Robot Programming and Control

Part I
Using the Arduino to control your robot

TOPICS:
- Robot Systems
- Microprocessor Control and Architecture
- The Integrate Development Interface: sketch programming syntax, output to serial monitor, variables, int, math, delay, pinmode, digitalwrite, if-else, tone, LED, breadboard, HIGH, LOW

Also refer to Arduino Programming Guide and Arduino Fritzing Reference in the website
ROBOTICS PLATFORM

SENSE
- LIDAR
- GPS
- IMU
- Vision Processing
- Filtering

THINK
- Path Planning
- Autonomy
- Kinematics
- Perception
- Localization

ACT
- Holinomic Drive
- Motion Control
- Biomimetic Motion
Where to we start?
How do we build and control robots?

- A good start is to learn microcontrollers and programming
- What is a Microcontroller?
- A microcontroller is a very small computer that has digital electronic devices (peripherals) built into it that helps it control things. These peripherals allow it to sense the world around it and drive the actions of external devices. An example of a use for a microcontroller is to sense a temperature and depending on the value sensed it could either turn on a fan if things were too warm or turn on a heater if things were too cool. You might already be aware that microcontrollers are in common devices like cell phones, microwave ovens, and alarm clocks that have buttons for you to input information and displays to tell you things. But there are even more microcontrollers embedded in things where you never see them. For example there are 30 or more microcontrollers in an automobile. These do everything from sensing the oxygen intake to setting the fuel air mixture to measuring the cabin temperature for controlling the air conditioning levels.
Controlling a motor with and without microcontroller
What is a Microcontroller

- A small computer on a single chip
  - containing a processor, memory, and input/output
- Typically "embedded" inside some device that they control
- A microcontroller is often small and low cost
- Examples: Atmel chip used in the Arduino
The Arduino Microcontroller: Atmel AVR Atmega 328 with special boot loader that interprets English like code.
Boot Strap Loader or Boot Loader is a relatively small program (Arduino) stored in read-only memory (ROM) along with a small amount of needed data, to access the nonvolatile device or devices from which the operating system programs and data can be loaded into RAM. This small program's only job is to load other data and programs which are then executed from RAM.
Arduino on experimental breadboard

What is Arduino? It's an open-source physical computing platform based on a simple microcontroller board, and a development environment for writing software for the board. Arduino can be used to develop interactive objects, taking inputs from a variety of switches or sensors, and controlling a variety of lights, motors, and other physical outputs. Arduino projects can be stand-alone, or they can communicate with software running on your computer.
What is a Development Board?

• A printed circuit board designed to facilitate work with a particular microcontroller.

Typical components include:
• power regulator circuit
• programming interface
• basic input; usually buttons and LEDs
• I/O pins
The Arduino Development Board

[Diagram of the Arduino Development Board with labels for various components such as Pin D13 LED, Tx/Rx LEDs, USB to serial IC, Power LED, Reset, In-circuit serial programming header, Atmel ATmega 328, 5 volt low dropout regulator, DC power jack, 3.3 volt low dropout regulator, Analog I/O pins, Digital I/O pins, and other features.]
LilyPad – Arduino Chip

• LilyPad is popular for clothing-based projects.
## Arduino Board Specs

<table>
<thead>
<tr>
<th>Specification</th>
<th>ATmega328</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Microcontroller</strong></td>
<td>ATmega328</td>
</tr>
<tr>
<td><strong>Operating Voltage</strong></td>
<td>5V</td>
</tr>
<tr>
<td><strong>Input Voltage (recommended)</strong></td>
<td>7-12V</td>
</tr>
<tr>
<td><strong>Input Voltage (limits)</strong></td>
<td>6-20V</td>
</tr>
<tr>
<td><strong>Digital I/O Pins</strong></td>
<td>14 (of which 6 provide PWM output) pulse width modulation</td>
</tr>
<tr>
<td><strong>Analog Input Pins</strong></td>
<td>6</td>
</tr>
<tr>
<td><strong>DC Current per I/O Pin</strong></td>
<td>40 mA</td>
</tr>
<tr>
<td><strong>DC Current for 3.3V Pin</strong></td>
<td>50 mA</td>
</tr>
<tr>
<td><strong>Flash Memory</strong></td>
<td>32 KB (ATmega328) of which 0.5 KB used by bootloader</td>
</tr>
<tr>
<td><strong>SRAM</strong></td>
<td>2 KB (ATmega328)</td>
</tr>
<tr>
<td><strong>EEPROM</strong></td>
<td>1 KB (ATmega328)</td>
</tr>
<tr>
<td><strong>Clock Speed</strong></td>
<td>16 MHz</td>
</tr>
</tbody>
</table>
There are three pools of memory in the microcontroller used on Atmel-based Arduino boards:

