Topic	**FRUIT HANGING WITH ROPE**	
Class Description	**The student will learn to create a rope body using the matter.js library. The student will also attach a fruit to a rope using constraints.**	
Class	**C29**	
Class time	**45 mins**	
Goal	● Create a rope body. ● Attach the fruit with the rope.	
Resources Required	● Teacher Resources ○ VS Code Editor ○ Laptop with internet connectivity ○ Earphones with mic ○ Notebook and pen ● Student Resources ○ VS Code Editor ○ Laptop with internet connectivity ○ Earphones with mic ○ Notebook and pen	
Class structure	**Warm-Up Slide show option** **Teacher-Led Activity** **Student-Led Activity** **Wrap-Up Slide show option**	**10 mins** **10 mins** **20 mins** **5 mins**

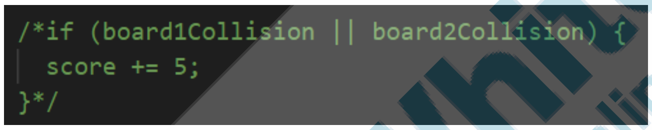
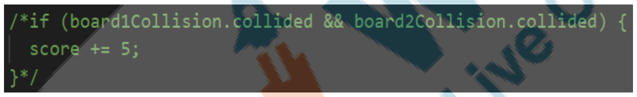
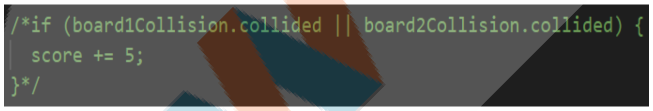
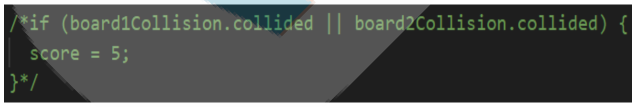
WARM-UP SESSION - 10 mins

Teacher starts slideshow from slides 1 to 9
Refer to speaker notes and follow the instructions on each slide.

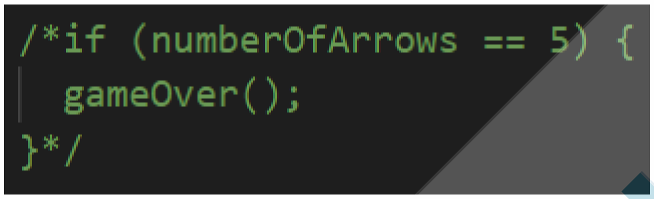
Teacher Action	**Student Action**

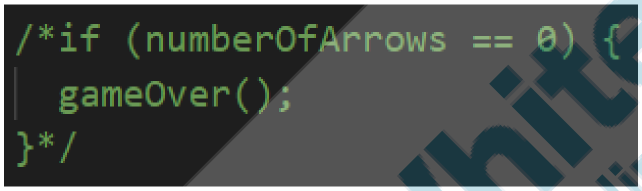
Hey <student's name>. How are you? It's great to see you! Are you excited to learn something new today? ***Run the presentation from slide 1 to slide 3.*** **The following are the warm-up session deliverables:** • **Greet the student.** • **Revision of previous class activity.** • **Quizzes**	**ESR**: Hi, thanks, Yes I am excited about it! Click on the slide show tab and present the slides

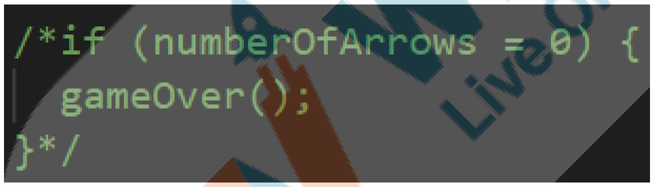
QnA Session	
Question	**Answer**

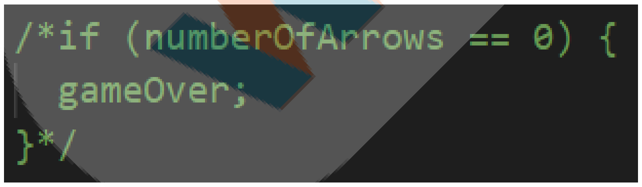
Select the correct option to write a condition which will check if arrows collide with the boards then increase the score by 5.
 	
A.  `/*if (board1Collision || board2Collision) {`
`score += 5;`
`}*/`
 	
B.  `/*if (board1Collision.collided && board2Collision.collided) {`
`score += 5;`
`}*/`
 	
C.  `/*if (board1Collision.collided || board2Collision.collided) {`
`score += 5;`
`}*/`
 	
D.  `/*if (board1Collision.collided || board2Collision.collided) {`
`score = 5;`
`}*/`
	C

Select the correct option to call the **gameOver()** function if the **numberOfArrows** is equal to zero.	**B**

A.

```
/*if (numberOfArrows == 5) {
  gameOver();
}*/
```

B.

```
/*if (numberOfArrows == 0) {
  gameOver();
}*/
```

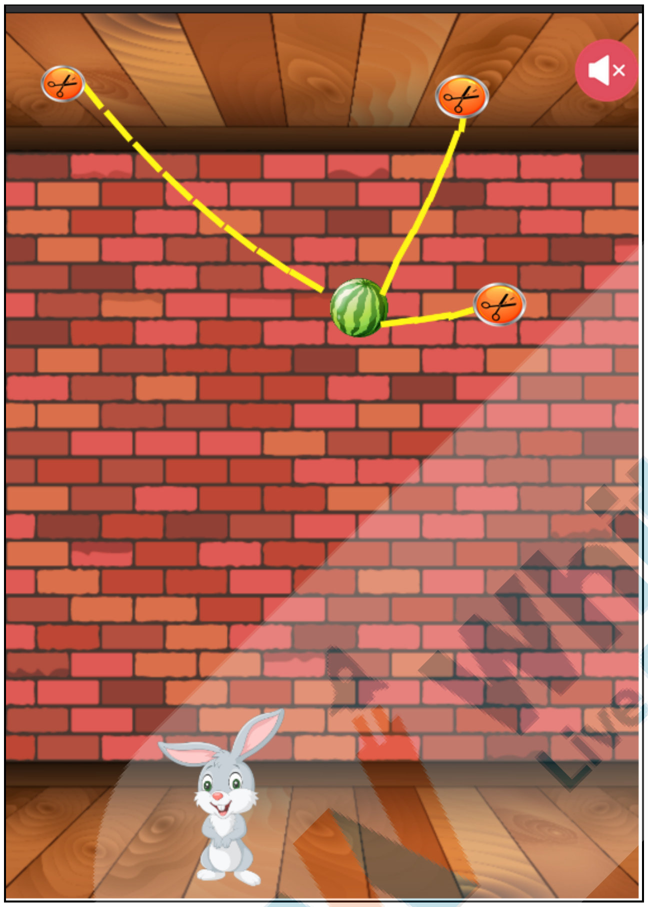
C.

```
/*if (numberOfArrows = 0) {
  gameOver();
}*/
```

