Disclaimer

- Many of these slides are mine
- But, some are stolen from various places on the web
  - todbot.com – Bionic Arduino and Spooky Arduino class notes from Tod E.Kurt
  - ladyada.net – Arduino tutorials by Limor Fried
Part 1 – Arduino SW

Remember, Arduino calls programs “sketches”
Procedure

Using Arduino

- Write your sketch
- Press Compile button (to check for errors)
- Press Upload button to program Arduino board with your sketch

Try it out with the “Blink” sketch!

Load “File/Sketchbook/Examples/Digital/Blink”

Get the Blink Example
Blink Sketch (program)

/*
* Blink
*
* The basic Arduino example. Turns on an LED on for one second,
* then off for one second, and so on... We use pin 13 because,
* depending on your Arduino board, it has either a built-in LED
* or a built-in resistor so that you need only an LED.
*/

int ledPin = 13;                               // LED connected to digital pin 13

void setup() {                                  // run once, when the sketch starts
  pinMode(ledPin, OUTPUT);      // sets the digital pin as output
}

void loop()                                      // run over and over again
{
  digitalWrite(ledPin, HIGH);         // sets the LED on
  delay(1000);                               // wait for a second
  digitalWrite(ledPin, LOW);          // sets the LED off
  delay(1000);                               // wait for a second
}

Variables

int ledPin = 13;      // LED connected to digital pin 13

- ledPin is a variable that holds a 16-bit value
  - 16 binary digits is enough for -32768 to 32767
  - Default starting value is 13
  - There are other data types you can use
Data Types on Arduino

- By default, types are signed unless you say “unsigned”…

<table>
<thead>
<tr>
<th>Type</th>
<th>Size (bits)</th>
<th>Size (bytes)</th>
<th>Minimum</th>
<th>Maximum</th>
</tr>
</thead>
<tbody>
<tr>
<td>boolean</td>
<td>1</td>
<td>1</td>
<td>0 (false)</td>
<td>1 (true)</td>
</tr>
<tr>
<td>unsigned byte</td>
<td>8</td>
<td>1</td>
<td>0</td>
<td>255</td>
</tr>
<tr>
<td>byte</td>
<td>8</td>
<td>1</td>
<td>-128</td>
<td>127</td>
</tr>
<tr>
<td>unsigned int</td>
<td>16</td>
<td>2</td>
<td>0</td>
<td>65,535</td>
</tr>
<tr>
<td>int</td>
<td>16</td>
<td>2</td>
<td>-32,768</td>
<td>32,767</td>
</tr>
<tr>
<td>unsigned long</td>
<td>32</td>
<td>4</td>
<td>0</td>
<td>4,294,967,295</td>
</tr>
<tr>
<td>long</td>
<td>32</td>
<td>4</td>
<td>-2,147,483,648</td>
<td>-2,147,483,647</td>
</tr>
<tr>
<td>float (double)</td>
<td>32</td>
<td>4</td>
<td>-3.4028235E+38</td>
<td>3.4028235E+38</td>
</tr>
</tbody>
</table>

Functions

```cpp
def setup() {  
    // run once, when the sketch starts  
    pinMode(ledPin, OUTPUT); // sets the digital pin as output  
}

def function_name(<arguments>) {  
    // function-body  
}
```

- “void” means no value is returned
- `pinMode(ledPin, OUTPUT);` // call another function
Pseudo-code Silly Function

clean-cat wash-the-cat (dirty-cat cat) // a procedure for washing the cat
{
  turn on the shower.
  find the cat.
  grab the cat.
  put cat under shower.
  wait 3 minutes. // wait for cat to get clean.
  release clean-cat.
}

Required Arduino Functions

void setup() { // run once, when the sketch starts
  // typically pin definitions
  <initialization statement>;
  ... // and other init stuff
  <initialization statement>;
}

void loop() { // run over and over again
  <main loop statement>;
  ... // the guts of your program
  <main loop statement>;
  ... // which could include calls
  <main loop statement>;
}
Arduino Language (C/C++)

- `pinMode(pin, mode);`
  - `pin` is a number, `mode` can be `INPUT` or `OUTPUT`

- `digitalWrite(pin, value);`
  - `Value` can be `HIGH (1)` or `LOW (0)`

- `digitalRead(pin);`
  - Returns an `int` – value either `HIGH` or `LOW`

- `delay(val);`
  - Pauses for `val` milliseconds (1/1000’s of a sec)
  - `val` can be up to `unsigned long max`

