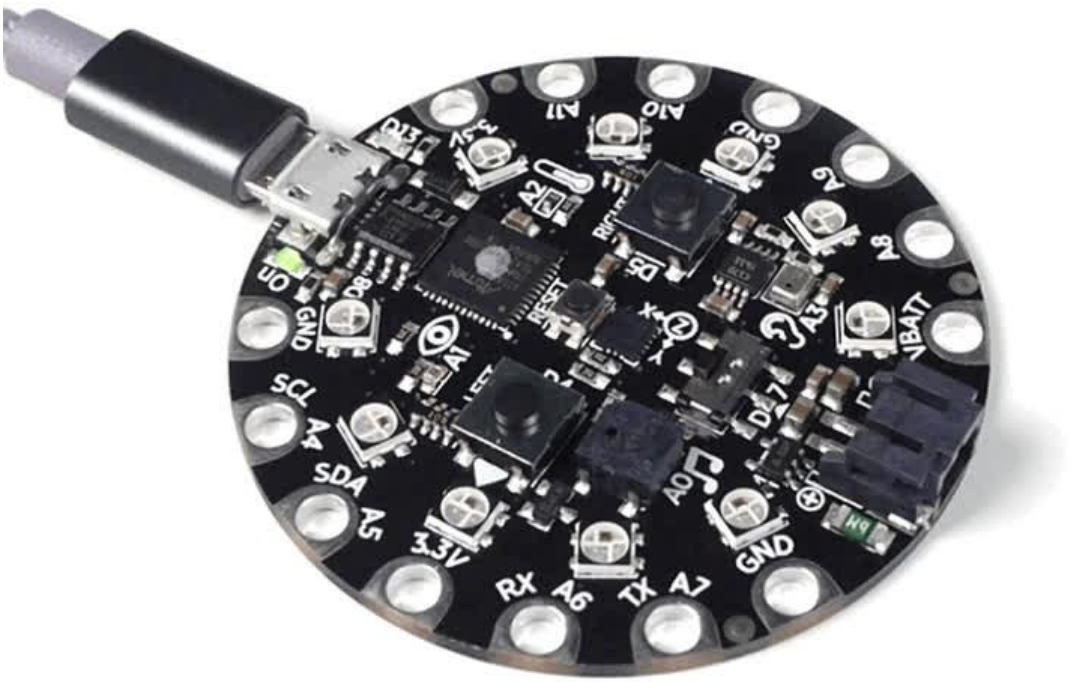
Circuit Playground Morse Code Flasher

Created by Collin Cunningham



https://learn.adafruit.com/circuitplayground-morse-code-flasher-makecode-circuit-python

Last updated on 2022-12-01 03:07:15 PM EST

Table of Contents

Overview 3

- What You'll Need
- What is Morse Code?

Program It 5

MakeCode 5

- Customize the Message
- A Tour of the Code
- On Start
- Functions

CircuitPython 11

- . . . - - - . . .
- A Tour of the Code
- Imports
- Variables
- MorseFlasher Object
- Let's do this!

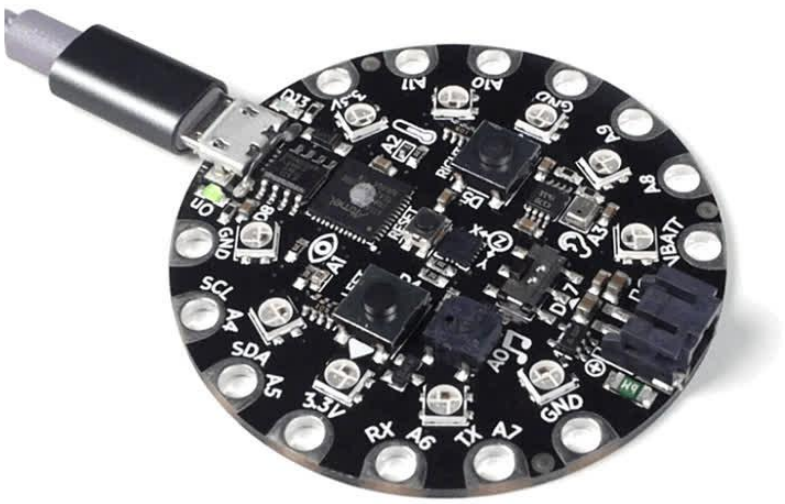
Change it! 17

- Change the message
- Customize the color of the LEDs
- Adjust the speed of the LED flashes
- More ideas

Overview

This project uses a Circuit Playground Express board to convert text into Morse Code using Circuit Playground's built-in neopixel LEDs.

