



Topic	SCOPE OF VARIABLES	
Class Description	The student learns about the global and local scope of variables. The student writes a reset function for the game to reset the game from within the game itself.	
Class	C17	
Class time	50 mins	
Goal	<ul style="list-style-type: none"> • Change the scope of some variables from local to global to be used anywhere in the code. • Write a reset function to restart the game when the reset icon is pressed. • Set up a local environment to run the Trex code on the local machine. 	
Resources Required	<ul style="list-style-type: none"> • Teacher Resources: <ul style="list-style-type: none"> ○ VS Code Editor ○ Laptop with internet connectivity ○ Earphones with mic ○ Notebook and pen • Student Resources: <ul style="list-style-type: none"> ○ 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 - 10mins		

<div>  </div> <p>Teacher starts slideshow from slides 1 to 9</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? Are you excited about today's class?</i></p> <p>Run the presentation from slide 1 to slide 3.</p> <p>Following are the warm up session deliverables:</p> <ul style="list-style-type: none"> Connecting the student to the previous class. Explain through examples the concept of local and global variables. Quizzes 	<p>ESR: Hi, thanks. Yes, I am excited about it.</p> <p>Click on the slide show tab and present the slides.</p>
<p>WARM-UP QUIZ</p> <p>Click on In-Class Quiz</p>	
<p>Continue the warm up session</p>	
Activity details	Solution/Guidelines
<p>Run the presentation from slide 4 to slide 9 to set the problem statement.</p> <p>Following is the warm up session deliverables:</p> <ul style="list-style-type: none"> To Bounce a box using the OOP concept. 	<p>Narrate the story by using hand gestures and voice modulation methods to bring in more interest in students.</p>
<div>  </div> <p>Teacher ends slideshow</p>	
<p>TEACHER-LED ACTIVITY 1 - 10mins</p>	
<p>Teacher Initiates Screen Share</p>	
Teacher Action	Student Action
<p>Step:1 Teacher-led Activity 1</p>	<p><i>The teacher downloads the</i></p>

In the last class, we learned about object-oriented programming and created a **box** class along with functions to show the change of speed and change of width.

But there was one limitation, the dimensions and coordinates for the box were specified while creating the class.

What if we wanted to create a box of our choice of coordinates. We could not create it.

But in today's activity, we are going to learn more about **constructor arguments**, just like we did in the last class. The **constructor()** function executes when we create an object. This will allow us to create a box of our choice of coordinates.

***Note:** This activity is the continuation of the previous class's OOP activity, where we have the **box.js** file created.*

Constructor() arguments allow us to set the properties of the object such as setting the x, y position or setting a particular speed to the object, etc.

Like in the last class, we already have a file **box.js** which is added in the **index.html** file using the **<script>** tag.

First, we are going to create the class and its functions. The change here will be how we create the **constructor()** function.

If you remember, in the last class we kept the brackets of the **constructor()** function empty, but this time we are going to pass the parameters here which will allow you to give custom parameters.

Here we are setting the parameters like **x, y, w, h, and vx**, which stands for x-position, y-position, width, height, and x speed of the box.

code for [teacher activity 1](#) and runs in the VS Code Editor.

The teacher writes the code.

```
1  class Box
2  {
3      constructor(x,y,w,h,vx)
4      {
5          this.x =x;
6          this.y =y;
7          this.w =w;
8          this.h = h;
9          this.vx = vx;
10     }
11
12 }
```

Unlike the last time, where we gave the values for the properties such as **this.x = 100**.

Here we are setting them equal to the value passed by you.

So when you create the object, you are expected to provide values as we did with the **createSprite()** function.

Once done with the **constructor**, we will create the function to show and move the box like we did the last time.

In the **show()** function, we are going to create a rectangle using the **rect()** function which will take parameters like **this.x, this.y, this.w**, and **this.h**.

This is going to create a rectangle at our specified position and dimensions.

The next step is to create the **move()** function. In this function we are going to change the **x-position** with the **vx** which is the velocity in the x-direction, and this too will be

passed by us at the time of creating the object.

