Girls Who Code At Home

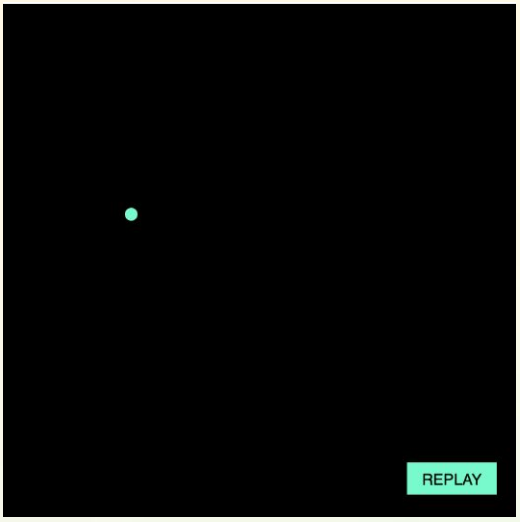
Meteor Catcher Game: Part 3

Make the Meteor Fall

Activity Overview

At the end of Part 2, you used the coordinate system to draw the first component of your game - the meteor! Then you set the color of your meteor and sketch background. In this part, you will learn how to create and use variables in p5.js to move the meteor across the screen at a specified speed. We will combine variables and arithmetic operators like **+** and **=** to simulate motion. Yes, it is magical, but really it's just simple math! Click here to preview what you will learn by the end of the activity.

You should have already completed Part 1 and Part 2 of the **Meteor Catcher Game Series** before embarking on this activity.



Note: *We included a replay button so you can reset the meteor's behavior. If we did not include this, you would see only a black box once the meteor fell off the bottom of the screen. We will fix this in the next activity,* ***Part 4,*** *with a conditional.*

Learning Goals

By the end of this activity you will be able to...

- ❏ describe how to simulate basic motion in a program.
- ❏ program different behaviors into elements using variables and arithmetic operators.

Materials

- ➔ p5.js Online Editor
- ➔ Meteor Catcher Game Sample Project
- ➔ Meteor Catcher Game Part 3 Reference Guide

PART 1	PART 2	PART 3	PART 4	PART 5
Planning + Intro to p5.js	Draw the Meteor	Make the Meteor Fall	Catch the Meteor	Add a Randomizer
Identify the parts of a game and learn about the basics of program flow in p5.js	Build a basic shape to represent your meteor, update the style, location, size, and color of your meteor.	Simulate motion for your meteor using variables	Build a catcher that follows your mouse to catch the falling meteors.	Add randomness to make your game more challenging and fun!

You can also follow along with Part 3 in the Meteor Catcher Game video tutorials!

Women in Tech Spotlight: Robin Hunicke

Image Source: UCSC DAMN

Robin is a video game designer that teaches at UC Santa Cruz and is the co-founder of Funomena. Robin began her career at Electronic Arts as a Lead Designer, where she designed MySims. She then worked with thatgamecompany, an independent game company, where she was one of two women on the team to produce the game **Journey**. Journey won several Game of the Year awards and was even nominated for the 2013 Grammy Awards for Best Score Soundtrack for Visual Media.

With her team at **Funomena**, Robin began to create video games utilizing all different platforms including virtual reality goggles. Her team has made experimental games, including Luna and Woorld. Despite the small margins on the production of Virtual Reality (VR) games, Robin believes it is important to develop games that take risk and pushes her creativity. In 2008, Robin was named Gamasutra's Top 20 Women Working in the Video Game Industry and in 2009 she was awarded Microsoft's Gaming Award for Design.

Robin is a large advocate for diversity in the gaming industry. Her work mainly revolves in amplifying the work and voices of underrepresented groups. Much of her work as a professor at UC Santa Cruz offers students a program that combines both art and programming courses for game design.

Watch this **video** to learn more about Robin and how she works to be a positive force in the game industry. Learn more about Robin by reading her **short faculty bio** and reading about her AR game **Woorld**, exploring the game **Journey**, or her other projects.

Reflect

Being a computer scientist is more than just being great at coding. Take some time to reflect on how Robin and her work relates to the strengths that great computer scientists focus on building - bravery, resilience, creativity, and purpose.

CREATIVITY

How does Robin approach games in a different way than expected? What are the advantages of approaching a project in an unexpected way?

Share your responses with a family member or friend. Encourage others to read more about Robin to join in the discussion!

Step 1: Using variables in p5.js (5-10 mins)

Review variables in JavaScript (3-5 mins)

Before we dive in, let's do a quick review on variables. Variables are containers that are used to store information (data) in a computer program. They are particularly powerful because we can easily change the value of a variable over the course of our program.