Use it as an emergency beacon, a spy communicator, or to help you learn Morse code for yourself.



What You'll Need

To build this project, you'll need the following items:

- Circuit Playground Express ()
- Micro USB cable ()
- Your desktop computer

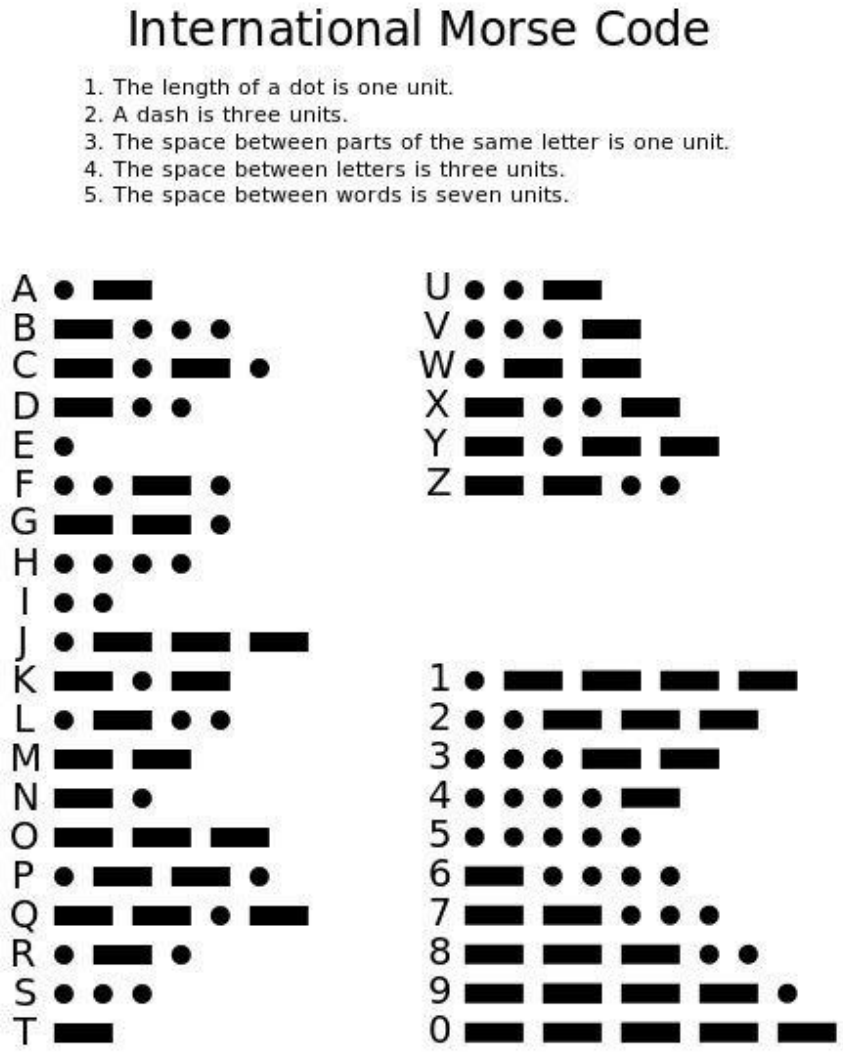
What is Morse Code?

Wikipedia defines () Morse Code as the following:

> Morse code is a method of transmitting text () information as a series of on-off tones, lights, or clicks that can be directly understood by a skilled listener or observer without special equipment. It is named for Samuel F. B. Morse (), an inventor of the telegraph ().

So, Morse code is essentially a way to communicate using short and long tones or flashes of light.

Below is a chart that shows each letter of the alphabet along with its translation into Morse 'dots' and 'dashes':



The above chart also provides rules for the length of each Morse code - for example; one dash should last three times as long as one dot.

With those basic rules in mind, let's program the Circuit Playground board to display messages as Morse code flashes of light ...

Program It

To get the Morse Flasher running, you'll need to program your Circuit Playground Express with the project's code.

Connect your board to your computer using a Micro USB cable and choose how you'd like to program it. The project code is available for MakeCode or Circuit Python.

The code for this project was written in both MakeCode and CircuitPython languages - so you can choose which one you'd like to use:

Morse Code Flasher for MakeCode ()

or

Morse Code Flasher for Circuit Python ()

MakeCode

To program your Circuit Playground Express board with MakeCode, attach your computer via a micro USB cable, you should see it appear as a an attached drive named "CPLAYBOOT".

In the MakeCode window below, click the pink download button in the lower right corner and move the downloaded .uf2 file to the "CPLAYBOOT" drive.

