Topic	**Capstone Class: T-Rex Game Code**	
Class Description	**Students will continue rewriting the code for Trex game on the p5 editor. Students will learn to use a switch statement to instruct the computer to perform different actions based on different conditions. Students will also learn how to start a small local web server, download the files and host it on it so that they can see the files. Students will also rewrite code for adding score and spawning different game objects such as clouds and obstacles.**	
Class	**C18**	
Class time	**45 mins**	
Goal	● Rewrite the Trex Game code for spawning game objects - clouds and obstacles - and adding score. ● Use switch statement to assign different actions for the computer based on the different conditions. ● Start a small local webserver and host the files locally to see the game.	
Resources Required	● Teacher Resources ○ p5 editor login ○ Laptop with internet connectivity ○ Earphones with mic ○ Notebook and pen ● Student Resources ○ p5 editor login ○ Laptop with internet connectivity ○ Earphones with mic ○ Notebook and pen	
Class structure	**Warm Up** **Teacher-led Activity** **Student-led Activity** **Wrap up**	**5 mins** **5 min** **25 mins** **5 mins**

CONTEXT

- **Continue rewriting code so that the game can run on any browser and we can host the game online.**

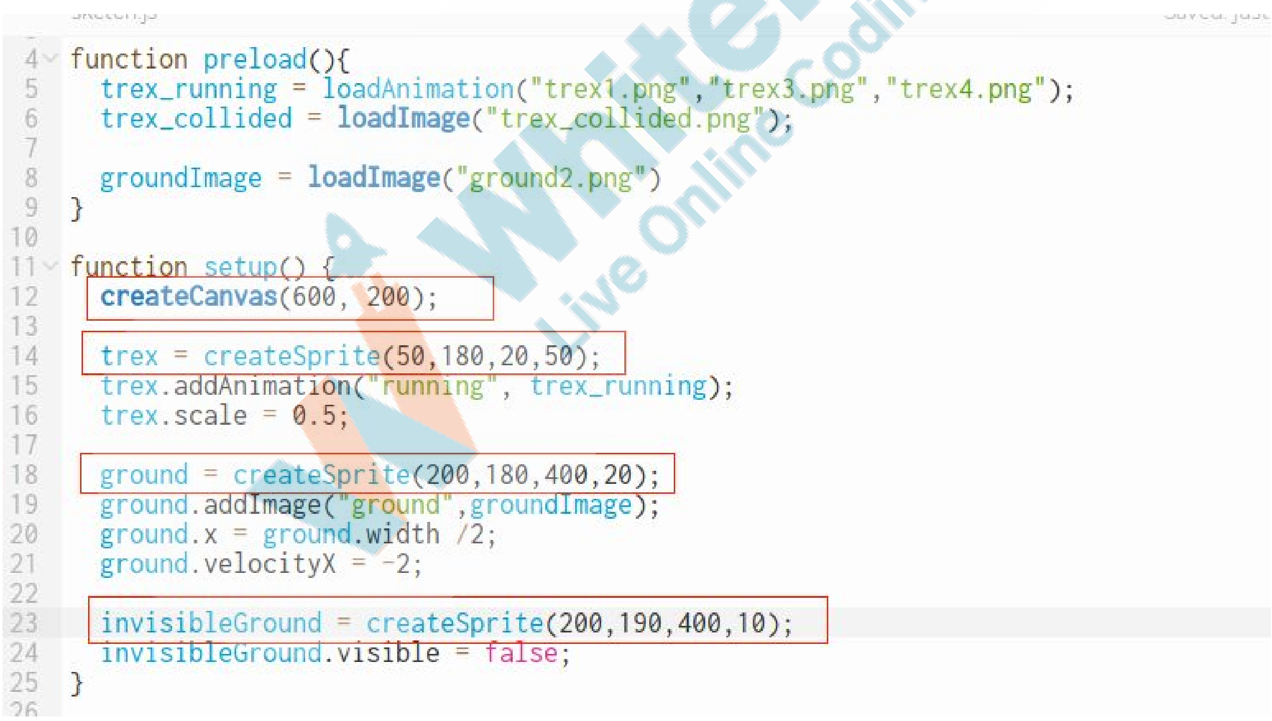
● **Abstract explanation of web server and how it hosts web content.**		
Class Steps	**Teacher Action**	**Student Action**
Step 1: **Warm Up** **(5 mins)**	Hi <Student Name>. Welcome to the Capstone Class. Do you remember why this class is special? It's time to bring out your coder's hat and take the Trex game to the next level!! Excited? Let the fun ride begin! But before we proceed, do you want to recollect what we were doing in the last class?	ESR: - We learned about the different kinds of files - html, css, javascript - which is used to display web content online. - We learned about the p5.play library and we used it to write our game. - We started to write our game outside code.org on a p5 editor so that our game could run outside code.org on any browser.
	Awesome, can you also recollect what does html, css and javascript files do?	ESR: - html : uses tags to display the content. Also used to add the different javascript libraries. - javascript: where we write our code. - css: used for formatting the page.
	Great. Today, we will continue to re-write code for the trex game so that it can run outside code.org platform.. Today, we will also learn how to start a web server. Do you know what is a web-server? What does the name tell you?	 ESR: varied

<table>
<tr><td></td><td>As the name suggests, a web server is a computer which serves web pages. When you enter google.com on your browser, there is a computer/webserver somewhere which is listening for requests, and as soon as it gets your request - it serves or shows you the google.com page.

Isn't that interesting?</td><td>ESR:
Yes!</td></tr>
<tr><td></td><td>At the end of the class, we will be creating our own little webserver on our computer which will serve our game. Let's start today's class.</td><td>-</td></tr>
<tr><td colspan="3"><b>Teacher Initiates Screen Share</b></td></tr>
<tr><td colspan="3"><b><u>CHALLENGE</u></b>
● <b>Make small adjustments in the code based on the p5.play documentation.</b></td></tr>
<tr><td><b>Step 2:
Teacher-led Activity
(5 min)</b></td><td>Can you recall why were we rewriting our code from code.org to p5 editor?</td><td>ESR:
code.org had made it easier for us to write our code to make games/programs. However, it would only run inside code.org platform. We want our code to run on any browser anywhere - so were rewriting the code.</td></tr>
<tr><td></td><td>Yes, and we also have the advantage of changing certain things that code.org platform did not allow us to do.

