CptS 111 Introduction to Algorithmic Problem Solving (http://piazza.com/wsu/fall2016/cpts111/home)

Washington State University (https://wsu.edu)

Gina Sprint (http://eecs.wsu.edu/~gsprint/)

Lab12 Lists (10 pts)

Learner Objectives

At the conclusion of this lab, participants should be able to:

- Declare lists uni-dimensional and multi-dimensional lists
- Index into lists
- Apply common list operations
- Implement basic list functionality
- Program hardware via a Python interface

Prerequisites

Before starting this lab, participants should be able to:

- Apply loops
- Perform File I/O
- Work with strings

Lab Format

Task 0 and Pair Task

Task 0 will be completed as a group with step-by-step guidance from your TA. After Task 0, your TA will group you into pairs and assign you a task (one of Tasks 1-10).

Additional Tasks

Once you complete Task 0 and your assigned task with a partner (pair task), you are to continue solving additional lab tasks until the end of lab. You are encouraged to complete *all* of the lab tasks. Practice, practice, practice!

Completing a Task

When you complete a task, show your TA your code and output. Also, help others with this task either in person or by answering questions in your Piazza (http://piazza.com/wsu/fall2016/cpts111/home) lab section group. Helping your peers benefits both you and your classmates, "While we teach, we learn." Seneca (Roman philosopher, mid-1st century AD).

Lab Dismissal

You are dismissed from lab when your TA dismisses you, or after 3 hours, whichever comes first. Your TA will dismiss you once you have demonstrated mastery of the material (completed a certain number of tasks). Should you demonstrate mastery before the end of lab, you are encouraged to stay and perform a combination of the following:

- Help your peers
- Work on additional tasks
- Work on the current programming assignment

Take advantage of your TA and your peers as resources and make the most of lab time!

Acknowledgments

Content used in this assignment is based upon information in the following sources:

- Adafruit LED Art with Fadecandy (https://learn.adafruit.com/led-art-with-fadecandy)
- scanlime's Github Fadecandy repository (https://github.com/scanlime/fadecandy)

Task 0

For this lab, we are going to solve of mix of standard tasks, as well as tasks that utilize hardware. The hardware we are going to play with is a combination of an Adafruit (https://www.adafruit.com/) Fadecandy (https://learn.adafruit.com/assets/12274) microcontroller and a 8x8 LED NeoPixel matrix (https://www.adafruit.com/products/1487). The Fadecandy/LED matrix hardware/software is a DIY (do it yourself) project that is available through Adafruit at this site (https://learn.adafruit.com/led-art-with-fadecandy).

Notes

1. If for some reason you are unable to get the necessary software installed and working properly, *there are tasks in this lab that you can solve without the hardware.* Tasks that require the hardware are labeled.
2. Please **be gentle** with the hardware, especially the wires and the connection points
3. Please **do not turn the lights on full brightness**. It is incredibly bright! To programmatically dim the lights, use less intense RGB values for your colors. For example, let `brightness = 50`:
    A. `white = (brightness, brightness, brightness)` instead of `white = (255, 255, 255)`
    B. `red = (brightness, 0, 0)` instead of `red = (255, 0, 0)`
    C. `green = (0, brightness, 0)` instead of `green = (0, 255, 0)`
    D. `blue = (0, 0, brightness)` instead of `blue = (0, 0, 255)`

Note: You can explore RGB color values at this website (http://www.colorpicker.com/).

The following steps for Task 0 are going to walk you through downloading and installing the necessary software to program the Fadecandy micro-controller with Python :) I would like to give credit to this Adafruit project page (https://learn.adafruit.com/led-art-with-fadecandy) for information about the hardware/software setup.

