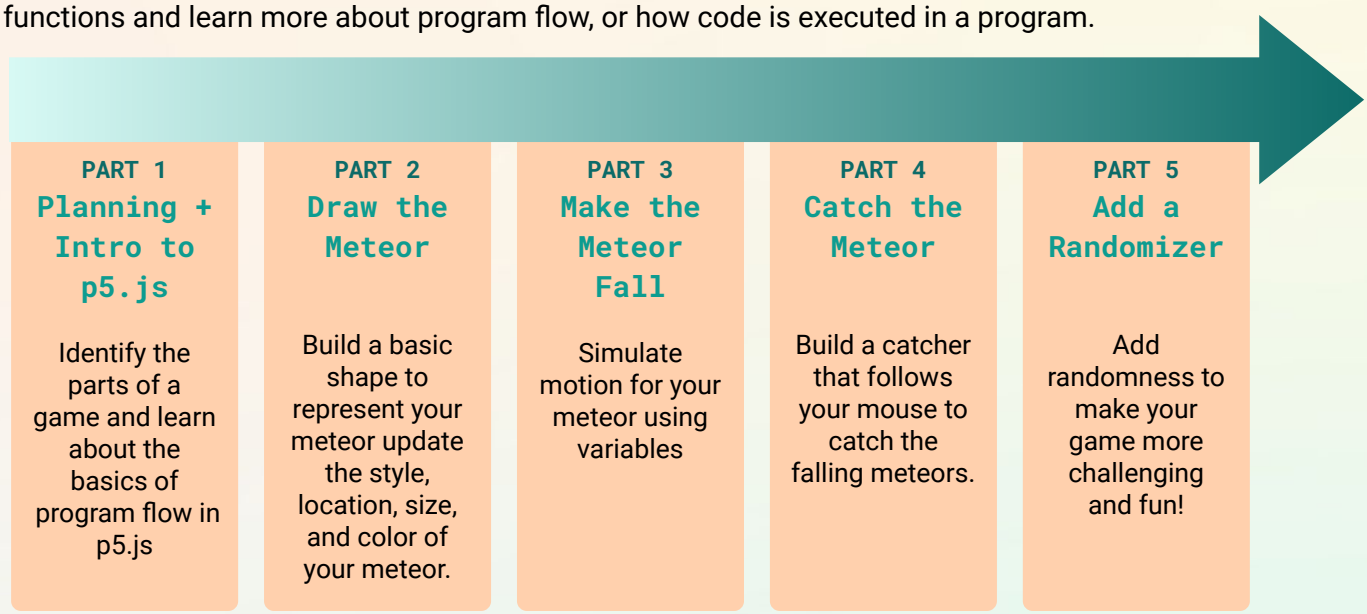
Girls Who Code At Home

Meteor Catcher Game: Part 1

Planning + Intro to p5.js

Activity Overview

In this project, you will learn to program a collecting game using **p5.js**, a JavaScript library created especially for artists and designers. In **Part 1** of this project, you will learn about how different parts of a game work together as a system and explore the basics of p5.js. You will create sketches using functions and learn more about program flow, or how code is executed in a program.



Learning Goals

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

- ❏ identify the parts of a game and explain how they work together as a system in the program
- ❏ describe the inspiration behind p5.js and navigate around the environment.
- ❏ describe the flow of the program using relevant terminology, such as conditionals and control flow.

Materials

- ➔ p5.js Online Editor
- ➔ Meteor Catcher Game Sample Project
- ➔ Meteor Catcher Game Part 1 Reference Guide

Prior Knowledge

Before embarking on this project, we recommend that you:

- ❏ can explain what a **variable** is in your own words and describe how they can be used in a program.
- ❏ can explain what a **conditional** is in your own words and describe how they can be used in a program.

Women in Tech Spotlight: Cassie Tarakajian

Image Source: NYU Tisch

Cassie Tarakajian is a software developer, hardware engineer, creative technologist, musician, and educator. Cassie also identifies as **non-binary** and uses the pronouns they/them. Cassie first learned how to code in college, taking an Introduction to **Java** class. They recall learning only by using a very simple text editor to complete very simple text-based tasks. Later on, Cassie was asked to contribute to an open-source project called **p5.js**, a JavaScript library for making interactive art and sound based on the Processing platform. As the creator and lead maintainer of the p5.js web editor, they developed the environment where people could write and run code right in the browser, making it easy for beginners to use. This platform is also fully accessible to members of the community who are visually impaired, offering compatibility with screen readers and high contrast view.

On top of Cassie's work in p5.js, they are an Engineer at **Cycling '74**, cofounder of **Girlfriends Lab**, and an Adjunct Professor at **Tisch School of the Arts at NYU**. Cassie continues to embody all of these roles everyday to prove that the image of a programmer is not limited to just one narrative.

Watch this **video** (to 6:00) to learn more about Cassie and their work with p5.js. Want to learn more about Cassie and their work? Check out their **personal website**. Read this **article** on their work around p5.js and how they got started.

Reflect

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

PURPOSE

When creating p5.js it was important to Cassie to create an environment that made learning how to code easy while also being accessible to a variety of needs. A large component of accessibility in p5.js is compatibility with screen readers and offering customizable settings. Why do you think it was important to Cassie to build a web editor "for all"?

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

Step 1: Explore the meteor catcher game (10-15 mins)

Meet the Parts of a Game (2 min)

All (or a great deal of) games have six parts: a goal, a challenge, core mechanics, components, rules, and space. These parts work together as a system that generates play among the people involved. As a game designer, it is essential that you understand how all the parts work together as a system to program your game.

- ➔ **Goal**: What does a player or team have to do to win the game?
- ➔ **Challenge**: What obstacles are in the player's way of reaching the goal?
- ➔ **Core Mechanics:** What core actions or moves does the player do to power the play of the game?
- ➔ **Components**: What parts make up the materials of play?
- ➔ **Rules**: What relationships define what a player can and cannot do in a game?
- ➔ **Space**: Where does the game take place?

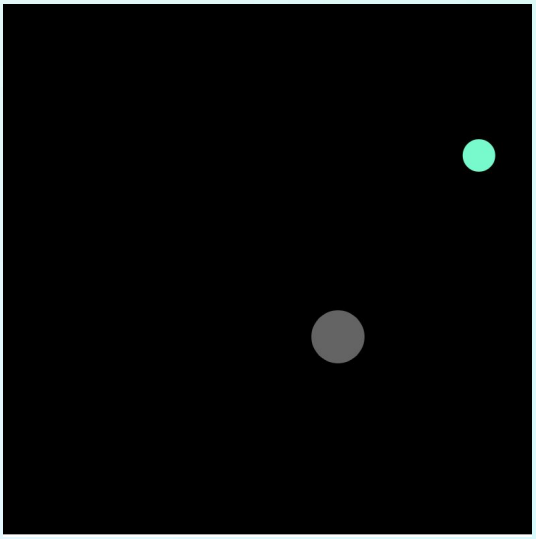
Example

Tic Tac Toe Example

- ➔ **Goal**: Be the first to get three in a row
- ➔ **Challenge**: You don't know where your opponent will place their symbol
- ➔ **Core Mechanics:** Blocking and writing
- ➔ **Components**: Writing utensils, paper, players, Xs and Os
- ➔ **Rules**: There are 2 players. Each player alternates turns writing their symbols until the grid is full or one player gets three in a row, vertically, horizontally, or diagonally.
- ➔ **Space**: 3x3 grid on a piece of paper

Play the Game (1 min)

The best way for a game designer to practice their skills is to play games! Let's try out the game we will be building. Click this link to play the game for about 30 seconds. As you play, try and identify the parts of this game.



We will plan and build a game about catching meteors to learn new programming skills, but in the Extensions section you will have a chance to customize it

Identify the parts of a game (5-10 mins)

Our goal in this step is to understand how the system of our Meteor Catcher game works by breaking down big problems into smaller parts. This process is called **decomposition**.

Let's say your task is to make 100 pizzas for a group of kids. A huge job, but if we break it down into smaller steps and subproblems, it is much more manageable. For example, make all the dough first, cook all the sauce, assemble five at a time, then cook them in batches.

There are lots of different ways you can break a complex problem into simpler parts. Since we already have the game to play, we will take a **reverse engineering** approach. This means that instead of just trying to build the game from scratch, we will deconstruct the finished product to figure out how it was made. First, we'll define all the parts of the Meteor Catcher game then use those to write pseudocode. Try breaking down the parts of the Meteor catcher game in the space below. Don't forget to check your ideas with the **Reference Guide**

The parts of a meteor catcher game

➔ Describe the **goal** of Meteor Catcher, the game you just played in the last step. What does a player or team have to do to win the game?

➔ The **components** of Meteor Catcher include the meteor, catcher, walls, and player. Each component has unique properties (e.g. size, color, shape, etc) and actions (i.e. the things it does - the verbs you associate with that component) that contribute to the game's system. For example, a meteor property would be round and a meteor action would include falling from top of screen to the bottom. *Spend **2-3 minutes** thinking about the properties and actions for each component in the table below.*

COMPONENT	PROPERTIES	ACTIONS
What are the essential pieces for play?	*What are the **attributes** or **characteristics** of the component? (e.g. size, color, shape, etc)*	*What does it **do**? What **verbs** do you associate with it?*

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

➔ Describe the **space** of the game. Where does it take place? (Note that sometimes the space can be more than one thing. For example, chess takes place on the chess board, but it also takes place in a living room, park, or cafeteria.)

➔ Define the **challenge**. What obstacles are in the player's way of reaching the goal?

➔ Describe the game's **core mechanic**. What core actions or moves does the player need to make to play the game? What core actions or moves does the player do to power the play of the game?

➔ Write a list of the game's **rules**. Rules determine what we can and cannot do in our game. They can be applied to players, components, the space, etc.

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

Step 2: Write pseudocode for your game (5-10 mins)

In this step, try writing out the instructions for your program at a high level on a piece of paper or on the computer. This is called **pseudocode**. Pseudocode is a plain language description of what your code will do. It helps you think through the flow and logic of your program so you can determine the steps you need to take to write your program. Pseudocode can look very code-like without using specific code syntax. For example, you might use if or other core keywords that apply to all programming languages. Always start with pseudocode!

We have already filled in a few things to get you started. You should also use the parts of the game from above to help you determine what you need to include. If you get stuck, ask yourself questions about the game. For example:

- ➔ What needs to happen at the start of the game?
- ➔ What needs to happen while the game is being played?

Try to be specific, but don't worry if you don't capture everything or if you write down something you don't know how to do. Everyone writes pseudocode differently! We will return back to this pseudocode during the project, so be sure to save your work.

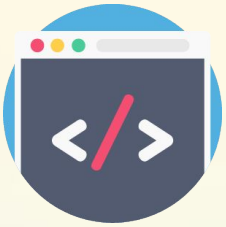
```
// Starter pseudocode
Declare any variables

DO THIS ONCE
  Set the size of the canvas to 400 pixels by 400 pixels

DO THIS EVERY LOOP
  Set the background color
  // We will discuss why this lives in draw() vs setup() later on!
  Draw the meteor
  // Try adding the rest on your own!
```

What happens in the game after the meteor is drawn on the screen? Try replaying the game if you can't remember.

***Tip**: The components and rules will be especially helpful!*

Don't forget to check your ideas with the Reference Guide on pg 4.

Step 2: Meet p5.js! (10-15 mins)



P5.js (or just p5) allows you to create interactive art for web browsers. It is a tool for creative coding - projects that use code for expression instead of just functionality. P5.js is a library for JavaScript, a programming language that allows you to add interactivity on the web. Being a library means that p5 is JavaScript, but the creators made a collection (or library) of specialized functions/methods so you don't have to do everything from scratch. Since it is web-based, you can easily share all of your work!
You can read more about the origins and community on the **p5 homepage**. Check out the **Showcase page** to see some example projects people have made with it.

The p in p5.js stands for Processing. Processing is a programming language built for artists and designers to integrate code into their projects. Processing was designed for beginners to easily create a range of interactive media from animations to data visualizations to musical instruments to games to large scale installations. Visit the **Processing Foundation homepage** to learn more about it.

Create Your Account (3-5 mins)

There are two ways you can use p5.js: the online web editor or a text editor and copy of p5.js that you download to your local computer. The easiest way to get started with p5 is the online editor. This allows you to write code and run your program in a web browser. In this tutorial, we are only going to use the web editor to reference steps and illustrate examples.

To get started with the web editor, you need to create an account.

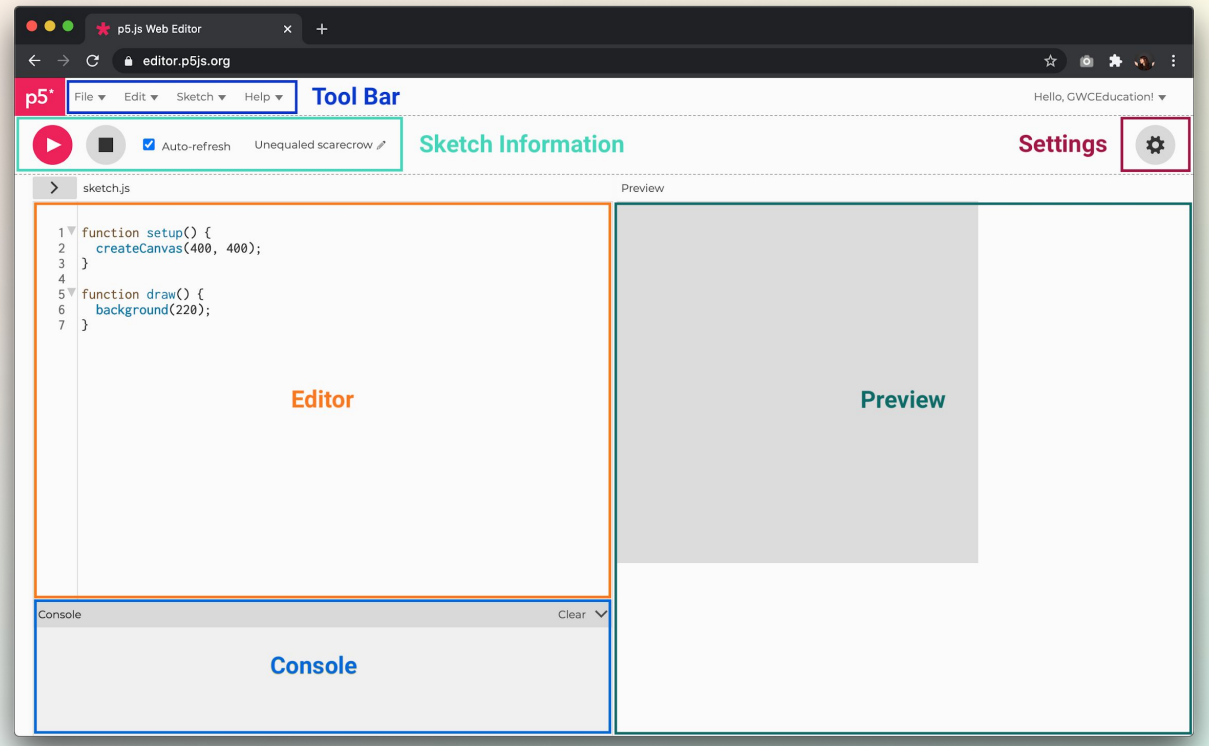
- ❏ Go to **https://editor.p5js.org/signup**.
- ❏ **Sign Up.** Fill in all the fields (username, email, password, and password confirmation) then click "Sign up" or you can choose to sign in with Google or GitHub.
- ❏ **Confirm your Email.** You will receive an email to confirm and verify your account (check Spam if it doesn't show up in 3-5 minutes). Click the link, then sign in with your shiny new credentials (i.e. username and password).
- ❏ **Save your credentials in a safe place so you can log in again.** If you do forget your password, go to the **Log In** page and click "Reset Your Password" at the bottom.

