




Topic	BASICS OF THE PHYSICS ENGINE	
Class Description	The student will learn to create various kinds of shapes and will also learn to add the physics properties to these objects while observing their behavior.	
Class	C20	
Class time	45 mins	
Goal	<ul style="list-style-type: none"> • Create various kinds of shapes. • Apply physics to these objects and observe their behavior. 	
Resources Required	<ul style="list-style-type: none"> • Teacher Resources <ul style="list-style-type: none"> ○ Laptop with internet connectivity ○ Earphones with mic ○ Notebook and pen ○ VS Code • Student Resources <ul style="list-style-type: none"> ○ Laptop with internet connectivity ○ Earphones with mic ○ Notebook and pen ○ VS Code 	
Class structure	Warm-Up Teacher-led Activity Student-led Activity Wrap-Up	5 mins 15 mins 20 mins 5 mins
WARM-UP SESSION - 5 mins		
<div>  </div> <p>Teacher starts slideshow from slides 1 to 19</p> <p>Refer to speaker notes and follow the instructions on each slide.</p>		
Activity details		Solution/Guidelines
<p><i>Hey <student name>. How are you? It's great to see you!</i></p> <p><i>Are you excited to learn something new today?</i></p> <p>Run the presentation from slide 1 to slide 3.</p> <p>The following are the warm-up session deliverables:</p> <ul style="list-style-type: none"> • Connecting students to the previous class. 		<p>ESR: Hi, thanks, yes I am excited about it!</p> <p>Click on the slide show tab and present the slides.</p>

- Warm-up Quiz.

QnA Session

Question	Answer
<p>Select the correct code block to make the tower reset again and again on the Y-axis.</p> <p>loading...</p>  <p>A.</p> <pre>if(tower.y > 400){ tower.y = 300 }</pre> <p>B.</p> <pre>if(tower.x > 400){ tower.y = 300 }</pre> <p>C.</p> <pre>if(tower.x > 400){ tower.y = 300 }</pre>	<p>A</p>

<p>D.</p> <pre>if(tower.x > 400){ tower.x = 300 }</pre>	
<p>What does the following code do?</p> <pre>if(invisibleBlockGroup.isTouching(ghost) ghost.y > 600){ ghost.destroy(); gameState = "end" }</pre> <p>A. It destroys the door, when the ghost is touching the invisible block group.</p> <p>B. It destroys the invisible block group, when the ghost is touching them.</p> <p>C. It destroys the ghost, when the ghost is touching the door.</p> <p>D. It destroys the ghost, when the ghost is touching the invisible block group.</p>	<p>D</p>
Continue the warm-up session	
Activity details	Solution/Guidelines
<p>Run the presentation from slide 4 to slide 19 to set the problem statement.</p> <p>The following is the warm-up session deliverables:</p> <ul style="list-style-type: none"> • Introduction to the Physics Engine. 	<p>Narrate the slides by using hand gestures and voice modulation methods to bring in more interest in students.</p>
<div>  Teacher ends slideshow </div>	
TEACHER-LED ACTIVITY - 15 mins	
Teacher Initiates Screen Share	
Teacher Action	Student Action
<p>Step 2:</p> <p>Teacher-led Activity</p>	<p><i>The student observes and learns.</i></p>

*The teacher downloads the code from GitHub [Teacher Activity 1](#) unzips and saves it as **Class20**, and opens this code in VSC.*

As you can see, we are given one new file named **matter.min.js**.

It is a library file that allows our JavaScript code to use 2D physics engine properties.

When we use this library in our project folder, we'll also need to add it to the **index.html** file so that we can use its functionalities in our code.

We'll add the physics engine library called **matter.min.js** in **index.html**.

*The teacher adds the **mattermin.js** library available in the given folder in the **index.html** file.*

The student Observes

```
<!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="sketch.js"></script>

</body></html>
```

In order to make use of this library, we need to import the necessary modules in our **sketch.js**. We import these modules from the **matter.min.js** library.

The student observes and learns.