For example, do you remember what was the screen size we had to live with in code.org?</td><td>ESR:
Width: 400
Height: 400</td></tr>
</table>

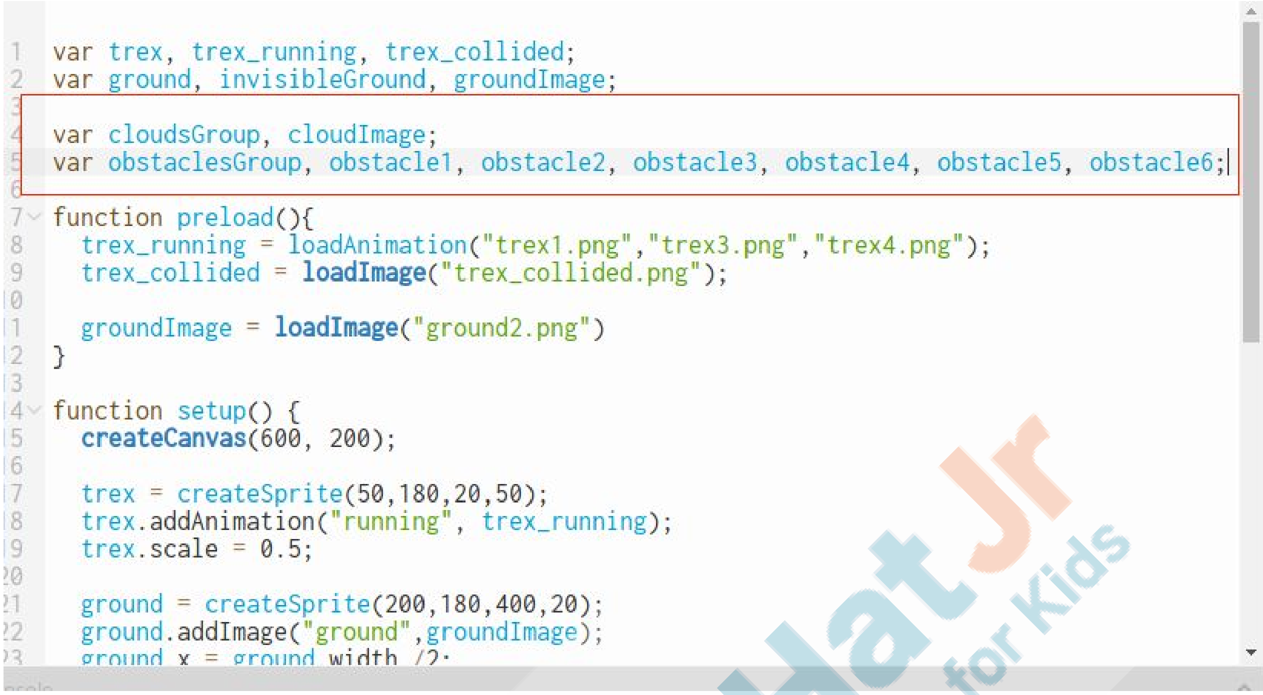
	Could we change it?	No!
	But we don't need to restrict ourselves with that size now. Let's change the canvas size we have so that we have a longer width and smaller height. We will also have to adjust the size of other places where we are creating the sprites so that they are created within the new canvas size. Teacher opens **Teacher Activity 1** and adjusts the canvas size. Teacher also makes corrections to other places in the code.	Student observes and learns.



```
function preload(){
  trex_running = loadAnimation("trex1.png","trex3.png","trex4.png");
  trex_collided = loadImage("trex_collided.png");

  groundImage = loadImage("ground2.png")
}

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

  trex = createSprite(50,180,20,50);
  trex.addAnimation("running", trex_running);
  trex.scale = 0.5;

  ground = createSprite(200,180,400,20);
  ground.addImage("ground",groundImage);
  ground.x = ground.width /2;
  ground.velocityX = -2;

  invisibleGround = createSprite(200,190,400,10);
  invisibleGround.visible = false;
}
```

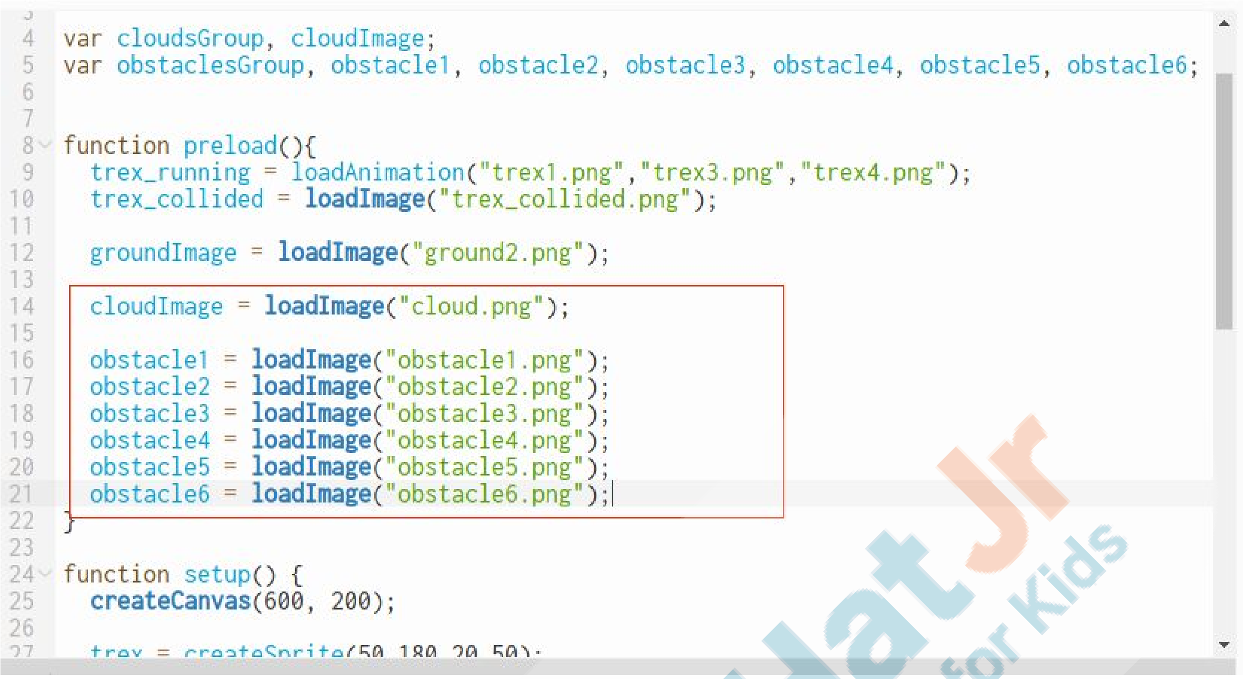
	So, it is slightly more difficult to rewrite code but it gives us a lot more freedom.	

	Alright, let's create the clouds and the obstacles in our game in p5 editor before we use web server to host our game. How does that sound?	ESR: Exciting!
Teacher Stops Screen Share		
	Now it's your turn. Please share your screen with me.	
• **Ask Student to press ESC key to come back to panel** • **Guide Student to start Screen Share** • **Teacher gets into Fullscreen**		
ACTIVITY • **Student modify the javascript code of the trex game for the p5 editor.** • **Student run the code to identify errors.**		
Step 3: **Student-Led Activity** **(25 mins)**	Guide the student to create variables which will store the cloud and the obstacle images. Also guide the student to create variables to store cloud group and obstacle group.	Student opens **Student Activity 1.** Student writes code to create the variables.