If your internet connection is intermittent or you would rather work in an editor locally, you can explore the second option. See this **Getting Started page** for the materials you will need and instructions on how to download the library. If you need more support, don't be afraid to Google!

If working offline, the examples might look different, but the outcome will be the same.

Explore the environment (5-8 mins)

Now that you have an account, let's examine the interface of the p5 online editor. This is an IDE or integrated development environment that allows you to write and run programs in one place. The programs written in p5.js are called sketches. You can think of this environment like a sketchbook that already has your tools at your fingertips!



➔ **Tool Bar:** At the top of the page is the toolbar.

- In the **File** menu, you can create a sketch, save a sketch, duplicate a sketch, share a sketch in multiple formats, download sketch files, open a sketch, and open examples. Note: Some of these options will not show up until you save your sketch.
- The **Edit** drop drop down allows you to tidy your code, find a character or word in your sketch, and navigate through them.
- In the **Sketch** menu, you can add files or folders to your code and run or stop your sketch.
- You can find always helpful keyboard shortcuts, a link to the p5 reference page, and more about p5 in the Help menu.

Check out some of the keyboard shortcuts on p5.js **here**.

- **Sketch Information:** Below the toolbar is a play button and a stop button. The play button starts running the program. The stop button stops the program. You can check the 'Auto-refresh' box if you want the program to keep running after you make changes instead of having to click the play button again.

  To the right, you will see a pre-populated title for your sketch. To rename your sketch, click the pencil icon and type in the new title.

- **Settings:** You can access the settings by clicking the gear icon to the left of the Sketch Information. Here you can change the theme, text size, and accessibility settings (we will talk more about accessibility in a bit). We highly recommend you turn autosave on in the General settings.

- **Editor:** The editor is where you write your code. Each line has a number so you can easily reference it. The small arrows next to a number mean that you can click it to collapse the text. For example, if you don't need to see multi-line comments, you can collapse those.

- **Preview:** This window displays the results of your code when you run the program.

- **Console:** Below the editor is the console. This window prints information about your program, such as error messages or data that you want access to in a program, like the value of a variable.

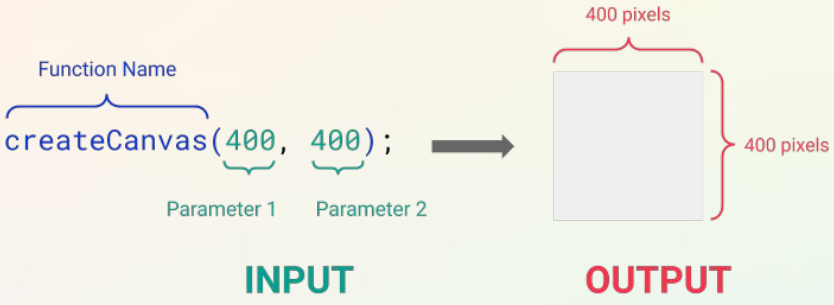
Accessibility in p5.js (2 mins)

p5 developers have placed a high priority on making the editor accessible to those who are visually impaired. These tools are in active development and are part of a larger ongoing research project hosted at NYU.The online editor website and editor itself are readable by screen readers. Much of the accessibility development has been toward making the visual output in the preview window readable by a screen reader. For more information about using this functionality see **this page on the p5 website**.

As you continue learning how to program across different languages and platforms, you should always keep accessibility and inclusivity at the forefront. Historically, designers, engineers, and programmers did not prioritize people with disabilities as they created software and hardware. With the rise of facial recognition and other software, this also applies to people of color, women, and other marginalized communities since the implicit biases of programmers can translate into their code. This is beginning to change as awareness increases, but there is still much work to be done. Take the time to ensure everyone can use what you build!

Step 4: Examine p5.js functions (5-10 mins)

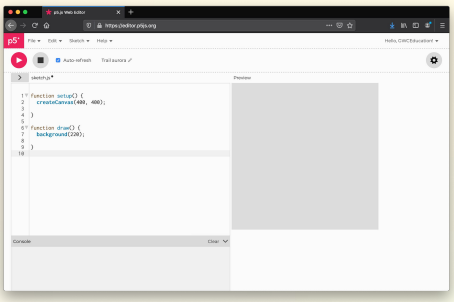
A **program** is a set of instructions you create for a computer to follow. Instead of writing the same instructions over and over, we can group instructions into chunks so we can reuse them later. These chunks are called **functions**. Functions are lines of code that perform a set of actions. You can think of them like verbs - they *do* something. In p5, we give instructions to our program in the form of functions. Most of the functions you will use are defined in the p5.js library (you can also create your own functions, but we will not cover how to do this in the current tutorial).



When we call or use the function, the program runs the code inside it. For example, one of the most important functions is the `createCanvas()` function. This function creates the canvas element that draws the graphics and displays the sketch. In other words, it determines the screen size. But how do we tell the function what size screen we want? To do this, we pass **parameters** through the function to get the output we want. Parameters are input values that the function uses to execute the function. Let's examine the syntax of the `createCanvas()` function:

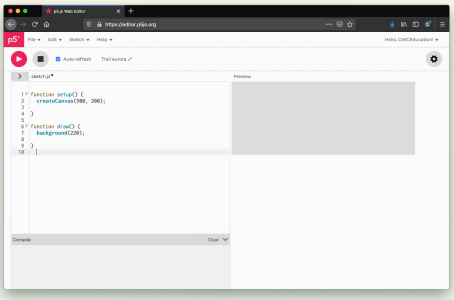
JAVASCRIPT	DESCRIPTION
`createCanvas(width, height);`	➔ `createCanvas`: The function name. ➔ `()`: We use parentheses to tell our program that it needs to call the function. Sometimes we include parameters or inputs in the function inside our parentheses. ➔ `width`: The first parameter that sets the width of the canvas in pixels. ➔ `height`: The second parameter that sets the height of the canvas in pixels. ➔ `;`: All lines of code in JavaScript must end with a semicolon.

The parameters set the dimensions of the canvas in pixels. Pixels are the graphic building blocks of digital screens. Each pixel represents a single point on the screen and has a single color. You will need to include this function in every p5 sketch.



Try changing the parameter values to resize the canvas:

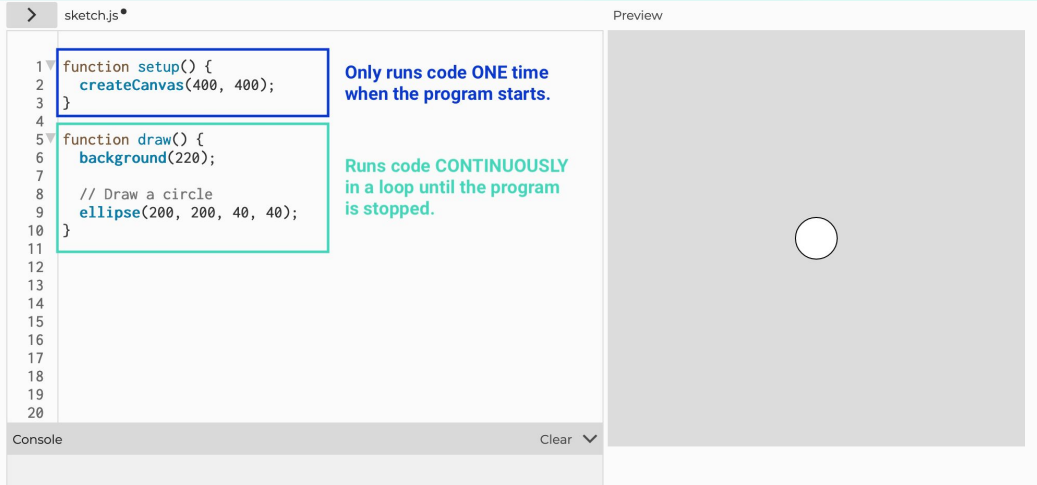
- ❏ **Open the p5 web editor and login.** You may have noticed that the sketch came prepopulated with starter code, including our friend `createCanvas()`. The default size of the canvas is 400 pixels wide and 400 pixels high.
- ❏ **Click the play button to run the code.** Notice the size of the canvas.
- ❏ **Try changing one or both values, then click the play button to run the code again.** Check the auto-refresh box so you don't have to hit the play button after each change you make.



Voila! A gray box the size of your parameters should appear In the preview window.

Step 5: Learn about program flow (5-10 mins)

We know how to give our program instructions, but where do we put those instructions? When do they run? Does the order of those instructions matter? Can functions go inside other functions? All of these questions relate to **program flow**. This refers to the order in which the program runs your lines of code. In p5.js, the program runs each line of code in sequence. This means it runs the first line of code, then line 2, then line 3, etc.. Later we will learn how to control your program's flow with conditionals, but first we need to know about the two core functions in p5.js: `setup()` and `draw()`.



	DEFINITION *What is it?*	CONTENTS *What should I put inside it?*
`setup()`	The `setup()` function runs only one time when your program starts. There is only one per sketch and it cannot be called again after the first time.	Any functions that you want to run immediately when the program starts, such as screen size with `createCanvas()`, background color (sometimes), and to load media such as images and fonts as the program starts. If you create any variables here, you cannot access them in `draw()` or other functions.
`draw()`	The `draw()` function runs the lines of code contained inside its block **continuously** until the program is stopped. It is the main loop and it is where the action happens. There is only one per sketch and it is called after the `setup()` function.	Anything that you want to happen repeatedly.

To get a better understanding of the differences between them, we will examine how a sketch changes based on where we put the background() function. This sets the color used for the background of the p5.js canvas. It can take many different color value parameters including RGB and hex values. We will talk more about color in the next section. For now, we will just pass a single value between 0 (black) and 255 (white) for a grayscale color.

JAVASCRIPT	DESCRIPTION
`background(redValue, greenValue, blueValue);`	➔ `background`: The function name. ➔ `()`: We use parentheses to tell our program that it needs to call the function. Sometimes we include parameters or inputs in the function inside our parentheses. ➔ `redValue`: The red value between 0 and 255. ➔ `blueValue:` The blue value between 0 and 255. ➔ `greenValue`: The green value between 0 and 255. ➔ `;`: All lines of code in JavaScript must end with a semicolon.

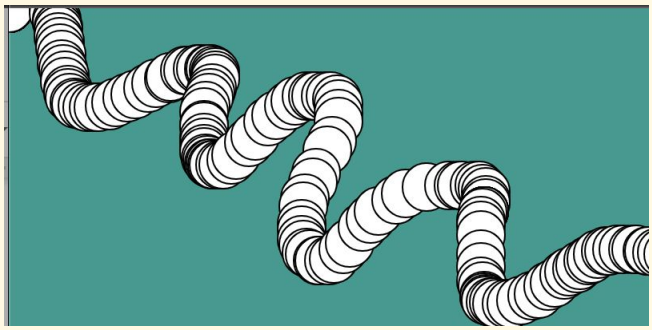
Placing the background function in the `setup()` or `draw()` of a sketch yields very different results. Consider this code:

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

function draw() {
 ellipse(mouseX, mouseY,50,50);
}
```

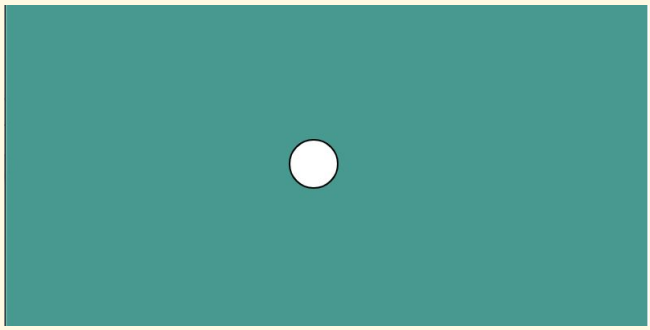
Right now, this sketch does not have a background. It creates a canvas and draws a circle or ellipse to the screen at the position of the mouse. Since the `ellipse()` function is in `draw()`, our program paints a new circle to the screen each time the program loops through, about 60 times per second, forever or until we tell the program to stop.

Let's take a look at two example sketches to illustrate the different program flow of the `draw()` and `setup()` functions. One has `background()` in `setup()` and one has it in `draw()`.

SKETCH 1



SKETCH 2



- ❏ **Open Sketch 1 and move your mouse over the canvas. T**he circle follows your mouse and leaves a trail of other circles on its path.
- ❏ **Open Sketch 2 and move your mouse over the canvas.** Now the circle follows your mouse and does not leave a trail.
- ❏ **Think about it:** Which sketch has the `background()` function in `setup()`? Which sketch has the `background()` function in `draw()`? Why?

Don't forget to check your ideas with the Reference Guide on pg 5.

Step 6: Check for Understanding (2 mins)

Take a moment to check your understanding of program flow as it relates to the `setup()` and `draw()` functions in p5.js.

You decide to surprise your friend by making their favorite for dinner: one batch of dumplings. You remember the steps, but want to make sure you have them in the right order. You find a past program you wrote for dumplings, but it is incomplete!
Based on what you learned about program flow, where would you put the following actions, or "functions", in your "program" so it runs properly?

Functions you need to add:

- ➔ `Close wrapper`
- ➔ `Measure filling ingredients`
- ➔ `Remove dumpling from pan`

Current program:

```
setup() {
  Mix filling ingredients
  Collect all dumpling wrappers
}

draw() {
  Spoon filling into wrapper
  Place dumpling in pan
  Cook dumping
  Eat dumpling
}
```

Don't forget to check your ideas with the Reference Guide on pg 6.

Step 7: 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 pseudocode with us! Don't forget to tag `@girlswhocode #codefromhome` and we might even feature you on our account!

Stay tuned for more Girls Who Code at Home projects!

Girls Who Code At Home

Meteor Catcher Game: Part 1

Reference Guide

Meteor Catcher Game: Part 1 - Reference Guide

In this document you will find all of the answers to some of the questions in the activity. Follow along with the activity and when you see this icon, stop and check your ideas here.

Step 1: Identify the Parts of A Game

The parts of a meteor catcher game

➔ Describe the **goal** of Meteor Catcher, the game you just played in the last step. What does a player or team have to do to win the game?

The goal of Meteor Catcher is to catch as many meteors as possible. Right now, this game doesn't have a very defined goal. We left it open so you can customize your own after you build the base game.

➔ The **components** of Meteor Catcher include the meteor, catcher, walls, and player. Each component has unique properties (e.g. size, color, shape, etc) and actions (i.e. the things it does - the verbs you associate with that component) that contribute to the game's system. For example, a meteor property would be round and a meteor action would include falling from top of screen to the bottom.

*Spend **2-3 minutes** thinking about the properties and actions for each component in the table below.*

COMPONENT	PROPERTIES	ACTIONS
What are the essential pieces for play?	*What are the **attributes** or **characteristics** of the component?*	*What does it **do**? What **verbs** do you associate with it?*
meteor	➔ Round ➔ Teal ➔ Random diameter between 10 and 40 pixels	➔ Fall from the top of the screen to the bottom ➔ Moves at a random speed ➔ Starts in different locations at the top of the screen ➔ Can intersect with the catcher ➔ Can intersect with the ground (i.e. bottom of screen)
catcher	➔ Round ➔ Transparent white ➔ 40 pixels in diameter	➔ Follows the mouse. ➔ Can "catch" or collect meteors by intersecting with them.
walls	➔ 400 pixels in width ➔ 400 pixels in height	➔ Intersects with meteors. ➔ The bottom wall causes a new meteor. to appear if a meteor touches it.
player	➔ Likes space! And Meteor showers!	➔ Moves the mouse to catch the falling meteors.

➔ Describe the **space** of the game. Where does it take place? (Note that sometimes the space can be more than one thing. For example, chess takes place on the chess board, but it also takes place in a living room, park, or cafeteria.)

The game takes place in a 400 by 400 square window on a webpage. From a story perspective, it takes place in actual outer space.

➔ Define the **challenge**. What obstacles are in the player's way of reaching the goal?

The challenge is to catch the falling meteors. Each one starts at a different location, falls at a different speed, and renders at a different size each time a new meteor appears on screen.

➔ Describe the game's **core mechanic**. What core actions or moves does the player need to make to play the game? What core actions or moves does the player do to power the play of the game?

Players catch meteors. They move their mouse around the screen to collect them.

➔ Write a list of the game's **rules**. Rules determine what we can and cannot do in our game. They can be applied to players, components, the space, etc.

- Meteors fall from the top of the screen to the bottom.
- Only one meteor falls at a time.
- The catcher follows the mouse.
- The player's catcher must intersect with the meteor in order to catch the meteor.
- If a meteor is caught, it disappears and a new meteor is drawn at the top of the screen.
- If a meteor touches the bottom of the screen, it disappears and a new meteor is drawn at the top of the screen.

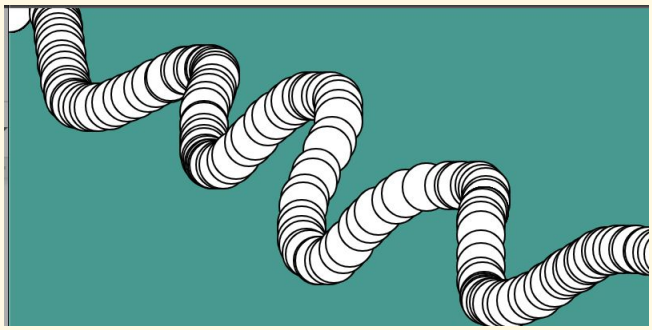
Step 2: Write pseudocode for your game