- `millis();`
  - Returns number of milliseconds since the program started running
  - Returns an `unsigned long` – overflows in ~50 days

Blink Sketch (program)

```cpp
/**<*
 * Blink
 * *
 * The basic Arduino example. Turns on an LED on for one second,
 * then off for one second, and so on... We use pin 13 because,
 * depending on your Arduino board, it has either a built-in LED
 * or a built-in resistor so that you need only an LED.
 * */

int ledPin = 13; // LED connected to digital pin 13
void setup() { // run once, when the sketch starts
   pinMode(ledPin, OUTPUT); // sets the digital pin as output
}

void loop() { // run over and over again
   digitalWrite(ledPin, HIGH); // sets the LED on
   delay(1000); // wait for a second
   digitalWrite(ledPin, LOW); // sets the LED off
   delay(1000); // wait for a second
}
```
Blink Modifications

- Change so that blink is on for 50msec and off for 50msec
  - What happens?
- Change so that blink is on for 10ms and off for 10ms
  - What happens?
- Change to use an external LED rather than the one on the board
  - Pay attention to current! Use a current-limiting resistor!

Making Circuits

heart pumps, blood flows  voltage pushes, current flows
Wiring it Up

wiring diagram schematic wiring it up

Electricity flows in a loop. Can stop flow by breaking the loop

Wiring it Up

wiring diagram schematic

Arduino Diecimila board has this circuit built-in
To turn on LED use `digitalWrite(13, HIGH)`
External LED

- Remember Ohm’s Law
  - \( V = IR \quad I = \frac{V}{R} \quad R = \frac{V}{I} \)
- Every LED has a \( V_f \) “Forward Voltage”
  - How much voltage is dropped passing through the LED
- \( R = \frac{(V - V_f)}{I} \)
  - Example – If \( V_f \) is 1.9v (red LED), and \( V = 5 \)v, and you want 15mA of current (0.015A)
    - \( R = \frac{(5 - 1.9)}{0.015} = 3.1/0.015 = 206\Omega \)
    - Exact isn’t critical – use next size up, i.e. 220\Ω
    - Or be safe and use 330\Ω or 470\Ω
    - This would result in 9.4mA or 6.6mA which is fine

LEDs and Resistors

- On LEDs, polarity matters.
  - Shorter lead is “negative” side, goes to ground
  - Longer lead is “positive” side, goes to power
- Polarity doesn’t matter on resistors

- Anode +
- Cathode -
- Current flows from Anode to Cathode
- Lights up when current flows
Proto Boards

AKA Solderless Breadboards

Wire it Up
Wire it Up

plugged into "ground" bus

Resistor Color Codes

What's the color code for a 220Ω resistor?

What's the color code for a 1kΩ resistor?

What's the color code for a 470Ω resistor
Resistor Color Codes

What’s the color code for a 220Ω resistor?
- red
- red
- brown  
- gold

What’s the color code for a 1kΩ resistor?
- brown
- black
- red  
- gold

What’s the color code for a 470Ω resistor?
- yellow
- violet
- brown  
- gold

We’re using 4-band 5% resistors with a 1/4 watt rating.
Wire it Up
- Wire up an external LED of your choice, and change the Blink program to use that external LED
- Choose your resistor based on the Vf of the LED you’re using

Blink Subtlety
- When the `delay(val);` function runs, nothing else can happen
  - Arduino just sits there counting milliseconds
  - For blink this is just fine, but later you may want other things to be going on while the Arduino is counting
  - Load BlinkWithoutDelay from the examples
  - Let’s look at what it does…
  - C “if” statement
    - `if (condition) { do something};`
    - `if (condition) {do something}`
    - else {do something else};
BlinkWithoutDelay

const int ledPin = 13; // const says this won't change
int ledState = LOW; // used to set the state of the LED
long previousMillis = 0; // used to store last time LED changed
long interval = 1000; // interval at which to blink the LED

void setup() {
  pinMode(ledPin, OUTPUT); // set LED pin mode
}

void loop () {
  // check to see if it's time to change the LED value
  if (millis() – previousMillis > interval) {
    previousMillis = millis(); // save the time you made the change
    if (ledState == LOW) { ledState = HIGH; } // toggle the state of the LED
    else { ledState = LOW; } ;
    digitalWrite(ledPin, ledState); // set the LED with ledState
  }
  // you can do other things here if it's not time to change the LED state
}

Comparison Operators

x == y (x is equal to y)
x != y (x is not equal to y)
x < y (x is less than y)
x > y (x is greater than y)
x <= y (x is less than or equal to y)
x >= y (x is greater than or equal to y)

