

What Will You Learn?

In this activity you will use an LED to send messages by using Morse code.

Why Should You Learn This?

Morse code was an early form of communication that used a series of dots and dashes as a code for letters. Often the code was sent over a wire as sound and then translated by someone on the other end, but the code could also be transmitted using light. Today's challenge will extend your knowledge of making an LED blink and will push you to make the LED blink in a specific pattern.

Here's What You'll Need:

Arduino Microcontroller



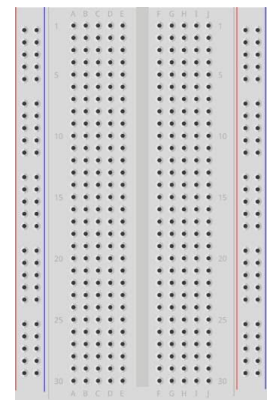
fritzing

LED



Notice that one of the wires of the LED is slightly longer than the other, the long leg is the positive side and the short leg is the negative side. This will be important when you go to plug in our LED.

Breadboard



fritzing

Red Wire



Black Wire



A to B USB Cable



A to B USB Cable
You will power the Arduino through the USB cord.

Resistor



fritzing

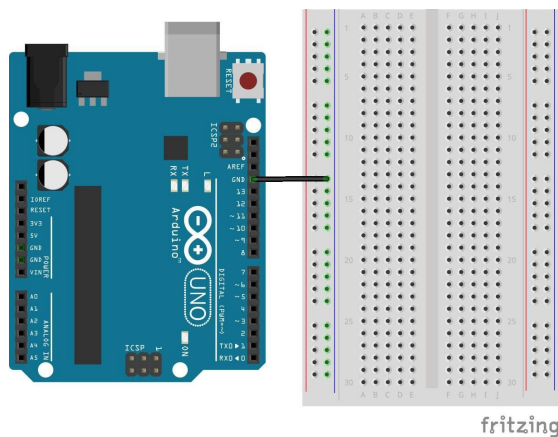
Use a range of 100 – 400 ohms, (see "LED Circuit" activity to see why).

Safety First!

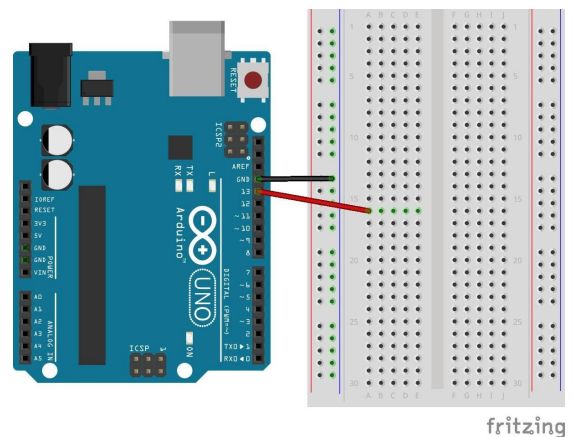
Never allow the red and black wires to touch together while they are connected to a powered microcontroller, as that creates a short circuit. A short circuit can potentially cause the wires and/or microcontroller to get hot enough to burn the skin. In addition, in the event of a short circuit, the microcontroller can potentially catch on fire.

Let's Get Connected!

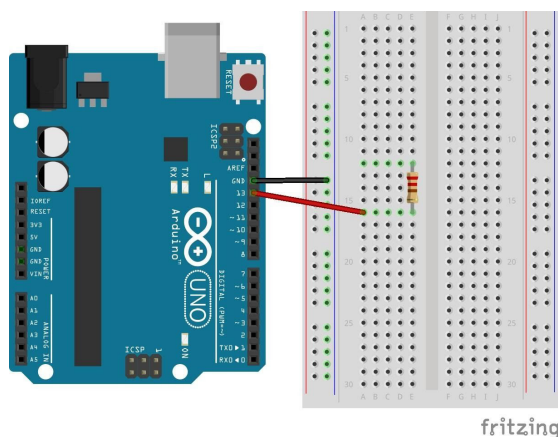
1. Start by connecting the black wire from GND to the negative rail of the breadboard.



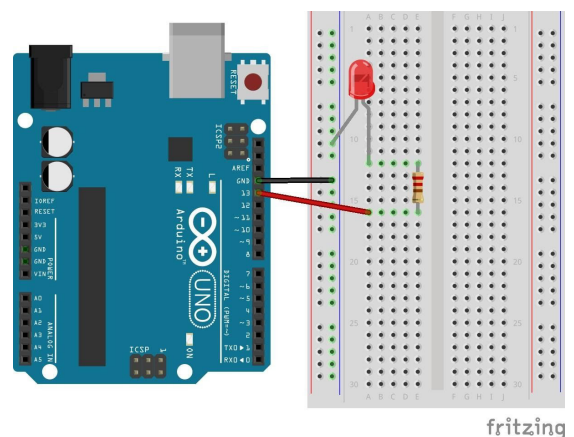
2. Next, connect the red wire from pin 13 to any row on the breadboard (row 16 is used below).