```
Declare any variables

DO THIS ONCE
  Set the size of the canvas to 400 pixels by 400 pixels

DO THIS EVERY LOOP
  Set background color
  Draw the meteor
  Make the meteor fall
  Draw the catcher to follow the mouse
  Find/Calculate the distance between meteor and the catcher
  Test to see if meteor and catcher have intersected. If they intersect,
    Redraw the meteor at the top of the screen to a random location,
    Give it a new speed,
    Set a new diameter.
  Test to see if meteor and bottom wall have intersected. If they intersect,
    Redraw the meteor at the top of the screen to a random location,
    Give it a new speed,
    Set a new diameter.
```

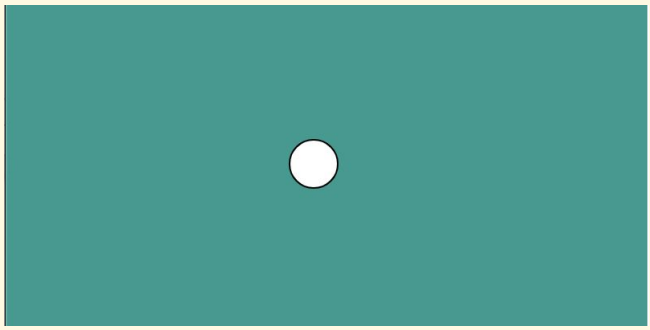
***Remember**: There is more than one way to write a program, so there is more than one way to write your pseudocode as well!*

Step 5: Learn about program flow

SKETCH 1



SKETCH 2



Sketch 1 has the `background()` function in `setup()`. This means that the background is only drawn one time. Since the `ellipse()`function is in `draw()`, p5 draws a new circle at the mouse position on the screen every time the program runs through a loop. You could describe it like this: Fill background, draw circle, draw circle, draw circle, draw circle, draw circle, etc.

```
// Sketch 1
function setup() {
  createCanvas(400, 400);
  background(220);
}

function draw() {
  ellipse(mouseX, mouseY,50,50);
}
```

Sketch 2 has the `background()` function in `draw()`. This means the program draws the background and the circle every time the program loops through. This gives the appearance that the circle moves smoothly through space as we move the cursor, even though the program is drawing new circles just like in the first sketch. You could describe it like this: Fill background, draw circle, fill background, draw circle, fill background, draw circle, fill background, etc.

```
// Sketch 2
function setup() {
  createCanvas(400, 400);
}

function draw() {
  background(220);
  ellipse(mouseX, mouseY,50,50);
}
```

Step 6: Check for Understanding

The actions in `setup()` only need to happen one time since you are only making one batch. The actions in `draw()` need to happen for each dumpling you make. Since you are making multiple dumplings, we place these actions in `draw()`.

```
setup() {
  Measure filling ingredients
  Mix filling ingredients
  Collect dumpling wrappers
}

draw() {
  Spoon filling into wrapper
  Close wrapper
  Place dumpling in pan
  Cook dumping
  Remove dumpling from pan
  Eat dumpling
}
```

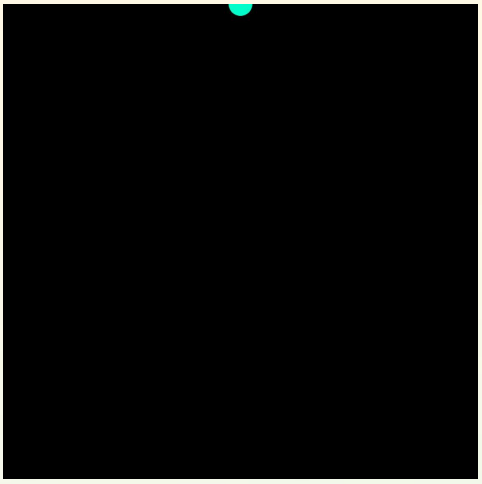
girls who code

Girls Who Code At Home

Meteor Catcher Game: Part 2

Draw the Meteor

Activity Overview



Welcome back! Last week you explored the p5.js environment and began planning the parts of your meteor catcher game. In this part, you will use the coordinate system to draw the first component of your game: the meteor! You will also learn more about how to use color in p5.js and set the color for your meteor and sketch background.

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

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!

Learning Goals

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

- ❏ describe the p5.js coordinate system and its relationship to pixels on the screen.
- ❏ use built in functions and commands to draw basic shapes on the coordinate plane.

Materials

- ➔ p5.js Online Editor
- ➔ Meteor Catcher Game Sample Project
- ➔ Meteor Catcher Game Part 2 Reference Guide

You can also follow along with Part 2 in the Meteor Catcher Game video tutorials!

Women in Tech Spotlight: Phoenix Perry

Image Source: Hackaday.io

When Phoenix was younger, her parents bought her an **Atari** home game console. Being introduced to this new world of technology sparked Phoenix's interest in technology and she began learning how to program in **BASIC**. After graduating with an undergraduate degree, Phoenix began her career as a Web Designer for **Evite**. Here she endured many endless nights, a normal behaviour in the company, and took pain killers from the strains on her wrists due to lack of rest. She then developed **carpal tunnel**, a common nerve syndrome common in programmers due to increased tensions in the bones and ligaments on your hand.

Phoenix spent many years away from the technology industry due to her condition, working as an Art Director. She then came across the **Integrated Digital Media program at NYU Tandon**, where she became an Adjunct Professor and Researcher to teach about the design, play, and embodiment of game development. It was through this experience she went on to become a co-founder of **Code Liberation**. Code Liberation hopes to teach, prepare, and support women, nonbinary, femme, and girl-identifying people to pursure jobs in STEAM. They offer free classes, workshops, game james, hackathon, and social game nights to women of all ages and at different stages of their careers. Phoenix uses her experience to help the next generation of women develop skills needed to join the technology industry. She has now partnered with **University of Arts London** to open a chapter of Code Liberation in the United Kingdom.

Watch this **video** to learn more about Phoenix and why diversity is important in technology. Want to learn more about Phoenix? Watch her **TED Talk** , explore her **personal website**, and read this **article** to learn more about Phoenix's academic journey and hardships to get to where she is today!

Reflect

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

BRAVERY

After developing carpal tunnel, Phoenix found herself unable to work and move. This made it difficult for her return to the technology industry. Discuss how Phoenix displayed bravery in returning to industry while also supporting and preparing other women at Code Liberation.

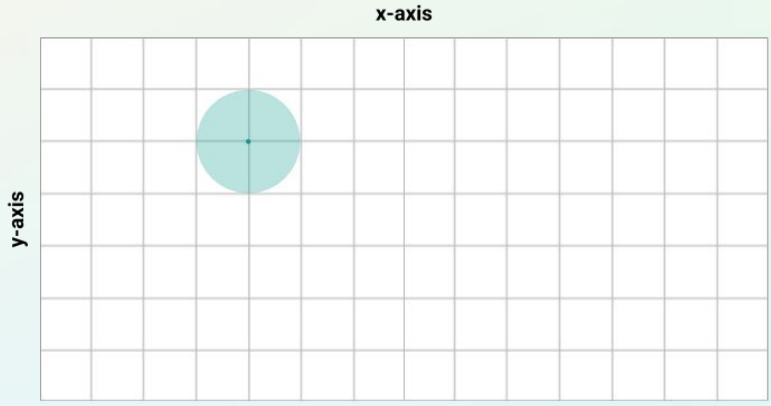
Share your responses with a family member or friend. Encourage others to read more about Phoenix to join in the discussion!

Step 1: Command a circle (2-4 mins)

Let's say your friend gives you an instruction to draw a circle on a piece of paper. You could either just draw the circle or ask for more information. If you ask for more information, you might inquire - where on the paper? How big? What color? A perfect circle or more of an oval? To answer the question of where, your friend might say "A third of the way from the left side and three-fourths of the way down towards the bottom." That kind of instruction might work if you are talking to a human and don't need to be specific. But it won't work for a computer! Instead we can use the coordinate system to specify a location for elements we want to display in our program.

The coordinate system is a system that uses one or more numbers to identify the location of a point in space. Coordinate systems can be on a 2D plane or in 3D space. Coordinate planes (i.e. 2D) have an x-axis that runs horizontally and a y-axis that runs vertically to form a grid. They use an ordered pair to signify a point: (x position, y position).

p5.js can also work in three dimensional space with a z-axis. You may see the option for a z-axis parameter in p5 documentation on some functions, but you should not include it.

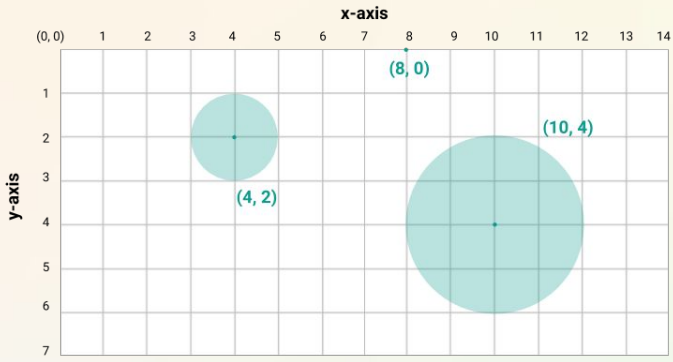


Consider the coordinate plane above. Think about the instructions you would give to a computer to draw the circle in that location.

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

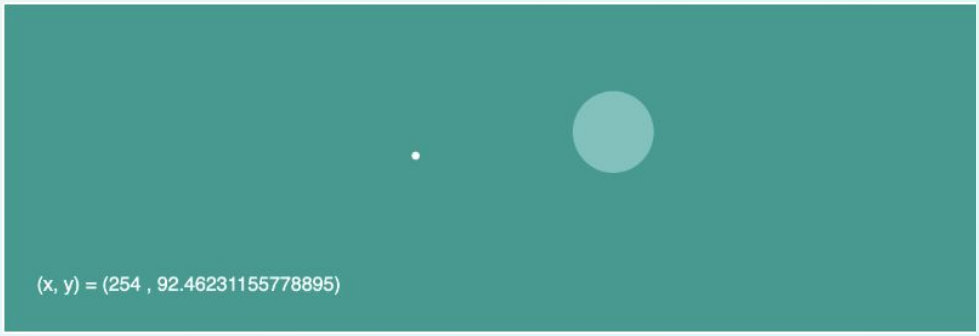
Step 2: Practice using the coordinate system (5-8 mins)

Each pixel on your screen has a unique address in the coordinate system. In order to draw pixels to the screen, we must give our program the x coordinate (i.e. the location on the x-axis) and the y coordinate (i.e. the location on the y-axis) of the pixel.



The origin or (0, 0), on the screen coordinate system is located at the top left corner. As you move right on the screen, the value of the x-coordinate increases. As you move down on the screen the value of the y-coordinate increases.This may appear a little different than the coordinate system layouts you have seen in math class, where the origin is in the center or at the bottom left corner.

Explore this **sketch** and use your mouse around to try to estimate the center, width, and height of the circle.



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

Step 3: Create your project sketch (5-10 mins)

In the remaining sections, we will start writing the code for your game. First we need to create your main project sketch.

- ❏ **Log into the p5.js online editor.** The editor automatically gives you a blank sketch with starter code. *Alternatively, you can create a new sketch by going to File > New.*
- ❏ **Click the pencil icon to name the sketch** to something that you can easily recognize like Meteor Catcher Game v1. *Note: This is in the sketch information area below the toolbar.*

- ❏ **Next, go to File and click Save.** You can also save by using the keyboard shortcut Command S (Mac) or Control S (Windows). Be sure to navigate inside the editor before using these shortcuts.
- ❏ **Create a multiline comment at the very top of your sketch with the following information:**
  - ❏ **Title of program:** This should be the same as the sketch name.
  - ❏ **Version of program:** Is this the first version or second? If you make big changes, it's good practice to create a new version.
  - ❏ **Author**: By (your first name and last name initial).
  - ❏ **Description**: A sentence or two about what it does.

At the top of your sketch, you will include a comment that gives basic information about the sketch. Use **code comments** to remind yourself of how something works, the reasoning for a decision, or a follow up task.

- ➔ Single line comments use a double forward slash, `//`.
- ➔ Multiline comments use a forward slash and asterisk, `/*`, to open it and an asterisk and forward slash, `*/`, to close it.