Beware of x=y;  This does an assignment, not a comparison!
Moving on…

Varying LED Brightness
Same circuit as Blink circuit but pin 9 instead of pin 13

The PWM pins work with the "analogWrite(value)" command where "value" ranges between 0 and 255. To turn LED to half-bright, use analogWrite(9, 128)

Pulse Width Modulation

- analogWrite(pin, value);
  - value can be 0 to 255
  - Must be one of the “PWM pins”: pins 3, 5, 6, 9, 10, 11
  - Don’t need to set pinMode to OUTPUT (but won’t hurt)
C “for” loop

for (initialization; condition; increment) {
    // do something
}

int i; // define an int to use as a loop variable
for (i = 0; i <= 255; i++) {
    analogWrite(pin, i);
    delay(50);
}

C Compound Operators

x = x + 1;
x += 5; // same as x = x + 5
x++; // same as x = x + 1
x = x - 2;
x -= 3; // same as x = x - 3
x--; // same as x = x - 1
x = x * 3;
x *=5; // same as x = x * 5
Moving on…

- Write a program to make the LED flicker like a flame
  - Choose a random intensity
  - For a random amount of time
  - Use `analogWrite(ledPin, val)`
  - Also, `random(min, max);` will return a random number between min and max.
    - `randomSeed(int);` will initialize the random function

Candle Program

```cpp
int ledPin = 9;   // select pin for LED output
int bright = 0;   // Variable to hold LED brightness
int time = 0;   // variable to hold delay time

void setup () {
    randomSeed(0);   // initialize the random function
    pinMode(ledPin, OUTPUT);   // LED should be output
}

void loop() {
    bright = random(100, 255);   // random brightness value
    analogWrite(ledPin, bright);   // set the LED brightness

    time = random(10,150);   // random time in ms
    delay(time);   // delay for that time
}
```
Silly LED Tricks

LED Wiring – 2 ways

To turn ON: `digitalWrite(9, HIGH)`
To turn OFF: `digitalWrite(9, LOW)`
To set brightness: `analogWrite(9, val)`

To turn ON: `digitalWrite(9, LOW)`
To turn OFF: `digitalWrite(9, HIGH)`
To set brightness: `analogWrite(9, 255 - val)`
Getting Input (Digital)

- Switches make or break a connection
- But Arduino wants to see a voltage
  - Specifically, a “HIGH” (5 volts)
  - or a “LOW” (0 volts)

How do you go from make/break to high/low?

Switches

- Digital inputs can “float” between 0 and 5 volts
- Resistor “pulls down” input to ground (0 volts)
- Pressing switch sets input to 5 volts
- Press is HIGH, Release is LOW

Why do we need the “pull down” resistor?
Another Switch

- Resistor pulls up input to 5 volts
- Switch sets input to 0 volts
- But now the sense is inverted
  - Press is LOW
  - Release is HIGH

“pull-up”

A Switch

“gap”

connect when pushed

always connected together

Pressing the button, “closes the gap”
Using a Switch

Using digitalRead()

- In setup(): use pinMode(myPin, INPUT) to make pin an input
- In loop(): use digitalWrite(myPin) to get switch position
  - If doing many tests, use a variable to hold the output value of digitalRead().
  - e.g. val = digitalRead(myPin)
Moving on…

- Write a program that reads the value on an input pin
- Use the button to change from blinking fast to blinking slow
Moving on…

- Write a program that reads the value on an input pin
- Use the button to change from blinking fast to blinking slow

```c
int ledPin = 13; // choose the pin for the LED
int ledVal = 0; // variable for reading the pin status
int delayVal = 1000;

void setup() {
  pinMode(ledPin, OUTPUT); // declare LED as output
  pinMode(swPin, INPUT); // declare pushbutton as input
}
void loop() {
  uint8_t swVal = digitalRead(swPin); // read input value
  if (swVal == HIGH) {
    delayVal = 10000;
  } else {
    delayVal = 1000;
  }
  digitalWrite(ledPin, HIGH); // blink the LED and go OFF
  delay(delayVal);
  digitalWrite(ledPin, LOW);
  delay(delayVal);
}
```

Multiple Switches

Same sub-circuit, just duplicate
Make Your Own Switches

- Anything that makes a connection
- Wires, tin foil, tinfoil balls, ball bearings
- Pennies!
- Nails, bolts, screws

- Or repurpose these tiny switches as bump detectors or closure detectors
Analog Input