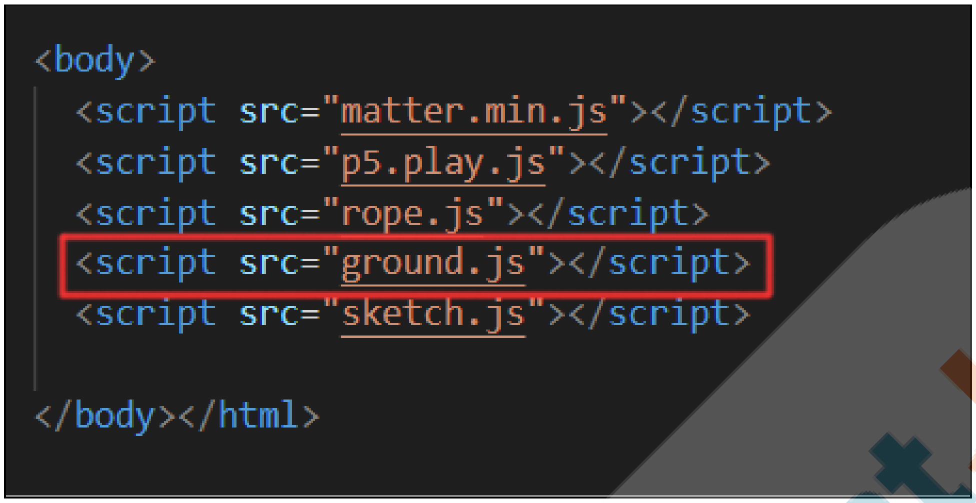
D.

```
/*if (numberOfArrows == 0) {
  gameOver;
}*/
```

Continue the warm-up session

Activity details	**Solution/Guidelines**

Run the presentation from slide 4 to slide 9 to set the problem statement. **Following are the warm-up session deliverables:** ● Appreciate the student on his performance in the quizzes. ● Create a Ground class.	Narrate the slides by using hand gestures and voice modulation methods to bring in more interest in students.
Teacher ends slideshow	
TEACHER-LED ACTIVITY - 10 mins	
Teacher Initiates Screen Share	
CHALLENGE ● **Create the ground class.**	
Teacher Action	**Student Action**
Teacher-Led Activity 1: *The teacher downloads Teacher Activity 1 and runs it in the VS Code Editor.* In the previous classes, we got started with the physics library **matter.js** and built a Pirate Invasion with it.	
From this class onwards, we are getting started with a new game, called **Feed the Bunny**! Before we get started with the code, let's see a quick overview of the game. *Note: Image to be presented from the Visual Aid.* In this game, we have a watermelon fruit hanging from the rope. The bunny is waiting at the bottom to eat the watermelon fruit. When the user clicks on the cut the rope button, the	

watermelon falls and the bunny eats it. As we have seen in the visual aid, we will have different arrangements of ropes in the further classes, in a way so that the user has to choose which rope to cut for the watermelon to land on the bunny.	
	
So let's get started with this game: Our Bunny will be standing on the ground. The first thing we need to do is to create the ground body and for that, we will create the ground class. ● Create a new file and name it **ground.js**. ● Now add this file in the **index.html,** so that we can use it in our code.	



```
<body>
  <script src="matter.min.js"></script>
  <script src="p5.play.js"></script>
  <script src="rope.js"></script>
  <script src="ground.js"></script>
  <script src="sketch.js"></script>

</body></html>
```

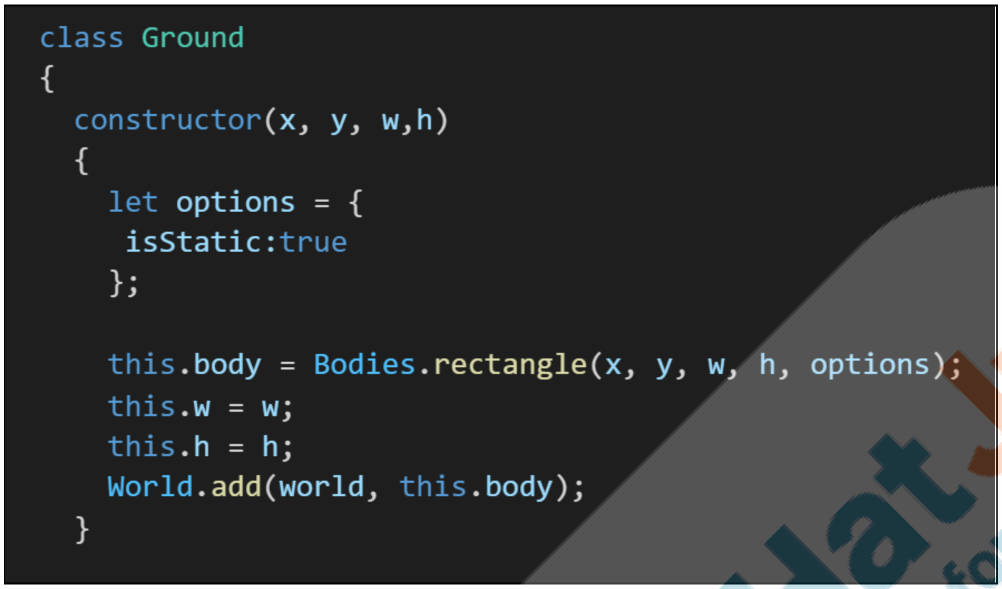
Open the **ground.js** file, here we are going to create the **Ground** class.

