

# Bouncy Ball Coding Tutorials

Wojciech Golaszewski   Cosmin Vladianu   David Al Mjali   Adris Khan

## Tutorial 1 – Bouncing Ball Physics Simulation

Coding Curriculum Group Project

# Bouncing Ball

1/8 First steps – creating the program



# Bouncing Ball

Every program using p5.js is based on 2 main functions. Function setup, which is always called first the program – this is where the program starts. We use this function only once, to declare all initial condition. You will see in a minute. Function draw – this function is called 60 times every second and is the main function where everything is managed as the program runs. Let's put them to our codes in the first place:

```
function setup()
{

}

function draw()
{

}
```

This is how functions are structured in JavaScript. We write 'function' to declare a function, then name of the function. Name can be whatever we want. I am using setup and draw here because p5.js refers to functions with those names specifically, but when you write you own function – name it however you like. In braces () you put arguments for a function. You will see what that means in a minute. These two don't take any arguments, so we are leaving braces blank.

## First steps – creating the program

1/8

Then we have curly brackets {} where we put what our functions will do. Let's do it! Add this to setup (do not remove the draw function below):

```
function setup()
{
  createCanvas(960, 540);
  background(0);
}
```



createCanvas is a function which takes 2 arguments. It creates a window (a 'canvas') on a webpage of width and height you specify – in this case 960 by 540 pixels, which is 16:9 format. background function takes a value from 0 to 255 – 0 is black and 255 is white. Do you already suspect what we will receive with some values in between? Now save this file and run open index.html file in your browser! You should see a black 960 x 540 pixels rectangle.



### Challenge!

Try putting different values into createCanvas and background functions and refresh the webpage of index file! See what happens for yourself!

# I Bouncing Ball

```
function setup()
{
  createCanvas(960, 540);
  background(0);
}
```

What are those ; semicolons? In JavaScript they are used to mark an end of an operation – here, calling a function. You will see a lot of them in your codes.

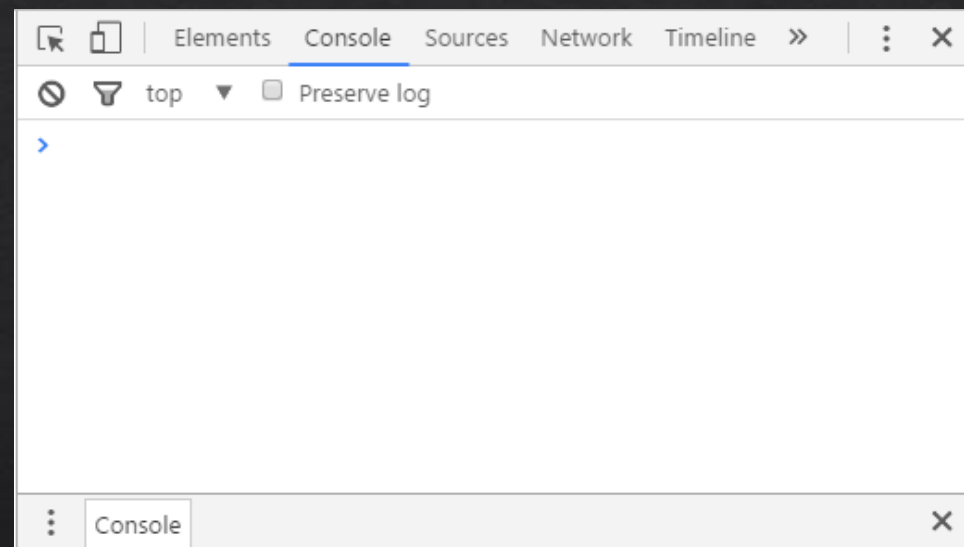
One more thing. Calling doesn't equal to declaring. We declare setup function here – we put curly brackets {} and some stuff between them – we DECLARE what the function is supposed to do.

We CALL createCanvas and background – they are already DECLARED somewhere in p5.js file and we just want to use them – we call them. Some functions take arguments, some don't. These two do. This way we tell them what we want precisely. If a function doesn't take any arguments – it always does the same thing when we call it. We will use such functions later.