- **Flash memory** (program space), is where the Arduino sketch (program) is stored.
- **SRAM** (static random access memory) is where the sketch creates and manipulates variables when it runs.
- **EEPROM** is memory space that programmers can use to store long-term information.

Flash memory and EEPROM memory are non-volatile (the information persists after the power is turned off). **SRAM** is volatile and will be lost when the power is cycled.

The ATmega328 chip found on the Uno has the following amounts of memory:

- **Flash** 32k bytes (of which .5k is used for the bootloader)
- **SRAM** 2k bytes
- **EEPROM** 1k byte
What can you do with 14 inputs/outputs?

- Digital IO (LEDs, switches)
- Analog IO (resistive sensor data)
- Serial Connection (Sensors, GPS, etc)

Program from your computer

Your limit is only your creativity!
Atmel microcontroller architecture
General Purpose Registers

Part of RAM memory. Their purpose is predefined by the manufacturer and cannot be changed therefore. *Since their bits are physically connected to particular circuits within the microcontroller, such as A/D converter, serial communication module etc., any change of their state directly affects the operation of the microcontroller or some of the circuits.* For example, writing zero or one to the registers controlling an input/output port causes the appropriate port pin to be configured as input or output. In other words, each bit of this register controls the function of one single pin.

**PC – Program Counter** is an engine running the program and points to the memory address containing the next instruction to execute. After each instruction execution, the value of the counter is incremented by 1. For this reason, the program executes only one instruction at a time just as it is written.
Instruction decoder is a part of the electronics which recognizes program instructions and runs other circuits on the basis of that.

Arithmetical Logical Unit (ALU) performs all mathematical and logical operations upon data.

Accumulator It is a kind of working desk used for storing all data upon which some operations should be executed (addition, shift etc.). It also stores the results ready for use in further processing. One of the registers called the Status Register, is closely related to the accumulator, showing at any given time the "status" of a number stored in the accumulator.
**Watchdog Timer** is an electronic timer that is used to detect and recover from computer malfunctions. The corrective actions typically include placing the computer system in a safe state and restoring normal system operation. Used in systems where humans cannot access or determine problems – Ex. Space Probe

**Oscillator** generates even pulses that enable harmonic and synchronous operation of all circuits within the microcontroller. It is usually configured as to use quartz-crystal or ceramics resonator for frequency stabilization.
Special Function Registers (SFR) or general purposed registers
Part of RAM memory. Their purpose is predefined by the manufacturer and cannot be changed therefore. Since their bits are physically connected to particular circuits within the microcontroller, such as A/D converter, serial communication module etc., any change of their state directly affects the operation of the microcontroller or some of the circuits. For example, writing zero or one to the SFRs controlling an input/output port causes the appropriate port pin to be configured as input or output. In other words, each bit of this register controls the function of one single pin.
Input/Output registers are the ports
Bits and Bytes (nibble)—how computers communicate

- **Most Significant Bit (MSB)**
  - 1011100110101011

- **Least Significant Bit (LSB)**

- **Nibble** (4 bits)
  - NIBBLE NIBBLE NIBBLE NIBBLE

- **Byte** (8 bits)
  - BYTE BYTE

- **Word** (16 bits)

- **Bit positions:**
  - 8 7 6 5 4 3 2 1

- **Table:**

<table>
<thead>
<tr>
<th></th>
<th>$2^7$</th>
<th>$2^6$</th>
<th>$2^5$</th>
<th>$2^4$</th>
<th>$2^3$</th>
<th>$2^2$</th>
<th>$2^1$</th>
<th>$2^0$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>128</td>
<td>64</td>
<td>32</td>
<td>16</td>
<td>8</td>
<td>4</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

- $2^7 = 128$
- $2^6 = 64$
- $2^5 = 32$
- $2^4 = 16$
- $2^3 = 8$
- $2^2 = 4$
- $2^1 = 2$
- $2^0 = 1$

