Acceleration & Dampening w/ Processing

LAB 00c: Processing - Acceleration

Start with the simple example...

In class, we ended up with an example of a ball bouncing from side to side.

You added the part which allows the ball to bounce off the top and bottom.

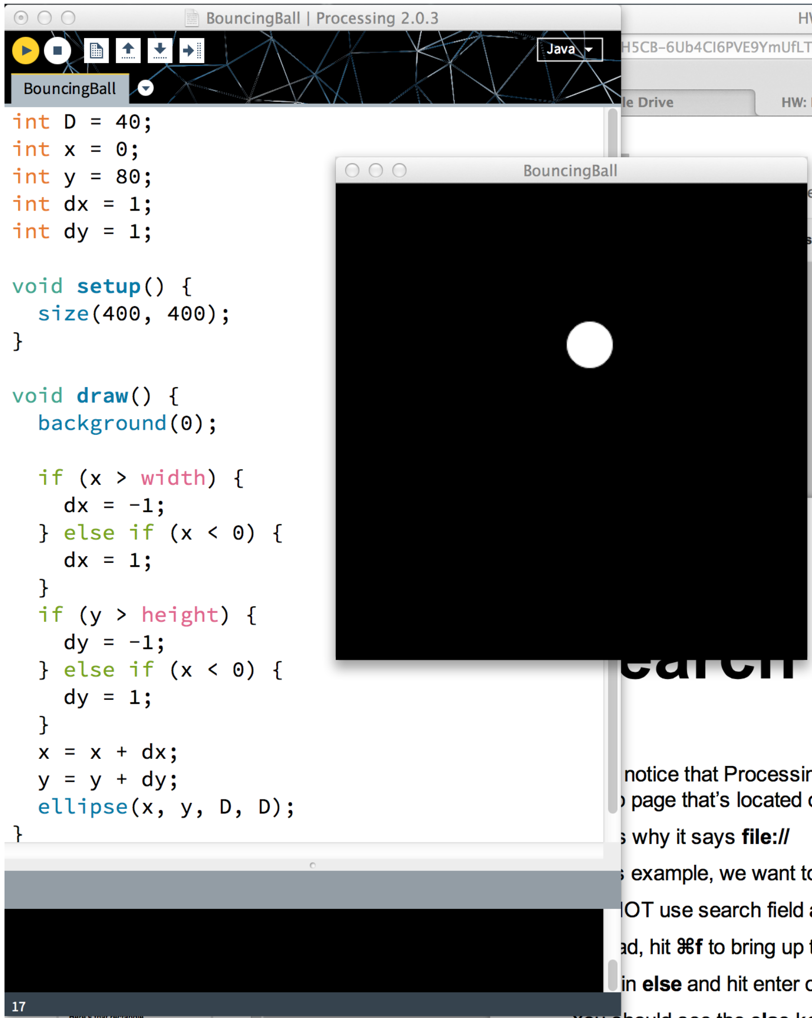
To make sure we're all starting from the same place, make sure you have something that looks like this.

If you don't have this thing working, now's the time to ask for help.

If you want to start with the exact same thing, you can copy it from this image. Or if you prefer cut and paste, you can look at this project on GitHub:

https://github.com/AgencyAgency/P5Demos/blob/master/BouncingBall/BouncingBall.pde

Okay, have your baseline project set up and bouncing? Good...



```
int D = 40;
int x = 0;
int y = 80;
int dx = 1;
int dy = 1;

void setup() {
  size(400, 400);
}

void draw() {
  background(0);

  if (x > width) {
    dx = -1;
  } else if (x < 0) {
    dx = 1;
  }
  if (y > height) {
    dy = -1;
  } else if (x < 0) {
    dy = 1;
  }
  x = x + dx;
  y = y + dy;
  ellipse(x, y, D, D);
}
```

If you have any questions...

Processing has a lot of documentation to help you out. It also has a ton of tutorials.

The tutorials can be found here on the Processing website:

> http://processing.org/tutorials/

You can also find Processing example projects here:

> http://processing.org/examples/

And, free online books on Processing:

> http://www.learningprocessing.com
>
> http://processing.org/books/

And, videos:

> http://vimeo.com/channels/natureofcode

I can’t stress enough how awesome this is. These resources above (and there are more) are the equivalent of someone handing you a precious gift. You’ll most likely be more excited about this as we continue further down the rabbit hole.