To computers, analog is chunky

- Many states, not just two (HIGH/LOW)
- Number of states (or "bins") is resolution
- Common computer resolutions:
  - 8-bit = 256 states
  - 16-bit = 65,536 states
  - 32-bit = 4,294,967,296 states
Analog Input on Arduino

- Our version uses ATMega328p
- six ADC inputs (Analog to Digital Converter)
- Voltage range is 0-5v
- Resolution is 10 bits (digital values between 0-1023)
- In other words, 5/1024 ~ 4.8mV is the smallest voltage change you can measure

`analogRead(pin);`
- reads an analog pin
- returns a digital value between 0-1023
- analog pins need no pinMode declaration

Analog Input

Sure sure, but how to make a varying voltage? With a potentiometer. Or just pot.

![Potentiometer Diagram](image-url)
Potentiometers

Moving the knob is like moving
where the arrow taps the voltage on the resistor

- turned anti-clockwise
  - 0 volts
  - gnd
- turned clockwise
  - 5 volts
  - gnd
- somewhere in the middle
  - 2.3 volts
  - gnd

Arduino Analog Input

Red to Vcc
Purple to A0
Blue to Gnd
Moving on…

- Write a program to read an analog value from a pot and use that value to control the brightness of an LED
  - Fade the LED by turning the pot

- Useful function is
  
  \[
  y = \text{map}(x, 0, 1023, 50, 150);
  \]

- Also remember
  
  \[
  \text{analogWrite}(\text{pin}, \text{value});
  \]
  
  - PWM value from 0-255
potFade

```c
int potPin = 0;   // the analog input pin from the pot
int ledPin = 9;   // pin for LED (a PWM pin)
int val;    // Variable to hold pot value

void setup () {
    pinMode(ledPin, OUTPUT);   // declare ledPin as output
}

void loop() {
    val = analogRead(potPin);   //read the value from the pot
    val = map(val, 0, 1023, 100, 255);  // map to reasonable values
    analogWrite(ledPin, val);
}
```

What good are pots?

- Anytime you need a ranged input
  - (we’re used to knobs)
- Measure rotational position
  - steering wheel, etc.

- But more importantly for us, potentiometers are a good example of a resistive sensor
Sensing the Dark

- Pots are example of a voltage divider
- Voltage divider splits a voltage in two
- Same as two resistors, but you can vary them

\[ +5V \quad = \quad +5V \]

Sensing the Dark: Photocells

- aka. photoresistor, light-dependent resistor
- A variable resistor
- Brighter light \( \Rightarrow \) lower resistance
- Photocells you have range approx. 0-10k

[Photocell schematic symbol]
Photocell Circuit

![Photocell Circuit Diagram]

Photocell Arduino Sketch

Can use as before, sketch “analog_read_led”

```cpp
int potPin = 2; // select the input pin for the potentiometer
int ledPin = 13; // select the pin for the LED
int val = 0; // variable to store the value coming from the sensor

void setup() {
  pinMode(ledPin, OUTPUT); // declare the ledPin as an OUTPUT
}

void loop() {
  val = analogRead(potPin); // read the value from the sensor
  digitalWrite(ledPin, HIGH); // turn the ledPin on
  delay(val); // stop the program for some time
  digitalWrite(ledPin, LOW); // turn the ledPin off
  delay(val); // stop the program for some time
}
```

Wave your hand over it = blink faster
Point it towards the light = blink slower
Moving on…

- Connect a photocell instead of a pot to your fading circuit
- Do you get the same range of fade as with the pot?
- Why or why not?

Resistive sensors

- thermistor (temperature)
- flex sensor (bend, deflection)
- force sensors (pressure)
- also air pressure and others
LED Brightness Functions

Then turn those numbers into an array

```c
byte bright_table[] = { 30, 30, 30, 40, 50, 60, 70, 80, 90, 100, 
110, 120, 130, 140, 150, 160, 170, 180, 190, 200, 
210, 220, 230, 240, 250, 250, 240, 230, 220, 210, 
200, 190, 180, 170, 160, 150, 140, 130, 120, 110, 
100, 90, 80, 70, 60, 50, 40, 30, 30, 30 };
```

Use any pattern of numbers you like
but they must range between 0-255

0 = full off
127 = half on
255 = full on

LED Brightness Functions

Once you have your table...

```c
byte bright_table[] = { 30, 30, 30, 40, 50, 60, 70, 80, 90, 100, 
110, 120, 130, 140, 150, 160, 170, 180, 190, 200, 
210, 220, 230, 240, 250, 250, 240, 230, 220, 210, 
200, 190, 180, 170, 160, 150, 140, 130, 120, 110, 
100, 90, 80, 70, 60, 50, 40, 30, 30, 30 };
```

