JavaScripterio Documentation

*In this tutorial, we are going to recreate a popular online multiplayer game called slitherio. You can check it out here* [*http://slither.io/*](http://slither.io/)*. After we are done with our game, we are going to have all of the basic functionalities of the real one. We are going to have a player, which will follow the mouse. We will have enemy bots, who will wander around the screen. We are going to be able to eat food, get bigger and longer, we are going to be able to sprint (which will make us smaller with time), we are going to be able to kill enemies, and they are going to be able to kill us. Lots of stuff to do, so let us get started!*

**In order to understand the code we write better, you may want to read about vectors and vector math. Have a** [**look**](https://www.mathsisfun.com/algebra/vectors.html) **at that site for more information.**

# What are we going to use?

## WebStorm

You probably already have it but if you don’t you can download it from here [https://www.jetbrains.com/webstorm/download/#section=windows](https://www.jetbrains.com/webstorm/download/%23section=windows).

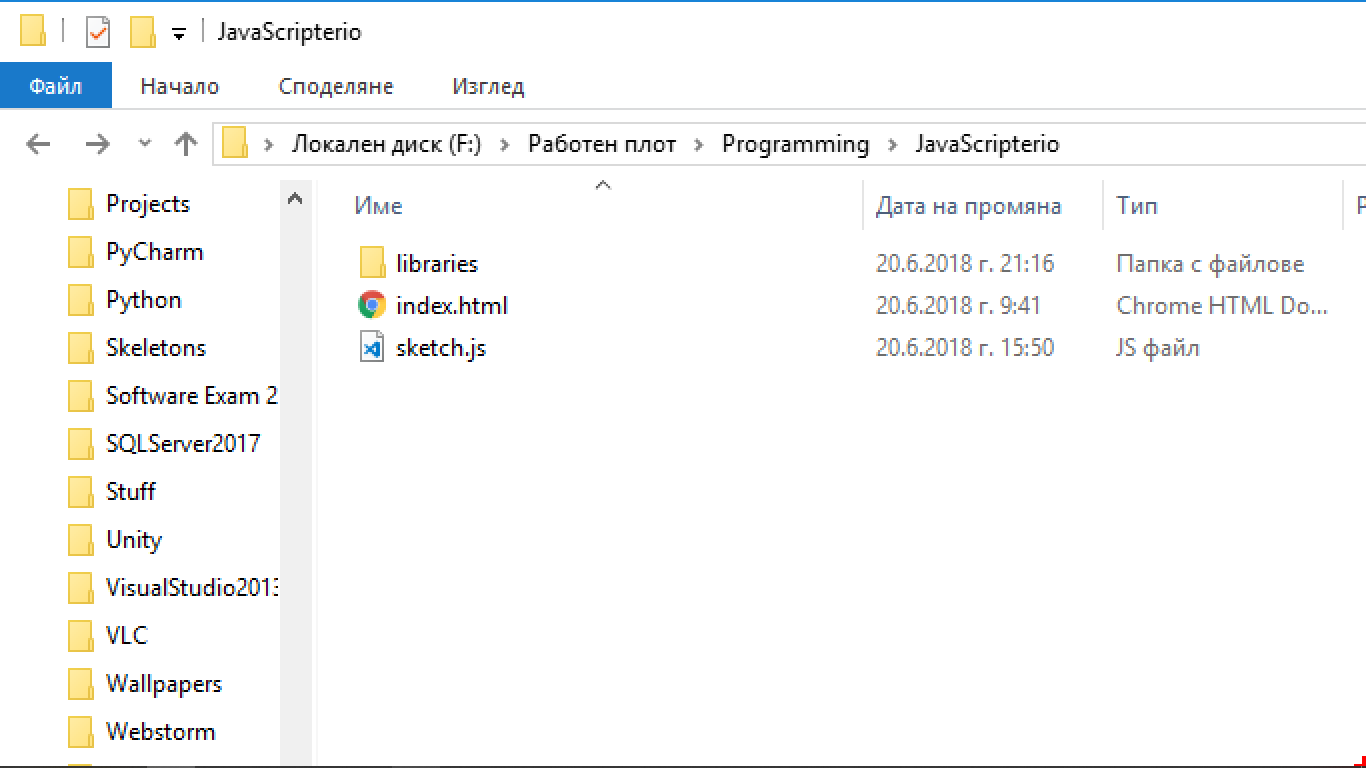
## P5 Library

In order to draw easily we are going to use the p5 library. Unzip the zip-file given to you. It contains all of the files you are going to need. You may want to read the reference for p5 to help you follow along. <https://p5js.org/reference/>

# Setting up the project

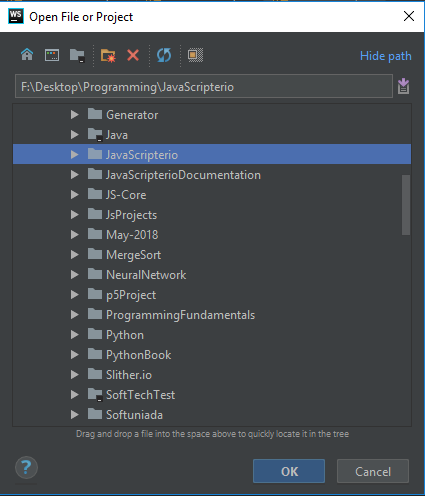
## Creating the folder

First, we need to create the folder that is going to contain all of our code for the game. Choose a directory on your computer and create a folder named **“JavaScripterio”**. After that **unzip** the archive provided to you in that folder. As a result, you should have this folder:

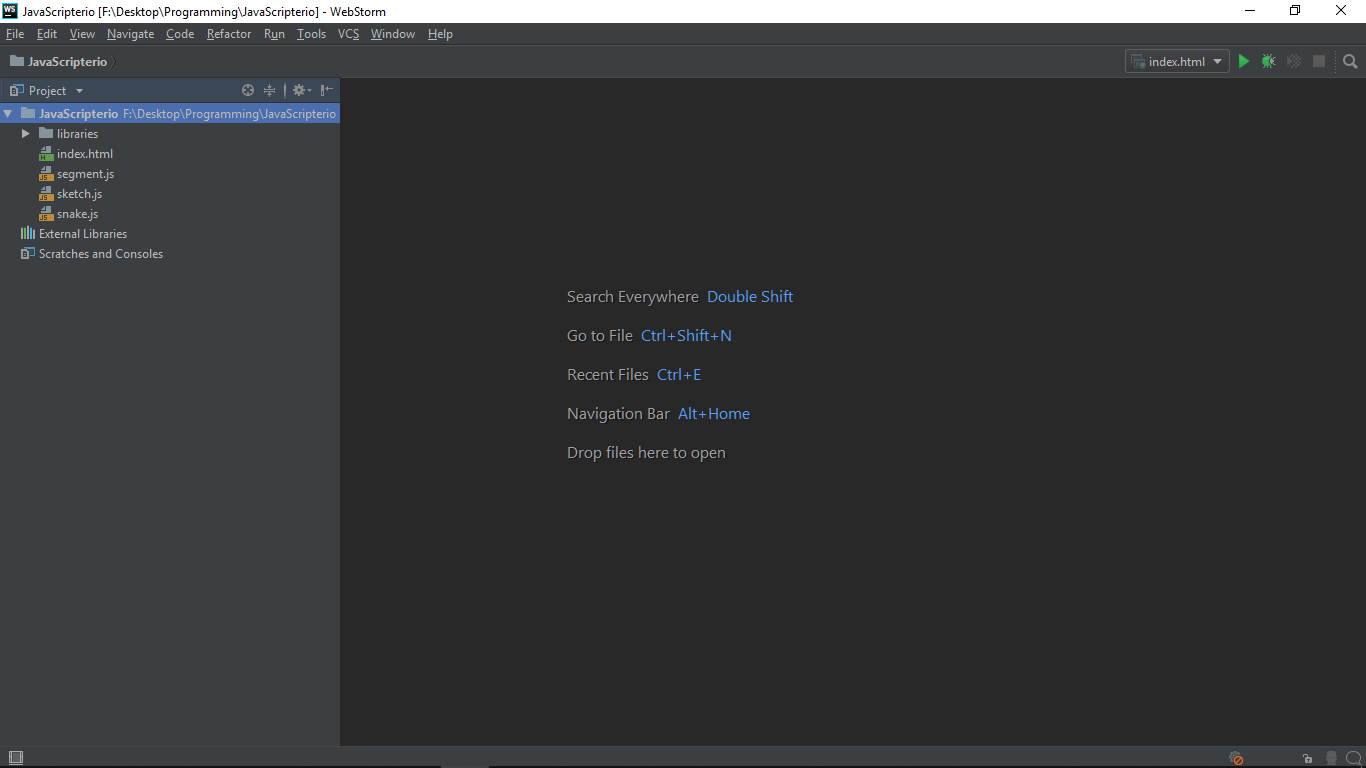


## Open the folder in WebStorm

Open WebStorm and open the folder that we created. You can do that from **“File -> Open”.** Then you find the **“JavaScripterio”** folder and click **“Open Folder”.**



After you do that, you should see the following:



Great! Now we are ready to start coding our game!

# Before we start

## About the sketch.js file

This is our **main JavaScript file**. It is going to contain all of the logic, we are going to use it to draw our game using the setup and draw functions.

## About setup and draw

Those functions are part of the **p5 library**. They will help us build our project.

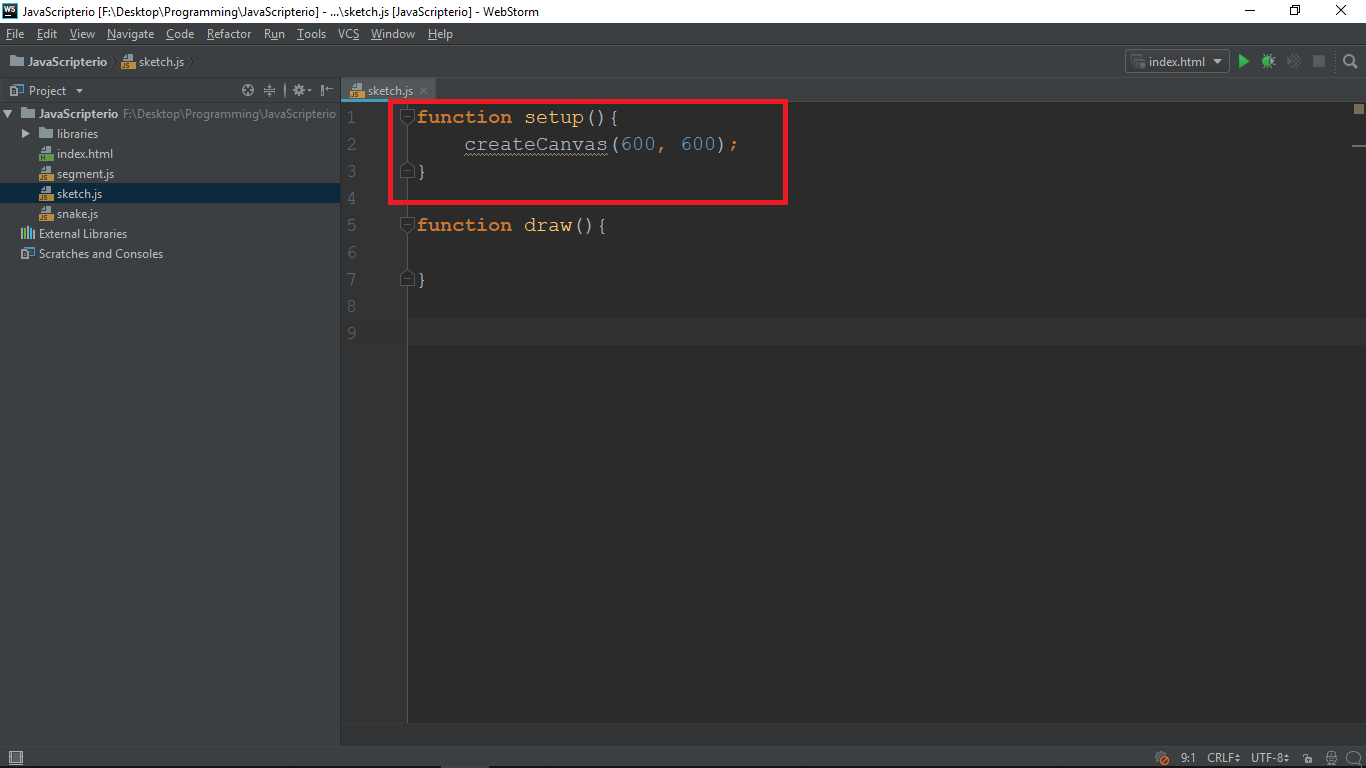
* **The setup() function –** this function is called only **ONCE** when we start our project. This is the place where we create our **canvas** (the place we are going to draw our game). In the setup function, we will also create our **player**, the **enemies** and the **food** (all the stuff we need to create only once).
* **The draw() function –** this function is been called over and over again while our project is running. It is like a **while(true)** loop.

## About the index.html file

This is the file we are going to run in the browser. There we are going to link all the JavaScript files, which hold our code. Right now, it only has the p5 files, sketch file and a little bit of CSS for our canvas.

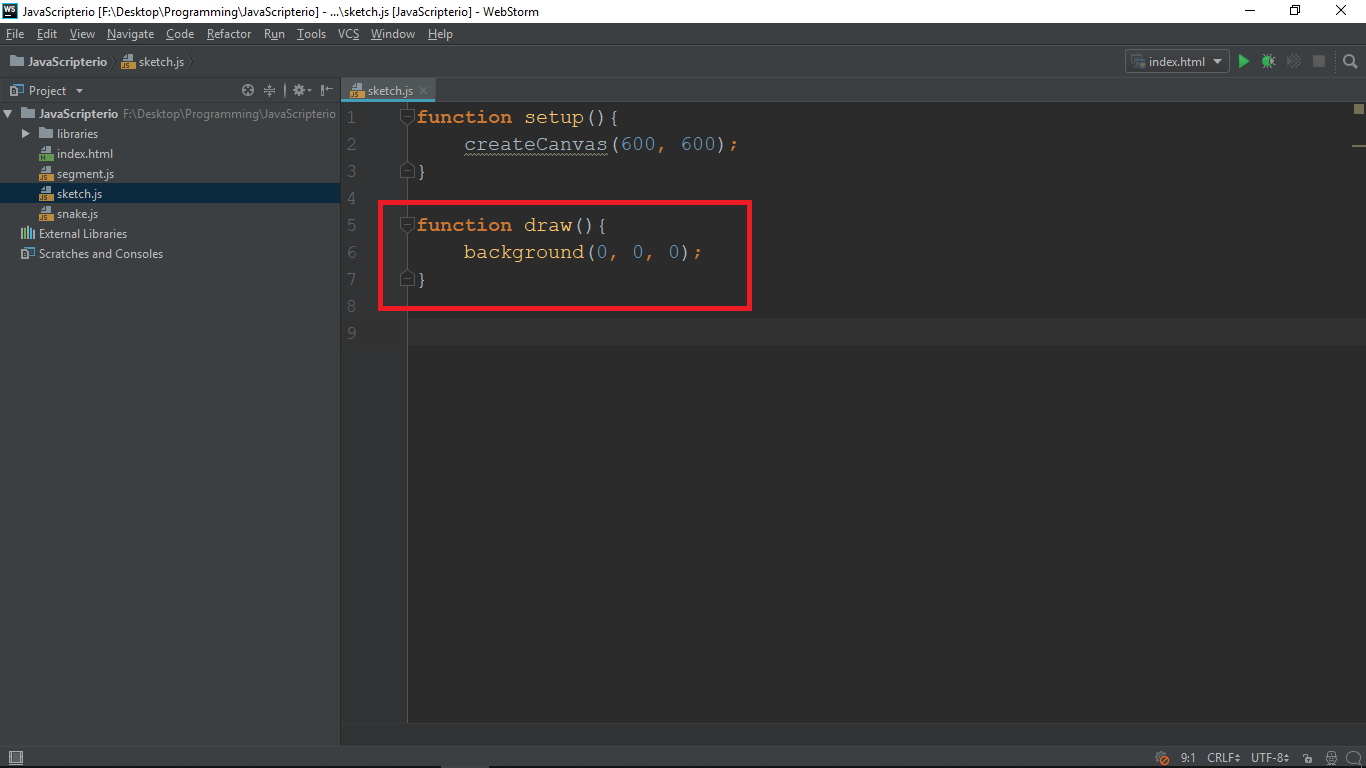
# Creating the canvas

Okay, the first thing we want to do is to create our canvas and set a background. In the **setup function,** write the following:



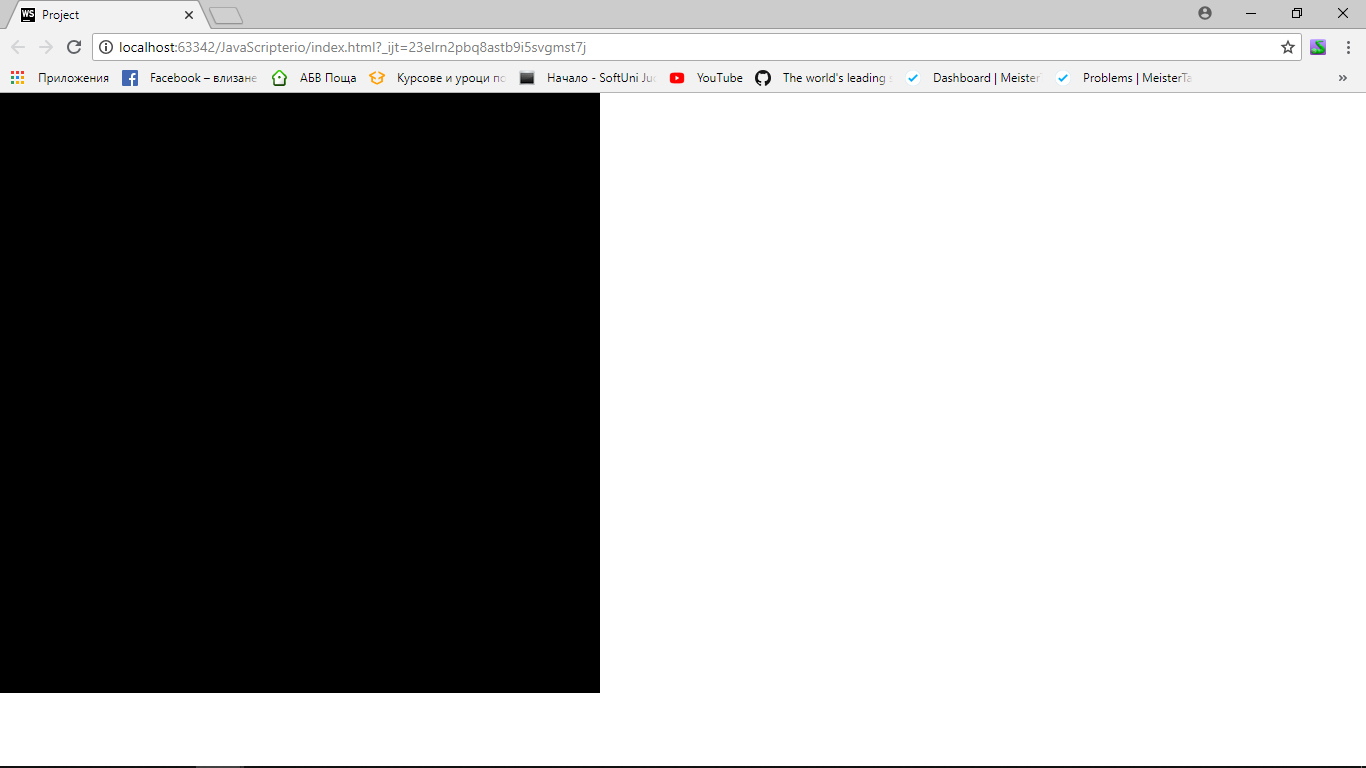
* **createCanvas()** is a function build-in in the p5 library. It requires two parameters: **width** and **height)**. We are going to create a square canvas with **size 600**.