The teacher writes the code.

```

1  class Box
2  {
3      constructor(x,y,w,h,vx)
4      {
5          this.x =x;
6          this.y =y;
7          this.w =w;
8          this.h = h;
9          this.vx = vx;
10     }
11
12     show()
13     {
14         rect(this.x,this.y,this.w,this.h)
15     }
16
17     move()
18     {
19         this.x = this.x+this.vx;
20     }
21
22 }
```

This function will enable us to move our box from left to right on the canvas when we call this function.

Our class and class functions are now ready; let's now create objects.

To create an object, first, we make a variable as **var box**.

In the **setup()** function we create the object as **box1 = new**

Box(100,100,50,50,2).

The first two parameters are the x and y positions, the next two are width and height, and the last parameter is the x-direction speed.

But we won't be able to see any box on the screen.
Can you tell why?

Very good!

Now, in the **draw()** function, we call the class functions using objects as:

box.show()
box.move()

ESR:



We haven't called the function to display it.

```
var box;

function setup()
{
  createCanvas(400, 400);
  box = new Box(100,200,20,20,2);
}

function draw()
{
  background(220);
  box.show();
  box.move();
}
```

Output:

	
When we run the code, we can see the box moving in the x (horizontal) direction.	
Teacher Stops screen share	
Now it's your turn. Please share your screen with me.	
<div>  Teacher starts slideshow from slides 10 to 12 Refer to speaker notes and follow the instructions on each slide. </div>	
We have one more challenge for you. Can you solve it? Let's try. I will guide you through it.	ESR: Yes!
STUDENT-LED ACTIVITY - 5mins	
<ul style="list-style-type: none"> • 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. 	
<p style="text-align: center;"><u>ACTIVITY</u></p> <ul style="list-style-type: none"> • Create a class function to move the box in vertical direction. • Create a box object and call the functions. 	
Teacher Action	Student Action

In the last activity we have seen how to create a box class, object with **constructor()** arguments, and create a move function to move in the horizontal direction.

In this activity we are going to do a few additions. First we will add y speed in our **constructor()**, so that we can move our box in y direction as well.

*The teacher will guide the student to add vy in the **constructor()** argument*

We already have the box class we created in the previous activity. Here you will add the y speed in the **constructor()** argument and then write a function to move the box object in the y direction as well.

In the **constructor()** function add one more parameter in the **constructor()** argument. This will be vy, in which is y the speed.

The student downloads the [Student activity 1](#) code and opens it in VS code.

```

1  class Box
2  {
3      constructor(x,y,w,h,vx,vy)
4      {
5          this.x =x;
6          this.y =y;
7          this.w =w;
8          this.h = h;
9          this.vx = vx;
10         this.vy = vy;
11     }

```

We have the show function to display the box and we also have the move function which can move the box in the horizontal direction when called.

Now let's create one more function and call it **moveup()**
 In this function we are going to change the y position of our box by subtracting the vy value from it.


```
moveup()  
{  
  this.y = this.y - this.vy;  
}
```

You have created the moveup() function. Now go to the sketch.js.

we will create the object for the box and call the functions for it.

The benefit of object oriented programming is that we can define the function and call them whenever we want.

To create the object, first we make a variable as **var box**. In the setup function we create the object as `box1 = new Box(100,100,50,50,2,1)`.

The first 2 parameters are x and y position, next two are width and height and the last two parameters are x speed and y speed.

This will create an object of the box, now in the **draw()** We are going to call the functions to show the box and move it.

If you remember we had two functions to move the box, but we are not going to call both of them, the reason is if we call both the functions then our box will move in the diagonal direction but we want our box to move only in the upward direction. Hence we call the **show()** and **moveup()** function.

Run the code to see the output.

```
var box;


function setup()
{
  createCanvas(400, 400);
  box = new Box(100,200,20,20,2,1);
}

function draw()
{
  background(220);
  box.show();
  box.moveup();
}
```

output:



We have created the box class and created a **constructor()** with arguments.
We also created and called the function to move the box.
Now let's move on to the T-rex game.