...the rest is just programming

1. Get a `bright_table` value
2. Send it out with `analogWrite()`
3. Advance counter into `bright_table`
4. Wait a bit
5. Repeat
Glowing Eyes Sketch

```
int potPin = 0;
int ledPin = 10;

// the table containing the "curve" the brightness should take
byte bright_table[] = { 30, 30, 30, 40, 50, 60, 70, 80, 90, 100,
  110,120,130,140,150,160,170,180,190,200,
  210,220,230,240,250,260,270,280,290,300,
  310,320,330,340,350,360,370,380,390,400,
  410,420,430,440,450,460,470,480,490,500,
  510,520,530,540,550,560,570,580,590,600,
  610,620,630,640,650,660,670,680,690,700,
  710,720,730,740,750,760,770,780,790,800,
  810,820,830,840,850,860,870,880,890,900,
  910,920,930,940,950,960,970,980,990,1000,
  1100,1200,1300,1400,1500,1600,1700,1800,1900,2000,
  2100,2200,2300,2400,2500,2600,2700,2800,2900,3000,
  3100,3200,3300,3400,3500,3600,3700,3800,3900,4000,
  4100,4200,4300,4400,4500,4600,4700,4800,4900,5000,
  5100,5200,5300,5400,5500,5600,5700,5800,5900,6000,
  6100,6200,6300,6400,6500,6600,6700,6800,6900,7000,
  7100,7200,7300,7400,7500,7600,7700,7800,7900,8000,
  8100,8200,8300,8400,8500,8600,8700,8800,8900,9000,
  9100,9200,9300,9400,9500,9600,9700,9800,9900,10000,
  10100,10200,10300,10400,10500,10600,10700,10800,10900,11000,
  11100,11200,11300,11400,11500,11600,11700,11800,11900,12000,
  12100,12200,12300,12400,12500,12600,12700,12800,12900,13000,
  13100,13200,13300,13400,13500,13600,13700,13800,13900,14000,
  14100,14200,14300,14400,14500,14600,14700,14800,14900,15000,
  15100,15200,15300,15400,15500,15600,15700,15800,15900,16000,
  16100,16200,16300,16400,16500,16600,16700,16800,16900,17000,
  17100,17200,17300,17400,17500,17600,17700,17800,17900,18000,
  18100,18200,18300,18400,18500,18600,18700,18800,18900,19000,
  19100,19200,19300,19400,19500,19600,19700,19800,19900,20000,
  20100,20200,20300,20400,20500,20600,20700,20800,20900,21000,
  21100,21200,21300,21400,21500,21600,21700,21800,21900,22000,
  22100,22200,22300,22400,22500,22600,22700,22800,22900,23000,
  23100,23200,23300,23400,23500,23600,23700,23800,23900,24000,
  24100,24200,24300,24400,24500,24600,24700,24800,24900,25000,
  25100,25200,25300,25400,25500,25600,25700,25800,25900,26000,
  26100,26200,26300,26400,26500,26600,26700,26800,26900,27000,
  27100,27200,27300,27400,27500,27600,27700,27800,27900,28000,
  28100,28200,28300,28400,28500,28600,28700,28800,28900,29000,
  29100,29200,29300,29400,29500,29600,29700,29800,29900,30000,
};

int max_count = 50; // number of entries in the bright_table
int count = 0; // position within the bright_table
int vol = 0; // variable for reading pin status

void setup() {
  pinMode(ledPin, OUTPUT); // sets the digital pin as output
}

void loop() {
  analogWrite(ledPin, bright_table[count]); // sets the LED brightness
  count++; // moves counter to next position in table
  if (count > max_count) {
    count = 0; // if at end of table, back to start
  }
  vol = analogRead(potPin);
  vol = vol/4; // scale it down so it's quicker
  delay(vol);
}
```

Communicating with Others

- Arduino can use same USB cable for programming and to talk with computers
- Talking to other devices uses the “Serial” commands
  - `Serial.begin()` – prepare to use serial
  - `Serial.print()` – send data to computer
  - `Serial.read()` – read data from computer
Serial Communication

- `Serial.begin(baud-rate);`
  - baud-rate is 300, 1200, 2400, 4800, 9600, 14400, 19200, 28800, 57600, or 115200
  - Sets serial bit rate

