ArduinoIntroTutorial

Michele Pratusevich

November 24, 2013

1 MEET YL2 Individual Project: The Arduino Tutorial

1.1 The plan for you, reader of this tutorial

Your instructions are to work through this tutorial. If you don't understand something, ask each other for help. Only come ask me if something is unclear and no one can figure it out. Of it you think you are doing something that is dangerous to you and / or others. The tutorial takes you through roughly these steps:

1. Read about hardware, Arduinos
2. Play with the Arduino-program-language with some simple programs
3. Play with serial connections to the computer
4. Learn how breadboards work
5. Learn how to attach a button to an Arduino
6. Plan your individual project
7. Work work work!

1.2 What is hardware? Why do we care? (What you CAN DO with hardware) ## (top)

- Interact with the physical world - react, make something move, etc.
  - User presses buttons, turns dials
  - Sensors get information from environment
  - Control motors
- Run code on tiny computers that can stand alone (without a full computer)

1.3 What you WILL DO with hardware

- We won't control anything - for now we will focus on getting input, either from the user or from the physical world.
- To start with, you will work on getting input from the user through a physical external button press. If there is time, you will use some other sensor to get information from the user or the environment in a different way.

1.4 A bit about hardware in general

Hardware is a broad term used to describe physical components outside of a computer. This includes electronics, circuits, and chips (the things that make the inside of a computer). Some facts, terms, information:

- Hardware is serious business. It is sensitive physical pieces that carry electricity to perform a certain task. Connecting wires the wrong way, putting physical stress on the parts, or making the parts dirty, wet, sticky, etc. is a danger to your physical safety. Please respect the hardware, and make sure to take good care of it, both for it's sake and for your sake.
- **microcontroller** - a small computer, usually without a display. Has limited code that it understands, and is often coded for a specific purpose in the physical world away from a computer. It, like any other computer, needs to get power from somewhere.
- **microprocessor** - another word for a small computer, but one that is more powerful than a microcontroller
- Since you can't directly type code onto a microcontroller, you type your code in a computer, then compile and upload it ("burn" it) onto the microcontroller. Usually you do this through a USB cable that talks with the microcontroller using a language called **serial communication**. If you tell the microcontroller to talk to the computer, it will do so through the **serial console**.
- Another name for the USB port is a **COM port**.

Some electronics concepts you will need to learn about hardware:

- **Components** are pieces of electrical hardware, like resistors, buttons, and wires
- **Digital** vs. **analog** - When something is digital, it has two values it can take, ON (1) or OFF (0). There is no such thing as in-between. When something is analog, it can take any value between ON (1) and OFF (0). Digital inputs and outputs are used for detecting things like button presses and for counting numbers, while analog inputs are used for detecting things like dial or slider position
- **Sensors** are electrical components that measure something. They can be temperature sensors, distance sensors, light sensors, sound sensors, etc.
- In the world of both digital and analog electronics in microprocessors, the maximum (ON) voltage is usually 5 volts, and this is referred to as **power**. The opposite, or OFF voltage, is 0 volts, and is referred to as **ground**, often abbreviated **GND**.
- A **breadboard** is a grid used putting together electrical components. Like this:

1.5 Arduino: the easiest way to get started

The best way to get started in getting comfortable with these tools is by using a microcontroller platform that is easy to use and has a lot of tutorials. This is the Arduino platform that we are going to use.

You can see a few labeled parts on the board. Try to find them on your own boards.

The cable that connects the Arduino to the computer is a USB cable that connects to a USB port (it is on the back of the laptop) and to the USB connector on the Arduino. This USB cable is also the cable that gives the Arduino electricity. Go ahead and do that now. You should see the red "power" light be on in the Arduino.

What is on the Arduino and what is on the computer

The workflow is like this: you write code on the computer in an editor, compile it on the computer, and send it to the Arduino (using the serial protocol) through the cable. As soon as this process is complete, the code executes immediately on the Arduino. The Arduino can then talk to the computer (if you tell it to) by printing things to the computer through the **serial console**. The way to "restart" a program running on the Arduino is to press the **reset button**.

1.6 How we work with Arduino and the Arduino IDE ## (top)

- Coding our Arduino programs on Ubuntu using the Arduino IDE (**IDE** stands for integrated development environment, which basically means a program that gives us a place to write our code)
- Arduino uses a special language; it looks like C++, but it reads like Python (and like English). The best way to learn the language is to look at examples and play with the code yourself. So there will not be an explanation of how the code works - you will learn by going through the example code. After you finish coding your program, the way to test it is to upload it to the Arduino and make sure it does what you want it to do.

Opening the Arduino IDE

It is already installed on the Ubuntu machines. To open it, open the program explorer and start typing "Arduino":

It will open a program like this:

The white space is where you will type your code, and the black is where the error messages will be shown.

Opening Arduino sample code from the IDE

One of the awesome things about working with Arduino is the wealth of resources available for Arduino. These include "official Arduino sample code." To open the tutorials, go to "File" -> "Examples", and then going to the category you want. Each tutorial will tell you which example code to open.

1.7 Tutorial 1: BLINK ## (top)

Open the "Blink" example code from the "1. Basics" example code menu. DO NOT attach a resistor and LED to the Arduino - the light on the Arduino board itself is connected to pin 13, so there is no need to risk blowing up our LEDs. Look through the code, with this tutorial as a guide.

Upload the code to the Arduino using the upload button (the arrow in the white circle at the top of the IDE window):

When you are done uploading, it should look like:

If you have any orange text in the black box, something is wrong.