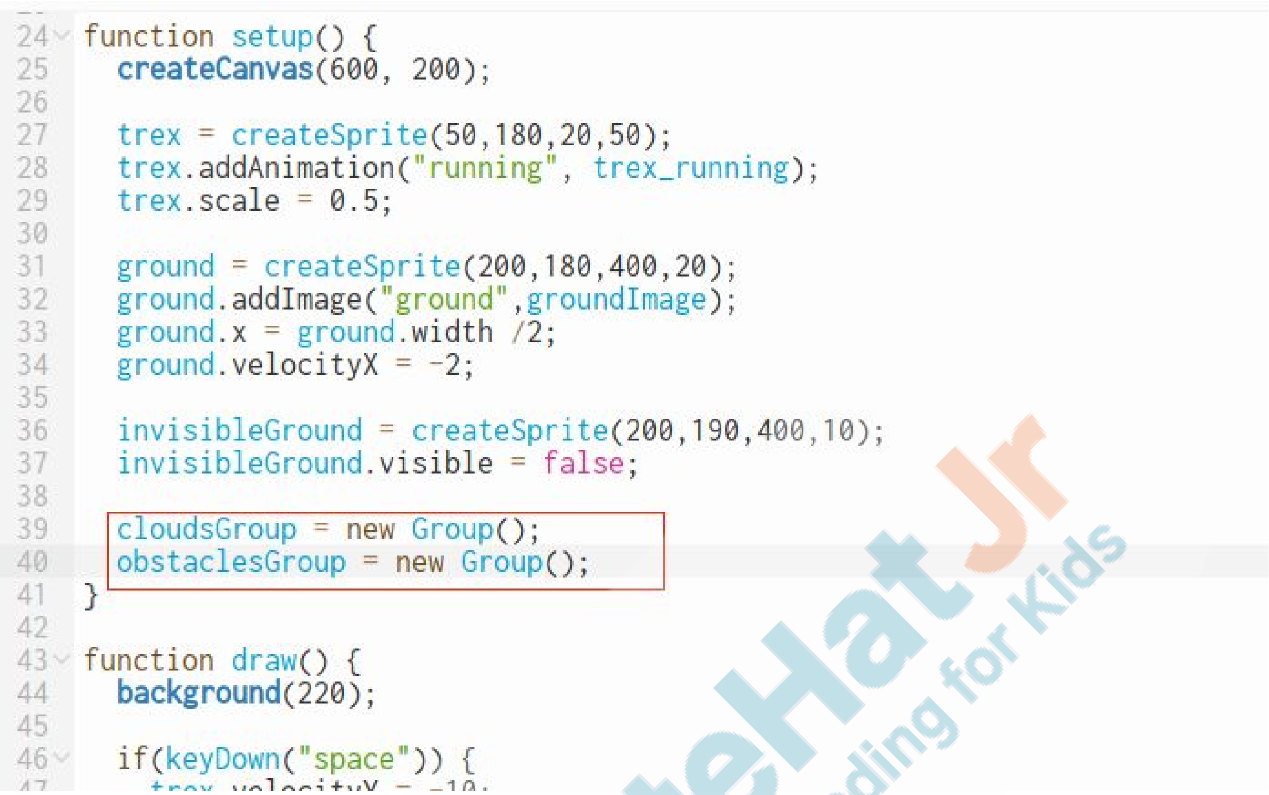
```
var trex, trex_running, trex_collided;
var ground, invisibleGround, groundImage;

var cloudsGroup, cloudImage;
var obstaclesGroup, obstacle1, obstacle2, obstacle3, obstacle4, obstacle5, obstacle6;

function preload(){
  trex_running = loadAnimation("trex1.png","trex3.png","trex4.png");
  trex_collided = loadImage("trex_collided.png");

  groundImage = loadImage("ground2.png")
}

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

  trex = createSprite(50,180,20,50);
  trex.addAnimation("running", trex_running);
  trex.scale = 0.5;

  ground = createSprite(200,180,400,20);
  ground.addImage("ground",groundImage);
  ground.x = ground.width /2;
```

	Let us load the images into these variables. Do you remember where we load the images? Let's do it. Guide the student to load the images into the variables.	ESR: Inside function preload() Student writes code to load the images inside function preload().



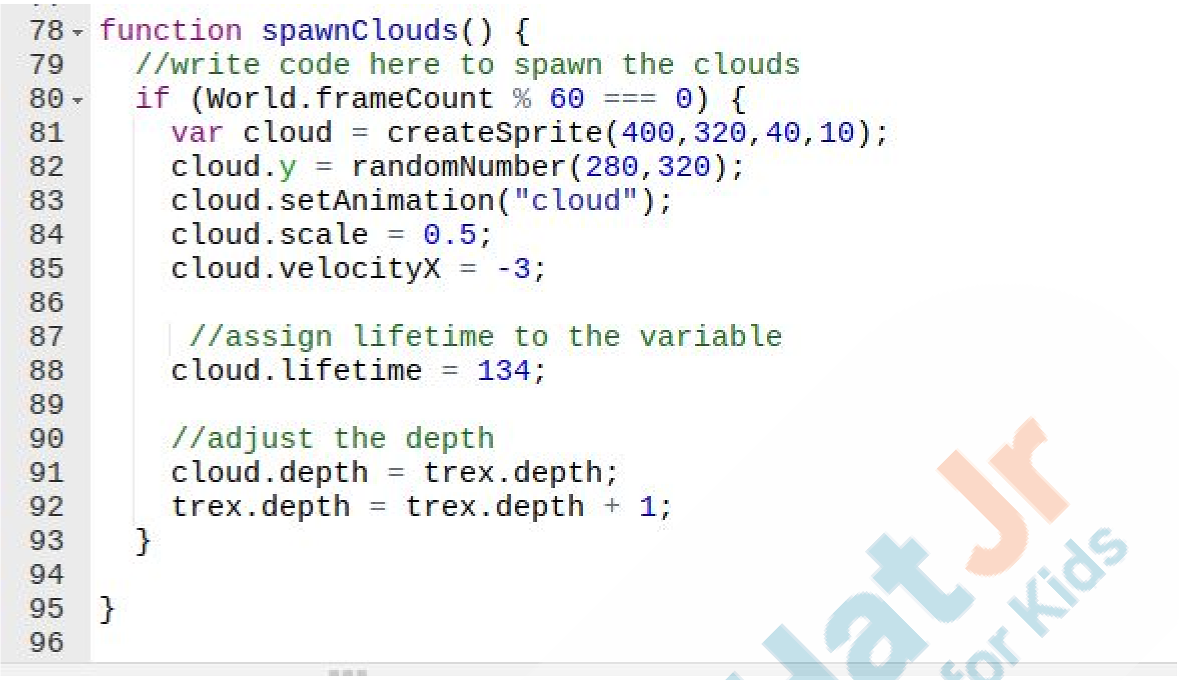
```
var cloudsGroup, cloudImage;
var obstaclesGroup, obstacle1, obstacle2, obstacle3, obstacle4, obstacle5, obstacle6;


function preload(){
  trex_running = loadAnimation("trex1.png","trex3.png","trex4.png");
  trex_collided = loadImage("trex_collided.png");

  groundImage = loadImage("ground2.png");

  cloudImage = loadImage("cloud.png");

  obstacle1 = loadImage("obstacle1.png");
  obstacle2 = loadImage("obstacle2.png");
  obstacle3 = loadImage("obstacle3.png");
  obstacle4 = loadImage("obstacle4.png");
  obstacle5 = loadImage("obstacle5.png");
  obstacle6 = loadImage("obstacle6.png");
}

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

  trex = createSprite(50,180,20,50);
```