- $2^7 + 2^6 + 2^5 = 128 + 64 + 32 = 224$

- $2^7 + 2^6 + 2^5 + 2^4 + 2^3 + 2^2 + 2^1 + 2^0 = 256$

- $10111001101011_{(2)} = 256_{(10)}$

- $10111001101011_{(2)} = 175_{(10)}$ (Decimal)
The Arduino chip has 6 analog inputs, labeled A0 through A5, each of which provide 10 bits of resolution (i.e. 1024 different values). By default they measure from 0 to 5 volts.
**Timers/Counters** most programs use these miniature electronic "stopwatches" in their operation. These are commonly 8- or 16-bit SFRs the contents of which is automatically incremented by each coming pulse. Once the register is completely loaded, an interrupt is generated!

**Interrupts** the Arduino UNO board has 2 interrupts IN01 and IN02 on pins 1 and 2 which means it can recognize two different events that can interrupt regular program execution. In industrial robots you want to be able to monitor for problems that could destroy equipment or hurt humans for example. Without interrupts you would only be able to run programs one line at a time and stop without putting a pause to it.
OCD – On Circuit Debugging (JTAG)

Adds two registers without the C Flag and places the result in the destination register Rd.

Operation:
(i) \( Rd \leftarrow Rd + Rr \)

Syntax: Operands: Program Counter:
(i) ADD Rd,Rr \( 0 \leq d \leq 31, 0 \leq r \leq 31 \) PC \( \leftarrow PC + 1 \)

R (Result) equals Rd after the operation.

Example:
add r1,r2 ; Add r2 to r1 \((r1=r1+r2)\)
add r28,r28 ; Add r28 to itself \((r28=r28+r28)\)

Note: you will be introduced to the breadboard later on
AVR Studio allows you to debug or program Atmel chips.
Why is the Arduino?

The word “Arduino” can mean 3 things

A physical piece of hardware

A programming environment

A community & philosophy

Video of story of Arduino
Proto shield w/mini breadboard
Ethernet shield w/micro-SD reader
Extreme shield stacking
Getting Started

1. Download & install the Arduino environment (IDE)
2. Connect the board to your computer via the UBS cable
3. If needed, install the drivers (not needed in lab)
4. Launch the Arduino IDE
5. Select your board
6. Select your serial port
7. Open the blink example
8. Upload the program
Connecting the USB cable to the Arduino and computer. Wait for driver to install.
Arduino IDE (integrated development environment)

See: http://arduino.cc/en/Guide/Environment for more information
Specify Serial Port and Board type
Configuring the Arduino Software

• Defaults to COM1, will probably need to change the COM port setting (PC uses COM7 sometimes).
• Appears in Device Manager (Win7) under Ports as a Comm port.
Status Messages

Uploading worked

Wrong serial port selected

Wrong board selected

nerdy cryptic error messages

todbot.com/blog/bionicarduino
A Little Bit About Programming

- Code is case sensitive
- Statements are commands and must end with a semi-colon
- Comments follow a // or begin with /* and end with */
Arduino Programming

• Based on C++ without 80% of the instructions.
• Programs are called 'sketches'.
• Sketches need two functions and a header
  - **Header**: has declarations, includes, etc.
  - `void setup( )`
  - `void loop( )`
• `setup( ) runs first and once.`
• `loop( ) runs over and over, until power is lost or a new sketch is loaded.`
• Easy to reuse C-code from other projects
• Libraries can be written in C++, lots of libraries available
Arduino C

- Arduino sketches are centered around the pins on an Arduino board.
- Arduino sketches always loop.
  
  - `void loop( ) {}`
  
  - The pins can be thought of as global variables meaning they are accessed throughout the whole program.
# Arduino Variables