Now if we open our browser we **will not see anything**. That is because we have created our canvas but **we do not draw it**. In order to see it we should **set a background.** In the **draw function**, type the following:



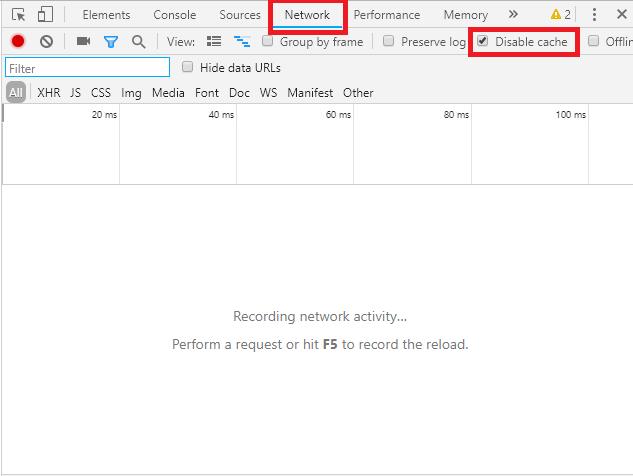
* **background()** is also a build-in function in the p5 library and it requires three parameters: **R**, **B** and **G** values. We want a black background so type three zeros.

Okay, now we can save the changes we made and open the browser. You can save the changes from the menu **(File -> Save All)** or with the keyboard **[Ctrl + S].** Start the game from the **index.html** file.



Congrats! We have our canvas

**Note: you may run into a problem that your changes are not made in the browser. That is because the page is being cached. We suggest opening the dev-tool (with F12 in Chrome) then choose “Network” and check “Disable cache”.**



**Keep the dev-tool opened all the time and you should be okay.**

# Classes we are going to create

In order to run our game we are going to need **four classes**:

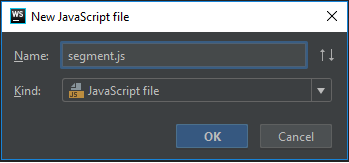
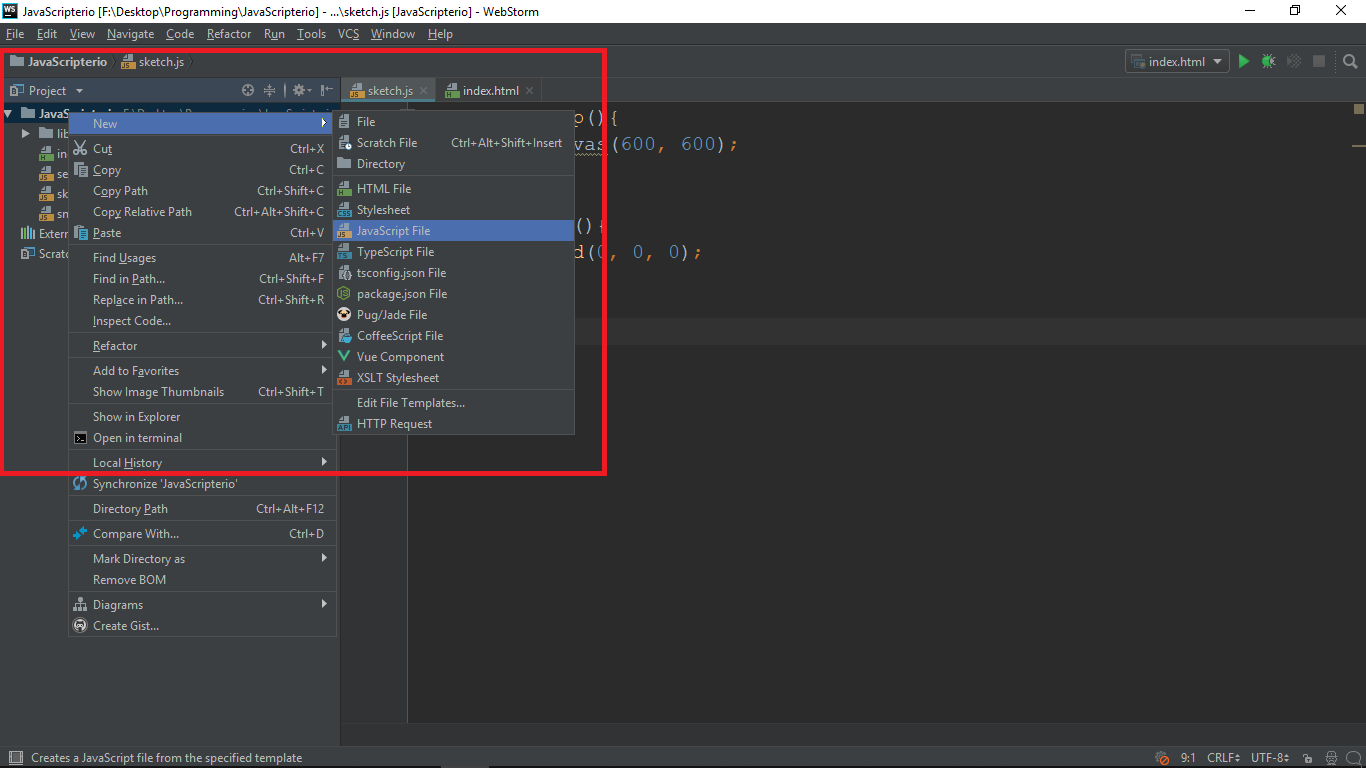
* **Segment –** we are going to use it to construct the snakes. A single snake will have a **head (a single segment)** and a **body (array of segments)**. This will make our snake flexible and it is going to move smoothly.
* **Food –** it is going to store the position of every piece of food, its size and its color
* **Snake –** it is going to have a head, a body and different functions to move, update and display.
* **Controller –** each snake is going to have a controller which is going to determine the direction in which every snake is going to move

More details on each class when we actually create it.

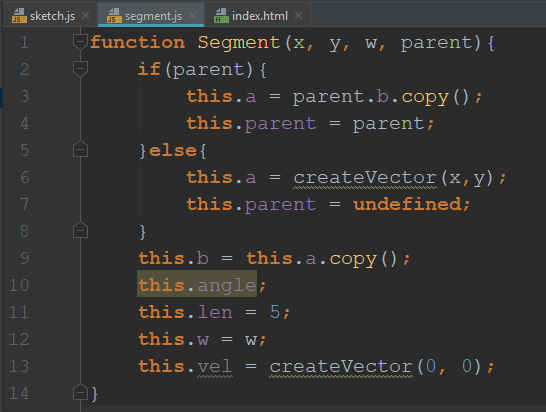
# Creating the segment class

Okay, so let us create the segment class, which we will use to construct the snakes.

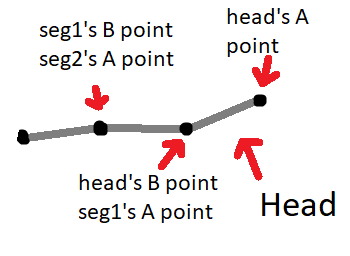
* Right click on the **“JavaScripterio”** folder and create file with name **“segment.js”**



* After creating the file open it and write the following code:

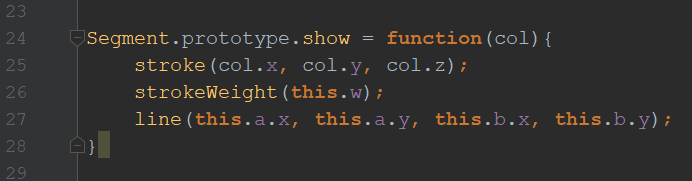


* Our segment is going to receive **three mandatory** and **one optional** parameters. We will always pass it **x, y** and **w** values. The **x** and **y** will store the current **position** of the segment (we are going to update it with each frame later). The **w** will store the **width** of the segment. The **parent** (optional) will determine whether the current segment has a parent or not. We will explain it in details in a bit.
* **this.a** and **this.b –** each segment is going to be a line. A line has **a** point and **b** point. In that way we are going to create a **chain (head, body)** where each segments **a point is connected to the b point of its child**, and the **b point is connected to the a point of the parent segment**. The **head has no parent** (it will **follow the controller**). In addition, the end of the body has no child. Here is an image, which might help you understand it better:



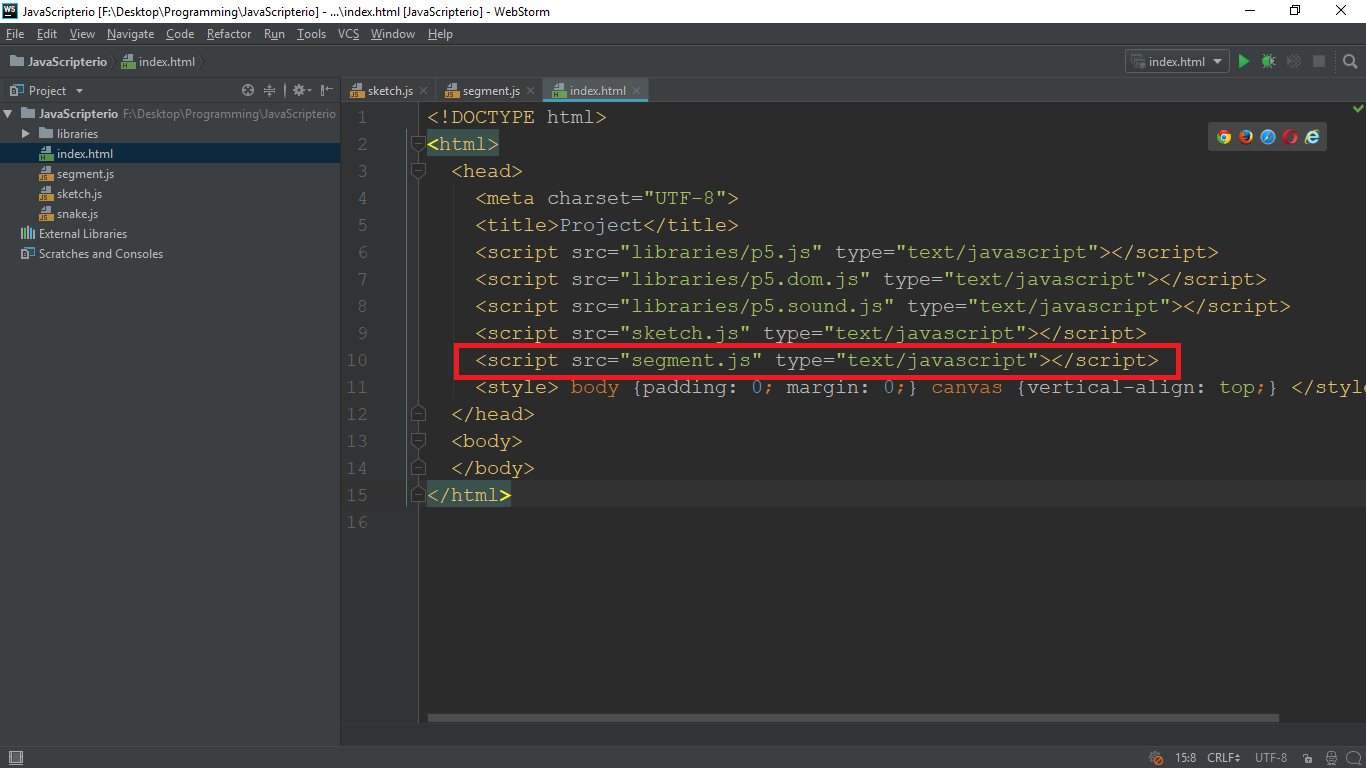
* So **each A point follows the B point of its parent segment** (we will implement that later)
* Each point is going to be a **Vector**. For now just accept that a Vector is a way to store **X** and **Y** value (like a map). If we give a parent to our segment we assign it, if not we make it “undefined”.
* **this.angle** – this we will use to calculate the current rotation of each segment. We just declare it. We will calculate it later
* **this**.**len** and **this.w –** these are the length and the width of each segment. We will use them to draw the segment and make some calculations
* **this.vel** – a velocity is a vector. We use it to store the xSpeed and ySpeed. The speed is zero at the beginning.

Now let us make the function that **displays** the segment and after that, we can go and try creating a segment. Write the following code in **the same file**:

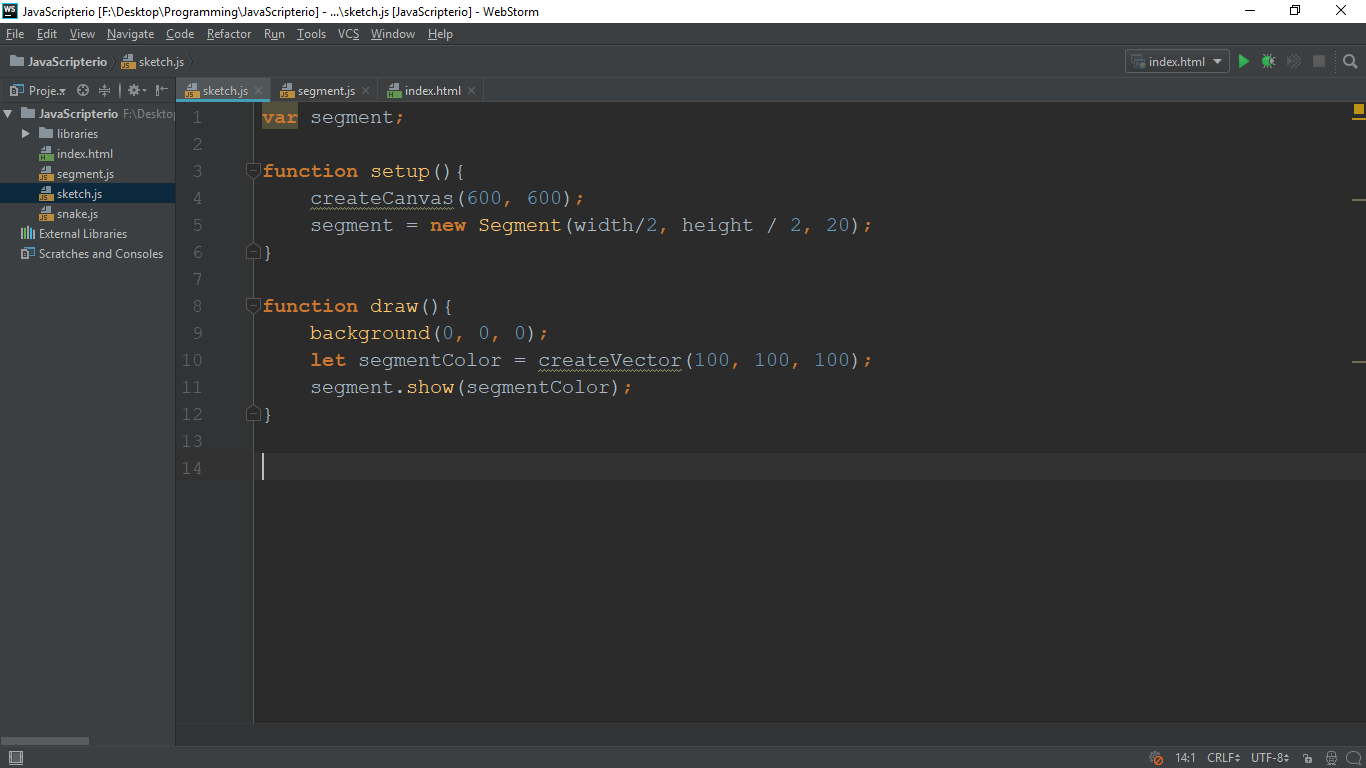


* **Segment.prototype** – by using prototype we say that the function we create belongs to the Segment class
* **col** – that is the color we are going to use to draw the segment (we will pass it in the function when we call it). It will be a **Vector3 (like Vector, but stores x, y, z).** We will use it to store the **R, B and G values**
* **stroke** – this is the color we want our “line” to be. It is a build-in function in p5 and it takes three parameters **(R, B, G values)**
* **strokeWeight** – the thickness of the line (also build-in function). We pass it the **width of our segment.**
* **line** – build-in function that has to receive coordinates of two points (x1, y1, x2, y2). We want to draw a line from the **A** to the **B** point, so we pass their coordinates

**Before we go to the sketch and create our first segment**, we have to **add the “segment.js” file** in the **index.html**, so it knows it has to use it. **Go to the “index.html”** and add a new script:

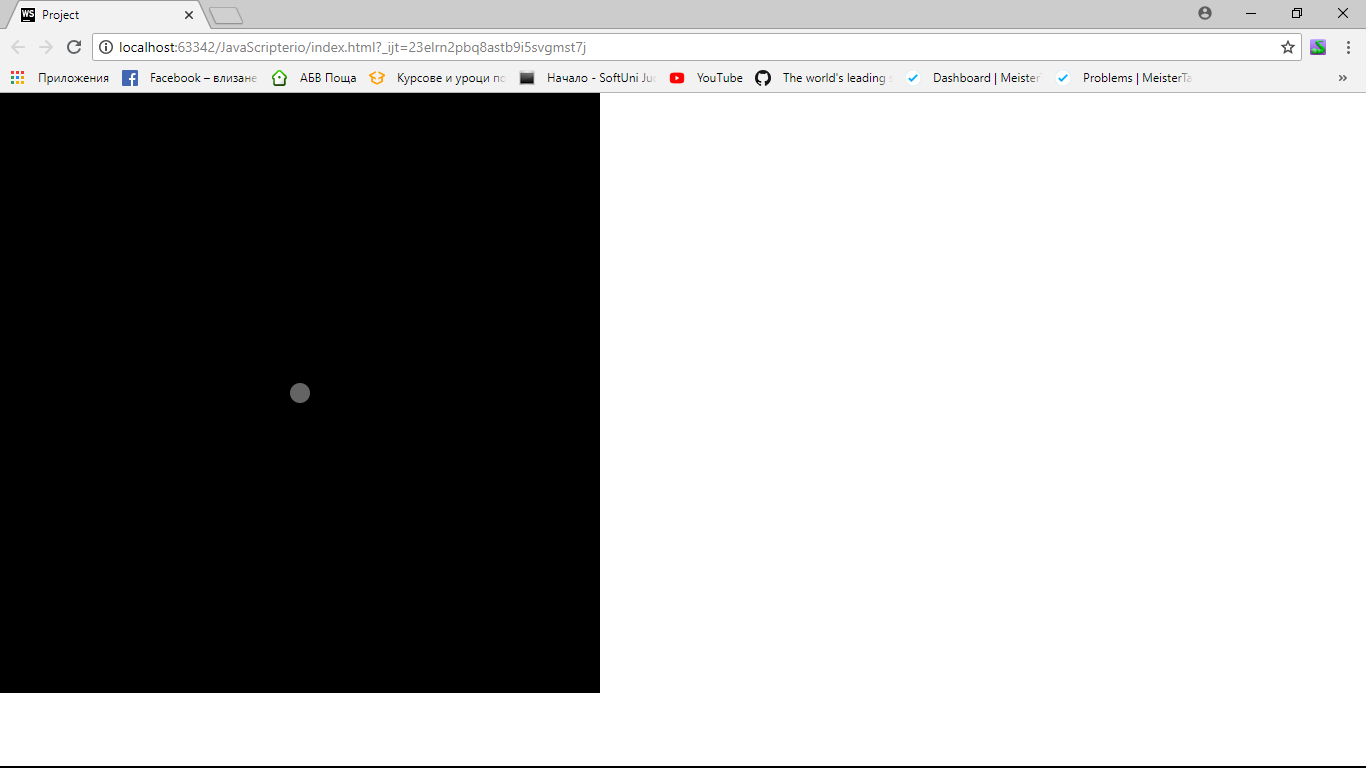


Now we can create our first segment and call the show function to display it. Go to **“sketch.js”** and create a **new segment**:

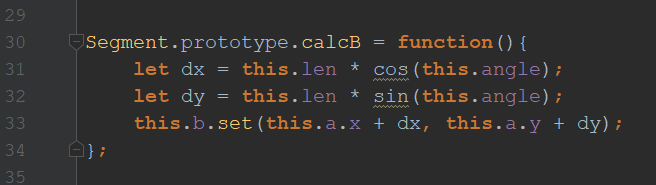


* we create a **global variable segment**
* **in the setup we initialize it** (in the middle of the canvas with width 20)
* **in the draw** function we **create a color we will pass to the show function** (remember it needs a color argument). You can use **your own** values for **R, B and G**.
* then we **call the show function** we wrote and pass it the color

**Save changes, open browser, refresh** and you should see something like this:



Now let us **add one more functionality in the segment** and then we will **create the Snake class**. We will **return to the segment when we need to** and **add more functionalities**. In order to build our snake we have to be able to **calculate the position of the B point** of each **segment based on the length and the angle** of that segment (**remember we just copied A above for the B segment**). Write the following code to calculate the position of the B point:

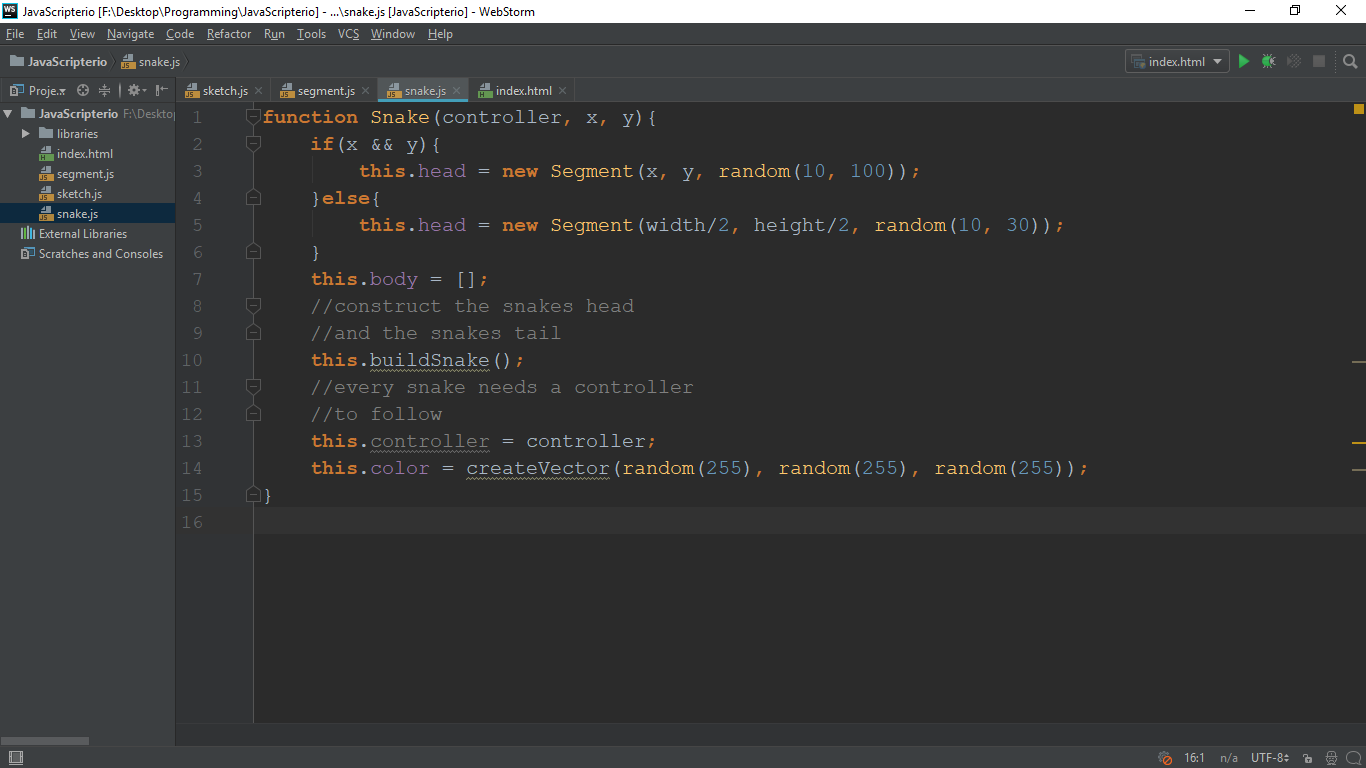


* **dx and dy** – we calculate the **distance from point A to the point B**. The **dx** is the **distance between the X of the A point and our X for B point we have to create**; **dy** is the **distance between the Y of the A point and the X of the B point we have to create**. We use [Trigonometric functions](https://en.wikipedia.org/wiki/Trigonometric_functions) to calculate that.
* **this.b.set** – setting the value of the vector to the X of the A point plus the dx, and the Y of the A point plus the dy. This is a **build-in function in p5**

We will get back to the segment class when we need to add something. Let us start with the snake class.

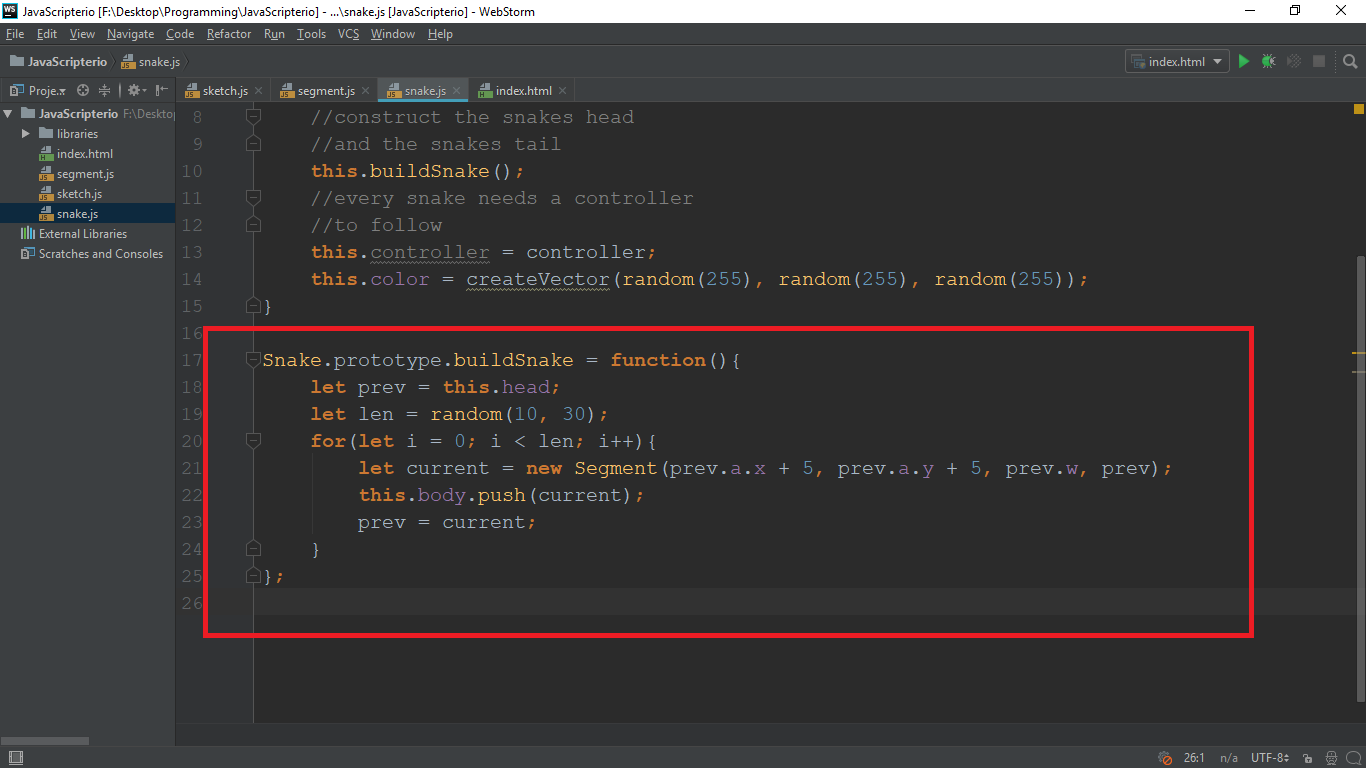
# Creating the snake class

Create a new JavaScript file named **“snake.js”** and open it. In the file, write the following:

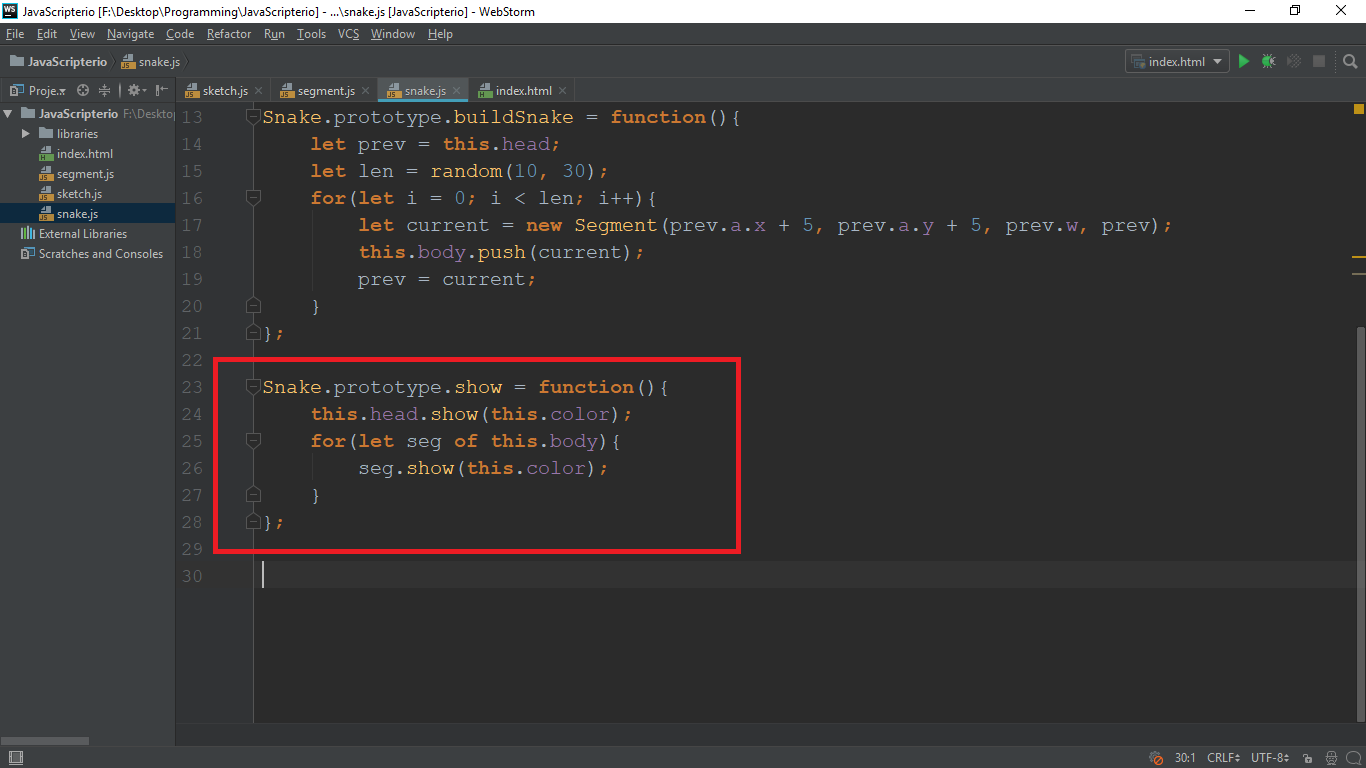


* This is going to be the **constructor** of the function. It is going to require a **controller** and **X and Y** coordinates which will be **optional**
* **head –** the head is going to be a **segment**. If we receive **x and y coordinates** we are going to **create the segment at that coordinates** with **random width** (you can adjust it if you want). However if we **do not** receive coordinates we just make it in **the center**.
* **body –** this will be an **array of segments**. They will **follow one another** to make the snake move smoothly
* **controller –** since every part of the body follows its previous segment, the head is going to follow the controller
* **color –** the color of the snake. We make it random
* **buildSnake function –** this will be the function that constructs the body of the snake. We still have not implemented it. So let us do it now.

After that code, write the following:



So, after having the function to build our snake we need a **function which will show it**. So let us add an **show** function for the snake.



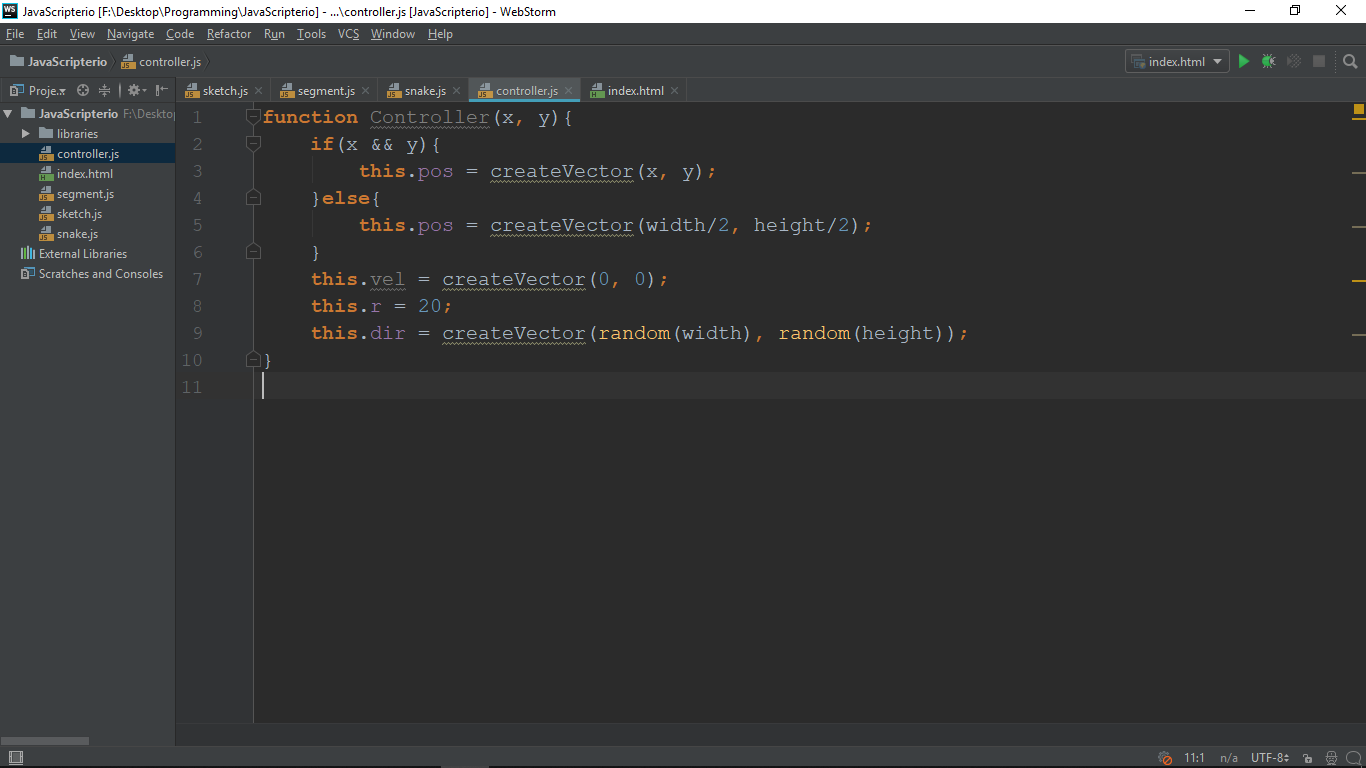
* **this.head.show(this.col)** – remember that the head is a segment and we already wrote a show function in it. So we just call it passing it the color of the snake
* The parts of the **body are also segments**. So we **loop through the body** and **show** each **segment.**

**Note: don’t forget to add the “snake.js” file to the “index.html”. Just like we added the segment**

If you reload the page now you **should see a dot**. That is because all of the **segments are on top of each other.** In order to see it better let us just **make our player follow the mouse**. We **will leave the snake class for now** and we are going to **create the controller class** (since the **head of the snake should follow the controller**).

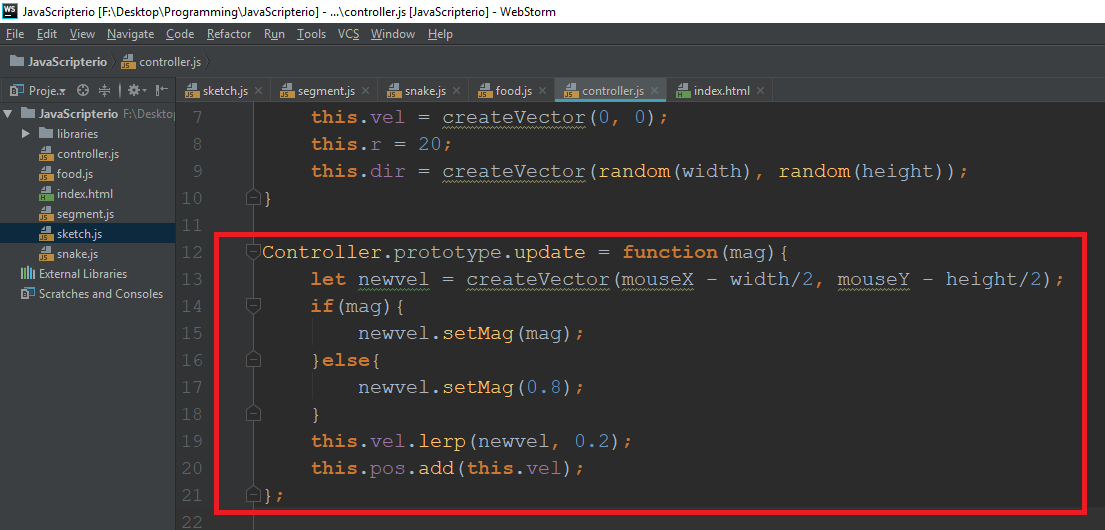
# Creating the controller class

Now add **a new JavaScript** file called **“controller.js”** Open it and type the following code:



* The controller class will have only two (optional) parameters. If it receives them it is going to be created on that position (x and y), and if not, we create it in the center of the screen
* **this.pos** – stores a **Vector (x and y) for the position** of the controller
* **this.vel** – stores **the xSpeed and the ySpeed** (starts with zero speed)
* **this.dir** – **stores x and y** for the direction it is going to move (**target**). **It starts at random** direction. Later in **our update function we will make it follow the mouse**.