3. Add the resistor. Use it to connect row 16 (where the red wire is) to any other row on the breadboard, (row 12 is used below).



4. Lastly, place the LED. The long leg (positive) will go in the row with the resistor (row 12) and the short leg (negative) will go in the negative rail. Be careful that the LED and resistor do not touch as this could cause the LED to burn out.



Time to Write Some Code!

First, you need to know what an 'S' and 'O' look like in Morse code. S is 3 dots and O is three dashes. Since you are using light, you need to decide how long a dash is and how long a dot is. It is recommended to make a dash 1 second (delay = 1000 ms) and a dot about 1/3 of a second (delay = 300 ms). Next, think about having the light off for a short time (use 300 ms again) between each dot and dash, and off for a longer time (use 2000 ms) between letters.

STOP here and try to figure out how to write the program yourself. Start by opening the blink code (in File > Examples > Basics > Blink) and then modify the code to make the light blink an SOS pattern in Morse code. If you get stuck, come back and read some of the hints below.

HINT 1: You may make mistakes when you are typing the same lines of code over and over again. To avoid any typos, copy (Ctrl + C) and paste (Ctrl + V) a set of 4 lines of code to turn the LED on and off. Just remember that all your lines of code in the loop must stay between the open and closed brackets. Once you've copied the code to turn an LED on and off, you can change the values of the delays in each line to make a sequence of flashes that stands for SOS.

HINT 2: Below is the code to make the "S" in Morse code followed by a space before the "O":

```
void setup() {  
    pinMode(13, OUTPUT);  
}  
  
void loop() {  
    digitalWrite(13, HIGH);           //First dot in the "S" code  
    delay(300);  
    digitalWrite(13, LOW);           //Delay between dots/dashes  
    delay(300);  
    digitalWrite(13, HIGH);          //Second dot in the "S" code  
    delay(300);  
    digitalWrite(13, LOW);           //Delay between dots/dashes  
    delay(300);  
    digitalWrite(13, HIGH);          //Third dot in the "S" code  
    delay(300);  
    digitalWrite(13, LOW);           //Long delay after letters  
    delay(2000);  
}
```

Hint 3: The final code is below. Your delay values may be slightly different from those below, and that is ok. Just test it out and see what happens!

```
void setup() {  
    pinMode(13, OUTPUT);  
}  
  
void loop() {  
    //3 dots, the code for the letter "S"  
    digitalWrite(13, HIGH);  
    delay(300);  
    digitalWrite(13, LOW);  
    delay(300);  
    digitalWrite(13, HIGH);  
    delay(300);  
    digitalWrite(13, LOW);  
    delay(300);  
    digitalWrite(13, HIGH);  
    delay(300);  
    digitalWrite(13, LOW);  
    delay(2000);  
    //3 dashes (long blinks), the code for the letter "O"  
    digitalWrite(13, HIGH);  
    delay(1000);  
    digitalWrite(13, LOW);  
    delay(300);  
    digitalWrite(13, HIGH);  
    delay(1000);  
    digitalWrite(13, LOW);  
    delay(300);  
    digitalWrite(13, HIGH);  
    delay(1000);  
    digitalWrite(13, LOW);  
    delay(2000);  
    //3 dots, the code for the letter "S"  
    digitalWrite(13, HIGH);  
    delay(300);  
    digitalWrite(13, LOW);  
    delay(300);  
    digitalWrite(13, HIGH);  
    delay(300);  
    digitalWrite(13, LOW);  
    delay(300);  
    digitalWrite(13, HIGH);  
    delay(300);  
    digitalWrite(13, LOW);  
    delay(5000);    //this extra-long delay signals the end of the message  
}
```

What Will You Learn?

Learn to make the Morse code program from “SOS LED Challenge Part 1” more efficient by using a programming tool called a “function”.

Why Should You Learn This?

By now you should have completed the SOS LED Challenge Part 1. Ready for another challenge? Although the program that you wrote in part one blinked an LED to make the SOS message, you probably had to repeat the same instructions over and over, which wasn't very efficient. Part 2 of the SOS challenge will teach you how to make your code simpler by using a programming tool called a “function”.