<p>To use the physics engine, we will use three objects created in the Matter.min.js library -> World, Engine and Bodies.</p> <p>Since these objects are a part of the Matter.min.js library, they are referenced as Matter.World, Matter.Engine and Matter.Bodies.</p> <p>Let's rename them as World, Engine, and Bodies so that it is easy to write and read our code. This is called namespaceing in programming.</p> <p><i>The teacher writes code to namespace Matter.World, Matter.Engine and Matter.Bodies.</i></p> <p>Here const is just like var - except you cannot change the value stored in a const anywhere in your program.</p> <ul style="list-style-type: none"> • The engine is used to create the physics engine. • World is used to create the physical world and add objects to it. • Bodies are used to create the physical objects which inhabit the world. <p><i>The teacher codes to import the modules in the sketch.js file.</i></p>	
<p>In the sketch.js file, make the following changes:</p> <pre>const Engine = Matter.Engine; const World = Matter.World; const Bodies = Matter.Bodies; const Body = Matter.Body;</pre>	
<p>We will also create a canvas of 400,400.</p> <p>Let us create our engine for our code. As soon as you create a physics engine, a new world is created using the physics engine.</p>	<p><i>The student observes and learns.</i></p>

Remember, this is an artificial world, where we can add different bodies and these bodies will have physical properties such as weight, velocity, position, etc.

This is very similar to the world we live in, such as you are in the world, and you have weight, height, and all the other such properties.

If someone bumps into you, you will move, if you jump you will come back down. All these rules are applied to all the objects, people, and animals around you.

These are called bodies and all bodies follow certain rules of the world.

Using the **matter.min.js** library we are creating an artificial world where we will add bodies and assign different behaviors to them so that it looks realistic.

Create a **canvas**, **engine**, and a **world** inside the function **setup()** in the **sketch.js** file.

```
function setup() {
  createCanvas(400,400);
  engine = Engine.create();
  world = engine.world;
}
```

Now let's create the ball.

Can you tell me what the shape of a ball is?

And, what are the properties of a ball?

Let's check in the **Bodies** module if we have any function which can help us to create a ball.

ESR:

The shape of the ball is a circle.

ESR:

The ball has a radius, material, color, bounciness, etc.

Correct, for our game which properties do we need to consider?

*The teacher opens the document from [Teacher Activity 2](#) and checks the **Bodies.circle()** function.*

Here we have a **Bodies.circle()** function that will help us to create the circle body. And to add the body to the world we'll use the **World.add()** and pass the **world** and **ball** to it.

This is very important to understand that we can create multiple worlds as well.

Remember the example, in our solar system, where we have different planets and on each planet, we have different sets of rules for example on mercury there is no air, so there will be no air friction. The force of gravity is very high on Jupiter. On mars, the atmosphere is very thin.

That is why when we add the body to the world we need to specify to which world we are adding, since we have created a single world in our code, we will add our ball to that world itself.

Create a ball body using the **Bodies.circle()** function and add the **ball** body to the **world**.

We can also add physics properties to the ball. To add the properties, we'll create a variable called **ball_options**. This object will store properties such as **restitution** and **frictionAir**.

Restitution means how bouncy the ball is, the more the restitution the more bouncy the ball will be.

Whereas **frictionAir** is the friction due to the air, more the friction slower the ball will move in the world.

ESR: The position of a ball on canvas, and its radius.

We can have more properties like **weight**, **stiffness**, **drag**, etc., but we will explore them in the future classes.

After defining these properties, we pass this object to the **Bodies.circle()** function.

Write the code inside **setup()** of **sketch.js** to create a variable **ball_options** and a **ball** object.

```
var ball_options = {
  restitution: 0.95,
  frictionAir: 0.01
}
```

```
ball = Bodies.circle(100,10,20,ball_options);
World.add(world,ball);
```

We have created the ball, but we still can't see it on the screen.

Can you tell me why we still can't see the ball?

We still can't see the ball because Physics Engine bodies do not get displayed on their own, it requires JavaScript objects or images to be placed on it. To display this ball, we will use an **ellipse()** function in the **draw()** function.

