1. **List Comprehensions**

   What are the values of the following Python expressions, where `ENGLISH_WORDS` is a list of all English words from the `english.py` module:

   (a) `[ d for d in range(10) ]`

   (b) `[ x ** 2 for x in range(10) ]`

   (c) `[ chr(ord("A") + i) for i in range(5) ]`

   (d) `[ w for w in ENGLISH_WORDS if len(w) == 1 ]`

   How would you express the following values using a list comprehension?

   (a) `[ "0", "1", "2", "3", "4", "5", "6", "7"]`

   (b) `[ 1, 10, 100, 1000, 10000, 100000 ]`

   (c) The total number of English palindromes (where the word is the same forwards and backwards)

2. **Opening titles for *The Matrix***

Eighteen years after the last installment in *The Matrix* trilogy that started in 1999, Warner Brothers released a fourth movie, *The Matrix: Resurrections*, in December 2021. Each of the movies opens with an iconic title sequence in which green characters appear in columns down the screen, producing images like this:

Some of the characters are digits, but most are reminiscent of East Asian characters, primarily Japanese *katakana*.

In this problem, your job is to write a Python program that simulates this title sequence using the following process:

- Set up the graphics window and add a black `GRect` that covers the entire screen.
- Initialize an array that will keep track of how many characters you have added to each column, so that you will always know where the "bottom" is to add a new character.
- Set up a time process using `set_interval`.
- In each time step:
  - Pick a random katakana character (character codes between `0x30A0` and `0x30FF`.
  - Pick a random column (you will need to decide how many columns to have).
  - Create a `GLabel` containing the character and set its color to green.
  - Add the `GLabel` at the bottom of its column
- Continue this process until a character reaches the bottom of the window.

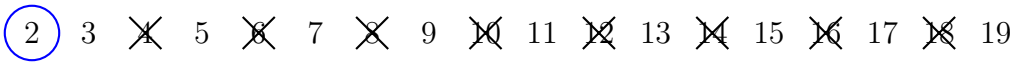
Although you might think you need a two-dimensional array to store the characters, all you need to do is place them in the window, much as you did with the bricks in Breakout. You do, however, need an array to keep track of how many characters there are in each column so that you can determine the y-coordinate and figure out when the program should stop.

3. **The Sieve of Eratosthenes**

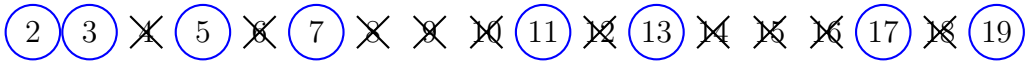
In the third century BCE, the Greek astronomer Eratosthenes developed an algorithm for finding all the prime numbers up to some upper limit. To apply the algorithm, you start by writing down a list of the integers from 2 up to but not including the limit. For example, if the limit were 20, you would begin by writing the following list:

2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19

You then circle the first unmarked number in the list, indicating that you have found a prime. Whenever you mark a number as a prime, you go through the rest of the list and cross off every multiple of that number, since none of those multiples can itself be prime. Thus, after executing the first cycle of the algorithm, you will have circled the number 2 and cross off every multiple of 2, as follows:



Eventually, every number in the list will either be circled or crossed out, as shown in this final diagram:



The circled numbers are the primes; the cross-out numbers the composites. This algorithm is called the *sieve of Eratosthenes*.

Write a function `create_prime_list(limit)` that uses the sieve of Eratosthenes to return a list of the primes starting at 2 and going up to but not including the `limit`. You should, for example, be able to replicate the following session:

```
>>> create_prime_list(20)
[2, 3, 5, 7, 11, 13, 17, 19]
```