Desktop Fan Project for the Arduino Inventors Kit

ME 120

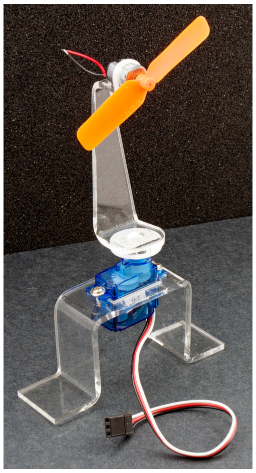
Gerald Recktenwald
Portland State University
gerry@pdx.edu

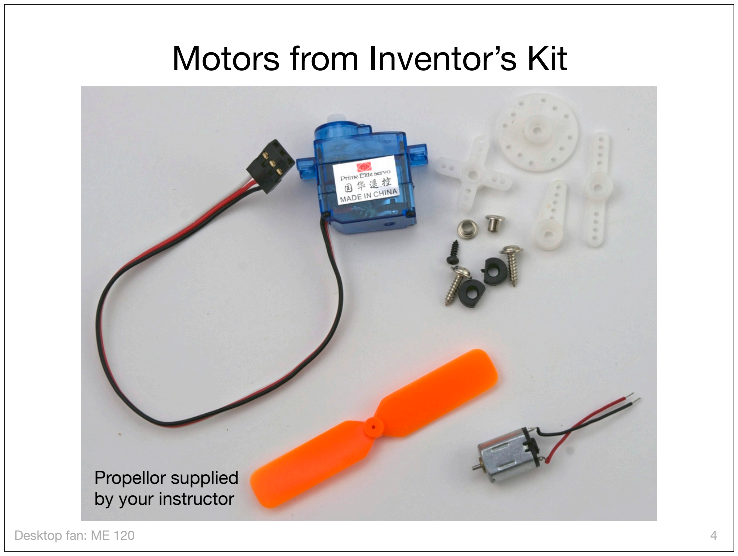
Goal

- Build a desktop fan from parts in the Arduino Inventor’s Kit
- Work in teams of two
- Learn new skills
  - Controlling a servo and DC motor
  - Make a 2D drawing with Solidworks
  - Send drawings to Laser cutter
  - Soldering
- Due in two weeks
  - In-class demonstration of your working fan

Tasks

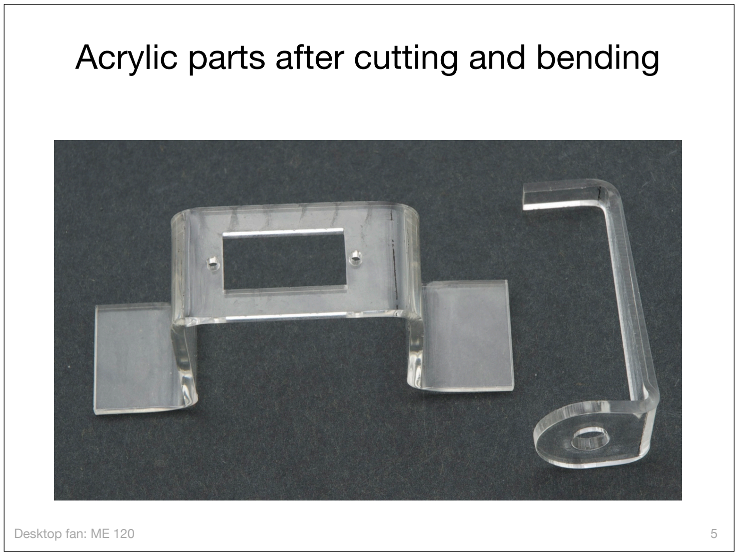
- Measure servo and DC motors
  - prepare for structural design
  - learn how to use your calipers
- Sketch design of support structure on paper
- Create Solidworks model of the base and DC motor support
- Cut acrylic parts
- Re-solder the DC motor leads
- Assemble the system
- Write Arduino program to



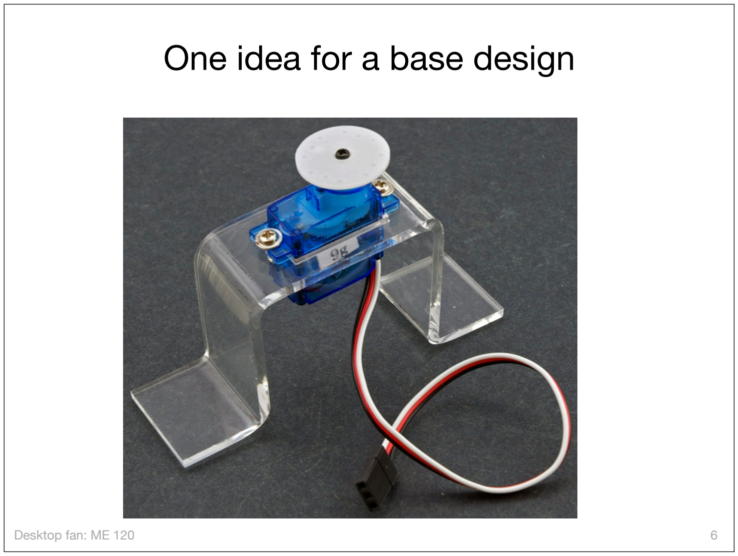


Motors from Inventor’s Kit

Propellor supplied
by your instructor

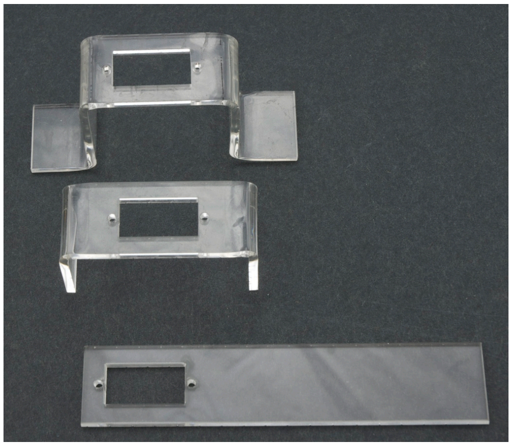


Acrylic parts after cutting and bending

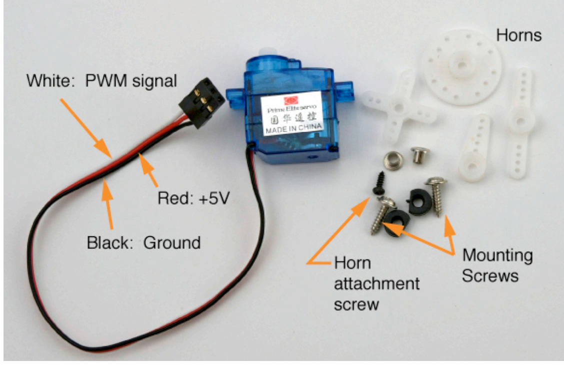


One idea for a base design

Alternative base designs



Servo motor parts



Fan Project: First Steps

1. Make a hand sketch of the structural parts
2. Measure the servo and mounting screws
3. Use measurements to add dimensions to the sketch
4. Redraw the sketch as a 2D “flat” drawing in Solidworks
5. Email the drawing to the instructor
    a. Laser cutter works on thin sheets in 2D
    b. Use the acrylic bender after parts are cut