- `Serial.print(arg);`
  - Sends `arg` to the serial output – can be number or string
  - `Serial.print(arg,format);` // formats the arg
  - format can be BYTE, BIN, OCT, DEC, HEX

- `Serial.println(arg);`
  - Same, but also prints a newline to the output

Serial Communication

- `Serial.available();`
  - Returns an int that tells you how many bytes remain in the input buffer

- `Serial.read();`
  - Returns the next byte waiting in the input buffer

- `Serial.flush();`
  - Clear the input buffer of any remaining bytes
Serial Example

```cpp
int incomingByte = 0; // for incoming serial data
void setup() {
    Serial.begin(9600); // opens serial port, sets data rate to 9600 bps
}

void loop() { // send data only when you receive data:
    if (Serial.available() > 0) { // read the incoming byte:
        incomingByte = Serial.read();

        // say what you got:
        Serial.print("I received: ");
        Serial.println(incomingByte, DEC);
    }
}
```

Arduino Says “Hi”

“SerialHelloWorld”

Sends “Hello world!” to your computer

Click on “Serial Monitor” button to see output

Watch TX LED compared to pin 13 LED
Telling Arduino What To Do

“SerialReadBasic”

You type “H”, LED blinks

In “Serial Monitor”, type “H”, press Send

Serial.available() tells you if data present to read

Arduino Communications

is just serial communications

- Psst, Arduino doesn’t really do USB
- It really is “serial”, like old RS-232 serial
- All microcontrollers can do serial
- Not many can do USB
- Serial is easy, USB is hard
Serial Communications

- “Serial” because data is broken down into bits, each sent one after the other down a single wire.
- The single ASCII character ‘B’ is sent as:
  \[ \text{’B’} = 01000010 \]
  \[ = \text{L H L L L H L} \]
  \[ = \text{HIGH} \text{LOW} \]
- Toggle a pin to send data, just like blinking an LED
- You could implement sending serial data with `digitalWrite()` and `delay()`
- A single data wire needed to send data. One other to receive.

Arduino & USB-to-serial

Arduino board is really two circuits
Arduino Mini

Arduino Mini separates the two circuits

Arduino Mini USB adapter

Arduino Mini

Arduino to Computer

USB is totally optional for Arduino
But it makes things easier
Arduino & USB

- Since Arduino is all about serial
- And not USB,
- Interfacing to things like USB flash drives, USB hard disks, USB webcams, etc. is not possible

Controlling the Computer

- Can send sensor data from Arduino to computer with Serial.print()
- There are many different variations to suite your needs:

```c
int val = 123;
Serial.print(val);  // sends 3 ASCII chars "123"
Serial.print(val, DEC); // same as above
Serial.print(val, HEX); // sends 2 ASCII chars "7B"
Serial.print(val, BIN); // sends 8 ASCII chars "01111011"
Serial.print(val, BYTE); // sends 1 byte, the verbatim value
```
Controlling the Computer

You write one program on Arduino, one on the computer.

**In Arduino:** read sensor, send data as byte

```cpp
class zero { 
val = analogRead(analogInput); // read the value on analog input
Serial.print(val/4, BYTE); // print a byte value out
delay(50); // wait a bit to not overload the port
}
```

**In Processing:** read the byte, do something with it

```java
import processing.serial.*;
Serial myPort; // The serial port

void setup() {
    String portname = "/dev/tty.usbserial-A30076WQ";
    myPort = new Serial(this, myPort, 9600);
}

void draw() {
    while (myPort.available() > 0) {
        int inByte = myPort.read();
        print(inByte);
    }
}
```

Controlling the Computer

- Receiving program on the computer can be in any language that knows about serial ports
- C/C++, Perl, PHP, Java, Max/MSP, Python, Visual Basic, etc.
- Pick your favorite one, write some code for Arduino to control
Controlling Arduino, Again

"SerialReadBlink"