If you have problems with your code not uploading (you'll see errors in the black error box at the bottom of the IDE), check out some common problems you might have.

Once you are done reading and understanding this code, do the following exercises:

1. Change the light to be on for 2 seconds, off for 2 seconds
2. Change the light to be on for 3 seconds, off for 1 second

You don't need to submit anything for these exercises, but you might want to save a copy of your modified programs in your Github folder. Answer for yourself the following question: What section of code happens once? What section of code continues to happen during execution?

1.8 Tutorial 2: Serial communication ## (top)

Now, let's play with how to send information from the Arduino to the computer's screen over serial. Make a new file and paste this code into it:

```
/*
 * Hello World!
 *
 * This is the Hello World! for Arduino.
 * It shows how to send data to the computer
 */


void setup()                    // run once, when the sketch starts
{
  Serial.begin(9600);           // set up Serial library at 9600 bps

  Serial.println("Hello world!");  // prints hello with ending line break
}

void loop()                       // run over and over again
{
                                  // do nothing!
}
```

Notice that there is nothing in the `loop()` method, so everything that happens will happen only once. Upload the code to the Arduino. Nothing happens. This is because you need to open the Serial Monitor (the thing that watches what comes to the computer over the serial port) by going to "Tools" -> "Serial Monitor". You should get a window that pops up like this:

To restart the program that has been burned into the Arduino, press and hold the RESET button on the Arduino (if you don't remember where it is, go back and look at the parts diagram). What do you expect to happen in the Serial Monitor? What actually happens?

Once you've gotten the code to work (and understand the result), do the following exercises:

1. Make the Arduino print "Hello, World!" over serial once every second.

2. Make the Arduino print "Hello, World!" over serial once every second, but only a total of five times.

Save your code from exercise 2 in your Github repository.

1.9 How a breadboard works ## (top)

Remember breadboards?

An interesting thing about breadboards is that the squares in each row are connected to each other. Horizontal rows are connected:

And the two vertical rows on either side of the board are connected:

For example, if I connect a wire from 5V on the Arduino to cell B2, then A2, C2, D2, and E2 are all automatically connected to 5V also.

One common way to take advantage of this is to connect one cell in the right column to GROUND (0 volts), then being able to connect any components you need to ground by connecting a wire between the component and the ground column.

For example, if you connect a wire from GND on the Arduino to the side column on the breadboard, like this:

Then every other square in that column will be connected to GND automatically.

1.10 Tutorial 3: Buttons ## (top)

Next, open up the DigitalReadSerial, or the Button tutorial. Follow along with this tutorial to understand the code. Build the circuit, taking care to connect the appropriate components in the right way. Our buttons look like this:

And the wires are super-easy to use and look like this:

Test it to make sure the code works.

When you are done, take a picture of your circuit and save it in your Github repository.

1.11 Your individual project ## (top)

Now that you have seen a few tutorials and hopefully have gotten a button to work on the Arduino, you need to choose a final project to work on.

You must choose one of these projects to do as your individual project, but no two people can do the same project. Projects are written in approximate order of difficulty, easier projects at the top. If you come up with a variation of one of these projects, that is fine, just talk to me to get it approved. Two people can work on a variation of the same project.

1. A simple switch - print the number 1 to the screen over and over again, until the user presses a button. Then print the number 2 to the screen. Every time the user presses the button, increase the number displayed to the screen by one.
2. Keep-alive Light Button. Pretend that the Arduino button is a light that is keepin ga baby chick alive. Keep the Arduino light turned on when the button is pressed. Only turn the light off when the button is no longer pressed.
3. Ask the user to press the button some number of times. After you don't hear a button press for 2 seconds, use that number to set how fast the LED blinks.
4. Make some kind of trivia game: ask the user a question randomly (from a fixed list, not from the internet), and the user must press the button the correct number of times to answer the question.
5. Every two seconds, ask the user for a button press 4 times. Keep track of the button presses like this: 1011 (where 1 is a press and 0 is not a press). At the end, convert this number from binary to decimal and print it to the screen.
6. Ask the user to press and hold the button. However long the user presses the button, set that to how fast the LED blinks.
7. A password detection application - when the user presses the button the correct number of times for the correct length of time, the password is correctly detected and the computer prints "success!" on the screen
8. A Morse code encoding application - the user can record messages using Morse code, which can be printed back to the screen when the user is done

Spend some time thinking (and deciding!) on which project you will work on. To see more information on the project description and requirements, look here.

If you want to see two more tutorials, here are two that come logically next. Otherwise, start working on your project! But first:

1. Draw a diagram of any components you need (remember, PLAN, then DO)
2. Draw a diagram of any circuit you need for your project

1.12 Tutorial 4: Analog Input (potentiometer)

A **potentiometer** is an electrical component that is the equivalent of a dial or a knob, used to get analog input. Our potentiometers look like this:

Follow this tutorial to learn how to use them.

1.13 Tutorial 5: Blink without delay

This tutorial, without actually hooking up the LED.

1.14 Setting up Arduino for yourself at home

Really easy on your own computer. Instructions here. If you want to take one home to work, talk to me first.

1.15 Resources

There are a zillion Arduino resources around - there is a lot to look through and it can be daunting. Here is a list of select resources that are worth your time, if you need to find anything:

- The official Arduino tutorials
- Arduino in a Nutshell
- http://www.ladyada.net/learn/arduino/index.html