Customize the Message

By default, the board will display Morse code for "SOS", but you can easily customize it with any capital letters or numbers. To do so, Modify the first block in within the "on start" block:

Simply replace "SOS" with your message - just be sure to use only capital letters:

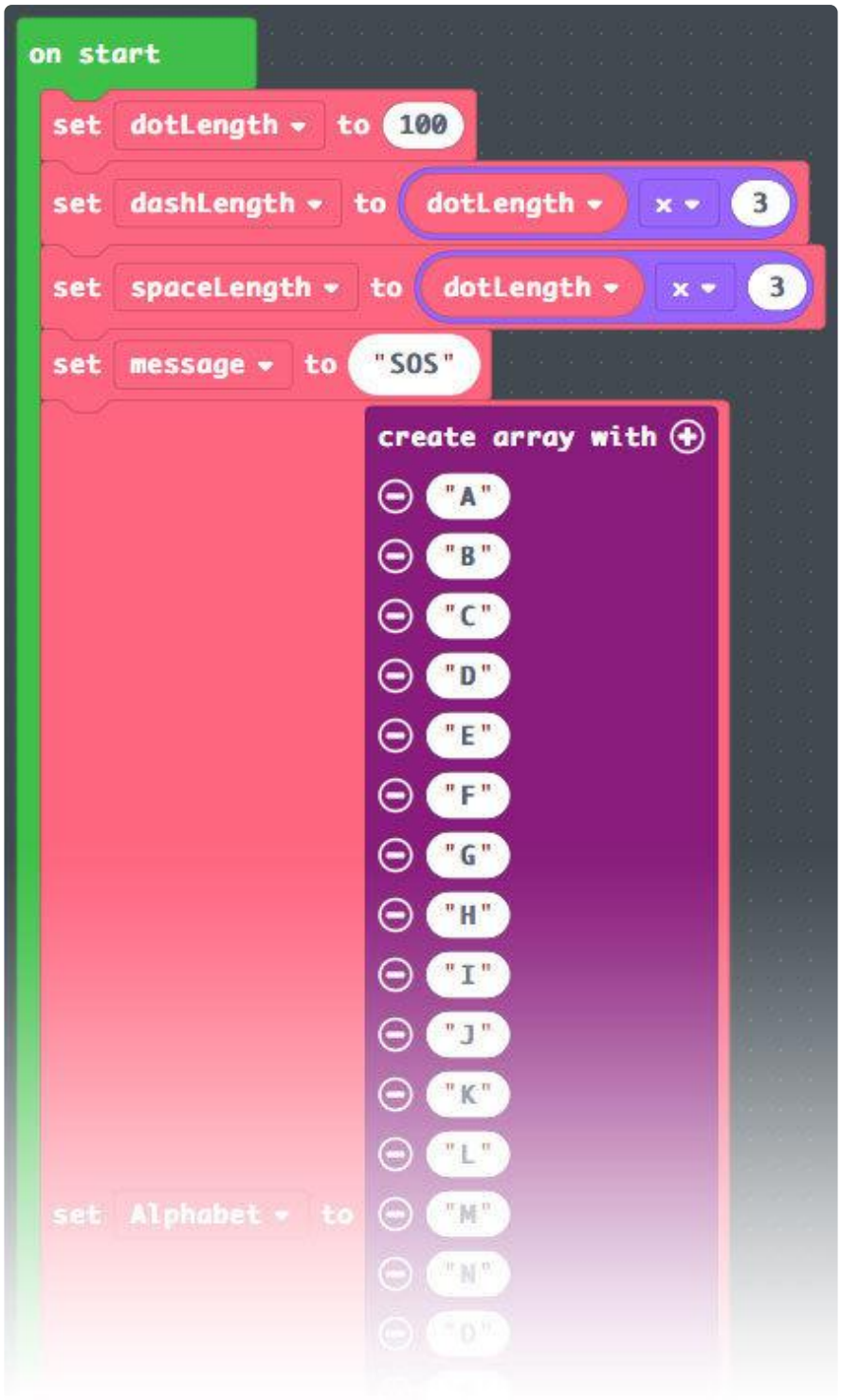
A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

A Tour of the Code

Let's take a look at all of the MakeCode blocks and learn a bit about how they work ...

On Start

This is where we define the variables we'll be using to store values we need to access from our code.



`message` is the message we'll be displaying in Morse code.