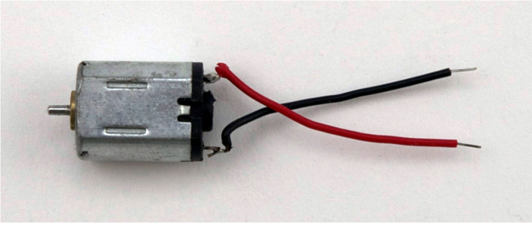
Watch this video to see the laser cutter and acrylic bender in action:
http://www.youtube.com/watch?v=DJA8EmBUfLo

Soldering Leads to the DC Motor

Desktop fan project
ME 120

Overview

The DC motor that comes with the Arduino Inventor’s Kit has short and delicate leads. We need to replace the leads with more robust wiring and soldered connections



Temperature-controlled soldering iron and flux

64-022
RadioShack
TECHNOLOGY PLUS
Non-Spill Paste
ROSIN SOLDERING FLUX
RELOJ DE PROYECCIÓN CONTROLADO
WARNING: EYE AND SKIN IRRITANT. KEEP OUT OF REACH OF CHILDREN. Read carefully other warnings on back.
Instructions: Clean surfaces to be soldered. Apply flux. Apply heat, melting flux, and apply solder. Wipe off residue.
PEEL
Net Wt.: 2 oz. (56.6g)

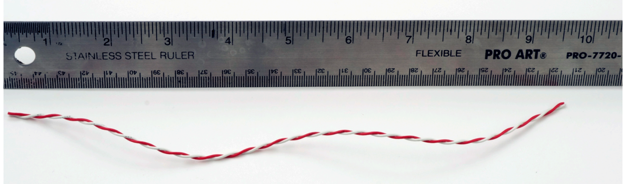
Soldering work surface with vise

Helping Hands

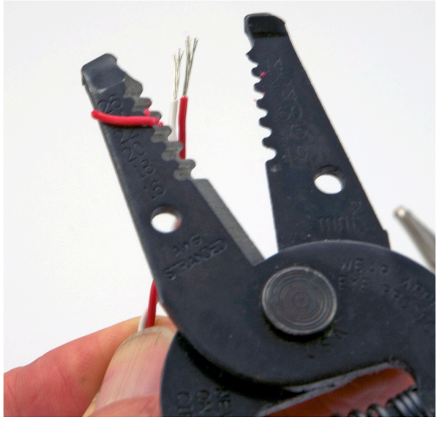
Procedure

1. Cut a length of wire
2. Strip and tin the ends of the wire
3. Make note of polarity
4. Remove (by desoldering) leads from DC motor
5. Insert tinned wire through tabs and bend into position
6. Secure new leads by soldering to motor tabs

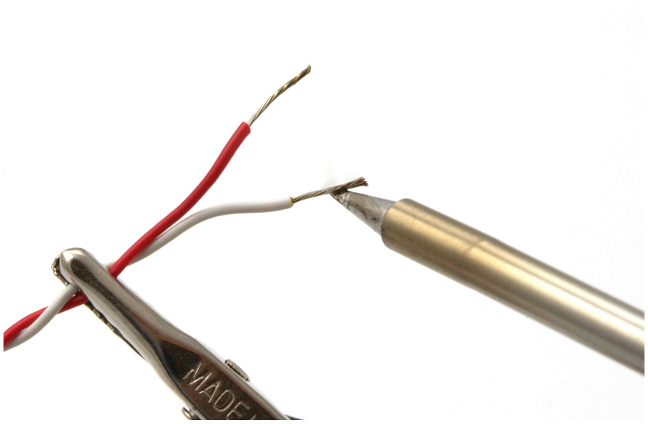
Cut new lead wires



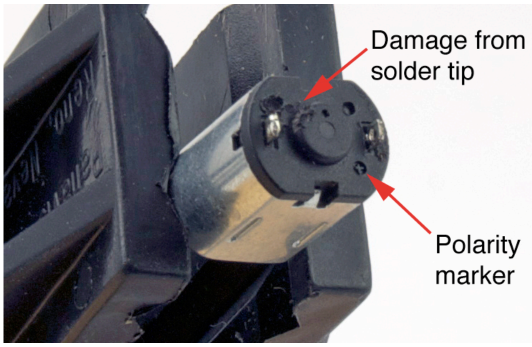
Strip the leads



Tin the leads



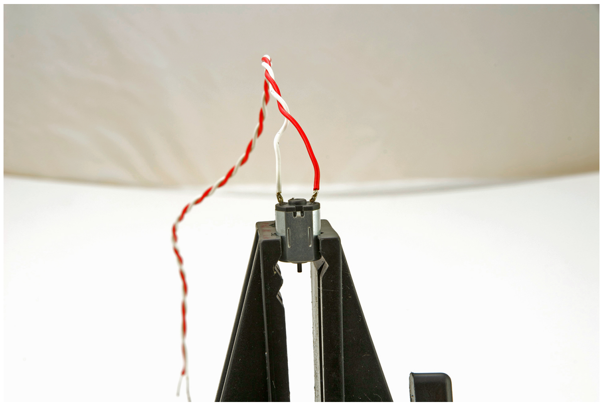
Remove the old leads and note the



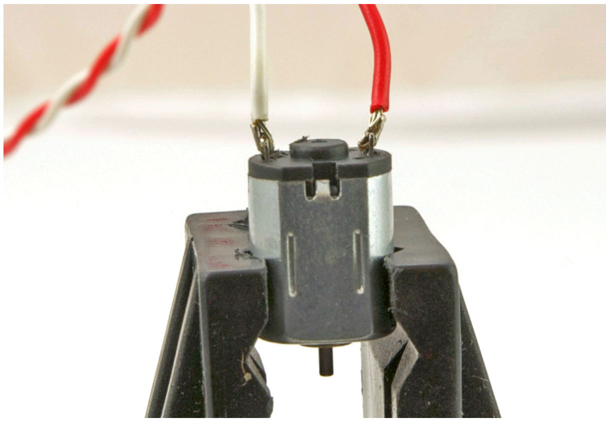
Damage from solder tip

Polarity marker

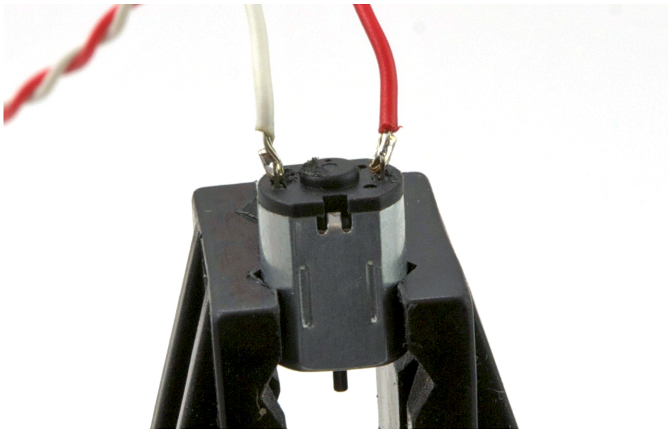
Motor supported. Extension wires in place



Bend the tinned wires around the



Secure the leads with solder

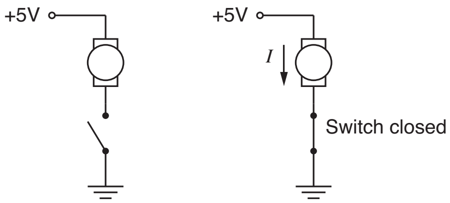


Basic DC Motor Circuits

Desktop fan project
ME 120

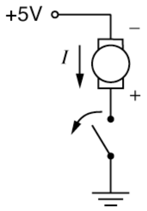
Simplest DC Motor Circuits

Connect the motor to a DC power supply



Current continues after switch is opened

Opening the switch does not immediately stop current in the motor windings.

