Solution 1: Simulating Gravity with Bouncing Balls

/*
 * File: BouncingBalls.js
 * -----------------------
 * This program graphically drops a bouncing ball each time user clicks.
 */

const GWINDOW_WIDTH = 500;
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 vel is increased each cycle
const BOUNCE_REDUCE = 0.75; // Amount Y velocity is reduced during bounce

/* Main program */
function BouncingBalls() {

    let gw = GWindow(GWINDOW_WIDTH, GWINDOW_HEIGHT);

    let createNewBall = function() {
        let newBall = GOval(0, 0, DIAMETER, DIAMETER);
        newBall.setFilled(true);
        newBall.setColor(randomColor());
        return newBall;
    }

    let clickAction = function(e) {
        let ball = createNewBall();
        let xVel = randomReal(MIN_X_VEL, MAX_X_VEL);
        let yVel = 0;
        gw.add(ball);

        let checkForCollision = function() {
            // Determine if ball has dropped below the floor
            if (ball.getY() > GWINDOW_HEIGHT - DIAMETER) {
                // Change ball's Y velocity to now bounce upwards
                yVel = -yVel * BOUNCE_REDUCE;

                // Assume bounce will move ball an amount above the floor
                // equal to the amount it would have dropped below the floor.
                let diff = ball.getY() - (GWINDOW_HEIGHT - DIAMETER);
                ball.move(0, -2 * diff);
            }
        }

        let step = function() {
            if (ball.getX() < GWINDOW_WIDTH) {
                yVel += GRAVITY;
                ball.move(xVel, yVel);
                checkForCollision();
            } else { // Simulation ends when ball exits right side of screen
                clearInterval(timer);
            }
        }

        let timer = setInterval(step, TIME_STEP);
    }

    gw.addEventListener("click", clickAction);
}
Solution 2: Spoonerisms

/**
 * Function: spoonerism
 * -------------------
 * Defines the spoonerism function according to the specifications
 * laid out in the first practice midterm.
 */
function spoonerism(phrase) {
    let sp1 = phrase.indexOf(' ');
    let sp2 = phrase.lastIndexOf(' ');
    let orig1 = phrase.substring(0, sp1);
    let orig2 = phrase.substring(sp2 + 1);
    let middle = phrase.substring(sp1, sp2 + 1);
    let vp1 = findFirstVowel(orig1);
    let vp2 = findFirstVowel(orig2);
    let transformed1 = orig2.substring(0, vp2) + orig1.substring(vp1);
    let transformed2 = orig1.substring(0, vp1) + orig2.substring(vp2);
    return transformed1 + middle + transformed2;
}

/**
 * Function: findFirstVowel
 * ------------------------
 * Returns the index of the first lowercase vowel, or -1 if no lowercase
 * vowel could be found.
 */
function findFirstVowel(str) {
    for (let i = 0; i < str.length; i++) {
        if (isEnglishVowel(str.charAt(i))) { return i; }
    }
}

/**
 * Function: isEnglishVowel
 * -----------------------
 * Returns true if and only if the provided string is of length 1, and
 * its one characters is a lowercase vowel.
 */
function isEnglishVowel(ch) {
    return ch.length === 1 & & "aeiou".indexOf(ch) >= 0;
}
Solution 3: String Split

Some thought questions to ensure you understand the solution:

- Why does the for loop test rely on <= instead of <?
- What’s the best description you have for what i is tracking on behalf of the algorithm?
- The internal if test checks to see if i === str.length first before advancing on to check the return value of indexOf?

```javascript
/**
 * Function: split
 * ------------
 * Returns an array of the applied string when exploded around
 * all of the characters within the supplied delimiter.
 */
function split(str, delimiters) {
    let start = 0;
    let fragments = [];
    for (let i = 0; i <= str.length; i++) {
        if (i === str.length || delimiters.indexOf(str.charAt(i)) !== -1) {
            let fragment = str.substring(start, i);
            fragments.push(fragment);
            start = i + 1;
        }
    }
    return fragments;
}
```