In this function, we'll pass the **x** position, **y** position, and **radius** of the ball created using **Bodies.circle()**.

We'll also need to continuously update the physics engine with the new changes. It is a necessary step as the function **draw()** is running every frame, so we need to update the engine to reflect changes as well as every frame.

To do so we'll write **Engine.update(engine)**.

ESR:
Varied.

The teacher codes to update the engine and draw an ellipse.

The x & y position is referred as **ball.position.x** and **ball.position.y**
draw() function in **sketch.js**.

So in the **ellipse()** function, **ball.position.x** will be = **100** and **ball.position.y** = **10** as given above in **Bodies.circle()**.

The teacher runs the below code and shows the output.

```
function draw()
{
  background(51);
  Engine.update(engine);

  ellipse(ball.position.x,ball.position.y,20);
}
```

Output:



<p>Can you tell me what's happening here?</p> <p>To stop this from happening, let's create a ground below so that when the ball falls it will land on the ground and not falls off the screen.</p> <p>But do you think, should the ground move?</p> <p>Correct! Our ground will be static. Which means it will stay at one position only. Let's check again in the Bodies document if we find anything which will help us to create a ground.</p> <p><i>The teacher opens the document from Teacher Activity 2 and shows the Bodies.rectangle() function to the student.</i></p> <p>Here we have the Bodies.rectangle() function. Using this function we can create the rectangular ground.</p> <p>Just like we created ball_options to add properties to the ball, now we will create the ground_options variable to add properties to the ground body.</p> <p>As the ground will be stationary, the property we should add here is a boolean variable that can be either true or false.</p> <p>Which is i</p>	<p>ESR: The ball falls off the screen every time.</p> <p>ESR: No, the ground doesn't move.</p>
--	--

sStatic:true.

If this is set to **true** the body will remain stationary else it can move, by default, this is set to **false**.

As the ground is always at the bottom, so we'll give the rectangle body the **x** and **y** position towards the bottom of our canvas.

In the **setup()** function, we'll create the ground using the **Bodies.rectangle()** function and add the ground to the **World**.

In the **draw()** function using **rect()** we'll create the rectangle and pass the **x**, **y** positions along with the **width** and the **height** of the ground.

The teacher codes to create the ground.

The teacher runs the code to show the output.

The student observes and learns.

In the **setup()** function.

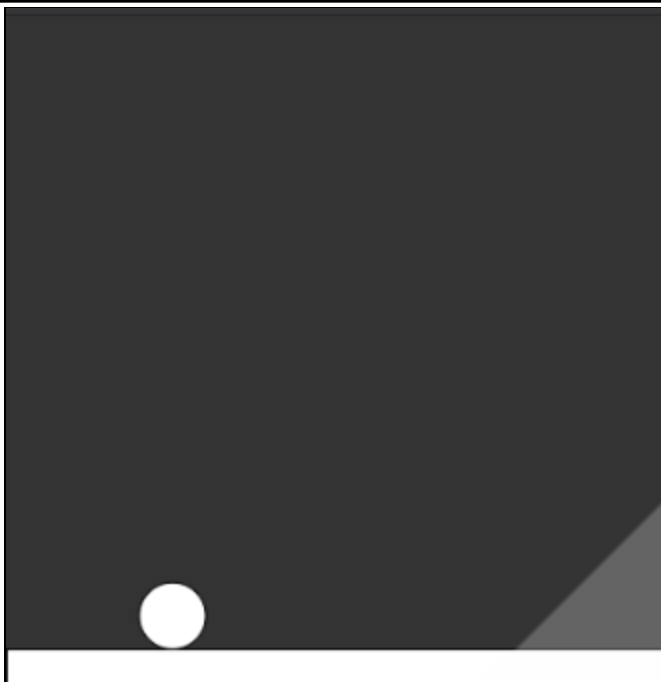
```
ground = Bodies.rectangle(200,390,400,20,ground_options);
World.add(world,ground);
```

In the **draw()** function.

```
function draw()
{
  background(51);
  Engine.update(engine);

  ellipse(ball.position.x,ball.position.y,20);

  rect(ground.position.x,ground.position.y,400,20);
}
```



Now we can see that the ball is bouncing on the ground.