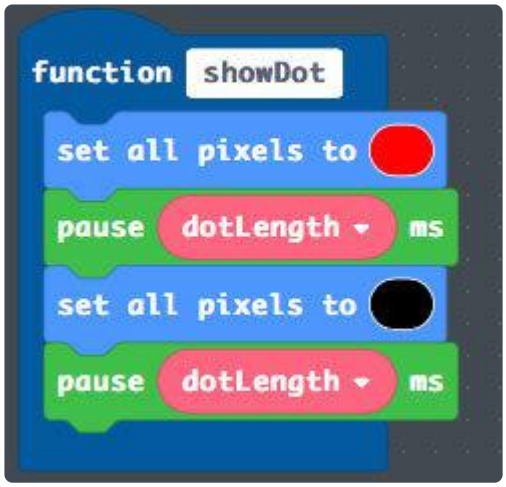
`dotLength` is the length of time (in seconds) one morse dot will take to display. `dotLength` is also used to determine the durations of the other time variables (`dashLength` & `spaceLength`) So, if you change `dotLength`, the other time variables will change in relation to it.

`Alphabet` is an array that contains the entire alphabet as capital letters.

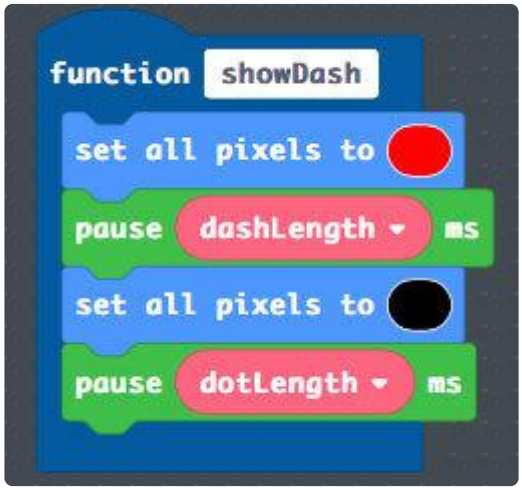
`Morse` is an array that contains the morse codes used to represent the entire alphabet.

Functions

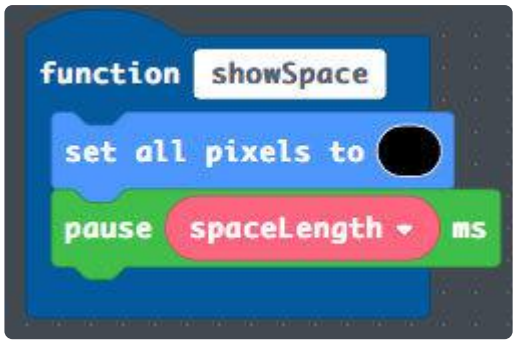
The functions do all the real work in our project. Let's look at each one and learn about what it does ...



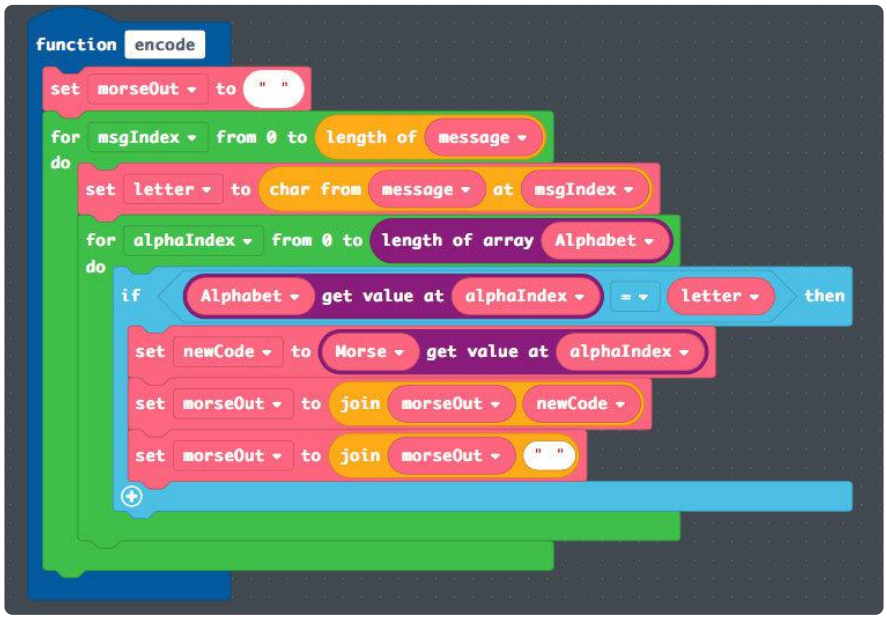
The `showDot` function turns the LEDs red, waits for one `dotLength`, then turns them off and waits for one `dotLength`.



