Problem 1: Simulating Physics with Bouncing Balls

Problem focus: interactive graphics + animations

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:

![Diagram of the trajectory of the dropped ball](image)
Problem 2: Spoonerisms

Problem focus: strings

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*:

```
> spoonerize("bunny rabbit")
runny babbit
> spoonerize("silly book")
billy suck
> 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 initial consonant substrings from the first and last words and then exchange those strings, 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 lecture example will certainly come in handy.
Problem 3: String Split

Problem focus: strings, and a little bit of arrays

JavaScript's String class includes a `split` method that accepts a separator string (e.g. " ") or "sh") and splits the receiving string into an array of substrings using the provided separator to determine where to make each split. Here are some examples illustrating how this built-in `split` method works:

```
"this is how split works".split("i") ⇒ ["th", "s", "s how spl", "t works"]
"abracadabra".split("a") ⇒ ["", "br", "c", "d", "br", "]
"sheepishly".split("sh") ⇒ ["", "eepi", "ly"]
```

Note that the last example above splits on a separator that's more than a single character. And when the separator appears at the beginning and end of the receiving string, the first and last entries of the split are empty strings.

There are scenarios, however, where you want to split around all of many different single-character separators, not just one. A top-level function called `split` — which doesn't exist in JavaScript, so you'll need to implement it yourself — might operate like this:

```
split("abcdefghijklmnopqrstuvwxyz", "aeiou") ⇒
["", "bcd", "fgh", "jklmn", "pqrst", "vwxyz"]
split("abracadabra", "abcd") ⇒ ["", ",", ",", ",", ",", "", ",", ""]
```

The separator string is really a set of many single-character strings, and each single-character string is a separator. The first of the two examples above splits around the five lowercase vowels, and the seconds splits a word around the first four letters of the lowercase alphabet.

Leverage your understanding of strings and Wednesday's introduction to arrays, implement the top-level function called `split`.

```javascript
function split(str, separators) {

split("abcdefghijklmnopqrstuvwxyz", "aeiou") ⇒
["", "bcd", "fgh", "jklmn", "pqrst", "vwxyz"]
split("abracadabra", "abcd") ⇒ ["", ",", ",", ",", ",", "", ",", ""]
```

```javascript
function split(str, separators) {

split("abcdefghijklmnopqrstuvwxyz", "aeiou") ⇒
["", "bcd", "fgh", "jklmn", "pqrst", "vwxyz"]
split("abracadabra", "abcd") ⇒ ["", ",", ",", ",", ""]
```