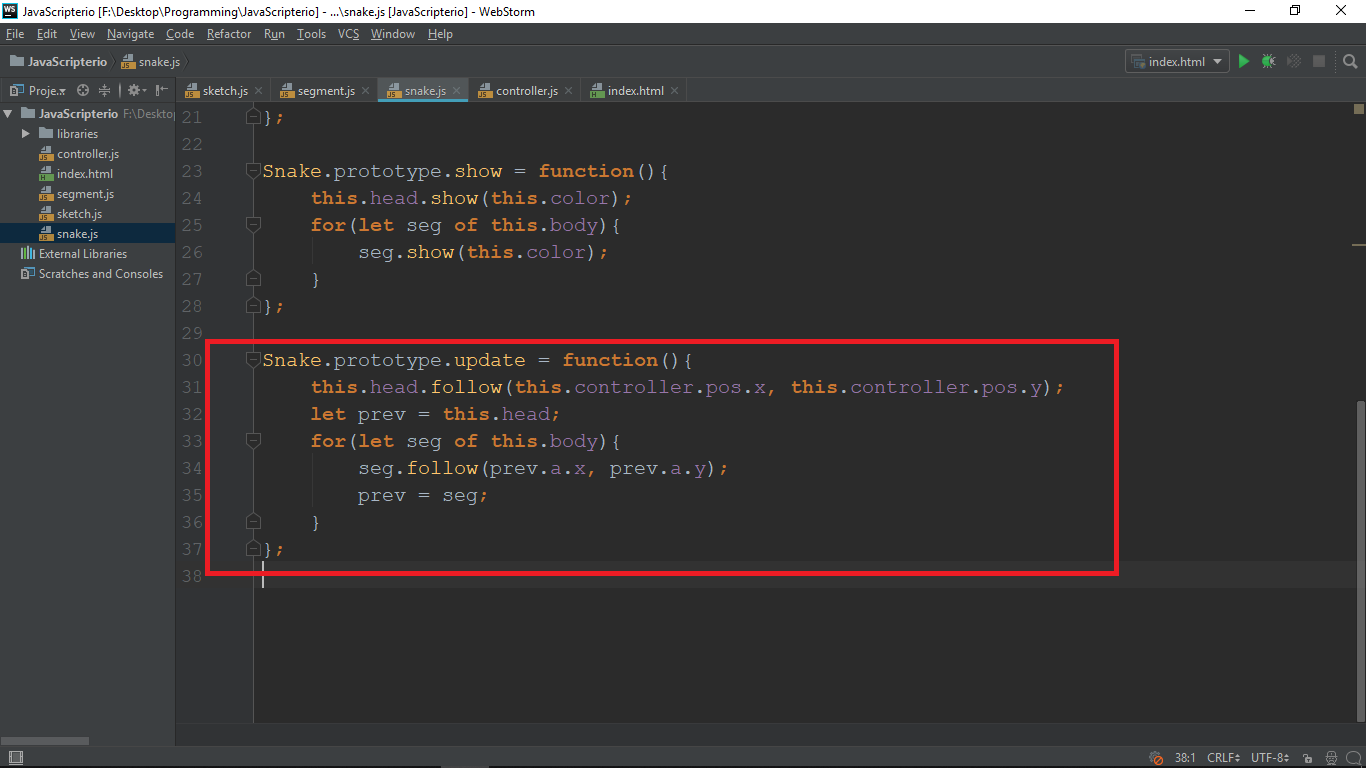
Now the final thing before we actually create a player with a controller is **to create that update function for the controller**. Add a function with the following code:



* **newvel** – this is the **new velocity** of the controller (or the **new direction it is going to move**). This is the **mouseX** and the **mouseY** (those are **included in the p5 library**, the **get the current position of the mouse**).
* **mag** – this is the **magnitude (or the size of the velocity vector).** We use **setMag** (build-in function) **to limit the speed**, because if we don’t the **speed will increase exponentially** (as you see on **line 20**, we **add the velocity to the position**, so we **have to limit it**) If our function **receives a mag, we set it, if not we give it a magnitude of 1.5**. You will see the use of it later when we create the sprint function (**when sprinting we want to move faster**)
* **lerp** – the lerp function is also **build-in function in p5**. This will help us **to turn smoothly**, **not directly in the position of the mouse**. We use **0.2** as a value, because **we want to turn slow (smaller value, the slower)**
* Finally **add the velocity vector to the position vector** to move (using **add**, a build-in function)

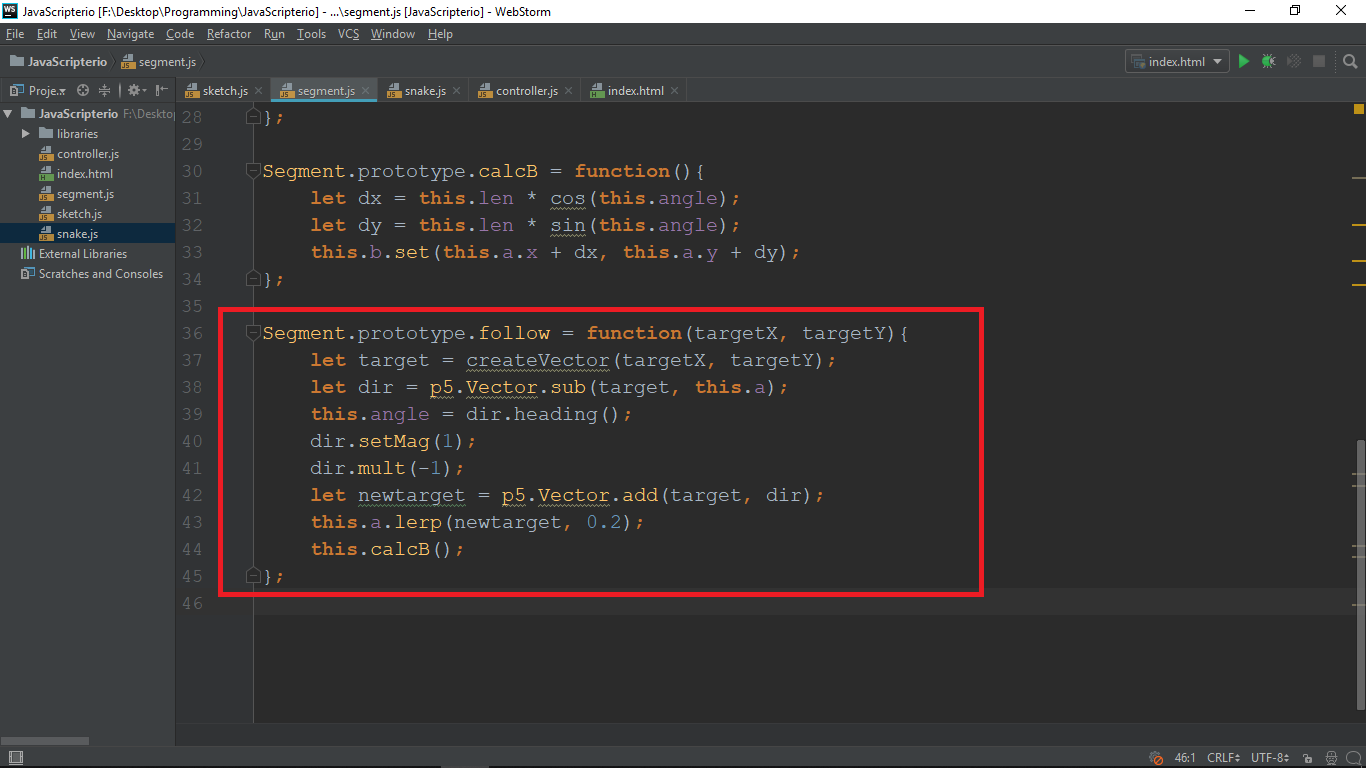
# Moving the player

Now let us **return to the snake class** and **add an update function** that will make the **head follow the controller, and each body part follow its parent**. Type the following code in the **snake class:**



* We want the **update function to make our snake move**. The way we do that is by saying to **the head of the snake to follow the mouse**. Then we create a **variable to store our previous segment** (starting from the **head**). Then we **loop through the body** of the snake and make **every segment follow our prev variable** (after each segment **set the prev to the current segment**).
* As you may notice, **we don’t actually have the follow function**. Since **the head** and **the body** parts **are segments**, we create that **follow function to the segment class**. Therefore, **that is our next step** and after that we can finally **create our player and make it move**.

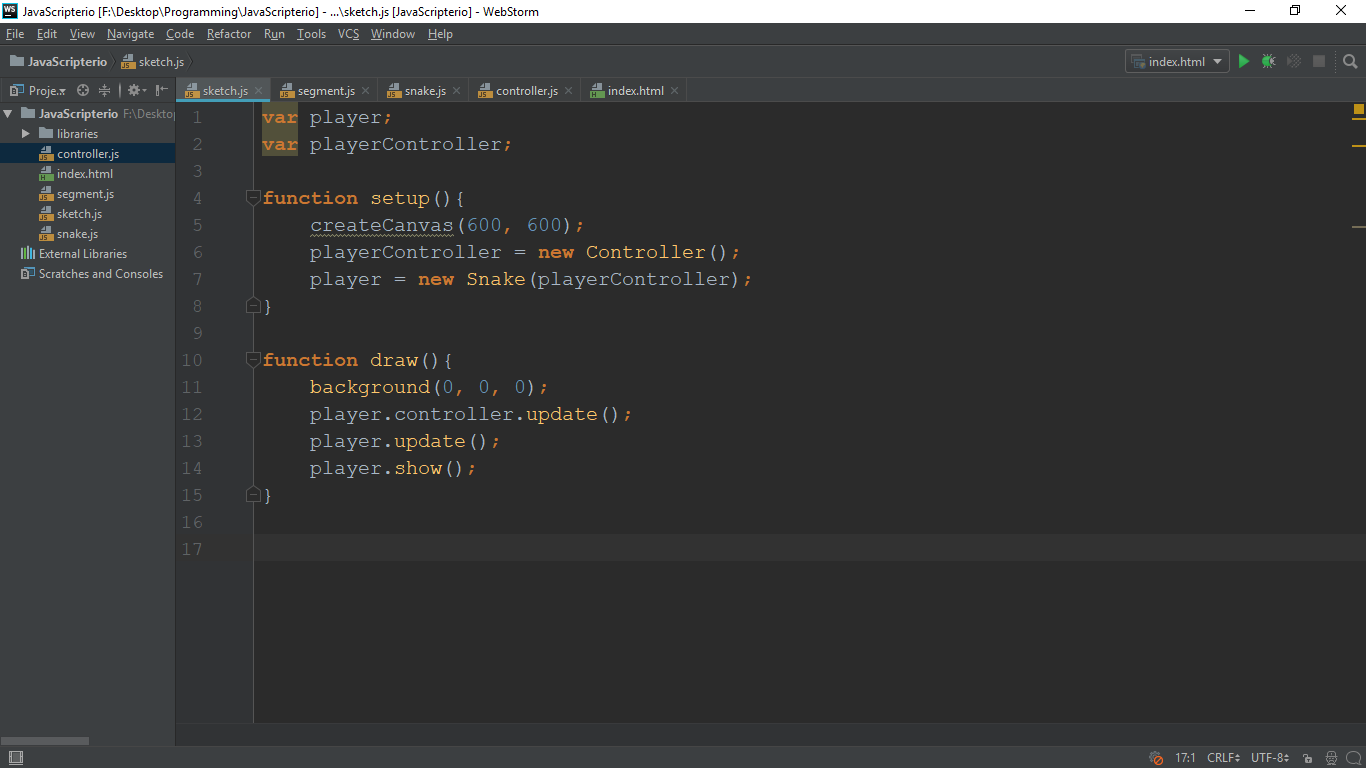
So go to the **segment class** and add the following function:



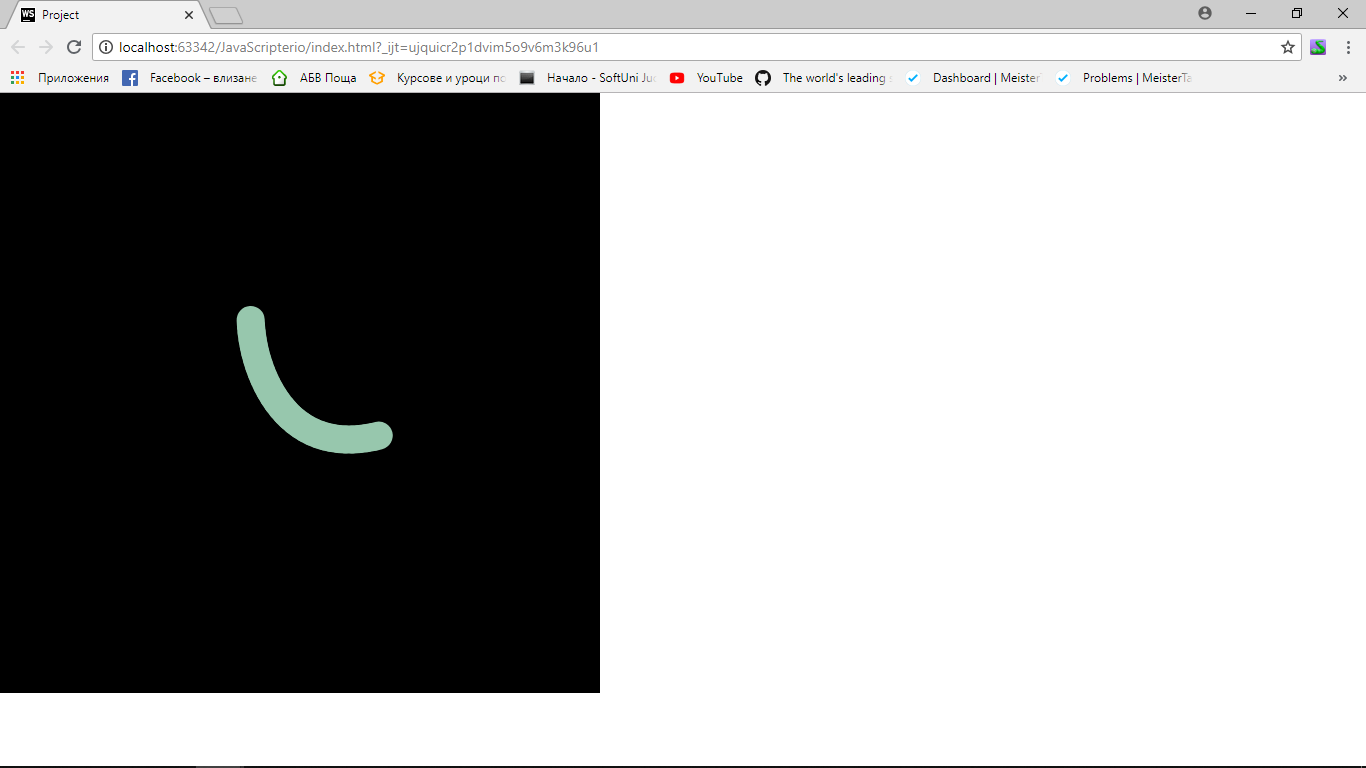
* So, we want our follow function to receive a **targetX and targetY** for the **position of our target** that the current **segment will follow**. (target variable)
* **dir** – we calculate the **direction we want the segment to move** by **subtracting the A position** of the segment (the first point) **from the target position**. We use the **build-in p5 function** for vector math (**p5.Vector.sub**)
* **this.angle = dir.heading()** – heading is a **build-in function** in p5 **to calculate the angle of a given vector** (we need the angle to **calculate the position of the B point** on **line 44**)
* After doing all of the math, we want to **limit the speed** (so we use **setMag** again), and then we **multiply the direction vector by -1**, because **otherwise the snake will be moving backwards.** Then we use **lerp** again to make the movement smooth.
* Finally, we **calculate the position of the B point** for that segment

Ok, now **save the code** you wrote, don’t forget to **add the files in the index.html** file

Now go to the **sketch.js** file, **remove the code for the segment** (we don’t need it anymore) and **add the code to create, update and show the player and its controller**. After that, the **sketch.js file should look like this:**



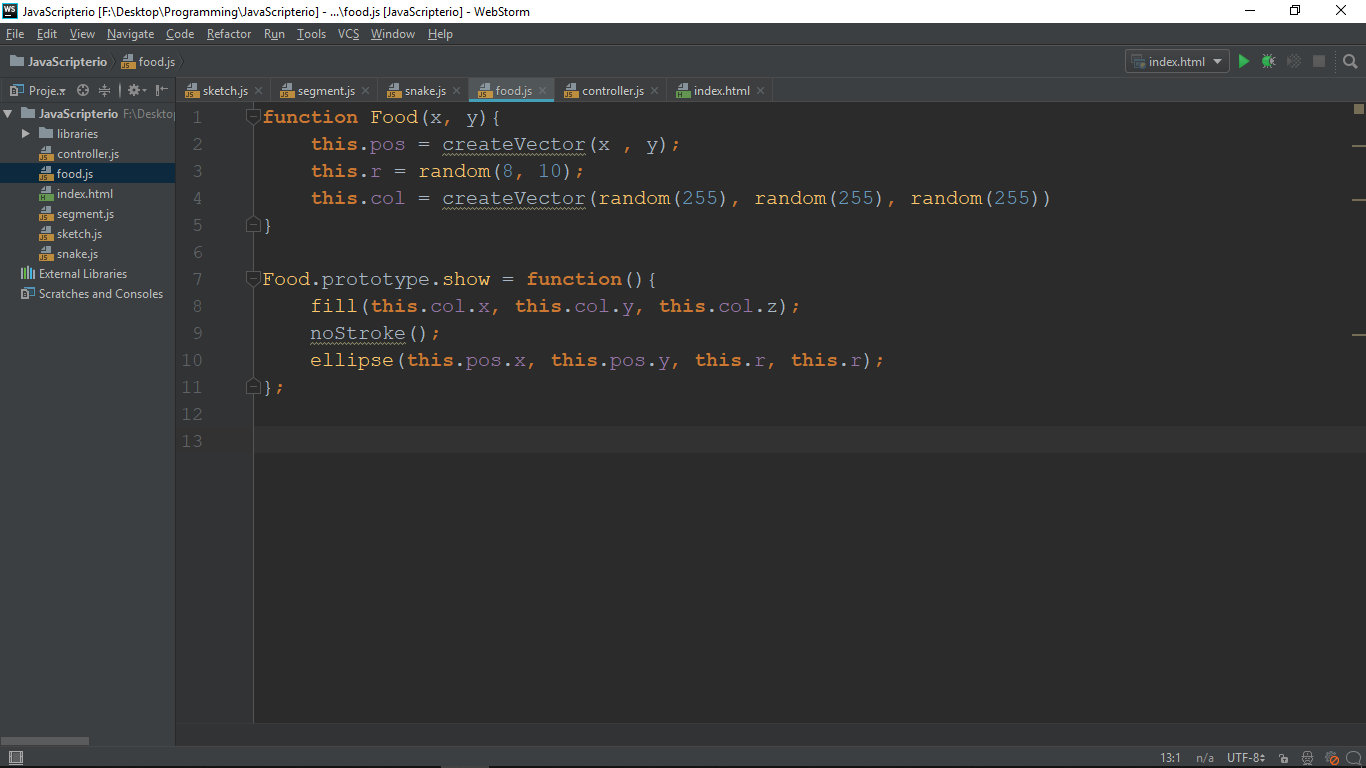
**Now save and reload the browser page.** Our player is now moving (the movement is not precise right now, but we are going to fix it later)



Perfect! Our next step is creating the food.

# Creating the food class

This is going to be quite simple. Just create a new JavaScript file with name **“food.js”.** **Don’t forget to add it to the index.html file!** After that write the following code:



### The constructor

* our food class will require a **X and a Y** for its **position**
* we create a **position property** (**a vector** at the X and the Y)
* than we have a **radius of the food** (to make the game cooler)
* and finally we have a **color of the food** (a vector with **random RGB colors**)

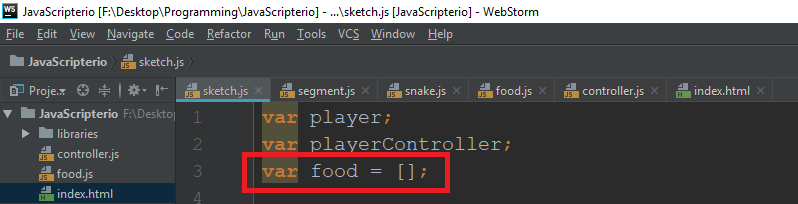
### The show function

* like every class we have created so far, for the food we also **needs a function that will display** it on the screen
* **fill** – this is a build-in p5 function that **sets a given color** (we make it the property of the color of the food)
* **noStroke** – also a build-in function that **takes away boundaries**.
* **ellipse** – draws an ellipse (**requires a X, a Y, width, and height**). Since our **food will be a circle** we use **its radius** for both of them.