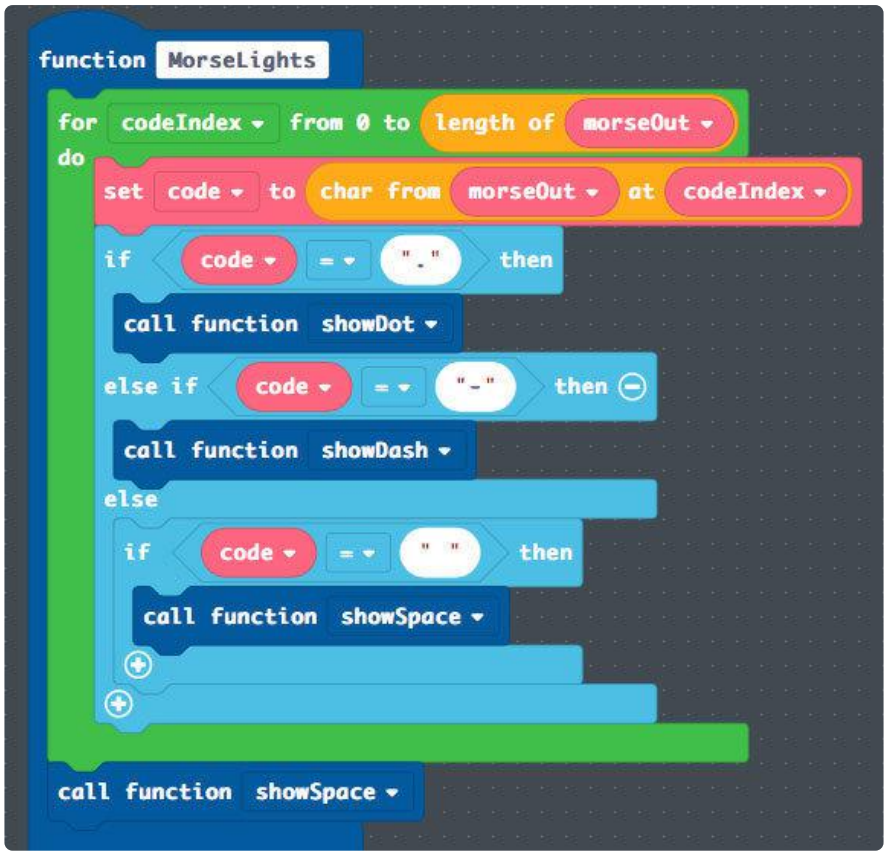
The `showDash` function turns the LEDs red, waits for one `dashLength`, then turns the LEDs off and waits for one `dotLength`.



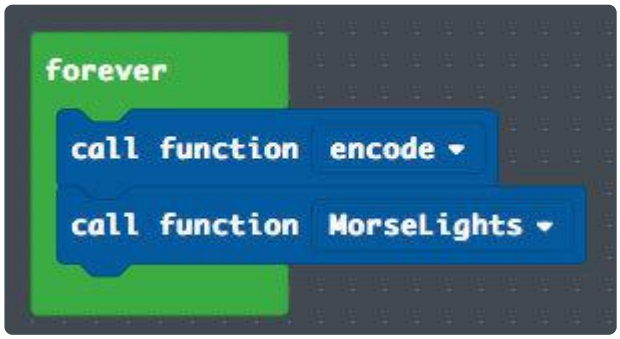
The `showSpace` function turns the LEDs off, then waits for one `spaceLength`.



The `encode` function translates the `message` variable into morse code using the `Alphabet` and `Morse` arrays. It takes each character from the `message`, finds that character's morse code equivalent, and adds that code to a variable named `morseOut`.



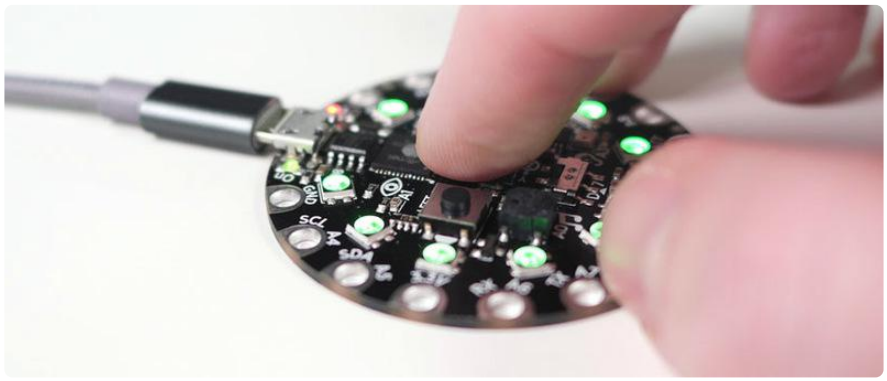
The `MorseLights` function takes all the dots and dashes stored in the `morseOut` variable and converts them into LED flashes. One by one, it checks each character in `morseOut`. If it finds a dot, it will call the `showDot` function. If it finds a dash, it will call the `showDash` function. And if it finds a space, it will call the `showSpace` function.