To create a variable, we need to declare it first. This tells the program that we want to create a container and name it. In JavaScript, we declare a variable like this: `let meteorDiameter;`. We can declare it and initialize it (or assign it a value) at the same time like this: `let meteorDiameter = 50;`. Let's break down the syntax:

JAVASCRIPT	DESCRIPTION
`let meteorDiameter = 50;`	➔ `let`: Keyword that tells the sketch to create a variable. ➔ `meteorDiameter`: The name of our variable. This name can be anything you like, but be sure to make it descriptive! It can only be one word, so we use camelCase. ➔ `=`: Assigns the value to the variable. This is also called initializing. ➔ `50`: The value currently stored in the variable. You can store any type of data in a variable: numbers, letters, strings, etc. ➔ `;`: All lines of code in p5.js must end with a semicolon.

Programmers typically create and define all of the variables that they will need at the top. These are called global variables. This means you can use those variables anywhere in your code. It also makes your code more readable for both the programmer and anyone else reviewing their code. If you need more of a refresher on variables, check out this video from the Coding Train.

Add variables (3-5 mins)

Right now, our ellipse does not contain any variables. If we want to simulate motion, we have to swap out these static values for variables so we can change the value of the x and y position over time. We need to declare and initialize variables for each parameter in our ellipse: x, y, and the width and height.

Naming your variables. You can use the variable names we use or create your own. If you use your own, remember to reference them correctly later on

Add the following variables above the `setup()` function:

- ❏ **Create a variable to store the x position and assign it a value of 200.** We named this variable `meteorX`, but you can create your own variable name.
- ❏ **Create a variable to store the y position and assign it a value of 0.** We named this variable `meteorY`, but you can create your own variable name.
- ❏ **Create a variable to store the width and height and assign it a value of 20.** These values will be the same since it is a circle. We named this variable `meteorDiameter`, but you can create your own variable name.

Now that we have the variables, let's use them! In the `ellipse()` function, replace the numerical values with the corresponding variable we just created for the following parameters:

- ❏ x position
- ❏ y position
- ❏ width and height

Don't forget to check your code with the Reference Guide on pg 2.

Step 2: Make observations about motion (5-10 mins)

Our goal in this part is to make the meteor fall from the top of the screen to the bottom of the screen. But how do we translate that into code? Let's consider an example to help us figure it out.

Examine this sketch below of a circle moving from left to right. Take 60 to 90 seconds to make observations about the behavior of the circle. Think about the following questions:

- ➔ What axis is the circle moving on?
- ➔ How is the position of the circle changing? What value in `ellipse()` would have to change to make this happen?
- ➔ Do you think the code to make this happen is in `setup()` or `draw()`?

Use your observations to write a line of pseudocode to tell the program how to move the ball. Do not move onto the next part until you are finished.

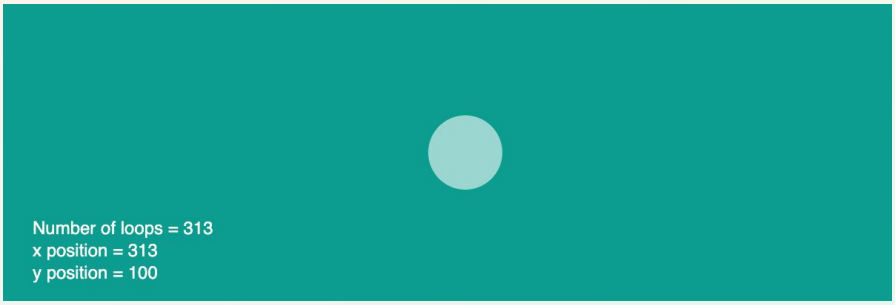
Don't forget to check your ideas with the Reference Guide on pg 2.

Writing Motion as Code

In order to mimic horizontal motion, we want the value of x to change each time the program loops. Remember - the program works on a loop. The program runs every line of code in `draw()` in sequence. Once it reaches the end of the program, it goes back to the top and starts all over again. It does this forever until you tell it to stop. **We can write motion as a line of code by setting the x value equal to itself plus a number:**

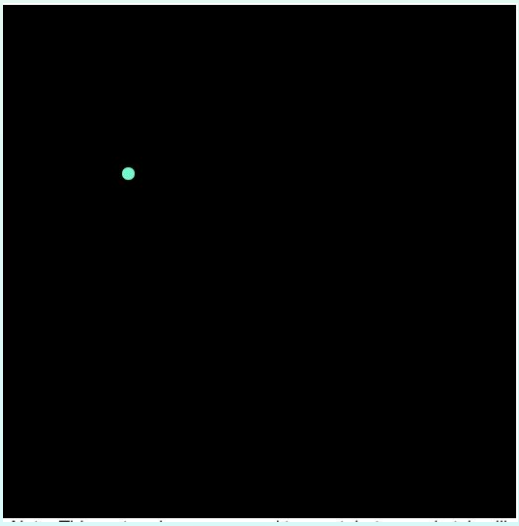
```
ellipse(xPosition, yPosition, 50, 50);
xPosition = xPosition + 1; // Can also be written as xPosition++
```

This means that the value of x will increase by that number every time the program loops. This number determines how slowly or quickly the meteor moves across the screen. In other words, it sets the speed. In the **example sketch** below, we set the speed to 1 so the x position increases by 1 with each loop:



Step 3: Add motion to your meteor (5-10 mins)

Our example circle moved horizontally, but we want the meteor to move vertically down the y axis as shown in the sketch below. This means we need to increase the y position instead of the x position. Click this link to preview an **example sketch**.



