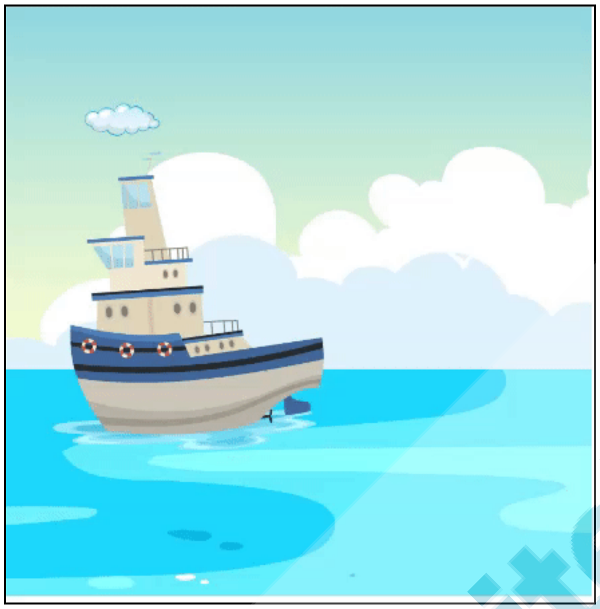
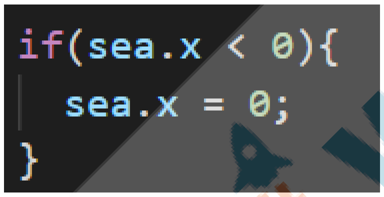
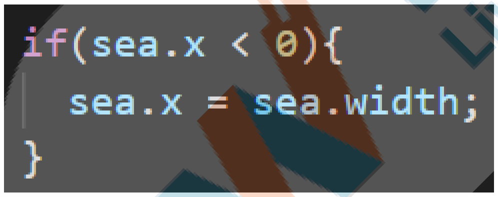
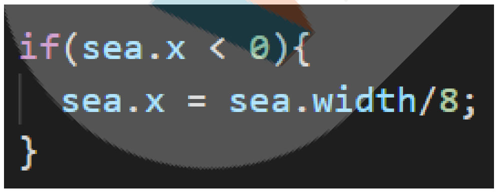
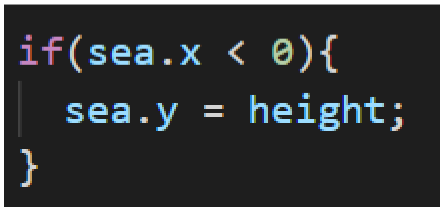
Topic	CODE DEBUGGING AND CODE INDENTATION	
Class Description	**The student learns to indent code to make it more readable. The student also uses the method of displaying a message on the console to debug the program.**	
Class	**PRO-C11**	
Class time	**50 mins**	
Goal	● Indent the code correctly to make it more readable. ● Identify an additional condition needed in the program to stop the Trex from jumping again while it is in the air. ● Create an invisible ground sprite to make the Trex run below the ground.	
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 Slides** **Teacher-Led Activity 1** **Student-Led Activity 1** **Teacher-Led Activity 2** **Student-Led Activity 2** **Wrap-Up Slides**	**10 mins** **10 mins** **5 mins** **10 mins** **10 mins** **5 mins**

● **WARM-UP SESSION - 10 mins**	
Teacher starts slideshow from slides 1 to 7 Refer to speaker notes and follow the instructions on each slide.	
Teacher Action	**Student Action**
Hey <student 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 4.*** **The following are the warm-up session deliverables:** ● Connecting students to the previous class.	**ESR**: Hi, thanks, yes I am excited about it! Click on the slide show tab and present the slides.
QnA Session	
Question	**Answer**

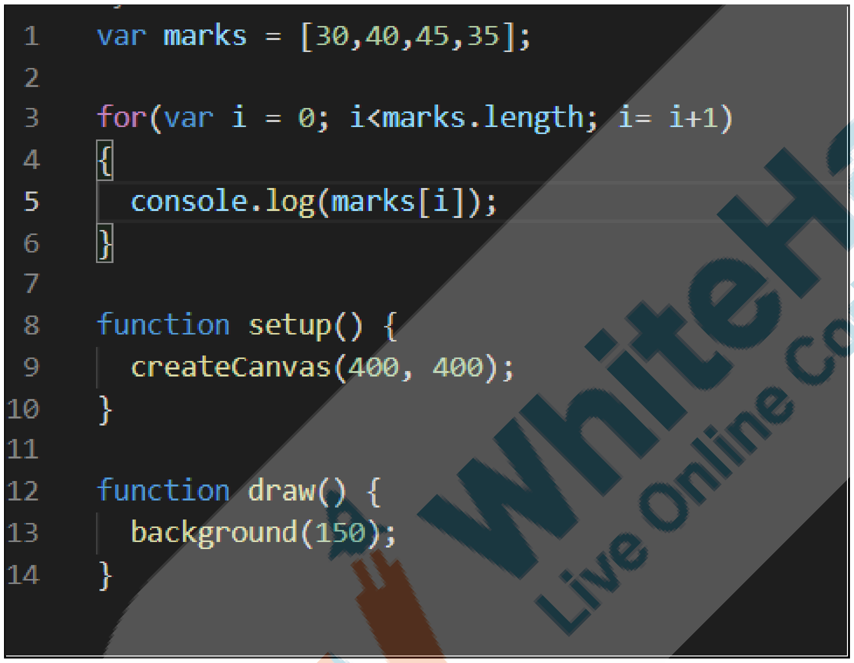
Which of the following statements is used to add an image to the sprite called sea?
 	
A. sea.addImage(seaImg);
B. sea.setAnimation(seaImg);
	A

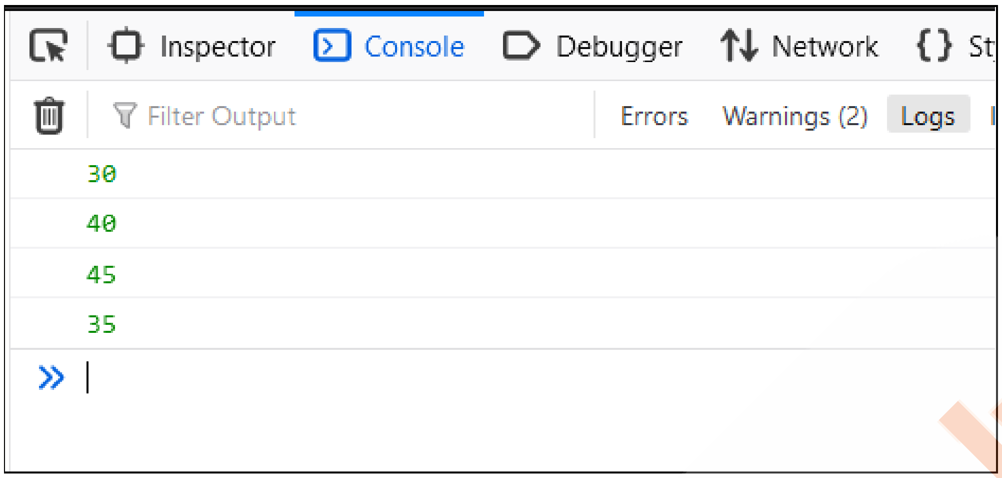
C. sea.setImage(seaImg);
D. sea.createImage(seaImg);	

Select the line of code that will make the sea background repeat forever.  A.  `if(sea.x < 0){ sea.x = 0; }` B.  `if(sea.x < 0){ sea.x = sea.width; }` C.  `if(sea.x < 0){ sea.x = sea.width/8; }` D.  `if(sea.x < 0){ sea.y = height; }`	**C**

Continue the warm-up session	
Activity details	**Solution/Guidelines**
Run the presentation from slide 5 to slide 7 to set the problem statement. ● Revise the for loop and use it for the program.	Narrate the story by using hand gestures and voice modulation methods to bring in more interest in students.
Teacher ends slideshow	
TEACHER-LED ACTIVITY 1 - 10 mins	
Teacher Initiates Screen Share	
Teacher Action	**Student Action**
Step 1: **Teacher-led** **Activity** Great! In the last class, we have done a coding activity with arrays, where we stored the marks in an array then calculated the average of our marks, but if you noticed, the method we used was very tedious and long. We programmers don't like long methods, we always want to do our work as fast as possible. So, in this activity, we are going to use a **for** loop in our code, and you will see how the **for** loop makes our life easy. If we want to display all the elements of an array, we don't have to write it manually for each element. We can directly use the **for** loop for this. In the **for** loop we first define a variable and give it a **starting condition**, in our case it is **0** because the index of the array starts from **0**,	*The teacher downloads Teacher Activity 1 and runs it in the VS Code Editor.*

then we provide the **stopping condition**, here we are setting it as the length of the array **(Number of elements in the array).** At the end we set the **increment** for the variable i.e, here we are incrementing **i** by **1** every time our loop runs. Then in the curly brackets of the **for** loop we will write: **console.log(marks[i])** so that when the loop runs, it will give us the elements from the **marks** array.	


`var marks = [30,40,45,35];`
 	
`for(var i = 0; i<marks.length; i= i+1)`
`{`
`  console.log(marks[i]);`
`}`
 	
`function setup() {`
`  createCanvas(400, 400);`
`}`
 	
`function draw() {`
`  background(150);`
`}`
 	
Output:	



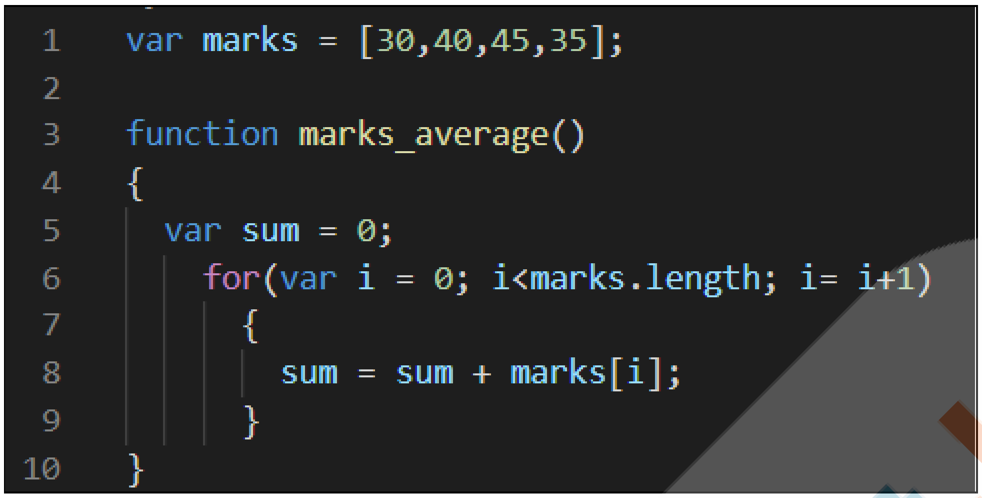
```
30
40
45
35
```

Now we learned to access elements from the array using a **for** loop. Let's write the function to calculate the average marks once again but more efficiently.

In the function, first, we need a **variable** to store the sum of the marks. This variable should start from **0,** every time our function runs, otherwise, we will have errors if we calculate the average of marks.

For the **for** loop we need a **starting condition** that we set as **0**, our **loop counter** will start from **0**, but we also need to define the **stopping condition** where our loop will stop, our stopping condition is the number of elements in the array. **(length of the array)**

Finally, since we want to go from one element to another, our counter should advance by **1** per loop.