<table>
<thead>
<tr>
<th>Numeric types</th>
<th>Bytes</th>
<th>Range</th>
<th>Use</th>
</tr>
</thead>
<tbody>
<tr>
<td>int</td>
<td>2</td>
<td>-32768 to 32767</td>
<td>Represents positive and negative integer values.</td>
</tr>
<tr>
<td>unsigned int</td>
<td>2</td>
<td>0 to 65535</td>
<td>Represents only positive values; otherwise, similar to int.</td>
</tr>
<tr>
<td>long</td>
<td>4</td>
<td>-2147483648 to 2147483647</td>
<td>Represents a very large range of positive and negative values.</td>
</tr>
<tr>
<td>unsigned long</td>
<td>4</td>
<td>4294967295</td>
<td>Represents a very large range of positive values.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Numeric types</th>
<th>Bytes</th>
<th>Range</th>
<th>Use</th>
</tr>
</thead>
<tbody>
<tr>
<td>float</td>
<td>4</td>
<td>3.4028235E+38 to -3.4028235E+38</td>
<td>Represents numbers with fractions; use to approximate real-world measurements.</td>
</tr>
<tr>
<td>double</td>
<td>4</td>
<td>Same as float</td>
<td>In Arduino, double is just another name for float.</td>
</tr>
<tr>
<td>boolean</td>
<td>1</td>
<td>false (0) or true (1)</td>
<td>Represents true and false values.</td>
</tr>
<tr>
<td>char</td>
<td>1</td>
<td>-128 to 127</td>
<td>Represents a single character. Can also represent a signed value between -128 and 127.</td>
</tr>
<tr>
<td>byte</td>
<td>1</td>
<td>0 to 255</td>
<td>Similar to char, but for unsigned values.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Other types</th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>string</td>
<td></td>
<td>Represents arrays of chars (characters) typically used to contain text.</td>
<td></td>
</tr>
<tr>
<td>void</td>
<td></td>
<td>Used only in function declarations where no value is returned.</td>
<td></td>
</tr>
</tbody>
</table>
Arduino C is Derived from C++

- These programs blink an LED on pin 13

**avr-libc**
```
#include <avr/io.h>
#include <util/delay.h>

int main(void) {
    while (1) {
        PORTB = 0x20;
        _delay_ms(1000);
        PORTB = 0x00;
        _delay_ms(1000);
    }
    return 1;
}
```

**Arduino C**
```
void setup( ) {
    pinMode(13, OUTPUT);
}

void loop( ) {
    digitalWrite(13, HIGH);
    delay(1000);
    digitalWrite(13, LOW);
    delay(1000);
}
```
Blinking LED – Our first program!

Notes: LED Light Emitting Diode
- LEDs have polarity
- LEDs should have a current limiting resistor or they can burn
- Pin 13 of the Arduino UNO board has a resistor so is ok to wire an LED directly to it
Testing your board and adding an External LED to pin 13 (OUTPUT EXAMPLE)

- File > Examples > Digital > Blink

- LED’s have polarity!
  - Negative indicated by flat side of the housing and a short leg

www.instructables.com
What’s a Breadboard?

• One of the most useful tools in an engineer or Maker’s toolkit. The three most important thing to remember:
  • A breadboard is easier than soldering
  • A lot of those little holes are connected, which ones?
  • Sometimes breadboards break
how to use the **breadboard** so you can play around with some electronics.

**Example of parallel circuit.** Two LEDs and a resistor.

**Example of series circuit.** Three LEDs.

**Power supply connections** (vertical)
- Terminal strips (horizontal)
- This line divides the breadboard in half, electricity will not conduct through it
- Indicates where breadboard conducts electricity (displayed only on right side of board)

**LED**
- The different lengths of the two wires coming out of the LED indicate which wire is positive and which is negative. The LED will not work if you hook it up backwards.

---

*Show video: breadboards*
Blinking LED Wiring: Example of OUTPUT

Connecting a LED to pin 13 of the Arduino

Adding a resistor 220 to 330 ohm

Using a breadboard to connect components

Preparing pin 13 to be an output port:
```
int led = 13; // declare the LED on pin 13
pinMode(led, OUTPUT); // pin 13 is an output
```

Notes:
Output pins can provide 40 mA of current, 3.3v port 50mA

Writing HIGH to an input pin installs a 20KΩ pull-up resistor
Blink: Turns on an LED on for one second, then off for one second, repeatedly.

