# More Exercises: Classes and Objects

# Creating a Simple Pong Game Using Processing

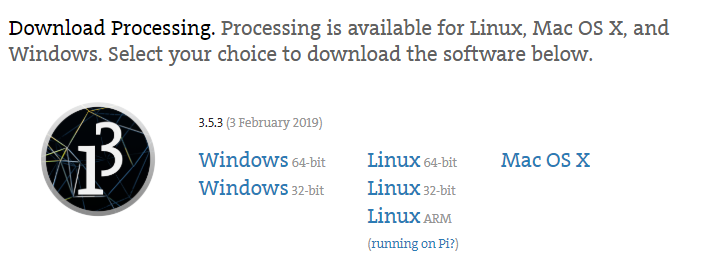
## Processing

### About Processing

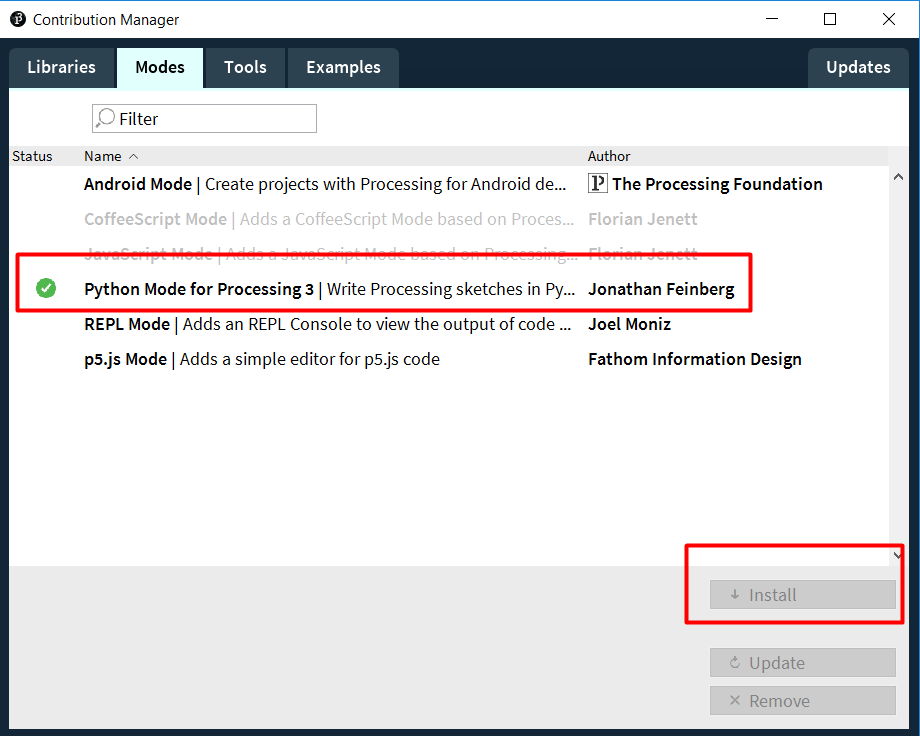
Processing is a **flexible** software **sketchbook** and a language for learning how to code within the context of the **visual arts**. In this exercise we are going to use **Processing** to create a simple **two-player** game of **Pong**

### Processing Installation Guide

Go to <https://processing.org/download/> and choose a download link according to your operation system



Extract the **zip** file and open **processing.exe.** Install the **python mode** for processing by clicking the **dropdown** on the upper right corner and choosing the **"Add Mode"** option



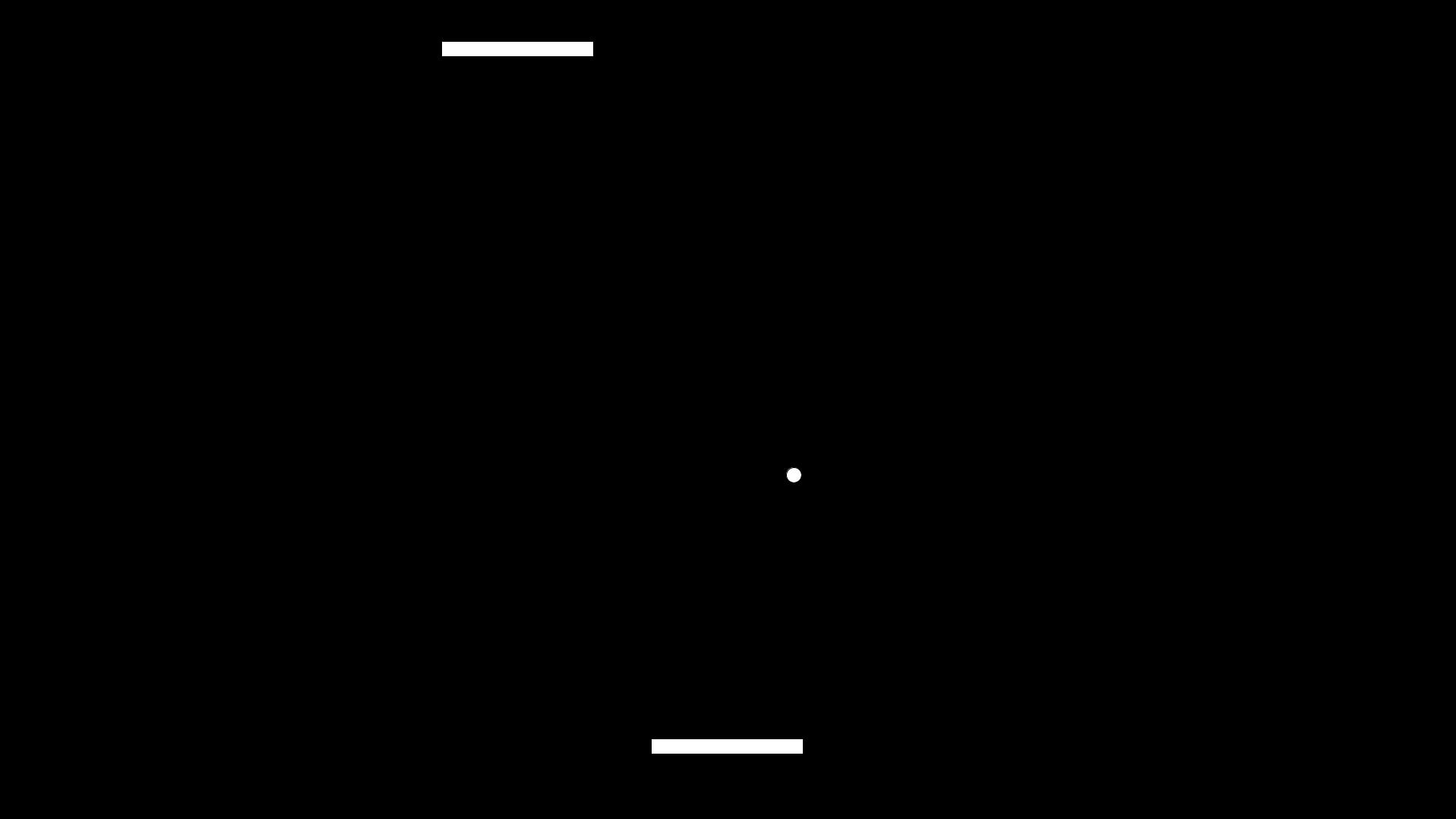
Now we are ready to open the **skeleton** and start **implementing** the missing pieces of the game

