CS51 - Assignment 8
Braille Text Analysis*

Due: November 13th, at 11:59pm



https://xkcd.com/315/

For this assignment, we will:

- Develop proficiency in handling text files by applying various techniques for file reading and writing.
- Refine string manipulation skills by implementing string operations to extract and modify information from files.
- Continue practicing with the turtle module.
- Master the creation and manipulation of lists, a fundamental data structure in Python, to store and organize information.
- Acquire proficiency in working with dictionaries, another essential data structure, to efficiently collect and retrieve information in a structured manner.
- Apply the aforementioned skills to collect and display information from files using Python programming.

*This assignment was created by Dr. Yasmine Elglaly and generously shared with the Pomona CS51 course staff.

1 Braille Background

- An estimated 1 billion people report having a disability. (source: World Health Organization)
- 85% of adults use the Internet (source: Pew Research Center)
- Many modern websites, mobile apps, and common Internet usage are inaccessible to people with disabilities, preventing a large portion of society from gaining knowledge of important services and information (source: WebAIM).
- In the US alone, 7 million people are estimated to have vision loss or blindness (source: CDC).

Braille is a tactile writing system used by people with visual impairments. It is traditionally written with a braille writer on embossed paper. Braille users typically use a refreshable braille display or terminal, an electromechanical device that displays braille characters by means of round-tipped pins raised through holes in a flat surface, which refreshes continuously.

Braille was the first writing system to use a binary encoding. The system, as devised in France by Louis Braille in the 1820s, consists of two parts:

- Mapping characters of the alphabet to arrays of six bits.
- Representing these six-bit characters with raised dots in a braille cell. Within an individual cell, the dot positions are arranged in a 3×2 grid (3 rows, 2 columns). A raised dot can appear in any of the six positions, leading to $2^6 = 64$ possible patterns, including one in which there are no raised dots.

A braille character is commonly described by listing the positions where dots are raised. The positions are universally numbered. The left column runs from top to bottom from 1 to 3, and the right column, from top to bottom, from 4 to 6. For example, the dot pattern 1-3-4 describes a cell with three dots raised at the top left column (1), bottom left column (3), and top right column (4). This corresponds to the letter 'm.'

Grade 1 braille, also known as uncontracted braille, is the most basic level and consists of a letter-for-letter translation of print, including the 26 letters of the alphabet, numbers, and punctuation. It is a phonetic system where each letter is represented by a unique braille cell. The following describes some patterns within grade 1 braille:

- The first ten letters of the alphabet, 'a'–'j,' use the upper four dot positions (black dots in Figure 1).
- The next ten letters, 'k'–t,' are similar to 'a–'j' with an addition of a dot at position 3 (red dots in Figure 1).
- The next ten letters are the same again, but with dots also at positions 3 and 6 (green dots in Figure 1). Here 'w' was left out as it was not a part of the official French alphabet during the time Louis Braille was alive.

