Programming your Arduino

In the previous projects, you’ve only been using your Arduino as a power source. While it certainly can be useful for that, it’s still not normally enough to make very good robots. As mentioned in the previous project, good robots can interpret data. In this project you’ll learn how to make your Arduino interpret data using software that you program onto it.

# Setup

Arduinos have their own programming language and Interactive Development Environment (IDE) made for programming them. Since that’s what you’ll be using for these projects, you’ll need to download them before you start programming your Arduino. Note that when you download the Arduino IDE, it should come with whatever you need to start programming, so you shouldn’t need to worry about downloading the Arduino language. Here are some steps for downloading the IDE:

1. Do a quick search for “Arduino ide.” The result you want will be from <https://www.arduino.cc>. If it asks you about cookies, the safest option is probably “Only required.”
2. Look for the download version you want. I use version 1.8.19 of the Legacy IDE. While you might be able to get away with using a different version, I don’t recommend it because I’ve heard that some newer versions have problems with certain things that 1.8.19 doesn’t have problems with.
3. Make sure to pick the download option that best suits your device.
4. The download will probably come in the form of a setup executable file. If it does, just run it and follow any instructions it gives you. I didn’t select or unselect any options it gave me, except possibly the option to create a desktop shortcut for the IDE. This option I made sure was selected.
5. Make sure the files are where you want them to be. If you chose the ZIP file option, you might need to extract its contents.
6. Once everything’s set up, you should now be able to use the Arduino IDE.

Congratulations! You now have the IDE installed! Now you can get to work programming your Arduino.

# The Arduino program

Software in the form of a program is what lets an Arduino “think.”

Open up the “LightSwitcher.ino” file for this project. Be sure to open it with Arduino IDE. The first few lines should look something like the following:

#define LED1 2

#define LED2 3

#define LED3 4

#define LED4 5

Naturally, this code may look confusing at first, but let’s go through it together. Lines 2-6 are used to define variables representing pin numbers- the same numbers you see on your Arduino board.

Lines 9 through 11 are creating variables- “n” keeps track of which light will be lit. “Benchmark” is used to control what value we need to receive from the analog pins to consider them “on” or “active,” and “wasPressed” is used to help keep track of inputs.

Lines 16-20 tell the board whether certain pins are to be used as inputs or outputs, while lines 23-26 assign some of the pins starting values.

For now I’m going to skip past lines 32-37. These will be explained a little bit later. Lines 40-55 decide which pins should be on and off according to the value of n. I’m also going to skip past line 55.

The delay(200) command on the second-to-last line tells the Arduino to wait 200 milliseconds (0.2 seconds, or a fifth of a second) before restarting the loop.

Going back to lines 32-37 and line 55, we need a way for the Arduino to recognize what counts as pressing a button. It might sound silly or redundant to say this, but there are three parts to someone pushing a button: The initial press, the period in which the button is held down, and the moment when the button is released. Though that may sound obvious, it’s an important distinction. Sometimes when we press a button we want something to react as soon as the button is pressed; sometimes we want it to react *while* a button is being pressed, and sometimes we want it to react *after* the button has been pressed. In this case, I wrote the program with the intention that the lights would change *after* the button has been pressed- as soon as the button is released. To use that, I made use of what I like to think of as a “staggered” variable- a variable that keeps track of a previous state, which I update at the *end* of the loop.

This sounds complicated, but it’s for a good reason. Since the Arduino only knows if the button is on or off, I need to be creative. A button has several phases it can be in:

However, if we look at the current and previous phase of the button for each of these states, we notice that each phase yields different states:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Phase | Button left alone | Button pushed | Button held | Button released |
| Current state | Off | On | On | Off |
| Previous state | Off | Off | On | On |

Looking at this we can see that if the button is being released, it is off *now*, but it *was* on just a moment ago. We check by comparing the current state of the button, as found by “analogRead(P1In), to the previous state, which is held by “wasPressed.”

But why do we update wasPressed near the end of the loop- after we use its value? This goes back to what I mentioned about it being “staggered.” By updating it at the end of the loop instead of the start of the loop, we can assign it a value at the end of the loop and then use that value in the next loop. When it loops, the variable holds what becomes the previous value.

If it still seems complicated, don’t worry about it too much. What’s important here is that it works.

# Building the circuit

Once again, you’ll need to build the circuit for this project. Your circuit should be similar to the following circuit diagram (see next page):

A circuit board with wires connected to it

Description automatically generated

# Programming the Arduino

Now that you’ve built your circuit, you’ll need to upload the code to your Arduino. If you’ve closed it or you haven’t opened it, open LightSwitcher.ino in Arduino IDE now. Make sure your Arduino is plugged into your computer with the correct USB cable. You may also need to select the correct board and port in the menus. Hit the upload button, and watch the Arduino do its magic!

# What does the code do?

The code, complicated as it may be or seem, does something pretty simple: it uses one button to control four lights. Each time you push the button a different LED should light up, with the previous one going dark.

# The big takeaway

This project is meant to help show you how an Arduino can take in data, process it, and output it.