## First steps – creating the program

1/8

Let's try something fun. Go to the index file in your browser and try to open a console. In Chrome press F12 and then click on 'Console' tab. This should appear on right side of your screen:



Console will be immensely helpful to you when you program. It is actually essential for every programmer.





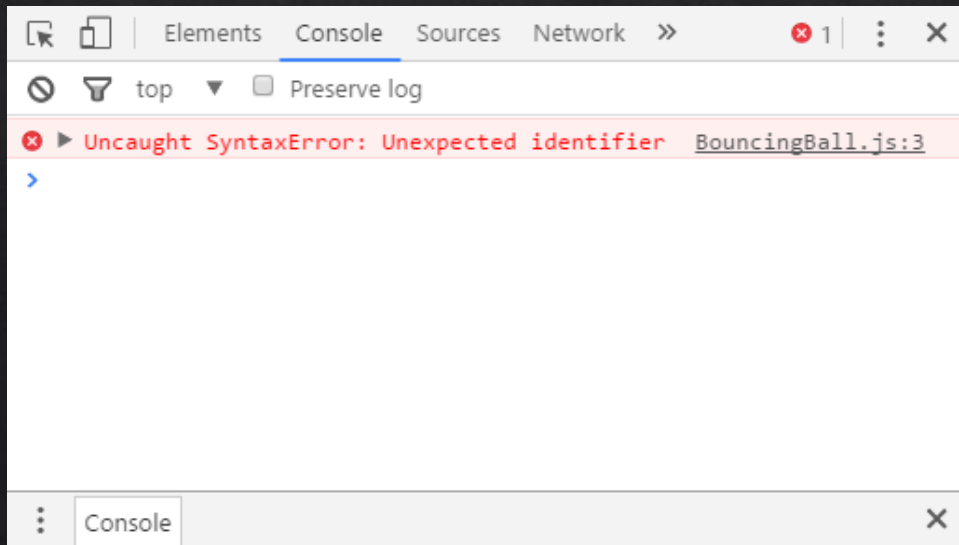
# Bouncing Ball

## First steps – creating the program

1/8

```
function setup()
{
  createCanvas(960, 540);
  background(0);
}
```

Let's make a mistake in our program. Put a random space in `createCanvas` on purpose. Now refresh the page in your browser. You should see this on your console:



`BouncingBall.js:3` means you got an error in line 3 of your `BouncingBall.js` file. It is an unexpected identifier error, which means that Chrome doesn't know what to do with `createCanvas` and `vas` statements.

Now remove the mistake and refresh the page. The error message disappeared.

Everyone does a lot of errors while programming and such tools like a console are essential for development. Additionally you can display some values from your program on the console. We will show you that in a next chapter.



### Pro Tip!

Beware of forgetting about `;` semicolons. I did that a lot when I started programming. In JavaScript such an error doesn't pop up on a console, so you must be extra careful about this. If something in your program goes wrong and weird things happen, think about semicolons in the first place.



# Bouncing Ball

Ok, now let's draw something. A ball to start with. We are actually going to do some fancy things with it later on. Modify your program to look like this:

```
function setup()
{
  createCanvas(960, 540);
  fill(255);
}

function draw ()
{
  background(0);
  ellipse(width/2, height/2, 30, 30);
}
```



fill is another p5.js function that takes an argument from 0 to 255. fill is used to define a colour used to draw every object in the program until we redefine fill again somewhere else. So if I put it into setup and nowhere else, everything will be drawn with the colour I put into fill. Here I am putting white – 255.

But 256 shades (counting from 0 to 255 gives us 256 values, not 255) of grey are boring, right? Try putting three arguments into fill or background. Like that: fill(10, 200, 60) and refresh index page in your browser!

## First steps – creating the program

1/8

Functions can be declared in such a way that they do different things depending on how many arguments you give them.

Drawing functions in p5.js work like this: with one value passed they will set our drawing colour to a shade of grey with 0 being black and 255 white. When passing 3 values, you set up colours from RGB palette: red, green and blue. fill(255,0,0); will go purely red. fill(255,255,0); gives yellow, fill(255,255,255); gives... try to think what it gives before you verify it in your program ;-).