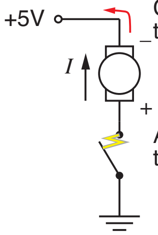


Inductive behavior of the motor causes current to continue to flow when the switch is opened suddenly.

Charge builds up on what was the negative terminal of the motor

Reverse current

Charge build-up can cause damage



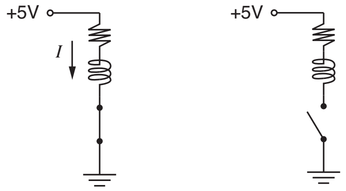
Current surge through the voltage supply

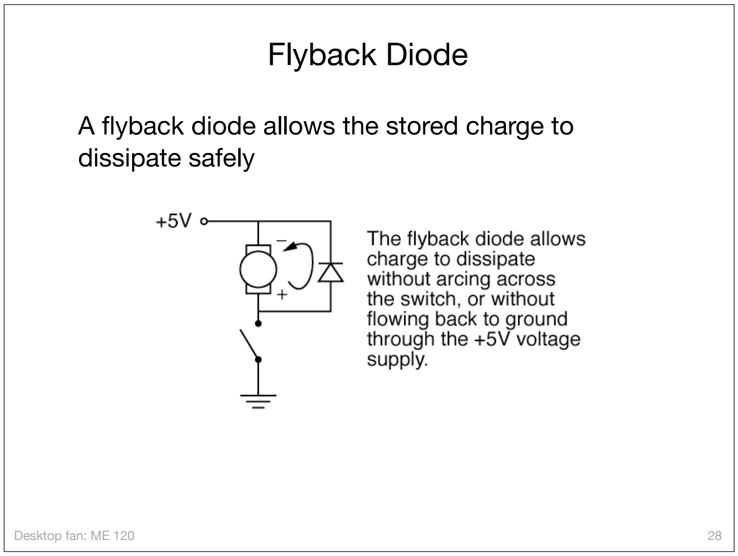
Arc across the switch

Motor Model

Simple model of a DC motor:

- Windings have inductance and resistance
- Inductor causes a storage of electrical charge in the windings
- We need to provide a way to safely dissipate the charge stored in the motor windings

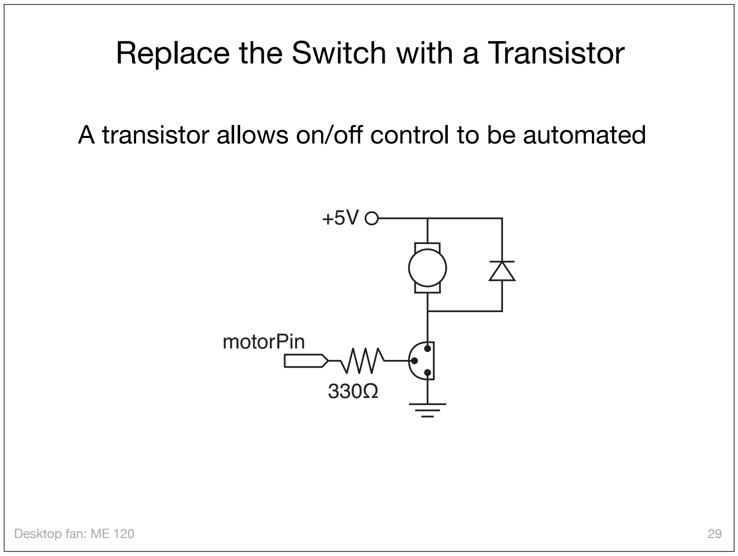




Flyback Diode

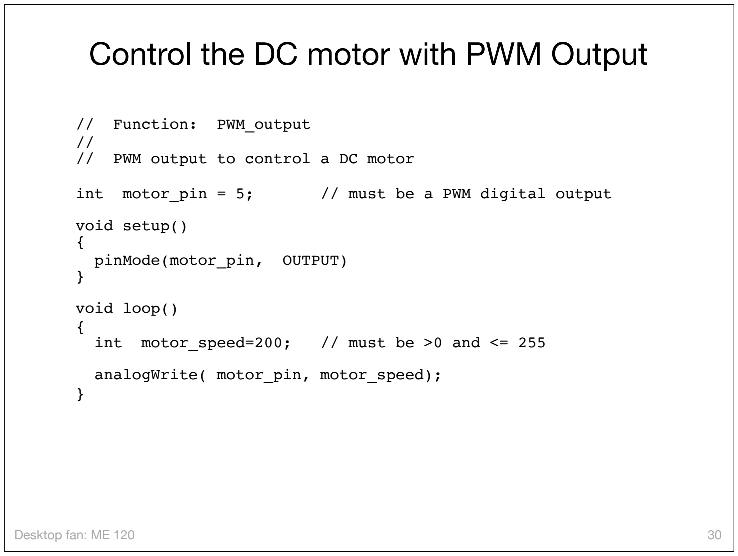
A flyback diode allows the stored charge to dissipate safely

The flyback diode allows charge to dissipate without arcing across the switch, or without flowing back to ground through the +5V voltage supply.



Replace the Switch with a Transistor

A transistor allows on/off control to be automated



Control the DC motor with PWM Output

```
//  Function:  PWM_output
//
//  PWM output to control a DC motor

int  motor_pin = 5;       // must be a PWM digital output

void setup()
{
  pinMode(motor_pin,  OUTPUT)
}

void loop()
{
  int  motor_speed=200;   // must be >0 and <= 255

  analogWrite( motor_pin, motor_speed);
}
```

Arduino Programming: PWM Control of DC motor speed

Desktop fan project
ME 120

Overview

Part I

- Circuits and code to control the speed of a small DC motor.
- Use potentiometer for dynamic user input.
- Use PWM output from an Arduino to control a transistor.
- Transistor acts as variable voltage switch for the DC motor.

Part II

- Consolidate code into reusable functions.
- One function maps 10-bit analog input to 8-bit PWM output.
- Another function controls the motor speed.
- Functions developed here are useful for more complex control tasks, e.g. the desktop fan project.

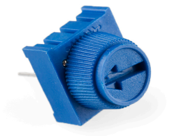
Part 1: Control motor speed with a pot

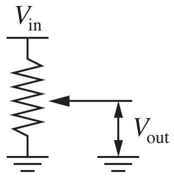
Increase complexity gradually. Test at each stage.