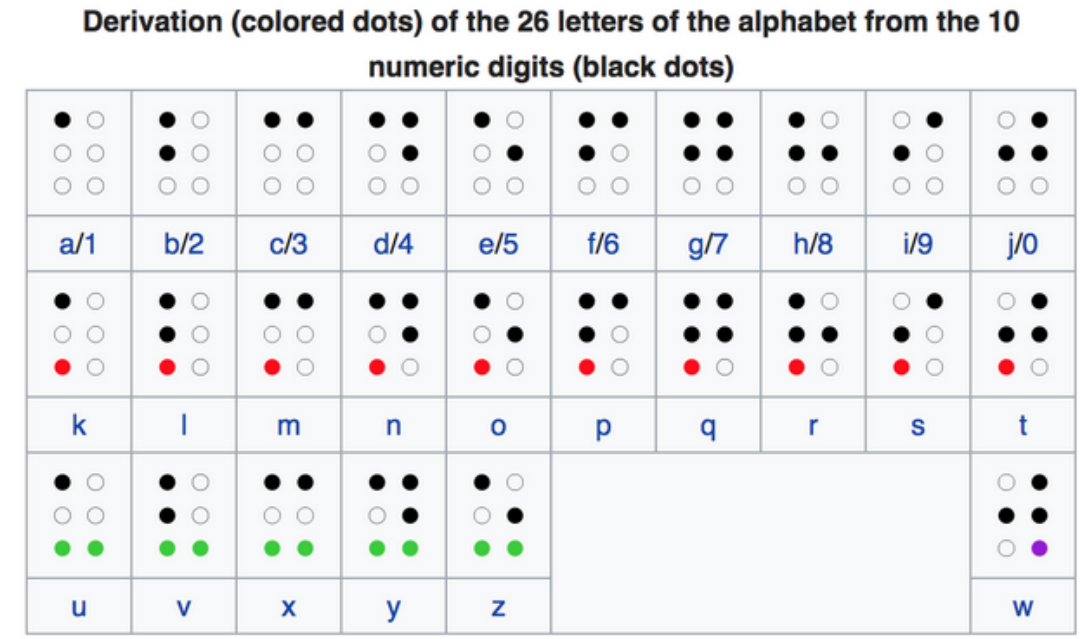


Figure 1: The 26 English alphabet letters and the corresponding grade 1 braille representations.

Grade 2 braille is just like grade 1 braille, except there are extra braille representations for commonly used words (e.g., "for," "the," "will") and letter combinations ("th," "-ing," "st"). This makes grade 2 braille a popular choice in most publications as it shortens the number of pages. Here, the keys should be strings corresponding to the word (e.g., "us"), as shown in Figure 2.

2 Getting Started

The provided skeleton code contains a `braille_dictionary` that is populated with the braille representations of the alphabet 'a-z'. The keys in the dictionary correspond to the ASCII characters, and the values are their braille representation. The braille representation is a list of 1's and 0's with raised dots represented as 1's and flat dots represented as 0's. For example, the key 'n' in the dictionary corresponds to the value: [1,0,1,1,1,0]. Note that as we discussed, the numbering starts in the upper left of the braille cell and goes down column one and then column 2.

2.1 Question 1

Your first task is to expand the `braile_dictionary` so that it now includes the 23 grade 2 words shown in Figure 2.

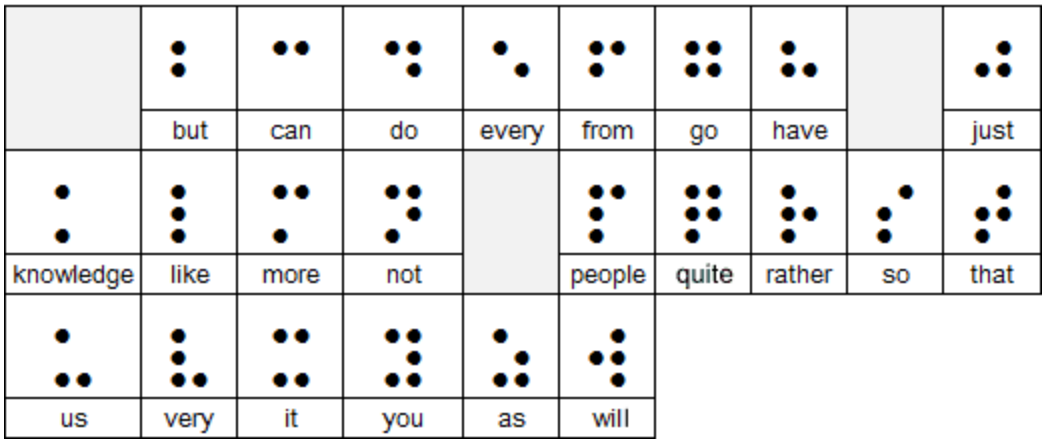


Figure 2: Example of grade 2 braille representations.