Ask me if you have questions…

Documentation → HELP

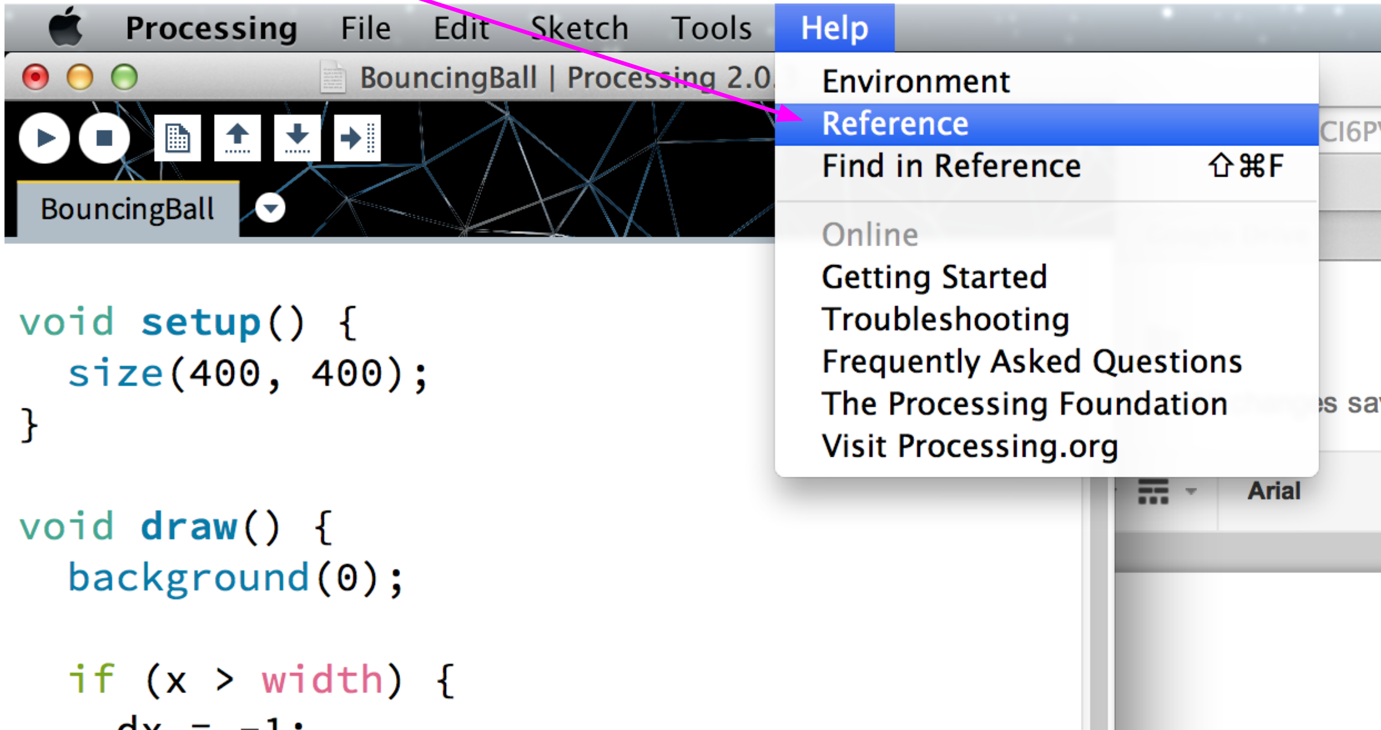
But, also please find the on-board Processing reference…

I call it “on-board” because you don’t have to have an internet connection to use them. You just have to launch the Processing.

Let’s say you have a question about the **else** clause of **if** statements.

Open the Processing app.

Click on **Help > Reference** in the top menu bar.



```
void setup() {
  size(400, 400);
}

void draw() {
  background(0);

  if (x > width) {
```

Search for “else”

You’ll notice that Processing launches your browser. But, notice that it’s showing you a web page that’s located on your computer (not on the internet).

That’s why it says **file://**

In this example, we want to find else on this page.

DO NOT use search field at the top of this web page.

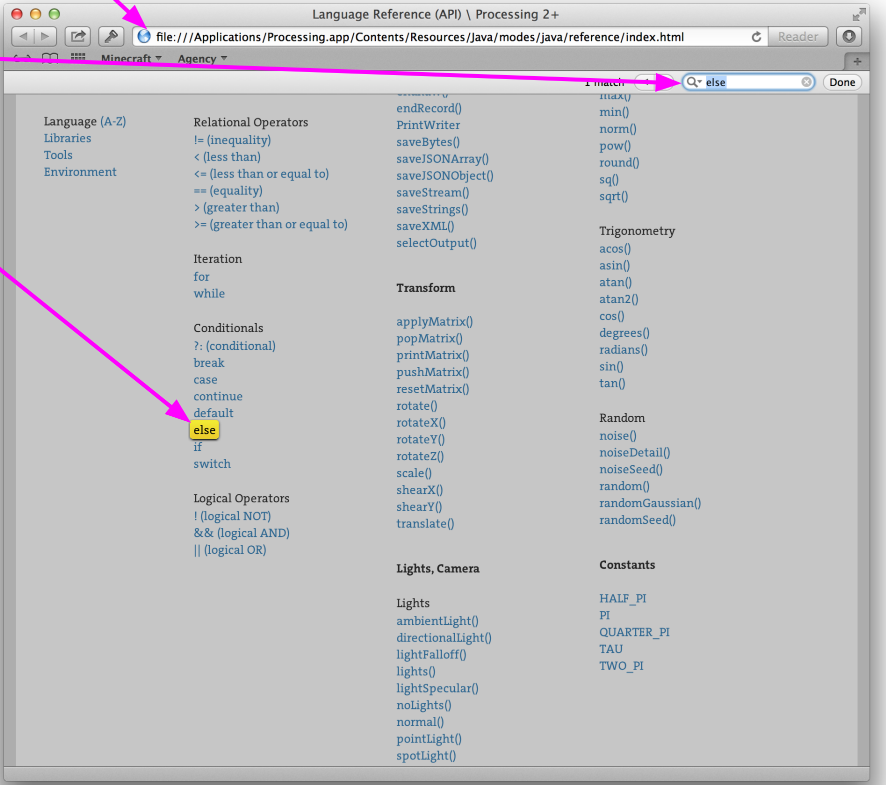
Instead, hit **⌘f** to bring up the browser search field.

