A Word on Safety

- Electronics are toxic to you
  - Lead in some of the parts
  - Wash up afterwards
- You are toxic to electronics
  - Static-sensitive: don’t shuffle your feet
  - Wires only bend so much

What is Arduino?

- Open Source Physical Computing Platform
  - open source: free to inspect & modify
  - physical computing. er, what? ubiquitous computing, pervasive computing, ambient intelligence, calm computing, everyware, spimes, blogjects, smart objects...
- A physical board, a programming environment, a development philosophy
- Tiny computer you can program
  - Completely stand-alone, talks to other devices

Physical computing as invisible computing
Can run off a battery
Can talk to other computers, cell phones, etc.

What is Arduino?

- Based on AVR-GCC, avr-libc, AVRlib and Processing (all open source projects)
- Very similar to Basic Stamp (if you know it)
  - but cheaper, faster, & open
- Uses AVR ATmega8 microcontroller chip

Basic Stamp uses PIC microcontroller chip.
PICs and AVRs are very comparable, one's not necessarily better than the other
AVRs are a little better if you're using a language like C (stack-based)
Don't need to worry about the chip particulars for now

What is Arduino?

- Why not just use a bare AVR ATmega8 chip?

- Arduino is also a standardized "bootloader"
  - A tiny program that loads other programs
  - It's alive during first 5 seconds

A bootloader is akin to an BIOS on a real computer. It handles the startup of the chip
After 5 seconds, your program runs
Don't need special programmer board with a bootloader
Arduino can work with other AVR chips, some are smaller than your fingernail, cost ~ 40 cents

What is Arduino?

- Capabilities
  - 8 kBytes of Flash program memory
  - 1 kByte of RAM
  - 12 MHz (Apple II: 1 MHz)
  - Inputs and Outputs
    - 13 digital input/output pins
    - 5 analog input pins

Digital I/O can read switches and buttons, control LEDs and motors
Analog input can read knobs or other varying sensors
Analog output can be done with PWM

What is Arduino?

But how do you program it?

- Write programs on your PC
- Download them into the Arduino board
- Arduino board can then be used by itself

No keyboard, mouse or display
Your PC becomes the “head”

Arduino Board

Also: USB input, power input, ICSP programming header

Arduino Board

Diagrammatic version, to simplify
But of a slightly older version of the board

Digital? Analog?

- Digital – only has two values: on/off
- Analog – has many (infinite) values

- Computers don't really do analog
- So they fake it, with *quantization*

Quantization = breaking up the analog range into bins. The number of bins is the resolution.
More bins = higher accuracy, but is more complex
Digital can be thought of as only two bins.

Arduino Software

That's the full code for blinking an LED, btw.
Arduino defines several useful functions like digitalWrite() and delay(). more on that later
Processing and Wiring not needed

Arduino & Processing

http://processing.org/

build generative art or other applets easily
not needed for Arduino, but can work with it

Arduino has essentially the same GUI as Processing
Easier than Arduino, since all software
Though similar UI and philosophy, Arduino is a different language
We'll use Processing later in the class to let the computer control Arduino & vice-versa

Installing Arduino

- Download software: http://arduino.cc/
  - Mac OS X PPC or Intel (must pick)
  - Windows 2000/XP
- Install drivers
  - In "drivers" folder, pick appropriate one
  - Windows: unzip driver, plug in board, setup
  - "macosx-setup-command" for Mac folk
- Reboot

Different Arduino downloads for each operating system
Different drivers for each OS too
"macosx-setup-command" must be run before reboot,
but, it will go away in next version

Using Arduino

- Programs are called “sketches”
- Load up example sketch “led_blink”

Errors

"Programmer is not responding"

Must select serial port

What's my serial port?

Mac: It's called "/dev/tty.usbserial-*something*"
Windows: Use Device Manager to find COM port

Using Arduino

```
void setup() {
  pinMode(ledPin, OUTPUT);      // sets t
}
void loop() {
  digitalWrite(ledPin, HIGH);   // sets t
  delay(1000);                  // waits
  digitalWrite(ledPin, LOW);    // sets t
  delay(1000);                  // waits
}
```

- Write program
- Compile (check for errors)
- Reset board
- Upload to board

Try it out with “led_blink”!