	We need to create a group which can hold all the clouds and all the obstacles. In code.org we did it by createGroup(). Here, we use new Group() to create a group. Remember we used something like this when we were designing our own Paddle object. “new” helps create a new Object from a blueprint. Here, we are creating a new Group object. We will learn more about what "new " is in later classes. Guide the student to create cloudsGroup and obstaclesGroup using new Group()	Student writes code to create the groups.



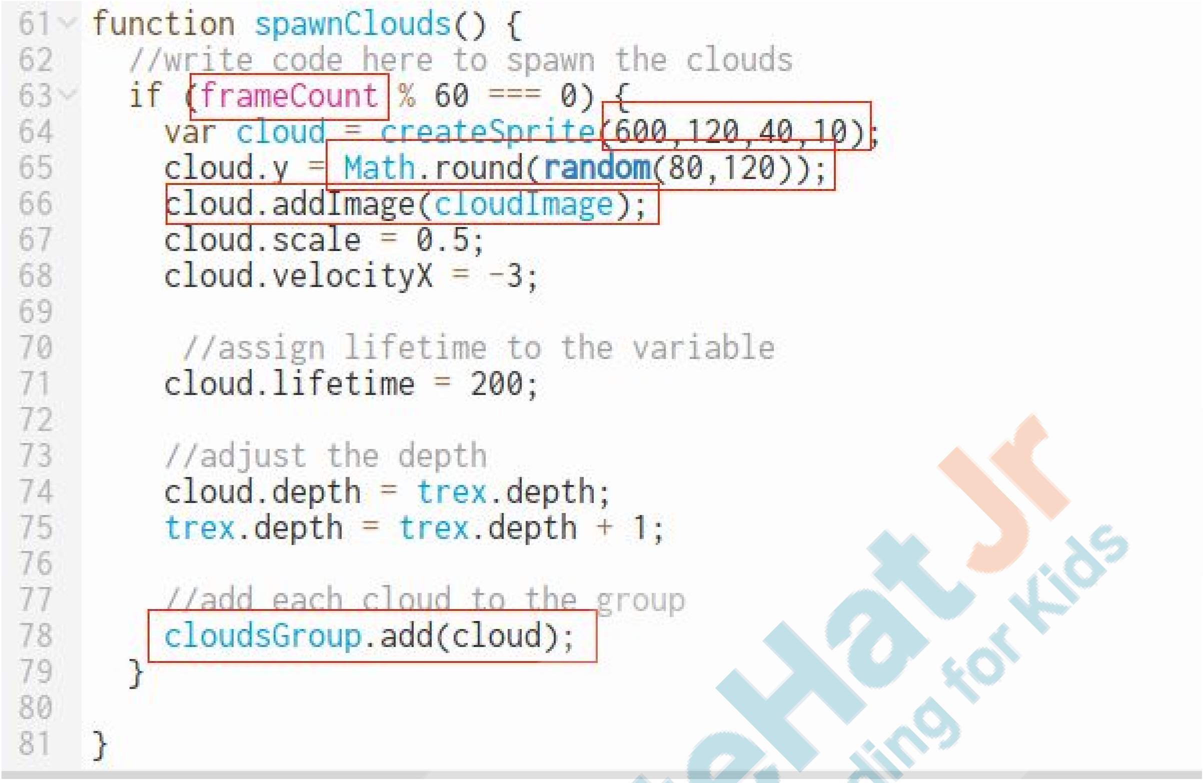
```
function setup() {
  createCanvas(600, 200);

  trex = createSprite(50,180,20,50);
  trex.addAnimation("running", trex_running);
  trex.scale = 0.5;

  ground = createSprite(200,180,400,20);
  ground.addImage("ground",groundImage);
  ground.x = ground.width /2;
  ground.velocityX = -2;

  invisibleGround = createSprite(200,190,400,10);
  invisibleGround.visible = false;

  cloudsGroup = new Group();
  obstaclesGroup = new Group();
}

function draw() {
  background(220);

  if(keyDown("space")) {
    trex.velocityY = -10;
```

	Let's refer to our code.org project to see how we created function for spawning clouds. We can use the code and make changes to it so that it can be run on our p5 editor.	Student Opens Student Activity 2. Student copies code for spawning clouds.



