Braille Translator

Andressa Chan, Eliza Huang, Mino Nakura

University of Washington | CSE 455 Final Project | Autumn 2019

PAUL G. ALLEN SCHOOL
OF COMPUTER SCIENCE & ENGINEERING

MAKING SENSE OF BRAILLE ONE DOT AT A TIME.

Our motivation for this project stems from our interest in accessibility. In previous classes, capstones, and internships, we've always had to think about developing for a wide audience. Our Braille Translator is a way to raise awareness about the Braille language and gain more insight on the difficulties that visually-impaired individuals face. Throughout this process, we not only learned a lot about the Braille alphabet, but also about how to use openCV and its related functions to classify a given Braille character.

Design

Image	• Braille sentence to translate
Model.py	• Segments image into individual letters
FeatureExtraction.py	• Processes image to detect Braille circles • Encode circles into a string representing grid numbers
CodeConverter.py	• Decodes string to letter
TRANSLATION!	• Cumulate letters to get full sentence translation

DID YOU KNOW...

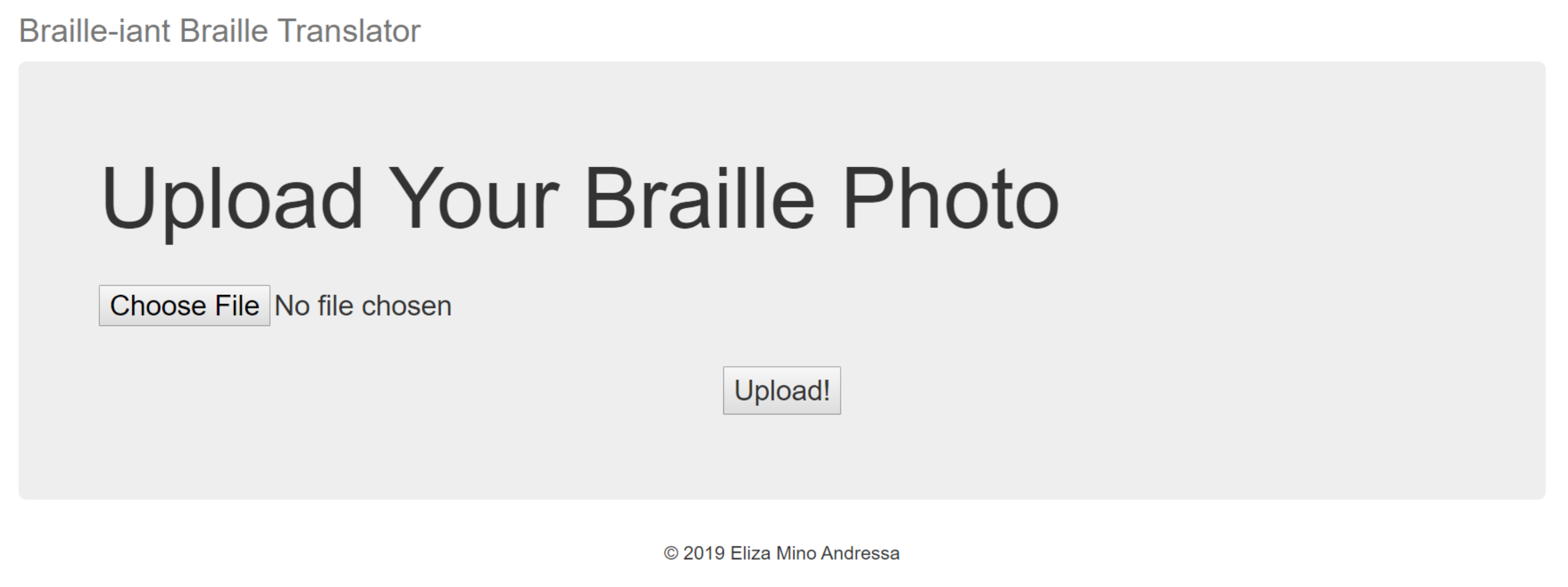
World Braille Day is on January 4th every year to celebrate Louis Braille's birthday, the inventor of Braille.