This meteor is programmed to reset, but your sketch will not do this until Part 5.

Follow these steps to make your meteor fall:

- ❏ **Create a new variable above `setup()` to store the speed.** We named our variable `speed` but you can name it whatever you like. Just be sure you reference it correctly later in your code.
- ❏ **Assign it a value that will make the meteor fall slowly.** *Hint: You can use decimals!*
- ❏ In the `draw()` function, add a line of code that changes the y position of your meteor to make it fall from the top of the screen to the bottom of the screen.

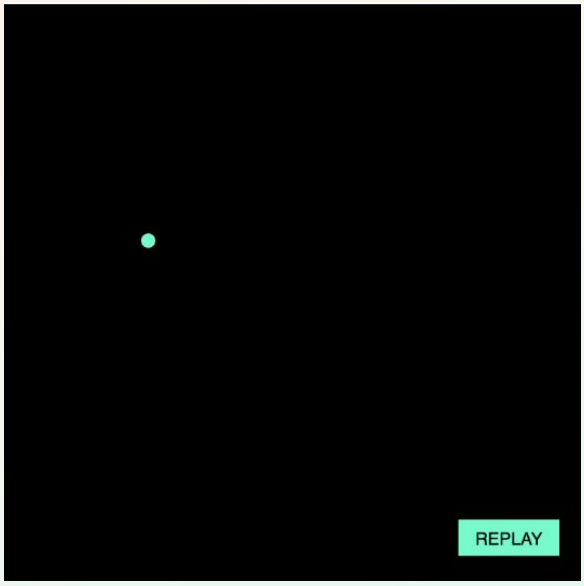
Don't forget to check your code with the Reference Guide on pg 3.

Step 4: Test Your Code (5 mins)

Let's test what we have written so far to make sure our program runs the way we want it to. Click the play button to run your sketch. You should have:

- ➔ A meteor that falls at a slow rate from the top of the screen to the bottom of the screen.
- ➔ It should disappear at the bottom.
- ➔ You should not have a Replay button.

Example



We included a replay button so you can reset the meteor's behavior. If we did not include this, you would see only a black box once the meteor fell off the bottom of the screen. We will fix this in the next activity, ***Part 4****, with a conditional.*

Not working the way you want it to? Try these **debugging tips:**

- ➔ Is your code inside the correct curly brackets?
- ➔ Do you have semicolons at the end of each line of code?
- ➔ Did you spell the variable and function names correctly?
- ➔ Are your functions in the correct location and sequence? Remember that order matters in program flow!
- ➔ Is your arithmetic operator in the correct place?
- ➔ Is the value of your speed variable too fast (high value) or too slow (low value)?
- ➔ Is your meteor falling from the top to the bottom? Did you update the y position variable of your meteor?

If you need a refresher on best practices for debugging, check out **this fantastic post** from the p5.js community.

Step 5: Check for Understanding

How would you change the speed equation to make the meteor move from the bottom of the screen to the top?

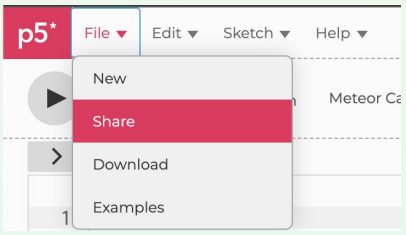
Don't forget to check your ideas with the Reference Guide on pg 3.

Step 6: Share Your Girls Who Code at Home Project! (5 mins)

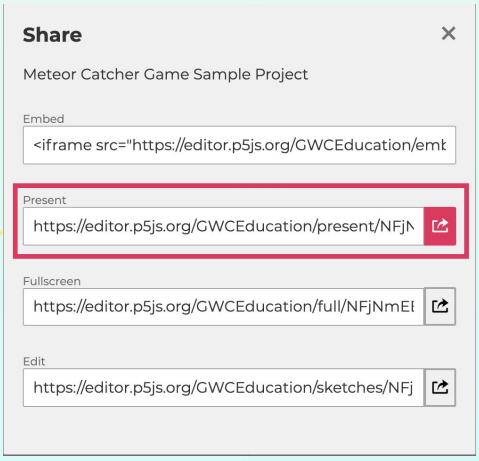
We would love to see your work and we know others would as well. Share your game with us! Don't forget to tag `@girlswhocode #codefromhome` and we might even feature you on our account!

Follow these steps to share your project:

- ➔ Save your project first.
- ➔ In the **File** Menu, choose the **Share** option in the dropdown menu.
- ➔ Choose the **Link** option in the dropdown menu.
- ➔ Copy the **Present** Link paste it wherever you would like to share it.



Project Link



Stay tuned for more Girls Who Code at Home projects!