On reset, board will flash on-board pin 13 LED really fast for a split-second to indicate bootloader exists
When uploading, TX/RX lights will flash as data is transferred
Then the board resets, pin 13 will flash fast again
Finally, your program will run

Arduino Board Lifecycle

Making Circuits

heart pumps, blood flows

voltage pushes, current flows

LEDs

- LED = Light-Emitting Diode
  - electricity only flows one way in a diode
- Needs a "current limiting" resistor, or burns out

physical characteristics

schematic symbol

Many types of integrated lenses.
Some project a narrow beam (like the ones in this class), some project a very wide beam

LED flashlight

wiring diagram

schematic

All LED circuits are essentially this: power source, current limiter, LED

Flat part of LED goes to negative, like bar in schematic

The higher the resistance, the dimmer the LED; the lower, the brighter

LED flashlight

Take out solderless breadboard, resistor, LED, and battery and make a circuit
LEDs have been marked a little as to what color they are, but color doesn't matter here

Solderless Breadboards

Insert wires into holes to make a connection.
Much easier, quicker than soldering
But, they wear out, are expensive ($8 for this little one)

Blinky LED circuit

“hello world” of microcontrollers

wiring diagram

schematic

In schematics signals often flow from top-left to bottom-right
Common nodes like “gnd” are given their own symbol
Pick any digital pin to hook up to, doesn’t matter which

Blinky LED Software

You've already seen it.

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

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

void loop()
{
  digitalWrite(ledPin, HIGH);  // sets the LED on
  delay(1000);                 // waits for a second
  digitalWrite(ledPin, LOW);   // sets the LED off
  delay(1000);                 // waits for a second
}
```

Arduino Sketch Structure

- Declare variables at top
- Initialize
  - `setup()` – run once at beginning, set pins
- Running
  - `loop()` – run repeatedly, after `setup()`

Pins can be changed in loop() too, but conceptually easier in setup()

Arduino "Language"

- Language is standard C (but made easy)
- Lots of useful functions
  - `pinMode()` – set a pin as input or output
  - `digitalWrite()` – set a digital pin high/low
  - `digitalRead()` – read a digital pin's state
  - `analogRead()` – read an analog pin
  - `analogWrite()` – write an "analog" PWM value
  - `delay()` – wait an amount of time
  - `millis()` – get the current time
- And many others. And libraries. And examples!

Also: serial library, LCD library, servo examples

Development Cycle

- Make as many changes as you want
- Not like most web programming: edit → run
- Edit → compile → upload → run

ATmega8 & Arduino

Recap: Blinky LED

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

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

void loop()
{
  digitalWrite(ledPin, HIGH);   // sets the LED on
  delay(1000);                  // waits for a second
  digitalWrite(ledPin, LOW);    // sets the LED off
  delay(1000);                  // waits for a second
}
```

Recap: Programming

Edit

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

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

void loop()
{
  digitalWrite(ledPin, HIGH);   // sets the LED on
  delay(1000);                  // waits for a second
  digitalWrite(ledPin, LOW);    // sets the LED off
  delay(1000);                  // waits for a second
}
```

Compile

Upload

Reset

Digital Input

Most inputs you'll use are variations on switches

Switches make or break a connection

knife switch
(SPST)

toggle switch
(SPDT)

Fundamentally, they're all like the simple knife switch
Single pole = only one circuit is being controlled
Double pole = two circuits are being controlled at once
Single throw = only one path for circuit
Double throw = two potential paths for circuit

Digital Input

- 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?

Switch to Volts: Positive Logic

- 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

"pull-down"

Don't want "pull-down" to be too small, or it uses a lot of current

Switch to Volts: Inverted Logic

- 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"

Inverted logic like this is common in microcontrollers

Arduino Digital Input

- Add switch circuit to any digital input (except pin 13)
- For output, use either existing pin 13 LED or wire up your own

Arduino Digital Input

Output is on-board pin 13 LED for now

Using the fact that two of the switch leads are connected.
Also, notice color coding. Blue is ground, purple is signal
But pin 13 LED is underneath! So gotta take a peak.

Making Jumper Wires

- strip off about 1/2” of insulation
- Can use wire strippers, cutters, or fingers
- Can be a pain, so I have some pre-cut wires

Analog Input

- 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

- Arduino (ATmega8) has six ADC inputs
- (ADC = Analog to Digital Converter)
- Reads voltage between 0 to 5 volts
- Resolution is 10-bit (1024 states)
- In other words, 5/1024 = 4.8 mV smallest voltage change you can measure

Analog Input

Sure sure, but how to make a varying voltage?

With a *potentiometer*. Or just *pot*.

+5V–
measure–
gnd–

Color coding: red goes to power, blue to ground, purple to ‘measure here’ (it’s a mix, see?)

Potentiometers

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

And that's actually how it works, btw, if you take apart a pot.
But I might have the directions reversed (clockwise vs. anti-clockwise).

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

Sensing the Dark: Photocells

- aka. photoresistor, light-dependent resistor
- A *variable* resistor
- Brighter light == lower resistance
- Photocells you have range approx. 0-10k

schematic symbol

Pretty cheap too. Can get a grab bag of 100 misc from Jameco for $20

Photocell Circuit

Looks a lot like the pot circuit, doesn't it?

Photocell Arduino Sketch

Can use as before, sketch "analog_read_led"

Change to 0