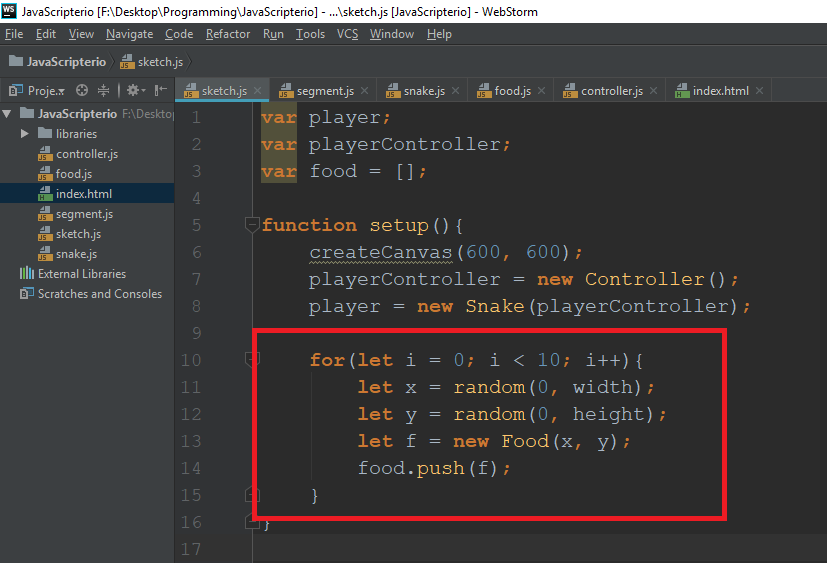
And that’s it for the food. Now let us add some food to our game

# Adding food to the game

Go to the **sketch.js** file and **add a new variable called food** and set it to **be an array** (since we will add many piece of food and what a better way to store it than an array)

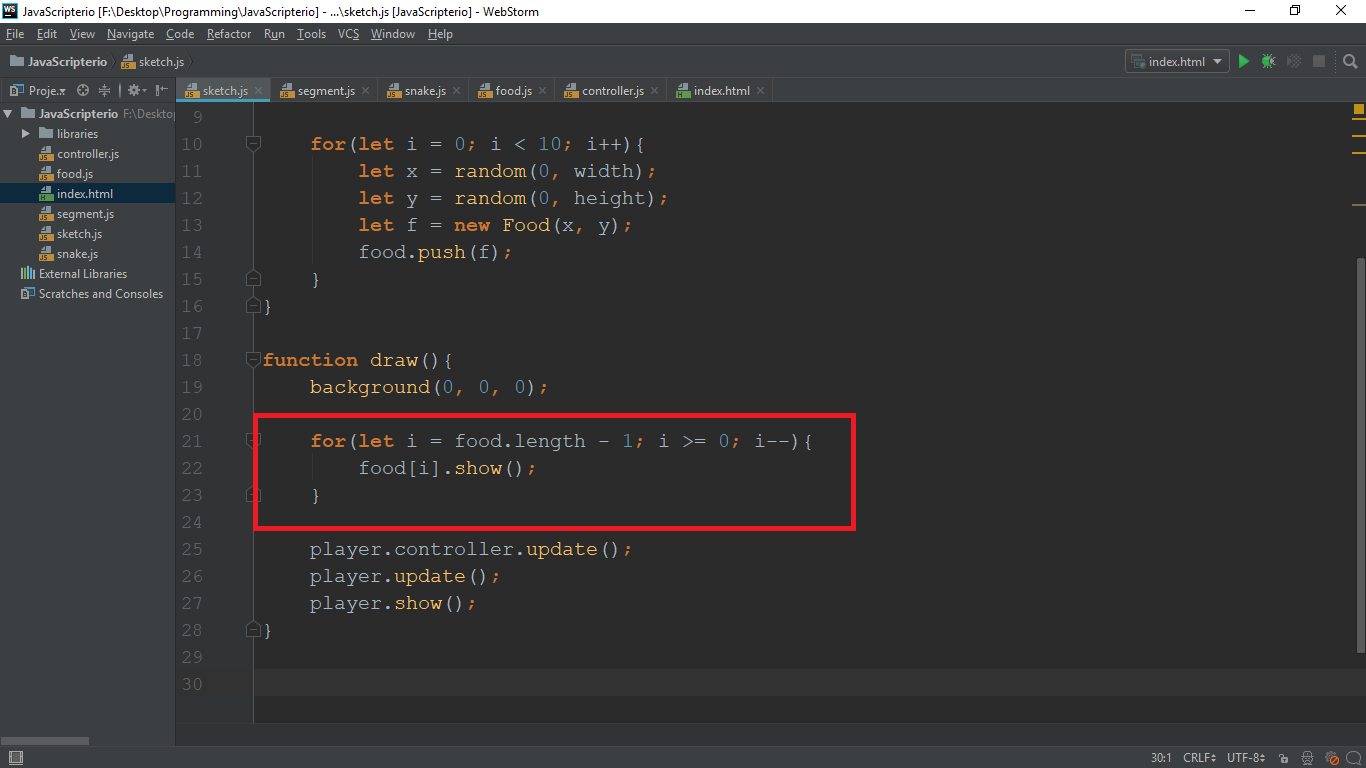


After that go write the following code in the **setup** function (we **create food only once**, that is why we write it in there):



* we make a **for-loop** with **10 iterations** (it will create **10 pieces of food**)
* we choose **random X’s** and **Y’s** (our **food** class **requires a X and a Y**)
* we **create a new piece of food** and **push it in the array**

And finally we need to **display** the food. Go to the **draw function** and write the following lines of code:



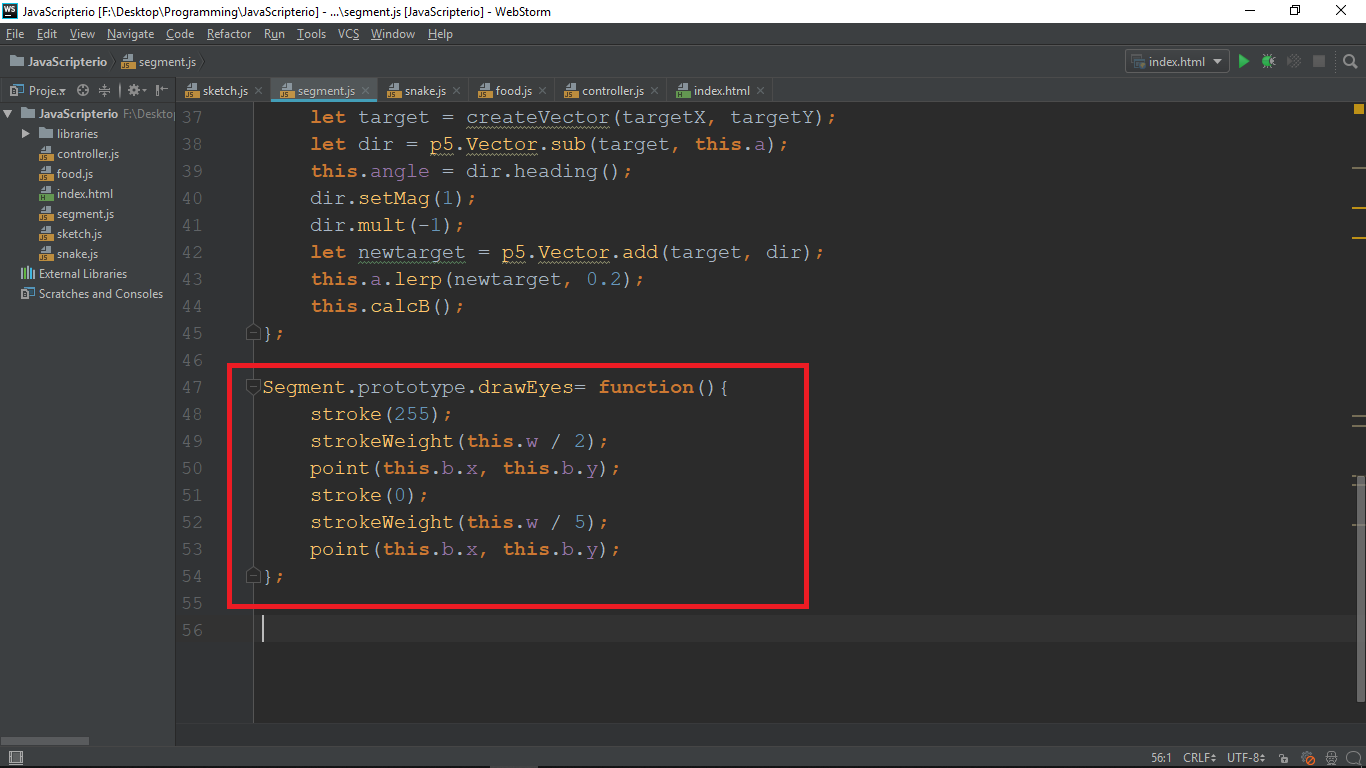
* The way we **display the food is again with a for-loop**. **Food[i]** is **the current food we are at** in the loop
* We loop through the food **backwards**, because later on **when we** add the functionality **to eat it**, we have to **remove the piece of the food array**. When **looping backwards we protect ourselves** from **skipping an element when deleting** one.

**Save** and **reload** browser. You should see something like this:



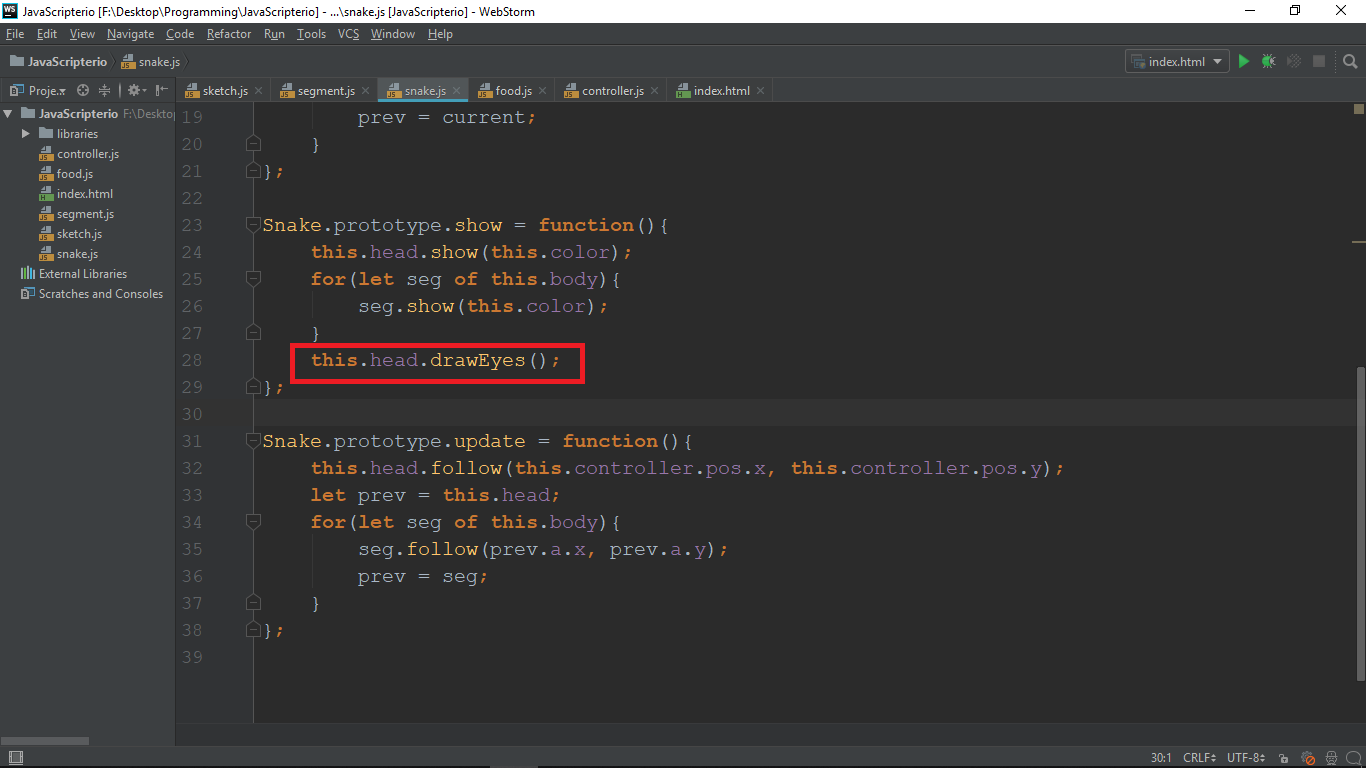
Let us now make our snake look a bit more like a snake. We are going to draw an eye on the head of the snake to make it look more realistic.

Go to the **segment.js** file and add a function named “**drawEyes”** with the following code:



* First, we want to draw a **big white point in the center of the head**. We **set the stroke** of the point **to be white**, and then we set the **stroke weight (or thickness)**. After that, we **draw a point at the B point of the head**.
* We also want to **make a smaller black point** in the **center of the white point**. So we set the **stroke to 0 (black),** we make the **stroke weight smaller** than the stroke weight of the white point and **draw it also at the B point position**. Now we are ready to add the **“drawEyes”** function in the head of the snake.

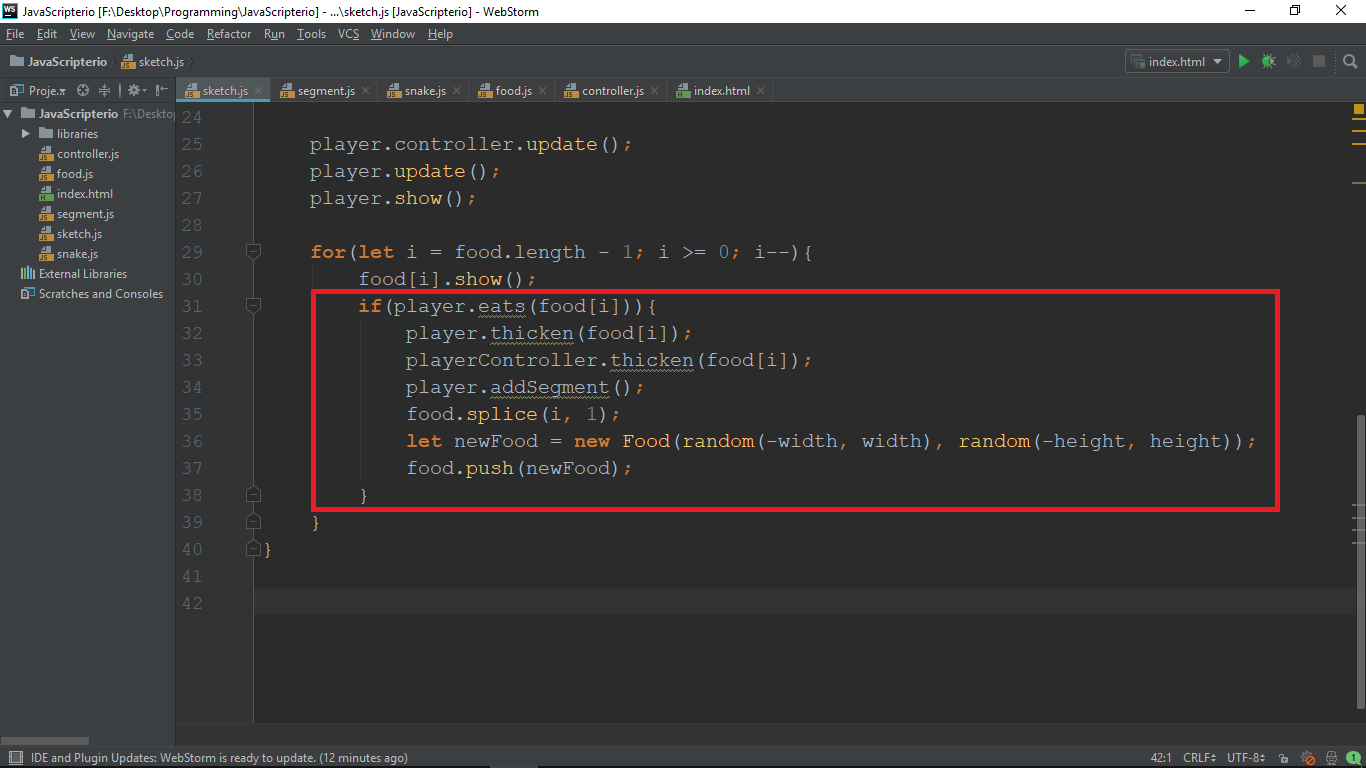
Now go to the **show function of the snake** and add on the last line **“this.head.drawEyes();”** (since the head is a segment it contains that function)



**Save** and **reload**. Our snake now has an eye!

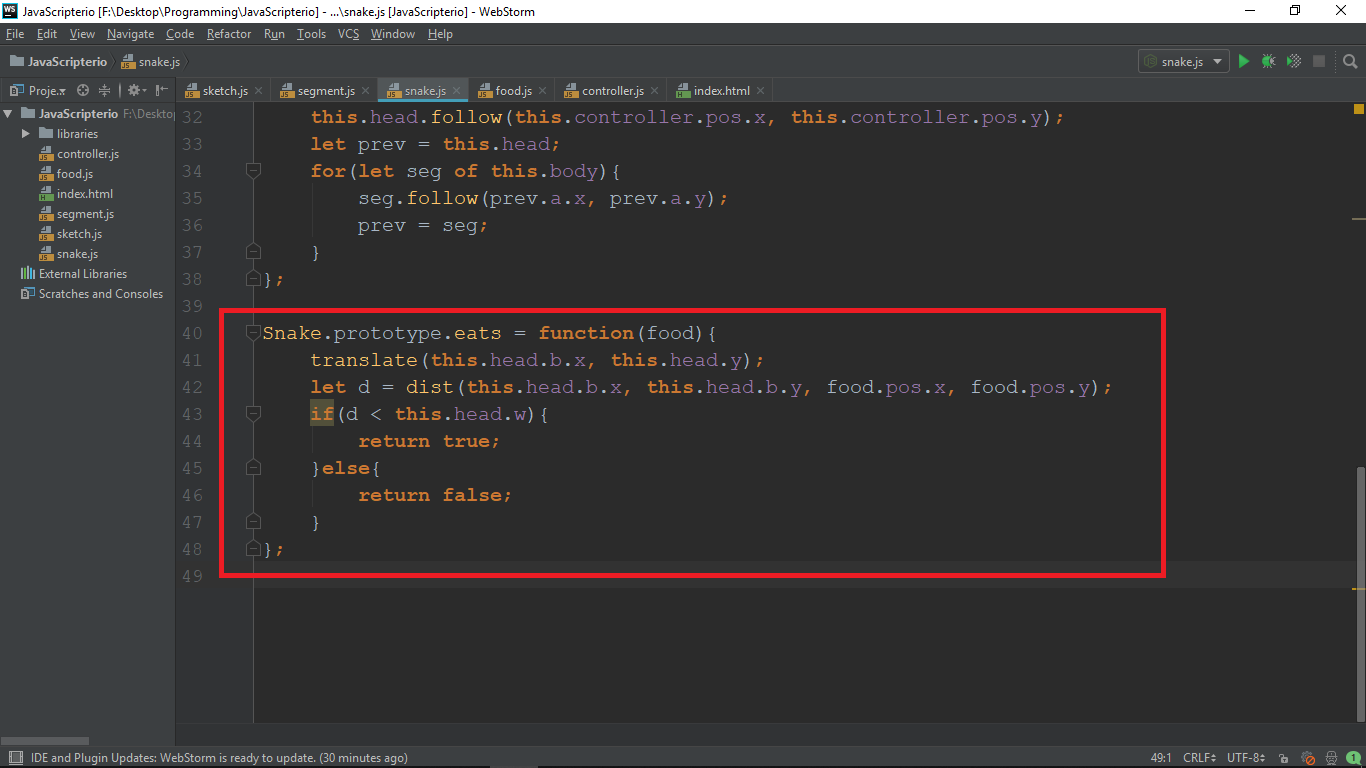
# Make the snake eat food

The next thing we want to do is to add a functionality for eating food. We need to check if the player eats a piece of food every frame. So we do that in the draw function. Go there and add the following code in the loop we draw the food:



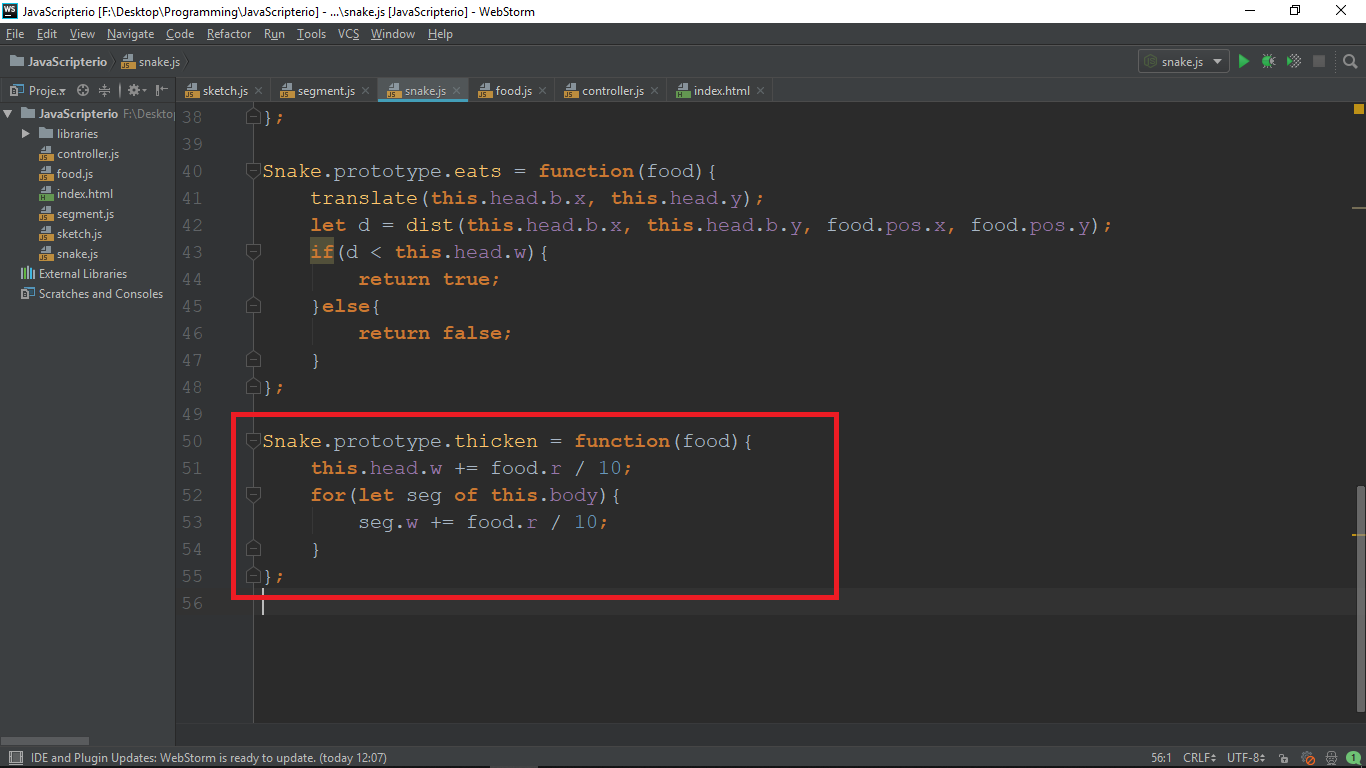
* For each piece of food we **check if the player eats it** (we will **implement that in a bit**)
* If the players eats the food we want the **snake to become thicker and longer** (we also make **the controller thicker**, because it is a point and **we want it to proportional to the snake**) (**we don’t have these functions** either, so **we will implement them in a bit**)
* We also want to **remove the piece of food that the snake has eaten**. We use the JavaScript function **splice**.
* After the snake eats a piece of food, we **create a new one on a random location** and add it to the array of food.

Now let us implement that **eats, thicken** and **addSegment** functions. Go to the **snake class** and **add** the following code:



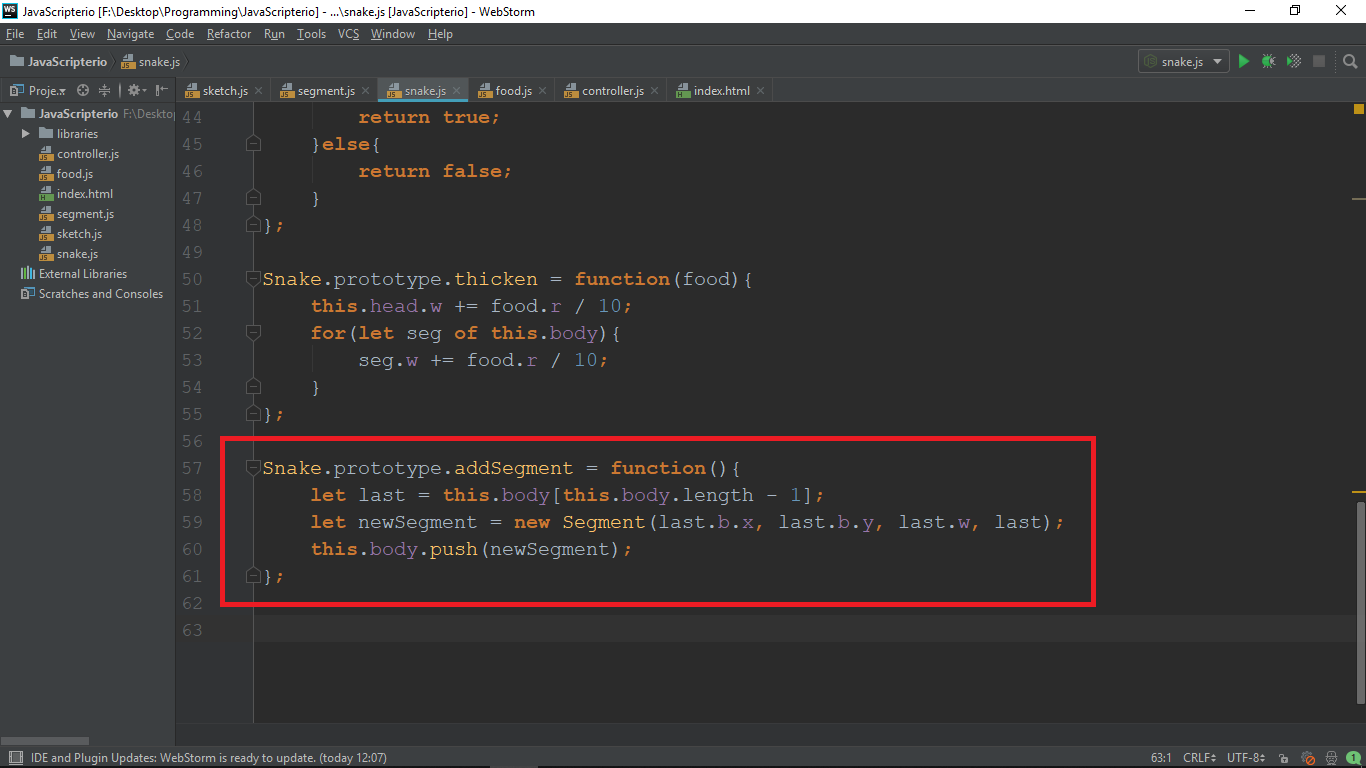
* The **eats function** will receive a **piece of food** and will **calculate the distance between the head and the food**. If the distance **is less** than the **width of the head**, we consider that piece of **food as eaten**
* **translate** – this is a build-in function in p5. What it does is, it makes the given X and Y (coordinates) **the center** of our coordinate system **(0, 0)**
* We use **dist** to calculate the **distance between the head and the food**. If the distance is **less than the width of the head**, we **return true** (which means that **the player has eaten the food**)

Ok, now let us add the function that will make our player thicker when he eats food. Add the following code:



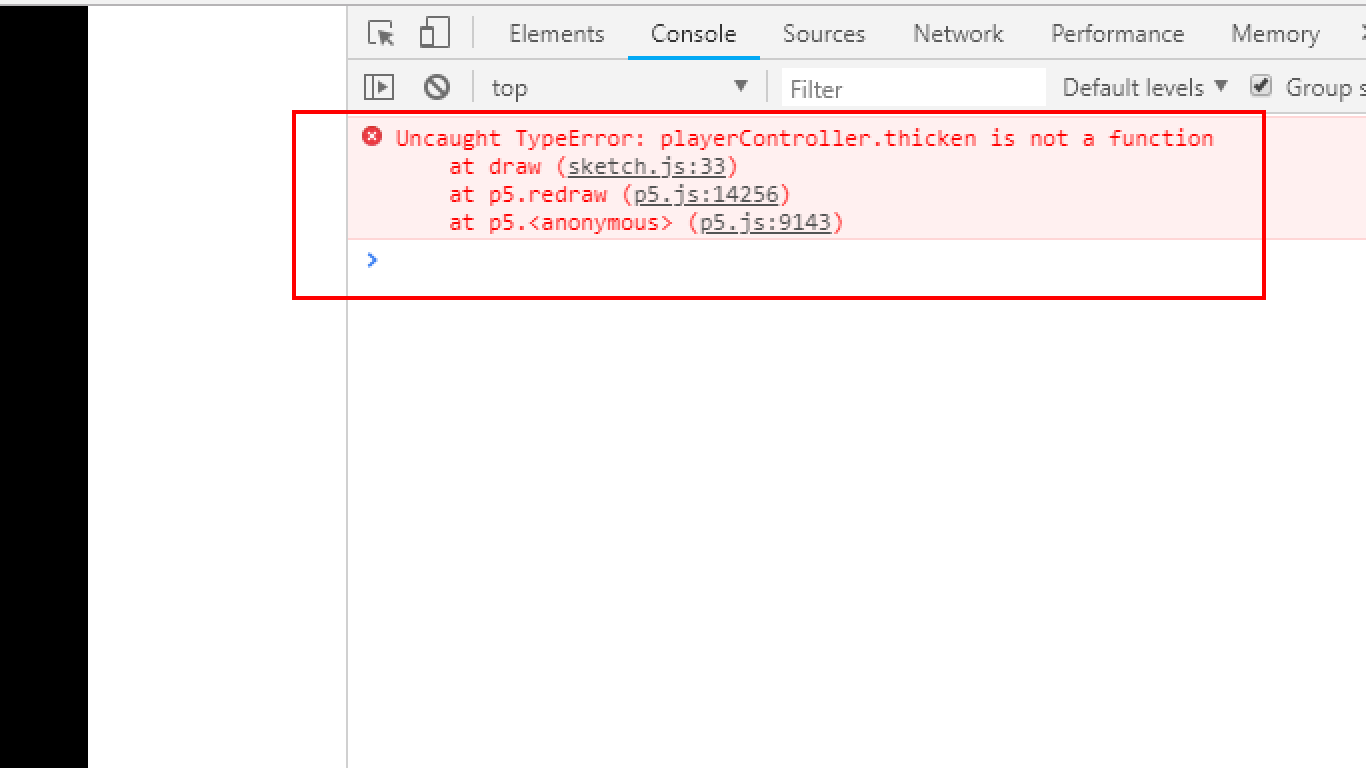
* The thicken function will **receive a piece of food**, in order to make our player **thicker based on the radius** of the food.
* We **increase the width of the head** with the **radius of the food divided by 10** (because we want to make it just a little bit bigger)
* Then we **loop through all of the body parts** and **make the thicker by the same amount.** (in order to be **symmetrical**)

Finally, we have to add the function that **makes our snake longer** (the **addSegment** function). Add the following code:

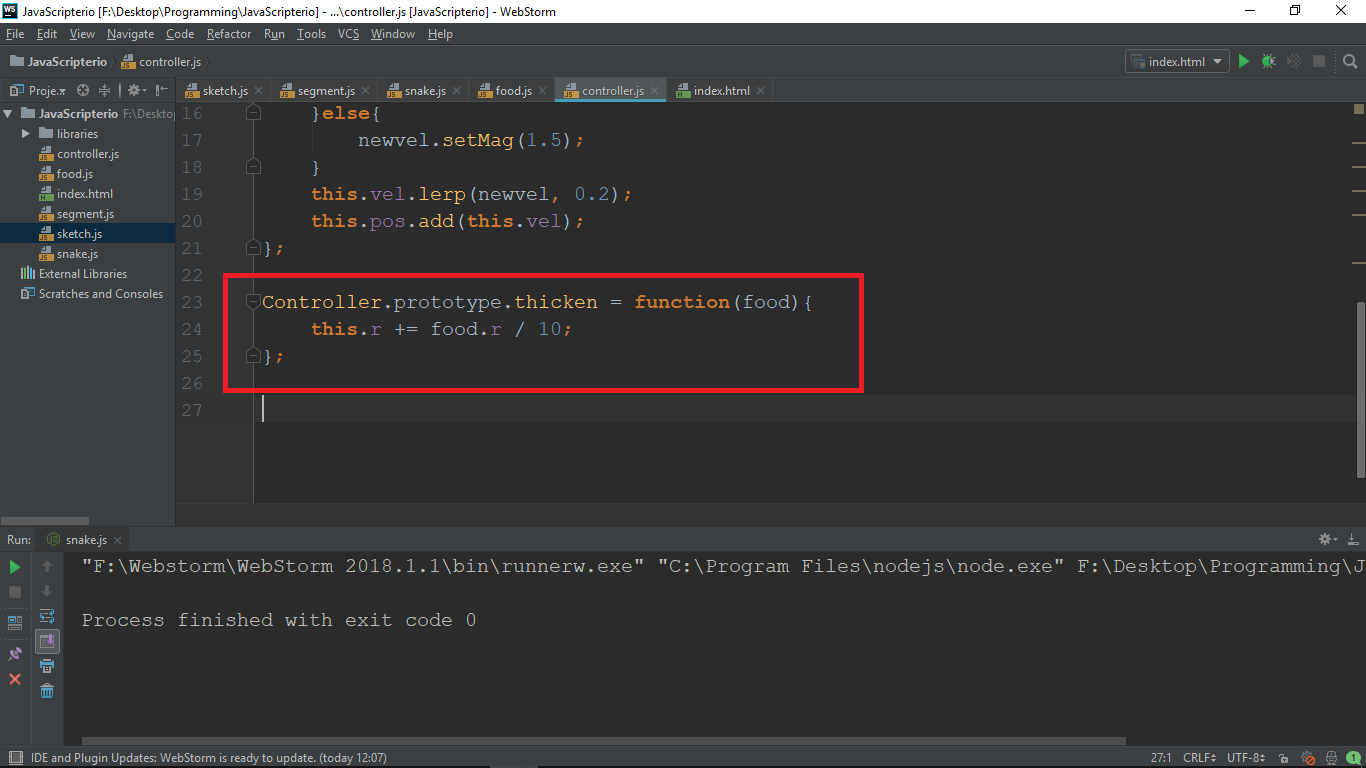


* In order to add a new segment at the back **we need a parent that it follows**. So, we **take the last segment of the body** and **create a new one**. We **pass the B point coordinates** as a parameter to follow. Finally **push** the new segment **to the body.**

**Save** and **reload.** If you try to eat food, you should see an error. To see it open the browser dev-tool.



* We forgot that we want to make our **controller thicker**. So let us go and **add a thicken function to the controller**. Open the **controller.js** file and add the following function:



* We just need to make the **radius of the controller bigger** by the **same amount** we did **in the snake**

**Save** and **reload.**

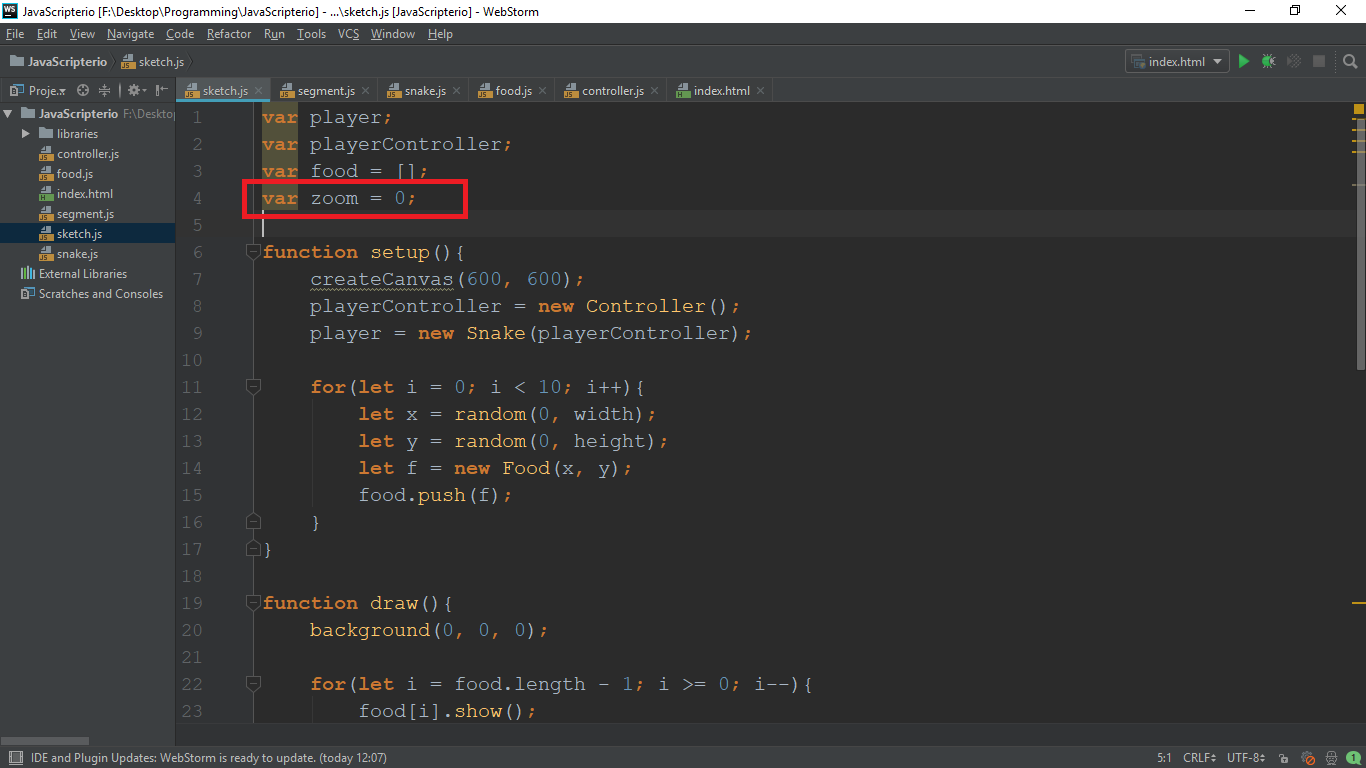
Now it will work. It still moves kind of incorrect so let us fix that. The reason it does that, is because of **the translate function** (in **changes the center of our coordinate system**). So now, we are going to **make our own translation function** to correct it.