In the physics engine, we don't have to write code to specify the collision between bodies, all the bodies can collide with each other and have an effect on the other body, like when a ball collides with the ground it moves in an upwards direction.

Would you like to create some of your own objects and try applying different physics properties?



Let's get you started then.

ESR:
Yes!

Teacher Stops Screen Share

STUDENT-LED ACTIVITY - 20 mins

- **Ask the student to press ESC key to come back to panel**
- **Guide Student to start Screen Share**
- **Teacher gets into Fullscreen**

<u>CHALLENGE</u>	
<ul style="list-style-type: none"> • Create a Rock body. • Create a stationary wall. 	
<div>  </div> <p>Teacher starts slideshow from slides 20 to 21 Refer to speaker notes and follow the instructions on each slide.</p>	
<i>Run the presentation slide 20 to slide 21 to set the student activity context.</i>	
<div>  </div> <p>Teacher ends slideshow</p>	
Teacher Action	Student Action
<p>Step 3: Student-led Activity</p> <p>In the previous activity, we have seen how we can create physics bodies, we can change their properties such as we can make them more bouncy using restitution, and make them static by setting isstatic:true flag.</p> <p>In this activity, we are going to create a rock that will be similar to creating a ball body. Then we will create a stationary wall at the center of the canvas.</p>	<p><i>Student downloads the Student Activity 1 link and opens in the VS Code.</i></p>
<p>First, you are going to create a rock body.</p> <p>This rock is going to be a circular body, just like a ball.</p> <p>First, we will create the rock_options for this, in which we will define the restitution property.</p>	

In the **setup()** function, define a variable as **var rock_options**. And we will set the restitution as **0.85**.

Can you guess what will be the effect of this value?

Great! When we reduce the value of restitution, the rock will be less bouncy.

ESR:

The rock won't bounce much.

```
var rock_options = {  
  restitution:0.85  
};
```

Now we can create the physics body for the rock using the **Bodies.circle()** function. In the function, we need to pass the x and y position of the rock, radius, and the **rock_options**.

The x-position will be **300**, the y-position will be **20**, and the radius as **10**.

Once we create the rock body, you need to add this to our world.

This you can do using the **World.add()** function.

```
rock = Bodies.circle(300,20,10,rock_options);  
World.add(world,rock);
```

You have created the physics body and added that to our world. But it won't be visible to us right now.

In the **draw()** function, we are going to create this **ellipse()** function, which will take **rock.position.x** and **rock.position.y** and the **radius** of the rock as arguments.

Now we can see one ball and one rock bouncing differently on the canvas.

Run the code to observe the output.

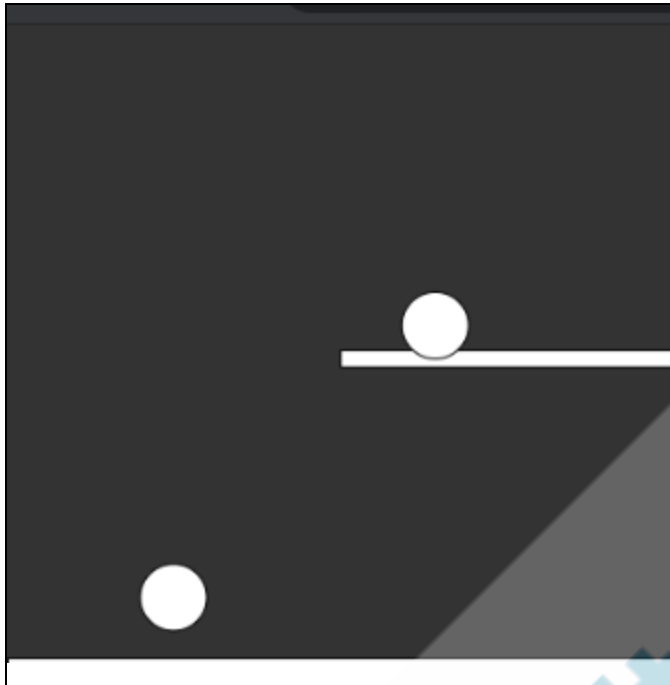
```
//rock  
ellipse(rock.position.x,rock.position.y,20);
```