Type in **else** and hit enter or done.

You should see the **else** keyword highlighted in the page.

Click it.

You’ll see a page with a ton of info on how to use **else**.


Language Reference (API) \ Processing 2+
file:///Applications/Processing.app/Contents/Resources/Java/modes/java/reference/index.html
Reader
Minecraft
Agency
1 match
else
Done

Language (A-Z)
Libraries
Tools
Environment

Relational Operators
!= (inequality)
< (less than)
<= (less than or equal to)
== (equality)
> (greater than)
>= (greater than or equal to)

Iteration
for
while

Conditionals
?: (conditional)
break
case
continue
default
else
if
switch

Logical Operators
! (logical NOT)
&& (logical AND)
|| (logical OR)

endRecord()
PrintWriter
saveBytes()
saveJSONArray()
saveJSONObject()
saveStream()
saveStrings()
saveXML()
selectOutput()

Transform

applyMatrix()
popMatrix()
printMatrix()
pushMatrix()
resetMatrix()
rotate()
rotateX()
rotateY()
rotateZ()
scale()
shearX()
shearY()
translate()

Lights, Camera

Lights
ambientLight()
directionalLight()
lightFalloff()
lights()
lightSpecular()
noLights()
normal()
pointLight()
spotLight()

max()
min()
norm()
pow()
round()
sq()
sqrt()

Trigonometry
acos()
asin()
atan()
atan2()
cos()
degrees()
radians()
sin()
tan()

Random
noise()
noiseDetail()
noiseSeed()
random()
randomGaussian()
randomSeed()

Constants

HALF_PI
PI
QUARTER_PI
TAU
TWO_PI

We want to add gravity.

We won’t add real gravity.

But, we do want the ball to speed up as it falls and slow down as it goes up.

That’s how gravity works.

We won’t simulate is exactly. We just want to create the **feel** of gravity. This is an art project, not a physics simulation.

In order to do this, we have to change how we’re moving the ball around. It’s just a few small changes. So, let’s get started...

Switching from ints to floats.

Take a look at what's going on here.

All I did was change the int declarations on **x**, **y**, **dx**, and **dy** to the **float** type.

int stands for integer. Integers are whole numbers. They don't have a decimal point.

In short, a **float** is a number with a decimal part.