```
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

Just like magic!
If circuit was configured the other way (photocell on bottom), then darkness would make it blink slower.

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

Can talk to not just computers.
Most things more complex than simple sensors/actuators speak serial.

Watch the TX/RX LEDS

- TX – sending to PC
- RX – receiving from PC
- Used when programming or communicating

(and keep an eye on that pesky pin13 LED too)

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 terminal from the olde days

Serial Communications

- "Serial" because data is broken down into bits, each sent one-by-one on a single wire:

```
'H'
   =   0 1 0 0 1 0 0 0
   =   L H L L H L L L
   =
```

- Toggle a pin to send data, just like blinking an LED
- Only a <u>single data wire</u> is needed to send data. One other to receive.

Note, a single <u>data</u> wire. You still need a ground wire.

Arduino & USB-to-serial

Arduino board is really two circuits

Original Arduino boards were RS-232 serial, not USB.

Arduino to Computer

USB is totally optional for Arduino
But it makes things easier

Original Arduino boards were RS-232 serial, not USB.

Pulse Width Modulation

- More commonly called “PWM”
- Computers can’t output analog voltages
  - Only digital voltages (0 volts or 5 volts)
- But you can fake it
  - if you average a digital signal flipping between two voltages.
- For example...

PWM

Output voltage is averaged from on *vs.* off time

output_voltage = (on_time / off_time) * max_voltage

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)

Controlling Arduino

- Any program on the computer, not just the Arduino software, can control the Arduino board
- On Unixes like Mac OS X & Linux, even the command-line can do it:

```
demo% export PORT=/dev/tty.usbserial-A3000Xv0
demo% stty -f $PORT 9600 raw -parenb -parodd cs8 -hupcl -cstopb clocal
demo% printf "1" > $PORT   # rotate servo left
demo% printf "5" > $PORT   # go to middle
demo% printf "9" > $PORT   # rotate servo right
```

Arduino PWM

why all the software, doesn't Arduino have PWM?

- Arduino has built-in PWM
- On pins 9,10,11
- Use `analogWrite(pin,value)`
- It operates at a high, fixed frequency (thus not usable for servos)
- But great for LEDs and motors
- Uses built-in PWM circuits of the ATmega8 chip -» no software needed

The PWM speed used for analogWrite() is set to 30 kHz currently.
When programming AVRs, PWM speed can be set to just about any value.

R,G,B LEDs

Three PWM outputs and *three* primary colors.
Just screams to be made, doesn't it?

With RGB you can make any color
(except black)

Put back on the ProtoShield for this.
Use either the 220 or 330 ohm resistors in your kit, if you don't have enough of one or the other
I have lots more 220 if you need them

R,G,B LEDs

Cut leads of resistors and LEDs to make for a more compact circuit.
Also, less likely to short against itself.

External Power

Arduino can run off USB power or external power

External power connector

USB connector

jumper switch to choose power source

External Power

You can use an AC adpater

Connector is standard barrel connector

Make sure it's "center positive"

Voltage can be 9-15V DC

Amps is > 200mA

Actually input voltage can be from like 7.5V to 35V, but don't go over 15V so the voltage regulator doesn't have to work so hard.

External Power

An easier way to connect a battery

also solves polarity concerns

Power connector input has protection diode.
Also it's easier with the connector

External Power

Battery life

How long does Arduino last on 9V battery?