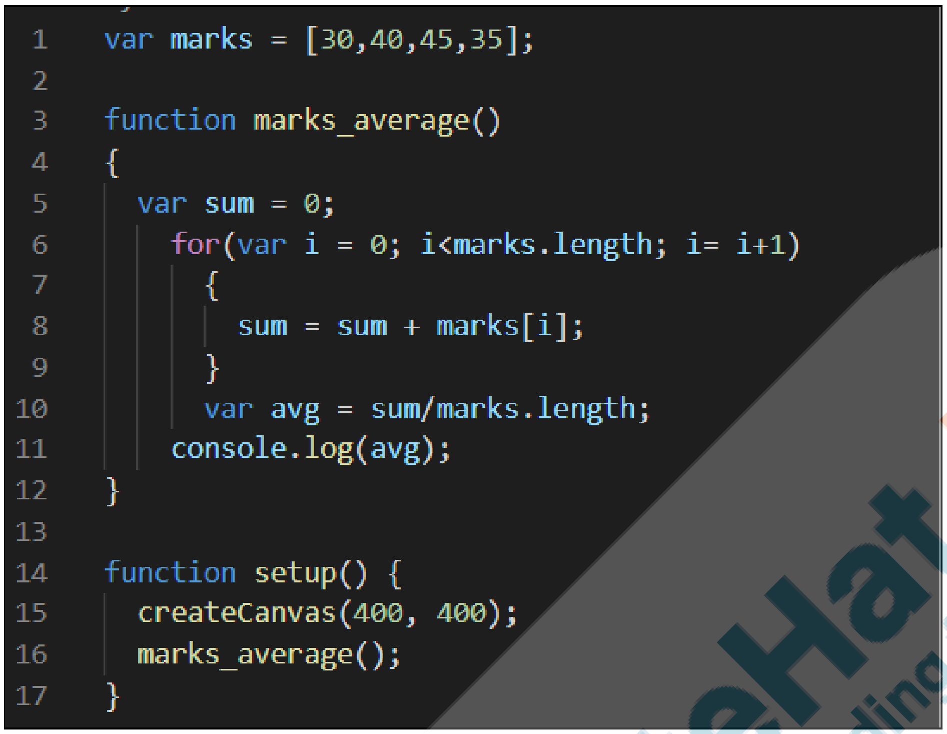
```
var marks = [30,40,45,35];

function marks_average()
{
  var sum = 0;
    for(var i = 0; i<marks.length; i= i+1)
      {
        sum = sum + marks[i];
      }
}
```

In the parentheses of the **for** loop, we will add the elements of the array in the **sum** variable, and in every interaction it will update the **sum** variable also by adding these elements in the **sum** variable.

For example, if you have **0** apples in the basket then I have added **1** apple to your basket, so now you have **1** apple.

If I add **1** more apple to your basket you will have **2** apples now, so the new apple is added to the previously present count of apples; the same thing we are doing here.

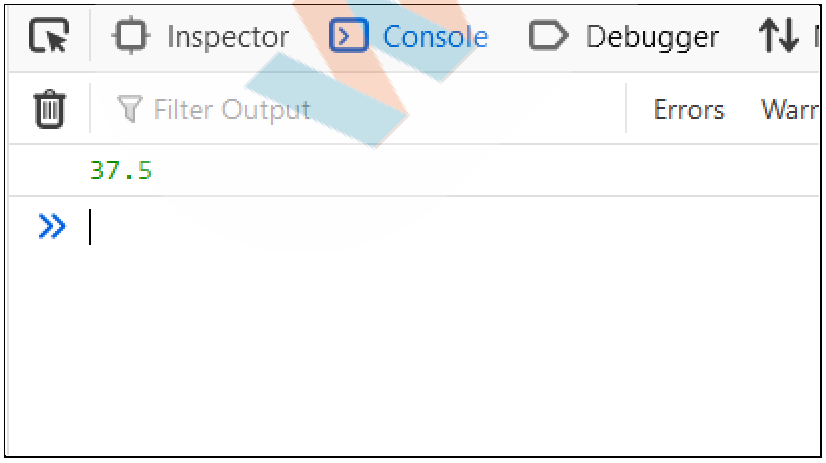
Our sum was at **0** in the beginning, but as we run the **for** loop, first we add the 1st element of the array in the **sum** then the 2nd element in that **sum**, and so on. In the end, we will have the sum of all the elements of the marks array.



```
var marks = [30,40,45,35];

function marks_average()
{
  var sum = 0;
    for(var i = 0; i<marks.length; i= i+1)
      {
        sum = sum + marks[i];
      }
      var avg = sum/marks.length;
    console.log(avg);
}

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

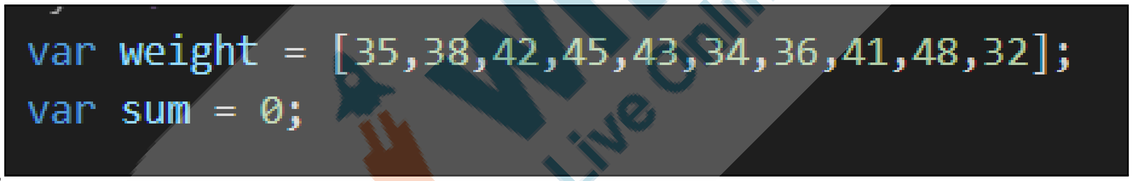
Once we get the sum, we will divide the sum by **marks.length**, and we will have our average of marks. And finally, we call the **marks_average()** function in the **setup()** function

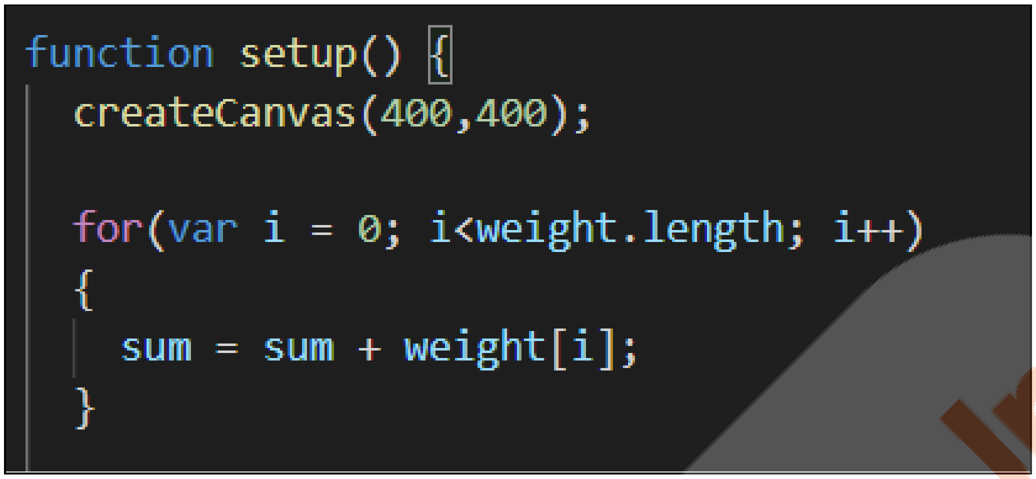
Output:



```
37.5
```

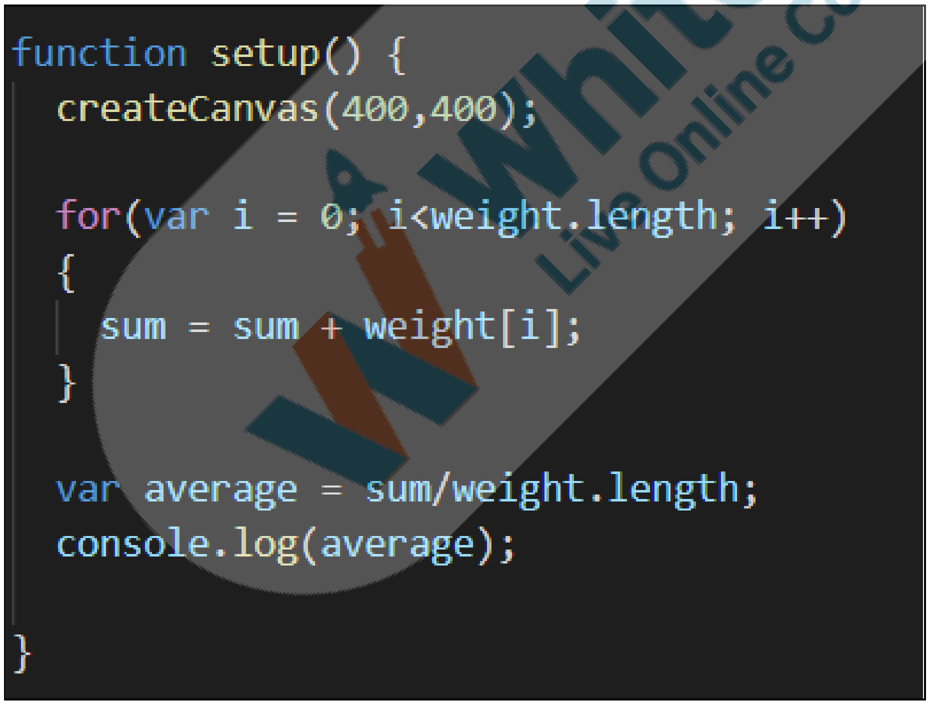
Teacher starts slideshow : Slides 8 to 9	
Now you are armed with the knowledge of for loops and arrays. Do you want to solve a challenge to reinforce your knowledge?	**ESR:** Yes!
Teacher ends slideshow	
Teacher Stops Screen Share	
Now it's your turn. Please share your screen with me.	
Student-Led Activity 1 - 5 mins	
• **Ask the student to press the ESC key to come back to the panel.** • **Guide the student to start Screen Share.** • **The teacher gets into Fullscreen.**	
ACTIVITY • **Create an array.** • **Calculate the average using for loop**	
Teacher Action	**Student Action**
Step-2 Student-led Activity In the last class, we learned to access the elements from the array manually. In this class, we will use the **for** loop to access all the elements of an array, and create the program to find the average weight from the array.	*The student downloads the code from the student activity 1 link.* *The student opens the code in VS Code editor and starts the live server.*

When we access the elements of the array manually we have to write a lot of code. if we have an array with 100 or 1000s of elements it won't be a very good practice to do this manually. For example you have the weight of your entire society or your entire school. In this case we can use for loop which will make our work very easy. First task is to create an array having weight values, we are going to create a little large array this time having at least 10 elements, you can add more if you want. **var weight = [30,32];** **Note:** Only 2 elements are shown in the array here, but students need to add 10 elements in the array. We are also going to initialize the sum variable as 0 at the starting of the code because we want the sum to be zero every time we run the program.	
 `var weight = [35,38,42,45,43,34,36,41,48,32];` `var sum = 0;`	
Now in the setup function we are going to go through each element of this array using the **for** loop.We will start our counter at 0 and go until we reach the last element of the array for that we are going to use weight.length and we are going to move in increments of 1. Inside the for loop we are going to add the weight array element in the sum and we are going to keep on adding each element in the sum until we reach the last element.	



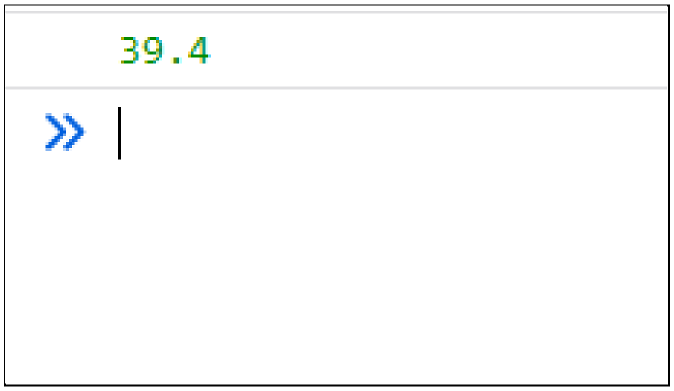
```
function setup() {
  createCanvas(400,400);

  for(var i = 0; i<weight.length; i++)
  {
    sum = sum + weight[i];
  }
```

Now all the element values have been added to the sum variable. to get the average weight we need to divide the sum with the length of the array or the number of elements inside the array.
Then we will display the result using **console.log()**.



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

  for(var i = 0; i<weight.length; i++)
  {
    sum = sum + weight[i];
  }

  var average = sum/weight.length;
  console.log(average);

}
```

Output



```
39.4
>> |
```

Once the student completes the activity, the teacher will start with the main activity of the class.	