2.2 Question 2

Fill in the function `translator` that takes an English sentence that is readable by humans and a dictionary. The function should first convert the string to all lowercase and then split the string into a list of words, and then iterate through the list of individual words. If a word is contained in the dictionary (i.e., it is a grade 2 braille word), it should substitute the corresponding word in the list with the associated braille character. If the word is not in the dictionary (i.e., it is not a commonly used word and does not have a grade 2 representation), it should iterate through the word string, character by character. It should find each character in the dictionary and replace the word with the substituted braille character in a new list (each original character is the key and the replacement braille character is the value associated with that key in the dictionary). Anything that is not contained in the dictionary (e.g., numbers and symbols) should be ignored. The function should return a nested list of braille characters, with raised dots represented as 1's and flat dots represented as 0's.

For example, this is what happens when we pass the result of `translator` to `print`:

```
>> braille_list = translator('every computer', braille_dictionary)
>> braille_list
[[1, 0, 0, 0, 1, 0], [[1, 0, 0, 1, 0, 0], [1, 0, 1, 0, 1, 0], [1, 0, 1, 1, 0, 0], [1, 1, 1, 1,
>> print(braille_list)
--------------------------------------------------
1 0 1 1 1 0 1 1 1 1 1 0 0 1 1 0 1 0
0 1 0 0 0 1 0 0 1 0 0 0 1 1 0 1 1 1
0 0 0 0 1 0 1 0 1 0 1 1 1 0 0 0 1 0
```

The first two columns correspond to the word 'every' while the rest spell out 'computer.'

2.3 Question 3

Fill in the `draw_braille_character` that takes one parameter, a list of 0's and 1's corresponding to a braille character. The function should iterate through the list and use the `turtle` module to draw six circles corresponding to the six circles in a braille character. Make sure to fill in the appropriate circles depending on the input. The raised dots should be represented by filled-in circles, and the flat dots should be represented as empty circles.

For example, calling the helper function `draw_word` we have provided you with will result in the following picture drawn:

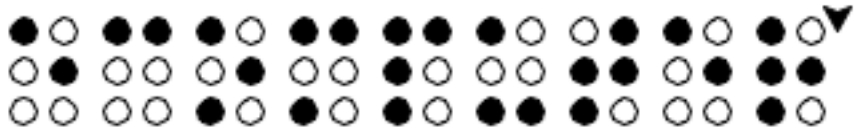


Figure 3: Example of grade 2 braille representations.

3 Extra credit

You might have noticed that our code is pretty incomplete if we were to build a full braille translator. We do not support numbers and punctuation, we don't have spaces, and some letters in grade 1 have the same representation as whole words in grade 2 representation. The same applies to numbers 1–0 in the first decade (row) in Figure 1. In practice, there are special characters that signify whether a character is a letter or a number, etc. Consider creating new functions that would address some of the challenges above when translating a sentence to braille code. Make sure you comment out at the top of your file and at the top of the newly introduced function(s) that you completed the extra credit.

4 Feedback

Create a file named `feedback8.txt` that answers the questions:

- How long did you spend on this assignment?
- Any comments or feedback? What things did you find interesting, challenging, or boring?

5 When you're done

Be sure that each function you write is preceded by a hash-marked comment that indicates the problem number, and inside it a triple-quoted comment that describes the function's purpose and

its inputs and outputs.

Any non-code answers may be included in a Python file as either triple-quoted strings or comments.

Any script elements that you want to test your programs should be placed at the bottom of the file, inside the

```
if __name__ == '__main__'
```

Submit your `assignment8.py`, which should contain your code and, at the top, the answers to the three questions, in comments, and `feedback8.txt` online using Gradescope. Be sure you're uploading them to the right assignment. Note that you can resubmit the same assignment as many times as you would like up until the deadline.

6 Grading

	points
Question 1	2
Question 2	4
Question 3	4
extra credit	1
total	10 (+1)