Arduino Coding Basics

In this article, I’m going to cover some of the basics of the programming language used by the Arduino IDE. The Arduino IDE uses the “Processing” programming language. It will look very similar to C/C++ or Java if you've used that for any previous projects.

This article is just an overview of some parts of the language to get you started and we will cover other commands in specific projects. You can also see more information about the language on the Arduino Language Reference Page.

Keep in mind that the more familiar you become with the Arduino code structure, the more comfortable you'll be experimenting and developing your own projects!

With everything, one of the best ways to understand this content is to practice. You should follow along inside the Arduino IDE while working through this instruction.

Comments

Comments are ignored by the microcontroller and are notes for yourself or other programmers who might look at your code. Comments are often used to review or update code that you've written in the past. Think of comments as being helpful tips or guides that you can leave for yourself.

Another great use of comments is for instruction and educational purposes. Much of the code that you'll use in this camp will be heavily commented so that you'll be able to read about the functionality of the code as you're working with it.

Commenting your code is one of the best habits that you can develop, and the sooner you start the more naturally you'll be able to integrate it into your workflow.

There are two types of comments: single line and double line.

A single line of comments will start with a // and the program will ignore everything after the // until the end of the line, for instance:

```
void setup() {
  // put your setup code here, to run once:

}

void loop() {
  // put your main code here, to run repeatedly:

}
```

Single line comments can also be placed at the end of a line of code, for example:

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

Single line comments are also a great way to turn off specific lines of your code if you are trying to troubleshoot an issue OR if you want to remove a line of code from being read but you don't want to delete it in case you need it for reference. We call this "*commenting out a line of* *code*" and it means to remove it from the program by commenting it but not actually deleting the line. Here is an example where the delay lines are turned off:

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

You can also have a multi-line comment, or a paragraph of comments. This would be commonly used to describe an upcoming section of code or at the top of a program to explain what the program does and who wrote it. To do this, you start the comment with a /* and end it with a */, for example:

```
/*
  Blink
  Turns on an LED on for one second, then off for one second

  Most Arduinos have an on-board LED you can control. On the
  Leonardo, it is attached to digital pin 13. If you're unsu
  pin the on-board LED is connected to on your Arduino model
  the documentation at http://www.arduino.cc

  This example code is in the public domain.

  modified 8 May 2014
  by Scott Fitzgerald
 */


// the setup function runs once when you press reset or powe
```

```
void setup() {
  // initialize digital pin 13 as an output.
  pinMode(13, OUTPUT);
}

// the loop function runs over and over again forever
void loop() {
  digitalWrite(13, HIGH);   // turn the LED on (HIGH is the
  delay(1000);              // wait for a second
  digitalWrite(13, LOW);    // turn the LED off by making th
  delay(1000);              // wait for a second
}
```

Functions

Functions are blocks of code that are designed to do specific tasks. Every Arduino program contains a *setup* and a *loop* function but you can also create your own functions. We will dive deeper into creating your own functions in future projects but for now you should know that setup runs once on the Arduino when the power is started and the loop function repeats over and over until power is removed from the Arduino.

Spacing

The Arduino language isn't too particular about spacing so it will read:

```
void loop() {
  digitalWrite(13, HIGH);
  delay(1000);
  digitalWrite(13, LOW);
  delay(1000);
}
```

and this code the same way:

```
void loop() {
digitalWrite(13, HIGH);
                    delay(1000);
          digitalWrite(13, LOW);
delay(1000);
}
```

The problem is that we are humans and humans definitely care about the spacing. It provides us context clues as to what is going on in the code.

Making sure that your code is readable from the start is another great habit, for several reasons. Having readable code goes a long way when you may eventually need help with a project. Properly and effectively using spaces will help pinpoint any problems that arise.

Two simple rules to help with spacing is:

- Whenever you see a "{", *tab* the next line in.
- Whenever you see a "}", *shift-tab* the next line out.
    - (shift-tab means to hold the shift key and press the tab key, this will result in your cursor being closer to the left side of the screen (a reverse tab if you will).

Case Sensitivity

The Arduino language **is** case-sensitive. This means that "digitalWrite", "DigitalWrite" and "digitalwrite" are three different things (and only one of them will work).

Pay attention to the uppercase and lowercase combinations and if you see an error like this one,

```
error: 'digitalwrite' was not declared in this scope
```

know that it might be a spelling or a case issue.

Semicolons

At the end of almost every line of code, there should be a semicolon ";". The exception is lines that end with a "{" or "}". Mistakenly forgetting a semicolon will usually result in an error like this:

```
Blink:26: error: expected ';' before 'delay'
```

If you see this error, you should look at the line above it for the missing semicolon. Can you spot the missing semicolon?

```
void loop() {
  digitalWrite(13, HIGH);
  delay(1000);
  digitalWrite(13, LOW)
  delay(1000);
}
```

Summary

That concludes the overview of the Arduino programming language. Next, we are going to learn about some specific language commands. The highlights listed above show just how much programming is like writing in any language. There are specific formats and guidelines that need to be adhered to in order to make your code readable (either by a person or a computer). In my experience, students who have trouble with programming are the same ones who have trouble writing in general (spelling mistakes, capitalization mistakes, etc.).