These are of type **int**

1

2

3

42

These are of type **float**

0.2

0.9

0.999994

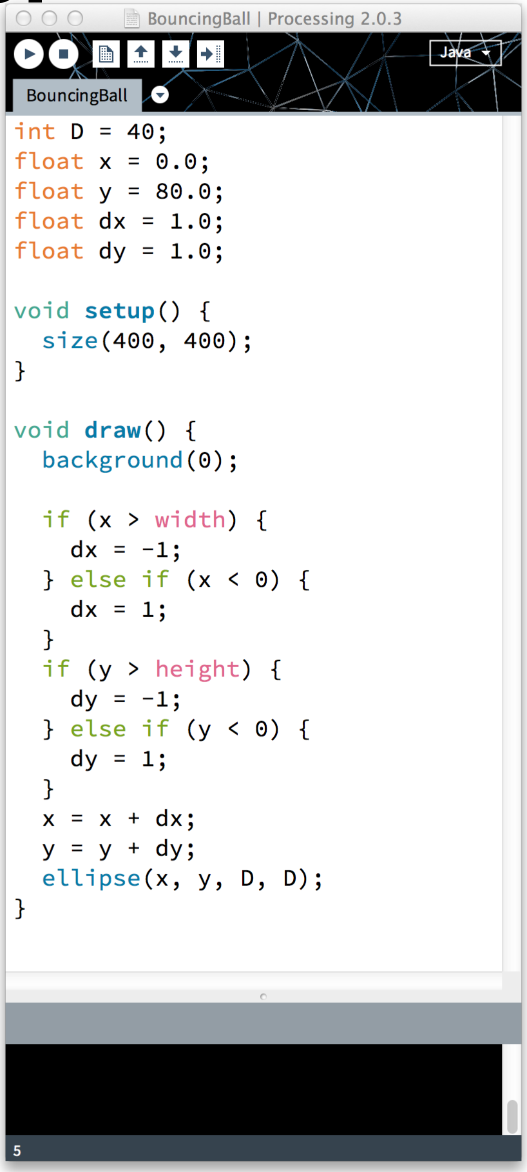
1.103

4.99

97.6

1000000000.001

Run this and you'll see that it looks the same.



```
int D = 40;
float x = 0.0;
float y = 80.0;
float dx = 1.0;
float dy = 1.0;

void setup() {
  size(400, 400);
}

void draw() {
  background(0);

  if (x > width) {
    dx = -1;
  } else if (x < 0) {
    dx = 1;
  }
  if (y > height) {
    dy = -1;
  } else if (y < 0) {
    dy = 1;
  }
  x = x + dx;
  y = y + dy;
  ellipse(x, y, D, D);
}
```

Switch directions using absolute value.

Do you remember what an **absolute value** is?

There's a function in Processing that computes it for you. It's called **abs()**.

Here are some examples of how you use it…

abs(10.0) → 10.0

abs(-10.0) → 10.0

abs(5) → 5.0

abs(-5) → 5

abs(42) → 42

abs(-42) → 42

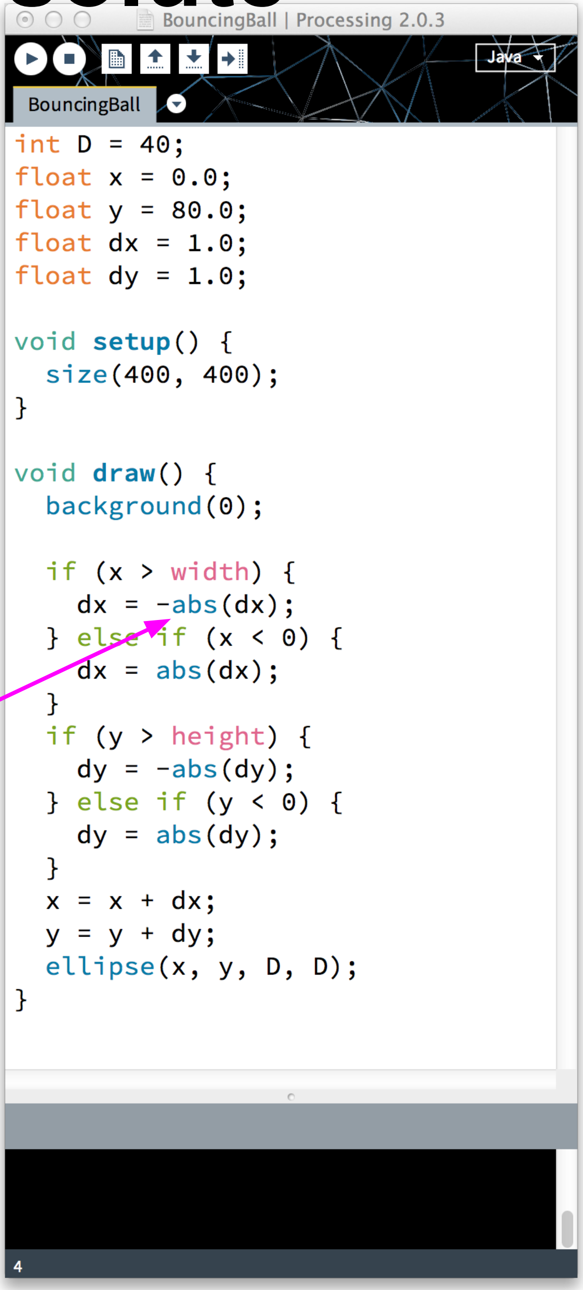
x = -10;

abs(x) → 10

You see, no matter what you pass to abs(), it always gives you back a positive number. It always strips away the negative sign.

What it's doing for you is giving you the amplitude of your value.

Anywho… here's how to use it.



```
int D = 40;
float x = 0.0;
float y = 80.0;
float dx = 1.0;
float dy = 1.0;

void setup() {
  size(400, 400);
}

void draw() {
  background(0);

  if (x > width) {
    dx = -abs(dx);
  } else if (x < 0) {
    dx = abs(dx);
  }
  if (y > height) {
    dy = -abs(dy);
  } else if (y < 0) {
    dy = abs(dy);
  }
  x = x + dx;
  y = y + dy;
  ellipse(x, y, D, D);
}
```

Wait. What?

What is this doing?

It may seem a little redundant. And, it might remind you of what we did on the iPad.

We're forcing **dx** to be **negative** when we want to move to the left.

We're forcing **dx** to be **positive** when we want to move to the right.

We're forcing **dy** to be **negative** when we want to move up.

We're forcing **dy** to be **positive** when we want to move down.

Why go through all this trouble?

We'll now we're only ever setting dx and dy to 1.0 in one place. Right here.