Student stops sharing the screen	
TEACHER-LED ACTIVITY 2- 10mins	
<div>  </div> <p>Teacher starts slideshow from slides 13 to 20 Refer to speaker notes and follow the instructions on each slide.</p>	
Teacher Initiates Screen Share	
<p><u>CHALLENGE</u></p> <ul style="list-style-type: none"> Change the scope of the variable to allow it to be accessed anywhere in the code. 	
Activity details	Solution/Guidelines
<p>Step 2: Teacher-led Activity Before we start with today's class, we will learn about scope in programming.</p> <p>The teacher opens Teacher Activity 2</p> <p>Have you ever wondered why we declare all the variables at the top of our program, rather than inside function setup() or function draw()?</p>	<p><i>The student can come up with his/her responses.</i></p>

```
var PLAY = 1;
var END = 0;
var gameState = PLAY;

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

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

var score;
var gameOverImg, restartImg
var jumpSound , checkPointSound, dieSound
```

Let's understand why.

Let's try to declare a variable called message inside function **setup()** which stores a message.

Let's print it inside the function **setup()** itself.

The student helps the teacher in writing this code.

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

  var message = "This is a message";
  console.log(message)

  trex = createSprite(50,160,20,50);
  trex.addAnimation("running", trex_running);
  trex.addAnimation("collided", trex_collided);

  trex.scale = 0.5;

  ground = createSprite(200,180,400,20);
```

Is the message printed on the console?

ESR:

Yes!

How many times?

ESR:

Once.

That's because - setup() is called only once.

What do you think will happen if we print the message inside **draw()**.

ESR:

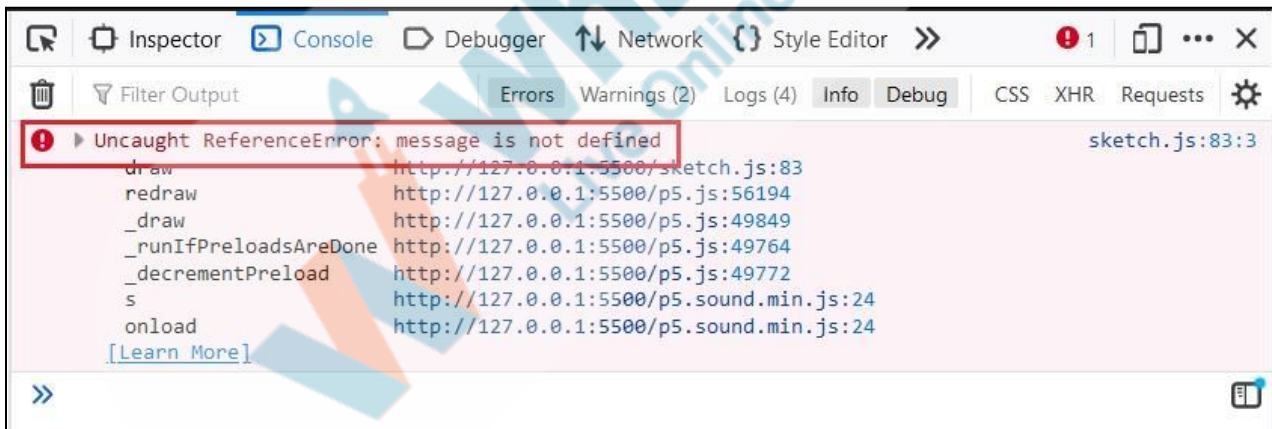
The message will be printed many times!

Let's see if that is what happens.