// Pin 13 has an LED already connected on most Arduino boards.
// give it a name: ledPin
int led = 13;

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

void loop() {
  digitalWrite(led, HIGH);  // turn the LED on (HIGH is the voltage level)
  delay(1000);              // wait for a second
  digitalWrite(led, LOW);   // turn the LED off by making the voltage LOW
  delay(1000);              // wait for a second
Specify the status and command of a pin with these functions

• `pinMode(pin, mode);`
  Designates the specified pin for input or output
  Example: `pinMode(led, OUTPUT);`

• `digitalWrite(pin, value);`
  Sends a voltage level to the designated pin
  Example: `digitalWrite(led, HIGH);`
Arduino Timing

- **delay**(ms);  ex. Delay(1000); is 1 sec
  - Pauses for a few milliseconds before continuing to the next line of code

- **delayMicroseconds**(us);
  - Pauses for a few microseconds. Sometimes necessary when you want to split time between two functions to make it appear as if you are doing two things at once.

Variable Scope
Where you declare your variables matters

```c
/*! Blink
Turns on an LED on for one second, then off for one second, repeatedly.

This example code is in the public domain.
*/

const int variable1 = 1;
int variable2 = 2;

void setup() {
    int variable3 = 3;
    // initialize the digital pin as an output
    // Pin 13 has an LED connected on most Arduino Boards
    pinMode(13, OUTPUT);
}

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

Constant / Read only
Variable available anywhere
Variable available only in this function, between curly brackets
The setup function comes before the loop function and is necessary for all Arduino sketches.

```c
void setup() {
    // initialize the digital pin as an output.
    // Pin 13 has an LED connected on most Arduino boards:
    pinMode(13, OUTPUT);
}
```
Setup

```java
void setup () {} 
```

The setup header will never change, everything else that occurs in setup happens inside the curly brackets

```java
void setup() { 
  // Initialize the digital pin as an output.
  // Pin 13 has an LED connected on most Arduino boards:
  pinMode(13, OUTPUT);
}
```
Setup

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

Outputs are declared in setup, this is done by using the `pinMode` function. This particular example declares digital pin # 13 as an output, remember to use CAPS.
void setup () {
    Serial.begin(9600);
}

Serial communication also begins in setup

This particular example declares Serial communication at a baud rate of 9600. More on Serial later...
Setup, Internal Pullup Resistors

```c
void setup () {
  digitalWrite (12, HIGH);
}
```

You can also create internal pullup resistors in setup, to do so `digitalWrite` the pin `HIGH`
void setup() // mandatory function in Arduino
{
  Serial.begin(9600); // sets the communication speed between computer and Arduino
  Serial.print("Hello!"); // print the word Hello! On the screen
}

void loop() // mandatory function in Arduino
{
  //Add code that you want to repeat here
}

What does it do? When you upload this program to the Arduino board and open the Serial Monitor window you will see the word Hello! Displayed. Useful when you want to check the status of a sensor and display its value on the screen.
Arduino does Simple math

// Program1 - SimpleMath
void setup()
{
  Serial.begin(9600);
  int a = 89;
  int b = 42;
  int c = a + b;
  Serial.print("a + b = ");
  Serial.println(c);
}
void loop()
{
  // Empty, no repeating code.
}

// Program 2 - Find Circumference of a Circle
void setup()
{
  Serial.begin(9600);
  float r = 0.75;
  float c = 2.0 * PI * r; // PI is a built-in constant
  Serial.print("circumference = ");
  Serial.println(c);
}
void loop()
{
  // Empty, no repeating code.
}
int pingPin = 7;

void setup()
{
  Serial.begin(9600);
  int a = 89;
  int b = 42;
  int c = a + b;
  Serial.print("a + b = ");
  Serial.println(c);
}

void loop()
{
  // Empty, no repeating code.
}
Functions

Functions allows a programmer to create modular pieces of code that perform a defined task and then return to the area of code from which the function was "called". The typical case for creating a function is when one needs to perform the same action multiple times in a program.

- Makes sketches smaller
- Helps organize and reuse code
- You repeat a lot less!

Mandatory Functions

- void setup()
- void loop()
Example creating the function myMultiplyFunction()

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

void loop() {
  int i = 2;
  int j = 3;
  int k;

  k = myMultiplyFunction(i, j); // k now contains 6
  Serial.println(k);
  delay(500);
}

int myMultiplyFunction(int x, int y){
  int result;
  result = x * y;
  return result;
}
```
Operators:

= is used to assign a value
== is used to compare values

And & Or

&& is “and”
|| is “or”
If Statements

```
void loop()
{
  // read the state of the pushbutton value:
  buttonState = digitalRead(buttonPin);

  // check if the pushbutton is pressed.
  // if it is, the buttonState is HIGH:
  if (buttonState == HIGH)
  {
    // turn LED on:
    digitalWrite(ledPin, HIGH);
  }
  else
  {
    // turn LED off:
    digitalWrite(ledPin, LOW);
  }
}
```

If Statement
If

if ( this is true ) { do this; }

```c
void loop(){
  // read the state of the pushbutton value:
  buttonState = digitalRead(buttonPin);

  // check if the pushbutton is pressed.
  // if it is, the buttonState is HIGH:
  if (buttonState == HIGH) {
    // turn LED on:
    digitalWrite(ledPin, HIGH);
  }
  else {
    // turn LED off:
    digitalWrite(ledPin, LOW);
  }
}
```
Conditional

\[
\text{if ( this is true ) \{ do this; \}}
\]

```c
void loop(){
  // read the state of the pushbutton value:
  buttonState = digitalRead(buttonPin);

  // check if the pushbutton is present.
  // if it is, the button state is HIGH:
  if (buttonState == HIGH) {
    // turn LED on:
    digitalWrite(ledPin, HIGH);
  }
  else {
    // turn LED off:
    digitalWrite(ledPin, LOW);
  }
}
```

Conditional inside parenthesis, uses ==, <=, >= or !
you can also nest using && or ||
Action

if ( this is true ) { do this; }

Action that occurs if conditional is true, inside of curly brackets, can be anything, even more if statements
Another Function Sample: Moving a two wheeled robot forward

```c
if((wLeft == 0) && (wRight == 0)) // If both front sensors contact
{
    forward(1000); // Back up 1 second
}
else // Otherwise, no sensor contact
{
    forward(20); // Forward 1/50 of a second
}

// Create Forward function, int time is how long to move

void forward(int time)
{
    servoLeft.writeMicroseconds(1700); // Left wheel counterclockwise
    servoRight.writeMicroseconds(1300); // Right wheel clockwise
    delay(time); // Maneuver for time ms
}
```

Note: This piece of code shows how a function is created and called. You can also pass parameters to a function like the time parameter. It also displays the if..else commands
A nice if/else sample to change blinking rate based on user input from keyboard

```cpp
void setup() {
  Serial.begin(9600);
pinMode(13, OUTPUT);  // LED on pin 13 of UNO
}

char rx_byte = 0; // declare receive variable

void loop() { // declare receive variable
  if (Serial.available() > 0) { // is a character available?
    rx_byte = Serial.read();
  }
  if (rx_byte == 'a') {
    digitalWrite(13, HIGH);
delay(500);
digitalWrite(13, LOW);
delay(500);
  }
  else if (rx_byte == 'b') {
    digitalWrite(13, HIGH);
delay(200);
digitalWrite(13, LOW);
delay(200);
  }
}
```

Run this program and open the Serial Monitor. Press letter a to make the LED blink slower or b to make it blink faster! Play around with the delay values or even the keyboard input values.
Pushbuttons – normally open switch

Piezo Elements - speaker

Notes: Piezo elements (speaker) makes a clicking sound each time is pulsed with current. Depending on the pulse/frequency so will be the tone or sound.
- Pushbuttons do not have polarity but must be connected correctly to work (close the circuit)
- Pull HIGH (positive 5volt) or LOW (ground) so that the microprocessor knows the status of the pin.
Using a switch to turn on an LED: Input
// Using a switch to turn on an LED
// constants won't change. They're used here to set pin numbers:
const int buttonPin = 2; // the number of the pushbutton pin
const int ledPin = 13; // the number of the LED pin

// variables will change:
int buttonState = 0; // variable for reading the pushbutton status

void setup() {
    // initialize the LED pin as an output:
    pinMode(ledPin, OUTPUT);
    // initialize the pushbutton pin as an input:
    pinMode(buttonPin, INPUT);
}

void loop(){
    // read the state of the pushbutton value:
    buttonState = digitalRead(buttonPin);
    // check if the pushbutton is pressed.
    // if it is, the buttonState is HIGH:
    if (buttonState == HIGH) {
        // turn LED on:
        digitalWrite(ledPin, HIGH);
    } else {
        // turn LED off:
        digitalWrite(ledPin, LOW);
    }
}
Making Sound with The Arduino