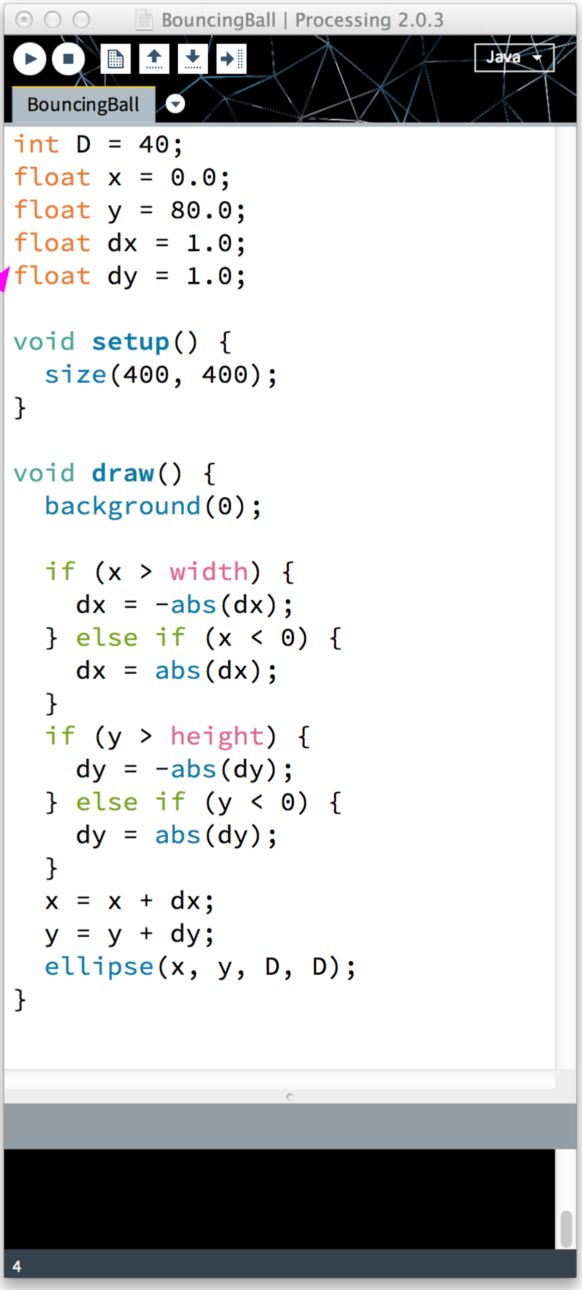
So, if we want to change how fast the ball moves, we have one variable to mess with.

Well, two variables, one for horizontal velocity (**dx**). One for vertical velocity (**dy**).

Go ahead and change **dx** and **dy** to something larger, like 10.0.

Run your app and see how fast it goes now.

Okay, switch it back to slow old 1.0. We need it to be slow for our gravity example...



```
int D = 40;
float x = 0.0;
float y = 80.0;
float dx = 1.0;
float dy = 1.0;

void setup() {
  size(400, 400);
}

void draw() {
  background(0);

  if (x > width) {
    dx = -abs(dx);
  } else if (x < 0) {
    dx = abs(dx);
  }
  if (y > height) {
    dy = -abs(dy);
  } else if (y < 0) {
    dy = abs(dy);
  }
  x = x + dx;
  y = y + dy;
  ellipse(x, y, D, D);
}
```

A quick artistic detour.

You may have notices that the ball doesn't actually bounce off the sides of the window.

This is because the **x** and **y** position of the ball is tied to its center. So, all the if-else **conditionals** we're using here are talking about the ball's center.

I want to bounce the ball when its **edge** reaches a side of the window. What do we change to accomplish this?

Well to make things a little simpler, I've created a radius variable, **R**.