Teacher Guides Student to Stop Screen Share

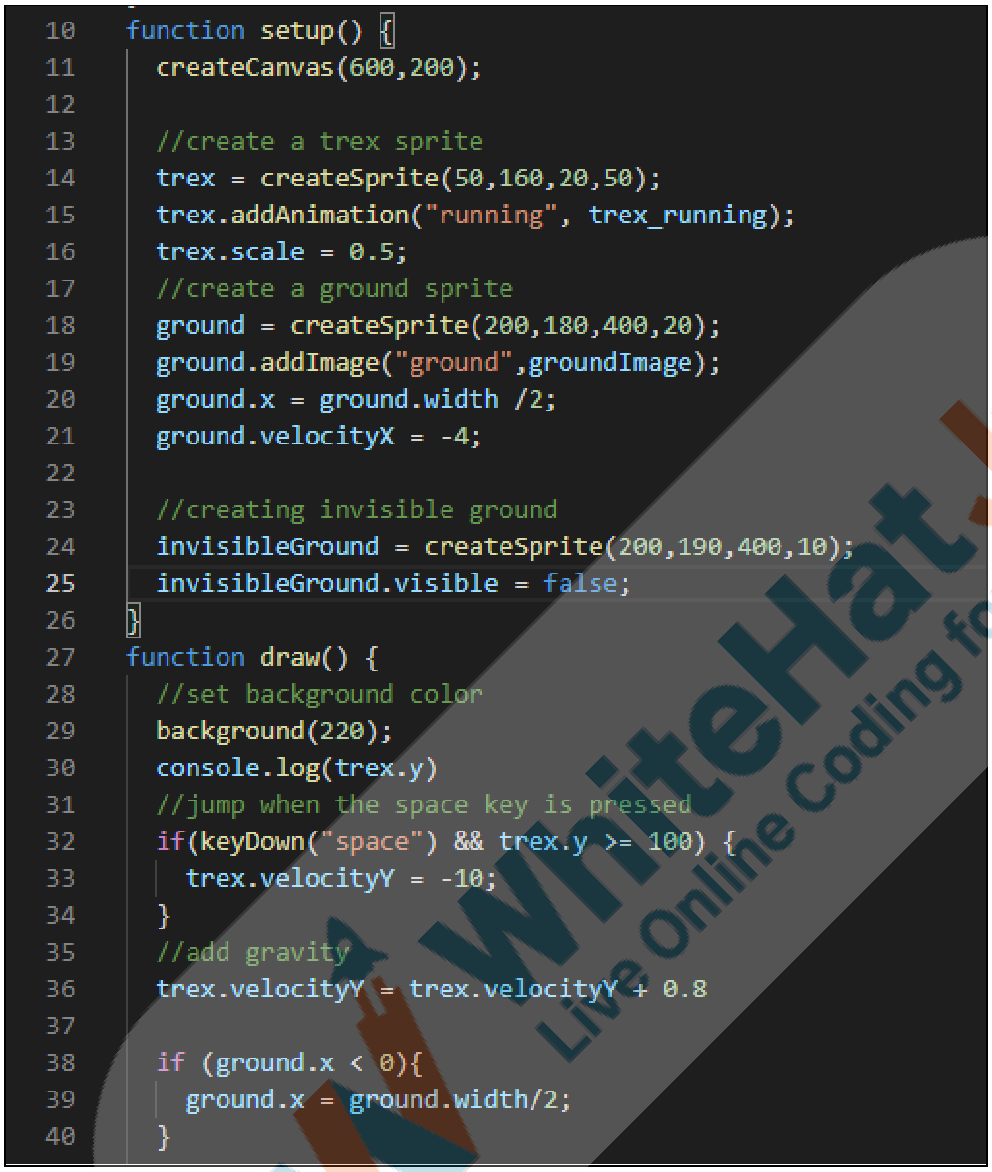
TEACHER-LED ACTIVITY 2 - 10mins

Teacher Initiates Screen Share

Teacher starts slideshow :Slide 10 to 15

Teacher Action	Student Action
Step 3: **Teacher-led Activity** *The teacher opens Teacher Activity 2.* This is the code from the last class. Do you see any problems with this code?	**ESR:** There is no spacing between the lines of the code. They are all together and difficult to read.
Yes, computers don't mind or read spaces. But it is important to give spaces in your code to make it easily	*The student listens and learns.*

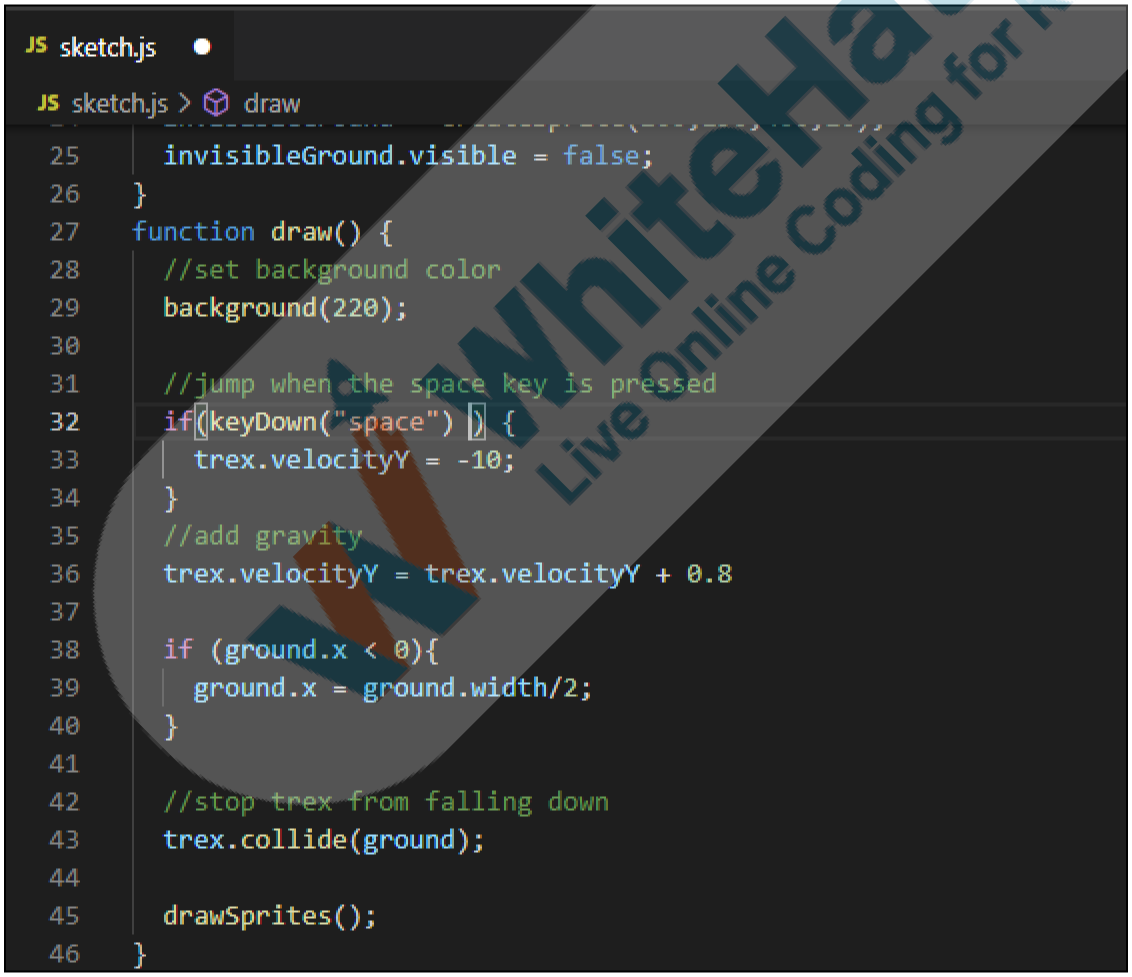
readable. Remember, you want other programmers to easily read your code. Giving proper spaces in your code makes your code easily readable. Let's try to add some space in your code.	
We try to leave a line after a meaningful block of code. Can you tell which code lines together make a meaningful block of code? The teacher leaves lines after each meaningful block of instruction.	*The student observes, comments, and learns.* **ESR:** Varied.



```
function setup() {
  createCanvas(600,200);

  //create a trex sprite
  trex = createSprite(50,160,20,50);
  trex.addAnimation("running", trex_running);
  trex.scale = 0.5;
  //create a ground sprite
  ground = createSprite(200,180,400,20);
  ground.addImage("ground",groundImage);
  ground.x = ground.width /2;
  ground.velocityX = -4;

  //creating invisible ground
  invisibleGround = createSprite(200,190,400,10);
  invisibleGround.visible = false;
}
function draw() {
  //set background color
  background(220);
  console.log(trex.y)
  //jump when the space key is pressed
  if(keyDown("space") && trex.y >= 100) {
    trex.velocityY = -10;
  }
  //add gravity
  trex.velocityY = trex.velocityY + 0.8

  if (ground.x < 0){
    ground.x = ground.width/2;
  }
```

There is no fixed rule for leaving these spaces. It is just like leaving spaces between paragraphs when you write a story. But do you see how easy it is to read and understand the code now?	**ESR:** Yes.