The ground will be a stationary body, so we only need to create the **constructor()** function and the function to display it.

In the constructor, we will specify the **x**, **y** positions, and the **width** and **height**, which will be entered by the user while creating the **ground** object for the **ground()** class.

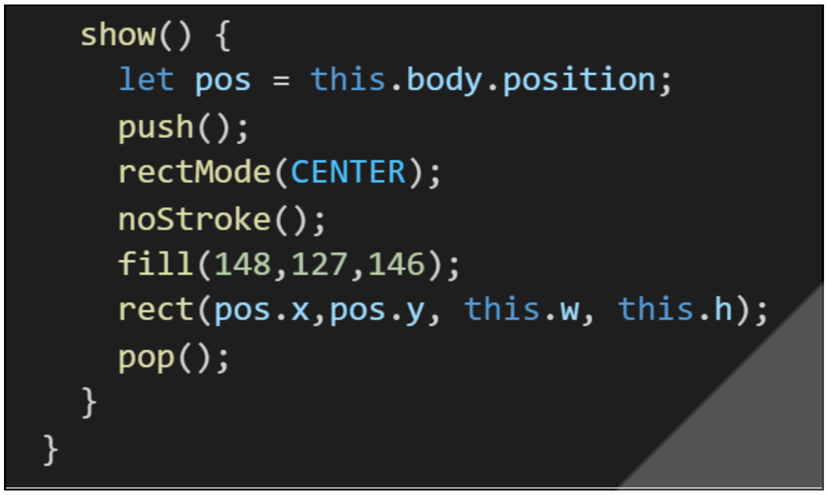
*Note: **Flags** are variables that we can use to change the state of an object, flags have only boolean values either **true** or **false**.*

For the **options**, we are only keeping the **isStatic** flag to be **true**, because we don't want the ground to move.

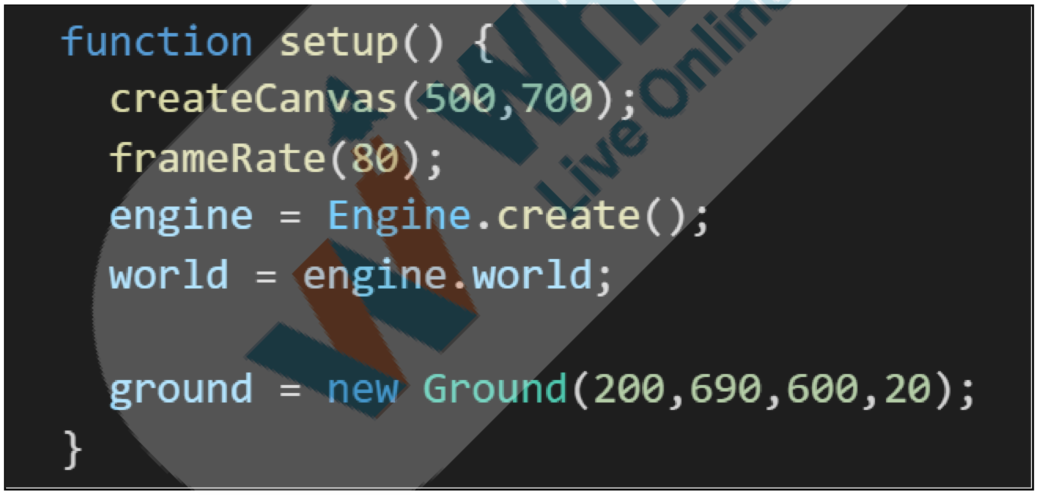


```
class Ground
{
  constructor(x, y, w,h)
  {
    let options = {
     isStatic:true
    };

    this.body = Bodies.rectangle(x, y, w, h, options);
    this.w = w;
    this.h = h;
    World.add(world, this.body);
  }
```