Setup Fadecandy Hardware

We are going to utilize the following hardware:

1. Fadecandy micro-controller

2. 8x8 LED NeoPixel matrix

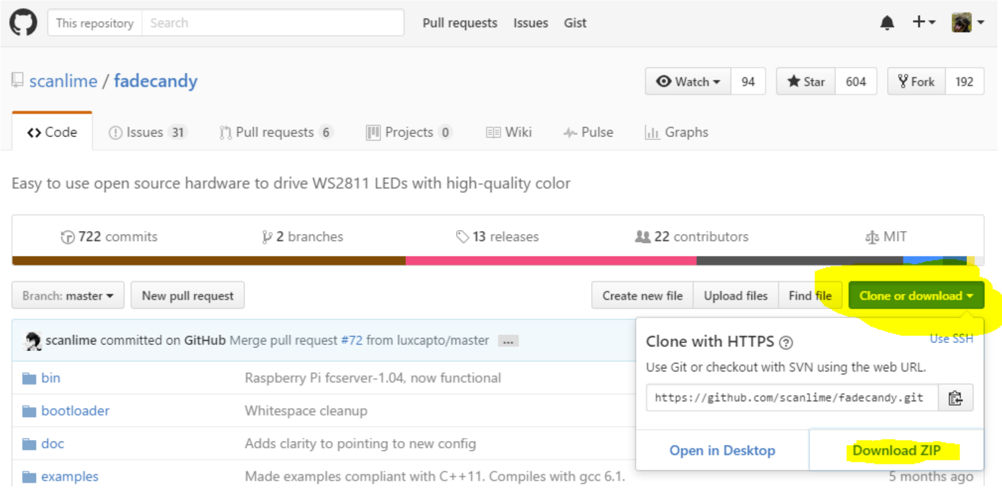
3. Mini USB cable
4. 5V power supply

To setup the hardware, perform the following steps:

1. Gently plug the USB cable into the Fadecandy board and into your computer
2. Gently plug the 5V power supply into an outlet

Setup Fadecandy Software

1. Open this link: https://github.com/scanlime/fadecandy (https://github.com/scanlime/fadecandy) and click the green "Clone or download" button. From the drop down menu, click "Download ZIP". Download this zip file (fadecandy-master.zip) to your computer.



Easy to use open source hardware to drive WS2811 LEDs with high-quality color

scanlime committed on GitHub Merge pull request #72 from luxcapto/master

bin	Raspberry Pi fcserver-1.04, now functional
bootloader	Whitespace cleanup
doc	Adds clarity to pointing to new config
examples	Made examples compliant with C++11. Compiles with gcc 6.1.