Output:



<p>To make things interesting, add one more wall at the center of the canvas, so that the second ball can bounce on that.</p> <p>In the setup() function, create the physics body for the wall.</p> <p>Using the Bodies.rectangle() function. You are not going to create a different variable that will store the physics properties, rather you are going to use ground_options itself created previously as we want to keep the center wall stationary.</p> <p>We will set the x-position of the wall as 300 because we want to create this wall toward the right side of the canvas and the y-position can be set as 200. This will create the body at the center of the canvas, the width as 200 and height as 20.</p> <p>After you have created this body, now you need to add this body to the world using the world.add() function.</p>	
<pre> wall = Bodies.rectangle(300,200,200,20,ground_options); World.add(world,wall); </pre>	
<p>Finally, in the draw() function, you are going to show the wall using the rect() function.</p> <p>In the rect() function we will pass the x, y positions of the wall body and set the width as 200, and height as 10.</p>	<p><i>Run the code to see the output.</i></p>
<pre> //wall rect(wall.position.x,wall.position.y,200,10); </pre>	

Output:



Now we can see that we have a ball and a rock body on the canvas, one ball is bouncing on the ground and the other one bounces on the wall at the center of the canvas.

Teacher Guides Student to Stop Screen Share

WRAP-UP SESSION - 5 Mins



Teacher starts slideshow from slides 22 to 32

Refer to speaker notes and follow the instructions on each slide.

Activity details

Solution/Guidelines

Run the presentation from slides 22 to 32.




Following are the wrap-up session deliverables:




- **Explain the facts and trivias**
- **Next class challenge**
- **Project for the day**
- **Additional Activity**

Guide the student to develop the project and share it with us.

Quiz time - Click on in-class quiz

Question		Answer
Matter.Engine, Matter.World, Matter.Bodies and Matter.Body are _____ under the Matter.js library. A. Libraries B. Modules C. Objects D. Classes		B
If we set the restitution of a body as 2.5, what will be its effect on the bounciness of the body? A. No bounce B. bounce with slow speed C. Bounciness will increase D. Bounciness will decrease		C
Which of the following commands is the correct way of adding a body to the World? A. world_name.add(World,object_name) B. World.add(object_name,world_name) C. World.add(world_name,object_name) D. world.add(object_name)		C
End the quiz panel		
<u>FEEDBACK</u> <ul style="list-style-type: none"> ● 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. 		
	Good! We also learned some other features in p5.js libraries such as push() and pop(), rotate() and translate().	Yes!
	You get Hats Off for your excellent work!	<i>Make sure you have given at least 2 Hats Off during the class for:</i>

		<div>Creatively Solved Activities  +10</div> <div>Great Question  +10</div> <div>Strong Concentration  +10</div>
<p>* This Project will take only 30 mins to complete. Motivate students to try and finish it immediately after the class.</p> <p>Project Overview</p> <p>FALLING BALLS</p> <p>Goal of the Project:</p> <p>In Class 20, you learned how to create various kinds of shapes using a physics engine. Also learned to add the physics properties to these objects and observe their behavior. In this project, you will practice and apply the same concepts to create a ball falling down using a physics engine game.</p> <p>Story:</p> <p>Kimely's younger brother loves playing simple games involving physics. He is learning to write code and surprise his younger brother with a game - Drop the balls. Since he has just begun his journey with a physics engine, he would need your help to complete this game.</p> <p>Can you help him in creating it?</p> <p>I am very excited to see your project solution and I know you will do really well.</p> <p>Bye Bye!</p>		<p>Note: You can assign the project to the student in class itself by clicking on the Assign Project button which is available under the projects tab.</p> <p><i>Students engage with the teacher over the project.</i></p>