### Processing Reference

On this link <https://py.processing.org/reference/> you can refer to all the **build-in** **methods** in processing that can be used.

## The Final Result

After implementing all the missing logic in our project the game should look like this:

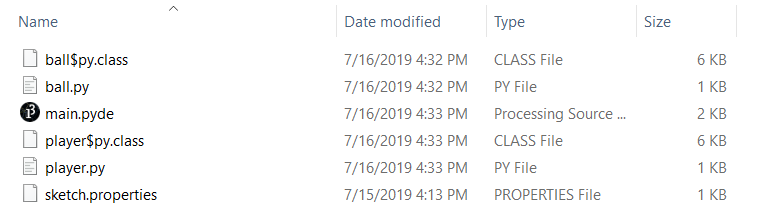




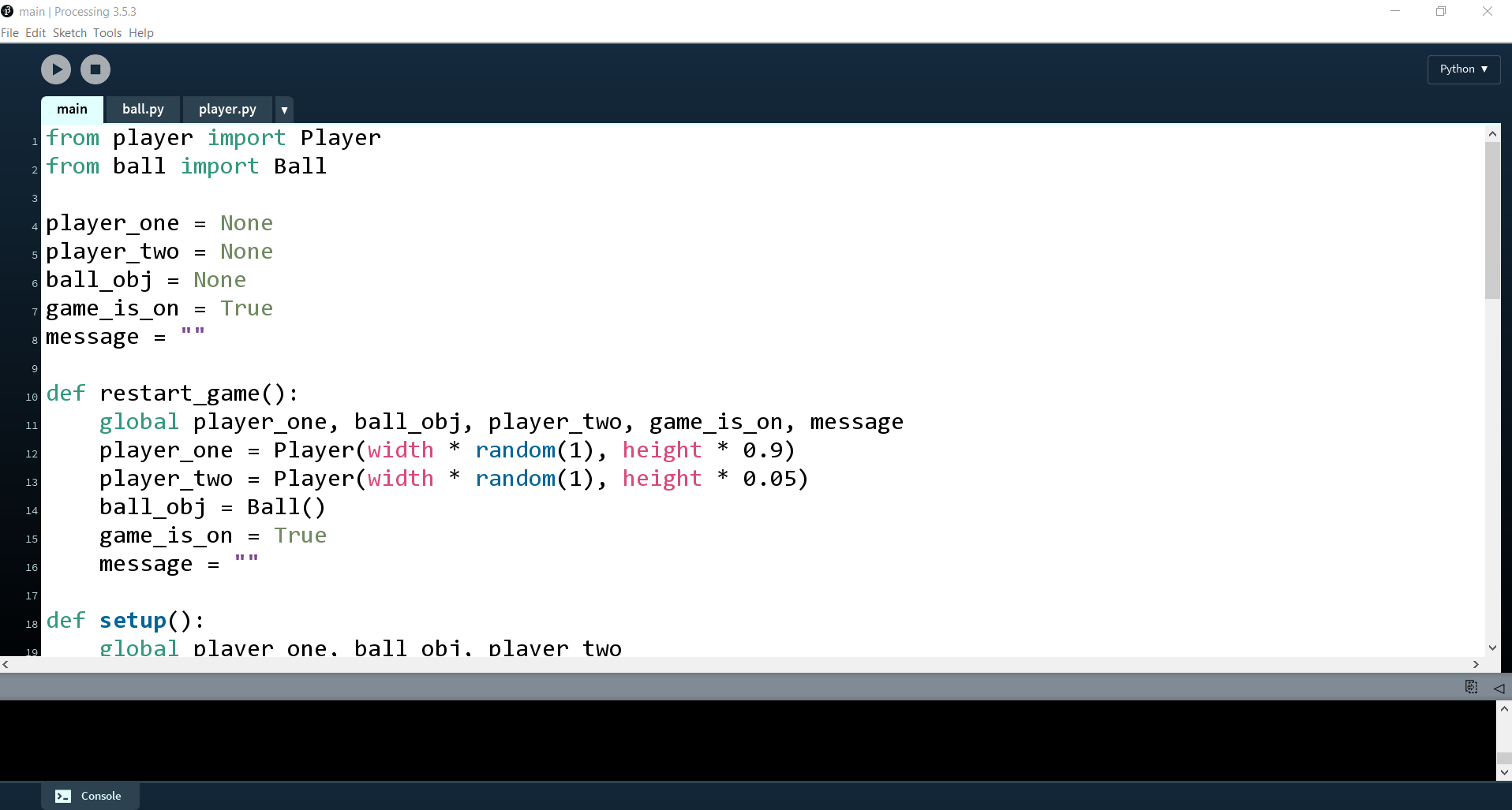
* The **first player** will use the **arrow keys** to move
* The **second player** will use the **"A"** and **"D"** keys to move
* After one of them wins, a **message** is shown and the game can be **reset** by clicking the **"SPACE"** key on the keyboard