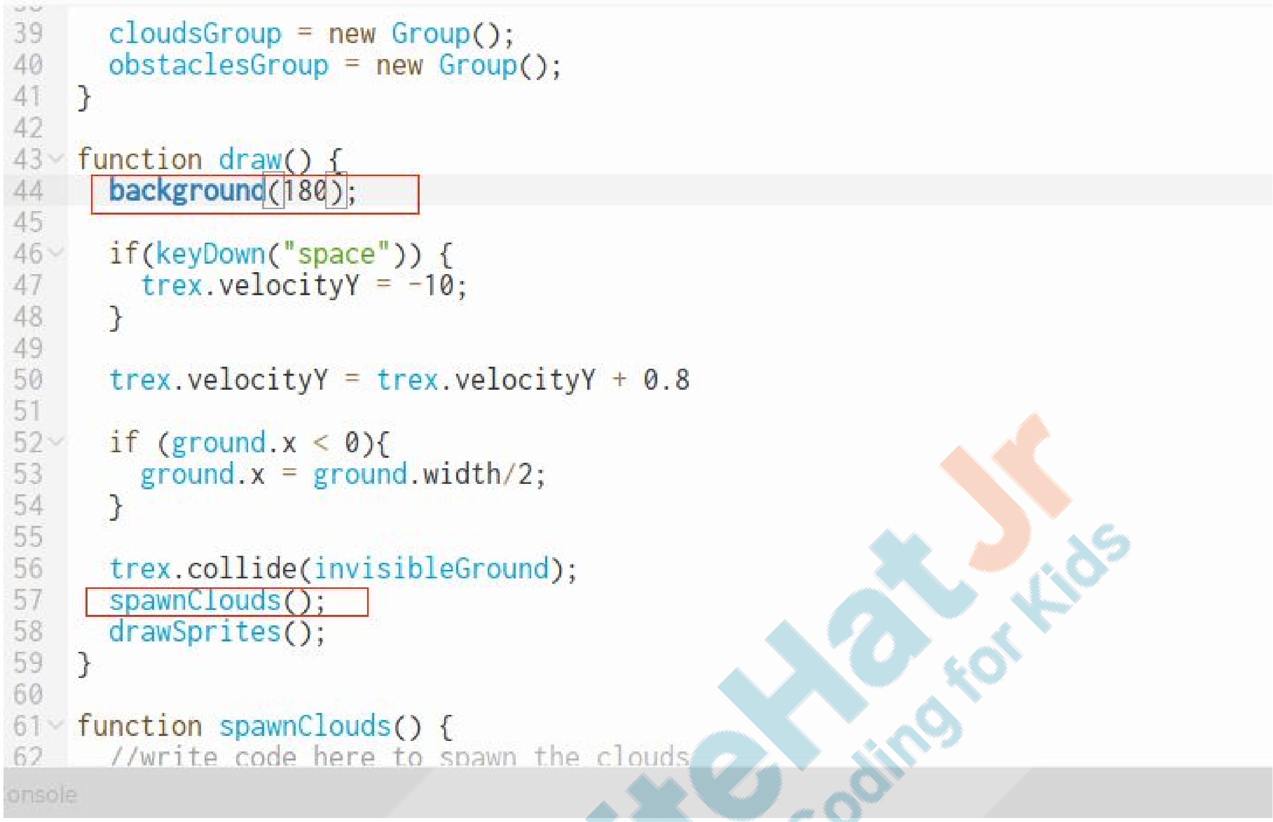
```
function spawnClouds() {
  //write code here to spawn the clouds
  if (World.frameCount % 60 === 0) {
    var cloud = createSprite(400,320,40,10);
    cloud.y = randomNumber(280,320);
    cloud.setAnimation("cloud");
    cloud.scale = 0.5;
    cloud.velocityX = -3;

     //assign lifetime to the variable
    cloud.lifetime = 134;

    //adjust the depth
    cloud.depth = trex.depth;
    trex.depth = trex.depth + 1;
  }

}

```

	Guide the student to make changes in their code in the function spawnClouds() so that it can run on the p5 editor. Notice we were using cloud.setAnimation("cloud"), but now we are calling cloud.addAnimation(cloudImage). Notice we were using a string inside the function but now we are using a variable. (Recall what is a string). We will see why it is important later.	Student makes the changes.



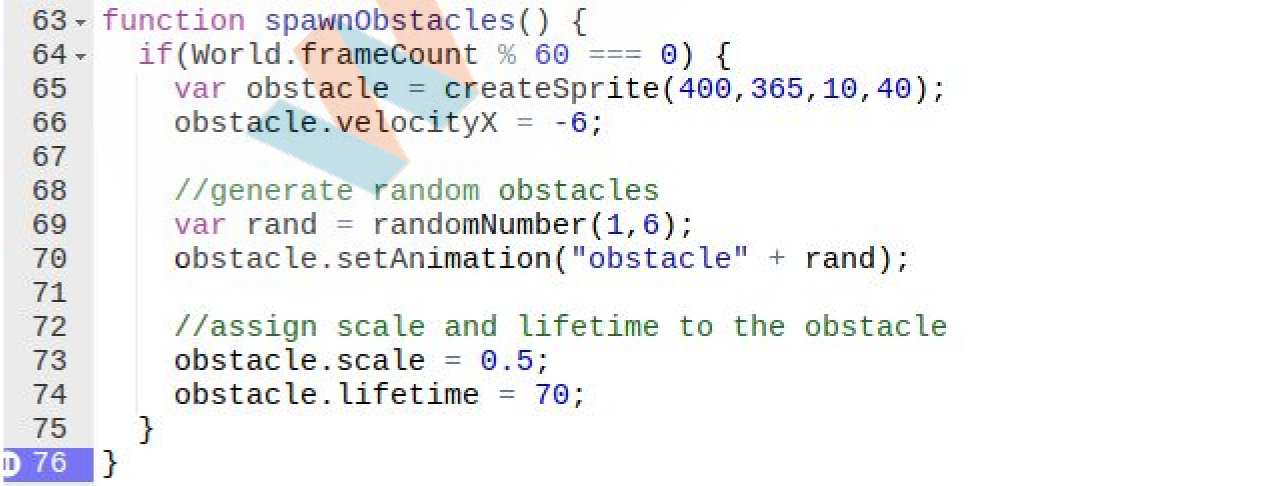
```
function spawnClouds() {
  //write code here to spawn the clouds
  if (frameCount % 60 === 0) {
    var cloud = createSprite(600,120,40,10);
    cloud.y = Math.round(random(80,120));
    cloud.addImage(cloudImage);
    cloud.scale = 0.5;
    cloud.velocityX = -3;

     //assign lifetime to the variable
    cloud.lifetime = 200;

    //adjust the depth
    cloud.depth = trex.depth;
    trex.depth = trex.depth + 1;

    //add each cloud to the group
    cloudsGroup.add(cloud);
  }

}
```

	Let's call the function in our code. Also, let's make the background a little darker - so that the clouds are clearly visible. Remind the student how the numbers inside the background can represent colors: 0 -> black 255-> white numbers in between are various shades of grey	Student makes the changes and runs the code.