```
// This is a single line comment

/*
This is a
multiline comment
*/
```

Step 4: Draw the meteor (3-5 mins)

Now that we have our project sketch saved, let's create our first game component: the meteor! First we will learn how to draw the shape at a specific location, then we will fill it with color to change the appearance.

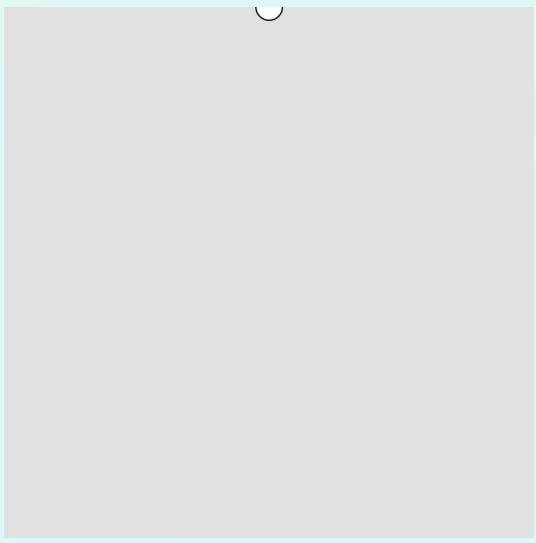
Shape and Location

Think back to the last step where we learned about the coordinate system. Based on what we know now, if we want to draw a shape, we have to tell the computer to draw each pixel individually. This gets pretty tedious very quickly, so p5 created pre-made functions that draws shapes for us. All we need to do is define the location where we want the shape, then give the width and height. Let's examine the syntax for a circle below:

JAVASCRIPT	DESCRIPTION
`ellipse(x, y, width, height);`	➔ `ellipse`: The function name. Ellipse is another word for oval. p5.js Reference ➔ `()`: We use parentheses to tell our program that it needs to call the function. Sometimes we include parameters or inputs in the function inside our parentheses. ➔ `x`: The x-coordinate at the center of the ellipse. ➔ `y`: The y-coordinate at the center of the ellipse. ➔ `width`: Sets the width of the ellipse in pixels. ➔ `height`: Sets the height of the ellipse in pixels. ➔ `,`: We use commas to separate the different parameters or inputs in the functions. ➔ `;`: All lines of code in p5.js must end with a semicolon.

Use the `ellipse()` function to draw the meteor to the screen:

- ❑ Add the `ellipse()` function to your sketch inside of the `draw()` function.
- ❑ Place it in the center of the canvas (your x-axis position) at the very top (your y-axis position). Reference the graph above if you need a refresher.
- ❑ Set the size of the circle, or ellipse, to be 20 pixels wide and 20 pixels high.
- ❑ Add a comment to remind yourself that this is the meteor.
- ❑ Run your code by pressing the play button to test it.



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

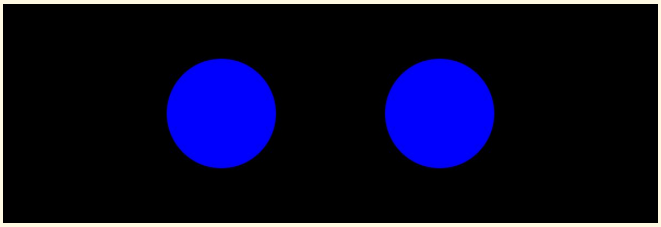
Step 5: Fill the meteor with color (5-8 mins)

To add color to shapes, we can use the `fill()` function. This will fill any shape drawn after the command with the specified color. In the example below, both circles are blue because they come after the `fill()` function:

JAVASCRIPT

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

function draw() {
  background(0);

  // Make both circles blue
  fill(0, 0, 255);
  ellipse(width / 3, height / 2, 100, 100);
  ellipse(width / 3*2, height / 2, 100, 100);
}
```

RESULT



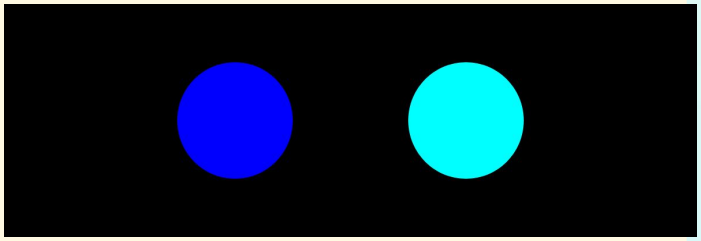
But what if we want the second shape to be teal instead of blue? We need to add another `fill()` function before that shape:

JAVASCRIPT

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

function draw() {
  background(0);

  // Blue circle
  fill(0, 0, 255);
  ellipse(width / 3, height / 2, 100, 100);
  // Teal circle
  fill(0, 255, 255);
  ellipse(width / 3*2, height / 2, 100, 100);
}
```

RESULT

This change would result in one blue circle and one teal circle



The `fill()` method use RGB color mode that uses a combination of red, green, and blue light to create a spectrum of colors.

- ➔ **(0,0,255)** would display a **blue** color
- ➔ **(0,255,255)** would give a **teal** color.

Check out the syntax for the `fill()` function using RGB color mode below. RGB color mode uses combinations of red, green, and blue light to create a range of digital colors. You can assign a value to each color between 0 to 255. For example, (255, 0, 0) would be red, (0, 0, 0) would be black, and (255, 136, 0) would be orange. You can play around with different values and colors using tools like color pickers or palette tools like Coolors.

JAVASCRIPT	DESCRIPTION
`fill(redValue, greenValue, blueValue);`	➔ `fill`: The function name. ➔ `()`: We use parentheses to tell our program that it needs to call the function. Sometimes we include parameters or inputs in the function inside our parentheses. ➔ `redValue`: The red value between 0 and 255. ➔ `blueValue:` The blue value between 0 and 255. ➔ `greenValue`: The green value between 0 and 255. ➔ `;`: All lines of code in p5.js must end with a semicolon.

Use the `fill()` function to add color to your meteor:

- ❏ Pick a color for your meteor and make a note of the RGB values.
- ❏ Call the `fill()` function with your meteor's RGB values (remember that location is important!).
- ❏ Press the play button to run your program when you are finished. The color of your meteor should change to the one you specified.

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

Step 6: Change the outline (3-5 mins)

You might have noticed that there is a black outline around your meteor. We can remove this outline using the `noStroke()` function. Calling this function means that none of the shapes in your sketch will have outlines. If you want to enable outlines, you need to call the `stroke()` function above that shape.

You can use other functions like `stroke()` and `strokeWeight()` to control the color and thickness of shape borders.

Remove the outline using the `noStroke()` function:

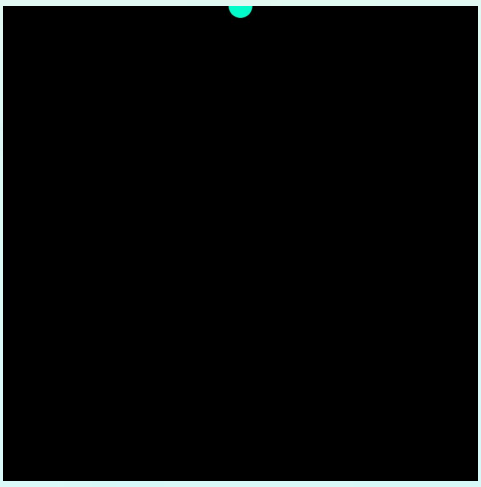
- ❏ Add the `noStroke()` function under the `background()` function. Since we don't want outlines on any of our shapes, we will place it at the top.
- ❏ Run the program when you are done.

An outline will no longer be visible on your meteor. Here is a close up screenshot of how the meteor should look at the top of your sketch.

Don't forget to check your code with the Reference Guide on pg 4.

Step 7: Add the background color (3-5 mins)

Put your color knowledge to the test! Fill in the canvas using the `background()` function. Right now, it only has one value, but similarly to the `fill()` function, it can take parameters for other color modes like RGB.



Change the background of your game:

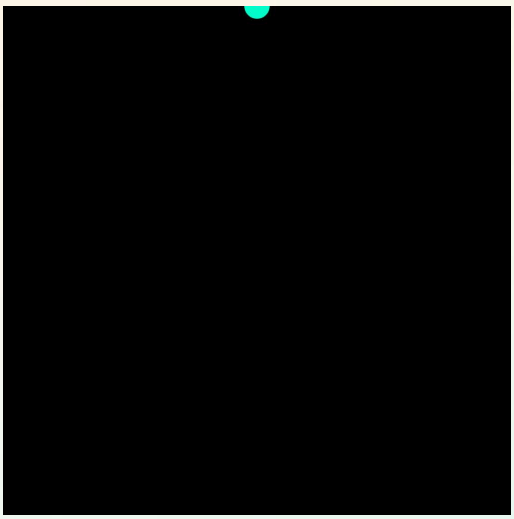
- ❏ Choose a background color and make note of its RGB values. (Here are the color pickers and palette tools like Coolors mentioned earlier.)
- ❏ Update the `background()` function inside `draw()` with your new color values.

Don't forget to check your code with the Reference Guide on pg 4.

Step 8: Test your code (3-5 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 400 by 400 canvas with a color background.
- ➔ A circle meteor in the top center of the canvas with a new color and no border.



Not working the way you want it to? If you have an error that prevents the code from compiling and running, p5.js will display an error message in the console. When something isn't working properly, start there to figure out the problem.

Otherwise, 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? Remember that JavaScript is also case-sensitive!
- ➔ Are your functions in the correct location and sequence? Remember that order matters in program flow!
- ➔ Are your parameter values within the correct range for the function? For example, is there an x value of 500 even though the canvas is 400 pixels wide? Are your RGB values between 0 and 255?

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

Step 9: Check for Understanding

Describe the location, size, and color of the shape in the code below:

```
function setup() {
  createCanvas(100, 100);
}

function draw() {
  fill(0, 0, 255);
  ellipse(50, 50, 5, 5);
}
```

Don't forget to check your ideas with the Reference Guide on pg 5.

Step 10: 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 pseudocode with us! Don't forget to tag `@girlswhocode` `#codefromhome` and we might even feature you on our account!

Stay tuned for more Girls Who Code at Home projects!

Girls Who Code At Home

Meteor Catcher Game: Part 2

Reference Guide

Meteor Catcher Game: Part 2 - Reference Guide

In this document you will find all of the answers to some of the questions in the activity. Follow along with the activity and when you see this icon, stop and check your ideas here.

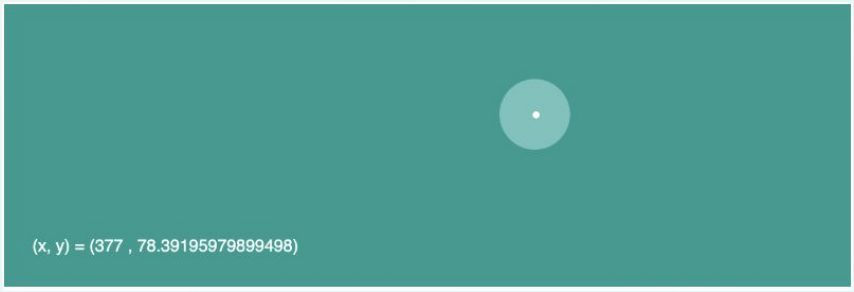
Step 1: Command a circle

There are lots of different ways you can write this! Here are two versions:

- ➔ Start on the left side. Count 4 spaces to the right, then 2 spaces down from the top. That is the center of the circle. Draw a circle around the center that has a radius of 1.
- ➔ Circle center at (4, 2). Circle has a diameter of 2.

Step 2: Practice using the coordinate system

Explore this **sketch** and use your mouse around to try to estimate the center, width, and height of the circle.



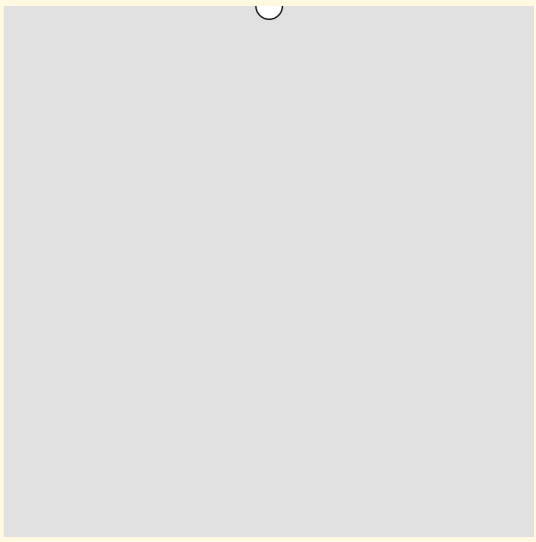
- ➔ Center: (376, 78)
- ➔ Width: 50
- ➔ Height: 50

Step 4: Draw the meteor

JAVASCRIPT

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

function draw() {
 background(225);

 // Draw the meteor
 ellipse(200, 0, 20, 20);
}
```

RESULT



Step 5: Fill the meteor with color

JAVASCRIPT

```
function setup() {
  createCanvas(400, 400);
}
function draw() {
  background(225);

  // Draw the meteor
  fill(0, 254, 202);
  ellipse(200, 0, 20, 20);
}
```

RESULT

Step 6: Change the outline

JAVASCRIPT

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

function draw() {
  background(225);
  noStroke();

  // Draw the meteor
  fill(0, 254, 202);
  ellipse(200, 0, 20, 20);
}
```

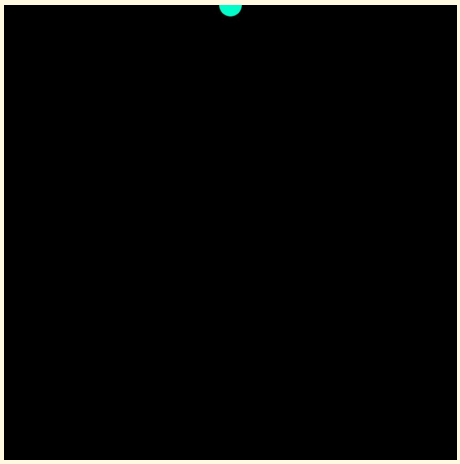
RESULT

Step 5: Add the background color

JAVASCRIPT

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

function draw() {
  background(0, 0, 0);
  noStroke();

  // Draw the meteor
  fill(0, 254, 202);
  ellipse(200, 0, 20, 20);
}
```

RESULT



Step 9: Check for Understanding

Describe the location, size, and color of the shape in the code below:

```
function setup() {
  createCanvas(100, 100);
}

function draw() {
  fill(0, 0, 255);
  ellipse(50, 50, 5, 5);
}
```

A red circle with a width and height of 5 in the center of the canvas. We know it's in the center because the x and y positions are half of the canvas' width and height.

Girls Who Code At Home

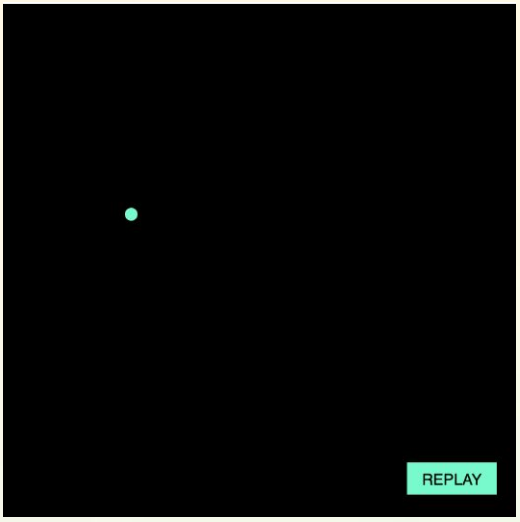
Meteor Catcher Game: Part 3

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.

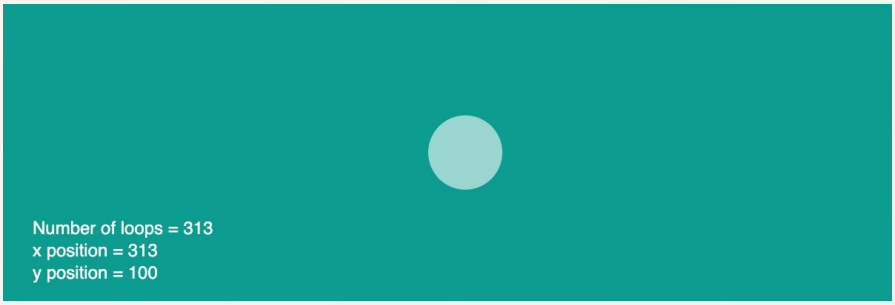
Don't forget to check your ideas with the Reference Guide on pg 2.

