

M.A.D.E.

INTERMEDIATE E-TEXTILES GUIDES

FOR **MUSIC** | **ART** | **DESIGN** | **EXPERIENCES**

FADING LEDs

Copyright © 2019 University of Pennsylvania



This work is licensed under the Creative Commons Attribution-NonCommercial 4.0 International License. To view a copy of this license, visit <http://creativecommons.org/licenses/by-nc/4.0/> or send a letter to Creative Commons, PO Box 1866, Mountain View, CA 94042, USA.

©2019 Google LLC All rights reserved. This curriculum module and research was supported by funding from Google's CS-ER program. Google and the Google logo are registered trademarks of Google LLC. The University of Pennsylvania is a recipient of Google's CS-ER program. The Google Grants program supports registered nonprofit organizations that share Google's philosophy of community service to help the world in areas such as science and technology, education, global public health, the environment, youth advocacy, and the arts. Google Grants is an in-kind advertising program that awards free online advertising to nonprofits via Google AdWords.

Any opinions, findings, and conclusions or recommendations expressed in this guide are those of the authors and do not necessarily reflect the views of Google, the University of Pennsylvania, Utah State University, University of Oregon, or Exploring Computer Science.

Please cite this work as Fields, D. A., Amely, J., Jayathirtha, G., Lindberg, L., Lui, D. & Kafai, Y. B. (2019). *M.A.D.E. Intermediate E-Textiles Guide: Fading LEDs*. Available online at <http://exploringcs.org/e-textiles/modules>.

EMMA ECCLES JONES
COLLEGE of EDUCATION
and HUMAN SERVICES
UtahStateUniversity.

Graduate School of Education
PennGSE



Exploring
Computer
Science

MADE - FADING LEDs

What you need to know first:

This module assumes that users have basic experience with making lighting patterns in Arduino (i.e., with `digitalWrite()` and `delay()` commands). For more introductory material, see the [Exploring Computer Science e-textiles curriculum unit](#), Buechley & Lui [Sew Electric](#), or other [introductory Arduino guides](#). Other ECS E-Textiles modules as well as supporting code samples that may be referenced in this text can be found through <http://exploringcs.org/e-textiles/modules>.

Table of Contents

[Fading LEDs](#)

- [1. What do you mean by “fading” an LED?](#)
 - [2. Task #1 - Make an LED Fade On | Simple](#)
 - [3. How to Use a for\(\) Loop](#)
 - [3.1 EXAMPLE Blinking with a for\(\) Loop](#)
 - [3.2 CHALLENGES | All the Blinks](#)
 - [4. How to Fade an LED with a for\(\) Loop](#)
 - [4.1 EXAMPLE | Fading ON for\(\) Loop](#)
 - [4.2 Debug Fading](#)
 - [4.3 CHALLENGES | Fading](#)
-

1. What do you mean by “fading” an LED?

You have turned on an LED, and you also learned how to make it blink. Is there anything else you can do with an LED? A new thing to try is to program an LED to work like a dimmer switch. That way you can set it at any point of brightness: off—low bright—medium bright—very bright—and any point between. You may have done this with the `analogWrite()` function already, and found that you can set pins **3 6 9** and **10** on the Circuit Playground (CP) to any number from **0** to **255**. With those pins, you can also do slooowly turning on or slooowly turning off. Want an LED to look like a heartbeat? By fading on and off, you can make that effect!

NOTE: Only certain pins on Arduino microcontrollers can “fade” lights: 3, 5, 6, 9, 10, and 11. Be sure that your LEDs are connected to these pins and not others, or they will only appear to blink instead of fade. Some Arduino microcontrollers like the LilyPad or Circuit Playground do not have all six fade-able pins available, so be sure to double check!