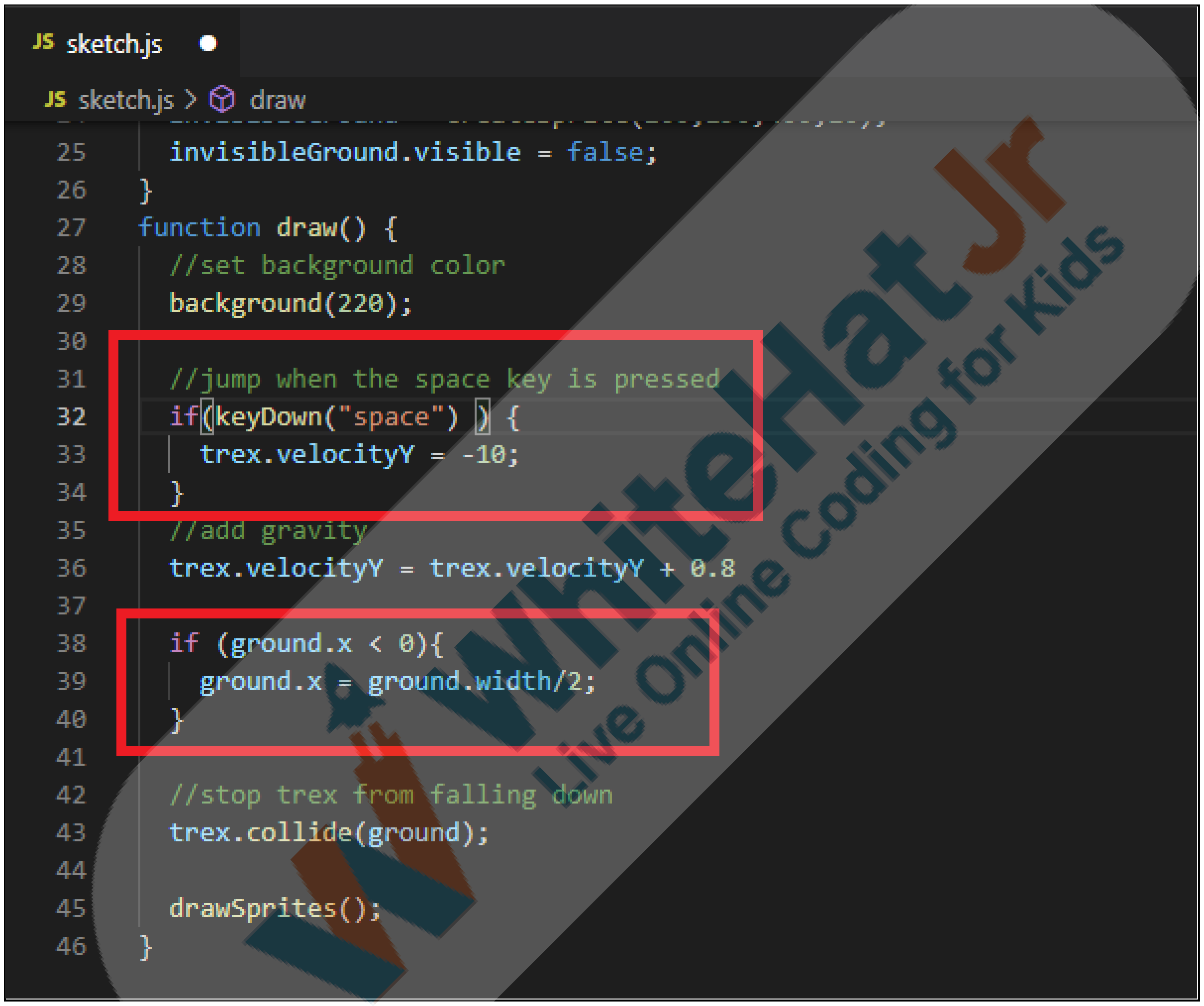
We also add some code indentation to lines to show that they are contained inside a block of code. Let me show you how. For example, lines **27** to **46** are contained inside the **draw()** function. We show it by indenting these lines by adding some space in front of these lines. This spacing should be consistent. *The teacher selects lines **27** to **46** and presses TAB.*	*The student observes and learns.*



sketch.js
sketch.js > draw

```
25      invisibleGround.visible = false;
26    }
27    function draw() {
28      //set background color
29      background(220);
30
31      //jump when the space key is pressed
32      if(keyDown("space") ) {
33        trex.velocityY = -10;
34      }
35      //add gravity
36      trex.velocityY = trex.velocityY + 0.8
37
38      if (ground.x < 0){
39        ground.x = ground.width/2;
40      }
41
42      //stop trex from falling down
43      trex.collide(ground);
44
45      drawSprites();
46    }
```

This is called **code indentation**. Similarly, the lines of code inside the **if** blocks or **for** blocks need to be indented as well.	*The student observes and learns.*



sketch.js
sketch.js > draw

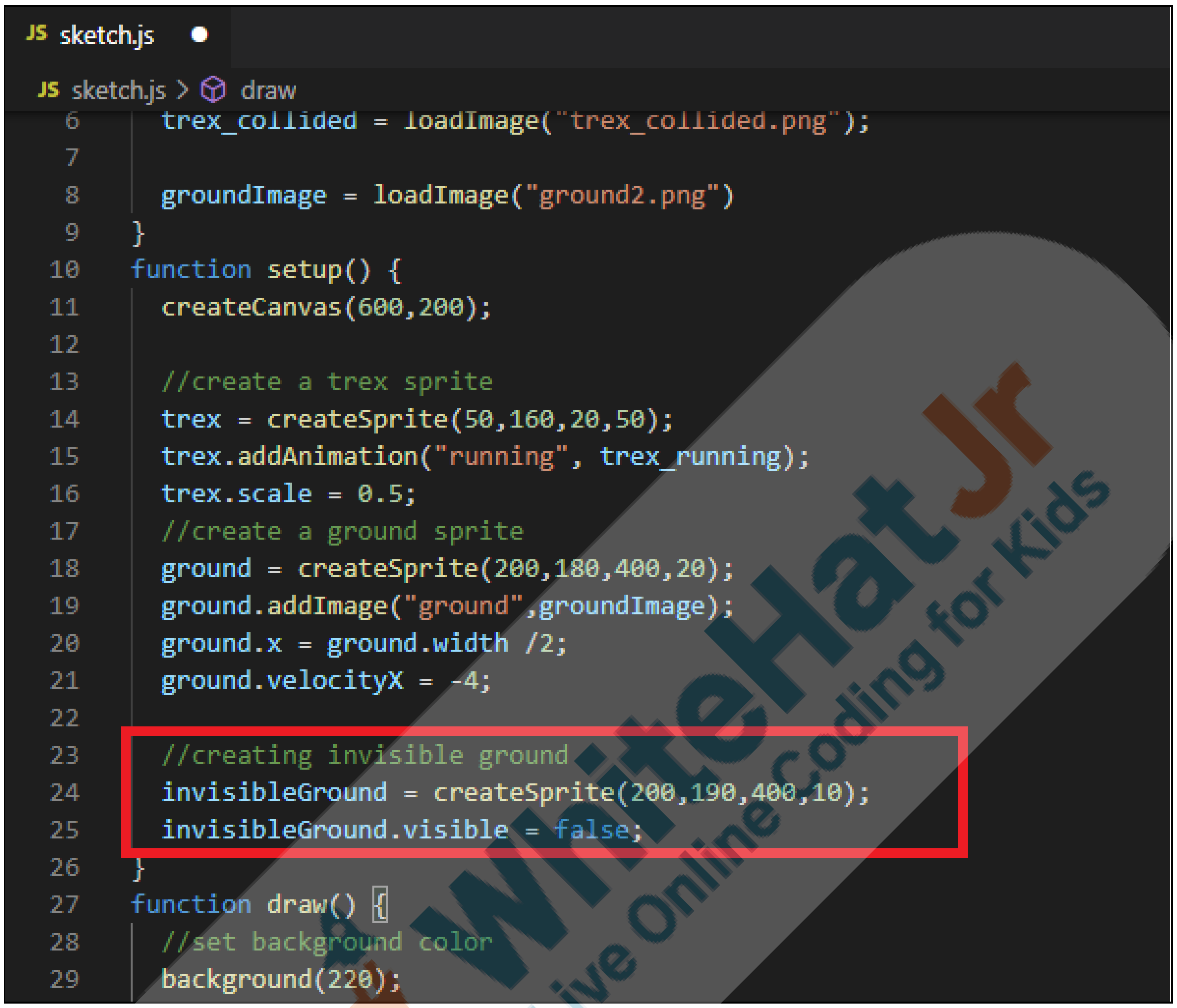
```
  invisibleGround.visible = false;
}
function draw() {
  //set background color
  background(220);

  //jump when the space key is pressed
  if(keyDown("space") ) {
    trex.velocityY = -10;
  }
  //add gravity
  trex.velocityY = trex.velocityY + 0.8

  if (ground.x < 0){
    ground.x = ground.width/2;
  }

  //stop trex from falling down
  trex.collide(ground);

  drawSprites();
}
```

Code indentation helps us understand the program structure easily. It also makes us less likely to make mistakes while typing out text - like missing out on closing curly brackets.	*The student listens and learns.*

WhiteHat Jr
Live Online Coding for Kids

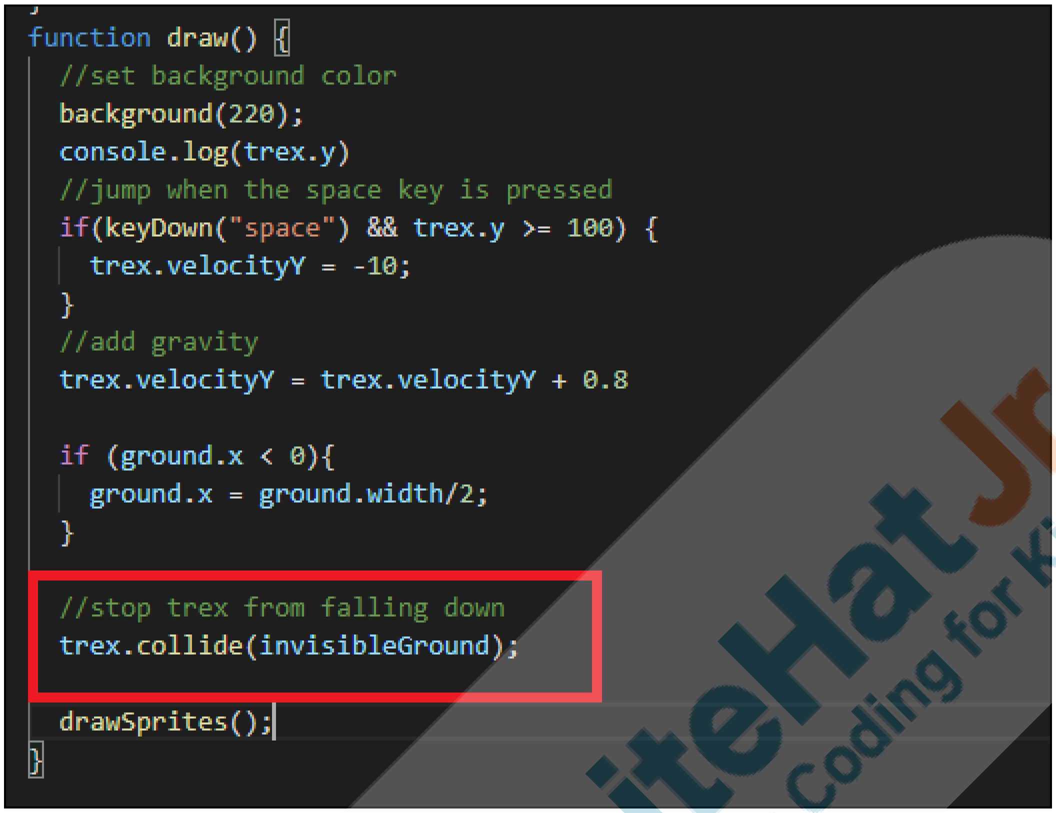
Teacher starts slideshow : Slide 16 to 32	
Ok. Now it's time to work on solving the two bugs we had in our program. Bugs are parts of the program that do not work as we want. We have **two** such bugs that we will solve today. - the dinosaur jumping in mid-air when the space key is pressed. What do we want instead?	**ESR:** We want the dinosaur to be able to jump only when it is touching the ground. It should be able to jump again only when it falls back on the ground.
Great! Our second bug is that our dinosaur is running a little over the ground. What do we want instead?	**ESR:** We want it to run a little below the ground.
Ok, let's fix these.	
Teacher ends slideshow	
Teacher Stops Screen Share	
STUDENT-LED ACTIVITY 2 - 10 mins	
• **Ask the student to press the ESC key to come back to the panel.** • **Guide the student to start Screen Share.** • **The teacher gets into Fullscreen.**	

ACTIVITY • **Identify and add an additional condition so that the Trex jumps only when it is in contact with the ground.** • **Create an invisible ground sprite which is below the ground and on which the Trex dinosaur is supported.**	
Teacher Action	**Student Action**
Step 4: **Student-Led Activity** Ok. Quickly fire up your activity and indent your code. *The teacher helps the student to properly indent their code.*	The student opens Student Activity 2
Let us first fix the second bug. The Trex right now is supported by the **ground** sprite. Collision with the **ground** sprite is not letting the Trex fall off the ground. As our Trex looks like it's running a little above the ground, we'll fix this by creating an invisible ground just below the original ground and make our Trex run on the invisible ground so that it looks like the Trex is running on the ground. Let us create an invisible ground sprite just below this ground. We want to do this so that rather than being supported by the ground and being above the ground, the Trex gets supported by an invisible ground just below the actual ground. Can you create another ground sprite just below the first ground and make it cover the entire width of the screen? For fun, let us call it **invisibleGround**. *Guide the student to create an invisible ground Sprite.*	*The student creates an invisible ground Sprite.* **ESR**: Yes!