Type a number 1-9 and LED blinks that many times

Converts typed ASCII value into usable number

Most control issues are data conversion issues

<table>
<thead>
<tr>
<th>Dec</th>
<th>Hex</th>
<th>ASCII</th>
<th>Char</th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td>00</td>
<td>#</td>
<td>$</td>
</tr>
<tr>
<td>32</td>
<td>01</td>
<td>@</td>
<td>A</td>
</tr>
<tr>
<td>33</td>
<td>02</td>
<td>!</td>
<td>B</td>
</tr>
<tr>
<td>34</td>
<td>03</td>
<td>&quot;</td>
<td>C</td>
</tr>
<tr>
<td>35</td>
<td>04</td>
<td>$</td>
<td>D</td>
</tr>
<tr>
<td>36</td>
<td>05</td>
<td>#</td>
<td>E</td>
</tr>
<tr>
<td>37</td>
<td>06</td>
<td>%</td>
<td>F</td>
</tr>
<tr>
<td>38</td>
<td>07</td>
<td>&amp;</td>
<td>G</td>
</tr>
<tr>
<td>39</td>
<td>08</td>
<td>(</td>
<td>H</td>
</tr>
<tr>
<td>40</td>
<td>09</td>
<td>)</td>
<td>I</td>
</tr>
<tr>
<td>41</td>
<td>0A</td>
<td>*</td>
<td>J</td>
</tr>
<tr>
<td>42</td>
<td>0B</td>
<td>+</td>
<td>K</td>
</tr>
<tr>
<td>43</td>
<td>0C</td>
<td>-</td>
<td>L</td>
</tr>
<tr>
<td>44</td>
<td>0D</td>
<td>.</td>
<td>M</td>
</tr>
<tr>
<td>45</td>
<td>0E</td>
<td>/</td>
<td>N</td>
</tr>
<tr>
<td>46</td>
<td>0F</td>
<td>?</td>
<td>O</td>
</tr>
<tr>
<td>47</td>
<td>10</td>
<td>0</td>
<td>P</td>
</tr>
<tr>
<td>48</td>
<td>11</td>
<td>1</td>
<td>Q</td>
</tr>
<tr>
<td>49</td>
<td>12</td>
<td>2</td>
<td>R</td>
</tr>
<tr>
<td>50</td>
<td>13</td>
<td>3</td>
<td>S</td>
</tr>
<tr>
<td>51</td>
<td>14</td>
<td>4</td>
<td>T</td>
</tr>
<tr>
<td>52</td>
<td>15</td>
<td>5</td>
<td>U</td>
</tr>
<tr>
<td>53</td>
<td>16</td>
<td>6</td>
<td>V</td>
</tr>
<tr>
<td>54</td>
<td>17</td>
<td>7</td>
<td>W</td>
</tr>
<tr>
<td>55</td>
<td>18</td>
<td>8</td>
<td>X</td>
</tr>
<tr>
<td>56</td>
<td>19</td>
<td>9</td>
<td>Y</td>
</tr>
<tr>
<td>57</td>
<td>1A</td>
<td>:</td>
<td>Z</td>
</tr>
<tr>
<td>58</td>
<td>1B</td>
<td>;</td>
<td>[</td>
</tr>
<tr>
<td>59</td>
<td>1C</td>
<td>&lt;</td>
<td>`</td>
</tr>
<tr>
<td>60</td>
<td>1D</td>
<td>=</td>
<td>{</td>
</tr>
<tr>
<td>61</td>
<td>1E</td>
<td>&gt;</td>
<td></td>
</tr>
<tr>
<td>62</td>
<td>1F</td>
<td>?</td>
<td>_</td>
</tr>
</tbody>
</table>

ASCII codes

Standard byte codes for characters

Mysterious \texttt{val = val - '0';} statement converts the byte that represents the character to a byte of that number

For example, if the character is '3', the ASCII code is 51

The ASCII code for '0' is 48

So, 51 - 48 = 3

This converts the character '3' into the number 3
Moving on… Servos

- Servo motors are small DC motors that have a range of motion of 0-180°
- Internal feedback and gearing to make it work
- easy three-wire interface
- position is controlled by PWM signals
PWM

Output voltage is averaged from on vs. off time

\[
\text{output\_voltage} = (\text{on\_time} / \text{off\_time}) \times \text{max\_voltage}
\]

- 3.75 Volts
- 2.5 Volts
- 1.0 Volts

PWM

- Used everywhere
  - Lamp dimmers, motor speed control, power supplies, noise making
- Three characteristics of PWM signals
  - Pulse width range (min/max)
  - Pulse period (= 1/pulses per second)
  - Voltage levels (0-5V, for instance)
Servomotors

- Can be positioned from 0-180° (usually)
- Internal feedback circuitry & gearing takes care of the hard stuff
- Easy three-wire PWM 5V interface

Servos are Awesome

- DC motor
- High-torque gearing
- Potentiometer to read position
- Feedback circuitry to read pot and control motor
- All built in, you just feed it a PWM signal
Servos, good for what?