### Challenge!

Try experimenting with different colours of the ball and background to see how they work. Google out 'RGB colour picker' and see what you can do with it.



Important! Very important! Do not be afraid of experimenting while programming. Experimenting is actually a vital part of learning how to program! Try putting different values and using different functions in various places to see what happens. Experiment as much as you can!



# Bouncing Ball

```
function draw ()  
{  
  background(0);  
  ellipse(width/2, height/2, 30, 30);  
}
```

ellipse() draws an ellipse. It takes four arguments – x axis and y axis coordinates of the centre of the ellipse and its width and height.

Here, width and height variables are declared in p5.js and denote the width and height of our canvas. In our case width equals to 960 and height to 540. To see it for yourself, add this line to your draw or setup function (below createCanvas() function):

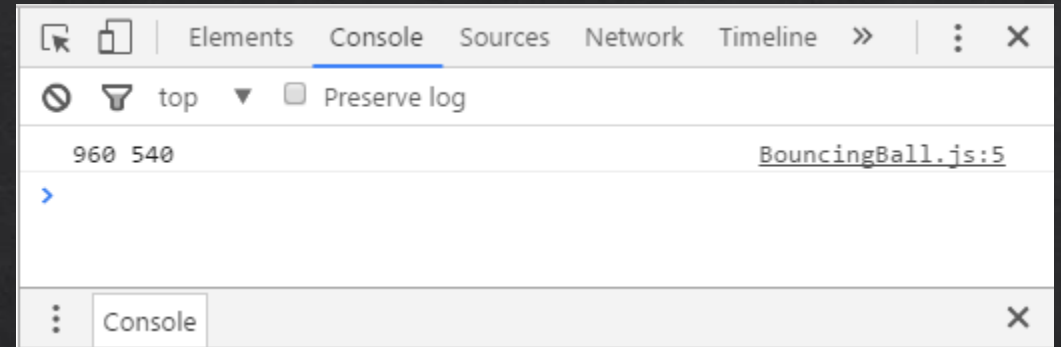
```
console.log(width + ' ' + height);
```

It will write values of width and height and a space between them. If you put this line in setup(), you will see this on the console:

If you put this line in draw(), you will see roughly the same thing, but with a bulb showing a rapidly increasing number – the number of times the console logged the same values. It will increase by exactly 60 every second – this is how frequently draw() is called.

## First steps – creating the program

1/8



Console logged that line 5 printed out '960 540', exactly what we wanted! Mmmm, perfect!

Wait a moment, but why would I do that? I know what width and height are! You will see the use of printing stuff out in the future, when you will create your own variables. Especially if something goes wrong in the program and you don't know what it is – you can print some values on the console and see for yourself if they are what you want them to be. If they are not, it means that you found a mistake you had made. Now you know where it is, which makes eliminating an error from your program (we call them bugs) much easier.





# Bouncing Ball

## First steps – creating the program

1/8

```
function draw ()
{
  background(0);
  ellipse(width/2, height/2, 30, 30);
}
```

I wanted my ball to be drawn in the middle of the canvas so I divided width and height by 2 (first two arguments are the coordinates). I want my ball to be a circle so I tell it to be an ellipse of the same width and height; and I made those 30 pixels.



### Challenge!

Try changing the values in we call ellipse() with. Change the ball's coordinates and shape. For example: make width/2 be width/2+100 to draw the ball 100 pixels to the right away from the canvas's centre.

If you have completed the latest challenge, you have probably noticed something is wrong with the y axis. Indeed, it is inverted. 0 is the top here and height is the bottom. Try drawing the ellipse in 1/3 height of the canvas. Instead of putting height/3 you should rather put height\*2/3. Star \* is multiplication in JavaScript.

To make things more readable and easier to think about, I put things like that:

```
ellipse(width/2, height - height/3, 30, -30);
```

This way I can still see that I wanted 1/3 of height. I just subtract it from height to invert it to normal. So, I think of coordinates as normal, with normal y axis, but whenever it comes to drawing, I subtract the values I want from height to get what I want. If you type this:

```
ellipse(width/2, height - height/2*3, 30, -30);
```

You will get the ball on 2/3 height of the canvas, which you can see right away thanks to the way you called the function.