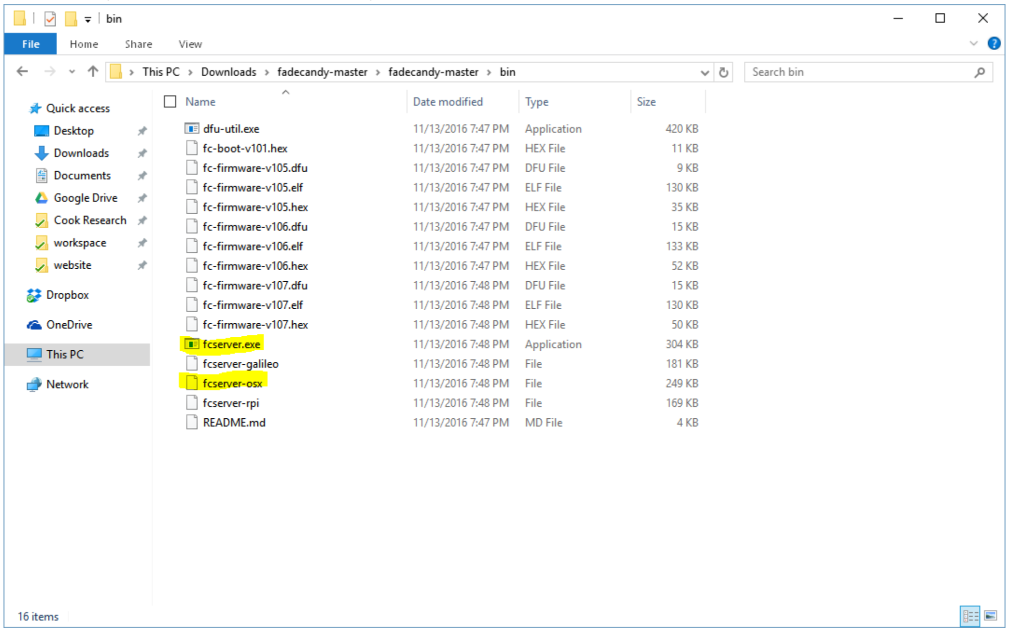
5 months ago

Clone with HTTPS

Use Git or checkout with SVN using the web URL.

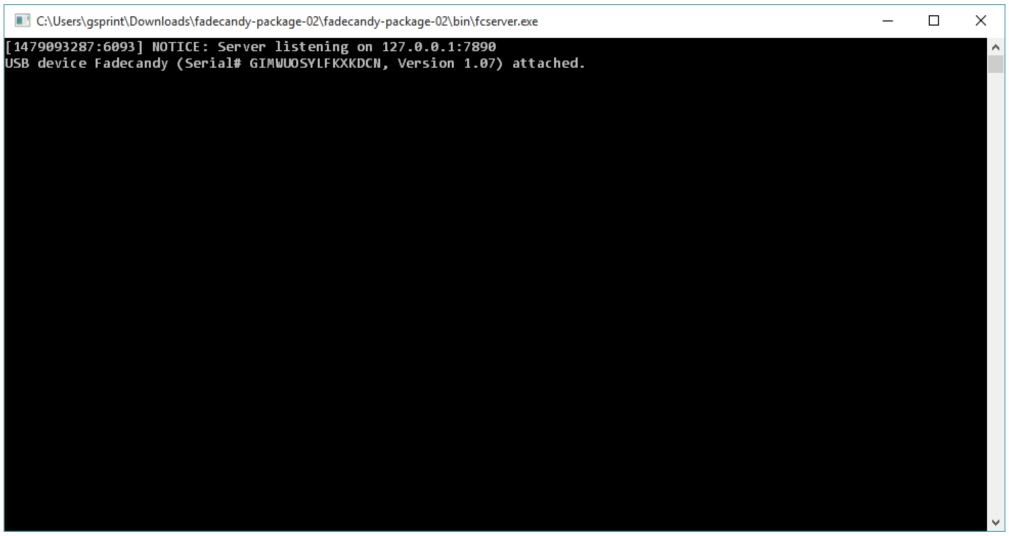
`https://github.com/scanlime/fadecandy.git`

2. Extract the downloaded zip file fadecandy-master.zip
3. Open the extracted folder fadecandy-master
4. Open the fadecandy-master\bin directory



Name	Date modified	Type	Size
dfu-util.exe	11/13/2016 7:47 PM	Application	420 KB
fc-boot-v101.hex	11/13/2016 7:47 PM	HEX File	11 KB
fc-firmware-v105.dfu	11/13/2016 7:47 PM	DFU File	9 KB
fc-firmware-v105.elf	11/13/2016 7:47 PM	ELF File	130 KB
fc-firmware-v105.hex	11/13/2016 7:47 PM	HEX File	35 KB
fc-firmware-v106.dfu	11/13/2016 7:47 PM	DFU File	15 KB
fc-firmware-v106.elf	11/13/2016 7:47 PM	ELF File	133 KB
fc-firmware-v106.hex	11/13/2016 7:47 PM	HEX File	52 KB
fc-firmware-v107.dfu	11/13/2016 7:48 PM	DFU File	15 KB
fc-firmware-v107.elf	11/13/2016 7:48 PM	ELF File	130 KB
fc-firmware-v107.hex	11/13/2016 7:48 PM	HEX File	50 KB
fcserver.exe	11/13/2016 7:48 PM	Application	304 KB
fcserver-galileo	11/13/2016 7:48 PM	File	181 KB
fcserver-osx	11/13/2016 7:48 PM	File	249 KB
fcserver-rpi	11/13/2016 7:48 PM	File	169 KB
README.md	11/13/2016 7:47 PM	MD File	4 KB