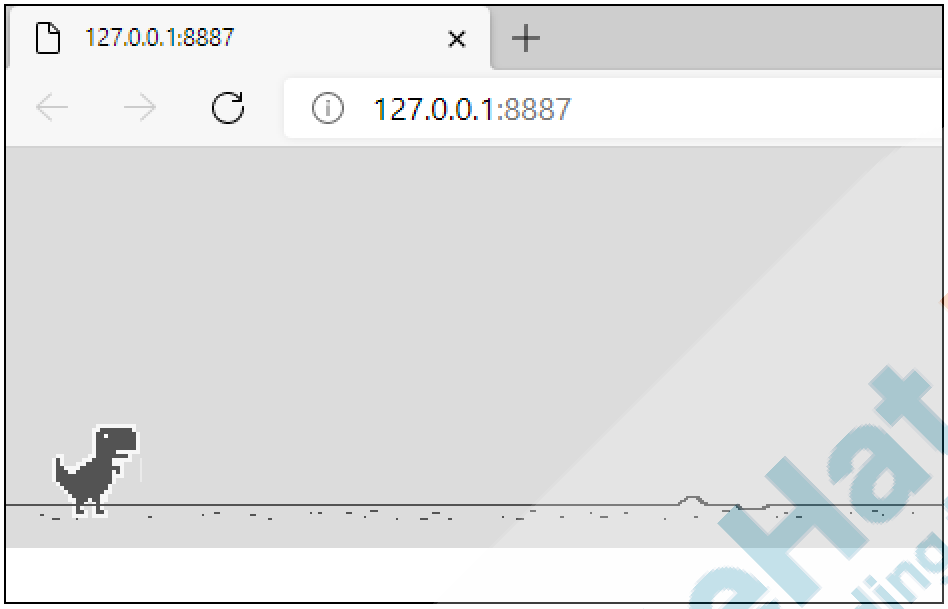
```
  trex_collided = loadImage("trex_collided.png");

  groundImage = loadImage("ground2.png")
}
function setup() {
  createCanvas(600,200);

  //create a trex sprite
  trex = createSprite(50,160,20,50);
  trex.addAnimation("running", trex_running);
  trex.scale = 0.5;
  //create a ground sprite
  ground = createSprite(200,180,400,20);
  ground.addImage("ground",groundImage);
  ground.x = ground.width /2;
  ground.velocityX = -4;

  //creating invisible ground
  invisibleGround = createSprite(200,190,400,10);
  invisibleGround.visible = false;
}
function draw() {
  //set background color
  background(220);
```

Now, instead of supporting the Trex on the ground, let us collide it with the invisible ground.	*The student modifies* ***trex.collide(ground)*** *to* ***trex.collide(invisibleGround)****.*

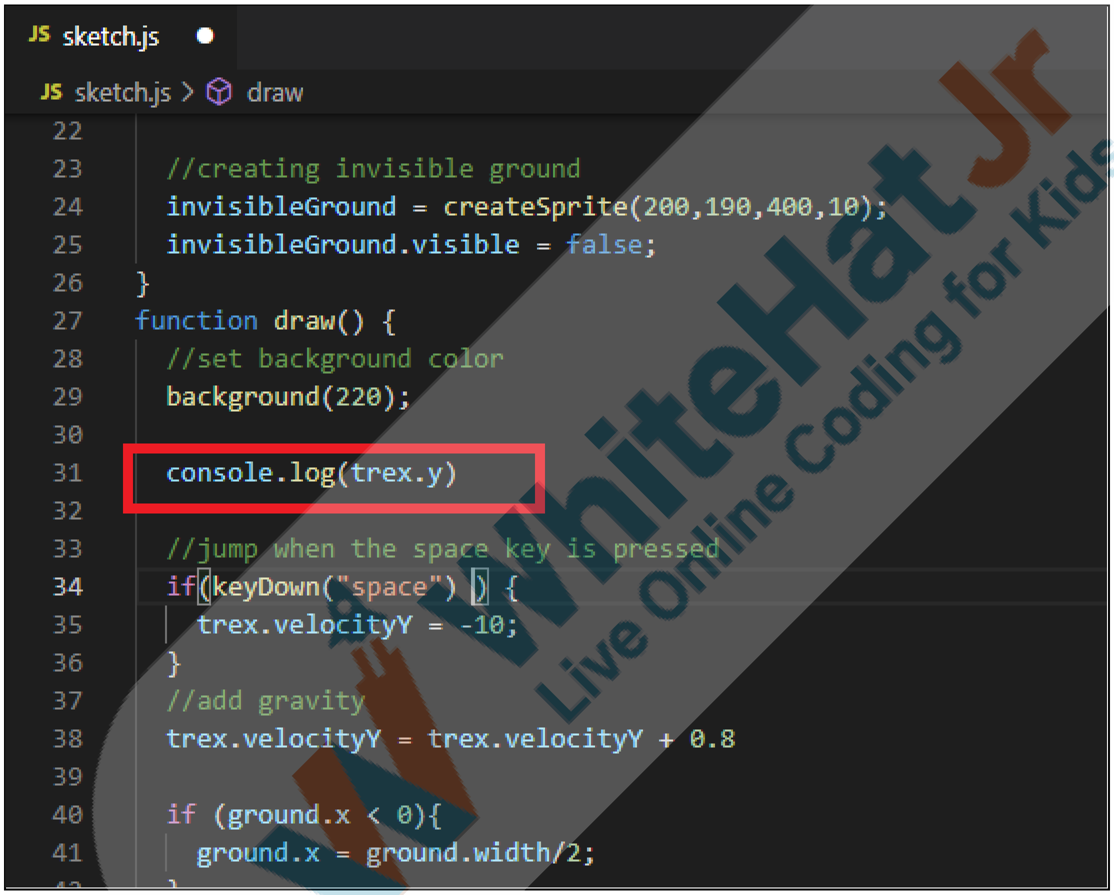


```
function draw() {
  //set background color
  background(220);
  console.log(trex.y)
  //jump when the space key is pressed
  if(keyDown("space") && trex.y >= 100) {
    trex.velocityY = -10;
  }
  //add gravity
  trex.velocityY = trex.velocityY + 0.8

  if (ground.x < 0){
    ground.x = ground.width/2;
  }

  //stop trex from falling down
  trex.collide(invisibleGround);

  drawSprites();
}
```

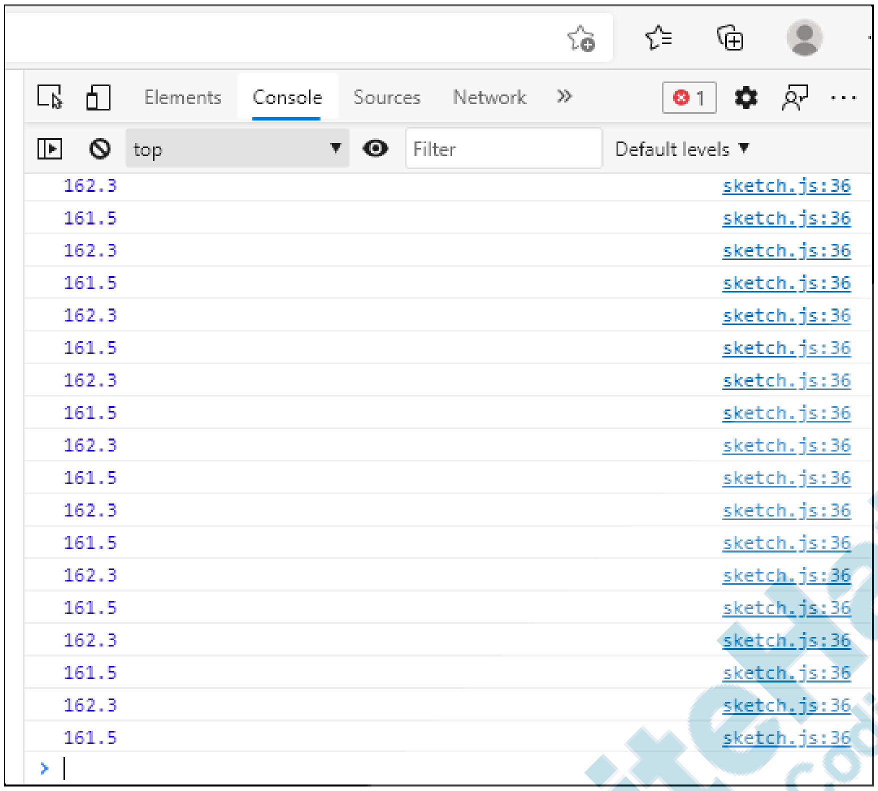
Now for the magic! Let us make this **ground** sprite invisible. There is an instruction in the **p5.play** library named **sprite.visible**. You need to make it **false** to make the ground invisible. By saying '**sprite.visible = false**', we are asking the computer to **NOT** make this sprite visible.	*The student writes* ***invisibleGround.visible = false*** *to make the ground invisible.* *The student runs the code to see the output.*
Now, add the following line of code anywhere outside the **draw()** function and after creating the **invisibleGround** Sprite: **invisibleGround.visible = false;** **Note- Click on the canvas where you see the output to use the keys to make the Trex jump.**	

We have the Trex running on the ground now!	
	
Let us try to fix the other bug with which solution?	**ESR:** Make the dinosaur jump only when it is on the ground.
When does the dinosaur jump now?	**ESR:** When the space key is pressed.
We want to make it jump when the space key is pressed and when it is on the ground as well, right?	**ESR:** Yes.
What do we need to do?	**ESR:** Add an additional **if** condition.