- Arduino board draws about 40 mA by itself
- Each LED adds about 20mA when on
- Each servo maybe 100 mA when running
- Switches, pots, etc. are effectively zero

- Battery capacity rated in milliamp-hours (mAh)
- 9V batteries have about 400 mAh capacity

Thus, Arduino by itself lasts 400/40 = 10 hours

Take all your power, add it up, divide it into your battery capacity to get time in hours.
There are techniques to make an AVR chip go into sleep mode, and draw microamps (1/1000 mA), but those techniques don't have nice Arduino-style wrappers yet.
For more on batteries and their capacities: http://en.wikipedia.org/wiki/List_of_battery_sizes

Getting started with ARDUINO

BETA Version

Written by Massimo Banzi.
With materials written by Massimo Banzi, Erica Calogero, David Cuartielles, Jeff Gray, Tom Igoe, David Mellis and Cristian Nold.
Illustrations by Elisa Canducci.

/ analogue inputs

As we have seen in the previous section Arduino is able to detect if there is a voltage applied to one of its pins and report it trough the digitalRead function. This is fine in a lot of applications but the light sensor that we have used before it's also able to tell us not just if there is light or not, it's also able to tell us how much light there is. This is the difference between an on/off sensor (simply telling us if something is there or not) and an analogue sensor whose value continuously changes. In order to read this type of sensors we need a different type of pin. In the lower-right part of Arduino you'll see 6 pins marked "Analog In", these are special pins that not only can tell us if there is a voltage applied to them or not but also it value. By using the analogRead function we can read the voltage applied to one of the pins. This function returns a number between 0 and 1023 representing voltages between 0 and 5 volts. For example if there is a voltage of 2.5 volts applied to pin 0 writing analogRead(0) will return 512 etc etc.

If you now build the circuit that you see in the illustration by using a 10k or 4.7k resistor and you run the piece of code you find here you'll see the led blinking at a rate that's dependent on the amount of light that hits the sensor.

A B C D E
F G H I J
GND
13
7
5V Gnd

/ the breadboard

The process of getting a circuit to work is largely based on making lots of changes to it until it behaves properly; it's a very fast iterative process that could be seen as the electronic equivalent to sketching. The design evolves in your hands as you try different combinations. In order to achieve the best results you want to use a system that will allow you to change the connections between components in the fastest, most practical and non-destructive way.

This requirement clearly rules out soldering, it's a time consuming procedure that puts every component under stress every time you heat them up and cool them down.

The answer to our problems comes from a very practical device called "Solder-less Breadboard".

As you can see from the picture it's a small plastic board full of holes, each one of them contains a spring-loaded contact. You can push a component's leg into one of the hole and it will establish an electrical connection with all the other holes in the same vertical column of holes. Each hole is at a distance of 2.54 mm distance from the others, since most of the components have their legs, known to techies as pins, are spaced at that standard distance therefore chips with multiple legs will fit nicely. Not all the contacts on a breadboard are created equal, there are some differences: the topmost and bottom row (coloured in red and blue and aptly marked with + and -) are

/ what is electricity

If you have ever done any plumbing at home, electronics won't be a problem for you to understand. Jokes aside, in order to understand how electricity and electric circuits work the best way is to build a mental model called the "water analogy". Let's take a simple device like a portable fan,

if you take it apart you will see that it contains a small battery a couple of wires going to an electric motor and one of the wires is interrupted by a switch. Now makes sure you have a new battery fitted in the device and activate the switch; the motor will start to spin providing the necessary refreshment. How does this work? Well imagine that the battery is a water pump and the switch is a tap while the motor is one of those wheels you see in watermills, when you open the tap water will flow from the pump and push the wheel into motion.

Now in this simple hydraulic system two parameters are important: the pressure of the water (this is given from how powerful is the pump) and the amount of water that will flow in the pipes (this depends from the size of the pipes and the resistance that the wheel will oppose to the stream of water hitting it).

You quickly understand that if you want the wheel to spin faster you need to increase the size of the pipes (but this works only up to a point) and increase the pressure that the pump can achieve. Increasing the size of the pipes allows more flow of water to go through them, effectively by making them bigger we have reduced the resistance they oppose to the flow of water. This works until a certain point where the wheel won't spin any faster because the pressure of the water is not strong enough and this is when we need the pump to be stronger.