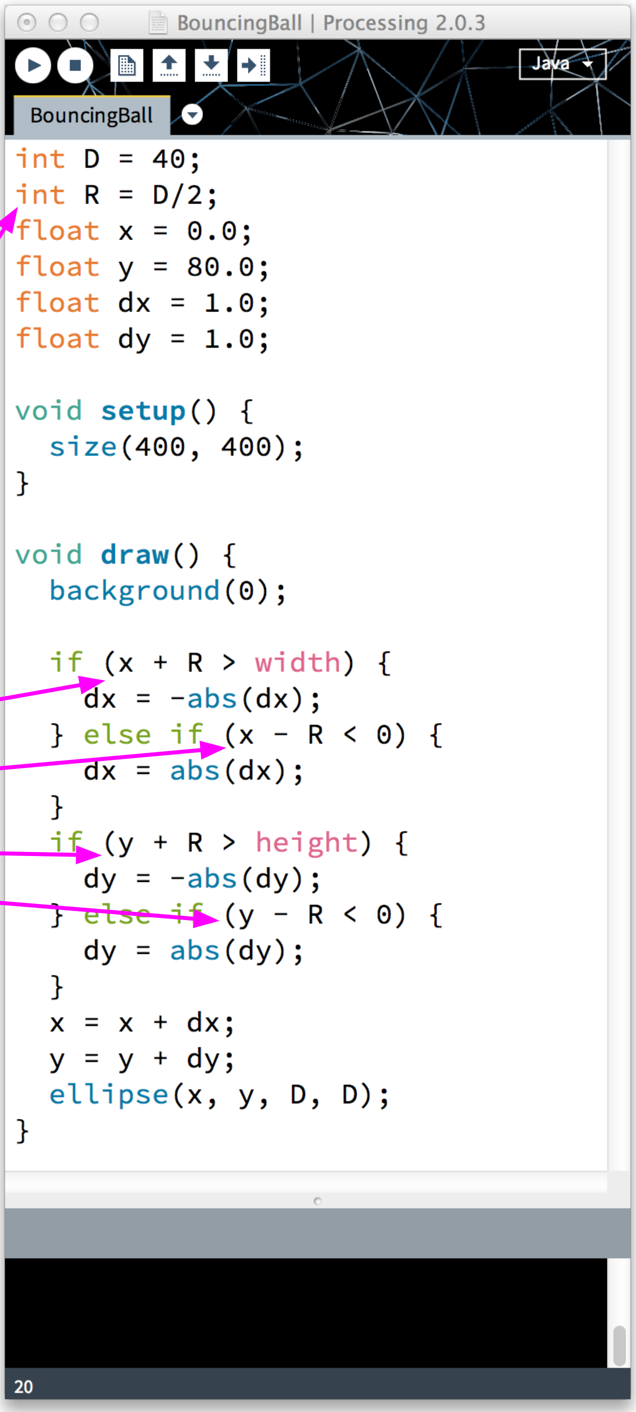
I then use it in the if-else conditionals to check if the ball's edge collides with the window frame.

I have to add it in four places.

Convince yourself that this makes sense.

I know this is a lot of math. Sorry. If you're annoyed, that's probably a good thing. Because we'll learn how to do things in less complicated ways in the future. And, if it seems tedious now, that means you'll be motivated to make it more elegant later.

Your ball should now bounce off its edges. Score one for realism.



```
int D = 40;
int R = D/2;
float x = 0.0;
float y = 80.0;
float dx = 1.0;
float dy = 1.0;

void setup() {
  size(400, 400);
}

void draw() {
  background(0);

  if (x + R > width) {
    dx = -abs(dx);
  } else if (x - R < 0) {
    dx = abs(dx);
  }
  if (y + R > height) {
    dy = -abs(dy);
  } else if (y - R < 0) {
    dy = abs(dy);
  }
  x = x + dx;
  y = y + dy;
  ellipse(x, y, D, D);
}
```

Add a variable for gravity.

But, what would make this cooler is if we added gravity. Let's add a force of gravity so that the ball slows down as it rises, and speeds up as it falls.

Here's the new gravity variable.

How does gravity affect velocity? It is like a constant force pulling downward (to the center of the Earth). We can approximate this by just adding **g** to our **dy**.

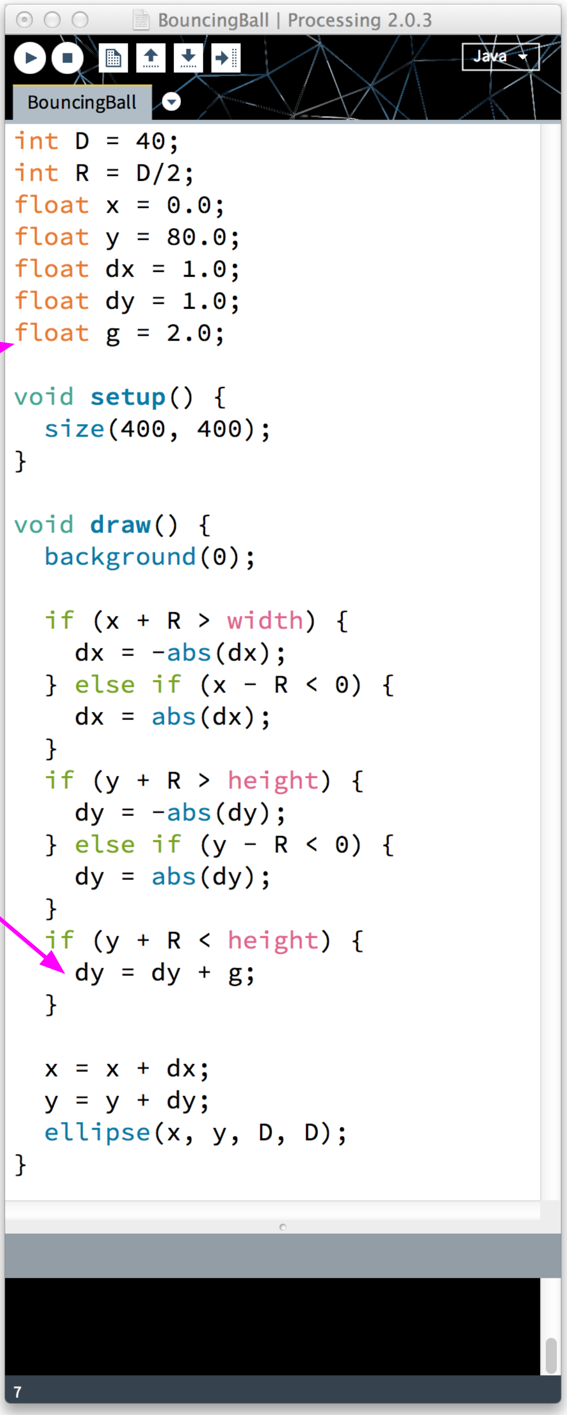
You'll also notice that I've placed this in another **if conditional**. I'm doing this to ensure that the ball never moves below the bottom of the window. It's a bit of a hack, but it works.