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:**

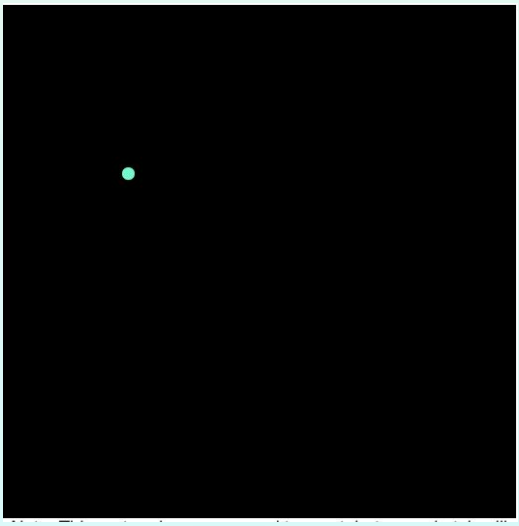
```
ellipse(xPosition, yPosition, 50, 50);
xPosition = xPosition + 1; // Can also be written as xPosition++
```

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.

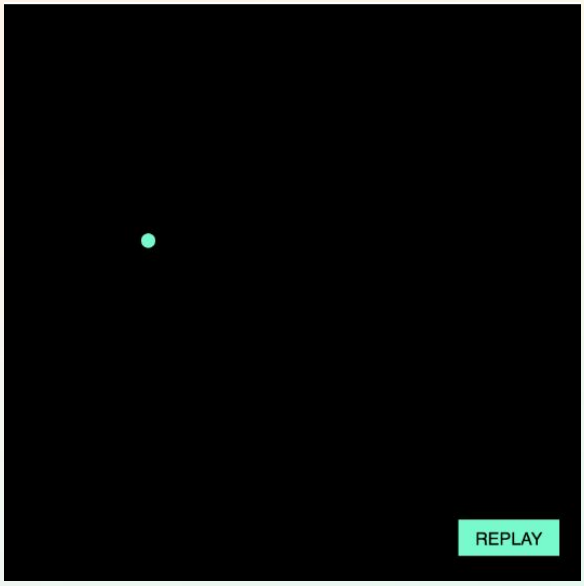
Don't forget to check your code with the Reference Guide on pg 3.

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?

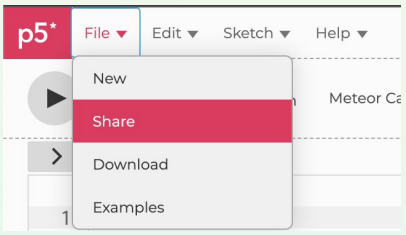
Don't forget to check your ideas with the Reference Guide on pg 3.

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

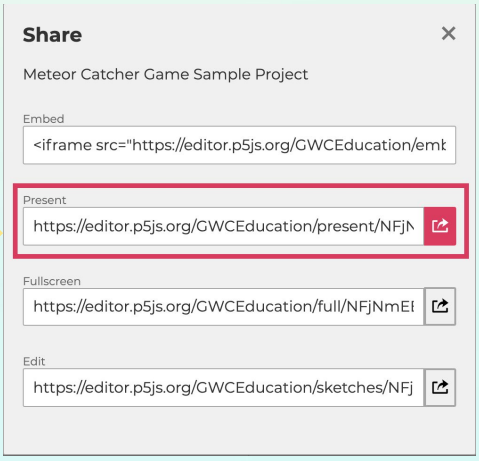
We would love to see your work and we know others would as well. Share your game with us! Don't forget to tag `@girlswhocode #codefromhome` and we might even feature you on our account!

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!

Girls Who Code At Home

Meteor Catcher Game: Part 3

Reference Guide

Meteor Catcher Game: Part 3 - Reference Guide

In this document you will find all of the answers to some of the questions in the activity. Follow along with the activity and when you see this icon, stop and check your ideas here.

Step 1: Using variables in p5.js

JAVASCRIPT

```
let meteorX = 200; //store the X position of the meteor
let meteorY = 0; //store the Y position of the meteor
let meteorDiameter = 20; //store diameter of the meteor

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

function draw() {
  background(0, 0, 0);
  noStroke();

  //Draw the meteor
  fill(0, 254, 202);
  ellipse(meteorX, meteorY, meteorDiameter, meteorDiameter);
 }
```

Step 2: Make observations about motion

Examine this sketch and use your observations to write a line of pseudocode to tell the program how to move the ball.

There are many different ways you could write this. Here are a couple:

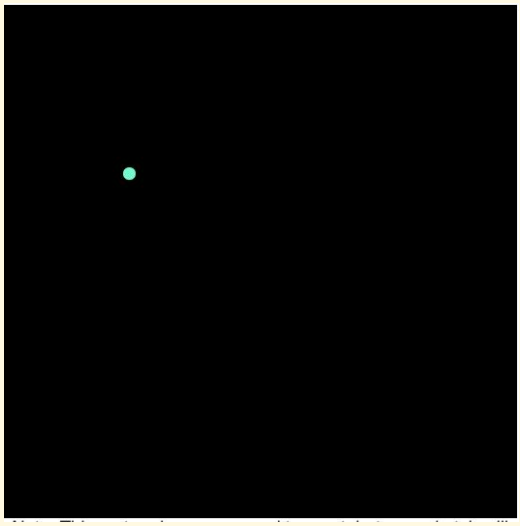
- ➔ Increase the value of x by a certain amount each time the program loops through.
- ➔ Add a small value to the `xPosition` variable in `draw()`.

Step 3: Add motion to your meteor

JAVASCRIPT

```
let meteorX = 200;
let meteorY = 0;
let meteorDiameter = 20;
let speed = 0.5; //store speed of the meteor

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

function draw() {
  background(0, 0, 0);
  noStroke();

  //Draw the meteor
  fill(0, 254, 202);
  ellipse(meteorX, meteorY, meteorDiameter,
      meteorDiameter);

  // Make the meteor fall
  meteorY = meteorY + speed;
}
```

RESULT



***Note**: In this example sketch the meteor is programmed to reset. Your sketch will not do this until Part 5.*

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?

Use the subtraction arithmetic operator **-** instead of the addition arithmetic operator **+**. This would cause the y value to decrease after each loop and change the vertical position of the meteor:

```
meteorY = meteorY - speed
```

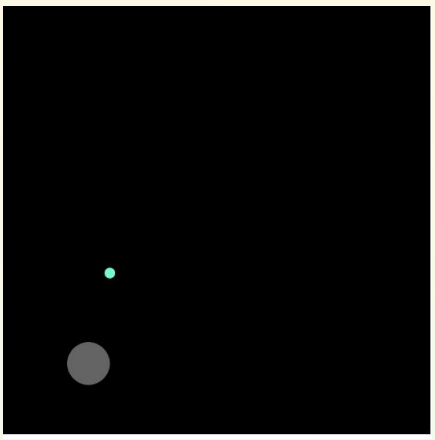
Girls Who Code At Home

Meteor Catcher Game: Part 4

Catch the Meteor

Activity Overview

At the end of Part 3, you learned how to create and use variables in p5.js to move the meteor across the screen at a specified speed. It's time to get interactive! In this part, you will learn how to add player input by programming a catcher that responds to mouse movement. This catcher is a translucent white ellipse that moves with the mouse. Click here to preview what you will learn by the end of the activity.



You should have already completed Part 1, Part 2, and Part 3 of the **Meteor Catcher Game Series** before embarking on this activity.

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 4 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 4 and Part 5 in the Meteor Catcher Game video tutorials!

Women in Tech Spotlight: Rebecca Cohen Palacios

Image Source: GamesIndustry.biz

If you have ever played the following video games – Elder Scrolls: Blades, Assassin's Creed Origins, Assassin's Creed Syndicate, or Shape Up (Kinect) – then you have viewed some of the work that Rebecca has contributed to create! More women are working in the gaming industry, and part of this is due to the work of Rebecca and the organization she has co-founded with Tanya Short, **Pixelles**.

Pixelles is a non-profit organization dedicated to encouraging more women in game development. Rebecca noticed that, in addition to video games still being perceived as a "for boys," there was also a large percentage of women in the gaming industry who left the field due to bias, a lack of support, and the glass ceiling. Pixelles tackles these issues by offering aspiring and mid-career women through free monthly workshops, mentorship, "make your first game" programs, a career accelerator, networking socials, and much more…

While co-directing Pixelles, Rebecca also currently works at **Behaviour Interactive** as a Senior UI Designer. Prior to her start in the gaming industry at Ubisoft, Rebecca spent 6 years as a Graphic and Web Developer.

Learn more about Rebecca and how her work with **Pixelles** seeks to close the gender gap in the gaming industry!

- **"Pixelles is Helping Mid-Career Mothers Stay in Games"**
- **"Top 7 Reasons Women Quit Game Development"**
- **"Montreal non-profit gives women a chance to break into male-dominated video game industry"**
- **"Empowerment Through Game Development": The Pixelles Game Incubator"**

Reflect

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

RESILIENCE

What are the struggles that women face in the gaming industry? How does Pixelles help women persevere in their careers?

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

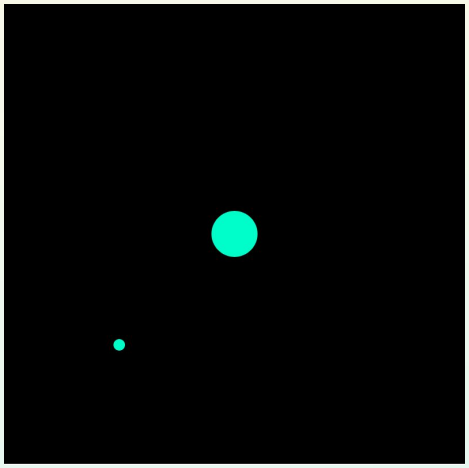
Step 1: Add the Catcher (5-10 mins)

Draw the Catcher (3-5 mins)

First, we need to draw the catcher to screen using the `ellipse()` function. Just as we did with the meteor, we will use variables to store the width and height of the catcher. We will use special variables for the x and y position in the next step, so we don't need to create variables to store these values.

- ❏ **Add a new variable above `setup()`to store the width and height of our catcher.** Since our catcher is a circle, we will use the same value for both. *We named our variable `catcherDiameter` but you can name it whatever you like. Just be sure you reference it correctly later in your code.*
- ❏ **Assign your catcher width and height variable a value of `40`.**
- ❏ **In the `draw()` function, draw the catcher using the `ellipse()` function.** Add this under your code for the meteor. Add a short comment to remind yourself that this is the catcher.
- ❏ Set the x and y parameters to **`200`**, then set the width and height parameters to the variable you created above.
- ❏ Run your code.

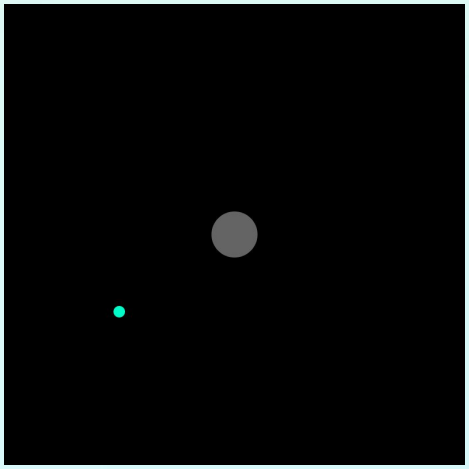
You should see a teal circle in the middle of the canvas:



Change the catcher color (3-5 mins)

We want our catcher to be white and semi-transparent so we can see our meteor through the catcher. This means we need to use our `fill()` function yet again! Earlier we mentioned that `fill()` will add color to all shapes below that command until you tell the program otherwise - sort of like cleaning a paintbrush and dipping it in another color.

- ❏ Create a new `fill()` function above the catcher ellipse. This will apply the new color to all shapes below it.
- ❏ Add the RGB values to make your catcher white.
- ❏ Add a fourth parameter to control the transparency and set this value to `100`. This optional parameter is called an alpha value. You can set to any value between 0 and 255 with 0 being completely transparent and 255 being opaque. You can try tinkering with a few different values to understand how it changes the display of the catcher.
- ❏ Run your code.



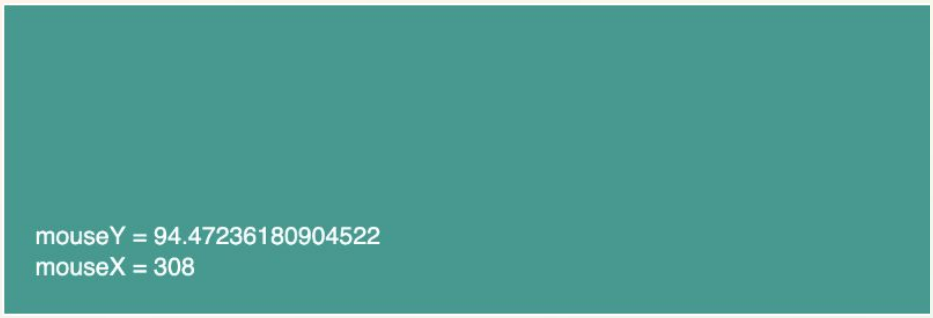
You should see the color of the catcher change to a semi-transparent white. In the image below, the catcher may appear gray. Since it is semi-transparent, some of the background color leaks through.

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

Step 2: Add player input (2-4 mins)

Right now, the catcher isn't doing much catching. We want a player to be able to control its movement using the mouse. This means we need to find a way to change the x and y values of the catcher so they match the x and y values of the mouse.

Luckily for us, there are two variables built into p5.js that do this for us! The variables `mouseX` and `mouseY` contain the position of the horizontal and vertical position of the mouse. Try moving your mouse in the **example sketch** to see how the values change.



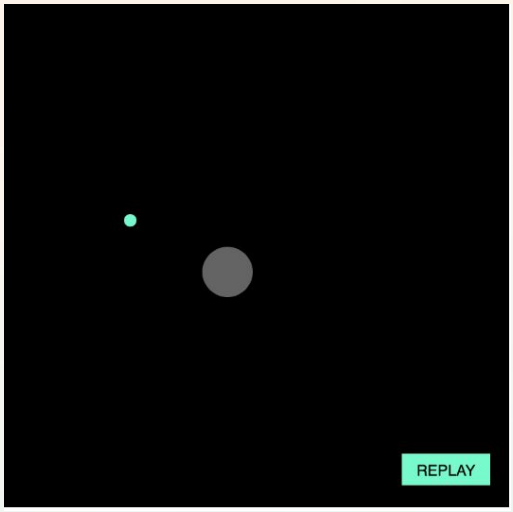
We can use `mouseX` and `mouseY` to attach the catcher to the mouse. This means that the x and y position of the mouse becomes the x and y position of the catcher. Let's update our catcher:

- ❏ Replace the current value of x in your catcher ellipse with `mouseX`.
- ❏ Replace the current value of y in your catcher ellipse with `mouseY`.

Step 3: Test your code (2-5 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 catcher that follows the movement of your mouse.
- ➔ The catcher should be white in color and semi-transparent.
- ➔ You should not have a Replay button.



Click here to run the example sketch.
***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 part with a conditional, but we're not there yet!*

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? Remember that JavaScript is case sensitive!
- ➔ Are your functions in the correct location and sequence? Remember that order matters in program flow!
- ➔ Do you have separate `fill()` functions above the `ellipse()` functions for both your meteor and catcher?
- ➔ If your catcher doesn't display, increase the alpha the value.
- ➔ Did you add `mouseX` and `mouseY` in the correct parameter location in your catcher ellipse?