Right! Let us try to display the current y position of the Trex when it is running on the ground. *Guide the student to log* ***trex.y*** *on the* ***console*** *window.*	*The student writes to log* ***trex.y*** *on the* ***console*** *window.*

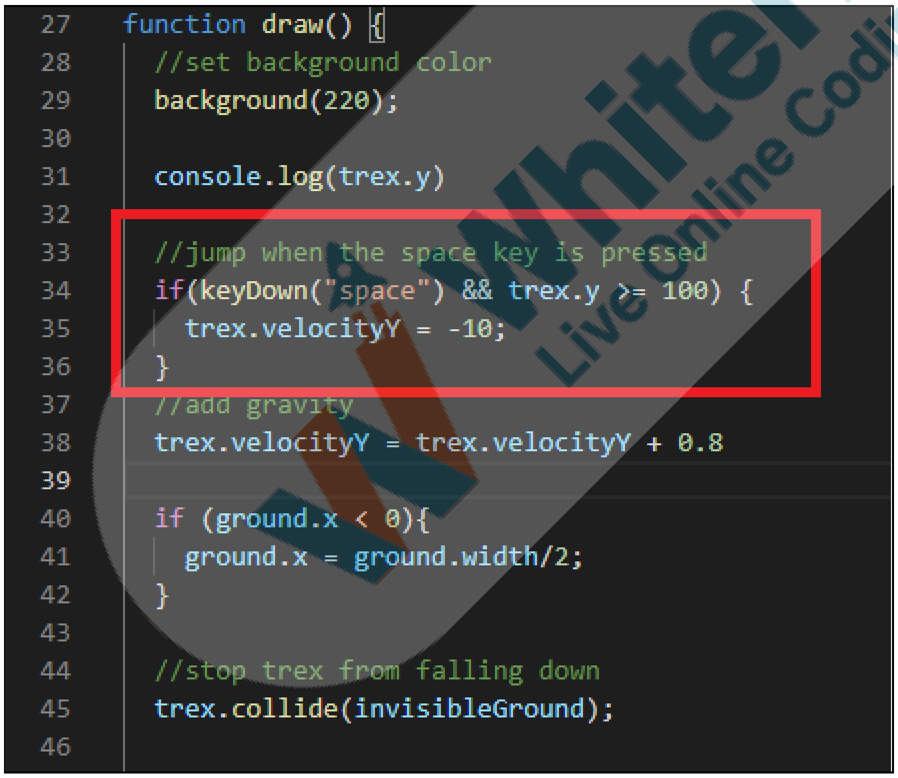


```
    //creating invisible ground
    invisibleGround = createSprite(200,190,400,10);
    invisibleGround.visible = false;
}
function draw() {
    //set background color
    background(220);

    console.log(trex.y)

    //jump when the space key is pressed
    if(keyDown("space") ) {
        trex.velocityY = -10;
    }
    //add gravity
    trex.velocityY = trex.velocityY + 0.8

    if (ground.x < 0){
        ground.x = ground.width/2;
```

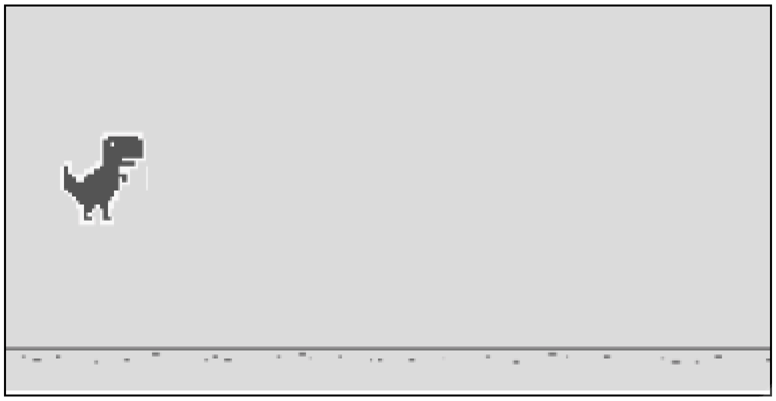


```
162.3    sketch.js:36
161.5    sketch.js:36
162.3    sketch.js:36
161.5    sketch.js:36
162.3    sketch.js:36
161.5    sketch.js:36
162.3    sketch.js:36
161.5    sketch.js:36
162.3    sketch.js:36
161.5    sketch.js:36
162.3    sketch.js:36
161.5    sketch.js:36
162.3    sketch.js:36
161.5    sketch.js:36
162.3    sketch.js:36
161.5    sketch.js:36
162.3    sketch.js:36
161.5    sketch.js:36
```

What do you see? **Note**: The y-position values would change between **162** and **162.8**. The values in the above screenshot are just for reference purposes.	**ESR:** The y position of the Trex changes between 162.3 and 161.5.
Right, and when it jumps what will happen to **trex.y**?	*The student runs the code and makes the Trex jump to see the change in the **trex.y** in the console.* **ESR:** **trex.y** reduces when the Trex jumps.

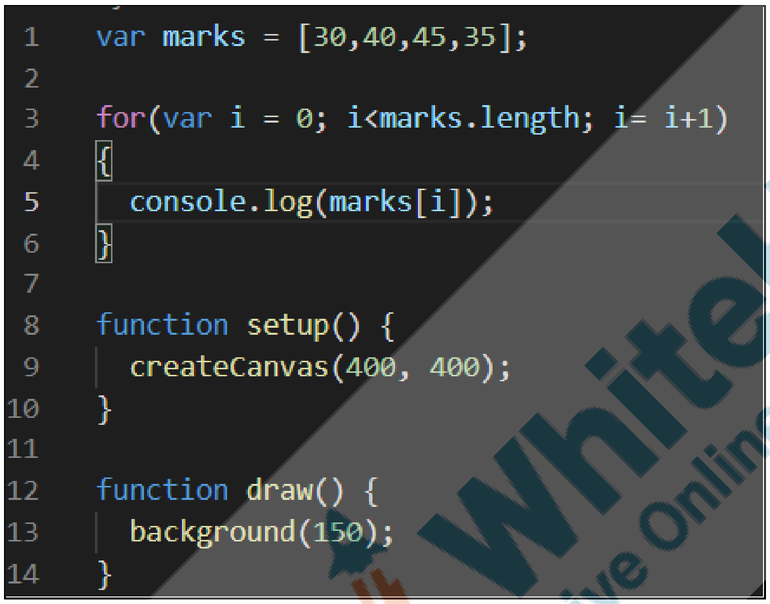
So we want the Trex to jump only when it is on the ground, that is, only when **trex.y >= 100**. How can we do that?	**ESR:** By adding an additional condition inside the **if** block where we make the Trex jump.
Alright, let's do that. *The teacher guides the student to write the additional condition inside the '**If block**'.*	*The student writes the code and runs the program.* *The student can press space repeatedly to see if the program works as expected.*
 **Output:**	

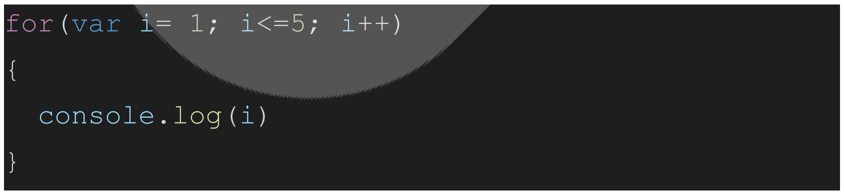
```
function draw() {
  //set background color
  background(220);

  console.log(trex.y)

  //jump when the space key is pressed
  if(keyDown("space") && trex.y >= 100) {
    trex.velocityY = -10;
  }
  //add gravity
  trex.velocityY = trex.velocityY + 0.8

  if (ground.x < 0){
    ground.x = ground.width/2;
  }

  //stop trex from falling down
  trex.collide(invisibleGround);

```



Awesome! We have covered very important concepts of the Trex game in today's class such as making the Trex jump with a key press and making sure Trex only jumps when it is on the ground.	
Teacher Guides Student to Stop Screen Share	
WRAP-UP SESSION - 5 Mins	
Teacher starts slideshow **Slide 33-42**	
Teacher Action	**Student Action**
Run the presentation from slide 33 to slide 42. **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 the in-class quiz	
Question	**Answer**

Debugging is ____________ **A. finding and fixing the bugs**. B. introducing bugs in code. C. running the code. D. writing the code.	**A**

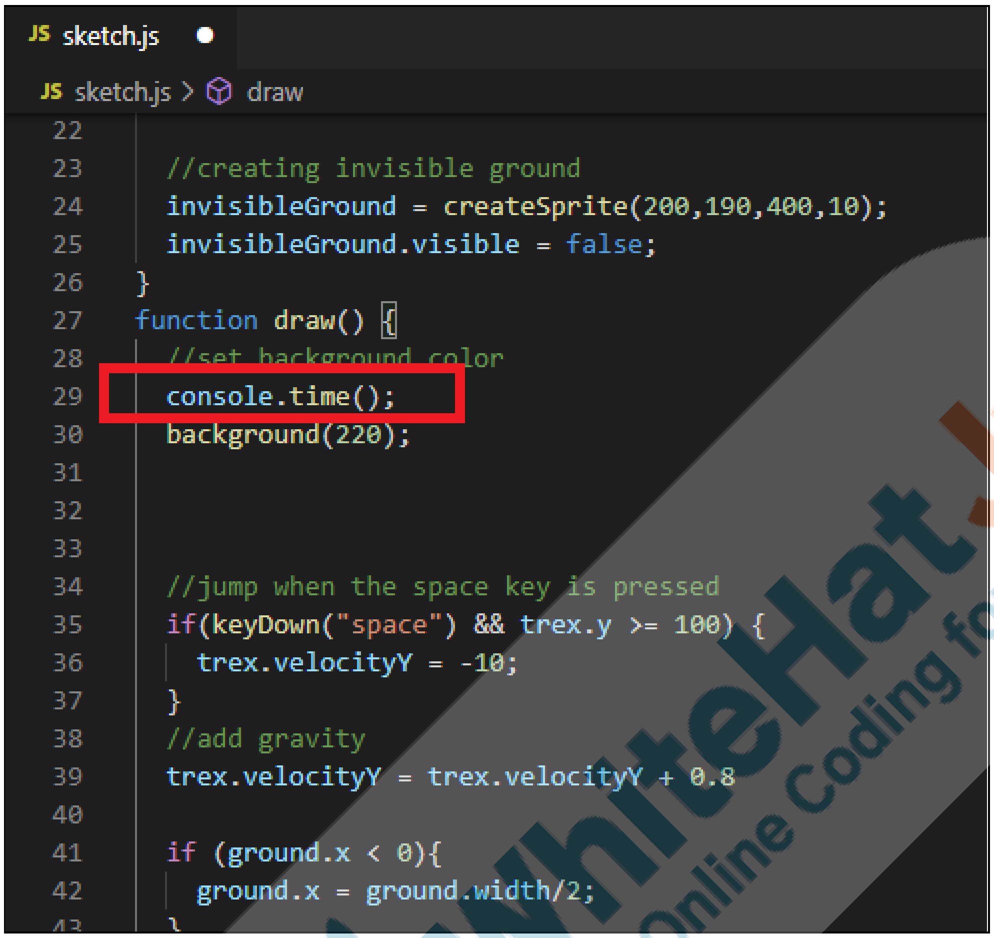
What will be the output of the following code block?

`var marks = [30,40,45,35];`
 	
`for(var i = 0; i<marks.length; i= i+1)`
`{`
`  console.log(marks[i]);`
`}`
 	
`function setup() {`
`  createCanvas(400, 400);`
`}`
 	
`function draw() {`
`  background(150);`
`}`
 	
A. 30,40,45
B. 40,45,35
C. 30,40,45,35
D. undefined
	C

Guess the output of the following code.