The ground is a rectangular body, that is why we are using the **Bodies.rectangle()** function to create this body. Once the body creation is done, we will add this body to the world.	
The **constructor()** function will only allow us to create the body, but if we want to show the ground in the game, then we need to write a function to display it. Create a new method **show()** within the **Ground** class. To draw a body, we need its **x** and **y** positions. Can you tell me how can we get the x and y positions of the body in the class? Great! We store the position value of the ground in a variable i.e. **pos**.	**ESR:** By using **this.body.position**.

 ``` show() { let pos = this.body.position; push(); rectMode(CENTER); noStroke(); fill(148,127,146); rect(pos.x,pos.y, this.w, this.h); pop(); } }```	
When we create a rectangle in p5, it takes the top-left point as the origin of the rectangle. But in order to change the origin point of the rectangle from the top left corner to the center, we use the **rectMode()** function. In this function, we pass **CENTER** as an argument. This will create a rectangle with the center point as origin. As we want to have a specific color and stroke to the ground, which is why we are using the **push()** function, and after this will add the color and stroke properties of the ground. **noStroke()** is to have no outline around the body. We use the **fill()** function to add color to a **shape.RGB** code of the color is passed to the **fill()** function as an argument. Then we create the shape of the ground, which is a rectangle using the **rect()** function. And then we write the **pop()** function. This will end our shape creation. Our **ground** class is complete, now we can create the object of the class. Can you tell me where we are going to do that?	**ESR:** Yes! In the **sketch.js** file.

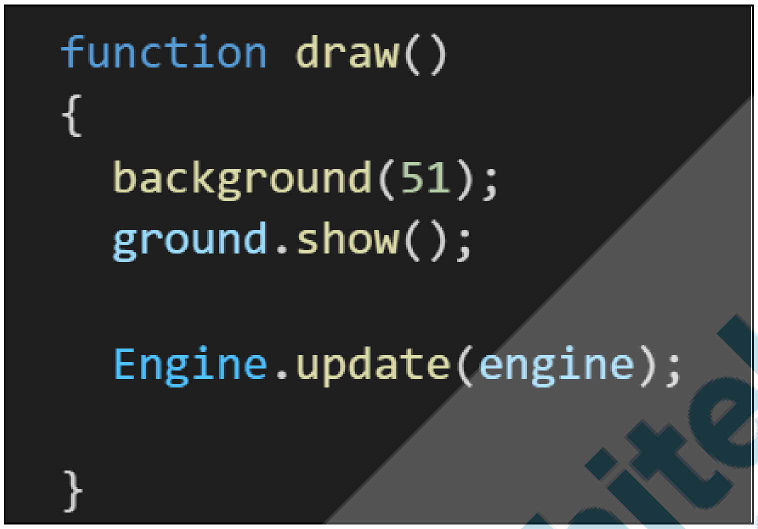
Very good! First, declare a variable as **ground**. Then in the **setup()** function create the object of the ground class and store that in this variable. Do you know how to create the object of the class? Great, let's write it. While creating a **ground** object, we need to provide the x, y positions and the width, height of the ground. Our game will be in portrait mode. That is why our canvas size is **(500,700)**. We need to keep this in mind while setting the ground position and width. X position is **200**, y is **690**, the width of the ground body is **200** and the height is **20**.	 **ESR**: By using the **new** keyword and **class name**.

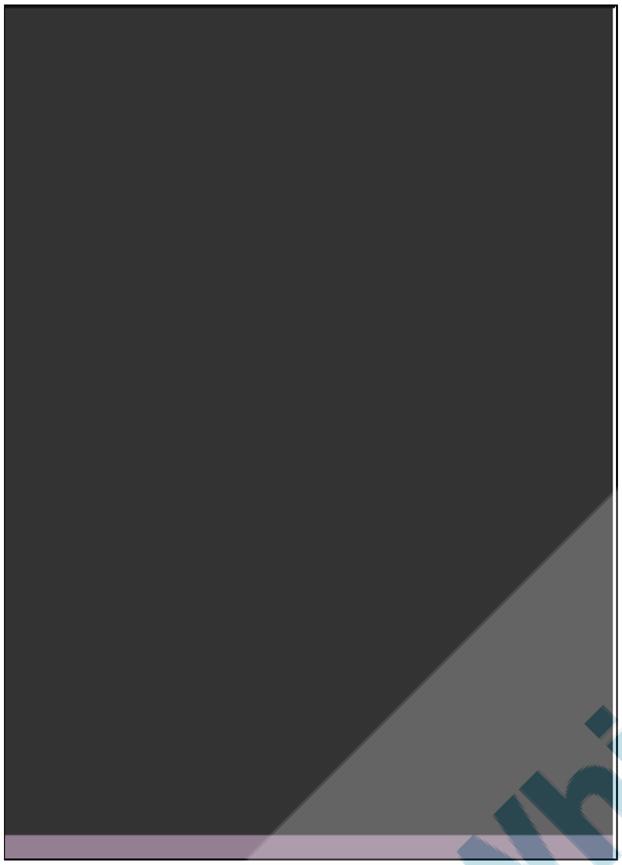


```
function setup() {
  createCanvas(500,700);
  frameRate(80);
  engine = Engine.create();
  world = engine.world;

  ground = new Ground(200,690,600,20);
}
```

Our object creation is now complete. Now we can display the ground.	
How do we display the ground?	**ESR:** By calling the **ground.show()** function.

Great, we do that in the **draw()** function. Now we can run our code. Click on the **Go Live** button and a browser window will open where we can see our canvas and ground.	


`function draw()`
`{`
`  background(51);`
`  ground.show();`
 	
`  Engine.update(engine);`
 	
`}`
 	
Output:	



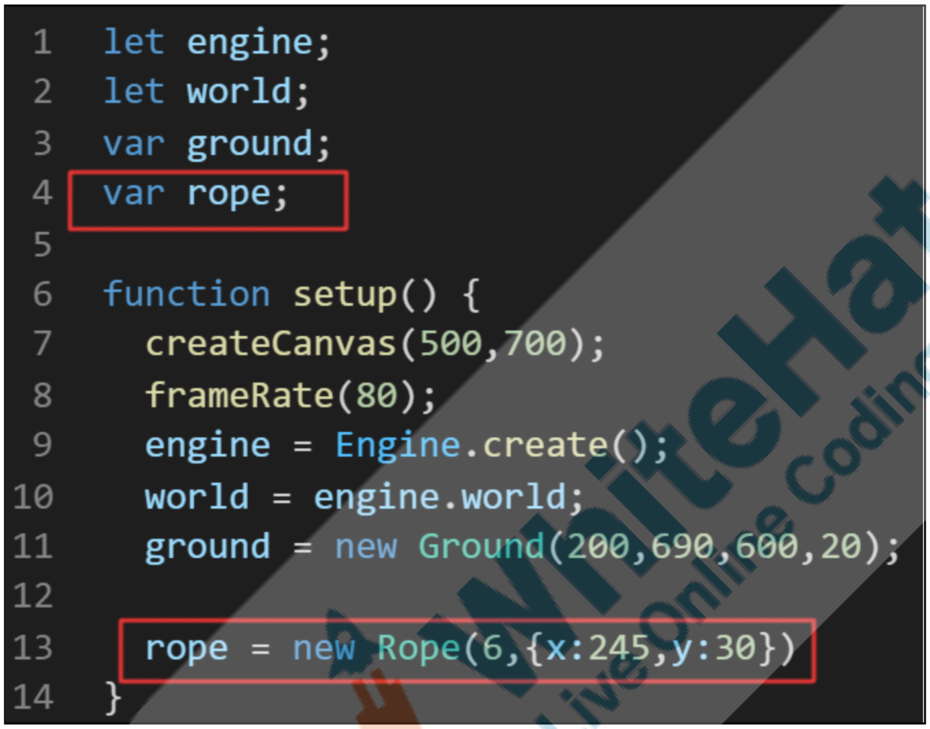
The first step is complete. Here we have created a **ground** class and **ground** object. Now we can see the ground body on the canvas. In the next step, you need to create the rope and the fruit body. Now it's your turn to write the code to add the rope and the fruit.	

Teacher starts slideshow : Slide 10 to Slide 18

Run the presentation slide to set the student activity context. ● Create an object of the Rope class. ● Explain the Composite module. ● Hang fruit to the Rope.	

Teacher ends slideshow	
Teacher Stops Screen Share	
STUDENT-LED ACTIVITY - 20mins	
● **Ask Student to press ESC key to come back to panel** ● **Guide Student to start Screen Share** ● **Teacher gets into Fullscreen**	
ACTIVITY ● **Create a chain and hang it on the canvas.** ● **Create the fruit body and attach it with rope.**	
Teacher Action	**Student Action**
Step: 2 - Student led Activity Now we need to create the rope object and hang it on a point in the canvas, then we will create a fruit body and attach it with the rope. We will also add background images and images for the fruit to make our game look visually appealing. If you notice we already have a file named **rope.js** which contains the code to create a rope. If you see the code for the **rope.js** .	**ESR:** *The student downloads the [Student Activity 1](#) code and runs it in the VS Code editor.*

• In the **constructor()** we are taking two parameters **nlinks** - number of links and **pointA** - points of connection. Using **Composites.stack()** function we create the multiple rectangular bodies and store it in the **rect** variable. Using the **Composites.chain()** function we create the chain of the rectangles. And then using the **Constraints.create()** we add the constraints to the chain which connects all the bodies of the chain together like we have string in a necklace. We have the **break()** function which helps us to break the chain.It simply makes the rope body null. *Teacher can check details on composites on* *https://brm.io/matter-js/docs/classes/Composites.html* For our code we are only going to use the **rope.js** to create the rope and **break()** function to break the rope when the user clicks on the cut button(which will be added in upcoming classes). Add **rope.js** in **index.html** To create a rope we need two pieces of important information, first is how long our rope will be, and where we want to hang our rope. To define the length of the rope, we define a number that creates those many sections in the rope. One section essentially is a rectangle, so you can imagine our rope to be multiple rectangles connected together. Next, we need the point where we are going to hang the rope, which will be a certain x and y position on the canvas. First, you define a variable named **rope**.	*The student adds **rope.js** in **index.html**.*

Then in the **setup()** function create an object from the **Rope** class. While creating the object first we write the length of the rope which we will set as **6**, and then the x and y position, x would be **245** and y as **30**.	
	
The next step is to call the **rope.show()** function in the **draw()** function so that we can see the rope hanging. Now comes the fun part to create the fruit body and hang it with our rope. Any idea how we can do it? First, let's create a fruit body. Declare a variable as **var fruit;**	**ESR**: Varied answers.