<p>Teacher ends slideshow </p>	
<p>Teacher Clicks </p>	
<p>ADDITIONAL ACTIVITY</p>	
<p>Teacher starts slideshow  from slides 33 to 37 Refer to speaker notes and follow the instructions on each slide.</p>	
<p>Additional Activities <i>Encourage the student to write reflection notes in their reflection journal using Markdown.</i></p> <p>Use these as guiding questions:</p> <ul style="list-style-type: none"> • What happened today? <ul style="list-style-type: none"> ◦ 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? 	<p><i>The student uses the Markdown editor to write their reflections in a reflection journal.</i></p>
<p>Additional Activity 1 <i><The teacher helps the student to clone the code from the Additional Activity 1></i></p> <p>Let's create a fan!</p> <p>To create a fan, what do we need? In today's class, we will create just one rotating blade, a small part of a fan.</p> <p>So first let's declare a variable wedge</p> <p>Using the Bodies.rectangle() function we'll create the wedge.</p>	<p><i>The student clones the code from Additional Activity 1.</i></p> <p>ESR: To create a fan we'll need a rectangle.</p>

<p>This wedge will also be static. So as for properties, we'll keep the isStatic flag as true</p>	<p><i>The student also codes to create the static Bodies.rectangle() body</i></p>
<div data-bbox="256 428 846 590"> <pre>var ops={ isStatic:true };</pre> </div> <p>Create a wedge passing the ops to it</p> <div data-bbox="256 638 1268 766"> <pre>wedge = Bodies.rectangle(100,200,100,20,ops); World.add(world,wedge);</pre> </div>	
<p>Let's now start creating a fan.</p> <p>We want this wedge to be rotating. So let's see if we have any function in the matter.min.js library which will help us to rotate the wedge.</p> <p><i>The teacher opens the doc from Teacher Activity 3 and shows the Matter.Body.rotate() function.</i> <i>Note: You can search with "ctrl+f" for "rotate" to locate the code.</i></p> <p>Here we have this function called the Matter.Body.rotate()</p> <p>This function takes two parameters:</p> <ol style="list-style-type: none"> 1) The body we want to rotate. 2) The angle by which we want it to rotate. <p>To give it an angle, let's create an angle variable and set it at a random value of 60.</p> <p><i>The teacher can show different outputs with different angles.</i></p> <p>We can set any value we want, but we cannot keep this parameter empty.</p>	

Now in the **rotate()** function let's pass the **wedge** and the **angle**.

You'll also create the rectangle using the **rect()** function at **pos x - 0** and **pos.y - 0**.

The teacher helps the student with the code.

The student runs the code to check the output.

In the **draw()** function.

```
function draw()
{
  background(51);
  Engine.update(engine);

  Matter.Body.rotate(wedge, angle);
  rect(0,0,100,20);
}
```

Output:



<p>Now, what do we see?</p> <p>Our wedge is now rotating, but we only see it rotating in one position.</p> <p>Every time the draw() function is being called, the new positions that we have passed to the wedge revert to the old positions.</p> <p>To solve this issue we use the push(), pop() and translate() functions.</p> <p>push() function captures the new setting. pop() function reverts to the old setting. translate() function will translate(move) the coordinates of the canvas. Normally 0,0 is set in the top left corner of the canvas. But with the translate() function, we will move it to the position of the wedge.</p>	<p>ESR:</p> <p>We can see that the rectangle is static, but the ball is getting hit and going out of the screen.</p>
<p>We are doing this because the rotate function rotates the shape around only the origin, if we don't translate the origin to the position of the wedge then the wedge will rotate around the top left corner, it will look like it is orbiting that point.</p> <p>But we want to rotate the wedge on its center point, which is why we need to translate the whole coordinate system to the center of the wedge.</p> <p>But we want this setting only for the wedge, not for other bodies that is why we keep these functions in between a push() and pop() function.</p> <p>So first we'll use the push() to capture the new settings.</p> <p>Then we'll use the translate() function to pass it to the x and y position of the wedge.</p> <p>Then using the rotate() function we'll give it the angle of rotation.</p>	

Finally, use the **pop()** function to revert to the old settings.

We would also want to increase the angle of rotation by **0.1** degrees so that when the draw function runs we can see the wedge rotating.

