# Table of Contents

**Introduction**  
3

**Lesson Parts**  
4
- Do you have everything you need?

**The Parts of a Sketch**  
5

**Comments**  
8
- /* Comments */
- // One line comments

**Your First Arduino Statement**  
10

**Your First Arduino Function**  
12
- Wizardry
- What is a function?
- Back to setup()
- The Secret of Setup

**Quiz time!**  
15
- Another Function
- The Legacy of loop()

**Inside the loop()**  
17
- Another Statement!
- digitalWrite
- delay
- Time for a Break!

**Practice: Changing a Delay**  
19
- Changing the delay

**Practice: Changing a Comment**  
23
- Changing Comments
- Always have accurate comments

**Exercises!**  
25

**Lesson Project**  
26
- Lesson Project: Dubstep LED Blinker
- Having trouble?

©Adafruit Industries
Introduction

OK you've gotten your Arduino set up and also figured out how to use the software to send sketches to the board. Powerful stuff!

But...just running example sketches is a little boring. What we really want to do is use our own creativity and skill to write new sketches! That's what we'll be doing in this lesson.

We'll start off easy by just modifying something that already works. You might not expect it, but most of the time as a hardware and software artist will just be modifying someone else's code examples!

This is known as the "Hey let's change something and see what happens" programming technique. It may not be formally taught, but everyone does it!

To start we will venture deep into the Blink sketch, looking at each line and trying to understand what its doing.

Then we will start hacking the sketch, and maybe even meet an internationally-famous DJ and design custom hardware for him!
Lesson Parts

Do you have everything you need?

Not much is needed for this lesson, just a USB cable and an Arduino or compatible.

Make sure you've gone through Lesson #0 () and Lesson #1 () first! This lesson assumes you have installed Arduino IDE software and drivers!

Assembled Arduino board, preferrably an Uno or Duemilanove (or whatever the latest version is)

Arduino compatibles will work but there's a lot of issues with ultra low cost 'Arduino compatibles' (e.g. eBay, Amazon, etc) where they have shoddy substitutions that can bite you later. It's good to have at least one known-genuine Arduino UNO!

Available at Adafruit (http://adafruit.it/50)

OR

You can also use an Adafruit Metro which is a drop-in replacement for the UNO, some components like the LEDs are in different locations.

Available at Adafruit (http://adafruit.it/2488)
USB Cable, any length. The cable should match your Arduino's USB connector. Official Arduino UNOs use USB "Printer Cable", a blocky cable. Some compatibles use USB Mini-B or Micro-B.

USB Cables available at Adafruit ()

A HUUUUUGE number of people have problems because they pick a 'charge only' USB cable rather than a "Data/Sync" cable. Make 100% sure you have a good quality syncing cable. Srsly, I can't even express how many times students have nearly given up due to a flakey USB cable!

The Parts of a Sketch

Start up the Arduino software and open the Blink example sketch, as you did in Lesson #1()

Then buckle up because you're going to take a journey into the heart of Blink!
The sketch itself is in the Sketch Writing/Text Input area of the Arduino software, which you may recall from the previous lesson:
Sketches are written in text, just like an essay, recipe, love letter or other document. When you select Compile/Verify from the menu, the Arduino software looks over the document and translates it to Arduino-machine-language - which is not human-readable but is easy for the Arduino to understand.

Sketches themselves are written in C and/or C++, which is a programming language that is very popular and powerful. It takes a bit of getting used to but we will go through these examples slowly.

Other languages you may have heard of include Scratch, BASIC, Swift, JavaScript, Python, or LOGO. (Part of inventing a new language is coming up with a cool and memorable name)

Let's peel off the text only from the Arduino IDE, so we can focus on the code itself:

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

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

This example code is in the public domain.
*/
```
// the setup function runs once when you press reset or power the board

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 voltage level)
  delay(1000);  // wait for a second
  digitalWrite(13, LOW);   // turn the LED off by making the voltage LOW
  delay(1000);  // wait for a second
}

Notice that some of the text has different coloring and shading! These visual hints will help a lot when reading and understanding a sketch.

Let's take our first small step and look at the first chunk of text at the top.…. 

Comments

/* Comments */

Let’s examine this sketch in detail starting with the first section:

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

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

©Adafruit Industries
This is a comment, it is text that is not used by the Arduino at all, its only there to help humans like us understand what's going on.

You can see, if something is a comment because there is a /* at the beginning and a */ at the end. Anything between the /* and */ is ignored by the Arduino completely.

In this example the person who wrote the comment decided to make it look pretty and add *'s down the side but this isn't necessary. Heck you could put ¯\_(ツ)_/¯ and other small artworks in there, doesn't matter.

Comments are very useful and I strongly encourage every sketch you make have a comment in the beginning with information like who wrote it, when you wrote it and what it's supposed to do. Future-you will thank present-time-you!

Let's move on to the next section

// One line comments

// the setup function runs once when you press reset or power the board
void setup() {
    // initialize digital pin 13 as an output.
    pinMode(13, OUTPUT);
}

The first line is light gray, and has some English text.

// the setup function runs once when you press reset or power the board

Since you've already experienced seeing light-gray English you might think "Hey this looks a lot like a comment!" And you would be right!

Turns out if you want to make a small comment, you can use // as well as /* */. // is often used for short, one line comments. You can use either style depending on whether you want longer or shorter comment text.
The comment is pretty handy, it tells us that coming up is the setup function, which runs once when you press Reset or power up the board. We'll talk about setup some more later.

For now we'll jump right into your first Arduino statement, so keep reading!

Your First Arduino Statement

Lets skip the next line, and go to the next two:

```cpp
// initialize digital pin 13 as an output.
pinMode(13, OUTPUT);
```

First line here is a...comment! Yes - We are already starting to recognize when a line of code is comment or not, nice work!

The second line is...not a comment. This is the first line of actual instruction code. It is also extremely unlike English (or any other human language).

This line of text, `pinMode(13, OUTPUT);`, is what is called a statement, which is basically like a computerized sentence. Much like human sentences end with a . (period), all Arduino sketch statements end with a ; (semicolon)

But what does this statement/sentence mean? Well, coding is very concise, compared to human language. If the Arduino was a person, and you wanted it to unlock the front door, you would probably call out and say something like:

```
Hey, Arduino, could you please unlock the front door for me? Thanks!
```

But that's very wordy. Arduino wants to get the job done fast, so instead of all the niceties you can skip straight to:

```
Set front door to unlocked
```

You're telling it the action you want it to do (set), and the things to do it on (front door) and the result you want (unlocked).

Now, the Arduino does not have a front door but it does have pins:
And what we want to do is basically to change the mode of one of these pins, and set it to an output pin (more on that later!)

So...

Arduino, please change the mode on pin number #13 to be an output.
Thank you.

Shortens to...

Set pin 13 mode to output.

Shortens to...

pinMode (13, OUTPUT);

And, turns out that those spaces are not so important either so we can squish it down to pinMode(13, OUTPUT);

Remember that at the beginning of the example, we were calling out to the Arduino to ask it to do something? That's how we describe this statement: it is a function call to the Arduino. The function of the call is to have it change the mode of the pin just like the function of calling to your roommate to unlock the door.

Here is the structure of a function call:
You might be wondering "OK, wait, so if the Arduino will do anything I ask - why don't I just tell it something like `writeMyHomework(CHEMISTRY)`; and take off the rest of the day?

Ahh, don't we all wish! It isn't like there's a magic wizard inside the Arduino - someone, somewhere had to describe what to do when it gets the `pinMode` request. If you wanted to ask the Arduino to do your Chemistry homework you'd still have to go through and explain to it every detail of how to do the homework. (It would be faster if you just did it yourself.)

But enough chitterchatter - let's continue on and look for more comments and statements!

### Your First Arduino Function

We just looked at an Arduino statement, and we're feeling pretty comfortable with that. Let's back up a little and look where the statement lives:

```c
// the setup function runs once when you press reset or power the board
void setup() {
  // initialize digital pin 13 as an output.
  pinMode(13, OUTPUT);
}
```

OK we've got those comments, each starting with `//`. We know comments are clues that will help you understand what a program does. This one has a cryptic message: `the setup function runs once when you press reset or power the board`.

Which is a little odd because you just learned that `pinMode` is the name of a function you can call. So what does the comment mean by "setup function" - shouldn't it say `pinMode`?

### Wizardry

Remember at the end of the last section we talked about how you couldn't just make up a function request like `writeMyHomework(CHEMISTRY)`? That there is no wizard inside
the Arduino that obeys your commands? That someone, somewhere has to tell the Arduino what a function has to do?

Well, you might be wondering how to do that and now you do because what you are looking at is the function definition of a new function called setup

What is a function?

Personally, I don’t like to use the word function because it’s a little technical. Instead I like to refer to these parts of code as a procedure because it’s a word more people are familiar with.

A procedure is a collection of statements. It’s used to group statements together so that we can refer to them all with one name. It’s just like a procedure that we use to perform a task step-by-step, such as at school, work or home.

Here is the format for a procedure:

<table>
<thead>
<tr>
<th>Returned value</th>
<th>Procedure name</th>
<th>(input values)</th>
<th>{ statements }</th>
</tr>
</thead>
<tbody>
<tr>
<td>void</td>
<td>setup</td>
<td>()</td>
<td>{ pinMode(13, OUTPUT); }</td>
</tr>
</tbody>
</table>

To better understand procedures, let’s use an analogy to the kinds of procedures we’re used to

// a procedure for washing the cat

```c
clean cat wash the cat (dirty cat)
{
    turn on the shower.
    find the cat.
    grab the cat.
    put cat under shower.
    wait three minutes. // wait for the cat to get clean!
    put cat under shower.
    release the cat.
}
```

This is a procedure for washing the cat.
The name of the procedure is wash the cat, it uses a dirty cat as the input and returns a clean cat upon success.

There are two brackets, an open bracket { and a closed bracket }, that are used to indicate the beginning and end of the procedure. Inside the procedure are a bunch of statements, indicating the correct procedure for washing a cat. If you perform all of the statements, in order, then you should be able to turn a dirty cat into a clean cat.

Back to setup()

Looking again at the setup procedure

// the setup function runs once when you press reset or power the board
void setup() {
  // initialize digital pin 13 as an output.
  pinMode(13, OUTPUT);
}

We see that it is named setup and it has no input values and it returns, umm... something called a void

void setup() {

Now you’re probably asking yourself “what is void?” Well that’s a computer-scientist way of saying nothing. That is, this procedure doesn’t return anything. If you studied the Old Testament you may remember the line And the earth was without form, and void in Genesis 1:2(). That verse is telling you there was nothing where the earth would be, because it was void

Having a void return doesn't mean it doesn't do anything, just that it doesn't have a tangible number, cat, or whatever, to show when its complete)

The setup procedure is pretty simple (simpler than washing a cat, that is for sure). It only contains one statement:

pinMode(13, OUTPUT);

That plus the brackets makes the setup function definition complete.

Note that function/procedure definitions are not statements, they do not have a ; at the end, but there are ; after each statement inside the function.
The Secret of Setup

Finally, you may be wondering, what is so special about this function definition?

Glad you asked!

setup() is a special function. As the comment explains, it runs once and only once, when the Arduino first powers up or when it is reset

Quiz time!

OK, skip down and you'll find this section of code:

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

So good - so... time for a Quick Quiz! (Don't worry, you won't be graded)

Quick Quiz #1!

What is this line of code:

```cpp
// the loop function runs over and over again forever
```

(A) A statement  
(B) A single line comment  
(C) A function definition  
(D) A bowl of cereal

Answer: **B it is a single line comment** (Click to reveal the answer!)

OK that was not so hard but hey, it's good practice! Also I'm getting a little hungry.

Now that you have passed the quiz lets skip the next line and....time for another Quick Quiz!
Quick Quiz #2!

What is this line of code:

```c
void loop()
{
...
```

(A) A statement
(B) A single line comment
(C) Part of a function definition
(D) A banana

Answer: C it is part of a function definition (Click to reveal the answer!)

If you didn't get that one right, now might be a good time to review the previous section! Orange you glad it wasn't a banana?

Another Function

You've had some practice with looking at the `setup` function. Now it seems we have encountered another, different function called `loop`. Like `setup`, this function returns nothing, a.k.a `void`, and has no inputs. The body of `loop` - the collection of statements - looks a bit different though:

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

The Legacy of `loop()`

You've already learned that the special function `setup()` determines what code is run when the Arduino first starts up. Not surprisingly, `loop()` is also special

The `loop()` function code runs after `setup()` has finished, and will be called over and over forever until the power is disconnected or the Arduino is reset

Now's a good time for yet another quick quiz:
Quick Quiz #3!

How many statements are in the body of the loop() function?

Answer: There are 4 statements inside of the loop() function definition (Click to reveal the answer!)

Good work passing those quizzes. If you weren't sure about what each line of code was, or the number of lines in the function, now is an excellent time to review the previous sections

Once you feel comfortable and you know those answers without hesitation, lets delve deep into loop...

---

Inside the loop()

We're on a roll understanding statements and comments so lets go to the next line - the first statement inside of loop(). You guessed it, its time for a quick quiz!

Quick Quiz #4!

What is this line of code:

digitalWrite(13, HIGH); // turn the LED on (HIGH is the voltage level)

(A) A statement
(B) A single line comment
(C) Part of a function definition
(D) Uh...both?

Answer: D! Yes it is a line that has both a statement and a comment (Click to reveal the answer!)

Really threw you for a loop, didn't it? (Ha, that was my little function name joke) But now you know, you can combine statements and comments. The comment is only the part after the // . The part before the // is the statement. As you can tell, if you have a really short comment, it can be convenient to double them up for compactness and clarity.
Another Statement!

If we separate it out, the statement looks like this: `digitalWrite(13, HIGH);` Which looks a lot like...A function call statement? (That's right!)

We've already been introduced to `pinMode` so now you will meet her friend, `digitalWrite`.

- `pinMode` is what you request of the Arduino when you want to set the mode of pins.
- `digitalWrite` is what you can request of the Arduino when you want to change the setting of the pins.

To think of it in terms of that front door we spoke of earlier, `pinMode` is like asking the Arduino to lock or unlock the door and `digitalWrite` is like asking it to open or close the door. You have to unlock the door before you open or close it!

`digitalWrite`

The nice thing about good comments is they really help you understand what the statement does.

For example, the comment next to this statement

```
digitalWrite(13, HIGH);  // turn the LED on (HIGH is the voltage level)
```

Tells you what the code actually does: it turns on the LED so it's lit up and shining. Likewise, if you skip down two more statements, you'll see