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

Step 4: Check for Understanding

Describe how this line of code would change the behavior of our catcher:

```
ellipse(200, 200, mouseX, mouseY);
```

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

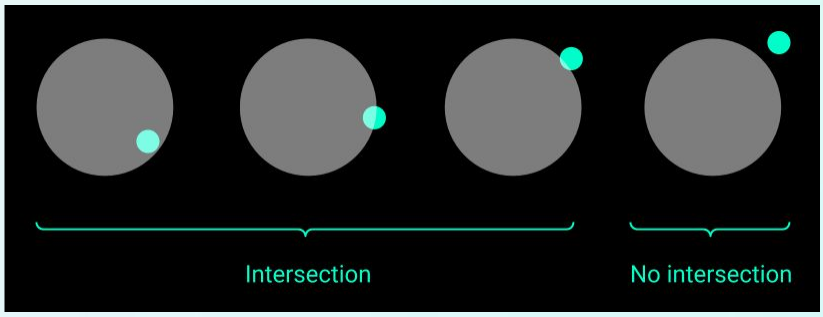
Step 5: Conceptualize "catching" (2 mins)

Image Source: Pixabay

In the last step, we figured out a way to have a player interact with our game. Now we want to figure out how to have different components in the game interact with each other. We want to determine when the catcher has "caught" the meteor and when the meteor has "hit" the bottom of the screen.

What actually happens when you catch something? Imagine throwing a ball or rolling it across the floor to a friend. When it touches their body and they contain it, we say they have caught the ball. We could also say that the ball has collided or intersected with their hand or foot (or paw).

We want to write code that will test to see if the meteor and catcher have intersected. This is also known as collision detection, a term for when two shapes touch. Here are some ways we might consider collision detection in our program:



Let's think about the information we have access to in our program that could help us: we know the x and y position of the meteor, the x and y position of the catcher, and the size of each one. Even though their positions are constantly changing, we can access those values through their x and y variables. Variables to the rescue once again!

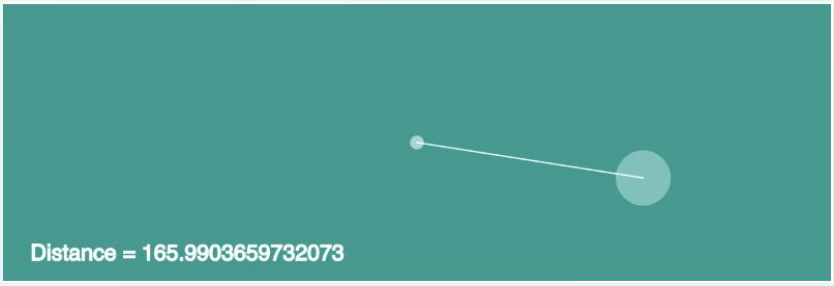
Step 6: Calculating distance (10-15 mins)

Meet the `dist()` function (3-5 mins)

We can use the information above to calculate how far the catcher is from the meteor at any given moment. p5 has a function that does these calculations for us: the dist() function. This function calculates the distance between two points. All we have to do is plug in the parameters! Here is the syntax:

JAVASCRIPT	DESCRIPTION
`dist(x1, y1, x2, y2);`	➔ `dist`: The function name. ➔ `()`: We use parentheses to tell our program that it needs to call the function. Sometimes we include parameters or inputs in the function inside our parentheses. ➔ `x1`: The x coordinate of the first point. ➔ `y1`: The y coordinate of the first point. ➔ `x2`: The x coordinate of the second point. ➔ `y2`: The y coordinate of the second point. ➔ `;`: All lines of code in p5.js must end with a semicolon.

Inspect the demo sketch. The text displays the value returned from the `dist()` function between the center circle and the mouse circle. Try moving the mouse circle around until it intersects with the center circle and watch as the distance values change.



Think about it: What range of distance values do you think represent intersection? Later on we will ask ourselves this question, when completing our project sketch to write a conditional so we can track “catching.”

Add `dist()` to your code (5-8 mins)

Let's add the `dist()` function to our project sketch. First we will create a variable to store the value returned from the function (i.e. the distance between the center of the catcher, and the center of the meteor). Next we will add the function. Finally we will use a new function, the `print()` function, to track the distance value as it changes.

- ❏ **Add a new variable above `setup()` to store the distance value.** We named our variable distance but you can name it whatever you like. Just be sure you reference it correctly later in your code. You do not need to assign it a value.
- ❏ **Call the `dist()` function under the catcher code.** If it's helpful, you can add a short comment to remind yourself what this line of code does.
- ❏ Set the parameters for x1 and y1 to the variables that store the x and y position of your meteor.
- ❏ Set the parameters for x2 and y2 to `mouseX` and `mouseY`.
- ❏ Store the results of the `dist()` function in your distance variable.

Don't forget to check your code with the Reference Guide on pg 4.

Print the `dist()` value to the console (3-5 mins)

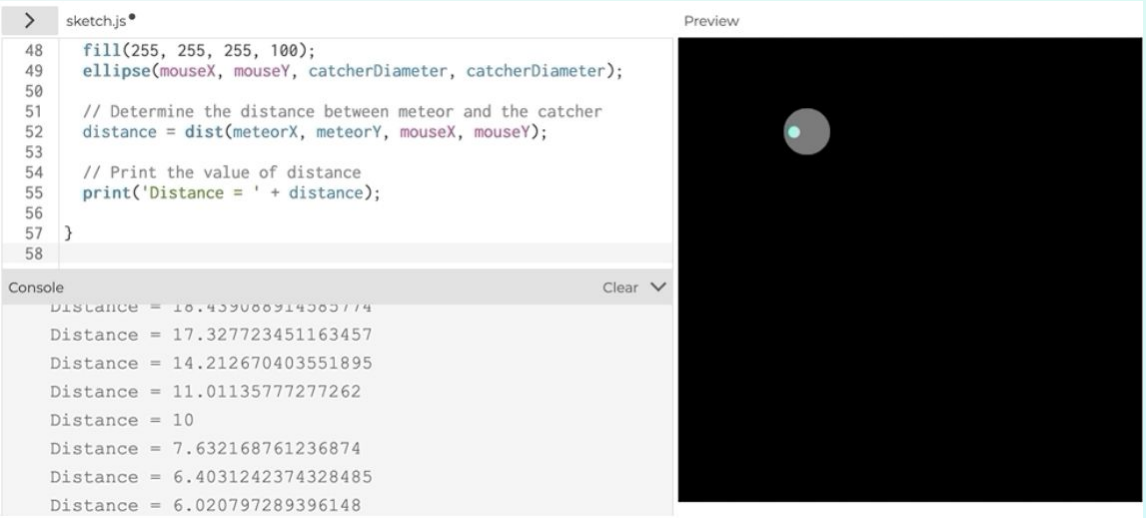
Now our program is constantly calculating the distance between the center of the meteor and the center of the catcher! But how can we check those values? The easiest way to do this is the `print()` function. It prints alphanumerical values, like words and numbers, to the console below the editor. Here is the syntax:

JAVASCRIPT	DESCRIPTION
`print('Distance = ' + distance);`	➔ `print`: The function name that prints messages to the console. ➔ `()`: We use parentheses to tell our program that it needs to call the function. Sometimes we include parameters or inputs in the function inside our parentheses. ➔ `''`: Single or double quotes tell the program we are printing strings or text. You can use either type of quote, just be consistent. ➔ `+`: Joins elements to print them together. ➔ `distance`: Prints the current value stored in the distance variable. ➔ `;`: All lines of code in p5.js must end with a semicolon.

Let's write this into our sketch:

- ❏ Add this line of code under the `dist()` function to print out the value of the distance variable: `print('Distance = ' + distance);`
- ❏ Run the sketch.

The text `Distance =` and a series of changing values should print to the console.

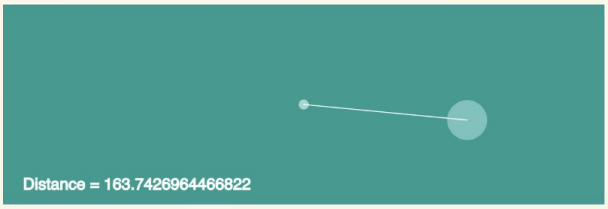


Step 7: Determining intersection (5 mins)

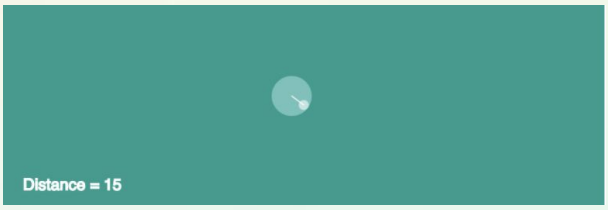
Define intersection using the distance value (2-3 mins)

We know that we want our program to behave a certain way if a catcher and meteor intersect. To do this, we need a numerical value that represents intersection. We can use the value stored in `distance` (remember that you may have used a different variable name) to determine when the catcher and meteor intersect.

Consider the [distance demo sketch] from earlier again: What value is distance equal to when the catcher covers or mostly covers the "meteor"?



After exploring the demo sketch, we can say that intersection occurs **if the distance is less than 15**. Now we can start making some decisions in our program using conditionals.



Review conditionals (2 mins)

Now we can tell our sketch what to do if an intersection occurs. For this we need a conditional statement. Let's do a quick refresher: **conditional statements** allow us to control the flow of our program. They use `if` statements to check if a condition is `true` or `false`. If a condition is true, then the computer will run the code inside the `if` statement.

Below is an example to illustrate the syntax for a conditional statement in JavaScript (remember that p5.js is a library for JavaScript). It tells our program to draw a circle in the top left corner of the canvas if the x position of the mouse is greater than 100.

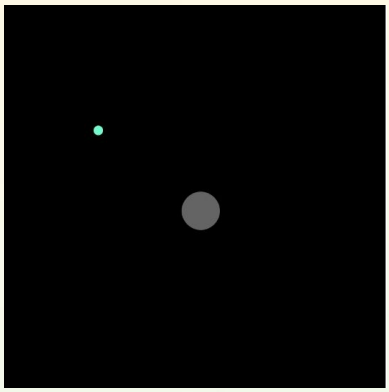
JAVASCRIPT	DESCRIPTION

`if(mouseX > 100){`
`  ellipse(0,0,50,50);`
`}`
	➔ `if`: Keyword to tell the program this is a conditional.
	➔ `()`: We use parentheses to tell our program that anything inside is part of the condition it will evaluate.
	➔ `mouseX > 100`: The conditional expression that the sketch will test to determine if it is true or false. You can use comparison operators like > (greater than) or <= (less than or equal to) or logical operators such as || (or) or && (and) to set up your statement.
	➔ `{}`: Curly brackets tell our program which lines of code to run if the condition is true.
	➔ `ellipse(0,0,50,50)`: The statement that will execute if the condition evaluates as true. You can also include other if statements here.
	➔ `;`: All lines of code in p5.js must end with a semicolon.

Step 8: Set up catcher conditional (5-10 mins)

Plan the catcher conditional (2-4 mins)

The first conditional we create will be for the catcher. We need to write an `if` statement that will test to see if the meteor and catcher intersect. Before writing any code, let's describe what we want to happen. Play the game again, then write pseudocode for the conditional. Try to be as specific as possible. You can use the pseudocode you wrote in the **Planning section of Part 1** to get started. Check out the **example sketch** before getting started.



Hint: You can use the variables for distance and the y position of the meteor that you created in prior steps. You will also need a number that represents intersection.

Don't forget to check your code with the Reference Guide on pg 5.

Add the catcher conditional (3-5 mins)

Next, translate your pseudocode to actual code:

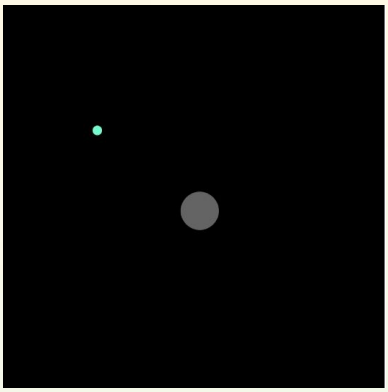
- ❑ Add an `if` statement that checks the value of the distance variable.
- ❑ **Add a line of code that updates the y position of your meteor.** Make sure it is inside the curly brackets of the conditional!
- ❑ Run your code.
- ❑ Try to make your meteor and catcher intersect. When they intersect, the current meteor should disappear and a "new" meteor should appear at the top of the screen.

If this does not happen, check the distance values printing to the console. When the catcher and meteor intersect, are the values in the range specified in the conditional statement? If not, you might need to adjust the number in your expression.

The `print()` function is very helpful for debugging! If at any point you are unsure of the value of a variable in your sketch, put a `print()` statement under it to double check.

Don't forget to check your code with the Reference Guide on pg 5.

Step 9: Set up screen bottom conditional (5-10 mins)



Plan the screen bottom conditional (2-3 mins)

Now let's write the conditional to test if the meteor has intersected with the bottom of the screen. If it has intersected with the bottom of the canvas, we want the program to redraw it to the top of the canvas. If you want a quick way to reference the height or width of the canvas in your code, you can use the keywords `height` and `width` in place of the numerical value. Check out the **example sketch** before getting started.

Remember: the height of our canvas is `400` pixels.

Write pseudocode for this conditional. Just as before, try to be as specific as possible. You can use the pseudocode you wrote in the **previous step** or the **Planning section of Part 1** to get started.

Don't forget to check your code with the Reference Guide on pg 5.

Add the screen bottom conditional (3-5 mins)

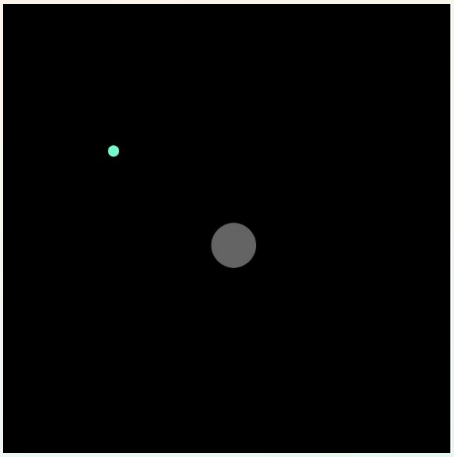
Now let's translate your pseudocode to actual code:

- ❏ Write the `if` statement and conditional expression.
- ❏ Add the statement that will run if the conditional is true. Make sure it is inside the curly brackets of the conditional.
- ❏ Run your code.
- ❏ Wait for the meteor to intersect with the bottom of your canvas. When they intersect, the current meteor should disappear and a "new" meteor should appear at the top of the screen.

Step 10: Test your code (5-10 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. It should:

- ➔ Print the distance value to the console.
- ➔ Redraw the meteor to the top of the canvas when the catcher and meteor intersect.
- ➔ Redraw the meteor to the top of the canvas when the meteor and bottom of the canvas intersect.



Click here to run the example sketch.

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!
- ➔ Are the parameters in your `dist()` function correct?
- ➔ Print values to the console using `print()` then check any `if` statements.

Don't forget to check your code with the Reference Guide on pg 6.

Step 11: Check for Understanding

Let's say you want to "catch" the meteor when the catcher barely touches the outer edge of the meteor. Would you increase or decrease the value in the expression of your first conditional statement?

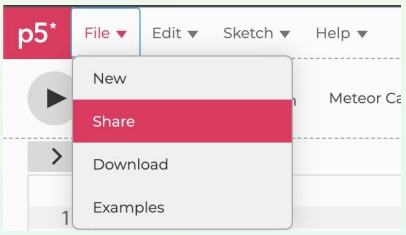
Don't forget to check your ideas with the Reference Guide on pg 7.

Step 12: Share Your Girls Who Code at Home Project! (5 mins)

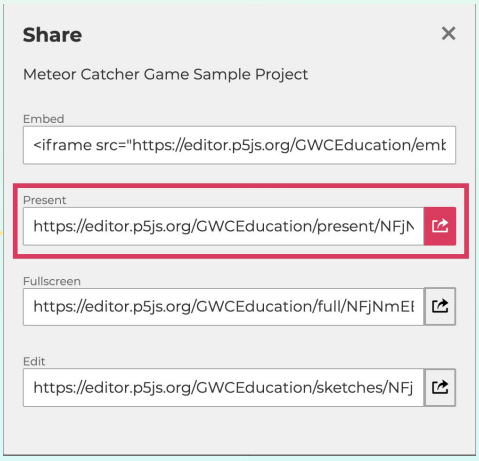
We would love to see your work and we know others would as well. Share your game with us! Don't forget to tag `@girlswhocode #codefromhome` and we might even feature you on our account!