The first is `tone()` which takes 2 required parameters (and an optional third).

```
tone(pin, frequency, duration)
```

OR

```
tone(pin, frequency)
```

`noTone(pin)` stops the tone
Simple Tone Example

```cpp
void setup()
{
  tone(8, 440, 1000);
}

void loop()
{
}
```

Tones are created by quickly pulsing a speaker on and off.
tone(8, 3000, 1000); // Play tone for 1 second
delay(100);
tone(8, 300, 1000);
delay(100);
tone(8, 2000, 1000);
delay(100);
tone(8, 600, 1000);
delay(100);
tone(8, 3000, 1000);
delay(1000); // Delay to finish tone
// Star Wars and LEDs
const int c = 261;
const int d = 294;
const int e = 329;
const int f = 349;
const int g = 391;
const int gS = 415;
const int a = 440;
const int aS = 455;
const int b = 466;
const int cH = 523;
const int cSH = 554;
const int dH = 587;
const int dSH = 622;
const int eH = 659;
const int fH = 698;
const int fSH = 740;
const int gH = 784;
const int gSH = 830;
const int aH = 880;
const int buzzerPin = 4;
const int ledPin1 = 10;
const int ledPin2 = 13;
int counter = 0;

void setup()
{
    // Setup pin modes
    pinMode(buzzerPin, OUTPUT);
pinMode(ledPin1, OUTPUT);
pinMode(ledPin2, OUTPUT);
}

void loop()
{
    // Play first section
    firstSection();
    // Play second section
    secondSection();
    // Variant 1
    beep(f, 250); beep(gS, 500); beep(f, 350); beep(a, 125); beep(cH, 650);
    delay(500);
}

void beep(int note, int duration)
{
    // Variant 2
    beep(f, 250); beep(gS, 500); beep(f, 375); beep(a, 500); beep(a, 375); beep(a, 650);
    delay(650);
}

void firstSection()
{
}

void secondSection()
{
    // Play tone on buzzer Pin
    tone(buzzerPin, note, duration);

    // Play different LED depending on value of 'counter'
    if(counter % 2 == 0)
    {
        digitalWrite(ledPin1, HIGH); beep(cH, 500);
delay(duration);
        digitalWrite(ledPin1, LOW);
    }
    else
    {
        digitalWrite(ledPin1, LOW);
        digitalWrite(ledPin2, HIGH); beep(cH, 125);
delay(duration);
    }
}
```c
{  digitalWrite(ledPin2, HIGH);
  delay(duration);
  digitalWrite(ledPin2, LOW);
}

//Stop tone on buzzerPin
noTone(buzzerPin);

delay(50);
//Increment counter
counter++;
}

void firstSection()
{
  beep(a, 500);
  beep(a, 500);
  beep(a, 500);
  beep(f, 350);
  beep(cH, 150);
  beep(a, 650);
  
  delay(500);
  
  beep(eH, 500);
  beep(eH, 500);
  beep(eH, 500);
  beep(fH, 350);
  beep(cH, 150);
  beep(gS, 500);
  beep(f, 350);
  beep(cH, 150);
  beep(a, 650);

  delay(500);
}

void secondSection()
{
  beep(aH, 500);
  beep(a, 300);
  beep(a, 150);
  beep(aH, 500);
  beep(gS, 500);
  beep(gSH, 325);

  delay(350);
}
```
Arduino Reference Summary

Arduino Digital and Analog I/O Pins

- Digital pins:
  - Pins 0 – 7: PORT D [0:7]
  - Pins 8 – 13: PORT B [0:5]
  - Pins 14 – 19: PORT C [0:5] (Arduino analog pins 0 – 5)
  - digital pins 0 and 1 are RX and TX for serial communication
  - digital pin 13 connected to the base board LED

- Digital Pin I/O Functions
  - pinMode(pin, mode)
    - Sets pin to INPUT or OUTPUT mode
    - Writes 1 bit in the DDRx register
  - digitalWrite(pin, value)
    - Sets pin value to LOW or HIGH (0 or 1)
    - Writes 1 bit in the PORTx register
  - int value = digitalRead(pin)
    - Reads back pin value (0 or 1)
    - Read 1 bit in the PINx register

Also refer to Arduino Programming Guide and Arduino Fritzing Reference in the website of folder.