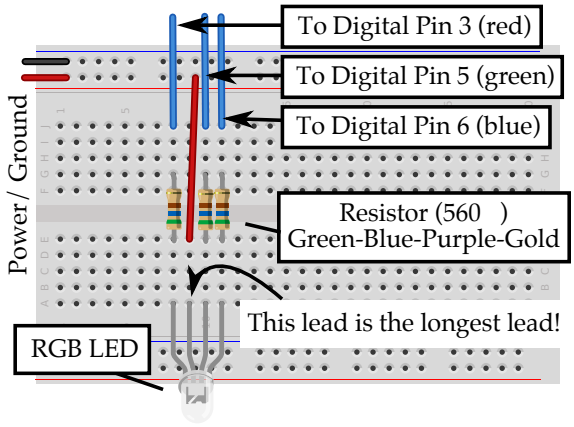

```
int pin_heart = 3;
byte old_sample, sample;

void setup() {
  Serial.begin(9600);
  pinMode (pin_heart, INPUT);

  Serial.println("Waiting for heart beat...");

  // Wait until a heart beat is detected
  while ( !digitalRead(pin_heart) ) {};
  Serial.println ("Heart beat detected!");
}

int heartbeat_count;

void loop() {
  sample = digitalRead(pin_heart);
  if (sample && (old_sample != sample)) {
    Serial.print("Heartbeat: ");
    Serial.println( heartbeat_count );
    heartbeat_count++;
  }
  old_sample = sample;
}

```

11. RGB LED Output Module

This output module enables your IoT device to display colored light using an RGB LED (three LEDs in a single housing), where RGB stands for 'red', 'green', and 'blue'. Using the RGB LED, we can generate any color we desire through color mixing. Do you still remember color wheels?

A sample circuit and Arduino code is shown below to get you started. Three leads of the RGB LED are set by digital output Arduino pins corresponding to 'red', 'green', and 'blue'; the longest lead of the RGB LED provides power. The example code will set the color of the RGB LED to violet. After setting up the circuit and programming the Arduino, check to make sure the RGB LED lights up violet. You can then experiment with other colors by choosing your own RGB values (the table below lists RGB values for common colors). You can also try to make the LED blink or change colors in time!

Color	RGB values
RED	(255, 0, 0)
ORANGE	(83, 4, 0)
YELLOW	(255, 255, 0)
GREEN	(0, 255, 0)
BLUE	(0, 0, 255)
INDIGO	(4, 0, 19)
VIOLET	(23, 0, 22)
CYAN	(0, 255, 255)
MAGENTA	(255, 0, 255)
WHITE	(255, 255, 255)
PINK	(158, 4, 79)

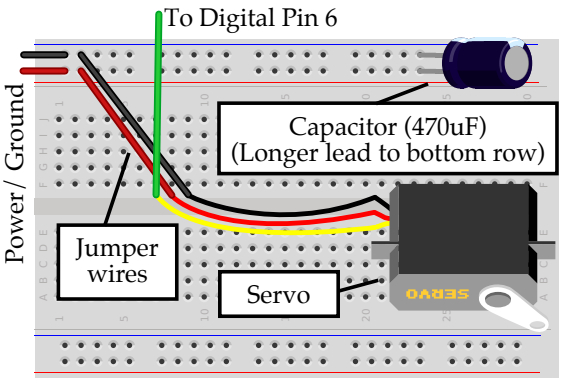



```
// Choose RGB LED pins (these must be PWM pins)
// redPin = 3, greenPin = 5, bluePin = 6
int pin_RGB[] = {3, 5, 6};

void setup() {
  for (int i = 0; i < 3; i++) {
    pinMode( pin_RGB[i], OUTPUT );
  }
}

void loop() {
  // Choose the RGB (red, green, blue) value
  // In this demo, we choose violet.

  byte my_color[] = {23, 0, 22};

  // Iterate through each of the three pins
  // (red, green blue).

  for (int i = 0; i < 3; i++) {

    // Write the analog value to the pin. We
    // use (255 - value) because our RGB LED
    // is common anode, which means the color
    // is fully on when we output
    // analogWrite( pin, 0 ) and fully off when
    // we output analogWrite( pin, 255).

    analogWrite( pin_RGB[i], 255 - my_color[i] );
  }
}
```