5. Run the fcserver program. This program runs in the background and connects your Python code with the Fadecandy board. It also has a simple web interface you can use to test your LEDs.
    - A. On Mac OS, double-click the fcserver-osx file in bin
    - B. On Windows, double-click fcserver.exe in bin
6. Check that fcserver is running. The fcserver program will run in a terminal window (console). If all is well, you should get a message like "Server listening on 127.0.0.1:7890". This tells you it's ready to accept connections from other programs on your computer.

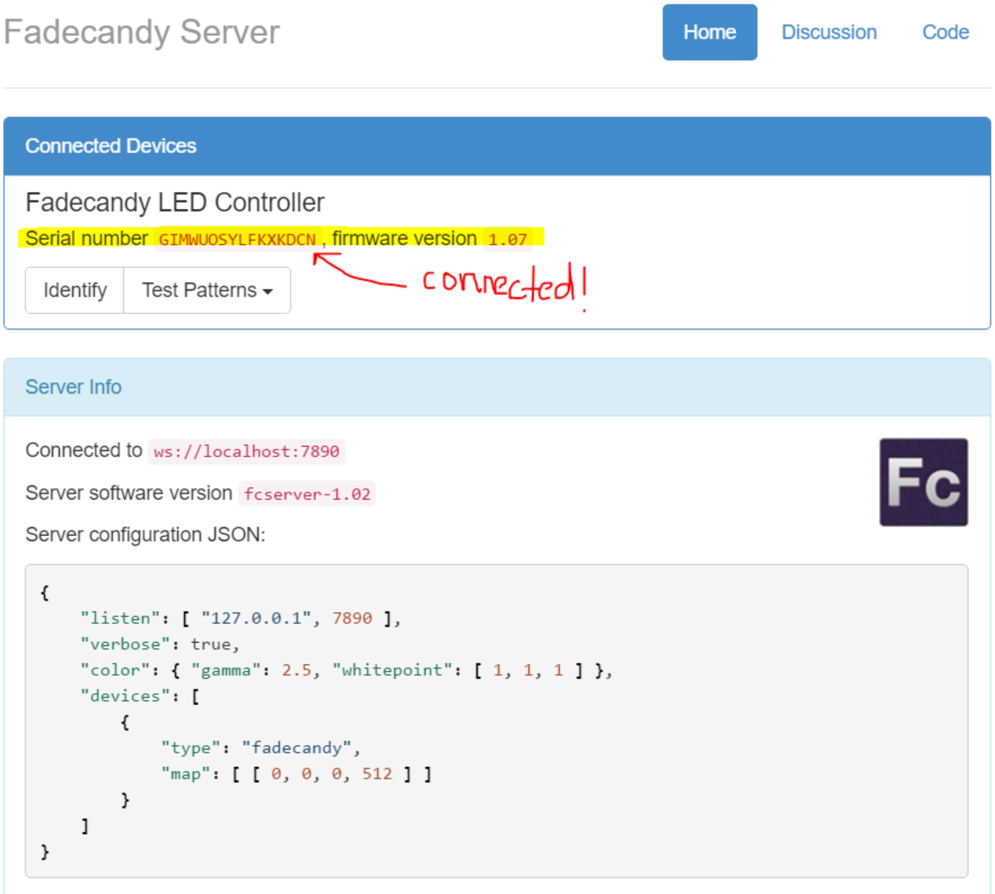

C:\Users\gsprint\Downloads\fadecandy-package-02\fadecandy-package-02\bin\fcserver.exe

```
[1479093287:6093] NOTICE: Server listening on 127.0.0.1:7890
USB device Fadecandy (Serial# GIMWUOSYLFKXKDCN, Version 1.07) attached.
```