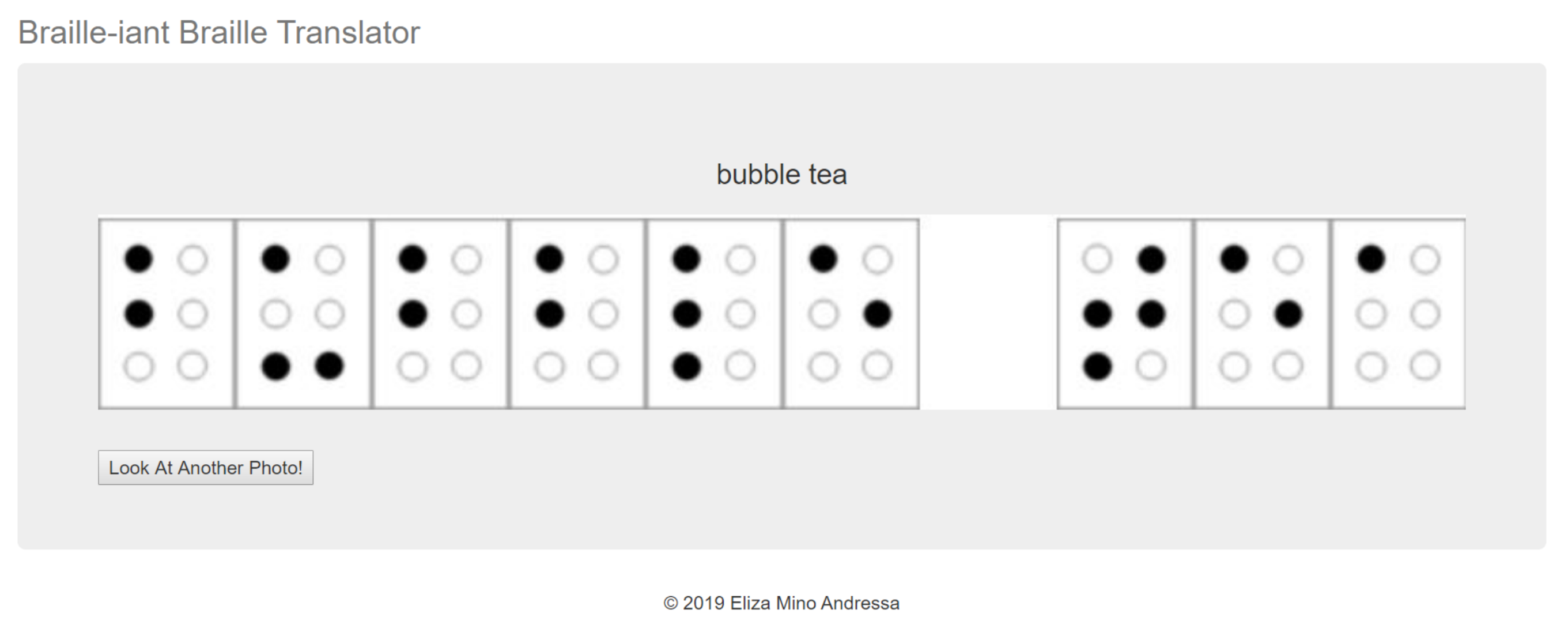
Braille symbols are formed within braille cells. A full braille cell consists of six raised dots that are numbered one through six. Cells can be used to represent a letter, number, punctuation, part of a word, or even a whole word.

With the help of corrective lenses, magnification, large print, and other accommodations 75% of the legally blind people are able to read print