```

79
80 function draw() {
81
82     background(180);
83     console.log(message);
84     //displaying score
85     text("Score: " + score, 500,50);
86
87
88     if(gameState === PLAY){
89         //move the
90         gameOver.visible = false;
91         restart.visible = false;
92
93         ground.velocityX = -(4 + 3* score/100)
94         //scoring
  
```

OUTPUT:



What do you see?

But we have defined “message”, haven’t we? We just printed it on the console.

It throws an error,
“message is not defined”.

*Allow the student to think
about this and why it might
be happening.*

<p>Variables in the program have a scope. They live and die within the scope.</p> <p>A variable declared inside a function or any block of code starting and ending with { } (for example - for loop) have scope defined only inside the function (or curly brackets).</p> <p>It cannot be accessed by any other function or any other block of code directly. Such variables are said to have local scope.</p> <p>Variables declared outside the functions at the top are said to have a global scope. This means you can access them anywhere in the code.</p>	<p><i>The student asks questions about scoping of variables.</i></p>
<p>Let us get started with the task for today.</p> <p>The teacher opens Teacher Activity 2 and plays the game from the last class.</p> <p>Look at the game we designed in our last class. We see the Game Over text and restart icon when the game ends.</p> <p>Currently, however, the restart icon doesn't work. We have to manually stop the code and play it again for us to re-play the game.</p> <p>The challenge for us is to simply reset the game by pressing the reset icon inside the game. Sounds simple, right?</p> <p>Let's see what challenges we will face in doing this.</p>	<p><i>The student observes and tries to understand the goal of the lesson.</i></p> <p>ESR: Varied.</p>



What we need is to tell the computer that if someone clicks on the reset icon in the game, it has to reset the game to start from the beginning.

How can we say that in the code?

ESR: Varied.

Let me try to write this down.

We must find an instruction which detects the mouse pressed over a sprite (since the reset icon is a sprite).

Let us look into the World tab in our toolbox.

The student detects the `mousePressedOver(sprite)` instruction.

Let us tell the computer that if the player presses the mouse over the reset sprite, it should reset the game.

The teacher writes the code for this.

There is an instruction in the p5.play library called **mousePressedOver()** which detects a mouse pressed over any sprite.

It accepts the sprite as an argument and returns true if the mouse is pressed over the sprite and false otherwise.

The student observes and asks questions.

Let us try to use `mousePressedOver()` and display something on our screen when the mouse is pressed over the restart sprite.

```

obstaclesGroup.setVelocityXEach(0);
cloudsGroup.setVelocityXEach(0);
}

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

if(mousePressedOver(restart)) {
  console.log("Restart the Game")
}

drawSprites();
}
  
```

output:

3 Restart the game sketch.js:117

Let's run the code and check the output.

The teacher presses the reset icon when the game is over.

The student observes the "Restart Game" message printed on the console.

Instead of printing something on the console, we can also call a `reset()` function which resets everything in the game to its original state.

For now we can write an empty `reset()` function which we will call in our code.

Student observes

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

if(mousePressedOver(restart)) {
  console.log("Restart the Game")
}

drawSprites();
}

function reset(){
}
```

Now, we need to write the code inside the reset function which will reset the game back to its original state.

Can you do that on your own?

ESR: Yes I can try.

Teacher starts slideshow



:Slide 21-23

Run the presentation for slide 21-23 to set the student activity context.

Great.

Can you tell me what should we do in the reset function?

The student may have additional thoughts. Listen and guide him/her accordingly.

ESR:

- We would like to start the game again. That is, we would like to go back to PLAY state.
- gameOver and reset sprites should be invisible again.

	<ul style="list-style-type: none"> Score should be reset to 0
<p>Great!</p> <p>Why don't you share your screen and show me how are you going to code this challenge?</p>	<p><i>The student shares his/her screen, opens the Student Activity, and adds more code to it.</i></p>
Teacher Stops Screen Share	
STUDENT-LED ACTIVITY 2- 10mins	
<ul style="list-style-type: none"> 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. 	
<p align="center"><u>ACTIVITY</u></p> <ul style="list-style-type: none"> Write a reset function to reset the game when the reset icon is pressed. 	
<p>Step 3:</p> <p>Student-Led Activity</p> <p>What are the different things that we need to reset the game?</p>	<p>ESR:</p> <p>Change the gameSate back to PLAY. Make the gameOver and restart sprites invisible again.</p>
<p>Let us do these and see what happens.</p> <p><i>Guide the student to change the gameState to PLAY and make the gameOver icons invisible.</i></p>	<p><i>The student opens Student Activity 2.</i></p> <p><i>The student writes code to make the changes and then runs to see what happens.</i></p> <p><i>The student presses the restart icon inside the game to see the output.</i></p>

```
function reset(){
  gameState = PLAY;
  gameOver.visible = false;
  restart.visible = false;
}

function spawnObstacles(){
  if (frameCount % 60 === 0){
    var obstacle = createSprite(600,165,10,40);
    obstacle.velocityX = -(6 + 3*score/100);
  }
}
```

What is happening?

ESR:

The ground scrolls for a while and again gameState changes back to END.

Why do you think this is happening?

ESR: Varied.

Remember, in the last class - we had changed the lifeTime of the obstacles so that they never disappear. As long as the obstacles are there, the T rex is going to collide with the obstacle immediately and the gameState will change back to END.