12. Servo Output Module

This output module enables your IoT device to control a high-torque standard servo. Servos are useful for interfacing directly with the environment; for example, servos can tilt, rotate, push other objects, or flip switches. Servos can be large or small, but the servo you will use is a standard servo that is useful in IoT environments. The position of the servo motor is set by the length of a pulse, and the servo expects to receive a pulse roughly every 20 ms. If the Arduino sends a pulse that is high for 1 ms, then the servo angle will be zero; if it is 1.5 ms, then it will be at its center position, and if it is 2 ms it will be at 180 degrees. The motor can rotate 180 degrees at max speed in about half a second, but slower and more fine-tuned turning is possible as well.

A sample circuit and Arduino code is shown below to get you started. Note that the servo's ribbon wire has red, brown, and yellow wires. Red is power, while brown is ground, and yellow is used to control the servo from the Arduino. The 470 uF capacitor ensures a steady power supply to the Arduino. Make sure the longer lead of the capacitor is connected to power while the shorter lead is connected to ground. The example code will sweep the servo position between 0 and 180 degrees back and forth. After setting up the circuit and programming the Arduino, check that the servo is sweeping back and forth steadily between 0 and 180 degrees. You can then attach arms or horns to the servo to control other objects and place the servo itself in more strategic locations. Also try experimenting with smaller delays by replacing `delay(15)` with a smaller number like 5. What happens when you try this and could it be useful?




```
#include <Servo.h>

int pin_servo = 6;

Servo servo;
int servo_angle = 0; // Servo position in degrees

void setup() {
  servo.attach(pin_servo);
}

void loop() {
  // Scan from 0 to 180 degrees

  for( servo_angle = 0; servo_angle < 180; servo_angle++ ) {
    servo.write( servo_angle );
    delay(15);
  }

  // Scan back from 180 to 0 degrees

  for( servo_angle = 180; servo_angle > 0; servo_angle-- ) {
    servo.write( servo_angle );
    delay(15);
  }
}
```

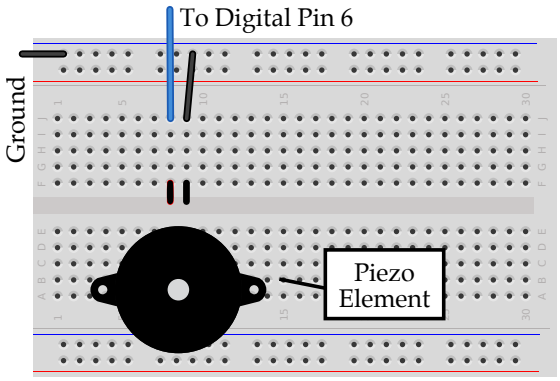
13. Piezo Output Module

This output module enables your IoT device to play tones using a piezo element. A piezo element makes a clicking sound each time it is pulsed with current. If we pulse it at the right frequency (for example 440 times a second to make the note middle A), these clicks produce notes.

A sample circuit and Arduino code is shown below to get you started. Note that the piezo element is powered directly from the Arduino signal pin. There is no need for any extra components; with the rectangular "TDX" label facing to your left, we directly connect the left lead of the piezo element to a digital output PWM pin on the Arduino and the right lead to ground.

The example code has two experiments. In the first, we play a middle A by pulsing the piezo element with a square wave at 440 Hz – i.e., 1136 µs HIGH and 1136 µs LOW. In the second experiment, we play a tune using the playNote() library function. You can use this function by giving it a note name (see the table below for notes) and a duration to play the note; the function will play the note using the same technique we saw in the first experiment. After setting up the circuit and programming the Arduino, listen to the notes produced by the piezo element and see if it makes sense. You can then experiment with other tunes by choosing your own notes! The table below lists the frequencies, periods, and the time to pulse the piezo element to play different notes across two octaves.