2. Task #1 - Make an LED Fade On | Simple

1. Write out the code to make an LED blink. If you need a bit of a refresher, you can find it by navigating to Blink_analogWrite (found at <http://www.exploringcs.org/e-textiles/modules/supporting-code>).
2. In the Setup Section, add `int led = 13;`. This uses the red LED that is already on the Circuit Playground. You might instead hook up an LED to pins **3 6 9** or **10** and **Ground**—just remember to change your `int led` pin number to match!
3. This is a good time to **Save** your file as myBlinkCode. Then **Save As...** and name your new file myFadingCode. This keeps the code you already typed in but starts a new file with it so you don’t accidentally change your myBlinkCode file.
4. Play with making an LED fade completely manually, aka the “lots of lines of code” method. It will look something like this...

```
.....
void loop(){
  analogWrite(led, 10);
  delay(10);
  analogWrite(led, 20);
  delay(10);
  analogWrite(led, 30);
  delay(10);
  analogWrite(led, 40);
  delay(10);
  analogWrite(led, 50);
```

MADE - FADING LEDs

```
//... until you get bored, or hit 255, whichever comes first  
}
```

.....
5. Load your code to see your initial LED fade on attempt.

3. How to Use a `for()` Loop

The “Lots of code” method is incredibly tedious, even with copy and paste! If only there existed a way that made it so we could stop typing so much. Ah ha! That is what a `for()` loop is for! A `for()` loop repeats a block of code a specific number of times, such as blinking an LED **10** times. Head on down to [Section 4: How to Fade an LED with a `for\(\)` loop](#) if you already know this well.

A `for()` loop **must** be placed inside a program section! Either the `void setup()` or `void loop()` will work. If you want it to repeat at any time, use the `void loop()` section. If it is outside of that and it is not written as part of a new function (which you haven't learned yet!) it won't work. If you write your own function, then the text `myfunction()`; has to be called (typed into) in either of those main program sections or it will not run.

3.1 EXAMPLE Blinking with a `for()` Loop

This is a code breakdown of a `for()` loop that blinks an LED **10** times and then stops. Keep in mind that it won't look like it stops, because it is also in the `void loop()` and will keep looping while the microcontroller is on. Compare this to the “Lots of code” method—they do the same thing!

```
void loop(){  
  for (int i = 0; i < 10; i=i+1){  
    digitalWrite(led, HIGH);  
    delay(1000);  
    digitalWrite(led, LOW);  
    delay(1000);  
  }  
}
```

There is a lot of stuff happening here! Below it is broken down line by line:

Line 1, `for()` loop:

MADE - FADING LEDs

```
      Initialization      Condition      Increment
      ↓                  ↓              ↓
for (int i = 0; i < 10; i=i+1){
```

Let's start with the three sections in the parentheses.

1. **Initialization**, `int i = 0`. This part says, "Create a variable and name it `i`. Set `i` to `0`, so that whenever you see `i`, think `0`. (This number changes!)"
2. **Condition** to be met, `i < 10`: how many times you want your code to repeat. As long as the equation "`i` is equal to **less than 10**" is **TRUE**, continue to the code in the Action Block. Once "`i < 10`" turns to **FALSE**, stop right here!
3. **Increment**, `i=i+1`. If you haven't stopped yet, add `1` to whatever number `i` currently is.

Zoom back out of the parentheses. "`for`" the code to continue, the equation in the parenthesis has to be **TRUE**. If it is still true, continue to the **Action Block**, the code within the curly braces `{}`.

Lines 2 through 6, the **Action Block**:

```
      Action Block
      ↓
digitalWrite(led, HIGH);
delay(1000);
digitalWrite(led, LOW);
delay(1000);
}
```

The code here blinks an LED on and off in `1000` millisecond increments (1 second), followed by the closing curly brace `}` to end the **Action Block** of the `for()` loop.

Line 7:

```
}
```

This one is easy, BUT VERY IMPORTANT! The `void loop()` has its own set of curly braces, and if you forget that ending curly brace `}`, your code won't work! The same happens if you have too many ending curly braces.

A note on **incrementing**:

The overall goal of incrementing is to eventually exit out of a `for()` loop. We control when that exit happens with incrementing. You can see in the setup of the `for()` loop above that we use `i=i+1`, but we can also use any other number, including negatives. If your `for()` loop never runs, check your `i`! If `i` solves to false on the first run through, your **Condition** isn't met, and the `for()` loop will not run. Remember that you need to change the `for()` loop conditions for the logic to still make sense!

MADE - FADING LEDs

Incrementing Examples:

<pre>int i = 0; i < 10; i=i+1</pre>	Same as the example above. i is 0 ; as long as i is less than 10 , add 1 .
<pre>int i = 10; i > 0; i=i-1</pre>	Reverse of the first example. i is 10 ; as long as i is more than 0 , subtract 1 .
<pre>int i = 255; i >= 0; i=i-5</pre>	i is 255 ; as long as i is more than or equal to 0 , subtract 5 from it.
<pre>int i = 0; i < 10; i++</pre>	Same as the first example. This one uses a code shortcut. You can use i++ or i-- if you only need to increment by 1 .