Let us destroy all the obstacles and clouds when we reset the game.

Guide the student to write the code to destroy all the obstacles and clouds in the game.

Note: Review group sprites and how their properties work with the student.

The student writes code to destroy the obstacles and clouds using the group properties.

The student runs the code to see the output.

```
function reset(){  
  gameState = PLAY;  
  gameOver.visible = false;  
  restart.visible = false;  
  
  obstaclesGroup.destroyEach();  
  cloudsGroup.destroyEach();  
}
```

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

What's wrong now?

ESR:


The Trex animation is still set to 'collided'.

So let's change it to running animation at reset.

The student writes code to change the Trex animation.

The student runs the code to see the output.

```
function reset(){  
  gameState = PLAY;  
  gameOver.visible = false;  
  restart.visible = false;  
  
  obstaclesGroup.destroyEach();  
  cloudsGroup.destroyEach();  
  
  trex.changeAnimation("running", trex_running);  
}
```

<p>Can you notice that even though the “restart icon” is invisible during PLAY state in the game, you can still press in the area to restart the game anytime?</p> <p>What can we do to fix this?</p>	<p><i>The student tries to press on the invisible “restart” icon in the game.</i></p> <p>ESR: We can move the condition where we check the mouse pressed over the reset icon inside Game End state only.</p>
<p>Let's do this and see if that works.</p>	<p><i>The student writes the code, runs and tests it.</i></p>
 <pre> else if (gameState === END) { gameOver.visible = true; restart.visible = true; if(mousePressedOver(restart)) { reset(); } ground.velocityX = 0; trex.velocityY = 0 //change the trex animation trex.changeAnimation("collided", trex_c </pre>	
<p>Everything seems to be working well. But there is still an issue. Can you notice it?</p>	<p>ESR: The score does not get reset.</p>
<p>What do you think we can do to reset the score in the game?</p> <p>Let's do it and see what happens.</p>	<p>ESR: Set score = 0 in the reset.</p> <p><i>The student writes code and checks the output.</i></p>


```

    drawSprites();
  }

  function reset(){
    gameState = PLAY;
    gameOver.visible = false;
    restart.visible = false;
    trex.changeAnimation("running", trex_running);

    obstaclesGroup.destroyEach();
    cloudsGroup.destroyEach();
    score = 0;
  }

```

What happened?

ESR:

The score temporarily resets to 0 but again restarts from its old value.

Can you guess why this is happening?

ESR: Varied.

In our game, our count value depends on the frameCount. The frameCount goes on increasing even when the game is in END state.

Can we use something else to update the count so that it resets and starts from 0 when the game is reset?

ESR: Varied.

Let us try to update count using frameRate. frameRate is the number of frames (images our game shows every second). frameRate is nearly constant throughout the game and nearly equal to 60.

The student makes the change in the code and runs to see the output.

Note: It might be different on different systems. For some it might be 30.

We could write: `score = score + Math.round(frameRate/60).`

When we divide the `frameRate` with 60, we may get the result in the form of decimal numbers, but we want the score to be updated only with integers(1,2,3,4 etc).

In order to do that we can use the **Math.round()** function. This function rounds off the value to the nearest integer.

For example if we have the number as 9.2, the round function will make it as 9 and give us an integer.

This will increase the count by 1 every frame.

When the game resets, the count becomes 0 and then the count starts from 0 again.

Let us change the code and see if it works this time.


```

if(gameState === PLAY){
  //move the
  gameOver.visible = false;
  restart.visible = false;

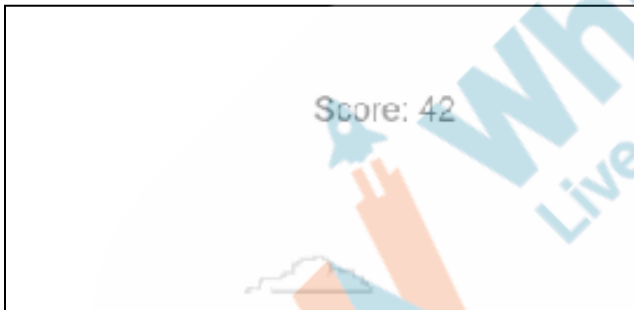
  ground.velocityX = -(4 + 3* score/100)
  //scoring
  score = score + Math.round(getFrameRate()/60);

  if(score>0 && score%100 === 0){
    | checkPointSound.play()
  }

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

```

output:







All working now. Great job! We did it.
 There still might be some bugs in the
 game. You can find and work on it as
 your Bug Hunter Bounty Project.