```
let engine;
let world;
var ground;
var rope;

function setup() {
  createCanvas(500,700);
  frameRate(80);
  engine = Engine.create();
  world = engine.world;
  ground = new Ground(200,690,600,20);

  rope = new Rope(6,{x:245,y:30})
}
```

Then in the setup function, we first create **fruit_options** for the fruit body. In the options, we are only defining the density of the fruit as **0.001**.

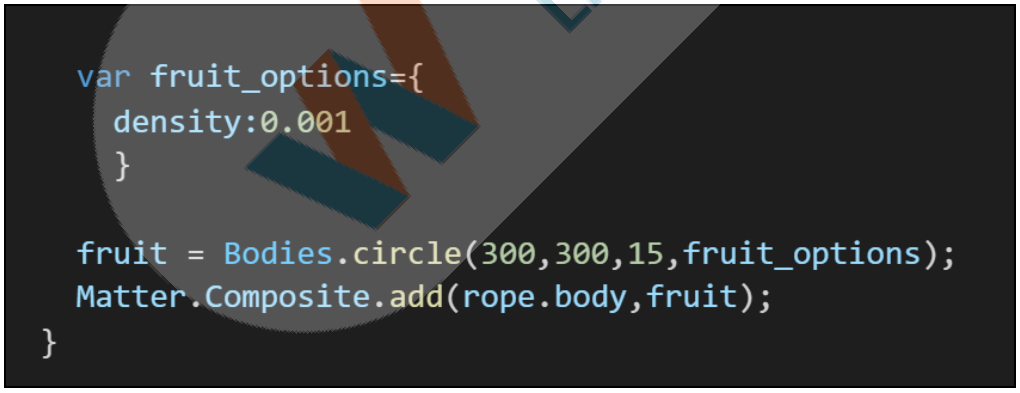
Then we create a fruit body using the **Bodies.circle()** function.
In the function, we need to provide the x and y positions and the radius of the fruit along with **fruit_options**.

Once we create the body, we need to add this body in the composite, now what is a composite?

As you have seen in the visuals, a composite consists of multiple bodies within it. When we want multiple bodies to have the same properties such as shape and size and behave in a certain manner, we make a composite of these bodies.

In our game, the rope we are creating is made up of multiple rectangles, hence we call it a **composite**. But we also have to add our fruit in the same composite.

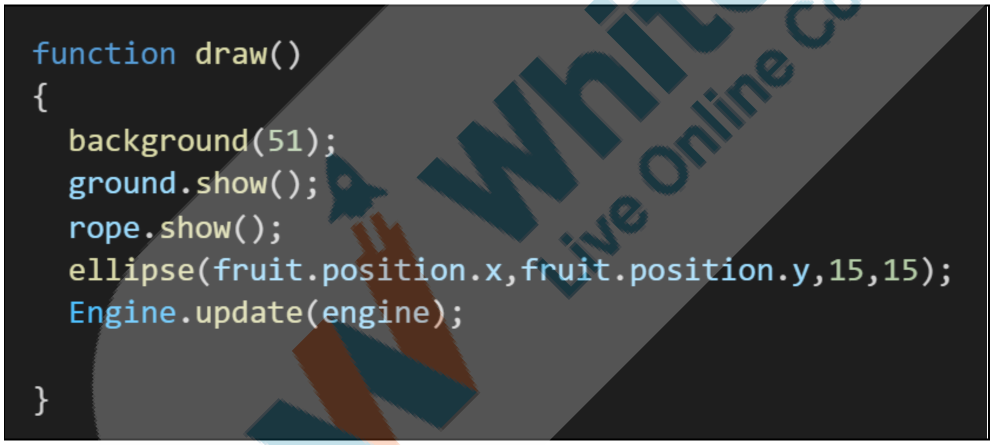
To add a body to the composite, we use the function, **Matter.Composite.add(name_of_composite, body_to_add**).