3.2 CHALLENGES | All the Blinks

With the in-depth breakdown of using a **for()** loop for blinking, now is a good time to put that knowledge into action. Try out the challenges below to cement your understanding!

- VERY EASY** Copy the “Blinking with **for()** loop” example from above (10 one-second blinks, with one second between each blink).
- EASY** Change the blinking pattern to make it slower or faster.
- MEDIUM** Add one line of code after the **for()** loop to create a pause before the blinking pattern starts again.
- MEDIUM** Change how often the blinking pattern repeats before pausing (for instance, 3 blinks or 5 blinks before repeating).
- MEDIUM** Change some numbers to make the LED blink in an irregular pattern, for instance with the timing (rhythm) of a galloping horse or a heartbeat.
- HARD** Make a blinking code for a letter in morse code. (See the [Morse Code Chart](#) for each letter’s value in blinks and delays, and the [Morse Code Translator](#) if you need it.)
- HARDER** Add another **for()** loop so that you have two letters (for example, “HI”). Use a pause between **for()** loops so that each letter is distinct. You might need to change variables.
- HARDEST** Using **for()** loops, make a short word in Morse Code (for instance, “HELLO,” “SOS,” “LOVE”). Call a function for each letter of your name to write it out in Morse Code blinks.

4. How to Fade an LED with a **for()** Loop

Now that you have used all the parts of the **for()** loop, let’s make an LED fade on. We have been using **i** to count each repetition of the **Action Block**. The trick here is that we can *also* use it to control the brightness of the LED.

MADE - FADING LEDs

When you use `analogWrite()`, you can control how bright an LED is within a range of 0 to 255. For example:

- `analogWrite(led, 0);` //turns the LED off
- `analogWrite(led, 255);` //turns the LED to maximum brightness
- `analogWrite(led, 100);` //turns the LED to a medium brightness


To make an LED fade on, we want to start the LED at 0 and slowly (incrementally) increase it to 255. To make it fade off we do the opposite, starting at 255 and decreasing (decrementing) it until it is 0. Example 4.1 is one way to do this.

4.1 EXAMPLE | Fading ON `for()` Loop

```
.....  
int led = 13;  
  
void setup(){  
  pinMode(led, OUTPUT);  
}  
  
void loop(){  
  for (int i = 0; i < 255; i = i+15){  
    analogWrite(led, i); //set the starting brightness of the LED  
    delay(500); //wait 500 milliseconds to see the dimming effect  
  }  
}  
.....
```

We are increasing this `for()` loop by 15 for each repeat (`i = i+15`). It only stops when `i < 255` returns as `False`, which also happens to be when the *brightness* is at the top of the range, at 255.

The most important line of code here is:

`analogWrite(led, i);`  Also used
for Brightness

Specifically, putting `i` where we would normally put the analog brightness number. This makes it so that the analog brightness keeps going up by 15 each time the Action Block is repeated!

MADE - FADING LEDs

4.2 Debug Fading

Start with the debugFADE code. (This code and code below found at <http://www.exploringcs.org/e-textiles/modules/supporting-code>).

- ❑ Debug (fix) the fading section to work correctly (see the directions in the code!).
- ❑ Copy/Paste the FADE ON code, and adjust it to make it FADE OFF.

Check your work: debugFADE_ANSWER code.

4.3 CHALLENGES | Fading

- ❑ **EASIEST** Get the debugFADE code to work correctly.
- ❑ **EASY** Make the LED fade very fast or very slowly.
- ❑ **MEDIUM** Code two LEDs to fade on and off at the same time.
- ❑ **HARD** Code one LED to fade on/off followed by the second LED fading on/off.
- ❑ **HARDER** Code one LED to fade on, followed by the other LED fading on, then the first fades off, followed by the second fading off.
- ❑ **SUPER DUPER HARD** Fade two LEDs in reverse of each other, *at the same time* (the first fades off as the second fades on, then in reverse). *Hint*: you want the opposite of what `i` is currently at for the second LED. This involves a little on-the-fly variable math. Check your work with this code: FADE_SuperHard.