How It Works

What The User Sees

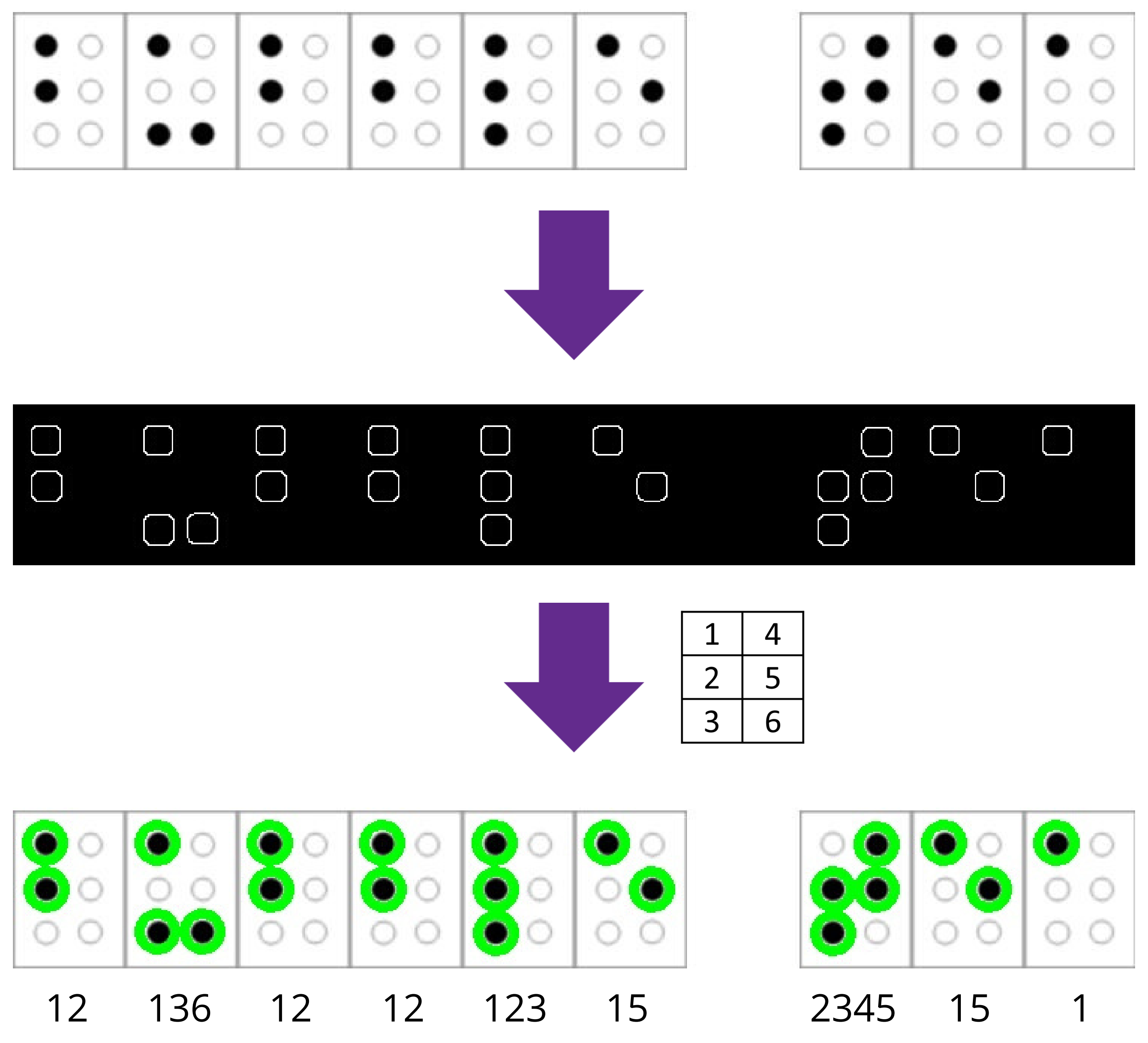

Braille-iant Braille Translator
Upload Your Braille Photo
Choose File No file chosen
Upload!
© 2019 Eliza Mino Andressa



How We Decode Braille

First, we figured out how wide and tall each Braille character is and broke up our images based on those measurements. We then passed in every image to our classifier function. In this function, we filtered image by dilating and eroding our image. By dilating and eroding, we try to emphasize the black circles and diminish unnecessary noise. Then, we run Canny edge detection to outline the necessary circles that make up the letter. Finally, we find the contours on the image, which are the black circles. Using the coordinates of the found contours, we map the coordinates to the corresponding grid and encode the braille circles as a string of grid numbers. Then, we simply decode the encoded string to the alphabet character. We repeatedly do this for every letter and cumulate a string which represents the Braille translation.

Behind The Scenes



Challenges

One persistent issue we faced was falsely detecting circles in our image, which led to improper translations. We tried many different filters to fix this issue, including resizing, sharpening, and grayscaling. In the end, however, we found that Canny edge detection, coupled with erosion and dilation, was the most effective filtering method to properly detect the circles. We also tried using the HoughCircles function to detect circles, but this proved to be difficult because there were many parameters we had to tune.

References

https://scotch.io/tutorials/getting-started-with-flask-a-python-microframework
http://www.cs.ubc.ca/projects/knoll/braille.html
https://gitlab.cs.washington.edu/nakuram/braille-iant
https://www.pharmabraille.com/pharmaceutical-braille/the-braille-alphabet/