What is fcserver listening for? Three things, actually:

   A. Programs using the Open Pixel Control (opc) protocol to talk to your LEDs
      a. This is how we are going to program the Fadecandy/matrix using Python
   B. Browser-based programs using WebSockets to talk to your LEDs
   C. Web browsers requesting an HTML-based user interface

7. Let's try out the browser interface. Now that fcserver is running, you can navigate your web browser to http://localhost:7890 (http://localhost:7890). You should see a page like this one:



Fadecandy Server

Connected Devices

Fadecandy LED Controller

Serial number `GIMWUOSYLFKXKDCN`, firmware version `1.07`

connected!

Server Info

Connected to `ws://localhost:7890`

Server software version `fcserver-1.02`

Server configuration JSON:

```
{
    "listen": [ "127.0.0.1", 7890 ],
    "verbose": true,
    "color": { "gamma": 2.5, "whitepoint": [ 1, 1, 1 ] },
    "devices": [
        {
            "type": "fadecandy",
            "map": [ [ 0, 0, 0, 512 ] ]
        }
    ]
}
```

   A. If you plug in a Fadecandy Controller board, it should show up in the Connected Devices section. No need to reload the web page.
   B. From the "Test Patterns" drop down button, select 50% brightness to make sure you are all set up to connect to your Fadecandy and matrix! If this doesn't work, ask your TA for help.

Run a Fadecandy Python Example

We are (finally) going to program the Fadecandy/matrix from Python! To do this, follow these steps:

1. Navigate up one directory from the fadecandy-master\bin folder to the fadecandy-master folder. Open the fadecandy-master\examples\python folder. Here you will see several .py file examples
2. Open the README.md file using a text editor. README.md contains a description of each .py file in the folder. Read about chase.py. **When executed, what is chase.py supposed to do?**
3. Open chase.py in Spyder
4. Run chase.py and watch your matrix! **Does the execution of chase.py do what you expected it to do?**

Now, let's walk through chase.py line by line (I've added line numbers):

```
Line 1:#!/usr/bin/env python

Line 3:# Light each LED in sequence, and repeat.

Line 5:import opc, time

Line 7:numLEDs = 512
Line 8:client = opc.Client('localhost:7890')

Line 10:while True:
Line 11:    for i in range(numLEDs):
Line 12:        pixels = [ (0,0,0) ] * numLEDs
Line 13:        pixels[i] = (255, 255, 255)
Line 14:        client.put_pixels(pixels)
Line 15:        time.sleep(0.01)
```

Line 5: `opc` (you can read about what opc does in README.md) and `time` are imported. The `opc` module is imported in order to create a `Client` object used to interface with the hardware via the fcserver (see line 8). The `time` module is imported to add a time delay, essentially forcing the program to "wait" (see line 15).

Line 7: a variable `numLEDS` is initialized to 512. Some Fadecandy projects connect to 8x64 LED NeoPixel matrices. Since we are connected to a 8x8 matrix, what should we change `numLEDS` to? Make this change and re-run chase.py. **How did this change affect the animation?**

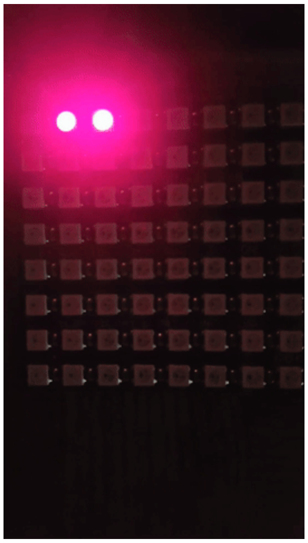
Line 8: an `opc.Client` object is initialized. If you are curious about the string argument, `'localhost:7890'`, read about it in the docstring of `__init__()` in opc.py.