Note Name	Frequency	Period	Time High
C	131 Hz	7634	3817
D	147 Hz	6802	3401
E	165 Hz	6060	3030
F	175 Hz	5714	2857
G	196 Hz	5102	2551
A	220 Hz	4546	2273
B	247 Hz	4048	2024
c	261 Hz	3830	1915
d	294 Hz	3400	1700
e	329 Hz	3038	1519
f	349 Hz	2864	1432
g	392 Hz	2550	1275
a	440 Hz	2272	1136
b	493 Hz	2028	1014



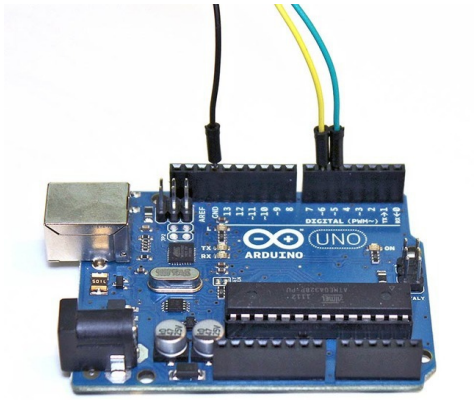

```
// Experiment 1: Playing a note

#include <Piezo.h>

int pin_piezo = 6;

void setup() {
  pinMode( pin_piezo, OUTPUT );
}

void loop() {

  // Play an A4 ( 440Hz, 2272us period ).
  // Dividing the period by 2 makes 1136us --
  // the time we hold the signal HIGH or LOW.

  while (1) {
    digitalWrite( pin_piezo, HIGH );
    delayMicroseconds( 1136 );
    digitalWrite( pin_piezo, LOW );
    delayMicroseconds( 1136 );
  }
}
```

```
// Experiment 2: Playing a tune

#include <Piezo.h>

int pin_piezo = 6;

void setup() {
  pinMode( pin_piezo, OUTPUT );
}

void loop() {

  // Set up the tune. The length is the
  // number of notes. The notes array
  // contains the tune -- a space is a
  // rest. The beats array contains how
  // many beats each note should take.

  int length   = 30;
  char notes[] = "ABcGGdcBAAAABcABcGGdcdeefedcdc ";
  int beats[]  = { 1, 1, 6, 1, 1, 6, 1, 1,
                   1, 1, 2, 1, 2, 6,
                   1, 1, 6, 1, 1, 6, 1, 1,
                   1, 2, 2, 1, 1, 1, 1, 4, 4 };
  int tempo    = 100;

  // Play tune using the playNote() function.

  for (int i = 0; i < length; i++) {
    playNote(pin_piezo, notes[i], beats[i] * tempo);
    delay(tempo / 2); // Pause between notes
  }

}
```

14. Printer Output Module

This printer output module enables your IoT device to print a paper copy of collected/processed data for the end-user. The printer is a thermal receipt printer, similar to what is used in most retail stores. A thermal printer works by selectively heating special thermochromic paper. Heated sections of paper turn black, creating the desired text or image, e.g., a barcode.

We'll be using the Arduino to control the printer. However, because the printer's thermal print head needs to be hot to work, it requires its own power supply. The printer comes with its own wall-wart power supply, which you should plug into the barrel connector to screw-down terminal converter. Ensure that positive (red) is connected to the + and negative (black) is connected to the -. The printer has small plugs on the bottom: a 2-pin (power) and a 3-pin (data). Connect the power cable to the 2-pin plug (red/black) and hook that up to the barrel connector converter. The data cable connects to the Arduino for control. Green to digital pin 5, yellow to digital pin 6, and black to ground.