```
    var fruit_options={
      density:0.001
      }

    fruit = Bodies.circle(300,300,15,fruit_options);
    Matter.Composite.add(rope.body,fruit);
  }
```

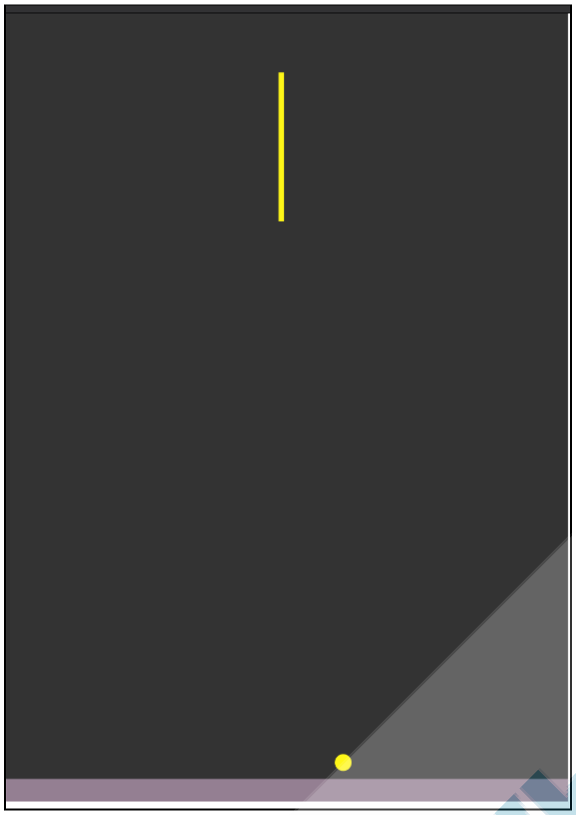
Here the **composite** is a body of the **Rope** class, and we are adding the fruit to it.

Now we have the fruit body in the composite, but we need to display this fruit as well. To display this body, we will create a circle, using the **ellipse** function in the **draw()** function. For the x and y positions of the circle, we will pass the x and y position values from the fruit body. When you run the code, the fruit will appear at the position defined and will fall on the ground. Click on the **Go live** button to see the output. Because we have not added any constraint between the fruit body and the rope. They are part of the same composite but not attached to each other with any constraint.	*The student runs the code and observes the output.*



```
function draw()
{
  background(51);
  ground.show();
  rope.show();
  ellipse(fruit.position.x,fruit.position.y,15,15);
  Engine.update(engine);

}
```

Output:



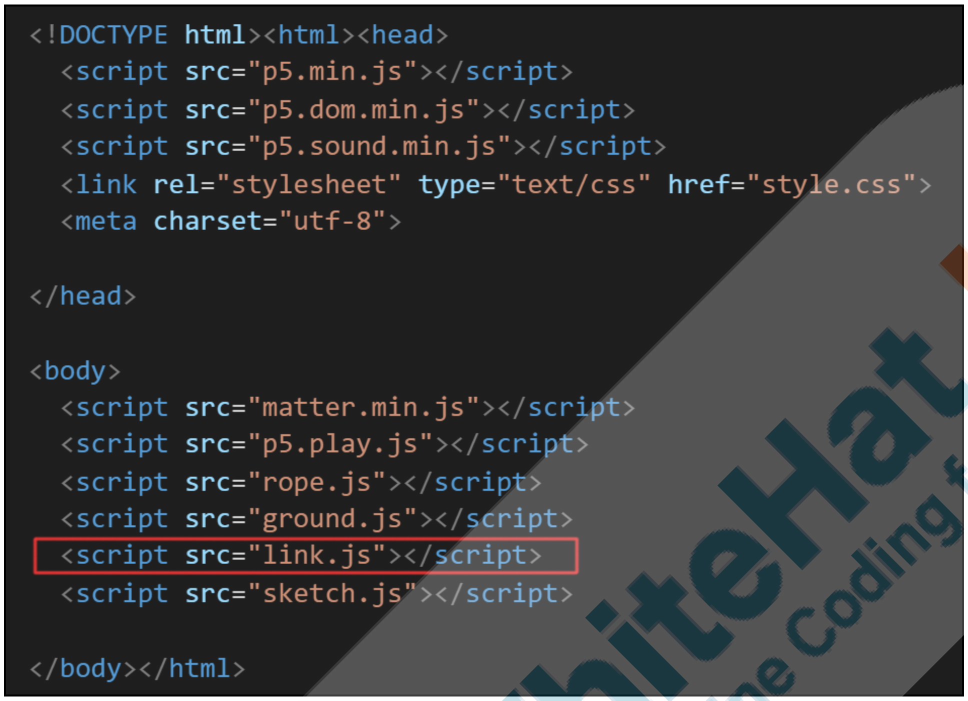
Now we need to write the code to hang the fruit with our rope.

For this, we are going to use constraints.
If you remember our rope is made up of various rectangles to be more specific **6** rectangles, which we specified while creating the rope object.
We will create a constraint between the fruit body and the last rectangle of our rope.

Let’s code it!

We create the class for the constraint because in the future classes we will hang the fruit with multiple ropes.

To make our code clean, we will keep all the code related to creating the constraint in the **link.js** file.

Create a new file named **link.js** and add this to the **html** file as source.	*The student creates a new file and names it **link.js.***
	
The constraint is going to be between 2 bodies, the last body (rectangle) of the rope and the fruit body. That is why in the link class when we write the constructor we need to keep 2 parameters as **bodyA** and **bodyB**. We want to connect the fruit body at the last rectangle of the rope. So we will create a variable to get the index of the last rectangle**(or element, composite can be assumed like an array)** That will be **var lastlink = bodyA.body.bodies.length-2** The last element will be **2** less than the length because the index always starts from **0**, and we also added fruit in the composite that increases the length **1**. So to get the last element index we need to subtract the 2 from the length. So the function **bodyA** is **rope.body.bodies[lastlink]**.	