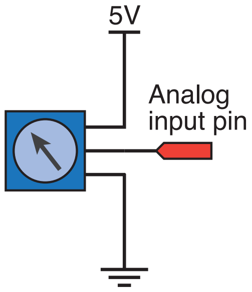
1. Use a potentiometer to generate a voltage signal
   a. Read voltage with analog input
   b. Print voltage to serial monitor to verify
2. Convert 10-bit voltage scale to 8-bit PWM scale
   c. Voltage input is in the range 0 to 1023
   d. PWM output needs to be in the range 0 to 255
   e. Print voltage to serial monitor to verify
3. Connect PWM output to DC motor
4. Write a function to linearly scale the data
5. Write a function to update the motor

Potentiometer Circuit

Use the potentiometer from the Arduino Inventor’s



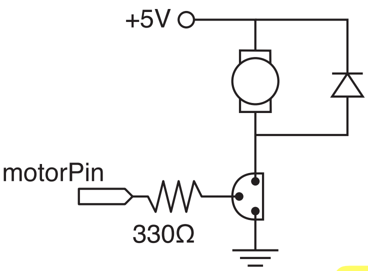




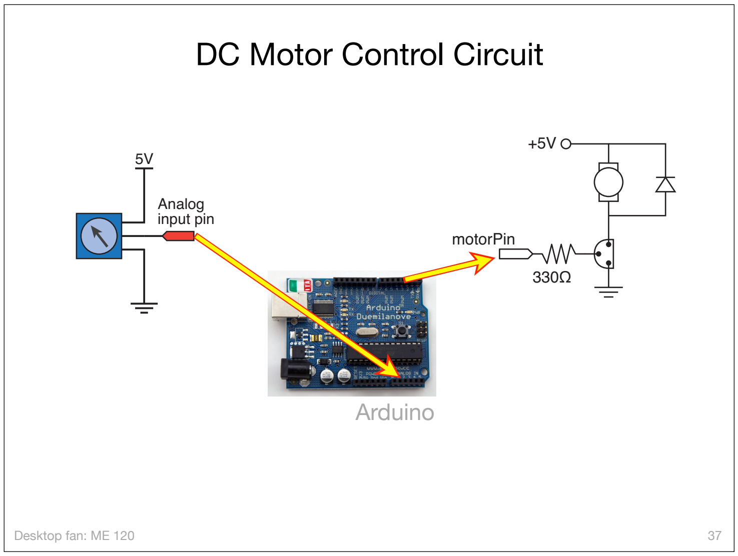
Code to print potentiometer reading

```
//  Function:  read_potentiometer
//
//  Read a potentiometer and print the reading

int  sensor_pin = 3;    //  Wire sweeper of pot to
                        //  analog input pin 3
void setup()
{
  Serial.begin(9600);
}

void loop()
{
  int  val;

  val = analogRead( sensor_pin );
  Serial.print("reading = ");
  Serial.println( val );
}
```

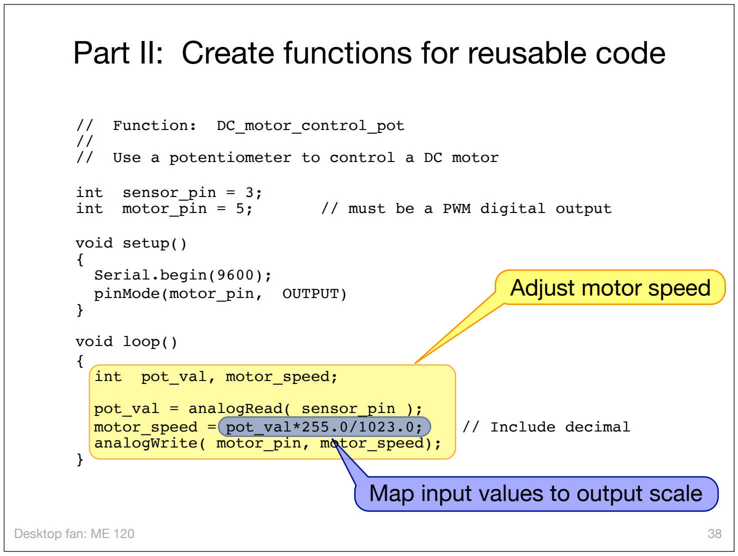
DC Motor Control Circuit



Add this to the breadboard with the potentiometer circuit



DC Motor Control Circuit

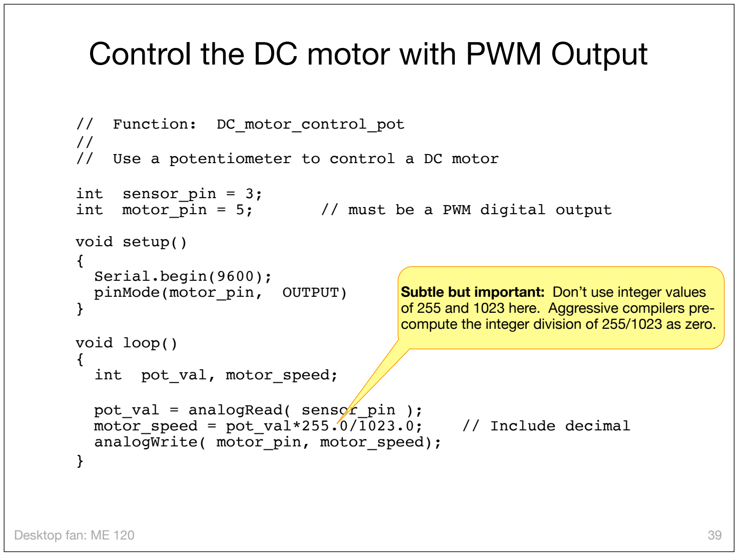


Part II: Create functions for reusable code

```
//  Function:  DC_motor_control_pot
//
//  Use a potentiometer to control a DC motor

int  sensor_pin = 3;
int  motor_pin = 5;       // must be a PWM digital output

void setup()
{
  Serial.begin(9600);
  pinMode(motor_pin,  OUTPUT)
}

void loop()
{
  int  pot_val, motor_speed;

  pot_val = analogRead( sensor_pin );
  motor_speed = pot_val*255.0/1023.0;    // Include decimal
  analogWrite( motor_pin, motor_speed);
}
```