I also make the second 30 negative. This is also to invert y axis, but it is not about ball's location, but dimensions. Circles are symmetrical so it doesn't matter actually, but we will invert rectangles this way, which we will do in a next part of the course.



# Bouncing Ball

2/8 Creating a Ball



# Bouncing Ball

Now, as we have learned how to use functions, let's create our own! We will use it to manage everything that comes with the ball we are going to make – we will store its radius, coordinates, movement speed, move it around and draw it on the screen. Therefore, what we are actually going to make with this function will be called an object. Let's think of our ball in that way. Our ball has some properties, like speed of movement. We will call those properties attributes. Our ball also complies to some rules, like the ones defining how it is supposed change its movement speed. Those rules are actually going to be described as functions.

To sum up: Every object consists of attributes and functions.

Let's get to work. Write this:

```
function BallObject() {  
  
}
```

I just wrote a function which will represent our ball object, which I called BallObject. In this function I will put all attributes and functions that describe our ball. I know that this may seem confusing. After all, I will use a function to make a so called object.

But don't worry. This is how JavaScript works and it will all clear out for you as we go. Let's fill up our newly created function:

```
function BallObject() {  
    var r = 15;  
    var x = width/2;  
    var y = r;  
}
```

I have created 3 attributes. My attributes are variables, which we declare by writing var. What follows is a name we want to call our variable with and an assigned value. Here I made a var r and assigned 15 to it. r is going to be radius of the ball. I made it 15 pixels. Now I declare variables which will store x and y coordinates of the ball's centre. I want my ball to start in the middle of the screen so I put its x coordinate to width/2, which is the middle. I also want my ball to start on the ground, so I put its y coordinate to be as much above the ground, as the centre of the ball is (mind that x and y are coordinates of the centre of the ball).

# I Bouncing Ball

Creating a ball

2/8

By now, we have defined some properties of our ball. However, our function `BallObject()` doesn't do anything beside defining those properties. It only provides a template. Our ball doesn't yet exist. Let's create it.

```
function setup()
{
  createCanvas(960, 540);
  fill(255);
  ball = new BallObject();
}

function BallObject() {
  var r = 15;
  var x = width/2;
  var y = r;
}

var ball;
function draw ()
{
  background(0);
}
```

Right above `draw` I declared a variable called `ball`. I haven't assigned any value to it yet. I have only declared it. It making up for it in `setup` – I am creating an object based on the template I defined as `BallObject()`. Now our ball exists in a variable I called `ball`.

The thing I just did, the process of creating an object based on a template, is called 'instantiation'. What I did, was creating a new 'instance' of `BallObject()`, hence we use a keyword 'new'.

You may wonder why I have declared variable `ball` outside any function. This is because I want it to be visible by every function in the program. When you declare things outside any function, you say that you declare them 'globally'.

Ok, now let's show our ball on the screen. Let's make a function in `BallObject()` to draw the ball.

```
function BallObject() {
  var r = 15;
  var x = width/2;
  var y = r;

  this.show = function() {
    ellipse(x, height-y, 2*r, 2*r);
  }
}
```



# I Bouncing Ball

Creating a ball

2/8

I declared a function `show()`, which will draw the ball on the screen. I could as well put the `ellipse()` call in the `draw()` function, but to keep my code better organised I will put everything what is related to the ball inside `BallObject()`.

You probably noticed that instead of declaring this function like I did with `object`, I put `this.` in front of it. Is another keyword in JavaScript and we will use it a lot. Look at those:

```
function show () {  
    ellipse(x, height-y, 2*r, 2*r);  
}  
  
this.show = function() {  
    ellipse(x, height-y, 2*r, 2*r);  
}
```

These two functions would do exactly the same thing if you called them. However, the major difference between them is where you can call them from. If I used the first one to declare `show()` inside `BallObject`, I would be able to access it (call it) only from `BallObject`. The rest of the program would simply didn't know that `show()` exists. If you called this function in `draw()`, the program wouldn't know where to look for `show()` and you would end up with an error.