# Make translation function

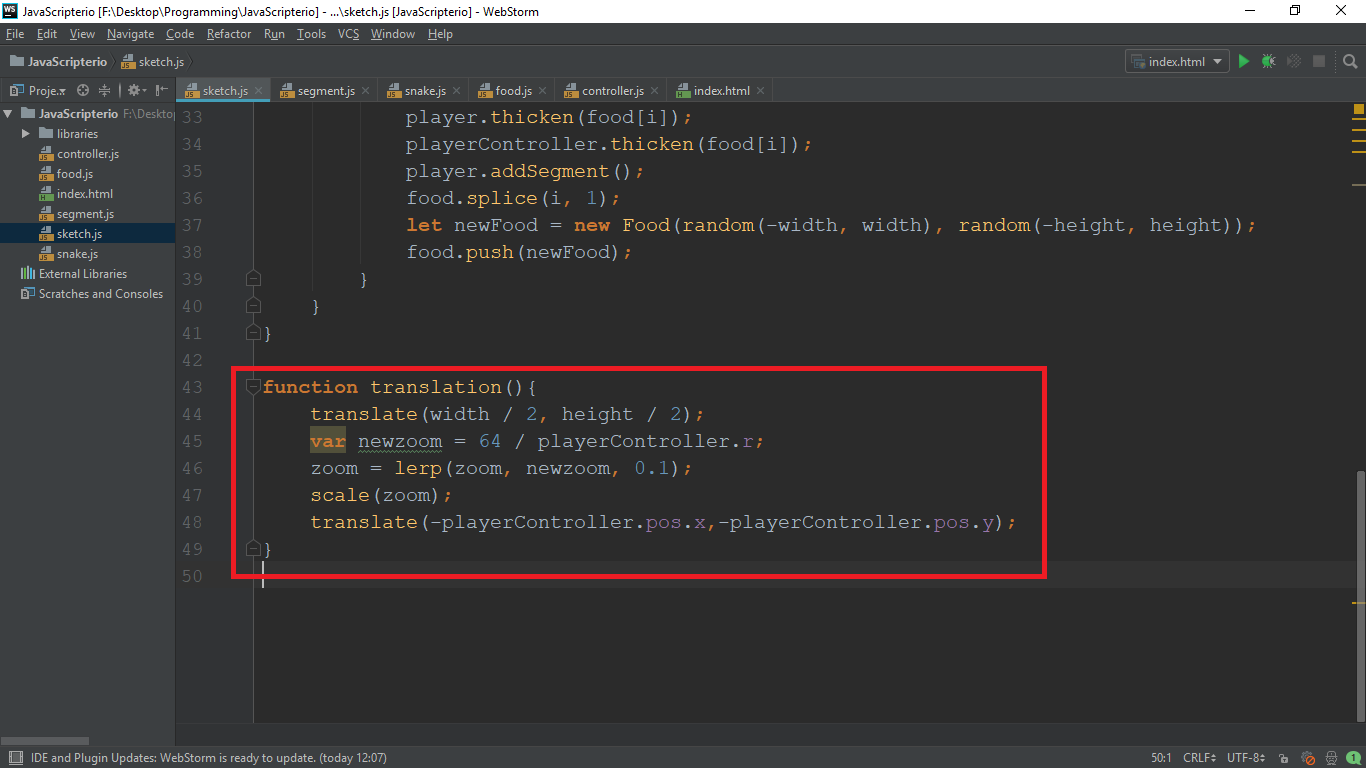
The point of that function is to **make our player move correctly**, but there are **two more things** that is going to do:

* When the **snake gets bigger**, we want the **screen to zoom out**.
* We want to make our **player the center of the coordinate system** **and move the environment**.

Firstly, go to **sketch.js** and create a global **variable zoom** and set it to **zero**

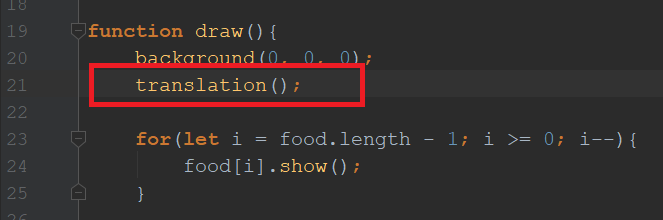


After the draw function add function **“translation”**. In there write the following code:

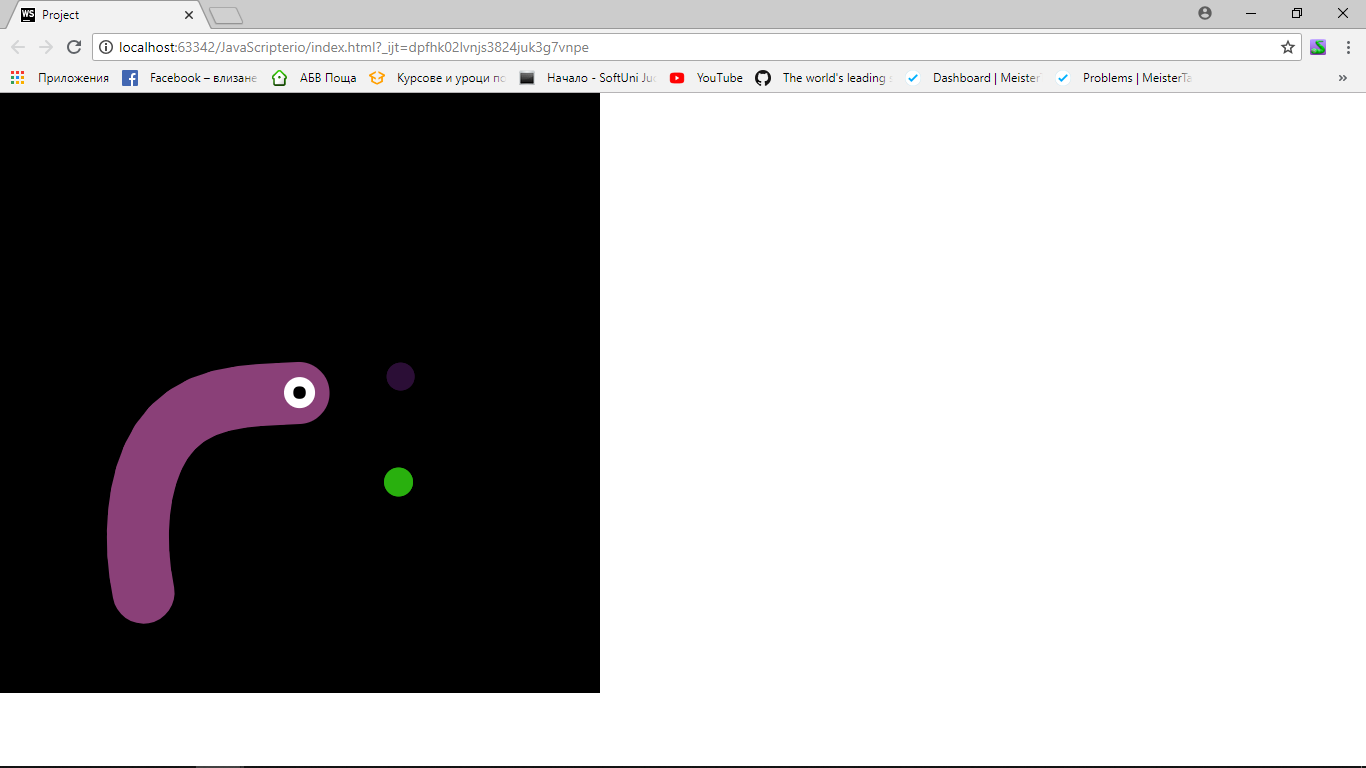


* Firstly, we want to **return the center of the screen** to be **the center of the canvas**
* Then we want to **calculate the new zoom.** That will **depend on the radius of the controller**
* Then we make **the zoom equal that new zoom** (we **lerp it** to make it **zoom out smoothly**)
* Then we use the **scale function** to actually **zoom out by the zoom** (it is a **build-in** function)
* Finally, we **translate again**. We **set the center** to be **–playerController.pos.x** and **–playerController.pos.y** (we make it negative so that **the player always appears in the center** of the canvas)

Now go to the **draw function** **and call that translation we wrote** by typing **“translation();”:**



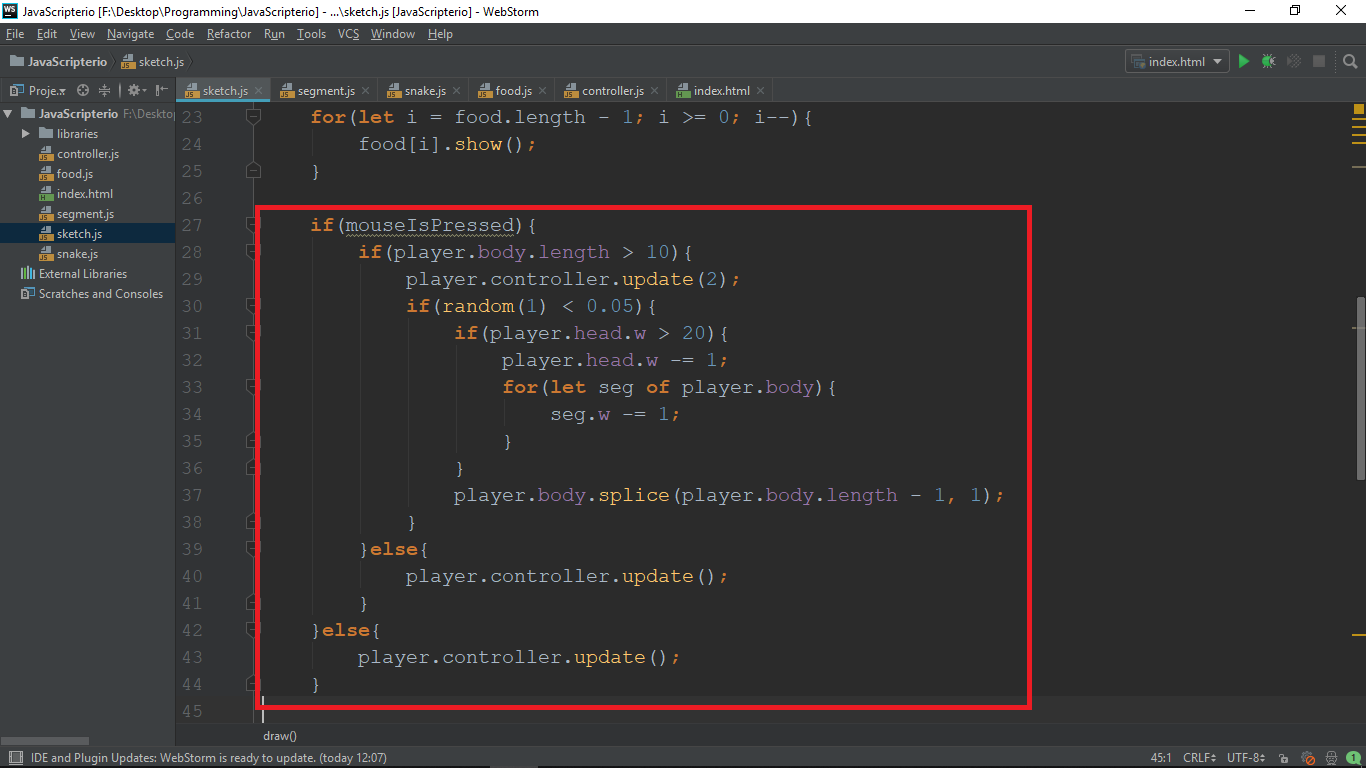
**Save** and **reload**. You should see something like this:



The **next step** is to make our player **sprint**.

# Sprinting

We will use the build-in **mousePressed** property to check if the player is pressing the mouse button and make the player move when it is pressed. Go to the **draw function** and **add the following code**:

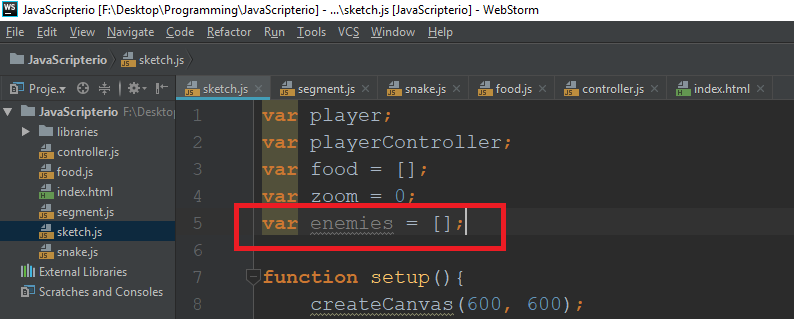


* When **sprinting** we want the snake to become **thinner and shorter**, but **we don’t want to get** **too thin or too short**, so we want to **do some checks**.
* Firstly we **don’t want to sprint if the length of the body is below 10**
* Then we **don’t want to remove a segment too often**, so we do **it 5% of the time**
* If the **body length is greater than 10**, we update the **controller with magnitude of 2**, **otherwise**, we use the default (**0.8**)

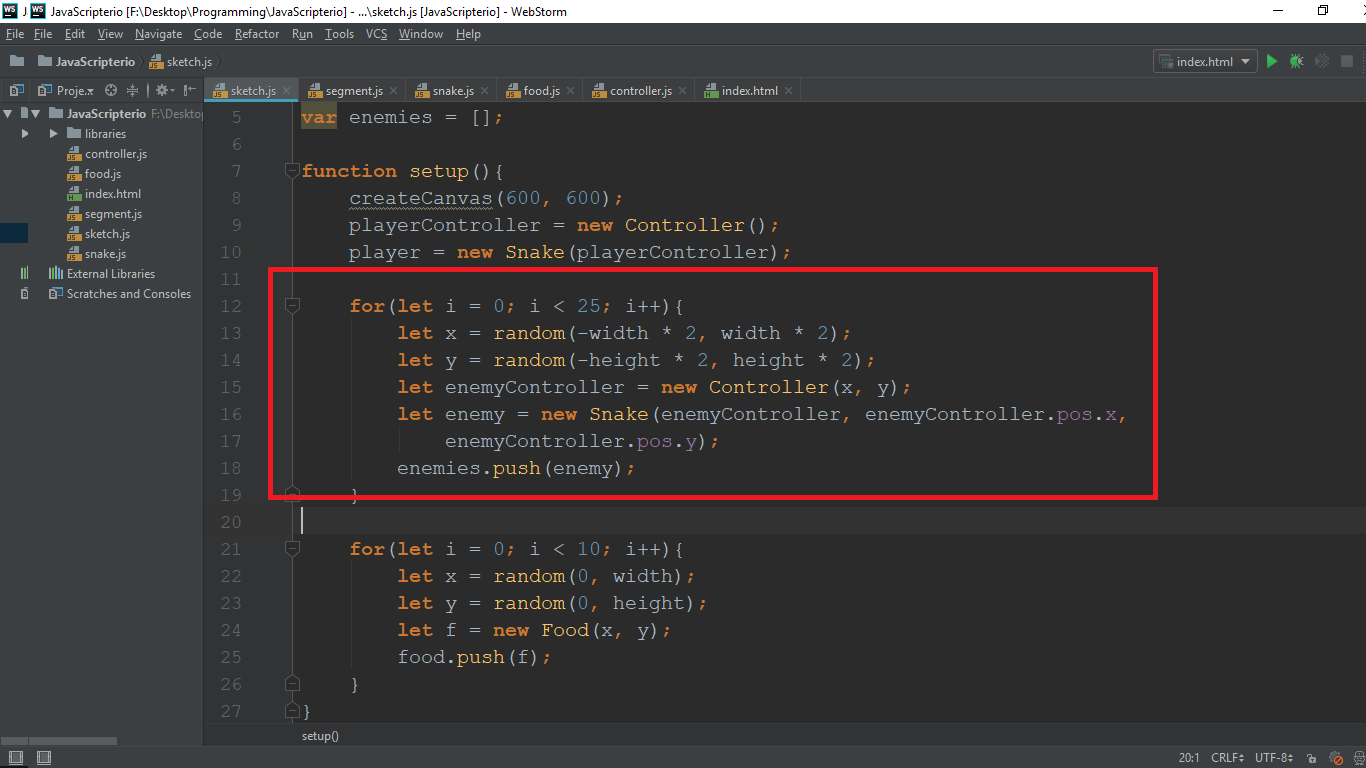
We are now almost done. The last part is to **add the enemies**, make **them move**, being able **to kill** and **be killed** by enemies.

# Adding enemies

First let us go to the sketch.js file and make global variable called enemies that will store an array of enemies.