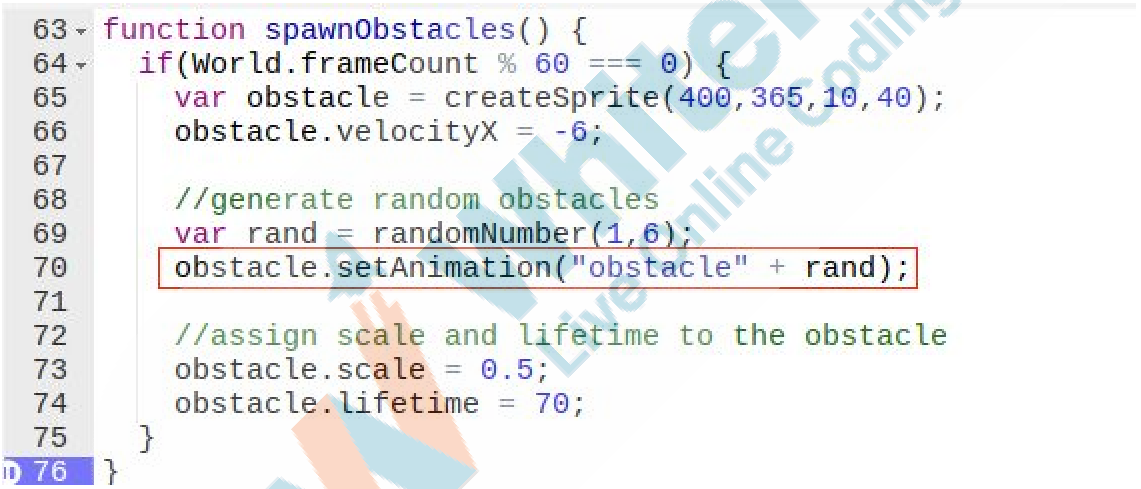
```
  cloudsGroup = new Group();
  obstaclesGroup = new Group();
}

function draw() {
  background(180);

  if(keyDown("space")) {
    trex.velocityY = -10;
  }

  trex.velocityY = trex.velocityY + 0.8

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

  trex.collide(invisibleGround);
  spawnClouds();
  drawSprites();
}

function spawnClouds() {
  //write code here to spawn the clouds
```

	Let's do something similar with our spawnObstacle function. Guide the student to copy code from code.org for spawnObstacles()	Student copies code for spawning obstacles.



```
function spawnObstacles() {
  if(World.frameCount % 60 === 0) {
    var obstacle = createSprite(400,365,10,40);
    obstacle.velocityX = -6;

    //generate random obstacles
    var rand = randomNumber(1,6);
    obstacle.setAnimation("obstacle" + rand);

    //assign scale and lifetime to the obstacle
    obstacle.scale = 0.5;
    obstacle.lifetime = 70;
  }
}
```

	Let's modify the code. We will face a peculiar problem in obstacle.setAnimation(). We were using string concatenation here to get different obstacles depending on the random number. (Help the student recall string concatenation). Now when we are using obstacle.addAnimation() , it expects us to supply it a variable. We cannot use string concatenation anymore. Any ideas on what we could do?	ESR: varied



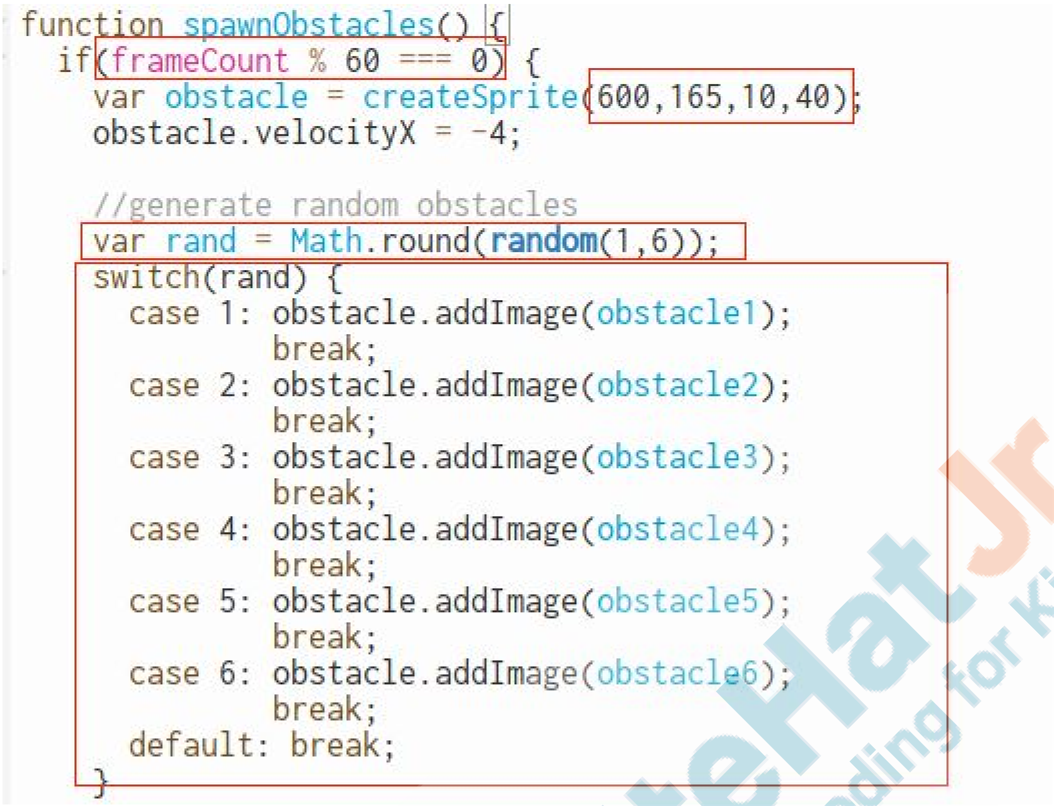
```
function spawnObstacles() {
  if(World.frameCount % 60 === 0) {
    var obstacle = createSprite(400,365,10,40);
    obstacle.velocityX = -6;

    //generate random obstacles
    var rand = randomNumber(1,6);
    obstacle.setAnimation("obstacle" + rand);

    //assign scale and lifetime to the obstacle
    obstacle.scale = 0.5;
    obstacle.lifetime = 70;
  }
}
```

	We could write several if-else statements instructing the computer to supply different variables to addAnimation depending on the value of the random number. Typically however, developers like to use switch statements in these cases when there are many conditions.	ESR: Student makes the changes and runs the code.

	We write the variable in switch which we are evaluating and which can change. We assign condition as different cases. The computer picks up a case depending on the condition, executes the instruction that follows and breaks out of the switch statement. There is often a default case, which the computer executes if none of the condition is met. Guide the student to write the switch statement and make other changes in spawn clouds()	

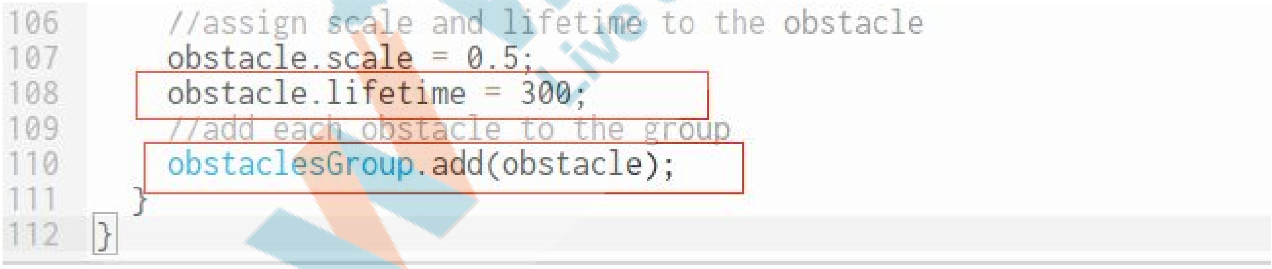
```
switch(expression) {
  case x:
    // code block
    break;
  case y:
    // code block
    break;
  default:
    // code block
}
```

```
In switch statement, the computer evaluates the expression, if the
expression equals 'x', only the code under case x is run.
In case no, cases are satisified, the code under default is run.
```