To make `show()` visible for the rest of our program, we must use this keyword. This denotes that we are creating a reference to `show()`. Now we can call it from any other place in our code, simply by telling our program where it should refer to when calling `show()`. We want to call `show()` in the 'ball' object we have just created, so we tell the program to refer to `show()` inside 'ball' object. And we do it like this:

```
function setup()  
{  
    createCanvas(960, 540);  
    fill(255);  
    ball = new BallObject();  
}  
  
function BallObject() {  
    var r = 15;  
    var x = width/2;  
    var y = r;  
  
    this.show = function() {  
        ellipse(x, height-y, 2*r, 2*r);  
    }  
}  
  
var ball;  
function draw ()  
{  
    background(0);  
    ball.show();  
}
```



# Bouncing Ball

3/8 Making the ball move



# Bouncing Ball

Making the ball move!

3/8

Before we continue, I would like you to add one line in the setup function:

```
function setup()
{
  createCanvas(960, 540);
  fill(255);
  noStroke();
  ball = new BallObject();
}
```

`noStroke()` removes contours around anything that we draw. If you refresh the page with your code now, you will see that the ball actually touches the ground. With the contour, which is black by default, ball appeared to be levitating and we would like to fix this :-).

Ok, now let's make the ball move! We are going to need new attributes of the ball to define its speed. Add this line below other 3 variables in `BallObject()`:

```
var yspeed = 5;
```

`yspeed` will define the ball's vertical speed. I have set it to 5, which will be 5 pixels per frame or 300 pixels per second, as `draw()` is called 60 times a second.

Now, right below the newly added line, let's write a new function to make the ball move:

```
this.move = function() {
  y += yspeed;
}
```

At every frame, I will add those 5 pixels of `yspeed` to my current location on y axis. So, if I start at height `r` in the first frame, which is 15, in the second frame I will be at height `r + yspeed`, which is 20. Then 25, 30, 35... and so on. You may already wonder what is going to happen when the ball reaches the top of the page. Save your file, refresh the page and find out!

Yes, it does exit the screen. We will deal with that by making the ball bounce of the walls in the next chapter. For now, your whole code should look like this:





# Bouncing Ball

Making the ball move!

3/8

```
function setup()
{
  createCanvas(960, 540);
  fill(255);
  noStroke();
  ball = new BallObject();
}

function BallObject() {
  var r = 15;
  var x = width/2;
  var y = r;

  var yspeed = 5;

  this.move = function() {
    y += yspeed;
  }

  this.show = function() {
    ellipse(x, height-y, 2*r, 2*r);
  }
}

var ball;
function draw ()
{
  background(0);
  ball.move();
  ball.show();
}
```



Before refreshing the page, always remember to save your file. Otherwise changes will not apply.

It's all great now, but the ball quite quickly exits the screen. We need to do something about it!



# Bouncing Ball

4/8 Adding wall collisions



# Bouncing Ball

To check if the ball is out of the bounds of the screen we have to introduce something called an if statement.

```
this.move = function() {  
    if (y+r+yspeed > height){  
        yspeed = -yspeed;  
    }  
    y += yspeed;  
}
```

Lines of a code within an if statement's curly brackets {} will execute only if a condition given in braces () is true. We can use symbols like <, >, <=, >= or == to compare values of variables. All of them correspond to appropriate mathematical symbols.



Very important! = vs ==

= is a value assignment. It CANNOT be used for comparison, like in maths. Only == is used for comparing equality of two values. It is a very common mistake to put = instead of == in ifs, which I did a lot as well. Watch out for this!

Now, what have I done here. Before I allow the ball to move further, I am checking where it is going to be in the next frame. Here I check if the ball is going to hit the ceiling.

The current position of the top of the ball is  $y + r$  (remember  $y$  represents position of the centre of the ball, so ball's top is  $y + r$ ). In the next frame, its position would be  $y + r + \text{yspeed}$ , because we add  $\text{yspeed}$  to its position every frame. Now we check if the position in the next frame would exit the bounds of the screen on the screen's top i.e. if  $y+r+\text{speed}$  is larger than height.