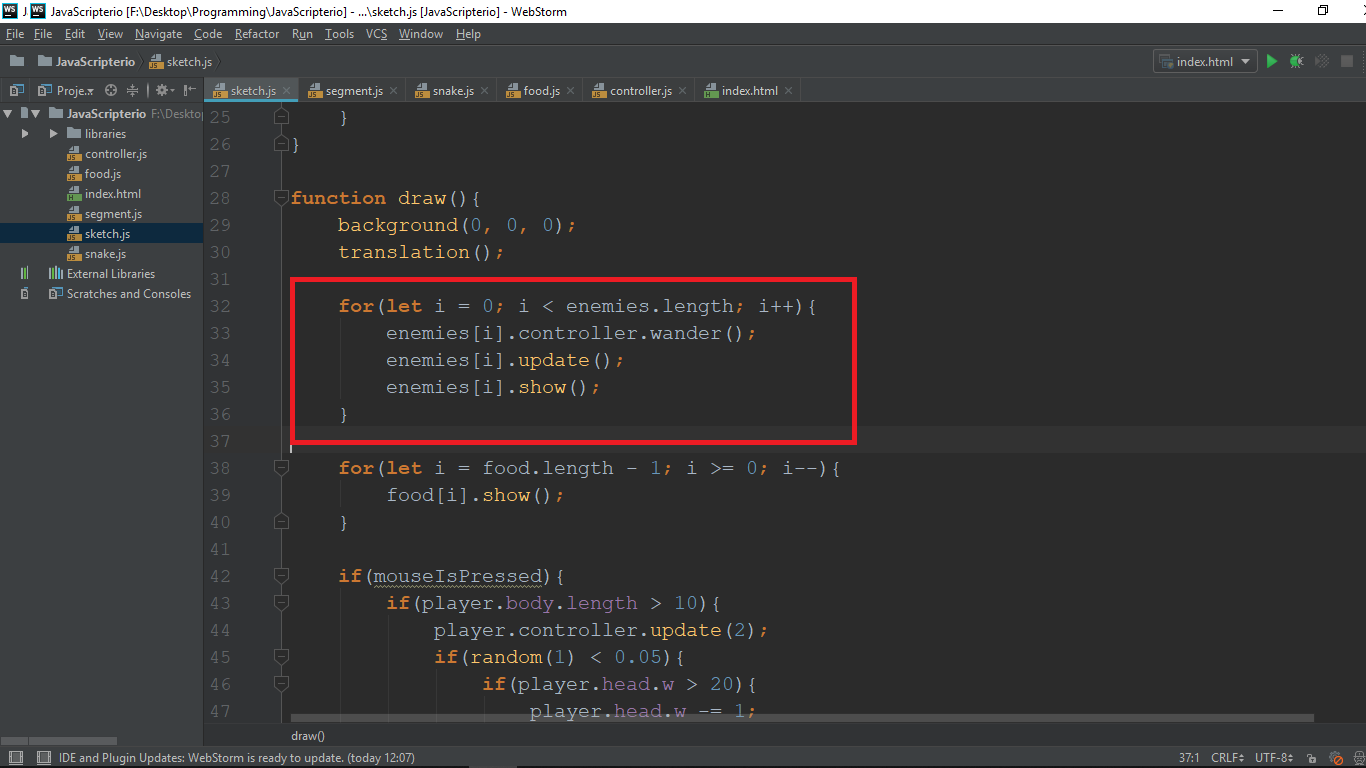


Then let us create some enemies and push them to the array:



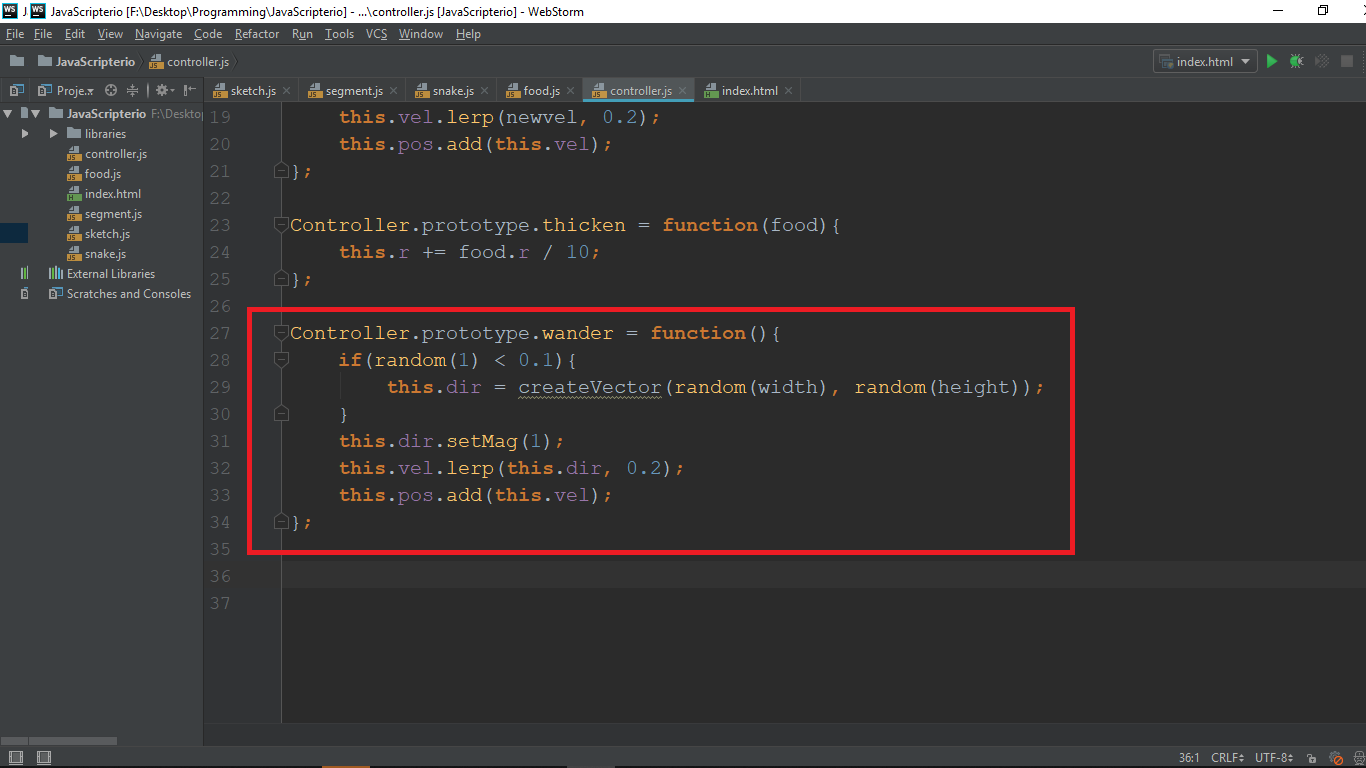
* We make a for-loop to create **25 enemies**. We choose a **random coordinates** for the position of each enemy and **create** them a **controller at that position**.
* Then we **create the enemy snakes** **passing them the controller and coordinates**
* We **add the enemy to the array**

Now let us go to the **draw function** to **loop through the enemies and draw them**



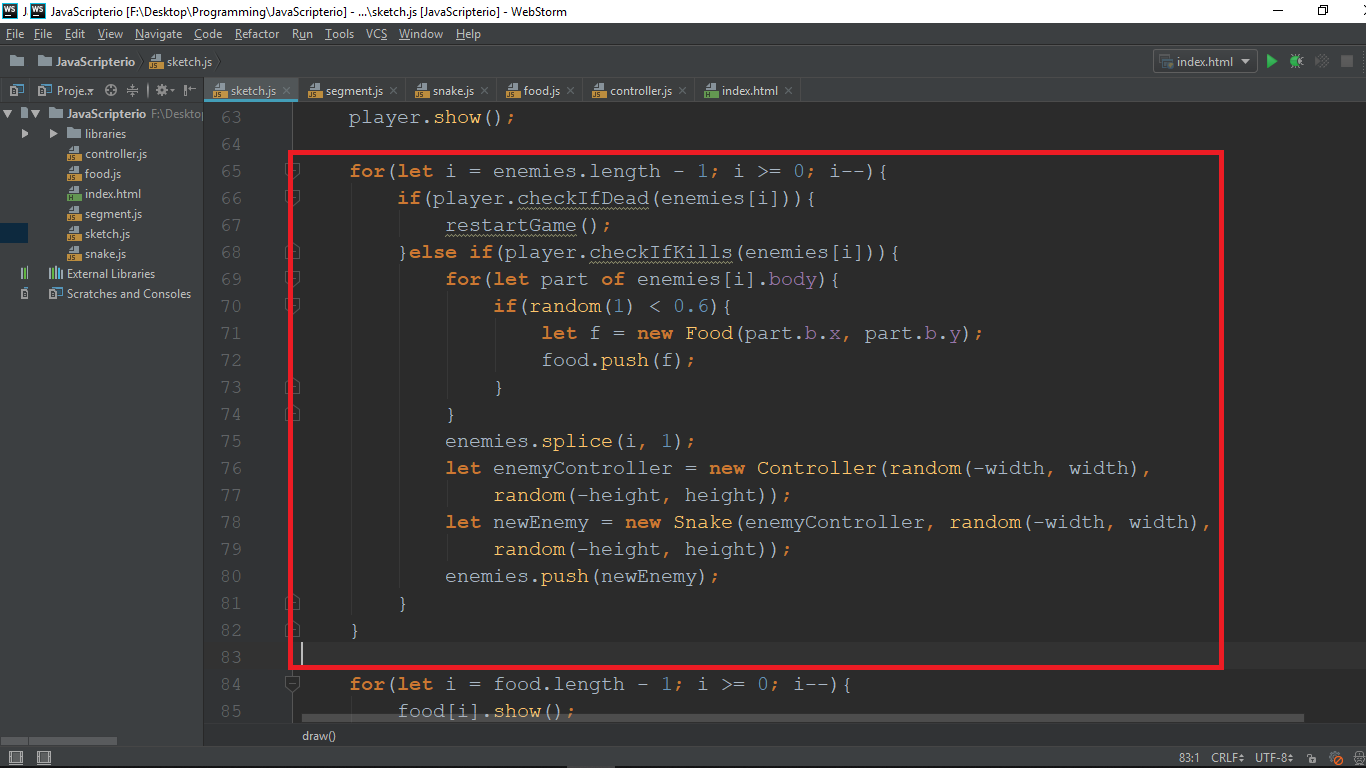
* As you might see, here we call a function called **wander**. We will **add that function to the controller class**. We need it, because the **enemy will not follow the mouse**, it **will follow random points.**

Go to the **controller class** and **add a wander function** with the following code:



* We will make **every 10% of the time** a **new random direction** that the enemy will follow (**instead of following the mouse**).
* We **set magnitude to one**. That will make the **enemy move slower than the player**.
* We **lerp it to make it turn smoothly**
* And finally we **add the velocity to the position**

The next step is to **check if player kills an enemy** or **if enemy kills player**. Add the following lines of code in the draw function:



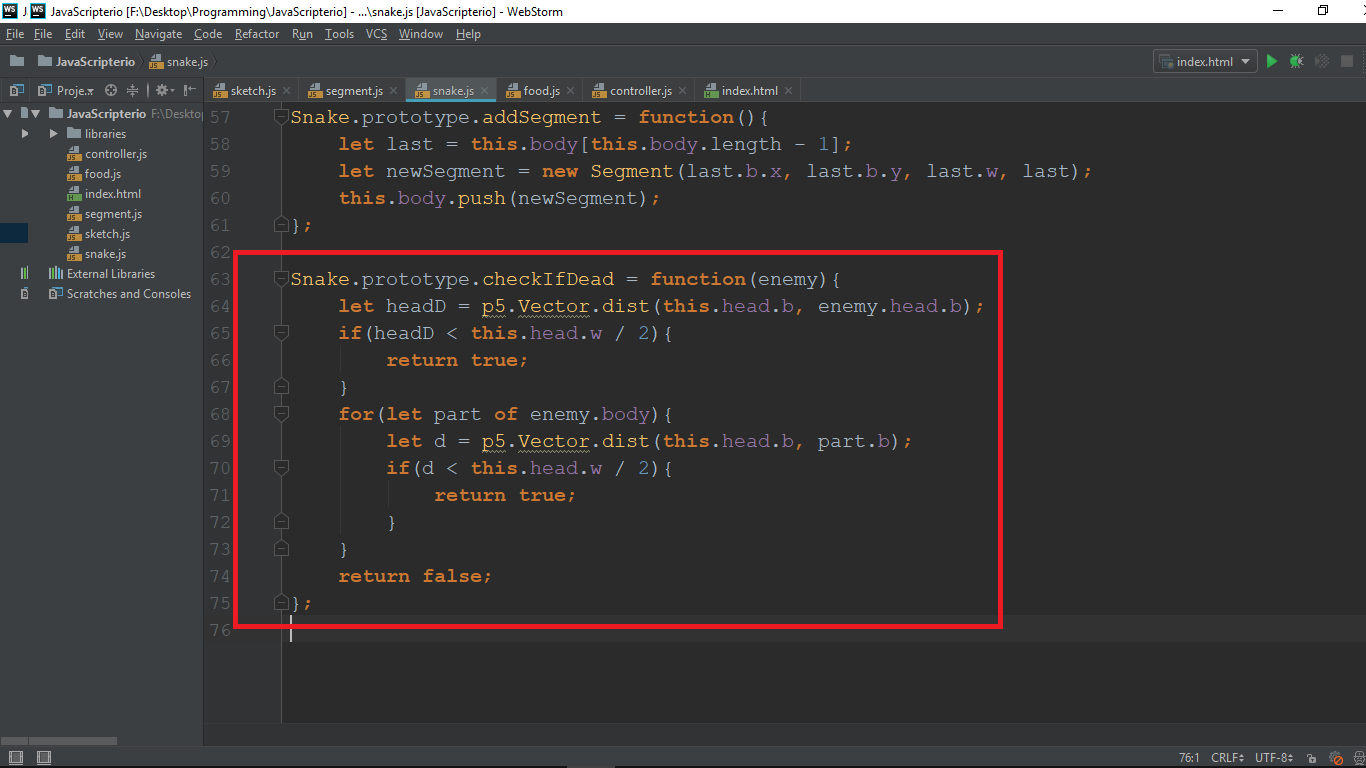
* First, we check **if the player dies by passing it the enemy** (we **will implement that in a bit**). If **the player dies**, we will **create a function to restart the game**.
* Then we check **if the player kills an enemy** by passing it. If it does, we **remove the enemy** (with the **splice function**) and **leave random pieces of food behind it**.
* Then we **create a new enemy with new enemy controller at a random position**.

Now we just need to implement the **checkIfKills, chekcIfDies** and **restartGame** functions and we are ready!

# End game, Killing enemies, Killing player

## checkIfDead function

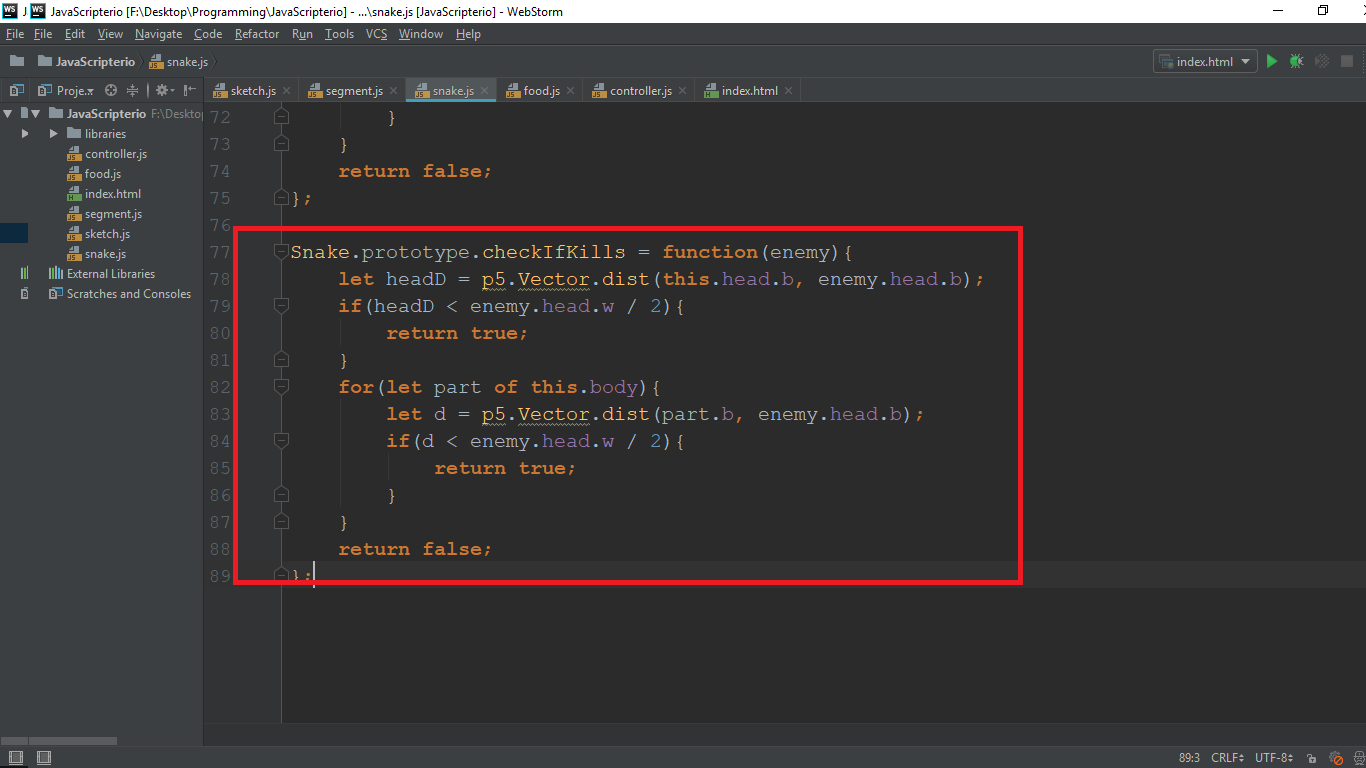
* Go to the **snake.js** and add that function with code:



* + the way we check if the player dies is **familiar to the function** checking if the **player eats the food** (we **calculate distance**)
  + In order for the player to be dead, we **need to hit** (**bite**) **an enemy**. So we **check the distance between the head of the player collides with the head of the enemy**, and then we **loop through the body of the enemy and check if player’s head collides with it**. (for collision we consider a distance less than the width of the head divided by 2)
  + If we **looped through all** of the parts of the body of the enemy it means **we didn’t have a collision so we return “false”**

## checkIfKills function

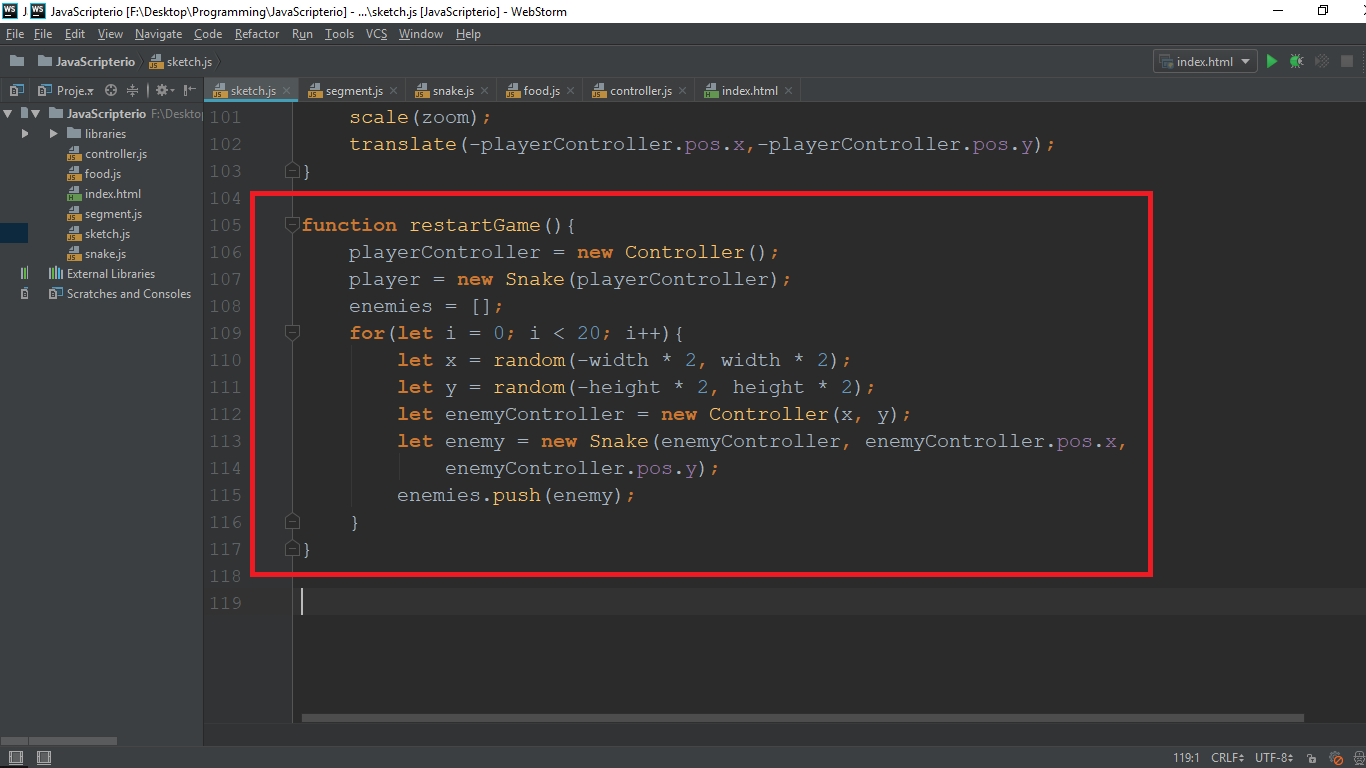
* Now **add** that function with the following code:



* + The function here is **like the previous** one but we **loop through the player’s body** and we **check if the enemy’s head collides with it**.

## restartGame function

* The restart game function will create new player at the center and new enemies. Go to the **sketch.js** and add the **restartGame** function with the following code:



* + We **create new player**, set the **enemies array to an empty array** and finally we **add new enemies on a random position**

**Save** and **reload** browser. **WE ARE DONE!**