```
digitalWrite(13, LOW);    // turn the LED off by making the voltage LOW
```

Which, apparently, turns it off.

Notice that the second input to the `digitalWrite` function has changed from `HIGH` to `LOW` but that the first input, 13, is the same!

These two statements act like someone flipping a switch on and off. On (HIGH), off (LOW). Simple!
delay

In between the digitalWrite calls are two other lines of code. To our luck, they are identical:

delay(1000);  // wait for a second

No need for a quiz: you are an expert at this now! You are looking at a statement with a short one line comment. Moreover, the statement is a function call.

delay is the third function you are meeting today. delay is what you can request of the arduino when you want to hold on for a moment and do nothing.

Which might seem a little odd - until you realize that many of the function calls that you can request complete in thousandth's of a second! In order to really see the LED on and off, the Arduino needs to pause:

- Turn on the light
- Wait a second
- Turn off the light
- Wait a second
- (and repeat!)

In particular, delay requires only one input - the amount of time to wait. That time is a number of milliseconds. Which are thousandths of a second (not whole seconds). So delay(1000); is not 1000 seconds, minutes or hours - it is 1000 * thousandths, a.k.a. one second.

Time for a Break!

You just did some serious reading, learning about comments and statements and you might be getting a little antsy. Now is a good time for a break where you will start putting what you learned into practice. Let's go to the next section where we will practice modifying a sketch.

Practice: Changing a Delay

Remember earlier when we mentioned the "Hey let's change something and see what happens" programming technique? It is time to put that into practice!
Changing the delay

Go back to your Arduino IDE window and highlight the 1000 in the first `delay` statement:

Then change that from a 1000 to a 500
Then change the second delay as well:
Now Save it as a new Sketch called MyBlink (or really anything but a descriptive name is wise). When you do that you'll get the warning that it's read-only.

Now before you upload this new, modified sketch to your Arduino I would like you to stop and think!

- What do you think will be the effect of this change?
- What will be different and what is the same?

Once you have meditated and you think you are sure of your answer. Click the Verify button (the checkmark button in the quick action button bar) and then the Upload button (arrow button, to the right of checkmark) to send the sketch to your Arduino. Check the status bar to make sure the upload was successful

Were you right about your prediction?

The effect of the change is... now the LED is on for half a second, and off for a half second! The LED is blinking twice as fast.
If the LED is not blinking faster, check:

- Did you make the changes to the delay function calls to make them 500?
- Did the compile/verify complete successfully? Go back to Lesson #1 to review if necessary
- Did the upload complete successfully? Go back to Lesson #1 to review if necessary

OK, not crazy exciting yet but it’s all about small steps.

Practice: Changing a Comment

Changing Comments

Next, let's do something different. Let's modify a comment. Go down to the last comment, and change it to say "wait for five seconds" like so:
Before we upload this sketch, what do you think the comment change will do to the behaviour of the Arduino and the blinking LED? Take a good guess, then upload the new sketch.

After uploading, what was the change you observed?

There was no change! (Click to reveal answer)

Don't forget: the comments in a sketch do not affect the code itself, they are just your notes and are not converted or sent to the Arduino.

Always have accurate comments

But now you have a comment that does not match the statement. Which should annoy you greatly, like an itch you cannot scratch. Comments should always be accurate.

So! Your job, if you choose to accept it, is to now modify the sketch one more time. This time, you will modify a statement to make it so that the comment is correct.
Before you reveal the answers, try to solve this quest on your own and upload the code to your Arduino to test it out.

Which statement must be modified?

The statement right before the comment:
```
delay(500); // wait for five seconds
```

What needs to be changed?

Instead of `delay(500)` we should change the input to the delay function into `delay(5000)` which will make the Arduino wait 5000 milliseconds (5 seconds)

Exercises!

Alright! You really are mastering Blink. This is a really big deal - and to show you how powerful you have become we're going to have you try these exercises to explore software and hardware:

Exercise 1.

Modify the code so that the light is on for 100 milliseconds and off for 900 milliseconds. This makes for a nice once-per-second second timer.

Exercise 2.

Modify the code so that the light is on for 50 milliseconds and off for 50 milliseconds. What happens?

*Intense strobe action!*

Exercise 3.

Modify the code so that the light is on for 10 ms (the shorthand for milliseconds) and off for 10 ms. What happens?

*The light is no longer blinking*

Now pick up the Arduino and gently wave it back and forth, in a dark room. What happens?

*The LED creates a dashed trail of light in the air.*
What do you think is happening here?
The LED is blinking, but its blinking so fast that our eyes can't pick it up, so it looks like a blur. When the Arduino is waved in the air, we see streaks of light from the blinks.

Like I said earlier, the Arduino can be really fast - faster than you can blink or your eyes can see!

Lesson Project

OK Students! Time to take everything you've learned and put it together into a final project.

Your homework for tonight is to come up with the topic for your final project and then bring it in to show tomorrow to the rest of the students.

CLASS DISMISSED

While walking home from school, you nearly walk right into Skrillex! 

"Wow," you say. "It's the famous American electronic music producer and DJ Skrillex"

"That's right," he replies. "It is a pleasure to meet you"

You apologize for almost bumping into him: "Sorry for being so distracted...I'm just thinking of what I should do for my electronics class lesson project. But all I know is how to blink an LED!"

"You know how to blink an LED?" Skrillex is impressed! "Maybe you can help me with a problem I have, and it could also be your lesson project..."

Photo Credit: Michael Nusbaum, www.mikenusbaum.com
Lesson Project: Dubstep LED Blinker

Skrillex is known for making electronic music in the Dubstep genre. This music has a tempo of 140 beats per minute (bpm). Your lesson project will help out a dubstep musician by pulsing an LED at about 140 bpm. (It is OK if your tempo is off by a little, just make sure it is within half a bpm)

What code sketch should you upload to your Arduino?
Hint: Remember that the LED on and off time together count as one beat.

Once you're done, celebrate by listening to some Skrillex tunes, knowing that you had a part in creating this music legend!

Having trouble?

Here's one way to look at building this project (But we really do suggest trying to write the code on your own!)

Let's work out the delays necessary to have the LED blink 140 times per minute.

140 bpm is the same as 140 / 60 = 2.3333... beats per second.

But we don't want beats-per-second we want seconds-per-beat so flip it around: 1/2.3333 = 0.428 second per beat. That is the same as 428 milliseconds
As the hint hinted, you need to have the LED on for half that time, and off for half. So the delay between each digitalWrite has to be \( \frac{428}{2} = 214 \) milliseconds.

Change both delay statements to `delay(214);`

Working backwards, \( 214 \) milliseconds * 2 = 428 milliseconds per beat or 0.428 seconds

\[ \frac{60}{0.428} = 140.1869 \text{ bpm} \], which is close enough!