Line 10: we have an infinite loop. Essentially, run until this program is killed.

Line 11: we have a loop iterating `numLEDS` number of times. For each iteration, `i`, of the loop (lines 11-15), the code does the following:

1. Line 12: Create a list of `numLEDS` RGB (R, G, B) tuples. *A tuple is an immutable list*. Each tuple is initialized to (0, 0, 0) which is black (no light). A list of pixel colors is created for each light.
2. Line 13: Modify the RGB tuple to (255, 255, 255), which is white. Try changing the tuple to (151, 30, 49) **How did this change affect the animation?**
    A. Do you know what color (151, 30, 49) is? Hint: GO COUGS!

3. Line 14: Send the pixel colors to the hardware. `put_pixels()` is a *method* of the `Client` object. Recall, **what is a method?**
4. Line 15: Suspend the program a certain number of seconds. `time.sleep(<number of seconds>)` accepts an argument, the number of seconds to "wait". You can read more about `time.sleep()` in the Python docs (https://docs.python.org/3/library/time.html#time.sleep). Change the 0.01 to 1 and re-run chase.py. **How did this change affect the animation?**

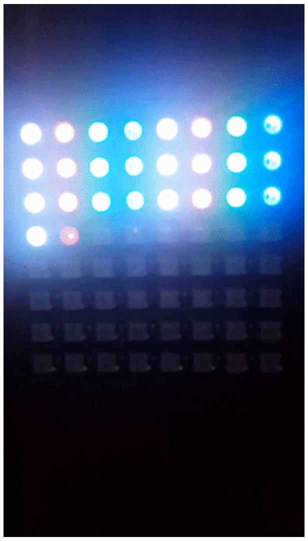


You are now ready to try Fadecandy/matrix programming on your own!

Note: Make sure that opc.py is in the *same folder* as your .py file that contains code (such as chase.py) to program the Fadecandy/matrix. Enjoy!

Task 1

Write a Fadecandy/matrix program that turns each light on one by one starting with the upper left corner and working toward the bottom right corner until all the lights are on. Once all lights are on, turn them off one at time starting with the bottom right corner. Hence, the first light to turn on should be the last light to turn off and the last light to turn on should be the first light to turn off.

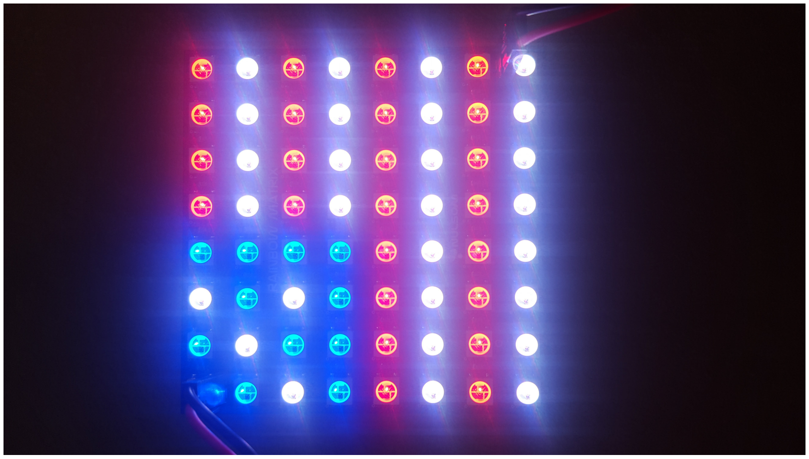


Task 2

Write a Fadecandy/matrix program that shows a graphic of your choosing (e.g., a flag, an emoji, a fun pattern, etc.) for 2 seconds, then turns the lights off for 5 seconds. Repeat the graphic on/off sequence 5 times.

Note: make sure your graphic is school appropriate.