Finally, the `forever` block runs two functions over and over in a loop. First it calls the `encode` function to encode our message, then it calls `MorseLights` to display our message in Morse code with the Circuit Playground's LEDs.

CircuitPython

To program your Circuit Playground Express board with CircuitPython code, first make sure your board is set up to use CircuitPython - check out the Quick Start guide () for instructions.



Once your board is set up and attached to your computer via a micro USB cable, you should see it appear as an attached drive named "CIRCUITPY". (You may need to double-click the Reset button to make it appear.)

Copy the code below and save it as a file named "code.py" on the "CIRCUITPY" drive.

```
# SPDX-FileCopyrightText: 2017 Collin Cunningham for Adafruit Industries
#
# SPDX-License-Identifier: MIT

# Circuit Playground Express CircuitPython Morse Code Flasher
# This is meant to work with the Circuit Playground Express board:
#   https://www.adafruit.com/product/3333
# Needs the NeoPixel module installed:
#   https://github.com/adafruit/Adafruit_CircuitPython_NeoPixel
# Author: Collin Cunningham
# License: MIT License (https://opensource.org/licenses/MIT)

import time

import board
import neopixel

# Configuration:
# Message to display (capital letters and numbers only)
message = 'SOS'
dot_length = 0.15  # Duration of one Morse dot
dash_length = (dot_length * 3.0)  # Duration of one Morse dash
symbol_gap = dot_length  # Duration of gap between dot or dash
character_gap = (dot_length * 3.0)  # Duration of gap between characters
flash_color = (255, 0, 0)  # Color of the morse display.
brightness = 0.5  # Display brightness (0.0 - 1.0)
morse = [
    ('A', '.-'),
    ('B', '-...'),
    ('C', '-.-.'),
    ('D', '-..'),
    ('E', '.'),
    ('F', '..-.'),
```

```
    ('G', '--.'),
    ('H', '....'),
    ('I', '..'),
    ('J', '.---'),
    ('K', '-.-'),
    ('L', '.-..'),
    ('M', '--'),
    ('N', '-.'),
    ('O', '---'),
    ('P', '.--.'),
    ('Q', '--.-'),
    ('R', '.-.'),
    ('S', '...'),
    ('T', '-'),
    ('U', '..-'),
    ('V', '...-'),
    ('W', '.--'),
    ('X', '-..-'),
    ('Y', '-.--'),
    ('Z', '--..'),
    ('0', '-----'),
    ('1', '.----'),
    ('2', '..---'),
    ('3', '...--'),
    ('4', '....-'),
    ('5', '.....'),
    ('6', '-....'),
    ('7', '--...'),
    ('8', '---..'),
    ('9', '----.'),
]


# Define a class that represents the morse flasher.


class MorseFlasher:
    def __init__(self, color=(255, 255, 255)):
        # set the color adjusted for brightness
        self._color = (
            int(color[0] * brightness),
            int(color[1] * brightness),
            int(color[2] * brightness)
        )

    def light(self, on=True):
        if on:
            pixels.fill(self._color)
        else:
            pixels.fill((0, 0, 0))
        pixels.show()

    def showDot(self):
        self.light(True)
        time.sleep(dot_length)
        self.light(False)
        time.sleep(symbol_gap)

    def showDash(self):
        self.light(True)
        time.sleep(dash_length)
        self.light(False)
        time.sleep(symbol_gap)

    def encode(self, string):
        output = ""
        # iterate through string's characters
        for c in string:
            # find morse code for a character
```

