In this assignment you will write a Processing program that uses functions to emulate the program BallBounce found posted in the Examples section of the course webpage:

https://classes.soe.ucsc.edu/cmps005j/Fall15/Examples/BallBounce.html

As usual no source code is provided, and your task is to reverse-engineer the program.

The program opens a 500 by 500 processing window containing a cyan background and a red ball. The ball proceeds to drop and bounce under the influence of gravity. Its initial behavior is identical to the example GravityBounce2 also posted at the Examples page. The difference is that it is possible for you to grab the ball with the mouse and propel it at the floor, ceiling and walls. Observe that the ball eventually comes to rest when its energy dissipates. *(See note at end of this document regarding the webpage version of this program.)*

Begin this project by studying GravityBounce2, a refinement of GravityBounce which we studied earlier in the quarter. This new version eliminates the continuous oscillation occurring at the end of the GravityBounce program. The key to eliminating this undesirable feature is to place the command

\[
\text{speed} = \text{speed} + \text{gravity};
\]

after the conditional

\[
\text{if( y > height ) speed} = \text{speed}\times(-0.95);
\]

rather than before (referring here to the GravityBounce source code.) See if you can figure out why this small change would eliminate the oscillation. Notice also that GravityBounce2 uses the `constrain()` function (see Processing reference) to keep the square in bounds. This eliminates another problem with GravityBounce in which the square can get stuck below the bottom of the screen.

GravityBounce2 uses three float variables (gravity, stopSpeed, and dissipation) to set the physical environment. Try changing these parameters to see what effect they have on the behavior of the bouncing square. Note that stopSpeed is the threshold below which speed is set to zero. This parameter is actually not necessary in GravityBounce2 to stop the square from bouncing (try setting it to zero), but it will be necessary in this project to stop the side to side motion of the bouncing ball.

This project is different from previous ones in that you have to not just emulate the behavior of the example BallBounce.html, but you must adhere to certain specifications listed below. In other words the project requires that you do things in a specific way.

**Specifications**

1. Include global float variables called gravity, stopSpeed, and dissipation with the same roles and initial values as in GravityBounce2.
2. Include global float variables called X, Y, Xspeed, and Yspeed to specify the state of the ball. It is suggested (not required) that you include a variable for the ball radius, which is 50 pixels.
3. Include the five functions displayBall(), holdBall(), moveBall(), updateSpeed() and mouseOnBall(). The operation and purpose of these functions is described below.
4. Your `draw()` function must be as given below, with no changes.

```java
void draw()
{
  background(0,255,255);
  displayBall();
  if( mousePressed && mouseOnBall() ){
    holdBall();
  }else{
    moveBall();
    updateSpeed();
  }
}
```

Each of the five required functions takes no arguments. Function `mouseOnBall()` returns a boolean, and the others return `void`. Function `displayBall()` first constrains the global variables `X` and `Y` to lie in bounds, then draws the red ball at its current location in the Processing window. Function `holdBall()` controls the location of the ball by setting `X` and `Y` to `mouseX` and `mouseY` respectively. It also sets `Xspeed` and `Yspeed` appropriately so that when the ball is released, it follows the trajectory of the mouse. Use `pmouseX` and `pmouseY` to accomplish this. Function `moveBall()` changes the position of the ball by adding `Xspeed` and `Yspeed` to `X` and `Y` respectively. Function `updateSpeed()` changes `Xspeed` and `Yspeed` appropriately so as to dissipate energy when bouncing off walls, and to become zero when they fall below the `stopSpeed` threshold. `Yspeed` must also be influenced by gravity. Function `mouseOnBall()` returns `true` if and only if the mouse lies within 50 pixels of the ball center. This is where a variable for the ball radius might be appropriate. Use the math function `dist()` (see Processing reference) to test for this.

An outline of this program is included below. It contains the required `draw()` function as well as stubs for all of the other required functions. Complete the project by filling in this outline. Start early and ask plenty of questions. Attach your source code `BallBounce.pde` to the assignment pa6 in eCommons before the end of the grace period.

// BallBounce
// cmps 5J
// programming assignment 6
//

// ball variables
float X, Y, Xspeed, Yspeed;

// environment variables
float gravity, stopSpeed, dissipation;

void setup() {
  size(500,500);
  smooth();

  // initialize ball variables

  // initialize environment variables
}
void draw() {
  background(0,255,255);
  displayBall();
  if( mousePressed && mouseOnBall() ){
    holdBall();
  }else{
    moveBall();
    updateSpeed();
  }
}

void displayBall(){
  // your code goes here
}

void holdBall(){
  // your code goes here
}

void moveBall(){
  // your code goes here
}

void updateSpeed(){
  // your code goes here
}

boolean mouseOnBall(){
  // your code goes here
}

*Note:
The program posted on the webpage exhibits some behavior that is not specified in these instructions. In particular the ball will sometimes stick to the walls or ceiling. This behavior is neither required nor intended and is caused by an imperfect translation of the program into a language suitable for posting in a webpage. When you run your program in processing the ball should simply bounce, never stick.