- Roboticists, movie effects people, and puppeteers use them extensively
- Any time you need controlled, repeatable motion
- Can turn rotation into linear movement with clever mechanical levers

Servos

- Come in all sizes
  - from super-tiny
  - to drive-your-car
- But all have the same 3-wire interface
- Servos are spec'd by:

  weight: 9g
  speed: 0.12s/60deg @ 6V
  torque: 22oz/l.5kg @ 6V
  voltage: 4.6–6V
  size: 21x11x28 mm

Our servos are: weight: 9g,
speed 0.12s/60deg at 4.8v,
torque (@4.8v) 17.5oz/in (1kg/cm)
voltage range: 3.0 – 7.2v
Servo Mounts & Linkages

Lots of ways to mount a servo

And turn its rotational motion into other types of motion

Servo Control

- PWM freq is 50 Hz (i.e. every 20 milliseconds)
- Pulse width ranges from 1 to 2 milliseconds
  - 1 millisecond = full anti-clockwise position
  - 2 milliseconds = full clockwise position

Ground (0V)
Power (+5V)
Control (PWM)
Servo Movement

<table>
<thead>
<tr>
<th>0 degrees</th>
<th>90 degrees</th>
<th>180 degrees</th>
</tr>
</thead>
<tbody>
<tr>
<td>1000 microsecs</td>
<td>1500 microsecs</td>
<td>2000 microsecs</td>
</tr>
</tbody>
</table>

In practice, pulse range can range from 500 to 2500 microsecs

Servo and Arduino

First, add some jumper wires to the servo connector.
Servo Example Program

#include <Servo.h>   // include the built-in servo library
Servo myservo;    // create a servo object to control the servo
int pos = 0;      // variable to store the servo position

void setup() {
    myservo.attach(9);  // attach servo control to pin 9
}

void loop() {
    for(pos = 0; pos < 180; pos++) {  // go from 0 to 180 degrees
        myservo.write(pos);  // move the servo
        delay(15);  // give it time to get there
    }
    for (pos = 180; pos>=1; pos--) {  // wave backwards
        myservo.write(pos);
        delay(15);
    }
}

Servo Functions

- Servo is a class
  - Servo myservo; // creates an instance of that class
- myservo.attach(pin);
  - attach to an output pin (doesn’t need to be PWM pin!)
  - Servo library can control up to 12 servos on our boards
  - but a side effect is that it disables the PWM on pins 9 and 10
- myservo.write(pos);
  - moves servo – pos ranges from 0-180
- myservo.read();
  - returns the current position of the servo (0-180)
Moving on…

- Write a program to control the position of the servo from a pot, or from a photocell
  - remember pot analogRead(); values are from 0-1023
  - measure the range of values coming out of the photocell first?
  - use Serial.print(val); for example
  - use map(...); to map values to 0-180

Side Note - Power

- Servos can consume a bit of power
  - We need to make sure that we don’t draw so much power out of the Arduino that it fizzes
  - If you drive more than a couple servos, you probably should put the servo power pins on a separate power supply from the Arduino
  - Use a wall-wart 5v DC supply, for example
LEDs – use current limiting resistors (remember color code!)
- drive from `digitalWrite(pin,val);` for on/off
- drive from `analogWrite(pin,val);` for PWM dimming (values from 0-255)

Buttons – current limiting resistors again
- active-high or active low (pullup or pulldown)
- read with `digitalRead(pin);`

Potentiometers (pots) – voltage dividers with a knob
- use with `analogRead(pin);` for values from 0-1023
Summary – Whew!

- photocells – variable resistors
  - use with current-limiting resistors (to make voltage divider)

- Serial communications – read a byte, or write a value
  - communicate to the Arduino environment, or your own program

- Servos – use Servo library to control motion
  - might need external power supply
  - range of motion 0-180°

- Also `setup()` and `loop()` functions, and various C programming ideas

More Later…

- DC Motors
  - use transistors as switches for larger current loads

- Stepper motors
  - Sort of like servos, but with continuous range of motion
  - Can also be more powerful

- I2C serial bus
  - Various LED driver chips
  - other serially-controlled devices

- Piezo buzzers
  - make some noise!
  - But you can also use them as input devices to sense movement

- IR motion sensors
  - simple motion and also distance sensors

- Accelerometers
  - Wii nunchucks, for example

- Others?
Assignment #2

- Form teams
  - three teams – each with one FA3800 student on it

- Check out some supplies
  - Arduino, pots, LEDs, servos, resistors, etc.

- Do something cool!
  - Rather fuzzy specification, but be creative
  - Due next week? Two weeks?