If this is true, we have to invert  $\text{yspeed}$ , to make the ball go down. To  $\text{yspeed}$ , we assign its negative.

Save the code and refresh the browser. Now the ball will bounce off the ceiling.



Challenge!

Try making the ball bounce off the floor yourself. Be careful how to take ball's  $r$  and  $\text{yspeed}$  into account. Mind that ball is going down now, not up. You will be able to see how we did this if statement in the next chapter.



```
function setup()
{
  createCanvas(960, 540);
  fill(255);
  noStroke();
  ball = new BallObject();
}

function BallObject() {
  var r = 15;
  var x = width/2;
  var y = r;

  var yspeed = 5;

  this.move = function() {

    if (y+r+yspeed > height) yspeed = -yspeed;
    if (y-r+yspeed < 0) yspeed = -yspeed;

    y += yspeed;
  }

  this.show = function() {
    ellipse(x, height-y, 2*r, 2*r);
  }
}

var ball;
function draw ()
{
  background(0);
  ball.move();
  ball.show();
}
```



[Don't show this code, just run it]

# Bouncing Ball

5/8 Going physics - gravity



# Bouncing Ball

Going physics

5/8

```
if (y+r+yspeed > height) yspeed = -yspeed;  
if (y-r+yspeed < 0) yspeed = -yspeed;
```

This is how we did them. I also removed curly brackets {} because if there is only one command within them, curly brackets are not necessary. If there are more things you want to do if a statement is true, you need to put all of them in curly brackets. Now gravity.

Let's make a new variable in BallObject and call it gravity:

```
var gravity = 0.25;
```

I assigned a very small value of 0.25 to gravity. In each frame, we will subtract this value from yspeed, thus simulating gravity. In real life, without taking air resistance etc. into account, a force acting on an object will make it increase its speed. With every period of a fixed time, this speed will change by the same value and the object will accelerate or decelerate. So what I am putting into the code is: in every frame, act with the gravity force pointed downwards – in every frame subtract a fixed value of 0.25 from yspeed. Add this to move() function:

```
yspeed -= gravity;
```

Now increase the initial yspeed from 5 to 12 (var yspeed = 12;), save and refresh and see how the ball behaves! We have gravity.



You have probably noticed that the ball bounces higher and higher. This issue is directly related to the infinite bounce problem, which we will cover in the next chapter.

Now, let's make it look more real life like. Modify our wall bounce if statements in the following way:

```
var coef = 0.8;  
if (y+r+yspeed > height) yspeed = -yspeed*coef;  
if (y-r+yspeed < 0) yspeed = -yspeed*coef;
```

I added a variable called coef. Now instead of simply reversing yspeed at every collision, I decrease it by a certain factor. Every time the ball bounces of a wall, it's speed will be decreased to 80% of its value. After some time, the value will be so small that the ball will be virtually brought to a halt. See how what the code does now!



It is also worth mentioning that if we declared `coef` inside `move()` function, we made it visible only in this function. If you wanted to use the `coef`, which you declared in `move()`, anywhere else in the code, your program wouldn't now it exists. In general, If you declare a variable in a function, it is only visible in this function. Unless you use `this` keyword, which we will cover later.

```
function setup()
{
  createCanvas(960, 540);
  fill(255);
  noStroke();
  ball = new BallObject();
}

function BallObject() {
  var r = 15;
  var x = width/2;
  var y = r;

  var yspeed = 12;

  var gravity = 0.25;

  this.move = function() {


    yspeed -= gravity;

    var coef = 0.8;
    if (y+r+yspeed > height) yspeed = -yspeed*coef;
    if (y-r+yspeed < 0) yspeed = -yspeed*coef;

    y += yspeed;
  }

  this.show = function() {
    ellipse(x, height-y, 2*r, 2*r);
  }
}

var ball;
function draw ()
{
  background(0);
  ball.move();
  ball.show();
}
```