## Exploring the Skeleton

You will be provided with a **skeleton**. It will contain a folder named **"main"**. In that folder you will see the following files:



To open the project **double-click** on the **"main.pyde"** file. Don't worry about the other files, they will open automatically in the editor.



### Explaining the Written Code

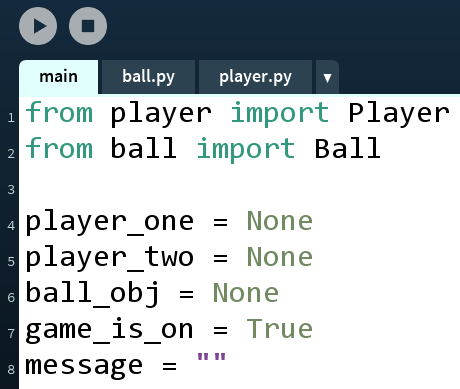
As you see, we have almost **90%** of the code already implemented. So let us explain what it does



* Here you can see the **3 files** we are going to work with
* The **main** file will contain the **main logic** of the game
* The **ball.py** file will contain the **Ball class** and all of its methods
* The **player.py** file will contain the **Player class** and all of its methods

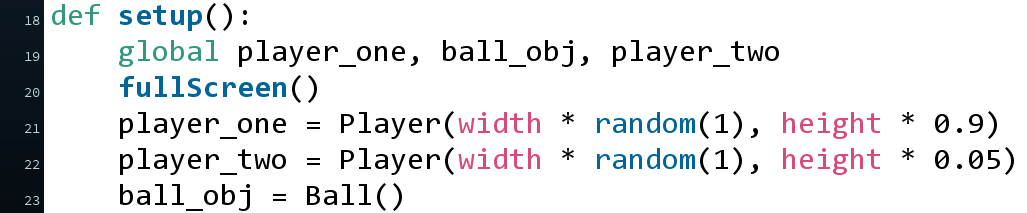
#### The main File

So let us start with the main file



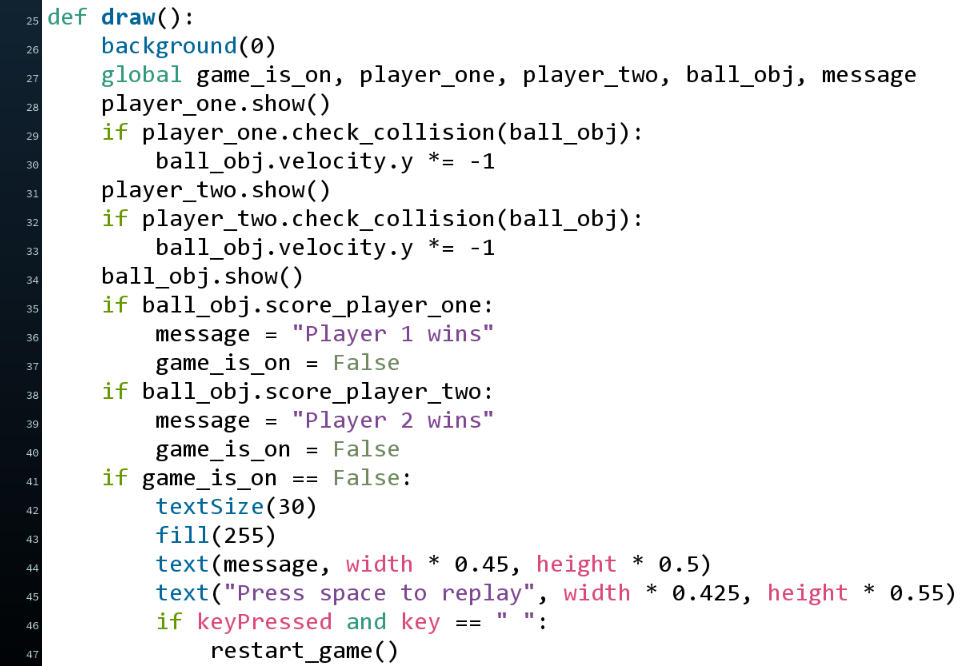
* Here on the top we **import the classes**, since they are in **other files**
* The next several lines are declarations of all the **global variables** we are going to need
  + We have the **two players**
  + We have the **ball object**
  + We have a **Boolean** variable that stores whether the **game is on** or not
  + And finally we have a **message** variable that will store whether the first or the second player is the **winner**

Now let us discuss the **setup()** function



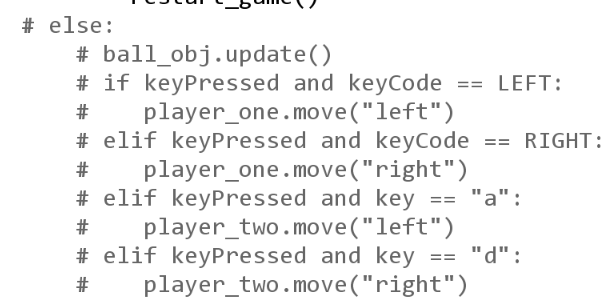
* This is one of the **built-in** methods in **Processing**. It is executed **automatically** after the project is **run** and the execution is only **once**
* Here you can see that we refer to the **global** variables by using the **global** keyword. Since the code in the function is in **different scope**, we have to **tell Python** which variables we **mean to use**
* Then we use the built-in **fullscreen()** function to set the game to use the full screen of our device
* Finally, we create the **two players** and the **ball object**. The Player class needs **x** and **y** parameters upon creation. We will see the Player class in details later. The **Ball class** however needs **no parameters** upon creation of an instance