Here is an example using an American Flag:



Task 3

Since the NeoPixel matrix is an 8x8 matrix, logically it makes sense to think of the matrix as a two dimensional list of pixels, i.e. a list of 8 items where each item is a list of 8 pixels. So far, we have been programming the Fadecandy/matrix such that it is a one dimensional list of pixels, i.e. a list of 64 items where each item is a pixel.

Part 1

So we can use 2D lists with the Fadecandy/matrix in the future, write a program that converts 2D lists of 8x8 items to 1D lists of 64 items and vice versa.

As part of your solution, define the following functions:

1. `convert_twoD_to_oneD(twoD)`: accepts a single parameter, `twoD`, that is a 2-dimensional list (8x8) of items. `convert_twoD_to_oneD()` returns a 1-dimensional list of the items in `twoD`, flattened out. As a small 2x2 example, `convert_twoD_to_oneD([[1, 2], [3, 4]])` would return `[1, 2, 3, 4]`
2. `convert_oneD_to_twoD(oneD)`: accepts a single parameter, `oneD`, that is a 1-dimensional list of 64 items. `convert_oneD_to_twoD()` returns a 2-dimensional list (8x8) of the items in `oneD`, structured in a grid. As a small 2x2 example, `convert_oneD_to_twoD([1, 2, 3, 4])` would return `[[1, 2], [3, 4]]`.

Note: The items may be anything (pixel tuples, strings, integers, etc.), the type of the items does not matter.

Now, write code that populates a 2-dimensional list with random RGB values in the range [0, 255] inclusive. Call your `convert_twoD_to_oneD(twoD)` to convert the 2D list to a 1D list and send the pixels to the board. Generate new random pixel colors every 0.25 seconds.

Here is an example matrix of random colors:

Part 2

Modify your solution so that your functions convert 1D and 2D lists of any size, instead of 8x8 and 64. You will not need to add any additional parameters to `convert_twoD_to_oneD(twoD)` to do this, but you will need to add parameters to `convert_oneD_to_twoD(oneD)` in order to know how many rows to extract from `oneD` and how many items should be in each row. I recommend one of the following approaches:

1. (easy) Uniform row length: `convert_oneD_to_twoD(oneD, num_rows)` where `num_rows` is an integer number of rows to extract. Note: you can assume each row has the same number of items in it (i.e. a rectangular matrix).
2. (challenge) Non-uniform row length: `convert_oneD_to_twoD(oneD, num_rows, num_items_per_row)` where `num_rows` is an integer of rows to extract and `num_items_per_row` is a 1D list of `num_rows` integers, where an integer in `num_items_per_row` at position `i` denotes the number of items for the `i`th row. Note: you are assuming each row does not necessarily have the same number of items in it (i.e. a jagged matrix).

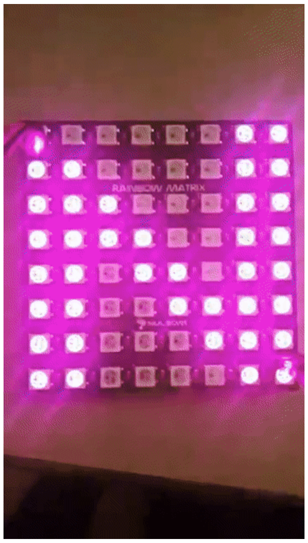
Task 4

Part 1

Write a Fadecandy/matrix program that continually shows the letters of your name, one letter at a time.

Note: You can design letters of your name using my frame designer (http://eecs.wsu.edu/~gsprint/cougarquest/lightarray/). For this tool, `B` is used to denote black, and `C` is used to denote color. You will want to declare variables `B = (0, 0, 0)` and `C = # your choice of color` in your code.

Here is an example showing "PYTHON":



Part 2

Modify how the letters of your name are displayed such that the letters *scroll* across the matrix from right to left, instead of being displayed one at a time.

Here is an example showing "PYTHON":