`for(var i= 1; i<=5; i++)`
`{`
`  console.log(i)`
`}`
 	
A.1,2,3,4,5
B 1,2,3,4
C 0,1,2,3,4
D. 0,1,2,3,4,5
	A

End the quiz panel	
In the next class, we will look at how to create floating clouds at different heights. Thank you *<student name>* for joining me in this class. I hope to see you again. Looking forward to our next class.	
You get Hats Off for your excellent work!	*Make sure you have given at least 2 Hats Off during the class for:* Creatively Solved Activities +10 Great Question +10 Strong Concentration +10
*** This Project will take only 30 mins to complete. Motivate students to try and finish it immediately after the class.** **Project Overview** **LET'S RUN JAXON** **Goal of the Project:** In this class, we have learned how to indent code and use the console to display the live position of an object. Using visible properties of ground we made ground invisible.	**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.** *Students engage with the teacher over the project.*

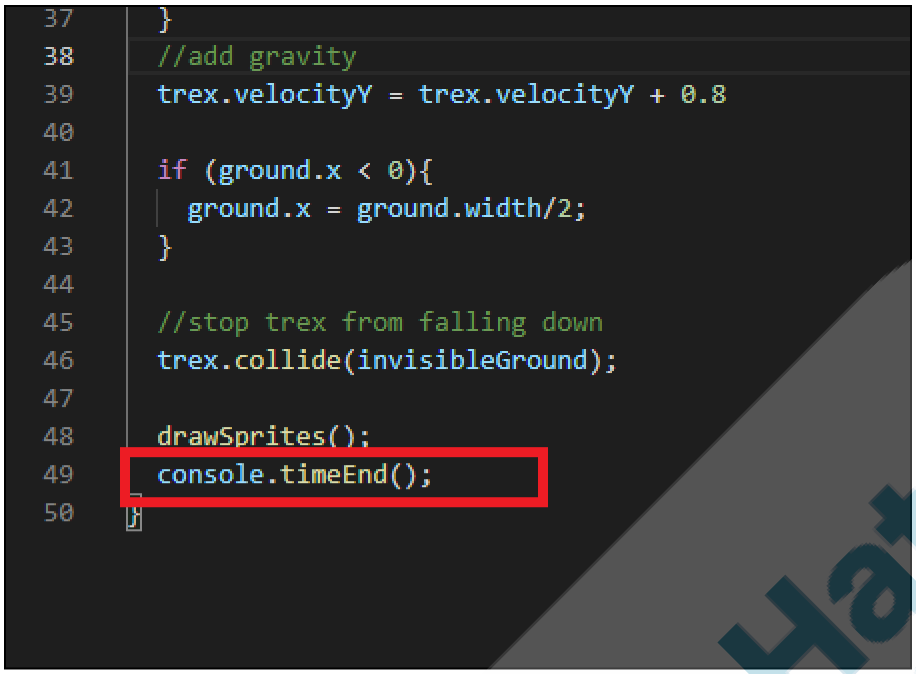
In this project, you have to create a vertically moving background and an animated boy sprite. Create two left and right invisible boundaries and the boy should collide with the left and right invisible boundaries so that the boy does not move out of the canvas. Also, make the boy move left and right using a mouse. **Story:** Jaxon was watching a running race on a sports channel on television; this inspired him to build a Racing Game, however, he thought to just start with one player first, and later on build it more to include more players. He called you up to ask if you can help with your coding knowledge. So he decided to build a computer game. Can you help Jaxon design the game? I am very excited to see your project solution and I know you will do really well. Bye Bye!	
Teacher ends slideshow	
Teacher Clicks ✖ End Class	
ADDITIONAL ACTIVITY - 1	
• **Ask the student to press the ESC key to come back to the panel.** • **Guide the student to start Screen Share.** • **The teacher gets into Fullscreen.**	

You can also use the console to find out how much time it takes your program to run. We use **console.time()** to start keeping a track of time and **console.timeEnd()** to stop and print the time on the console. This is used by programmers when they want to optimize and reduce the time taken by their program to run. *The teacher shows how to use **console.time()** and **console.timeEnd()** to measure the time taken by the **draw()** function to run.*	*The student learns how to use **console.time** and **console.timeEnd** to log the time taken by the program to run.*
Use the **console.time()** when the **draw()** function starts.	



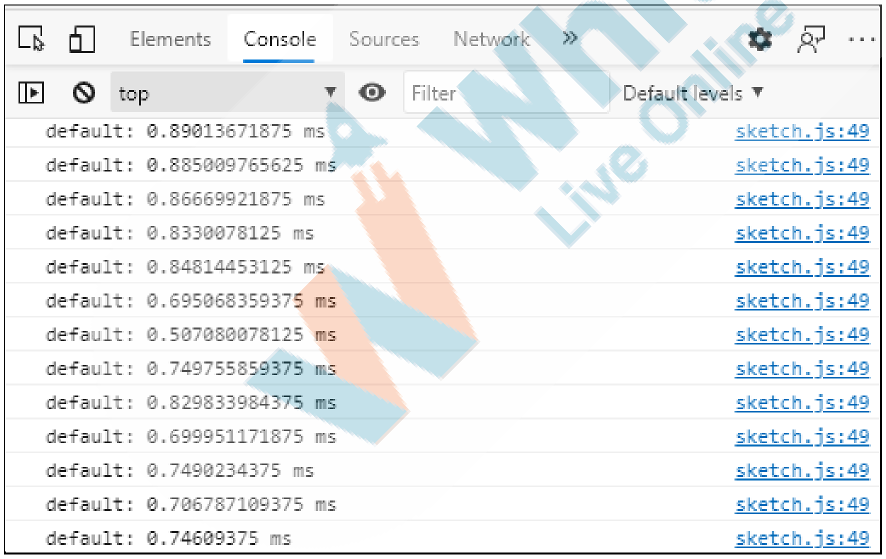
```
	
	//creating invisible ground
	invisibleGround = createSprite(200,190,400,10);
	invisibleGround.visible = false;
}
function draw() {
	//set background color
	console.time();
	background(220);



	//jump when the space key is pressed
	if(keyDown("space") && trex.y >= 100) {
		trex.velocityY = -10;
	}
	//add gravity
	trex.velocityY = trex.velocityY + 0.8

	if (ground.x < 0){
		ground.x = ground.width/2;
	}
```

Next, use the **console.timeEnd()** when the **draw()** function ends.



```
  }
  //add gravity
  trex.velocityY = trex.velocityY + 0.8

  if (ground.x < 0){
    ground.x = ground.width/2;
  }

  //stop trex from falling down
  trex.collide(invisibleGround);

  drawSprites();
  console.timeEnd();
}
```

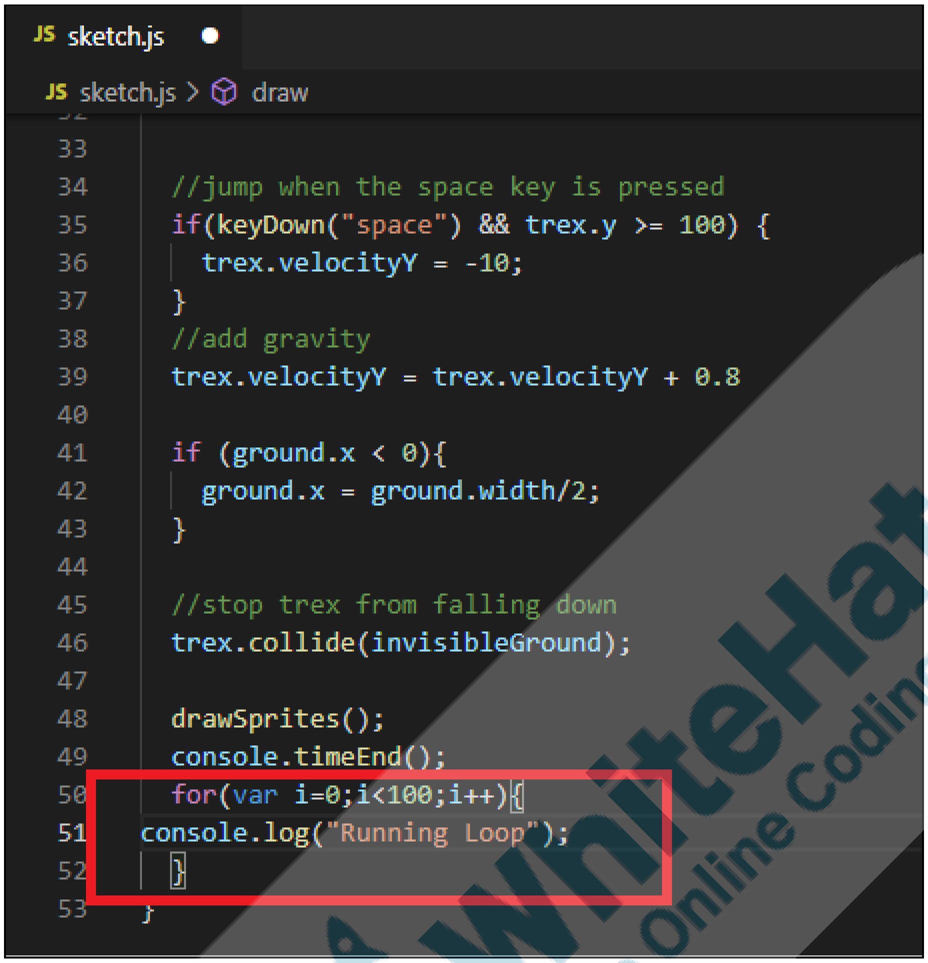
Output:



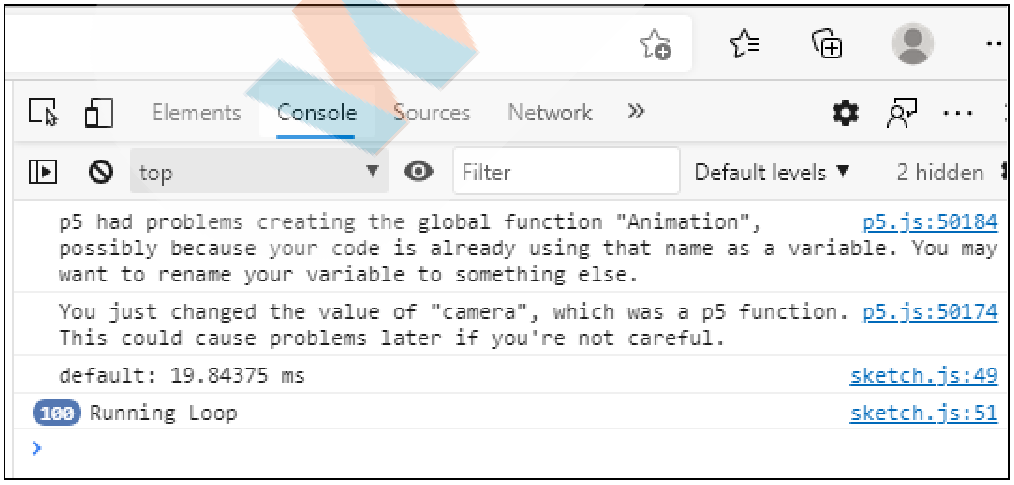
```
default: 0.89013671875 ms        sketch.js:49
default: 0.885009765625 ms       sketch.js:49
default: 0.86669921875 ms        sketch.js:49
default: 0.8330078125 ms         sketch.js:49
default: 0.84814453125 ms        sketch.js:49
default: 0.695068359375 ms       sketch.js:49
default: 0.507080078125 ms       sketch.js:49
default: 0.749755859375 ms       sketch.js:49
default: 0.829833984375 ms       sketch.js:49
default: 0.699951171875 ms       sketch.js:49
default: 0.7490234375 ms         sketch.js:49
default: 0.706787109375 ms       sketch.js:49
default: 0.74609375 ms           sketch.js:49
```

Similarly, you can also find out how long it takes for the function **setup()** or function **preload()** to run before your game can start.