Run it now.

You get some semi-realistic bouncing-with-gravity action here.

But, something is missing. What do you think would make it look more realistic?

One possible answer is **dampening**...



```
int D = 40;
int R = D/2;
float x = 0.0;
float y = 80.0;
float dx = 1.0;
float dy = 1.0;
float g = 2.0;

void setup() {
  size(400, 400);
}

void draw() {
  background(0);

  if (x + R > width) {
    dx = -abs(dx);
  } else if (x - R < 0) {
    dx = abs(dx);
  }
  if (y + R > height) {
    dy = -abs(dy);
  } else if (y - R < 0) {
    dy = abs(dy);
  }
  if (y + R < height) {
    dy = dy + g;
  }

  x = x + dx;
  y = y + dy;
  ellipse(x, y, D, D);
}
```

Add dampening.

Dampening is like friction. Whenever a ball bounces, it loses a little energy. You'll notice that as our sketch currently runs, the ball never comes to rest. That's because the velocity never reaches zero.

In the real world, energy would be lost to striking the ground, the walls, the air… and the ball would eventually slow down and stop.

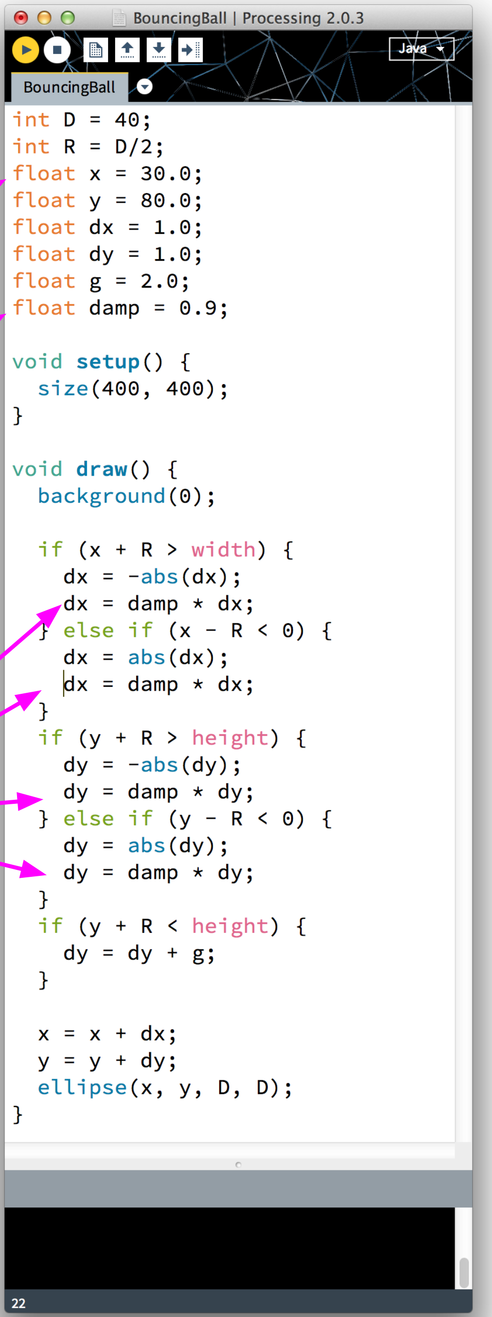
Note, I've moved the starting **x** position of the ball into the middle of the window, to avoid the ball getting *stuck* in the wall.

Here's the new dampening constant.

It's a bit repetitive, but I want to dampen the velocity any time the ball strikes a wall. There are four places where we want to do that.

Run it now. The ball should come to rest. I might jiggle around a little and roll back and forth. But, the bouncing is definitely ending.

Try other values for **damp** to see how that affects your simulation. For me, the sweet spot is between 0.7 and 0.9.



```
int D = 40;
int R = D/2;
float x = 30.0;
float y = 80.0;
float dx = 1.0;
float dy = 1.0;
float g = 2.0;
float damp = 0.9;

void setup() {
  size(400, 400);
}

void draw() {
  background(0);

  if (x + R > width) {
    dx = -abs(dx);
    dx = damp * dx;
  } else if (x - R < 0) {
    dx = abs(dx);
    dx = damp * dx;
  }
  if (y + R > height) {
    dy = -abs(dy);
    dy = damp * dy;
  } else if (y - R < 0) {
    dy = abs(dy);
    dy = damp * dy;
  }
  if (y + R < height) {
    dy = dy + g;
  }

  x = x + dx;
  y = y + dy;
  ellipse(x, y, D, D);
}
```

From the top...