Adjust motor speed

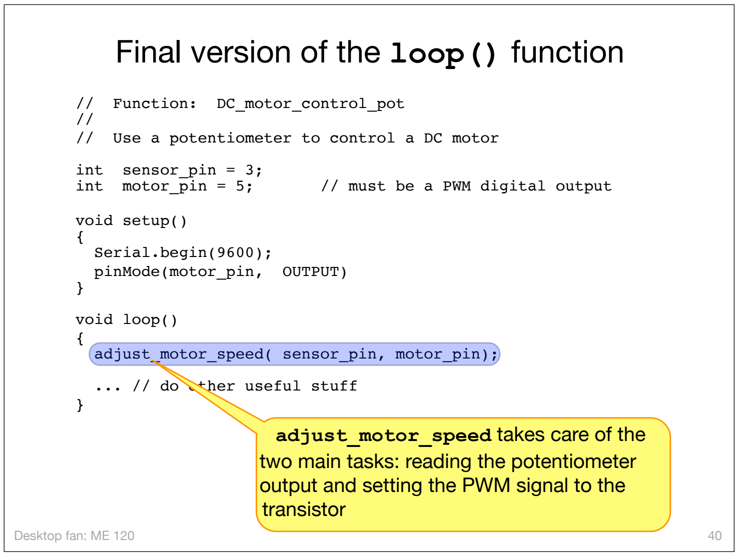
Map input values to output scale



Control the DC motor with PWM Output

```
//  Function:  DC_motor_control_pot
//
//  Use a potentiometer to control a DC motor

int  sensor_pin = 3;
int  motor_pin = 5;       // must be a PWM digital output

void setup()
{
  Serial.begin(9600);
  pinMode(motor_pin,  OUTPUT)
}

void loop()
{
  int  pot_val, motor_speed;

  pot_val = analogRead( sensor_pin );
  motor_speed = pot_val*255.0/1023.0;    // Include decimal
  analogWrite( motor_pin, motor_speed);
}
```

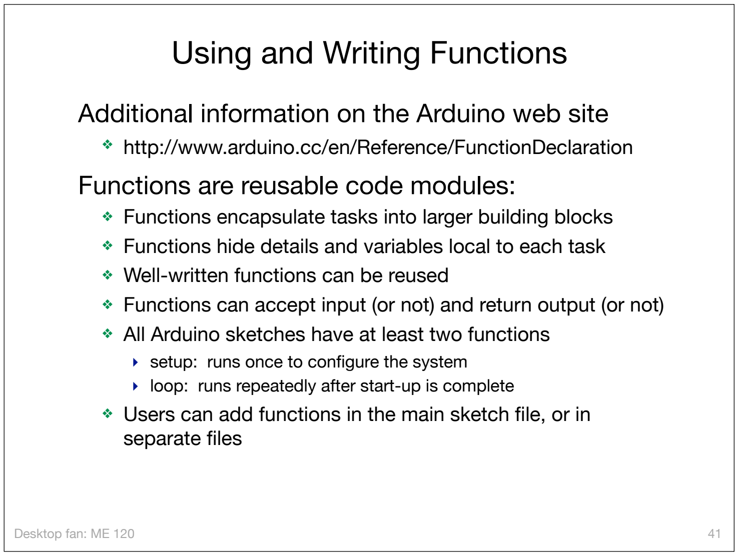
Subtle but important: Don't use integer values of 255 and 1023 here. Aggressive compilers pre-compute the integer division of 255/1023 as zero.



Final version of the `loop()` function

```
//  Function:  DC_motor_control_pot
//
//  Use a potentiometer to control a DC motor

int  sensor_pin = 3;
int  motor_pin = 5;       // must be a PWM digital output

void setup()
{
  Serial.begin(9600);
  pinMode(motor_pin,  OUTPUT)
}

void loop()
{
  adjust_motor_speed( sensor_pin, motor_pin);

  ... // do other useful stuff
}
```

adjust_motor_speed takes care of the two main tasks: reading the potentiometer output and setting the PWM signal to the transistor



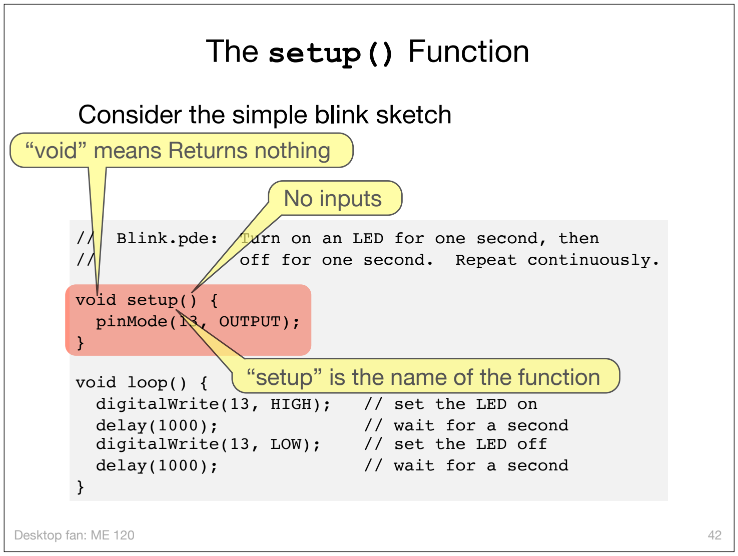
Using and Writing Functions

Additional information on the Arduino web site

- http://www.arduino.cc/en/Reference/FunctionDeclaration

Functions are reusable code modules:

- Functions encapsulate tasks into larger building blocks
- Functions hide details and variables local to each task
- Well-written functions can be reused
- Functions can accept input (or not) and return output (or not)
- All Arduino sketches have at least two functions
  - setup: runs once to configure the system
  - loop: runs repeatedly after start-up is complete
- Users can add functions in the main sketch file, or in separate files



The `setup()` Function

Consider the simple blink sketch

"void" means Returns nothing

No inputs

```
//  Blink.pde:  Turn on an LED for one second, then
//              off for one second.  Repeat continuously.

void setup() {
  pinMode(13, OUTPUT);
}

void loop() {
  digitalWrite(13, HIGH);   // set the LED on
  delay(1000);              // wait for a second
  digitalWrite(13, LOW);    // set the LED off
  delay(1000);              // wait for a second
}
```

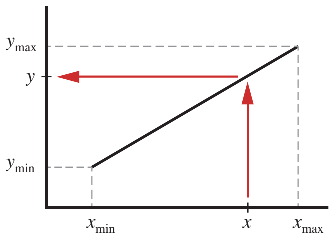
"setup" is the name of the function

A Function to Translate Linear Scales

Linear scaling from x values to y values:

$y = f(x)$

where f is a linear



$$\frac{y - y_{\min}}{y_{\max} - y_{\min}} = \frac{x - x_{\min}}{x_{\max} - x_{\min}}$$

$$\Longrightarrow \quad y = y_{\min} + (y_{\max} - y_{\min})\frac{x - x_{\min}}{x_{\max} - x_{\min}}$$

In words: Given x, x_{min}, x_{max}, y_{min}, and y_{max},

A Function to Translate Linear Scales

Enter the code at the bottom into your sketch

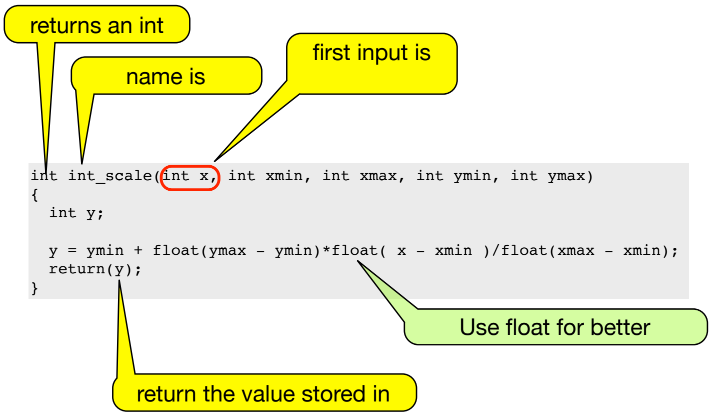
- The code is *not* inside any other program block (like setup or void)

How would you test that this function is working?