An Introduction to Functions

A function is a chunk of code that performs a useful action, and could be re-used elsewhere in a program. For example, you might want a chunk of code that blinks an LED on and off, to make one Morse code “dot”. The code inside the function would look like this:

```
digitalWrite(13, HIGH);  
delay(300);  
digitalWrite(13, LOW);  
delay(300);
```

Instead of writing this code over and over, a function lets you make a shortcut for the whole chunk of code by giving it a name. Functions have curly brackets to indicate the beginning and end of the chunk of code. The code below blinks an LED on and off, but now the function has been named “dot”.

```
void dot() {  
    digitalWrite(13, HIGH);  
    delay(300);  
    digitalWrite(13, LOW);  
    delay(300);  
}
```

Notice the word “void” and the empty set of parentheses after the word “dot”, just like when you use “void setup)” or “void loopQ”. The ‘void’ and empty parentheses indicate the values that would be the output and input of a function, respectively. In this exercise, the focus is on functions that don’t have values passed in and out of them. This is why you see the word ‘void’, meaning it’s not outputting a value, just performing actions, and the parentheses have nothing inside of them because the function doesn’t need a value to process.

Now, when you want your LED to blink, you can use one line of code to “call” your function instead of writing out the entire chunk of code again. Typically, functions are called into the loop function by writing this line of code:

```
dot();
```

Functions are very powerful and convenient! They let you reuse chunks of code very efficiently. Now that you’ve seen how a “dot” function is made, try making a function that flashes an LED as a Morse code dash (keeping the LED on for a longer period of time).