Next comes the **draw()** function



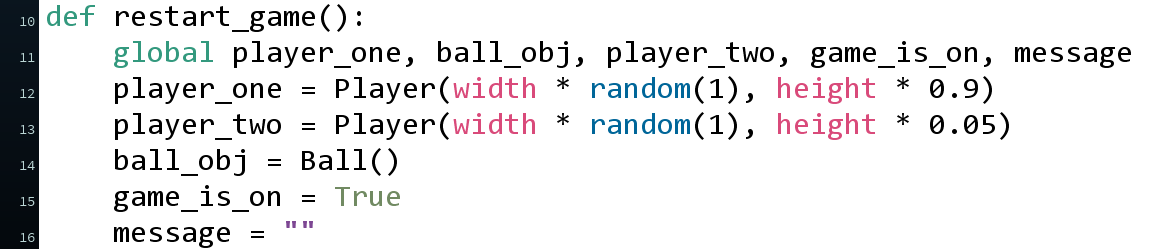
* The **draw()** function is again built-in and it is **called** until the **screen closes** (like **infinite loop**)
* In this section we **display** the **players and the ball**, we **check** of **collision** between the **players** and the **ball** and check if one of the **players wins**. More details about all of the methods in a bit

Now about the section that is **commented out** in the **draw()** function



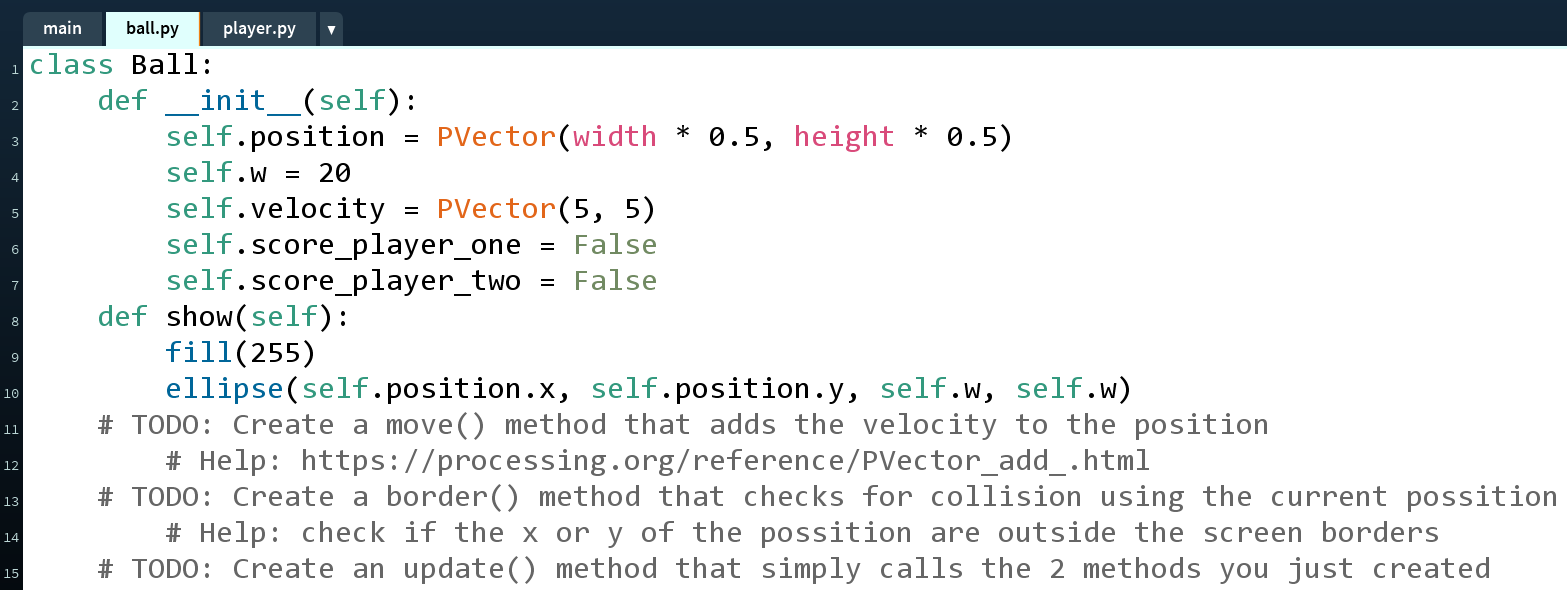
* Here are used methods that **we need to implement** and since we have not done that yet, we commented that section out, so we can **run** the program **without it crashing**
* The logic here **updates** the **ball** state (**movement**) and checks if a keyboard **button is pressed**, so the **players** can **move**