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!

Girls Who Code At Home

Meteor Catcher Game: Part 4

Reference Guide

Meteor Catcher Game: Part 4 - Reference Guide

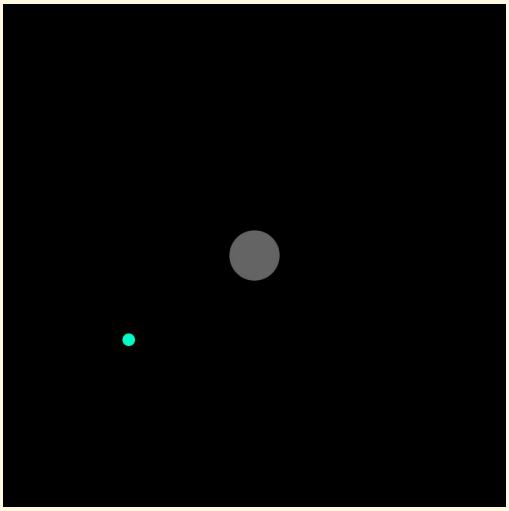
In this document you will find all of the answers to some of the questions in the activity. Follow along with the activity and when you see this icon, stop and check your ideas here.

Step 1: Add the Catcher

JAVASCRIPT

```
let meteorX = 200;
let meteorY = 0;
let meteorDiameter = 20;

let catcherDiameter = 40;

let speed = 0.5;

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

function draw() {
  background(0, 0, 0);
  noStroke();

  //Draw the meteor
  fill(0, 254, 202);
  ellipse(meteorX, meteorY, meteorDiameter,
       meteorDiameter);

  // Make the meteor fall
  meteorY = meteorY + speed;

  //Draw the catcher to follow the mouse
  fill(255, 255, 255, 100);
  ellipse(200, 200, catcherDiameter, catcherDiameter);
}
```

RESULT

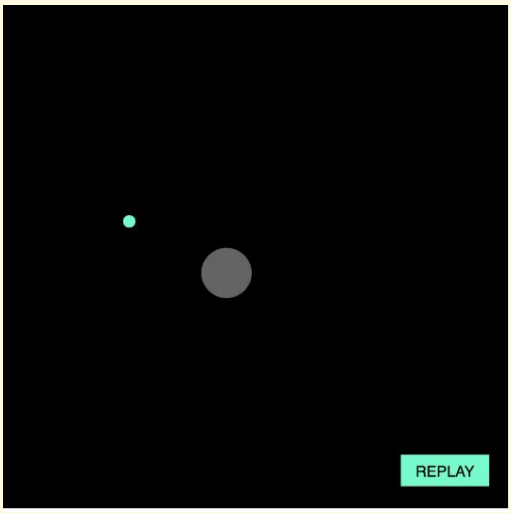


Step 3: Test Your Code

JAVASCRIPT

```
let meteorX = 200;
let meteorY = 0;
let meteorDiameter = 20;

let catcherDiameter = 40; // Store diameter of catcher

let speed = 0.5;

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

function draw() {
  background(0, 0, 0);
  noStroke();

  //Draw the meteor
  fill(0, 254, 202);
  ellipse(meteorX, meteorY, meteorDiameter,
       meteorDiameter);

  // Make the meteor fall
  meteorY = meteorY + speed;

  //Draw the catcher to follow the mouse
  fill(255, 255, 255, 100);
  ellipse(mouseX, mouseY, catcherDiameter,
       catcherDiameter);
}
```

RESULT



***Note**: In this [example sketch] 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 part with a conditional, but we're not there yet!*

Step 4: Check for Understanding

Describe how this line of code would change the behavior of our catcher:

```
ellipse(200, 200, mouseX, mouseY);
```

Instead of changing the position of the ellipse, the width and height of the ellipse would change based on the mouse position. To see what this looks like in action, try changing the line of code. Just be sure to change it back to the original code with the right variable:

```
ellipse(mouseX, mouseY, catcherDiameter, catcherDiameter);
```

Step 6: Calculating Distance

JAVASCRIPT

```
let meteorX = 200; // Store the X position of the meteor
let meteorY = 0; // Store the Y position of the meteor

let meteorDiameter = 20; // Store diameter of the meteor
let catcherDiameter = 40; // Store diameter of catcher

let speed = 0.5; // Store speed of the meteor

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

function draw() {
  background(0, 0, 0);
  noStroke();

  //Draw the meteor
  fill(0, 254, 202);
  ellipse(meteorX, meteorY, meteorDiameter, meteorDiameter);

  // Make the meteor fall
  meteorY = meteorY + speed;

  //Draw the catcher to follow the mouse
  fill(255, 255, 255, 100);
  ellipse(mouseX, mouseY, catcherDiameter, catcherDiameter);

  // Determine the distance between meteor and the catcher
  distance = dist(meteorX, meteorY, mouseX, mouseY);
}
```

Step 8: Set up catcher conditional

Plan the catcher conditional

There are different ways you can write this pseudocode. Here are a couple:

- If the `distance` value is less than 15 pixels, set the `meteorY` value to 0 at the top of the canvas.
- If the `distance` value is less than 15 pixels, reassign the variable for the y position of the meteor to 0.

Add catcher conditional

JAVASCRIPT

```
// Determine the distance between meteor and the catcher
  distance = dist(meteorX, meteorY, mouseX, mouseY);


// Test to see if meteor and catcher have intersected
if (distance < 15) {
    // Redraw  meteor to top of screen at a random location on x-axis
    meteorY = 0;
}
```

Step 9: Set up screen bottom conditional

There are different ways you can write this pseudocode. Here are a couple:

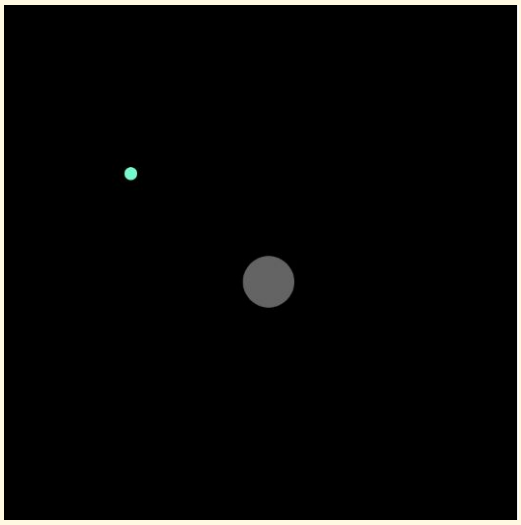
- If the `meteorY` value is greater than 400 pixels, set the `meteorY` value to 0.
- If the `meteorY` value is greater than the height of the canvas, assign the variable for the y position of the meteor to 0.

Step 10: Test Your Code

JAVASCRIPT

```
let meteorX = 200; // Store the X position of the meteor
let meteorY = 0; // Store the Y position of the meteor

let meteorDiameter = 20; // Store diameter of the meteor
let catcherDiameter = 40; // Store diameter of catcher

let speed = 0.5; // Store speed of the meteor
let distance; // Store distance between meteor and catcher

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

function draw() {
  background(0, 0, 0);
  noStroke();

  // Draw the meteor
  fill(0, 254, 202);
  ellipse(meteorX, meteorY, meteorDiameter,
meteorDiameter);

  // Make the meteor fall
  meteorY = meteorY + speed;

  // Draw the catcher
  fill(255, 255, 255, 100);
  ellipse(mouseX, mouseY, catcherDiameter,
catcherDiameter);

  // Determine the distance between meteor and the catcher
  distance = dist(meteorX, meteorY, mouseX, mouseY);

  // Print the value of distance
  print('Distance = ' + distance);

  // Test to see if meteor and catcher have intersected
  if (distance < 15) {
    // Redraw  meteor to top of screen at a random location
on x-axis
    meteorY = 0;
  }

  // Test to see if meteor has intersected with screen bottom
  if(meteorY > height) {
    meteorY = 0;
  }
}
```

RESULT



Click here to run the example sketch.

Step 11: Check for Understanding

Let's say you want to "catch" the meteor when the catcher barely touches the outer edge of the meteor. Would you increase or decrease the value in the expression of your first conditional statement?

You would *increase* the value. This would allow the statement to still evaluate as true with more distance between the center of the catcher and center of the meteor.

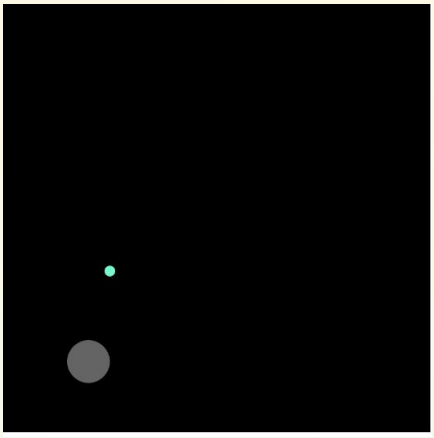
Girls Who Code At Home

Meteor Catcher Game: Part 5

Add a Randomizer

Activity Overview

In this final part, we will explore one way to make your game more challenging - and more fun! You will learn how to use the `random()` function to add complexity to the behavior of your meteor. Finally, customize your project by trying out one or more of our extensions. Click here to preview what you will learn by the end of the activity.



You should have already completed Part 1, Part 2, Part 3, and Part 4 of the **Meteor Catcher Game Series** before embarking on this activity.

Learning Goals

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

- ❏ explain how the `random()` function can be used to increase or decrease a game's challenge.

Materials

- ➔ p5.js Online Editor
- ➔ Meteor Catcher Game Sample Project
- ➔ Meteor Catcher Game Part 5 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 6 in the Meteor Catcher Game video tutorials!

Women in Tech Spotlight: Lisette Titre-Montgomery

Image Source: Lisette Titre-Montgomery

When you jam out to moves on Dance Central or create characters on The Sims, do you ever wonder about the technology that took you there? Lisette Titre-Montgomery works as an engineer and designer behind the most popular games on the market. With over seventeen years of experience in the game industry and thirteen shipped titles, she has successfully bridged the gap between the video gamer and the developer.

Lisette also dedicates her time to bringing more diversity into the gaming industry through mentoring and inclusive hiring initiatives with The White House. She continues to be an advocate for game based curriculums in K-12 education to engage students in STEAM education and careers with a program called **Gameheads**, based in Oakland, CA.

Watch this **video** to learn more about Lisette and some of the challenges she faced working on various games and how she overcame them.

Want to learn more about Lisette and her work?

- ➔ Check out her **personal website** to learn more about some of her various works and projects.
- ➔ Read this **article** about Lisette's background and how she came to the gaming industry.
- ➔ Look up **Lisette's profile** to see some highlights of her career and her advice for students interested in starting in the gaming industry!

Reflect

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

BRAVERY

Lisette had never worked in the gaming industry until her first job. What steps did she take to succeed in her role?

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

Step 1: Reflect on your game's challenge (2-4 mins)

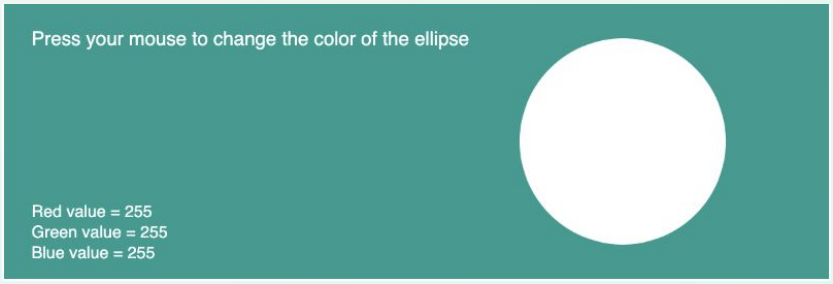
Right now, we have all the critical parts of our game covered: the components, core mechanic, game and space. But the game isn't very fun yet, mostly because it is not challenging enough. Finding the right difficulty level is key for game designers. One tool you can use to increase difficulty (and there are many) is to include an element of randomness. This prevents the player from anticipating what will come next by surprising them with new information that they must adapt to.

Where can we add randomness to the game to make it more challenging? Take one minute to think about it, then compare your ideas to the ones below.

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

Step 2: Explore the `random()` function (5 mins)

We can use the `random()` function to add randomness to our code. This function returns, or outputs, a random floating point number between a specified range of values. Floating point means the number might contain a decimal. This allows for a much larger spectrum of numbers. Explore this **example sketch** where each mouse click inside the canvas generates a random value for the red, green, and blue values in the `fill()` function.



Let's take a look at the syntax of the `random()` function.

JAVASCRIPT	DESCRIPTION
`random(min, max);`	➔ `random`: The function name. ➔ `()`: We use parentheses to tell our program that it needs to call the function. Sometimes we include parameters or inputs in the function inside our parentheses. ➔ `min`: The lower end of the random range you want that includes this number as an option. ➔ `max`: The higher end of the random range you want that excludes this number as an option. ➔ `;`: All lines of code in JavaScript must end with a semicolon.

Step 3: Update your pseudocode (2-4 mins)

Before we add in our new lines of code, let's brainstorm what we want to happen in human language first. Write pseudocode for each `random()` function. Try to be as specific as possible and use the value ranges specified. When you are done, check your answer below:

- ➔ **Meteor starting location.** Value range: 0 to 400
- ➔ **Meteor speed.** Value range: 0.5 to 4
- ➔ **Meteor size.** Value range: 10 to 30

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

Step 4: Add your code (5-8 mins)

Now we need to translate your pseudocode into actual code. All of the code we write will go inside both conditional statements underneath the existing statement. Right now, we are only redrawing the meteor to the top of the screen if it intersects with the catcher or bottom wall. Adding these statements in the conditional will randomize the location, speed, and size of the new meteor.

Locate the first conditional statement. Add these new lines of code under `meteorY = 0;`:

- ❑ Write the line of code that randomizes the meteor starting location to a value between 0 and 400.
- ❑ Write the line of code that randomizes the meteor's speed to a value between 0.5 and 4.
- ❑ Write the line of code that randomizes the meteor's size to a value between 10 and 30.

Locate the second conditional statement.

- ❑ Copy the statements you just wrote for the first conditional, and copy them under meteorY = 0; in the second conditional.

You may have noticed the lines of code inside the conditionals look the same! That's true! You could combine these conditionals with the **logical operator** "or" - ||. You could also combine them by nesting them together. We opted to separate them to make the extensions a bit less confusing. If you would like to combine these conditionals, you are welcome to!

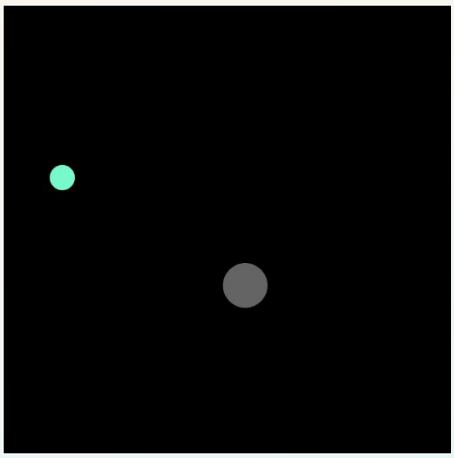
Step 5: Test your code (5-10 mins)

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.

Run your code and test the following:

- ❏ Try to make your meteor and catcher intersect. When they intersect, the current meteor should disappear and a "new" meteor should appear at a random x position, with a new size, and new speed.