# Bouncing Ball

6/8 Infinite bounce problem



# Bouncing Ball

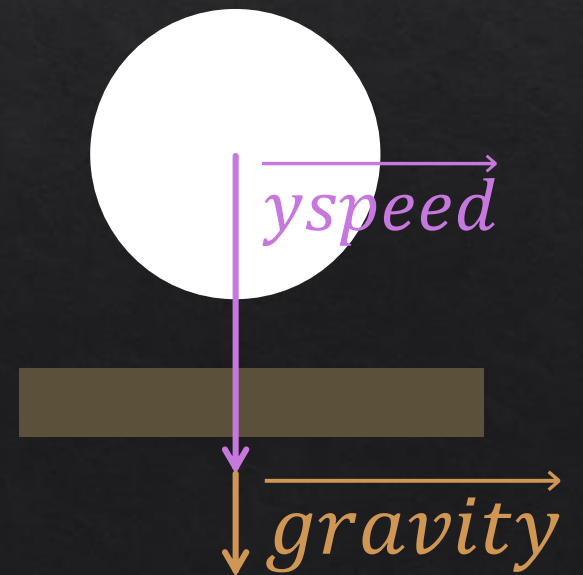
You have probably noticed that although the ball should be stopping, it ends up doing very small bounces. This happens because we don't deal with things that should happen between the frames.

Let me explain. In real life, the ball bounces off the ground in the same way as we simulate it (we assume ideal conditions). But in real life, we have infinitely many frames. Let's have a look what happens when the ball is falling down and is about to hit the ground, but it hasn't yet. We will subtract gravity from yspeed, thus increasing the speed of falling down. What we do right after, is that we state the ball is about to hit the ground and we reverse yspeed. The bounce happened somewhere between the frames. Let's say that if we had infinite frame rate, the bounce would happen in about a half of the time between the current frame and the next one. So, on average, the gravity would be accelerating the ball towards the ground for half of the time and decelerating for another half, after the bounce. Roughly, the influence of the gravity in this period of time would be cancelled out. We do not cancel it in our code, which we will update now:

```
if (y-r+yspeed < 0) yspeed = (yspeed+gravity) * -coef;
```

Now, at every bounce from the floor, we will add gravity back again to yspeed. Because we subtract it right above this line in our code, we now cancel it out. The infinite bounce disappears!

As a disclaimer, we would like to say that the way we are making our simulation is greatly simplified. We do not deal with things that happen between frames even without gravity and we always assume the average case when 'cancelling' gravity at a floor bounce. As this is a beginner programming tutorial and our simulation looks very decent anyway, we decided not to additionally complicate our codes, which are already immensely ambitious for beginners.



# Bouncing Ball

7/8 Adding side movement (x vector)





# Bouncing Ball

Adding side movement (x vector)

7/8

Let's add a new variable to BallObject():

```
var xspeed = 8;
```

And update it in move() just like we do with yspeed:

```
x += xspeed;  
y += yspeed;
```

Now, we need to add side wall collisions.



Challenge!

Like we did with both floor and ceiling collisions, try now implementing side wall collisions yourself! You will be able to see how it's done in the next chapter.

# Bouncing Ball

8/8 Keyboard interaction



# Bouncing Ball

Keyboard interaction

8/8

```
coef = 0.6;  
if ( x-r+xspeed < 0 || x+r+xspeed > width) xspeed *= -coef;
```

This is how we have implemented wall collisions. We changed the coef slightly (you do not have to declare it again with var, change reassign a value), which you can do however you like. In order to write two if statements that do exactly the same thing, I merged them into one, using || or. So, if the ball is about to bounce from the left or right wall, I invert xspeed and multiply it by coef.



Try putting background() to setup() rather than draw() and see what happens!

To sum things up, this is what our BallObject() looks like now:

```
function BallObject() {  
  var r = 15;  
  var x = width/2;  
  var y = r;  
  
  var xspeed = 8;  
  var yspeed = 12;  
  
  var gravity = 0.25;  
  
  this.move = function() {  
  
    yspeed -= gravity;  
  
    var coef = 0.8;  
    if (y+r+yspeed > height) yspeed = -yspeed*coef;  
    if (y-r+yspeed < 0) yspeed = (yspeed+gravity) * -coef;  
  
    coef = 0.6;  
    if ( x-r+xspeed < 0 || x+r+xspeed > width) xspeed *= -coef;  
  
    y += yspeed;  
    x += xspeed;  
  }  
  
  this.show = function() {  
    ellipse(x, height-y, 2*r, 2*r);  
  }  
}
```