Finally, let us explain the **restart\_game()** method



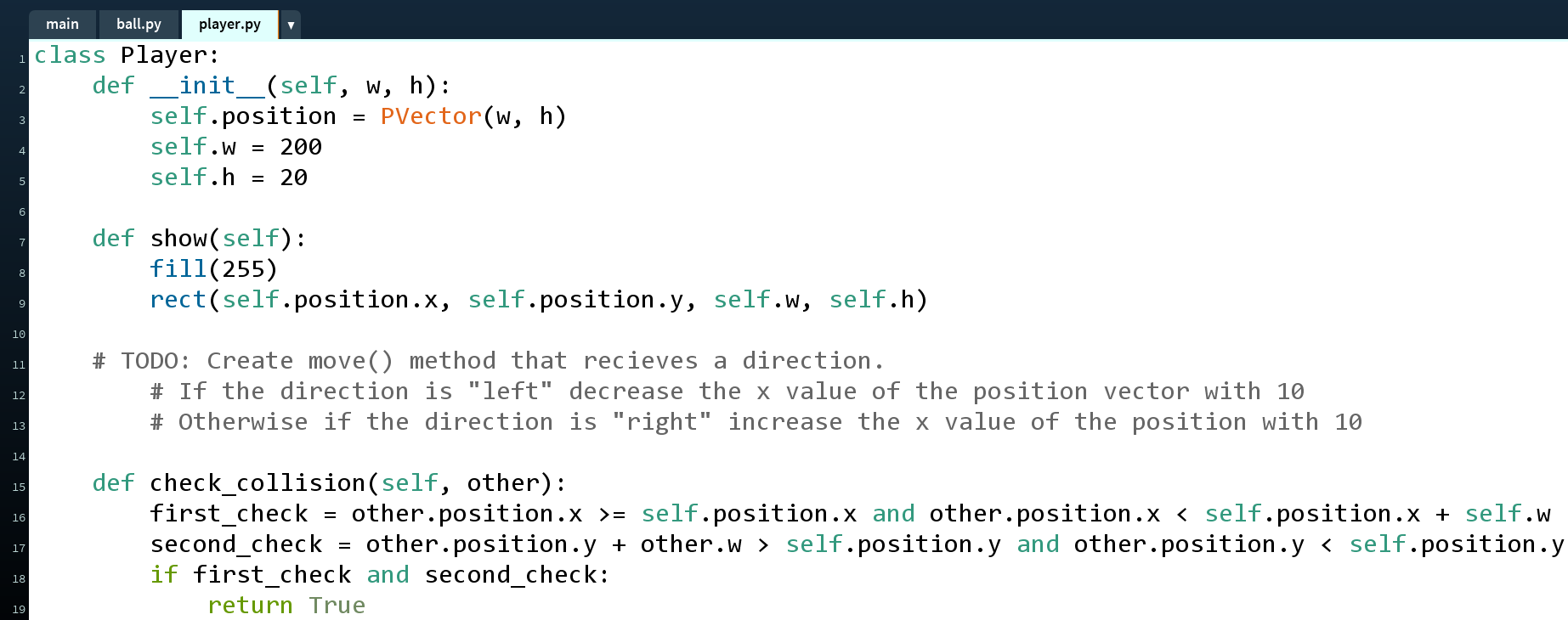
* Here we simply create **new environment** (players and ball), we **reset the message** and set the **game\_is\_on** variable to **True**

#### The ball.py File



* Here you can see that when we create a ball, we have a **position** variable (**PVector** is a way to store **2 values** in **one variable** in Processing - like a list and it also provides is with some **vector math** and **functionalities** that can be useful later)
* We also have a **width** of the ball (w - meaning its **radiu**s)
* The ball will also store whether we have a **winner** (**score\_player\_one** and **score\_player\_two**)
* The **show()** method uses built-in methods to **draw the ball** on the screen (already implemented)
* Finally we have some **TODO's** that we will check out and implement in the next step