```
<!DOCTYPE html><html><head>
  <script src="p5.min.js"></script>
  <script src="p5.dom.min.js"></script>
  <script src="p5.sound.min.js"></script>
  <link rel="stylesheet" type="text/css" href="style.css">
  <meta charset="utf-8">

</head>

<body>
  <script src="matter.min.js"></script>
  <script src="p5.play.js"></script>
  <script src="rope.js"></script>
  <script src="ground.js"></script>
  <script src="link.js"></script>
  <script src="sketch.js"></script>

</body></html>
```

bodyB is the fruit.

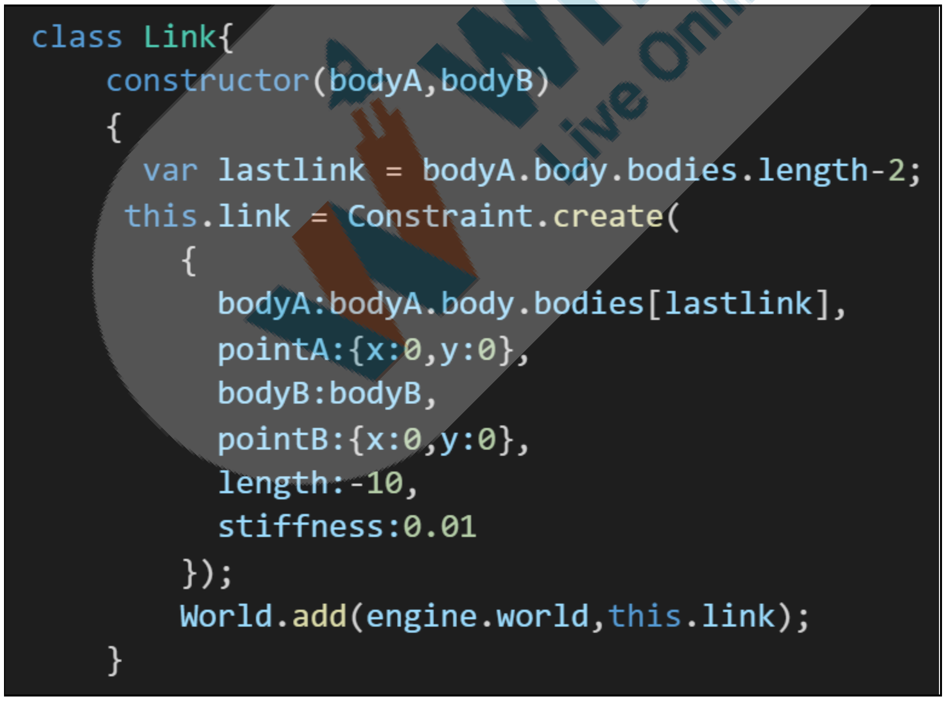
We will create the constraint using the **createConstraint()** function of the **matter.js** library.

To connect 2 bodies we also need to specify at which point, on the body of the constraint, is it going to be connected.

For both bodies or we can say **pointA** and **pointB,** which are the **X and Y positions** so, we will write these as **x:0** and **y:0**.

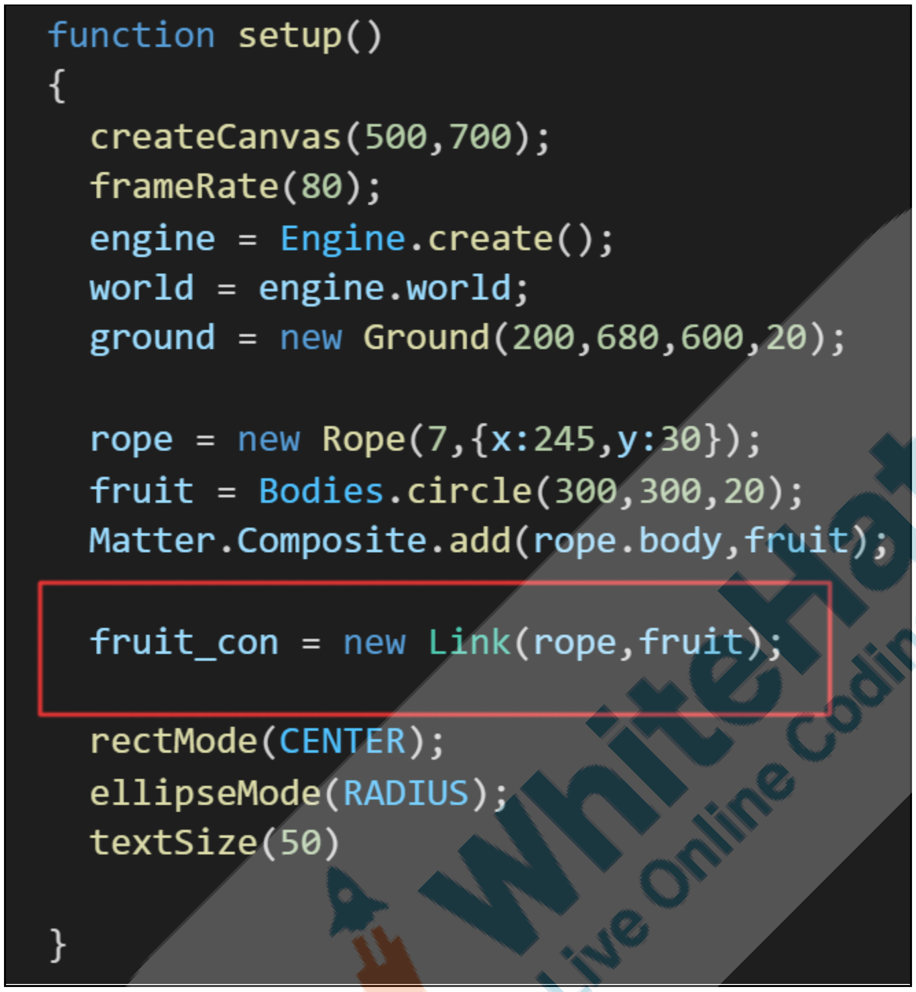
We need to define two more parameters:

- The length of the constraint, which we also keep **0** for now.
- The stiffness, which we write as **0.01**. This value of stiffness will prevent vibration of the constraint (Here vibration refers to the small movements in the rope). So that our constraint stays stable.



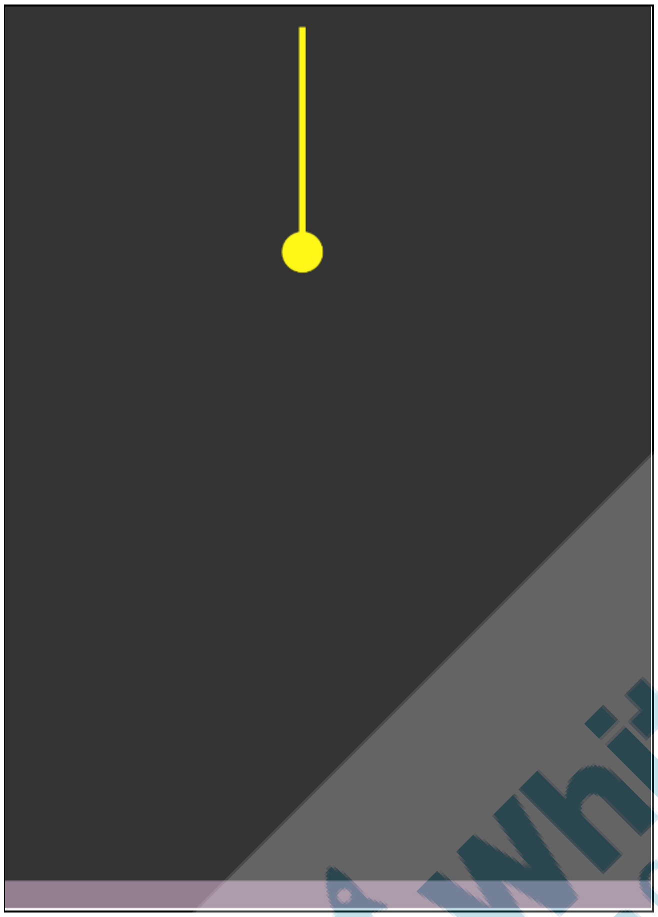
```
class Link{
    constructor(bodyA,bodyB)
    {
      var lastlink = bodyA.body.bodies.length-2;
     this.link = Constraint.create(
        {
          bodyA:bodyA.body.bodies[lastlink],
          pointA:{x:0,y:0},
          bodyB:bodyB,
          pointB:{x:0,y:0},
          length:-10,
          stiffness:0.01
        });
        World.add(engine.world,this.link);
    }
```

Once the constraint is complete, we will add this to our world. But before we run the code, we need to create the object of the **link()** class in the **sketch.js** file and pass the 2 bodies. To create the object, first declare a variable as **var fruit_con;** In the setup function we will create the object and assign it to the **fruit_con** variable; While creating the object of the class, we need to pass the rope and the fruit as the parameters in the **Link()** class object.	