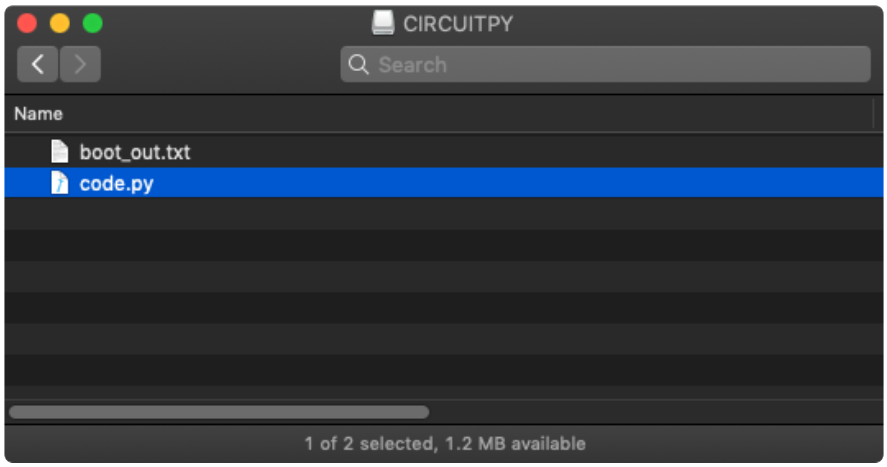
```
            for x in morse:
                if x[0] == c:
                    # add code to output
                    output += x[1]
            # add a space in between characters
            output += " "
        # save complete morse code output to display
        self.display(output)

    def display(self, code=".-.-.- "):
        # iterate through morse code symbols
        for c in code:
            # show a dot
            if c == ".":
                self.showDot()
            # show a dash
            elif c == "-":
                self.showDash()
            # show a gap
            elif c == " ":
                time.sleep(character_gap)


# Initialize NeoPixels
pixels = neopixel.NeoPixel(board.NEOPIXEL, 10, auto_write=False)
pixels.fill((0, 0, 0))
pixels.show()

# Create a morse flasher object.
flasher = MorseFlasher(flash_color)

# Main loop will run forever
while True:
    flasher.encode(message)
```



Once the new code.py file is saved, the board should restart and start displaying the SOS message in Morse code.

You should see flashes that correspond to the following sequence:

. . . - - - . . .

A Tour of the Code

So, what is all this CircuitPython code doing? Let's take a look piece by piece ...

Imports

First off, we import code libraries that allow us to do complex things with simple commands ...

```
import time
import board
import neopixel
```

Variables

Next, we define the variables we'll be using to store important values ...

```
message = 'SOS'                     # Message to display (capital letters and
numbers only)
dot_length = 0.15                   # Duration of one Morse dot
dash_length = (dot_length * 3.0)    # Duration of one Morse dash
symbol_gap = dot_length             # Duration of gap between dot or dash
character_gap = (dot_length * 3.0)  # Duration of gap between characters
flash_color = (255, 0, 0)           # Color of the morse display.
brightness = 0.5                    # Display brightness (0.0 - 1.0)
morse = [('A', '.-'), ('B', '-...'), ('C', '-.-.'), ('D', '-..'), ('E', '.'), ('F',
'..-.'), ('G', '--.'), ('H', '....'), ('I', '..'), ('J', '.---'), ('K', '-.-'),
('L', '.-..'), ('M', '--'), ('N', '-.'), ('O', '---'), ('P', '.--.'), ('Q',
'--.-'), ('R', '.-.'), ('S', '...'), ('T', '-'), ('U', '..-'), ('V', '...-'), ('W',
'.--'), ('X', '-..-'), ('Y', '-.--'), ('Z', '--..'), ('0', '-----'), ('1',
'.----'), ('2', '..---'), ('3', '...--'), ('4', '....-'), ('5', '.....'), ('6',
'-....'), ('7', '--...'), ('8', '---..'), ('9', '----.')]
```

`message` is the string of letters or numbers we want to display in Morse code.

`dot_length` is the length of time (in seconds) one morse dot will take to display. `dot_length` is also used to determine the value of all the other time variables (`dash_length`, `symbol_gap`, & `character_gap`) So, if you change `dot_length`, all the other time variables will change in relation to it.

`flash_color` is the color the LEDs will show when they flash.

`brightness` determines how bright the flashes will be.

Finally, the `morse` variable is an array of value pairs called “tuples”. Each tuple contains two values - the first one is the readable character we want to display in Morse code and the second is the sequence of Morse dots and dashes we’ll use to display that readable character.