So we'll also write **angle +=0.1**

The teacher helps the student with the code.

*The student codes to use the **push()**, **pop()** and **translate()**.*

In the **draw()** function

```
function draw()
{
  background(51);
  Engine.update(engine);

  Matter.Body.rotate(wedge,angle)
  push();
  translate(wedge.position.x,wedge.position.y);
  rotate(angle);
  rect(0,0,100,20);
  pop();

  angle +=0.1;
```

Now, what do we see?

ESR:

We can see the rotating wedge, the ball hits the

What can we do to stop the ball on the screen?

Yes, so let's start creating another static object, we can call it a wall.

Using the **Bodies.rectangle()** function we'll create a wall. We will pass the **ground_options** to the wall because we want it to stay stationary like the ground.

wedge and goes out of the screen.

ESR:

We can create another wall to which the ball will hit and stay in the frame.

The student codes to create the wall.

In the **setup()** function.

```
wall = Bodies.rectangle(300, 150, 70, 10, ground_options);
World.add(world, wall);
```

In the **draw()** function.

```
rect(wall.position.x, wall.position.y, 70, 20);
```

Complete code for reference given below:

```
function setup() {
  createCanvas(400, 400);

  engine = Engine.create();
  world = engine.world;

  var ball_options = {
    restitution: 0.95,
    frictionAir: 0.01
  }

  var ground_options = {
    isStatic: true
  };
```

```
var ops={
  isStatic:true
};

ground = Bodies.rectangle(200,390,400,20,ground_options);
World.add(world,ground);

ball = Bodies.circle(100,10,20,ball_options);
World.add(world,ball);

wall = Bodies.rectangle(300, 150, 70, 10, ground_options);
World.add(world,wall);

wedge = Bodies.rectangle(100,200,100,20,ops);
World.add(world,wedge);
rectMode(CENTER);
ellipseMode(RADIUS);
}
function draw()
{
  background(51);
  Engine.update(engine);


  Matter.Body.rotate(wedge,angle)
  push();
  translate(wedge.position.x,wedge.position.y);
  rotate(angle);
  rect(0,0,100,20);
  pop();

  angle +=0.1;

  ellipse(ball.position.x,ball.position.y,20);
  rect(ground.position.x,ground.position.y,400,20);
  rect(wall.position.x,wall.position.y,70,20);
```


}

Output:



So now our ball gets hit by the moving wedge and stays on the ground.

You can play around with more properties of objects like restitution, density, friction, etc.

Teacher ends slideshow 

Activity	Activity Name	Links
Teacher Activity 1	Teacher boilerplate code	https://github.com/pro-whitehatjr/pro-c20-ta-boilerplate
Teacher Activity 2	Matter.Bodies document	https://brm.io/matter-js/docs/classes/Bodies.html
Teacher Activity 3	Matter.Body.rotate()	https://brm.io/matter-js/docs/classes/Body.html

Teacher Activity 4	Teacher reference code	https://github.com/pro-whitehatjr/C20-basics-of-physicsEngine
Student Activity 1	Boilerplate	https://github.com/pro-whitehatjr/C20-Student-boilerplate-code
Student Activity 2	Matter.bodies document	https://brm.io/matter-js/docs/classes/Body.html#:~:text=Rotates%20a%20body%20by%20a.without%20imparting%20any%20angular%20velocity.
Teacher Reference	Complete reference code	https://github.com/pro-whitehatjr/pro-c20-reference-code
Additional Activity 1	Boilerplate	https://github.com/pro-whitehatjr/C20-Student-boilerplate-code
Teacher Reference visual aid link	Visual aid link	https://curriculum.whitehatjr.com/Visual+Project+Asset/PRO_VD/BJFC-PRO-V3-C20-withcues.html
Teacher Reference In-class quiz	In-class quiz	https://s3-whjr-curriculum-uploads.whjr.online/e9242af0-1600-4fae-a45f-627cc06ae439.pdf
Project Solution	Falling Balls	https://github.com/pro-whitehatjr/PRO-C20-V3-Solution