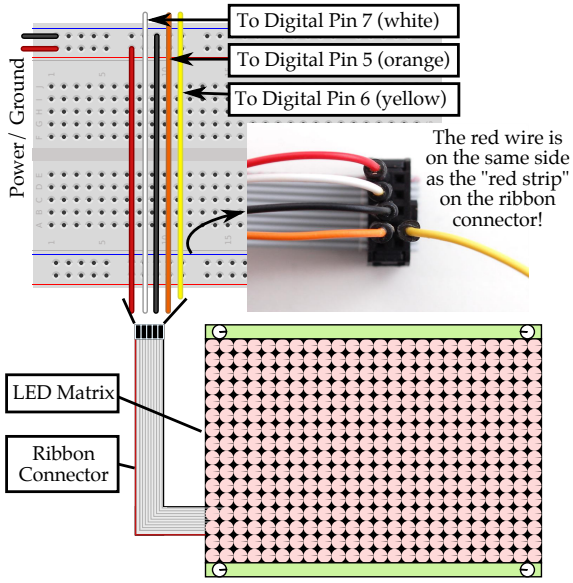

```
#include "SoftwareSerial.h"
#include "Adafruit_Thermal.h"
#include <avr/pgmspace.h>

int printer_RX_Pin = 5;
int printer_TX_Pin = 6;

Adafruit_Thermal
  printer(printer_RX_Pin, printer_TX_Pin);
```

```
void setup() {
  Serial.begin(9600);
  pinMode(7, OUTPUT);
  digitalWrite(7, LOW);
  printer.begin();
}

void loop() {
  print.println("Hello World!")

  // Set text justification (left, center, right)
  // Accepts 'L', 'C', 'R'
  printer.justify('C');
  printer.println("This Text Center Justified");

  // Set type size, accepts 'S', 'M', 'L'
  printer.setSize('M');
  printer.println("Medium Text");

  printer.feed(1);
}
```

15. LED Matrix Output Module

This output module enables your IoT device to control a 16x24 LED matrix which can display useful messages and graphics on its large screen; for example, the matrix can display digital clocks, thermometers, counters and meters, instrumentation readouts, industrial control indicators, etc. You can also chain multiples of these displays together! The panel works using a special HT1632C chip on the back which does most of the necessary work to display messages for you. Communication with the matrix happens through a 3-pin serial interface which is easy to work with.

A sample circuit and Arduino code is shown below to get you started. The LED matrix has a gray ribbon connector that has a "red strip" on one side of it. When connecting wires to the connector, the red wire for power will be on the same side as the "red strip" on the ribbon connector. The example code will draw a blinking rectangle on the screen. After setting up the circuit and programming the Arduino, check that the shape appears on the screen correctly. You can now experiment with drawing moving shapes or even scrolling text!




```
#include "HT1632.h"

int pin_data = 5; // orange wire
int pin_wr   = 6; // yellow wire
int pin_cs   = 7; // white wire

HT1632LEDMatrix matrix =
  HT1632LEDMatrix(pin_data, pin_wr, pin_cs);

void setup() {
  Serial.begin(9600);
  matrix.begin(HT1632_COMMON_16NMOS);
  matrix.fillScreen();
  delay(500);
  matrix.clearScreen();
}
```

RECTANGLES:

Use drawRect(x_coordinate, y_coordinate, rect_width, rect_height, value). In this example, we draw a rectangle at point (6, 4), that is half the width and half the height of the entire matrix. Then we clear it. Use "fillRect" to draw a filled rectangle.

```
void loop() {
  matrix.drawRect(6, 4,
            matrix.width()/2,
            matrix.height()/2, 1);
  matrix.writeScreen();
  delay(500);
  matrix.clearScreen();
  delay(500);
}

```

PIXELS:

Use drawPixel(x_coord, y_coord, value). In this example, we draw a pixel at (0, 0), delay 500 ms, clear the pixel, and wait another 500 ms.

```
void loop() {
  matrix.drawPixel(0, 0, 1);
  matrix.writeScreen();
  delay(500);
  matrix.drawPixel(0, 0, 0);
  delay(500);
}
```

LINES:

Use drawLine(x1_coord, y1_coord, x2_coord, y2_coord, value). In this example, we draw an X and then clear it.

```
void loop() {
  matrix.drawLine(0, 0,
            matrix.width()-1,
            matrix.height()-1, 1);
  matrix.drawLine(matrix.width()-1, 0,
            0, matrix.height()-1, 1);
  matrix.writeScreen();
  delay(500);
  matrix.clearScreen();
  delay(500);
}
```

CIRCLES:

Use *drawCircle(x_coord, y_coord, radius, value)*. The coordinates are for the center of the circle. In this example we draw a circle and clear it. Use "fillCircle" to draw a filled circle instead.

```
void loop() {
  matrix.drawCircle(
            matrix.width()/2-1, matrix.height()/2-1,
            8, 1);
  matrix.writeScreen();
  delay(500);
  matrix.clearScreen();
  delay(500);
}
```

TEXT:

Use:

```
- matrix.setCursor(x\_coord, y\_coord)
- matrix.print( text\_to\_print )
```

In this example, we draw two lines of text!