For now, take pride in your work.

Teacher Guides Student to Stop Screen Share

WRAP UP SESSION - 5 Mins

<p>Step 4: Wrap-Up (5 min)</p> <p>Let us quickly review what we learned today.</p> <p>What did we learn about scope today?</p>	<p>ESR:</p> <p>We learned about the scope of variables - global scope and local scope.</p> <ul style="list-style-type: none"> - Global scope variables can be used anywhere within the code. - Local scope variables can be used within the block of code between { } in which they are created.
<p>You get Hats Off for your excellent work!</p>	<p><i>Make sure you have given at least 2 Hats Off during the class for:</i></p> <div data-bbox="1019 829 1286 924">  +10 Creatively Solved Activities </div> <div data-bbox="1019 940 1286 1035">  +10 Great Question </div> <div data-bbox="1019 1052 1286 1146">  +10 Strong Concentration </div>
<p>Teacher starts slideshow  from slide 24 to slide 33</p>	
Activity details	Solution/Guidelines
<p>Run the presentation from slide 24 to slide 33</p> <p>Following are the warm up session deliverables:</p> <ul style="list-style-type: none"> ● Explain the facts and trivias ● Next class challenge ● Project for the day ● Additional Activity 	<p>Guide the student to develop the project and share with us.</p>
<p>WRAP-UP QUIZ Click on In-Class Quiz</p>	



Teacher ends slideshow

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.

In the next class, we will learn how to write code on our local machine!

We are going to have more fun in our next class when we learn more about how to write code on our local machine. Later, we will also modify our Trex code slightly so that our game can be run on a mobile phone.

Looking forward to our next class.

*** This Project will take only 30 mins to complete. Motivate students to try and finish it immediately after the class.**

Project Overview

Cycle Race

Goal of the Project:

In Class 17, you learned about the global and local scope of variables. Students will write a reset function for the game to reset the game from within the game itself.


In this project, you have to practice and apply what you have learned in the class and create different opponent cyclists. Also add a distance calculating functionality and Add cycle bell sound.

Story:

Rocky loves cycles and cycling computer games created by himself. But it allows him to play only once. He wants to

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.

The students engage with the teacher over the project.

<p>add functionality to reset the game. He has asked for your help to add reset functionality.</p> <p>Can you create a design for his game?</p> <p>I am very excited to see your project solution and I know you will do really well.</p> <p>Bye Bye!</p>	
<div> <div>Teacher ends slideshow</div>  </div>	
<div> <div>Teacher Clicks</div> <div>✕ End Class</div> </div>	
<p>Additional Activities</p> <p><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 her/his reflections in the reflection journal.</i></p>

Activity	Activity Name	Links
Teacher Activity 1	Template Code	https://github.com/pro-whitehatjr/Pro-c17-ta-1-template

Teacher Reference	Reference code for Teacher activity 1	https://github.com/pro-whitehatjr/pro-c17-ta1-reference
Teacher Activity 2	Trex Stage 7	https://github.com/pro-whitehatjr/PRO_C17_LP_TA1
Teacher Activity 3	Reference Code	https://github.com/pro-whitehatjr/PRO_C17_LP_TA2
Teacher Reference	Reference code for Student Activity 1	https://github.com/pro-whitehatjr/pro-c17-sa1-reference
Student Activity 1	Template Code	https://github.com/pro-whitehatjr/pro-c17-sa-1-template
Student Activity 2	Trex Stage 7.5	https://github.com/pro-whitehatjr/PRO_C17_LP_SA1
Visual Aid	Visual Aid	https://curriculum.whitehatjr.com/Visual+Project+Asset/PRO_VD/BJFC-PRO-V3-C17-withcues.html
In-class Quiz	In-class Quiz	https://s3-whjr-curriculum-uploads.whjr.online/c32a1e8a-7ca5-4902-961e-9b8777b0fd4b.pdf
Project Solution	Cycle Race	https://github.com/pro-whitehatjr/Project_C17_Cycle_Race