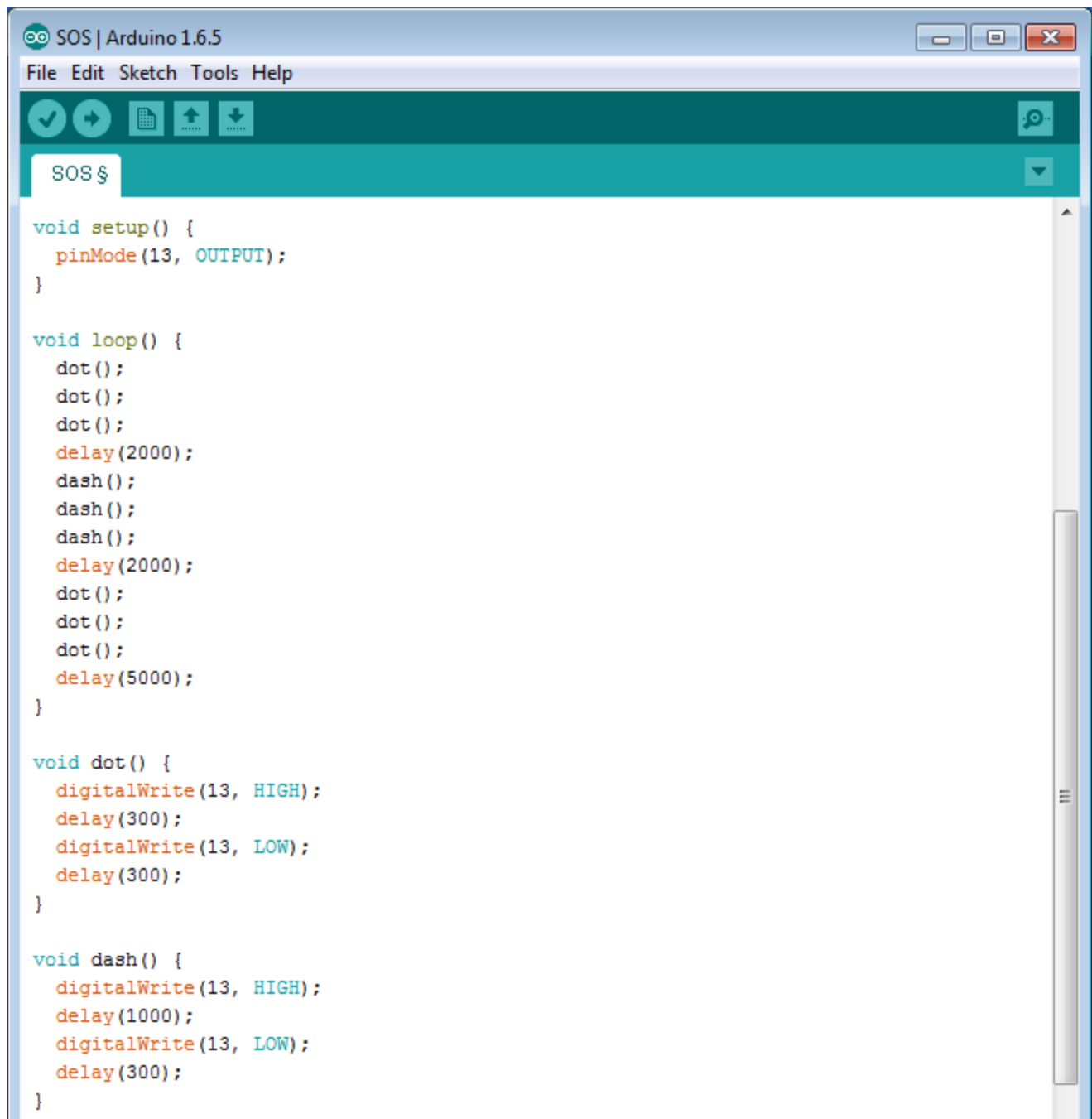
Using Functions for Morse Code

Now that you have your dot and dash functions, the Morse code program becomes a lot simpler. We just need to call our functions to make dots and dashes. Since the Morse code for S is dot dot dot, the first part of our loop would look like this:

```
void loop() {  
    dot();  
    dot();  
    dot();  
    delay(2000);  
}
```

At the end of the letter, you’ll want to add a delay of 2 seconds since you are at the end of a letter (switching from S to O).

Try to finish writing the SOS code using the functions you created. If you get stuck, the full version of the code is posted on the next page. Have fun!



What Will You Learn?

Learn how to write a program that can easily make Morse code by combining multiple functions.

Why Should You Learn This?

Now that you have learned how to make and call a function, try to make some more complex functions that can send Morse code for each letter in the Alphabet. This will make it easy to write Morse code messages!

Using Functions in Other Functions

Start with the code completed in the SOS LED Challenge Part 2, where you made functions for a Morse code dot and dash. In this activity, you are going to add to that code by making functions for each letter in the alphabet, starting with S and O.

Here's an example of a function called "morseS" that makes the LED blink the Morse code for the letter S. Notice that this function uses the "dot" function to make the LED blink on and off once.

```
void morseS() {  
  dot();  
  dot();  
  dot();  
  delay(2000);  
}
```

Using functions inside other functions shows you how powerful this programming tool can be. Using functions makes your code more efficient (you send a Morse code "S" using one line of code instead of seven), and it makes your code easier to understand. Looking at the morseS function, it is clear that an S is made up of three Morse code dots.

Making Functions for all of the Letters in SOS

Try making your own function that makes the LED blink the Morse code for an O (hint, you will need to use the “dash” function instead of the “dot” function).

Once you have the functions written for the S and O, you can easily flash the Morse code for SOS by calling the function for each letter.

For example:

```
void loop() {  
  morseS();  
  morseO();  
  morseS();  
}
```

The SOS code is on the last page of this activity.

The full Morse Code alphabet is below. Use this to create functions for each letter of the alphabet, remembering that each one should use the dash and dot functions.

A: ●-	J: ●---	S: ●●●
B: -●●●	K: -●-	T: -
C: -●-●	L: ●-●●	U: ●●-
D: -●●	M: --	V: ●●●-
E: ●	N: -●	W: ●--
F: ●●-●	O: ---	X: -●●-
G: --●	P: ●--●	Y: -●--
H: ●●●●	Q: --●-	Z: --●●
I: ●●	R: ●-●	

Once you have the entire alphabet written in Morse code functions, why settle at SOS? You could write your name in Morse code or the name of your school, or secret notes to a friend. The possibilities are endless!

The full code for the SOS Challenge is below.

A screenshot of the Arduino IDE interface. The title bar reads "SOS | Arduino 1.6.5". The menu bar includes "File", "Edit", "Sketch", "Tools", and "Help". Below the menu bar is a toolbar with icons for checking, undo, redo, and uploading. The main text area shows the following code:

```
void setup() {  
  pinMode(13, OUTPUT);  
}  
  
void loop() {  
  MorseS();  
  MorseO();  
  MorseS();  
}  
  
void dot() {  
  digitalWrite(13, HIGH);  
  delay(200);  
  digitalWrite(13, LOW);  
  delay(200);  
}  
  
void dash() {  
  digitalWrite(13, HIGH);  
  delay(500);  
  digitalWrite(13, LOW);  
  delay(200);  
}  
  
void MorseS() {  
  dot();  
  dot();  
  dot();  
  delay(1000);  
}  
  
void MorseO() {  
  dash();  
  dash();  
  dash();  
  delay(1000);  
}
```