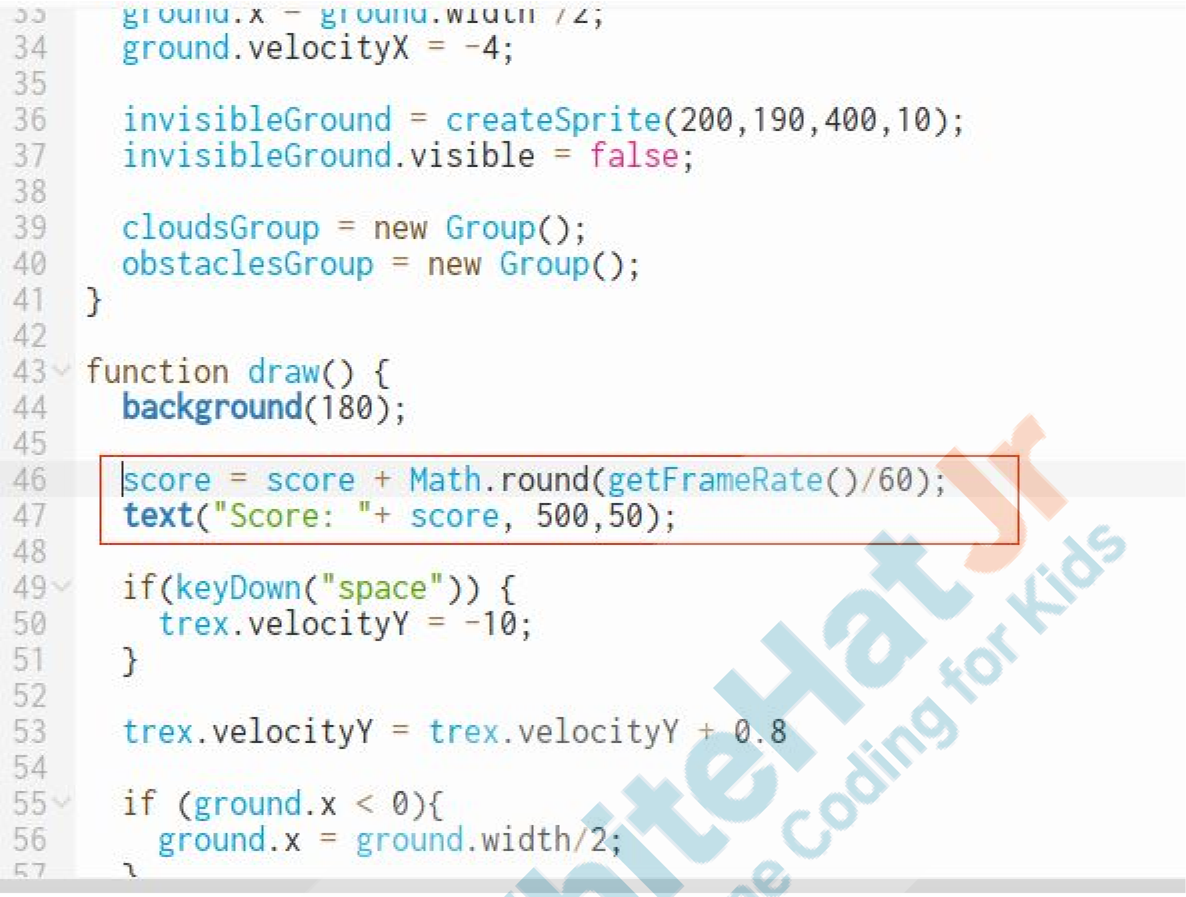
```
function spawnObstacles() {
  if(frameCount % 60 === 0) {
    var obstacle = createSprite(600,165,10,40);
    obstacle.velocityX = -4;

    //generate random obstacles
    var rand = Math.round(random(1,6));
    switch(rand) {
      case 1: obstacle.addImage(obstacle1);
              break;
      case 2: obstacle.addImage(obstacle2);
              break;
      case 3: obstacle.addImage(obstacle3);
              break;
      case 4: obstacle.addImage(obstacle4);
              break;
      case 5: obstacle.addImage(obstacle5);
              break;
      case 6: obstacle.addImage(obstacle6);
              break;
      default: break;
    }
```

	Let's call the spawnObstacles() inside function draw and see if our code works.	Student calls spawnClouds() and runs the code.



```
    //assign scale and lifetime to the obstacle
    obstacle.scale = 0.5;
    obstacle.lifetime = 300;
    //add each obstacle to the group
    obstaclesGroup.add(obstacle);
  }
}
```