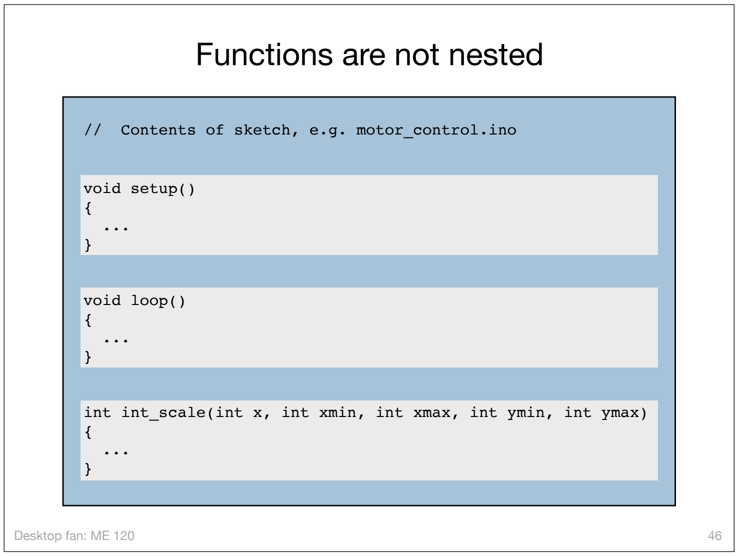
```
int int_scale(int x, int xmin, int xmax, int ymin, int ymax)
{
  int y;

  y = ymin + float(ymax - ymin)*float( x - xmin )/float(xmax - xmin);
  return(y);
}
```

N.B. This code is essentially a reimplementation of the built-in map function.
See http://arduino.cc/en/Reference/Map

A Function to Translate Linear Scales

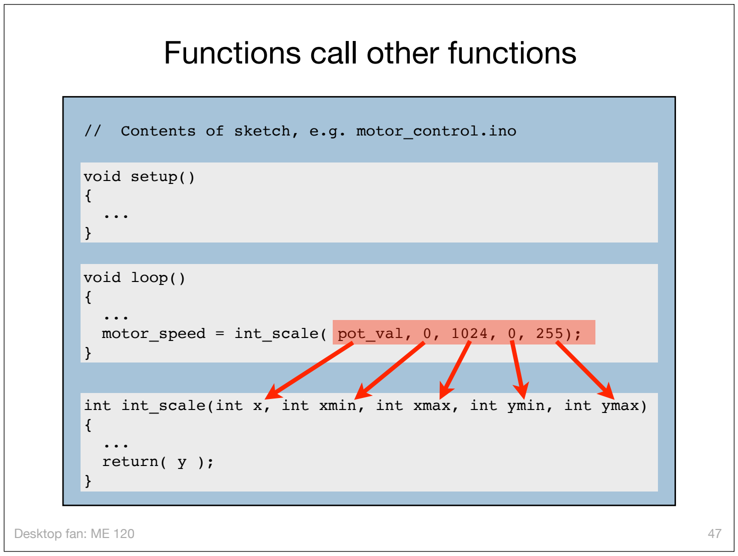


```
int int_scale(int x, int xmin, int xmax, int ymin, int ymax)
{
  int y;

  y = ymin + float(ymax - ymin)*float( x - xmin )/float(xmax - xmin);
  return(y);
}
```



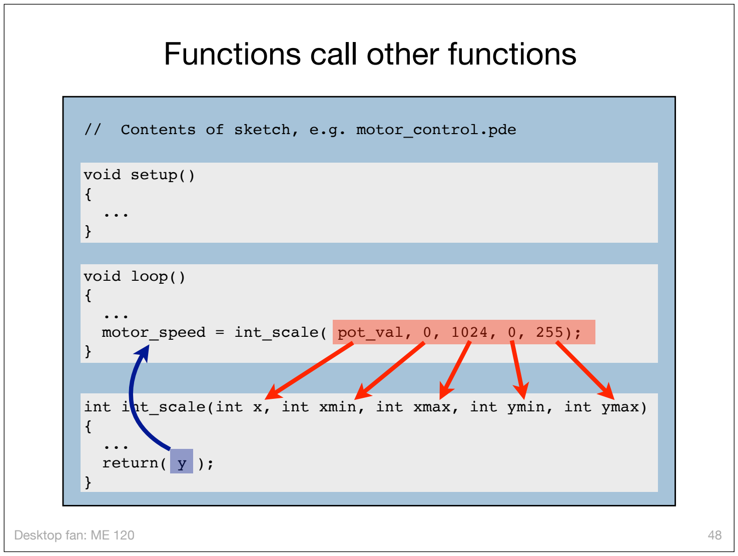
Functions are not nested

```
//  Contents of sketch, e.g. motor_control.ino

void setup()
{
  ...
}

void loop()
{
  ...
}

int int_scale(int x, int xmin, int xmax, int ymin, int ymax)
{
  ...
}
```



Functions call other functions

```
//  Contents of sketch, e.g. motor_control.ino

void setup()
{
  ...
}

void loop()
{
  ...
  motor_speed = int_scale( pot_val, 0, 1024, 0, 255);
}

int int_scale(int x, int xmin, int xmax, int ymin, int ymax)
{
  ...
  return( y );
}
```



Functions call other functions

```
//  Contents of sketch, e.g. motor_control.pde

void setup()
{
  ...
}

void loop()
{
  ...
  motor_speed = int_scale( pot_val, 0, 1024, 0, 255);
}

int int_scale(int x, int xmin, int xmax, int ymin, int ymax)
{
  ...
  return( y );
}
```

Use the int_scale function

```
//  Function:  DC_motor_control_pot
//
//  Use a potentiometer to control a DC motor

int  sensor_pin = 3;
int  motor_pin = 5;       // must be a PWM digital output

void setup()
{
  Serial.begin(9600);
  pinMode(motor_pin,  OUTPUT)
}

void loop()
{
  int  pot_val, motor_speed;

  pot_val = analogRead( sensor_pin );
  motor_speed = int_scale( pot_val, 0, 1024, 0, 255;
  analogWrite( motor_pin, motor_speed);
}

int int_scale(int x, int xmin, int xmax, int ymin, int ymax)
{
  int y;

  y = ymin + float(ymax - ymin)*float( x - xmin )/float(xmax - xmin);
  return(y);
}
```

A Function to update motor speed

Inputs

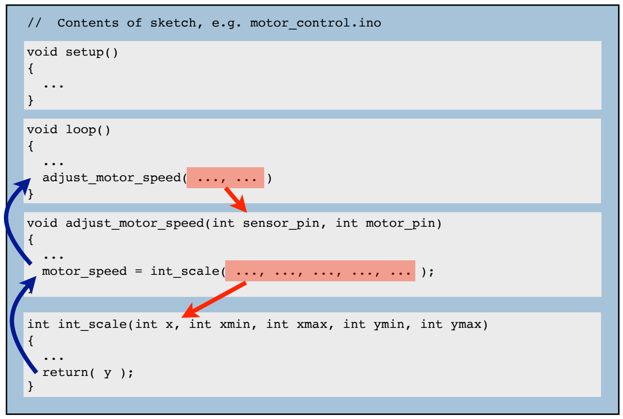
- sensor pin
- motor output pin

Tasks:

- Read potentiometer voltage
- Convert voltage from 10 bit to 8 bit scales
- Change motor speed

```
void adjust_motor_speed(int sensor_pin, int motor_pin)
{
  int  motor_speed, sensor_value;

  sensor_value = analogRead(sensor_pin);
  motor_speed = int_scale(sensor_value, 0, 1024, 0, 255);
  analogWrite( motor_pin, motor_speed);

  Serial.print("Pot input, motor output = ");
  Serial.print(sensor_value);
  Serial.print(" ");  Serial.println(motor_speed);
}
```

Functions call functions, call functions, ...



```
//  Contents of sketch, e.g. motor_control.ino

void setup()
{
  ...
}

void loop()
{
  ...
  adjust_motor_speed( ..., ... )
}

void adjust_motor_speed(int sensor_pin, int motor_pin)
{
  ...
  motor_speed = int_scale( ..., ..., ..., ..., ... );

int int_scale(int x, int xmin, int xmax, int ymin, int ymax)
{
  ...
  return( y );
}
```

Button Input:
On/off state change

Desktop fan project
ME 120

User input features of the fan

- Potentiometer for speed control
  - Continually variable input makes sense for speed control
  - Previously discussed
- Start/stop
  - Could use a conventional power switch
  - Push button (momentary) switch
- Lock or limit rotation angle
  - Button click to hold/release fan in one position
  - Potentiometer to set range limit

Conventional on/off switch

Basic light switch or rocker switch

- Makes or breaks connection to power
- Switch stays in position: On or Off
- Toggle position indicates the state
- NOT in the Arduino Inventors Kit

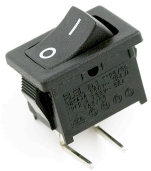


Image from sparkfun.com

Image from lowes.com

Momentary or push-button switches

- Temporary “click” input
  - Two types: normally closed or normally open
- Normally open
  - electrical *contact is made* when button is pressed
- Normally closed
  - electrical *contact is broken* when button is pressed
- Internal spring returns button to its un-pressed state

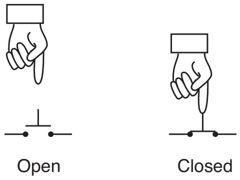
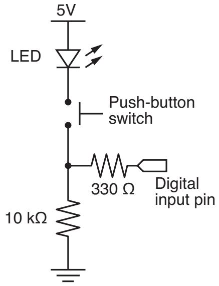


Image from sparkfun.com

Momentary Button and LED Circuit

Digital input with a *pull-down resistor*

- When switch is open (button not pressed):
  - Digital input pin is tied to ground
  - No current flows, so there is no voltage difference from input pin to ground
  - Reading on digital input is LOW
- When switch is closed (button is pressed):
  - Current flows from 5V to ground, causing LED to light up.
  - The 330Ω resistor limits the current draw by the input pin.
  - The 10k resistor causes a large voltage drop between 5V and ground, which causes the digital input pin to be closer to 5V.
  - Reading on digital input is HIGH



Programs for the LED/Button Circuit

1. Continuous monitor of button state
   - Program is completely occupied by monitoring the button
   - Used as a demonstration — not practically useful
2. Wait for button input
   - Blocks execution while waiting
   - May be useful as a start button
3. Interrupt Handler
   - Most versatile
   - Does not block execution
   - Interrupt is used to change a flag that indicates state
   - Regular code in loop function checks the sate of the flag

All three programs use the same electrical circuit

Continuous monitor of button state

```
int  button_pin = 4;           // pin used to read the button

void setup() {
  pinMode( button_pin, INPUT);
  Serial.begin(9600);          // Button state is sent to host
}

void loop() {
  int button;

  button = digitalRead( button_pin );
  if ( button == HIGH ) {
    Serial.println("on");
  } else {
    Serial.println("off");
  }
}
```

Serial monitor shows a continuous stream of “on” or “off”

This program *does not* control the LED

Wait for button input

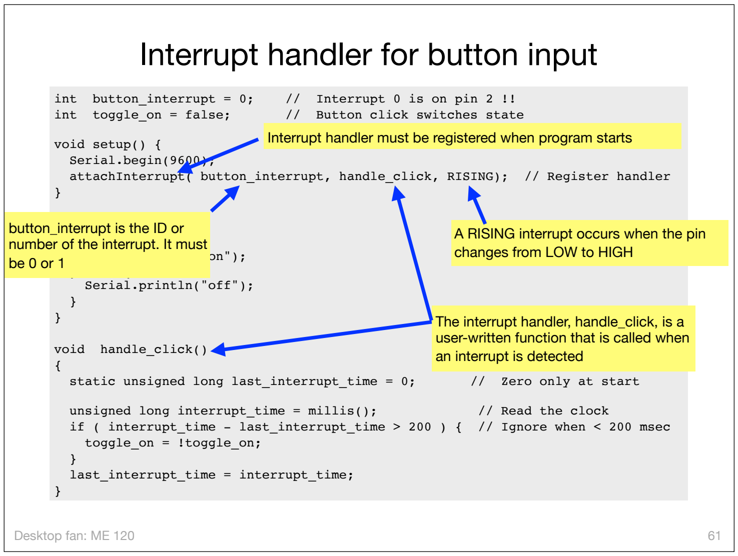
```
int  button_pin = 4;           // pin used to read the button

void setup() {
  int start_click = LOW;        //  Initial state: no click yet

  pinMode( button_pin, INPUT);
  Serial.begin(9600);
  while ( !start_click ) {
    start_click = digitalRead( button_pin );
    Serial.println("Waiting for button press");
  }
}

void loop() {
  int button;

  button = digitalRead( button_pin );
  if ( button == HIGH ) {
    Serial.println("on");
  } else {
    Serial.println("off");
  }
}
```

while loop continues as long as start_click is FALSE

Same loop() function as before

Interrupt handler for button input

```
int  button_interrupt = 0;    //  Interrupt 0 is on pin 2 !!
int  toggle_on = false;       //  Button click switches state

void setup() {
  Serial.begin(9600);
  attachInterrupt( button_interrupt, handle_click, RISING);  // Register handler
}

void loop() {
  if ( toggle_on ) {
    Serial.println("on");
  } else {
    Serial.println("off");
  }
}

void  handle_click()
{
  static unsigned long last_interrupt_time = 0;       //  Zero only at start

  unsigned long interrupt_time = millis();             // Read the clock
  if ( interrupt_time - last_interrupt_time > 200 ) {  // Ignore when < 200 msec
    toggle_on = !toggle_on;
  }
  last_interrupt_time = interrupt_time;
}
```