This can go on until the point when the wheel falls apart because the water flow is too strong and destroys it. Another thing you will notice is that as the wheel spins the axle will heat up a little bit, this is because no matter how good is the way we have mounted the wheel the attrition between the axle and the holes it is mounted in will generate heat. This is important to understand that in a system like this not all the energy you pump into the system will be converted into movement, some will be lost in a number of inefficiencies and will generally show up as heat emanating from some parts of the system.

So what are the important parts of the system as we described it before? The pressure produced by the pump is one, the resistance that the pipes and wheel oppose to the flow of water and the actual flow of water (let's say that this is represented by the number of litres of water that flow in one second) are the others.

Without going into much details electricity works a bit like water, you have a kind of pump (any source of electricity like a battery or a wall plug) pushes electric charges (imagine them like "drops" of electricity) down pipes represented by the wires where some devices are able to use them to produce heat (your grandma's thermal blanket)

light (your bedroom's lamp) sound (your stereo) movement (your fan) and much more.

So when you read on a battery 9V you can imagine the Voltage of the battery like the water pressure that can be potentially produced by this little "pump". This is measured in Volts from Alessandro Volta, the inventor of the first battery.

The flow of water has got and electric equivalent called "current" that is measured in Amperes from Andre Marie Ampere. Finally the resistance opposed to the flow of current by any means it travels through is called, yes you guessed right, resistance and it's measured in Ohms from the German physicist Ohm.

Mr ohm is also responsible for coming up with the most important law in electricity and the only formula you will really need to remember.

He was able to demonstrate that in a circuit the Voltage, the Current and the Resistance are all related to each other and in particular that the resistance opposed by the circuits determines the amount of current that will flow through it give a certain supply voltage.

It's very intuitive if you think about it: Take a 9V battery and plug it into a simple circuit while measuring current, you will find that the more resistors you add to the circuit the less current will travel through it. Going back to the water flowing in pipes, given a certain pump if I place a tap (which we can assimilate to a variable resistor in electricity) the more I close the tap, increasing resistance to water flow, less water will flow through the pipes. Mr Ohm summarised his law into this formulas:

R (resistance) = V (voltage) / I (current)

$V = R * I$

$I = V / R$

This is the only rule that you really have to memorise and learn to use, because in most of your work this will be the only one you will really need.

connected horizontally and are used to carry the power across the board so that when we need power or ground we can provide it very quickly with a short jumper (this is not a sweater or a funny insect but a short piece of wire used to connect two points in the circuits) The last this you need to know about breadboards is the in the middle there is a large gap that is as wide as the size of a small chip. This shows that the each vertical line of holes is interrupted in the middle so that when you plug a chip you won't short circuit pins that are on the two sides of the chip, clever eh?

/ patching

Robert Moog built his analogue synthesizers in a modular fashion and the musician could try endless combination by "Patching" together different modules with cables. This made the synthesizer look like an old telephone switch but, combined with the numerous knobs, was the perfect platform for tinkering with sound and innovating music. This technique has been translated into the world of software by programs like Max or Pure Data.

/ circuit bending

Circuit bending is one of the most interesting forms of tinkering.
It's the creative short-circuiting of low voltage, battery-powered electronic audio devices such as guitar effects, children's toys and small synthesizers to create new musical instruments and sound generators. The heart of this process is "the art of chance". It began in 1966 when Reed Ghazala by chance, shorted-out a toy amplifier against a metal object in his desk drawer, resulting in a stream of unusual sounds.

Thanks

The Arduino team is composed of:
Massimo Banzi, David Cuartielles, Tom Igoe, David Mellis and Gianluca Martino.

The Arduino team would like to thank the following people and institutions for the support in making this booklet:

Aliadi Cortelletti for typesetting the booklet.

Barbara Ghella, Stefano Mirti, Claudio Moderini.

Interaction Design Lab, Milano
Domus Academy, Milano
Interaction Design Institute, Milano
Malmö University, Faculty of Art, Culture and Communication (K3)

This booklet is released under a Creative Commons License:
Attribution-NonCommercial-ShareAlike 2.5

You are free:
* to copy, distribute, display, and perform the work
* to make derivative works

Under the following conditions:
* You must attribute the work in the manner specified by the author or licensor.
* You may not use this work for commercial purposes.
* If you alter, transform, or build upon this work, you may distribute the resulting work only under a license identical to this one.
* For any reuse or distribution, you must make clear to others the license terms of this work.
* Any of these conditions can be waived if you get permission from the copyright holder.
http://creativecommons.org/licenses/by-nc-sa/2.5/deed.en