```
function setup()
{
  createCanvas(500,700);
  frameRate(80);
  engine = Engine.create();
  world = engine.world;
  ground = new Ground(200,680,600,20);

  rope = new Rope(7,{x:245,y:30});
  fruit = Bodies.circle(300,300,20);
  Matter.Composite.add(rope.body,fruit);

  fruit_con = new Link(rope,fruit);

  rectMode(CENTER);
  ellipseMode(RADIUS);
  textSize(50)

}
```

Now run the code and see the output. Isn't it amazing?	*The student runs the code and observes the output.*



We are done for today. We created the ground class and the fruit body, and we added a rope to our canvas while attaching the fruit body with it. In the next class, we are going to add the ability to break the rope and have the fruit fall. We will also create our bunny.	

Teacher Guides Student to Stop Screen Share

WRAP-UP SESSION - 5 Mins

Teacher starts slideshow from slide 19 to slide 28	
Activity details	**Solution/Guidelines**
Run the presentation from slide 19 to slide 28. **Following are the wrap-up session deliverables:** ● **Appreciate the student.** ● **Revise the current class activities.** ● **Discuss the quizzes.**	
Quiz time - Click on in-class quiz	
Question	**Answer**
To make the ground a stationary body, which of the following properties is used? A. isStatic:true B. isStatic:false C. isStatic = true D. isStatic = false	**A**
To stop a body from having an outline which function is used? A. noStoke = true B. stroke(null) C. Stroke = null D. noStroke()	**D**
Which of the following is true about a composite? A. We need to add a body to the composite to apply its properties. B. A composite consists of multiple bodies within it. C. When we want multiple bodies to have the same properties such as shape and size and behave in a certain manner, we make a composite of these bodies.	**D**

D. All of the above		
End the quiz panel		
FEEDBACK ● **Encourage the student to make reflection notes in Markdown format.** ● **Compliment the student for her/his effort in the class.** ● **Review the content of the lesson.**		
*** This Project will take only 30 mins to complete. Motivate students to try and finish it immediately after the class.** **Project Overview** **CRUSH THE ZOMBIES** **Goal of the Project:** In this project, we are going to create the stone, base, and link class. Using the constraints, join the bridge with the joints and stack it with the stones. **Story**: A far away village is always troubled by the zombie. The only way to kill the zombie is to drop a stone in its head. You have mostly seen that the zombie travels under the bridge to get to the village, so you plan to stack the bridge with the stones and drop it on the zombie when it comes under the bridge. I am very excited to see your project solution and I know you will do really well. Bye Bye!		*The student engages with the teacher over the project.*
	You get hats off.	Make sure you have given at least 2 Hats Off during

	See you in the next class then.	the class for:
Teacher ends slideshow		
Teacher Clicks		
ADDITIONAL ACTIVITIES		
Additional Activities *Encourage the student to write reflection notes in their reflection journal using Markdown.* Use these as guiding questions: ● What happened today? ○ Describe what happened. ○ The code I wrote. ● How did I feel after the class? ● What have I learned about programming and developing games? ● What aspects of the class helped me? What did I find difficult?		*The student uses the markdown editor to write her/his reflections in the reflection journal.*

Links:

Activity Name	Description	Link
Teacher Activity 1	Boilerplate code	https://github.com/pro-whitehatjr/C29_TA1
Student Activity 1	BoilerPlate code	https://github.com/pro-whitehatjr/C29_SA1
Teacher Activity 2	Reference Code	https://github.com/pro-whitehatjr/c29_complete_code
Project Solution	Crush the Zombies	https://github.com/whitehatjr/zombie-crush-1
Teacher Reference visual aid link	Visual aid link	https://curriculum.whitehatjr.com/Visual+Project+Asset/ADV_VA/BJFC-PRO-V3-C29-withcues.html
Teacher Reference In-class quiz	In-class quiz	https://s3-whjr-curriculum-uploads.whjr.online/b2c0dc9d-eaa0-4a92-a9b6-cb04fdae9bc7.pdf
Project Solution	Crush The Zombies	https://github.com/whitehatjr/zombie-crush-1