Interrupt handler for button input

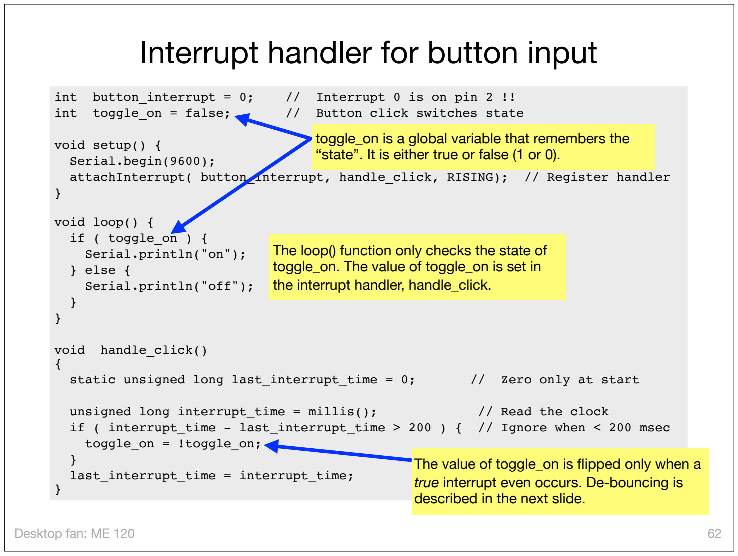
```
int  button_interrupt = 0;    //  Interrupt 0 is on pin 2 !!
int  toggle_on = false;       //  Button click switches state

void setup() {
  Serial.begin(9600);
  attachInterrupt( button_interrupt, handle_click, RISING);  // Register handler
}

                                     on");

    Serial.println("off");
  }
}

void  handle_click()
{
  static unsigned long last_interrupt_time = 0;       //  Zero only at start

  unsigned long interrupt_time = millis();             // Read the clock
  if ( interrupt_time - last_interrupt_time > 200 ) {  // Ignore when < 200 msec
    toggle_on = !toggle_on;
  }
  last_interrupt_time = interrupt_time;
}
```

Interrupt handler must be registered when program starts

button_interrupt is the ID or number of the interrupt. It must be 0 or 1

A RISING interrupt occurs when the pin changes from LOW to HIGH

The interrupt handler, handle_click, is a user-written function that is called when an interrupt is detected



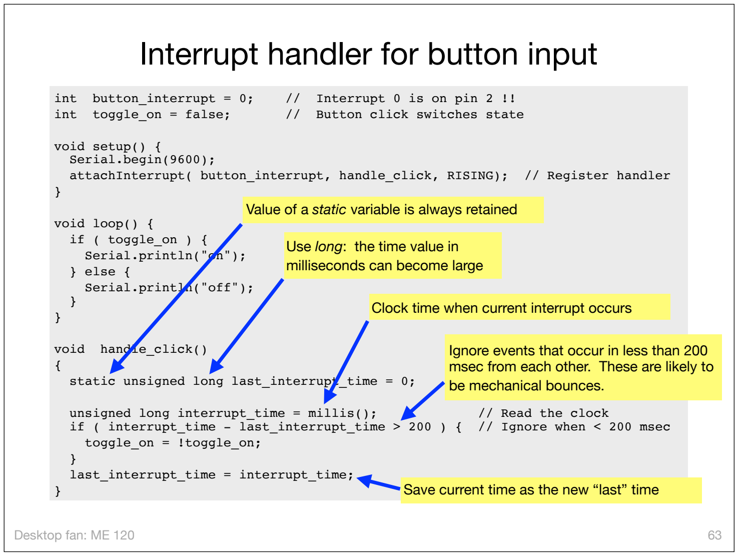
Interrupt handler for button input

```
int  button_interrupt = 0;    //  Interrupt 0 is on pin 2 !!
int  toggle_on = false;       //  Button click switches state

void setup() {
  Serial.begin(9600);
  attachInterrupt( button_interrupt, handle_click, RISING);  // Register handler
}

void loop() {
  if ( toggle_on ) {
    Serial.println("on");
  } else {
    Serial.println("off");
  }
}

void  handle_click()
{
  static unsigned long last_interrupt_time = 0;       //  Zero only at start

  unsigned long interrupt_time = millis();             // Read the clock
  if ( interrupt_time - last_interrupt_time > 200 ) {  // Ignore when < 200 msec
    toggle_on = !toggle_on;
  }
  last_interrupt_time = interrupt_time;
}
```

toggle_on is a global variable that remembers the "state". It is either true or false (1 or 0).

The loop() function only checks the state of toggle_on. The value of toggle_on is set in the interrupt handler, handle_click.

The value of toggle_on is flipped only when a *true* interrupt even occurs. De-bouncing is described in the next slide.



Interrupt handler for button input

```
int  button_interrupt = 0;    //  Interrupt 0 is on pin 2 !!
int  toggle_on = false;       //  Button click switches state

void setup() {
  Serial.begin(9600);
  attachInterrupt( button_interrupt, handle_click, RISING);  // Register handler
}

void loop() {
  if ( toggle_on ) {
    Serial.println("on");
  } else {
    Serial.println("off");
  }
}

void  handle_click()
{
  static unsigned long last_interrupt_time = 0;       //  Zero only at start

  unsigned long interrupt_time = millis();             // Read the clock
  if ( interrupt_time - last_interrupt_time > 200 ) {  // Ignore when < 200 msec
    toggle_on = !toggle_on;
  }
  last_interrupt_time = interrupt_time;
}
```

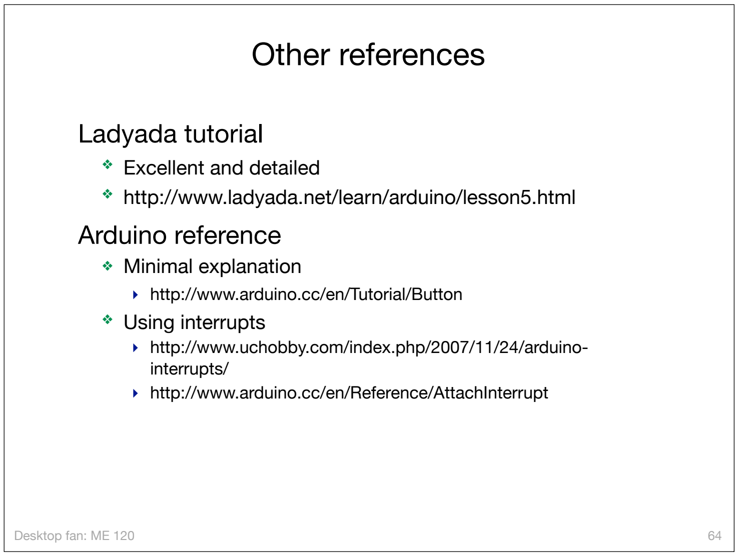
Value of a *static* variable is always retained

Use *long*: the time value in milliseconds can become large

Clock time when current interrupt occurs

Ignore events that occur in less than 200 msec from each other. These are likely to be mechanical bounces.

Save current time as the new "last" time



Other references

Ladyada tutorial

- Excellent and detailed
- http://www.ladyada.net/learn/arduino/lesson5.html

Arduino reference

- Minimal explanation
  - http://www.arduino.cc/en/Tutorial/Button
- Using interrupts
  - http://www.uchobby.com/index.php/2007/11/24/arduino-interrupts/
  - http://www.arduino.cc/en/Reference/AttachInterrupt