# Bouncing Ball

## Keyboard interaction

8/8

At the final step in this tutorial, we will add some keyboard interaction to make the ball jump upwards and sideways. Let's add this line of code right above our wall collision ifs.

```
if (keyIsDown("W".charCodeAt(0)) || mouseIsPressed) yspeed=6;
```

Every key on a keyboard is assigned a code, which is a number. p5.js's function `keyIsDown()` takes a number as argument. We want to extract this code from the letter W, so we tell the function `charCodeAt()` to extract the code from the first letter of a word "W". Because programming languages count from 0 not from 1, we tell it to extract the zeroth letter, rather than first. I know this sounds complicated, but this is how most programming languages work.

For example, if you want to control the ball with spacebar instead of W, simply write: `keyIsDown(32)`. If you want to check codes for other keys, google JavaScript keycodes or go to this website: <http://keycode.info/>.

What I have done here is that if W is or left mouse pressed I change the ball's speed to 6. If you hold the key, this if statement will assign 6 to `yspeed` at each frame when it is pressed.

If you want to take action only once upon pressing the key, make two new variables in `BallObject()` like this:

```
var dp=0, ap=0;
```

and add these to `move()`:

```
if (keyIsDown("D".charCodeAt(0)))  
{  
  if (dp==0) {xspeed+=3; dp=1;}  
}  
else dp=0;  
  
if (keyIsDown("A".charCodeAt(0)))  
{  
  if (ap==0) {xspeed-=3; ap=1;}  
}  
else ap=0;
```

Now we increase the `xspeed` by 3 when D is pressed and decrease it by 3 when A is pressed. Whenever I press them I assign 1 to `dp/ap`. When the key is released I assign 0 inside the else statement. An else statement is executed only when the statement inside if is false. If `dp/ap` is 0, we know that the key has been released, so we can use it again. If `dp/ap` is 1, we know that a key is still being pressed, so we cannot use it. This way, the key will work only at a press and you will have to release it to use it again.





# Bouncing Ball

Congratulations! You have now reached the end of the first tutorial! We must admit, that the content of it is very ambitious for a beginner, so if you felt comfortable with everything here, you can be really proud of yourself. We hope that we explained everything clearly to you and that you have enjoyed coding some fancy stuff with us.

On the top of that, try to experiment with the code you have written! Do whatever you like, what we have shown here is just an example of a program, which we aimed to make as diverse and fancy as possible.

Now, you can have a look on two other tutorials of ours, where we will make two simple games – a flappy bird and a maze ball!

Moreover, you can have a look at p5.js website (<https://p5js.org/>), where you can find documentation of p5, many examples and fun stuff. Additionally, there are many great and exhaustive tutorials out there on the Internet and YouTube, both on JavaScript itself and fun stuff you can do with it!

To sum things up, this is what our BallObject() looks like at the end:

```
function BallObject() {
  var r = 15;
  var x = width/2;
  var y = r;

  var xspeed = 8;
  var yspeed = 12;

  var gravity = 0.25;

  var dp=0, ap=0;

  this.move = function() {

    yspeed -= gravity;

    if (keyIsDown("W".charCodeAt(0)) || mouseIsPressed) yspeed=6;

    if (keyIsDown("D".charCodeAt(0)))
    {
      if (dp==0) {xspeed+=3; dp=1;}
    }
    else dp=0;

    if (keyIsDown("A".charCodeAt(0)))
    {
      if (ap==0) {xspeed-=3; ap=1;}
    }
    else ap=0;

    var coef = 0.8;
    if (y+r+yspeed > height) yspeed = -yspeed*coef;
    if (y-r+yspeed < 0) yspeed = (yspeed+gravity) *-coef;

    coef = 0.6;
    if ( x-r+xspeed < 0 || x+r+xspeed > width) xspeed *= -coef;

    y += yspeed;
    x += xspeed;
  }

  this.show = function() {
    ellipse(x, height-y, 2*r, 2*r);
  }
}
```