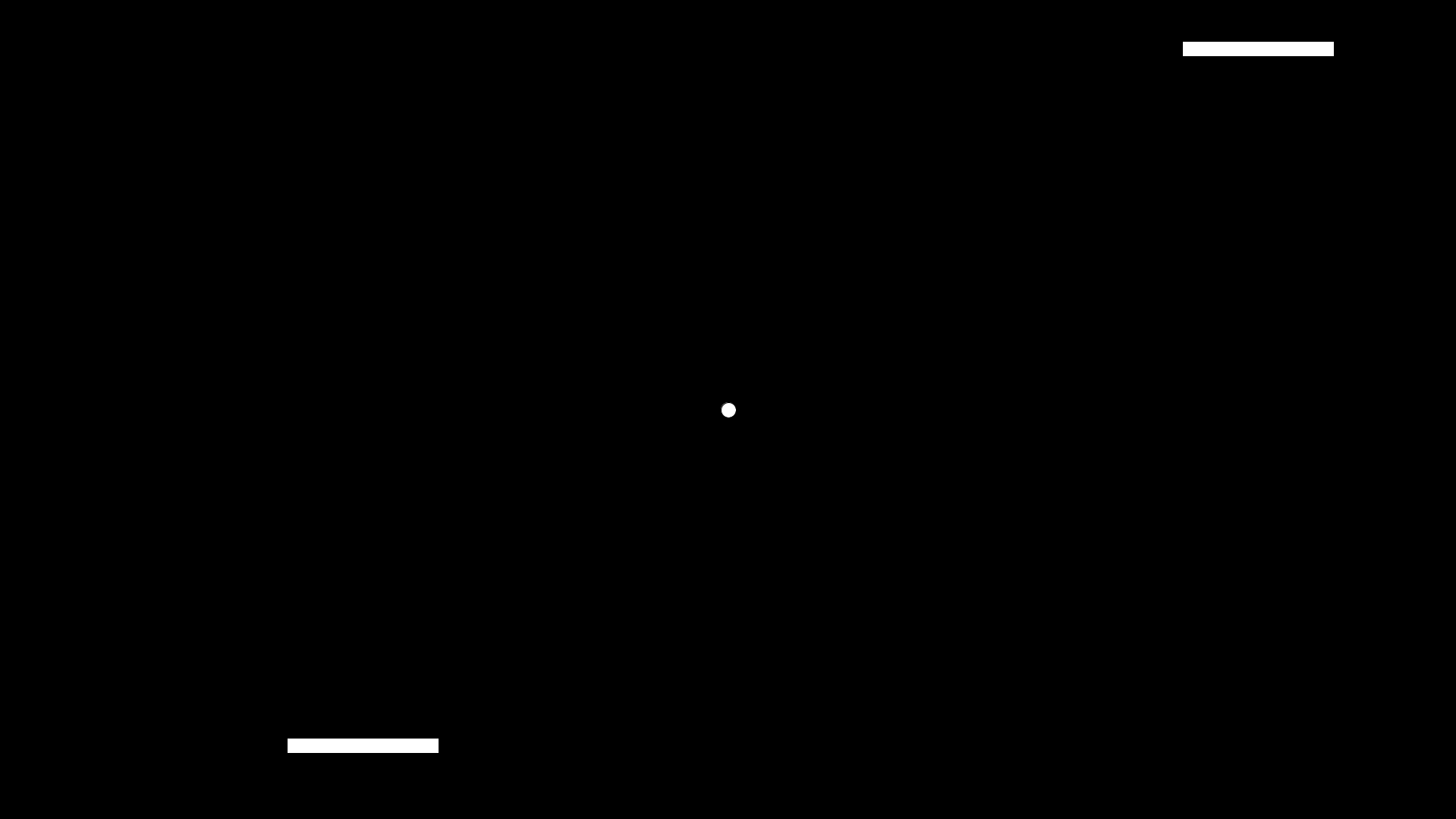
#### The player.py



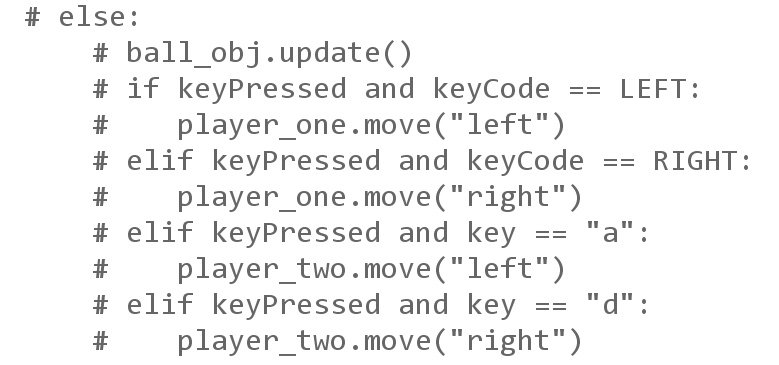
* The player will again have **position** vector (**2 values** for **x** and **y**). The parameters **w** and **h** will be passed to the **\_\_init\_\_()** method to set the initial position of the player
* Then we have the **show()** method that will display the player on the screen as a **rectangle**
* We also have a function that checks if the player **collides** with some other **objects** (in our case it will be the **ball**)
* Finally we have a **TODO** to create the **move** function that will receive a **direction** and depending on that we will **change** the **x** value of the **position** vector

### Running the Game

By pressing the **play button** on the **top left** in the editor, we start the project. A **window** will **open** and you should see something like this:



* The game will **start**, but **noting** will move
* This is because we have **not** yet **implemented** the classes **completely** and as you saw the part the code responsive for the movement is **commented out** in our main file:

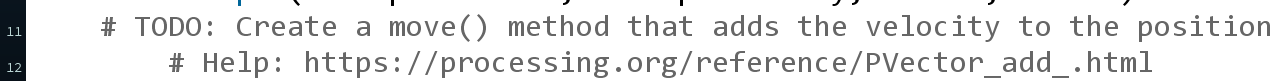


## Implementing the Logic

Now let us proceed to **writing the code**. We will start with the **ball.py** file