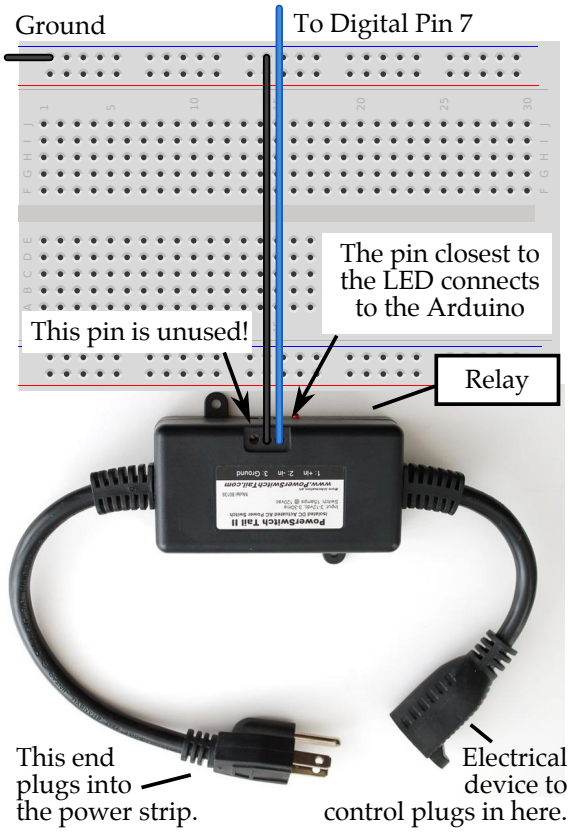
```

void loop() {
  matrix.setTextSize(1);  // size 1 == 8 pixels high
  matrix.setTextColor(1); // 'lit' LEDs

  matrix.setCursor(0, 0); // start at top left
  matrix.print("Curi");
  matrix.setCursor(0, 8); // next line, 8 pixels down
  matrix.print("e'14");
  matrix.writeScreen();
  delay(500);
  matrix.clearScreen();
  delay(500);
}
```

16. Relay Output Module

This output module enables your IoT device to control electrical devices such as heaters, small skillets, lights, etc. with a signal from an Arduino. This is ideal for making controllable lights, precision timing cooking devices, automatic shutoff water kettles and coffee roasters, home automation projects, or for controlling any device that plugs into the electrical wall socket.

A sample circuit and Arduino code is shown below to get you started. First, test the example code with the relay connected to the Arduino and *not* connected to the power strip / wall socket. Power strips / wall sockets run at a high voltage of 110 V, so it is good practice to test that the circuit works with the Arduino first, which runs at a much safer voltage of 3.3–5 V. To connect a wire to the relay pins, ask for a small flathead screwdriver and use it to loosen tiny screws above each pin (turn the screw counter-clockwise to loosen), which will open tiny wire holders in the slot below. Slip the wire into the holder and then use the screwdriver again to tighten (turn the screw clockwise to tighten) until the wire is steady and firm. Then program the Arduino.

The example code turns the relay on and off for 5s each in a loop. After programming the Arduino, you should see the LED turn on and off for 5s. The LED indicates when the device under control is receiving power. If this works, you can plug one end of the relay into a power strip / wall socket, and you can plug an appliance (e.g., a mini-fan) into the other end. Turn the fan on and make sure it is on for 5s and off for 5s. Try experimenting with other devices and see what you can control! Modify the 5s looped Arduino code if you want to control timing in other ways.




```
int pin_relay = 7;

void setup() {
  pinMode( pin_relay, OUTPUT );
}

void loop() {
  digitalWrite( pin_relay, HIGH );
  delay(5000);
  digitalWrite( pin_relay, LOW );
  delay(5000);
}
```