Okay. So a few things to remember…

This is just one way to **solve** this bouncing ball problem. It's sort of a physical simulation… sort of. We're not using real physics exactly, but we're applying the concepts of velocity, acceleration (gravity), and dampening.

How much more difficult do you think it would be to make this a **real** physics simulation? In concept, not much more difficult. But, probably pretty tedious… with little aesthetic benefit over what you've already accomplished.

So, it's nice to be guided by nature (and your understanding of physics). But, it's not the rule.

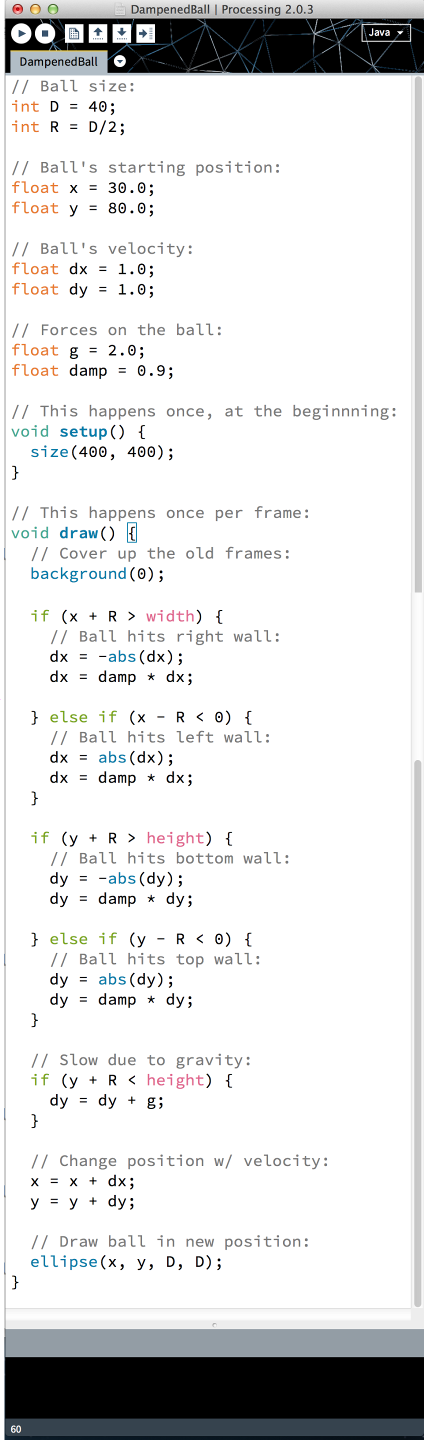
Take a look at my version of the completed sketch.

It's pretty small here. So, you can take a look on GitHub to see it at full size. Notice that I've added comments to each of the main parts.

https://github.com/AgencyAgency/P5Demos/blob/master/DampenedBall/DampenedBall.pde

Okay, and here are the caveats… there's a lot that could be improved here. But, rather than throw additional programming concepts at you, this solution just uses the three concepts that we've discussed so far:

1. variables (a.k.a. assignment)
2. conditionals (a.k.a. **if-else** statements)
3. loops (a.k.a. the **draw()** loop)



```
// Ball size:
int D = 40;
int R = D/2;

// Ball's starting position:
float x = 30.0;
float y = 80.0;

// Ball's velocity:
float dx = 1.0;
float dy = 1.0;

// Forces on the ball:
float g = 2.0;
float damp = 0.9;

// This happens once, at the beginnning:
void setup() {
  size(400, 400);
}

// This happens once per frame:
void draw() {
  // Cover up the old frames:
  background(0);

  if (x + R > width) {
    // Ball hits right wall:
    dx = -abs(dx);
    dx = damp * dx;

  } else if (x - R < 0) {
    // Ball hits left wall:
    dx = abs(dx);
    dx = damp * dx;
  }

  if (y + R > height) {
    // Ball hits bottom wall:
    dy = -abs(dy);
    dy = damp * dy;

  } else if (y - R < 0) {
    // Ball hits top wall:
    dy = abs(dy);
    dy = damp * dy;
  }

  // Slow due to gravity:
  if (y + R < height) {
    dy = dy + g;
  }

  // Change position w/ velocity:
  x = x + dx;
  y = y + dy;

  // Draw ball in new position:
  ellipse(x, y, D, D);
}
```

Next, the iPad

Next up is the iPad.

We can use what we learned in Processing to do something similar on the iPad.

Also, we can extend this example. The iPad has accelerometers. These are fancy sensors that can detect which way your iPad is oriented in space (how you're holding it). That's how it knows how to rotate the screen, so it's always face up.

We'll use the accelerometer to move the ball around on the screen… so, that your iPad ball will appear to be affected by gravity.