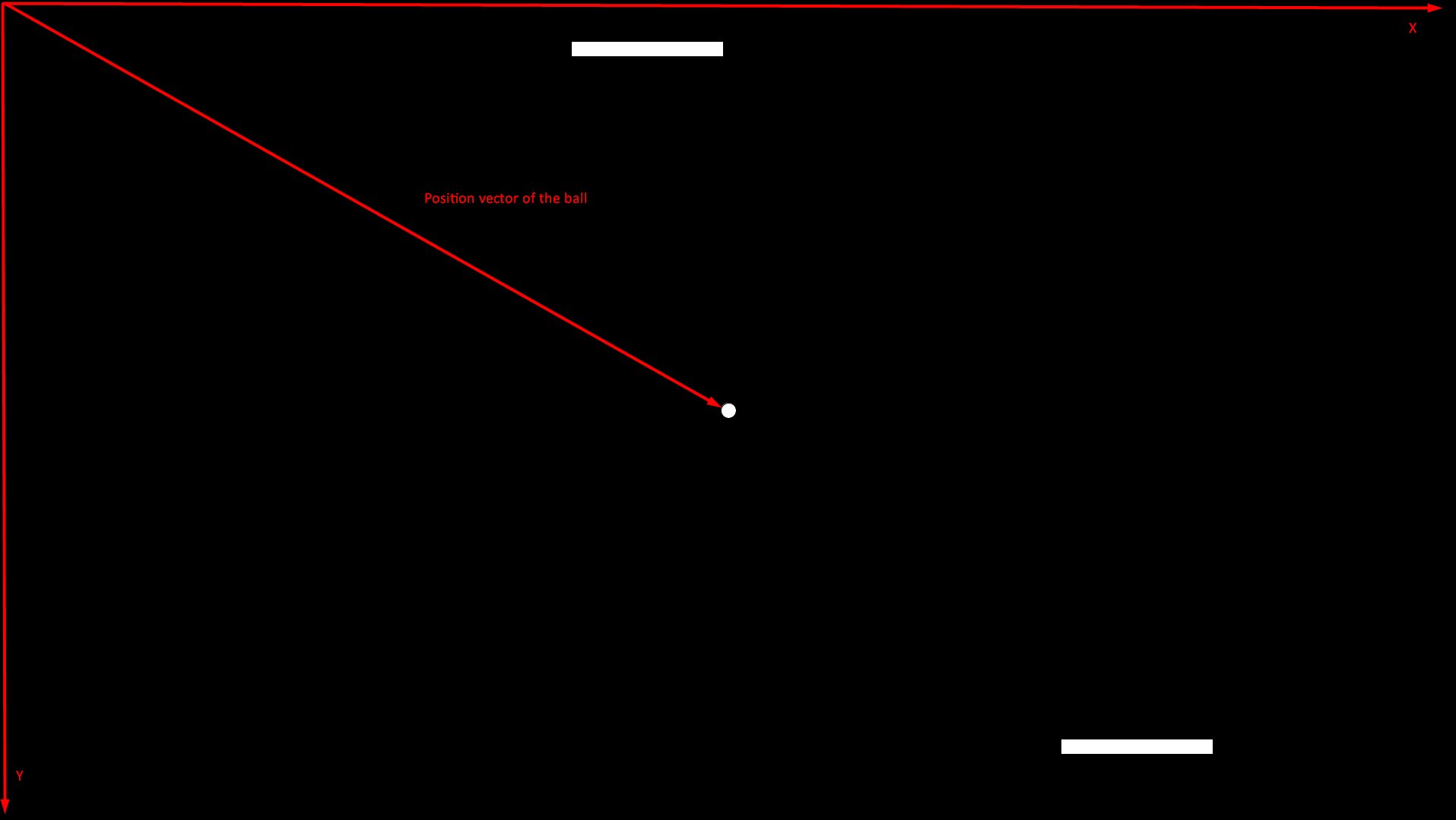
### The ball.py File

So, let us start with the **move()** function



Here, we have to **update** the **position** of the ball **by adding** its **velocit**y. But before we write the code for that, let us **explain** the function of these **two variables** (if you are **familiar with vectors**, you can **skip** this part).

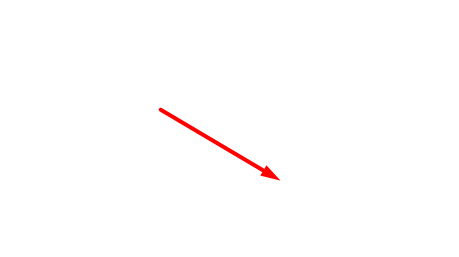
* The **position** is a **vector** and a vector is a **quantity** that has a **direction** and a **magnitude** as determining the **position** of and object relative to another. So let us take the picture below and draw the vector describing the position of the ball



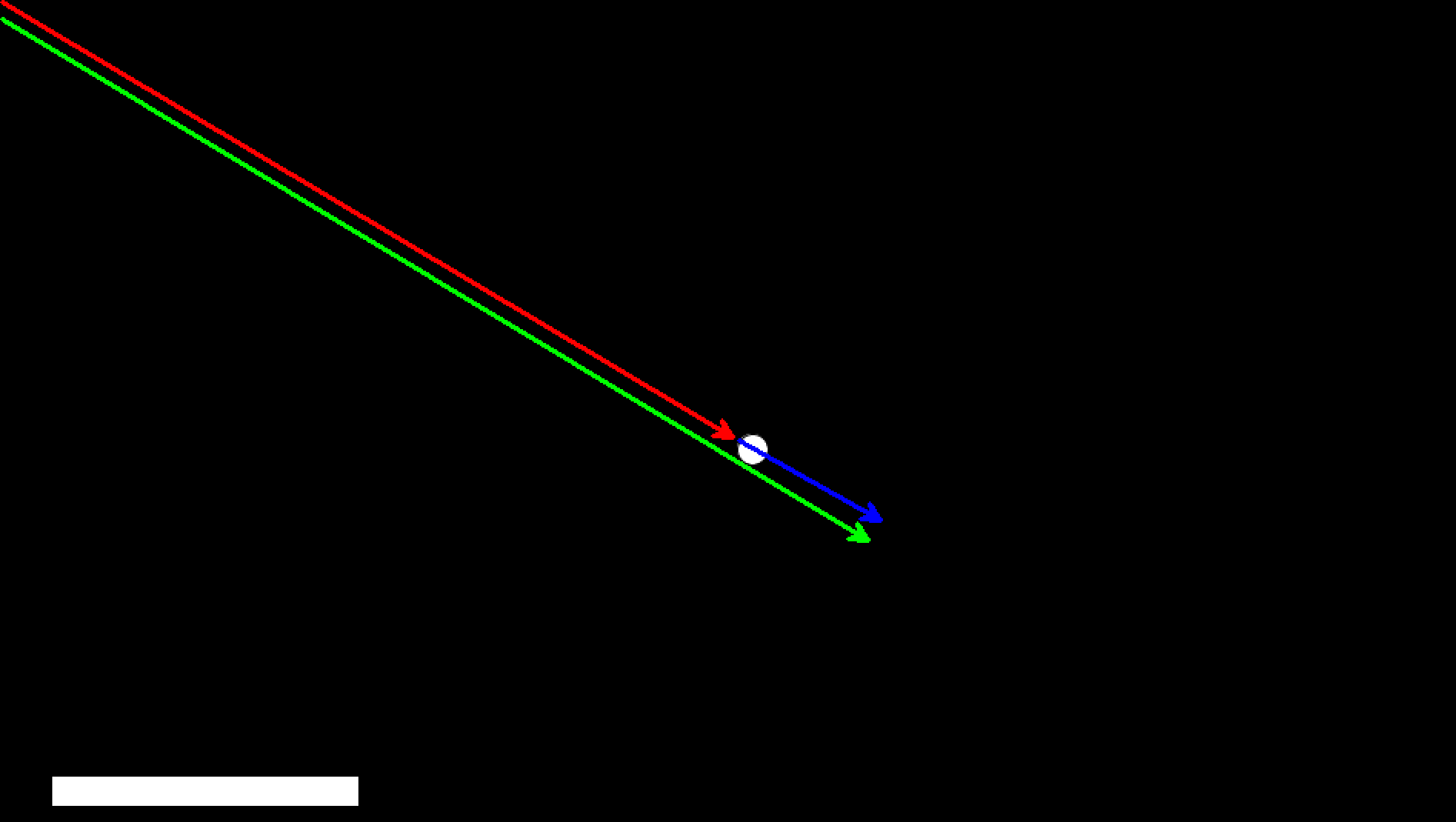
* When programming video games, in most cases the **coordinate system** is **not the same** as in **mathematics**. Here it is so to say **"transposed"** as it is drawn in the picture (the **0-0 point** is in the **upper left corner**)
* The vector describing the position of the ball is relative to the center of the coordinate system and it has **magnitude** (**or length**), **direction** (the **angle** at which it is pointing). By having these two properties, the **position** can be **calculated** and that is where our **ball** is
* The **PVector** in Processing **stores all** of that **info** within itself