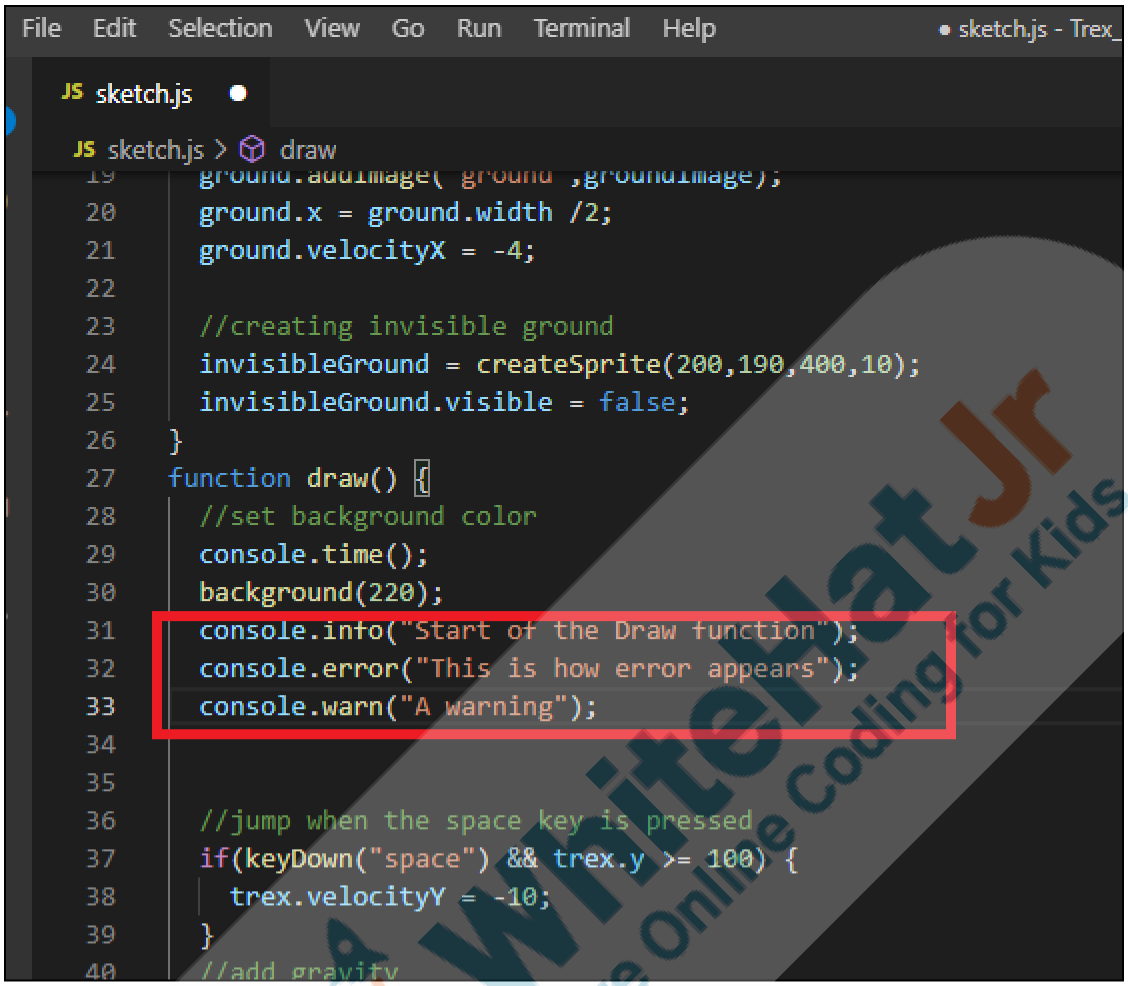
Note: Observe how it takes different time each time the **draw()** function runs. The variation is because your computer's processing speed depends on a lot of factors like - how heated your computer chips are, what are the other things your computer is doing - for example, what else is happening on your browser, etc. Let us write a simple **for** loop inside the **draw()** function. Log anything inside the **for** loop and check if the execution time of the **draw()** function changes. Also, observe the effect of this on your game. *Ask the student to explain the reason behind the lag in the game.*	*The student writes a simple **for-loop** inside the function **draw()** and observes the change in execution time of the draw function.* *The student observes the lag in the game - where every character slows down and gives an impression of the game being unresponsive.* **ESR:** Every frame in the game is rendered (drawn) each time the **draw()** function gets called. The lag in the game is because it takes longer for the next frame to render.


```
  //jump when the space key is pressed
  if(keyDown("space") && trex.y >= 100) {
    trex.velocityY = -10;
  }
  //add gravity
  trex.velocityY = trex.velocityY + 0.8

  if (ground.x < 0){
    ground.x = ground.width/2;
  }

  //stop trex from falling down
  trex.collide(invisibleGround);

  drawSprites();
  console.timeEnd();
  for(var i=0;i<100;i++){
console.log("Running Loop");
  }
}
```

Output:

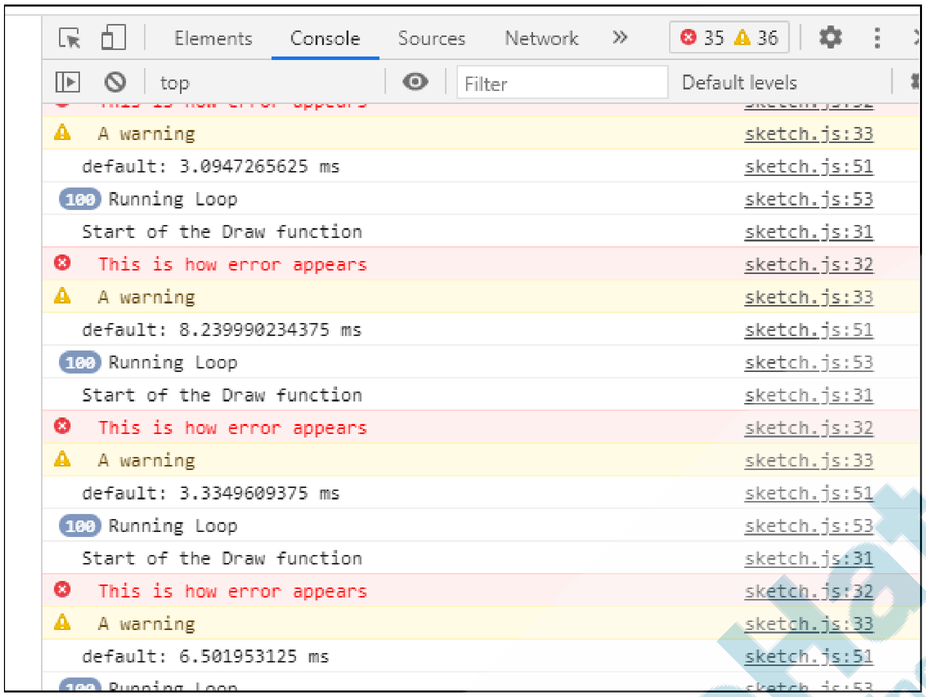

```
p5 had problems creating the global function "Animation",        p5.js:50184
possibly because your code is already using that name as a variable. You may
want to rename your variable to something else.
You just changed the value of "camera", which was a p5 function. p5.js:50174
This could cause problems later if you're not careful.
default: 19.84375 ms                                              sketch.js:49
100 Running Loop                                                  sketch.js:51
>
```

Good! Our goal should always be to write programs that run in the least time possible. There are other ways you can use the console. **console.log()** is used to print a simple message. You can use **console.warn()** to print a warning. The warning message is formatted differently. Similarly, you can use **console.error()** to print an **error()**. The error message is formatted differently. You can also use **console.info()** to print any information. *The teacher shows how to print information, errors, and warnings on the console.* These are especially helpful when you are working on a big project with several other developers. You want your program to be meaningful for them.	*The student experiments with different types of console messages.*



```
    ground.addImage("ground",groundImage);
    ground.x = ground.width /2;
    ground.velocityX = -4;

    //creating invisible ground
    invisibleGround = createSprite(200,190,400,10);
    invisibleGround.visible = false;
  }
  function draw() {
    //set background color
    console.time();
    background(220);
    console.info("Start of the Draw function");
    console.error("This is how error appears");
    console.warn("A warning");


    //jump when the space key is pressed
    if(keyDown("space") && trex.y >= 100) {
      trex.velocityY = -10;
    }
    //add gravity
```

Output:



```
⚠ A warning                                   sketch.js:33
  default: 3.0947265625 ms                    sketch.js:51
100 Running Loop                              sketch.js:53
  Start of the Draw function                  sketch.js:31
⊗ This is how error appears                   sketch.js:32
⚠ A warning                                   sketch.js:33
  default: 8.239990234375 ms                  sketch.js:51
100 Running Loop                              sketch.js:53
  Start of the Draw function                  sketch.js:31
⊗ This is how error appears                   sketch.js:32
⚠ A warning                                   sketch.js:33
  default: 3.3349609375 ms                    sketch.js:51
100 Running Loop                              sketch.js:53
  Start of the Draw function                  sketch.js:31
⊗ This is how error appears                   sketch.js:32
⚠ A warning                                   sketch.js:33
  default: 6.501953125 ms                     sketch.js:51
```

In this activity, we have learned how to use the different options available with the console. We can display the error using the console.error(), warning using console.warn() and any information using console.info().	
ADDITIONAL ACTIVITY - 2	
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. ● Code I wrote. ● How did I feel after the class? ● What have I learned about programming and developing games?	*The student uses the Markdown editor to write her/his reflection as a reflection journal.*

● What aspects of the class helped me? What did I find difficult?	

Links:

Activity	Activity Name	Links
Teacher Activity 1	Traversing Array with For loop	https://github.com/pro-whitehatjr/C-11_teacher_activity-1
Teacher Activity 1 solution	Reference code for Teacher Activity 1	https://github.com/pro-whitehatjr/C11_TA-1-reference
Teacher Activity 2	Unindented code	https://github.com/pro-whitehatjr/c-11_unindented_code
Teacher Activity 2	Solution for TA-2 indented code	https://github.com/whitehatjr/C_11_indented_code
Teacher Activity Ref	Complete code for reference	https://github.com/pro-whitehatjr/C-11_trex_stage_2
Teacher Reference	Reference code for Student Activity 1	https://github.com/pro-whitehatjr/pro-c11-sa1-reference
Student Activity 1	Template Code	https://github.com/pro-whitehatjr/Pro-c11-sa1-template
Student Activity 2	Code	https://github.com/pro-whitehatjr/C_11_Unindented_code_2

Teacher Reference	Teacher reference for Student activity 2 solution	https://github.com/whitehatjr/C_11_Student-Activity-2_Solution
Teacher Reference visual aid link	Visual aid link	https://curriculum.whitehatjr.com/Visual+Project+Asset/PRO_VD/BJFC-PRO-V3-C11-withcues.html
Teacher Reference In-class quiz	In-class quiz	https://s3-whjr-curriculum-uploads.whjr.online/749c07b8-c376-40b3-a649-1e5d282da041.pdf
Project Solution	Let's Run Jaxon	https://github.com/pro-whitehatjr/Project_C11_Let-s_Run_JAXON