- ❏ Wait for the meteor to intersect with the bottom of your canvas. When they intersect, the current meteor should disappear and a "new" meteor should appear at a random x position, with a new size, and new speed.



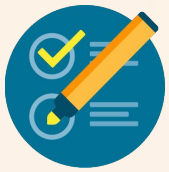
Click **here** to run the example sketch.

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?
- ➔ Are your functions in the correct location? Remember that order matters in program flow!
- ➔ Are the parameters in your `random()` function correct?

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

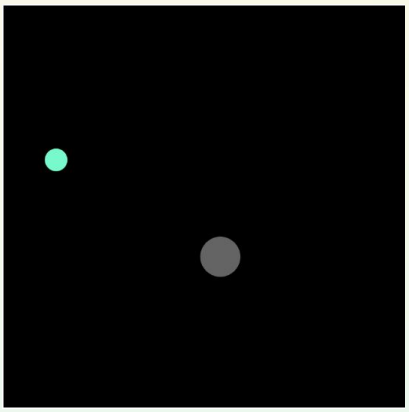
Step 7: Test your code and receive feedback (5-10 mins)



You have unlocked a major milestone: you finished building the core part of your project! Before moving forward, we will test your code to make sure your game is working properly. This is also a great opportunity to get feedback on your game from friends and family. Share your project and this checklist with your tester!

Checklist

Run your sketch, then use the checklist below to test your code:



Click here to run the example sketch.

- ❏ You have a 400 by 400 canvas. The background is filled with the color of your choice.
- ❏ Your game begins with a meteor (i.e. an ellipse) falling slowly from the center of the top of the canvas. The meteor has no outline and is filled with a color of your choice.
- ❏ There is a catcher (i.e. an ellipse) that is white in color and semi-transparent. The center of the catcher follows the movement of your mouse.
- ❏ Your program prints the distance between the meteor and catcher to the console.
- ❏ If the meteor and the catcher intersect, the game updates to draw a new meteor.
- ❏ If the meteor and the bottom of the canvas intersect, the game updates to draw a new meteor.
- ❏ New meteors begin at the top of the canvas at a random location on the x axis with a random speed and random size.

Debugging Tips

If your sketch isn't working the way you want it to, check the console first to see if there is an error. You can try to fix your code or try googling to find the answer. You can also review this resource from p5.js.

- ➔ 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!
- ➔ Are the parameter values in your functions correct? Specifically the `dist()` and `random()` functions?
- ➔ Print values to the console using `print()` then check any `if` statements.

You did it! You finished building the foundation for your project. In the next part of this activity you will have the chance to personalize your project further with some optional extensions.

Step 8: Extensions (5-45 mins)

Extension 1: Change the story with graphics (15-45 mins)

Right now, the game is about space and meteors - but it doesn't have to be! You can alter the narrative of the game by adding new graphics or images. In game design, this is referred to as skinning. The skin of a game is the appearance of the components and background of the game. You can reskin the game without changing the mechanics, but it can vastly change the meaning of the game and how the player feels about it.

Click here to see an example sketch of this extension

In this extension, try replacing one or more shapes and the background with images. Here are the basic steps you need to complete this extension:

- ❏ Find `.png` image files with a transparent background. You may need to resize them in Google Draw, then export them as a `.png` image file.
- ❏ Upload your `.png` files to your sketch.
- ❏ Declare a variable to store the image file.
- ❏ Initialize each image variable in `setup()` with the `loadImage()` function.
- ❏ Draw the images to the screen using the `image()` function in `draw()`. Be sure to use the correct x and y position variables.
- ❏ Pass the background image variable through the `background()` function.

Extension Resources

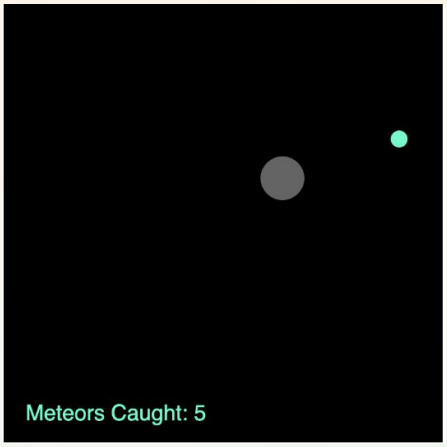
Below are some helpful resources we used to create this Extension. These will help you get started, but remember that there are lots more resources only a search engine away!

- ➔ p5.js Web Editor: Uploading Media Files - p5.js Tutorial by The Coding Train (start around 0:59)
- ➔ Load and Display an Image Example from p5.js. Note: Disregard the text that says To run this example locally, you will need an image file, and a running local server. This does not apply to the online editor.
- ➔ loadImage()
- ➔ image()
- ➔ imageMode()
- ➔ background()

Here is a link to our solution code for this extension. We recommend trying it yourself first and using this resource when you get really stuck or want to check your work.

Extension 2: Track the score (5-15 mins)

At some point in the tutorial, you may have asked yourself, “But what about points?!” Keeping track of a player’s score isn’t necessary to make it a game, but it is one way to give players immediate feedback about their gameplay. You can create a variable to keep track of the score, then add or increment the value of that variable each time the player earns points. You could also have the player start with a score, then subtract or decrement a value when they take a specific action.



Click here to see an example sketch of this extension

In this extension, try creating your own simple scoring system, then drawing the score to the screen as text. Here are the basic steps you need to complete this extension:

- ❏ **Create a variable to keep track of the score and set it to zero.** *Hint: Where do you think you'd want to place this variable if you wanted to access it anywhere in your sketch?*
- ❏ **Increment this variable if the meteor and catcher intersect.** *Hint: Where in your code do you check to see if the meteor and catcher intersect?*
- ❏ **Draw the value of the variable to the screen as text.**

Extension Resources

Below are some helpful resources we used to create this Extension. These will help you get started, but remember that there are lots more resources only a search engine away!

- ➔ Increment Decrement Example from p5.js
- ➔ Words Example from p5.js
- ➔ textSize()
- ➔ text(): *You can combine strings (i.e. a series of text characters) with variables in the* `text()` *function.*
- ➔ textAlign()
- ➔ string

Here is a link to our solution code for this extension. We recommend trying it yourself first and using this resource when you get really stuck or want to check your work.

Extension 3: Add a win state (10-20 mins)

Whether individual or collaborative, games often have a win state. Win states occur when players have successfully overcome the main challenge of the game, like the highest number of goals or most gold coins in 30 seconds or the completion a space mission. There are many different ways you can win a game.

Click **here** to see an example sketch of this extension

In this extension, create a system to test and alert a player if they have won, then reset the game so they can play again. Before you get started, think about what "winning" means in your game. *For example, is it the amount of times played, the number of points, etc.* In the extension example **here**, we use the number of points scored to determine if a player has won. If they have won and press the mouse, the game restarts. This extension requires you to create a function of your very own, so if you want to learn more about defining functions this is a good opportunity!

Here are the basic steps you need to complete this extension:

- ❏ Create a function to restart the game by resetting the position, score, and speed variables to their original values.
- ❏ Create a conditional to test if the score is equal to the number of points needed to win.
- ❏ Write a message to the screen alerting the player they have won and how to restart the game.
- ❏ Create a second conditional inside the first to test if the mouse is pressed. Call the restart function inside of it.

Extension Resources

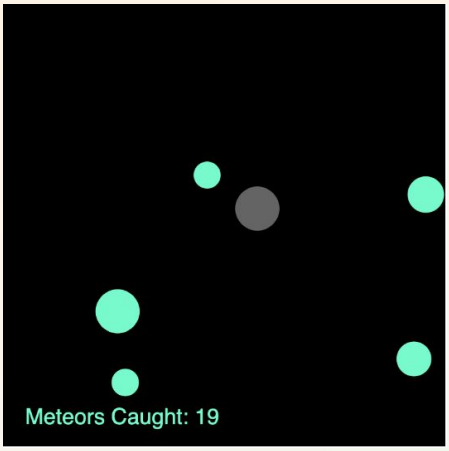
Below are some helpful resources we used to create this Extension. These will help you get started, but remember that there are lots more resources only a search engine away!

- ➔ **Function Basics - p5.js Tutorial** video from The Coding Train
- ➔ **function**
- ➔ **mouseIsPressed**
- ➔ **Increment Decrement Example** from p5.js
- ➔ **Words Example** from p5.js
- ➔ **textSize()**
- ➔ **text()**: *You can combine* ***strings*** *(i.e. a series of text characters) with variables in the* `text()` *function.*
- ➔ **textAlign()**
- ➔ **string**

Here is a link to our **solution code for this extension**. We recommend trying it yourself first and using this resource when you get really stuck or want to check your work.

Extension 4: Add more meteors (25-40 mins)

In the last part of the Building phase, we explored ways we could make the game more challenging by randomizing properties of the meteor. Another way to make the game more challenging is to make more meteors!



Click **here** to see an example sketch of this extension

***Note**: We also included a score tracker so you can see when a meteor has intersected, but you are not required to do that for this extension.*

There are a couple of ways you can approach this.

- ➔ **Approach #1:** Create new variables for a second meteor. Then duplicate all the code you wrote for the first meteor, but updating it with variables for the second meteor. If you think this sounds tedious, you are correct! It will work, but your code will be long and cumbersome.
- ➔ **Approach #2:** Use **arrays** and **for** loops to add more meteors! Arrays allow you to store an ordered list of elements in a single variable. That is, instead of just storing one value for a meteor's diameter, you can store 10, or 15, or 20 or 300 - and you can access them all using the index number. For loops allow you to loop through a section of code multiple times based on a condition. You could say that for loops and arrays are BFFs because you can use a `for` loop to iterate or cycle through an `array`. See the **Extension Resources** below to learn more.

In this extension, try using arrays and for loops to make four more meteors for a total of five. Arrays will allow you to store the meteorX variables you need for all the meteors in one place. For loops will allow you to cycle through each of those variables and use it or perform an action on it.

Here are the basic steps you need to complete this extension:

- ❑ Declare `arrays` to store the X position of the meteor, Y position of the meteor (these values should all be 0), the meteor diameter, the distance, and the speed.
- ❑ Create `for` loops in the `setup()` that will pre-populate the following arrays with initial random values: X position, meteor diameter, and speed.
- ❑ Create a `for` loop that draws the meteors to the screen.
- ❑ Create a `for` loop that cycles through the meteors to make them fall at different speeds.
- ❑ Draw the catcher.
- ❑ Create a `for` loop that cycles through meteors to determine the distance between each meteor and the catcher.
- ❑ **Create a `for` loop that cycles through meteors to test if one of the meteors has intersected with the catcher.** If it has intersected, redraw that meteor to the top of the screen at a random X position, and set a new random speed for that meteor. *Tip: You will need to add a conditional statement inside the for loop.*
- ❑ **Create a `for` loop that cycles through meteors to test if one of the meteors has intersected with the bottom of the screen.** If it has intersected, redraw that meteor to the top of the screen at a random X position, and set a new random speed for that meteor. *Tip: You will need to add a conditional statement inside the for loop.*

Extension Resources

Below are some helpful resources we used to create this Extension. These will help you get started, but remember that there are lots more resources only a search engine away!

- ➔ p5.js Tutorial videos from The Coding Train
  ***Note**: some of these videos were created before the online editor was built, but it will work the same*!
  - ◆ What is an Array?
  - ◆ Arrays and Loops
  - ◆ while and for loops
- ➔ Program Flow tutorial
- ➔ Iteration tutorial
- ➔ for If you are having trouble getting started, try using the first approach and create a second meteor by duplicating code from the first meteor. Any time you see a double (e.g. **`meteorY_1`** and **`meteorY_2`** or **`speed_1`** and **`speed_2`**) chances are you will need an array and a for loop to cycle through it.

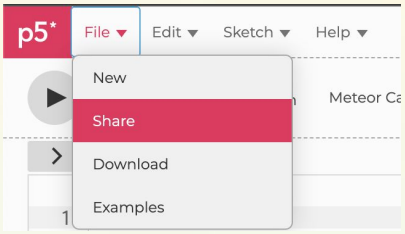
Here is a link to our solution code for this extension. We recommend trying it yourself first and using this resource when you get really stuck or want to check your work.

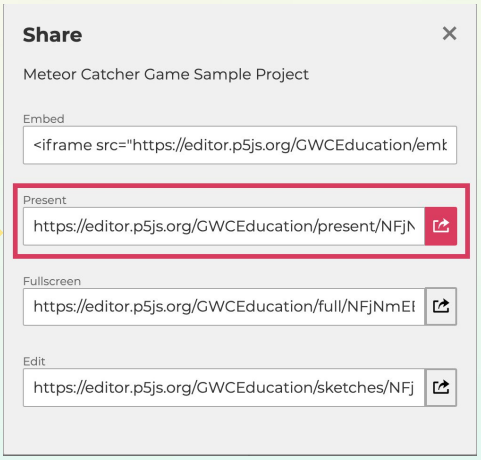
Step 9: Share Your Girls Who Code at Home Project! (5 mins)

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!

Girls Who Code At Home

Meteor Catcher Game: Part 5

Reference Guide

Meteor Catcher Game: Part 5 - Reference Guide

In this document you will find all of the answers to some of the questions in the activity. Follow along with the activity and when you see this icon, stop and check your ideas here.

Step 1: Reflect on your game's challenge

Where can we add randomness to the game to make it more challenging?

There are many places where we could add randomness. Here we will focus on the following properties of the meteor:

- ➔ starting location
- ➔ speed
- ➔ size

Make a note of your other ideas - you may want to include one when you start customizing your sketch!

Step 3: Update your pseudocode

There are different ways you can write this pseudocode, so if yours doesn't match exactly, that's ok!

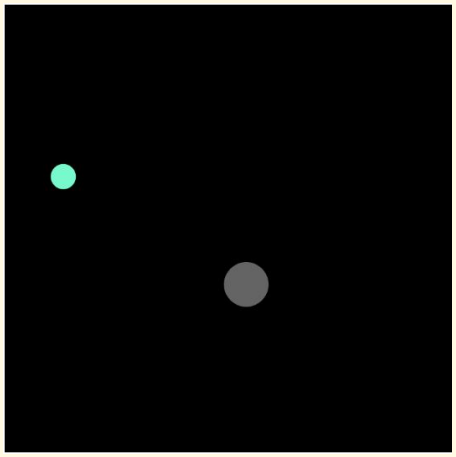
- ➔ **Meteor starting location:** Set `meteorX` equal to a random value between 0 and 400. We could also say set it to a random value anywhere on the width of the x axis.
- ➔ **Meteor speed:** Set the value of the `speed` variable to a random value between 0.5 and 4
- ➔ **Meteor size:** Set the value of `meteorDiameter` to be a random value between 10 and 30

Step 5: Test Your Code

JAVASCRIPT

```
// Test to see if meteor and catcher have intersected
if (distance < 15) {
  // Redraw  meteor to top of screen at a
     random location on x-axis
  meteorY = 0;
  meteorX = random(width);

  // Set meteor speed to random number between
     1 and 4
  speed = random(1,4);

  // Set meteor diameter to random number
     between 10 and 30
  meteorDiameter = random(10,30);
}

// Test to see if meteor has intersected with
   bottom wall
if(meteorY > height) {
  meteorY = 0;
  meteorX = random(width);
  speed = random(1,4);
  meteorDiameter = random(10,30);
}
```

RESULT



Click here to run the example sketch.

Step 6: Check for Understanding

Describe how this line of code would change the behavior of our catcher:

```
ellipse(mouseX, mouseY, random(80, 120), random(80, 120));
```

This line of code will change the size of oval shapes but continue to have a location that follows the mouse. If the ellipse is in `draw()`, the random value generated will change each loop.