MorseFlasher Object

Next up, we define the MorseFlasher object which will translate our readable characters to Morse code and handle all the LED flashing. It looks pretty big, but it's not too complicated once you break it down ...

```
class MorseFlasher:

    def __init__(self, color=(255,255,255)):
        #set the color adjusted for brightness
        self._color = (int(color[0]*brightness), int(color[1]*brightness),
int(color[2]*brightness))

    def light(self, on=True):
        if on:
            pixels.fill(self._color)
        else:
            pixels.fill((0,0,0))
        pixels.show()

    def showDot(self):
        self.light(True)
        time.sleep(dot_length)
        self.light(False)
        time.sleep(symbol_gap)

    def showDash(self):
        self.light(True)
        time.sleep(dash_length)
        self.light(False)
        time.sleep(symbol_gap)

    def encode(self, str):
        output = ""
        #iterate through string's characters
        for c in str:
            #find morse code for a character
            for x in morse:
                if x[0] == c:
                    #add code to output
                    output += x[1]
            # add a space in between characters
            output += " "
        #save complete morse code output to display
        self.display(output)

    def display(self, code=".-.-.- "):
        # iterate through morse code symbols
        for c in code:
            # show a dot
            if c == ".":
```

```
            self.showDot()
        # show a dash
        elif c == "-":
            self.showDash()
        # show a gap
        elif c == " ":
            time.sleep(character_gap)
```

Within the `MorseFlasher` object, there are several functions that define how it works:

The `__init__` function creates the MorseFlasher object and sets the color that it will show by multiplying an initial color by the `brightness` variable.

The `light` function turns the LEDs on or off depending on a given boolean (True or False) value.

The `showDot` function turns the LEDs on, waits for one `dot_length`, then turns them off and waits for one `gap_length`.

The `showDash` function turns the LEDs on, waits for one `dash_length`, then turns them off and waits for one `gap_length`.

The `encode` function takes a message and translates it into morse code using the `morse` array variable. It takes each character from the message, finds that character's morse code from the array of tuples, and adds that code to a variable named `output`. Once all the characters in the message have been translated, the `display` function is called and given the `output` variable.

Finally, the `display` function takes a string of morse code (dots, dashes, and spaces) and converts into LED flashes. One by one, it checks each character in the code. If it finds a dot, it will call the `show_dot` function. If it finds a dash, it will call the `show_dash` function. And if it finds a space, it will wait for the length of one `character_gap`.

Let's do this!

Once all of our functions have been defined, it's time to get the party started and use all that code we've written above ...

```
# Initialize NeoPixels
pixels = neopixel.NeoPixel(board.NEOPIXEL, 10, auto_write=False)
pixels.fill((0, 0, 0))
pixels.show()

# Create a morse flasher object.
flasher = MorseFlasher(flash_color)
```

```
# Main loop will run forever
while True:
    flasher.encode(message)
```

In order to easily control our LEDs, we create `pixels` which is a `Neopixel` object from the `neopixel` library we imported earlier. We want to make sure the LEDs are all turned off at the start, so we set them to black `(0, 0, 0)` and call `pixels.show()` to show the colors we just set.

Next we create the `MorseFlasher` object named `flasher` and we give it the color we want the LEDs to flash.

Finally - we set all the gears in motion. Inside the main loop (`while True:`) we call the flasher object's `encode` function and give it our `message` to display. Once the function has completed and shown the `message`, the main loop will begin again and our `message` will be displayed - FOREVER! ... or at least until you disconnect power or reprogram the Circuit Playground board ;)

Change it!

This project is a great starting point for learning to code for the Circuit Playground Express. Consider modifying it to make it your own.

Here's some ideas for changes you can make and some tips on how to make them in CircuitPython:

Change the message

By default, the code will display Morse code for "SOS", but you can easily customize it with any capital letters or numbers. To do so, Modify this line:

`message = 'SOS'`

Simply replace SOS with your message - just be sure to use only capital letters or numbers:

`A B C D E F G H I J K L M N O P Q R S T U V W X Y Z 1 2 3 4 5 6 7 8 9 0`

(Note that the MakeCode version of the code will only display letters, not numbers)

Customize the color of the LEDs

The color and brightness of the LEDs can be changed by modifying these two lines in the CircuitPython code:

```
flash_color = (255, 0, 0) # Color of the morse display.
```

```
brightness = 0.5 # Display brightness (0.0 - 1.0)
```

The flash_color numbers represent the amount of (Red, Green, Blue) mixed together to create the final displayed color. They can be any whole number between 0 and 25 5.

The brightness value changes how much of the flash_color is displayed. It can be set to a decimal value between 0.0 and 1.0

Adjust the speed of the LED flashes

The duration of LED flashes and the gap between them can all be changed with this line:

```
dot_length = 0.15 # Duration of one Morse dot
```

The dot_length value represents time in seconds, so changing 0.15 to 1.0 will make each 'dot' flash last one whole second - which is pretty long for morse code!

More ideas

Once you feel comfortable making changes to the code, try making some more advanced modifications like these:

- Set the display to only trigger after pressing one of the built-in buttons
- Set a different message to display for each button pressed
- Add an audio tone which plays in sync with the LEDs