And now, what do we mean with **adding** the **velocity** to the **position**

* The velocity is again a **vector** that we use for **speed**. It is the **change** of the position in **x** and **y** at each frame
* So, let us say that we want to change the speed of the ball with **5 units** per frame in both **x** and **y** axes. (the **PVector** velocity will be **PVector(5, 5)**). Our vector will look like this:

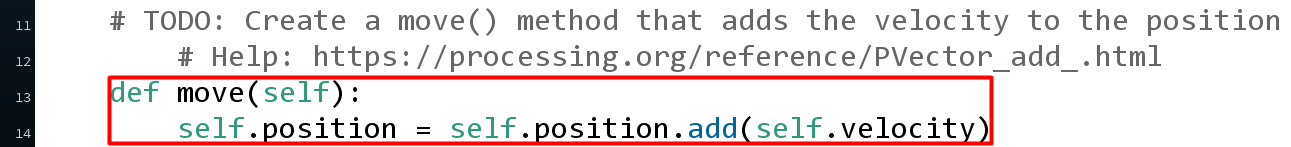


* The way we add two vectors is by putting the start of one of them on top of the end of the other and take the new vector (if they have same angles). So in our case it will look like this:



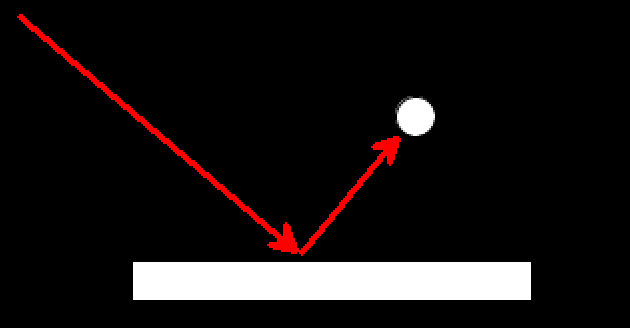
* The **red** arrow is the p**osition**
* The **blue** one is the **velocity**
* When we **add** them we get the green one (the **new position**)

So now, let us implement that in our **move()** method. Good news is that the **PVector** class has a function that **adds two vector**, so our code will look like this:

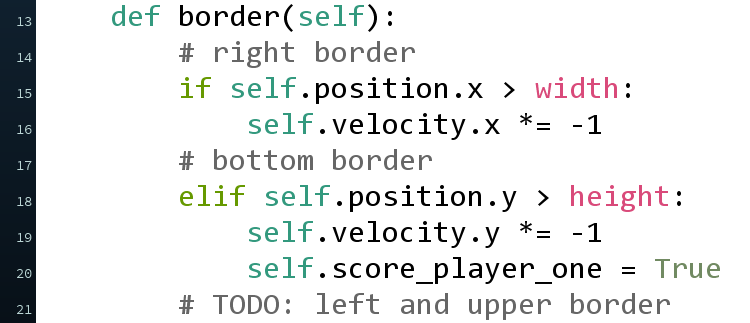


* We set the new position to be equal to the old one + the velocity

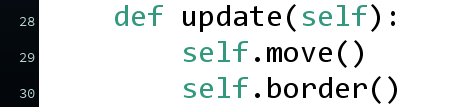
Now, the **border()** method. There we have to check if **x** and **y** of the position of the ball **leave the screen** (be **below zero** or **more than the width**) and **reverse** the **direction** of the **velocity** (**multiply** the corresponding axis by **-1**). Let us first visualize how it looks like for the **ball** **hitting the player**:



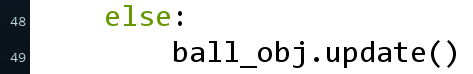
* By multiplying the **y** axis by **-1**, we want to **keep the x** speed but **reverse the y** (**bounciness** effect)
* For the **side walls** we will have to reverse the **x** axis



And now, let us finally create the **update()** method, that just calls the other two



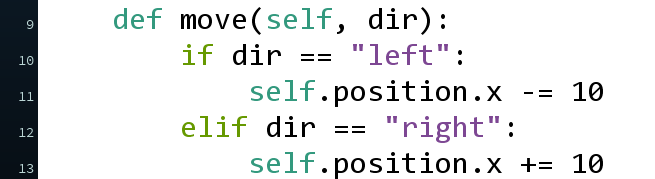
Now, return to the **main** file and **uncomment** the part with the **update of the ball** (since we already implemented all of it)



If you start the game now you can see the ball moving

### The player.py File

In the **player.py** file we only need to implement the **move()** method. It should check if the direction given is **left** or **right** and **change the position** of the player with **10 units**



Now **uncomment** the **rest of the code** in the **main** file and run the game.