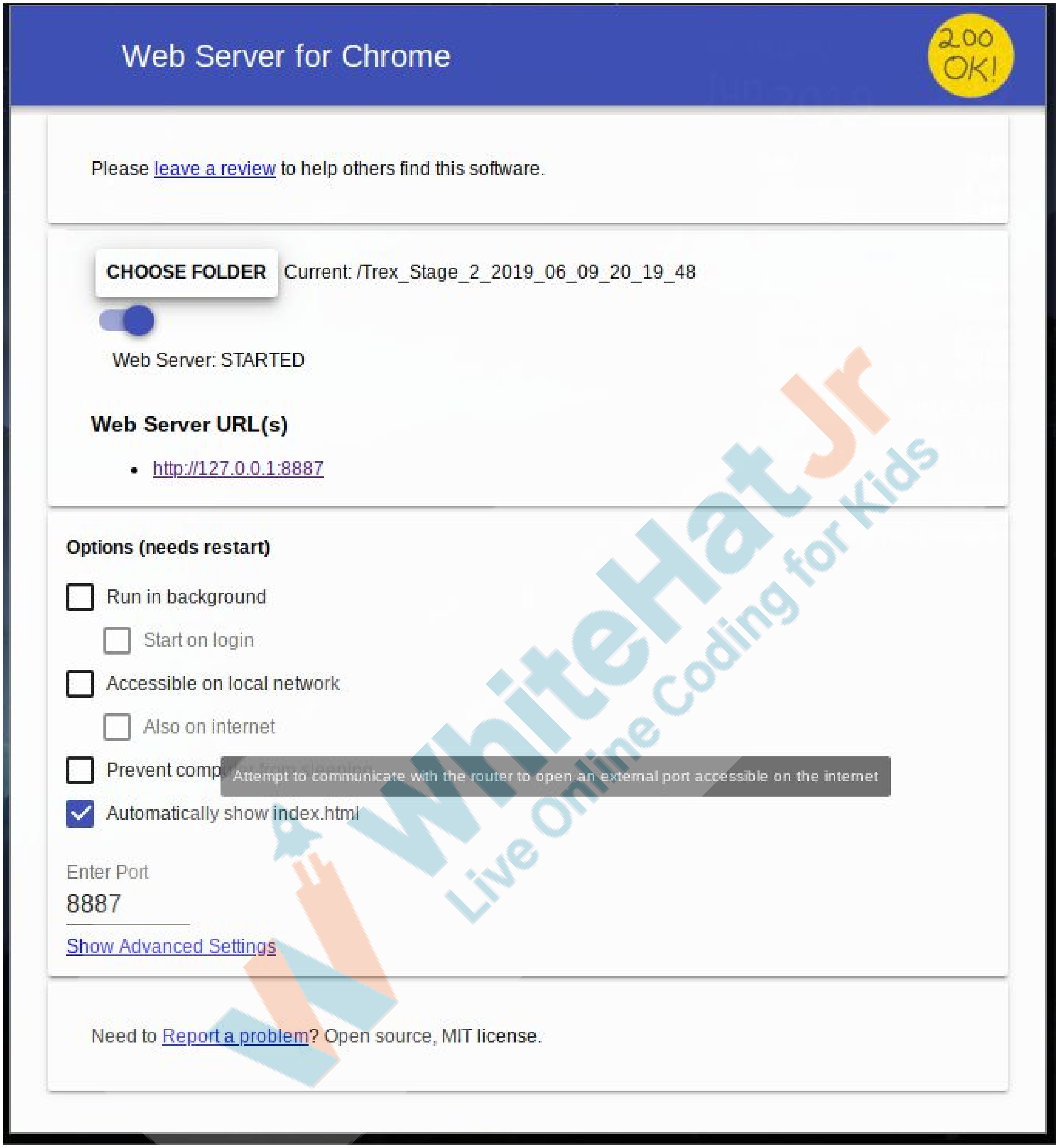
	Great! Quickly let's get the score on the board. Guide the student to get a score on the screen. They need to use getFrameRate() rather than framerate.	ESR: Student writes code to get the score on the top.



```
  ground.x = ground.width /2;
  ground.velocityX = -4;

  invisibleGround = createSprite(200,190,400,10);
  invisibleGround.visible = false;

  cloudsGroup = new Group();
  obstaclesGroup = new Group();
}

function draw() {
  background(180);

  score = score + Math.round(getFrameRate()/60);
  text("Score: "+ score, 500,50);

  if(keyDown("space")) {
    trex.velocityY = -10;
  }

  trex.velocityY = trex.velocityY + 0.8

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

	Awesome, next class we will use gameState and re-write the code to end the game when the trex touches our obstacle. But before we end the class, let's quickly set up a web server and run our code inside the browser. Let's download the files for this project. Go to File > Download	Student downloads the file for the project.
	Unzip the file into a folder. You can try to open index.html with your web browser. What happens?	Student unzips the file. Student runs index.html file with Chrome web browser and the project fails to load.

	Chrome browser blocks access to local files for safety purpose, so that online sites cannot access files on your local system. Therefore the images fail to reload. We need to start our own server to host our game. Install the chrome plugin for web server using the Student Activity Link and launch the app.	Student clicks on Student Activity 3 and installs the chrome plugin. Student chooses the folder.



Web Server for Chrome

Please leave a review to help others find this software.

CHOOSE FOLDER Current: /Trex_Stage_2_2019_06_09_20_19_48

Web Server: STARTED

Web Server URL(s)

- http://127.0.0.1:8887

Options (needs restart)

☐ Run in background

☐ Start on login

☐ Accessible on local network

☐ Also on internet

☐ Prevent computer from sleeping

Attempt to communicate with the router to open an external port accessible on the internet

☑ Automatically show index.html

Enter Port

8887

Show Advanced Settings

Need to Report a problem? Open source, MIT license.

Teacher Guides Student to Stop Screen Share

FEEDBACK

- **Encourage the student to make reflection notes in the markdown format.**
- **Complement the student for her/his effort in the class.**
- **Review the content of the lesson.**

Step 4: **Wrap-Up** **(5 mins)**	Let's quickly review what we learned in today's class	ESR: - We learned how to start a small local web server and how to host an application on it. - We learned how to use the switch statement. - We also re-wrote parts of trex code to run on p5 editor.
	What was the most exciting part?	ESR: varied. for example Starting the web server and running the game there.
	You get Hats Off for your excellent work! Now that we can start a web server and host our game on it, it doesn't seem to be too far fetched to put up our game online so that your friends can play the game. Next class, we will finish up rewriting the trex game and learn how we can host it online so that our friends and family can play the game. Stay hooked into this till then.	Make sure you have given at least 2 Hats Off during the class for: Creatively Solved Activities +10 Great Question +10 Strong Concentration +10
	Congratulations! You have set a new benchmark.	

	Are you ready to take up the challenge? You have to apply the programming constructs learnt during the past few classes to create the Monkey Go Happy app.	
Project Overview	MONKEY GO HAPPY - 2 **Goal of the Project:** Today you have learned how to use switch case to assign different actions based on different conditions. You also learned how to start a small local web server and host files locally to see the game. In this project you have to apply and practice what you have learned in the class and complete building the Monkey Game. ** This is a continuation of Project 16, so make sure to complete that project before doing Project 18. ** **Story:** A monkey has escaped from the zoo and is very hungry. Help the monkey collect Bananas by jumping over obstacles. You have already created the basic game in project 16. Now you have to make the game interesting by adding a scoreboard, images and animations to the game. To make it even more exciting, increase the size of the monkey after eating bananas and decrease its size if it falls or hits an obstacle.	Student engages engages with the teacher over the project.

	I am very excited to see your project solution and I know you will do really well. Bye Bye!	
	Teacher Clicks	
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? ● What aspects of the class helped me? What did I find difficult?	Student uses the markdown editor to write her/his reflection as a reflection journal.

Activity	**Activity Name**	**Links**
Teacher Activity 1	Trex stage 2 (P5)	https://editor.p5js.org/whitehatjr/sketches/fjtc9IuTN
Teacher Activity 2	Trex Stage 2 Final Code (Reference)	https://editor.p5js.org/whitehatjr/sketches/bjoP1EWLA
Student Activity 1	Trex Stage 2(P5)	https://editor.p5js.org/whitehatjr/sket

		ches/SzQ1rBSjj
Student Activity 2	Trex Stage 4 (code.org	https://studio.code.org/projects/gamelab/o9GGsCWsUmEgv84DffVcnt2EFakvA4ubIYfcRQd0Goo/edit
Student Activity 3	Web server plugin	https://chrome.google.com/webstore/detail/web-server-for-chrome/ofhbbkphhbklhfoeikjpcbhemlocgigb/related