Jerry Cain
CS 106AX

Handout #6
October 8th, 2025

Section Handout #2

Portions of this handout by Eric Roberts, Patrick Young, Jeremy Keeshin, Mehran Sahami, Nick Troccoli, and Kat Gregory.

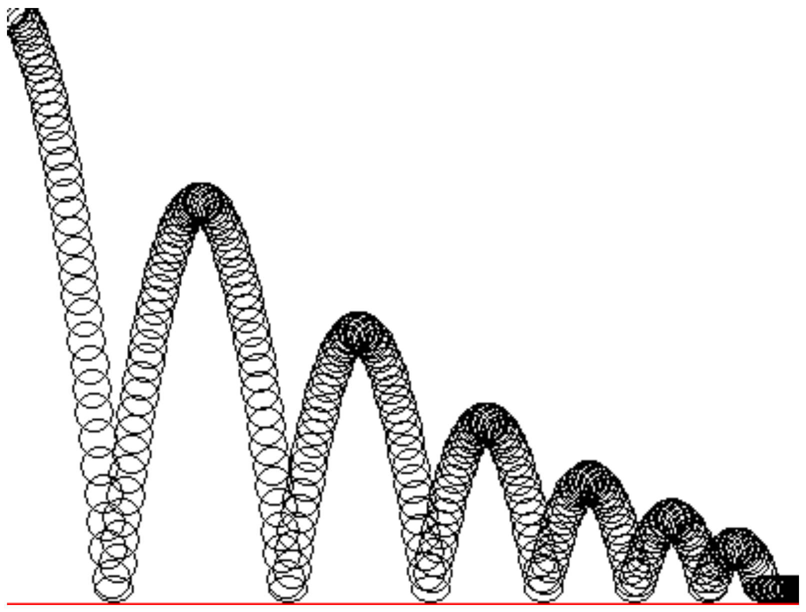
Problem 1: Bouncing Balls

Write an interactive graphics program that simulates the motion of balls bouncing under the influence of gravitational force.

Each time the user clicks, drop a randomly colored ball from the top left corner of the window. The ball should begin with a random velocity in the X direction and no velocity in the Y direction. It will fall, accelerating towards the bottom of the screen, until it hits the bottom of the window. As it bounces, the ball will reverse direction and begin decelerating upwards, but it will also lose some energy from the collision. It should continue bouncing until it rolls off the right-hand end of the window. Like the previous exercise, your program should be able to handle multiple balls bouncing at once if the user clicks in quick succession.

```
/* Constants */
const GWINDOW_WIDTH = 800;
const GWINDOW_HEIGHT = 300;
const DIAMETER = 20;
const MIN_X_VEL = 3;
const MAX_X_VEL = 15;
const TIME_STEP = 20;
const GRAVITY = 3; // amount Y velocity is increased each cycle
const BOUNCE_REDUCE = 0.75; // amount Y velocity is reduced during bounce
```

The diagram below illustrates the trajectory of the dropped ball:



Problem 2: Spoonerisms

A ***spoonerism*** is a phrase in which the leading consonant strings of the first and last words are inadvertently swapped, generally with comic effect. Some examples of spoonerisms include the following phrases and their spoonerized counterparts (the consonant strings that get swapped are underlined):

crushing blow ® *blushing crow*

sons of toil ® *tons of soil*

pack of lies ® *lack of pies*

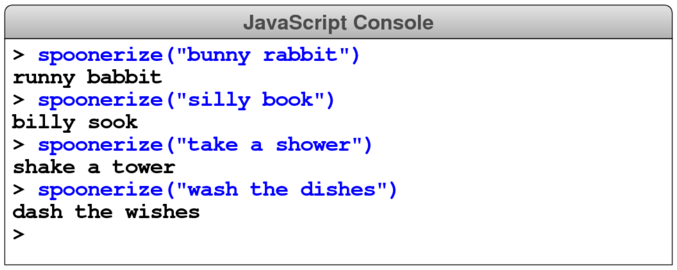
jelly beans ® *belly jeans*

flutter by ® *butter fly*

In this problem, your job is to write a function

```
function spoonerize(phrase)
```

that takes a multiword phrase as its argument and returns its spoonerized equivalent. For example, you should be able to use your function to duplicate the following console session in which all the examples come from Shel Silverstein's spoonerism-filled children's book *Runny Babbit:*



JavaScript Console

```
> spoonerize("bunny rabbit")
runny babbit
> spoonerize("silly book")
billy sook
> spoonerize("take a shower")
shake a tower
> spoonerize("wash the dishes")
dash the wishes
>
```

In this problem, you are not responsible for any error-checking. You may assume that the phrase passed to **`spoonerize`** contains nothing but lowercase letters along with spaces to separate the words. You may also assume that the phrase contains at least two words, that there are no extra spaces, and that each word contains at least one vowel. What your method needs to do is extract the leading consonant clusters from the first and last words and then exchange them, leaving the rest of the phrase alone.

Hint: Remember that you can use methods from the book. The **`findFirstVowel`** and **`isEnglishVowel`** methods from the Pig Latin example in Monday's slide deck will come in handy.

Problem 3: Look and Say

The **look-and-say** sequence is the sequence of numbers that starts out with:

```
1
11
21
1211
111221
312211
13112221
```

Each number in the sequence is generated by "reciting" its predecessor and then capturing what was said in a number format that should be clear from the example below.

Reciting 111221 out loud, you'd look and say: 3 ones, followed by 2 twos, followed by 1 one, or 312211. Reading 312211 aloud, you'd say: 1 three, 1 one, 2 twos, 2 ones, or 13112221.

look-and-say arrays are similar, but numbers are instead expressed as arrays of isolated digits, as with [3, 1, 2, 2, 1, 1]. Reading the array aloud, you'd say 1 three, 1 one, 2 twos, 2 ones, which we'd captured in array form as [1, 3, 1, 1, 2, 2, 2, 1].

For this problem, you're to write the **`lookandsay`** function, which accepts an array of single digit numbers and generates its successor according to the look-and-say rules outlined here. For example:

`lookandsay([3, 1, 2, 2, 1, 1])` returns **`[1, 3, 1, 1, 2, 2, 2, 1]`**
`lookandsay([1, 3, 1, 1, 2, 2, 1, 1])` returns **`[1, 1, 1, 3, 2, 1, 2, 2, 2, 1]`**

Work together to produce a working implementation coded to the following prototype:

```
function lookandsay(digits) {
   let result = []; // ultimately return this result
```