



Girls Who Code At Home

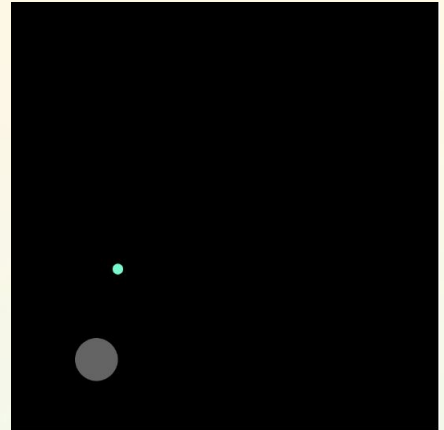
Meteor Catcher Game: Part 4

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.



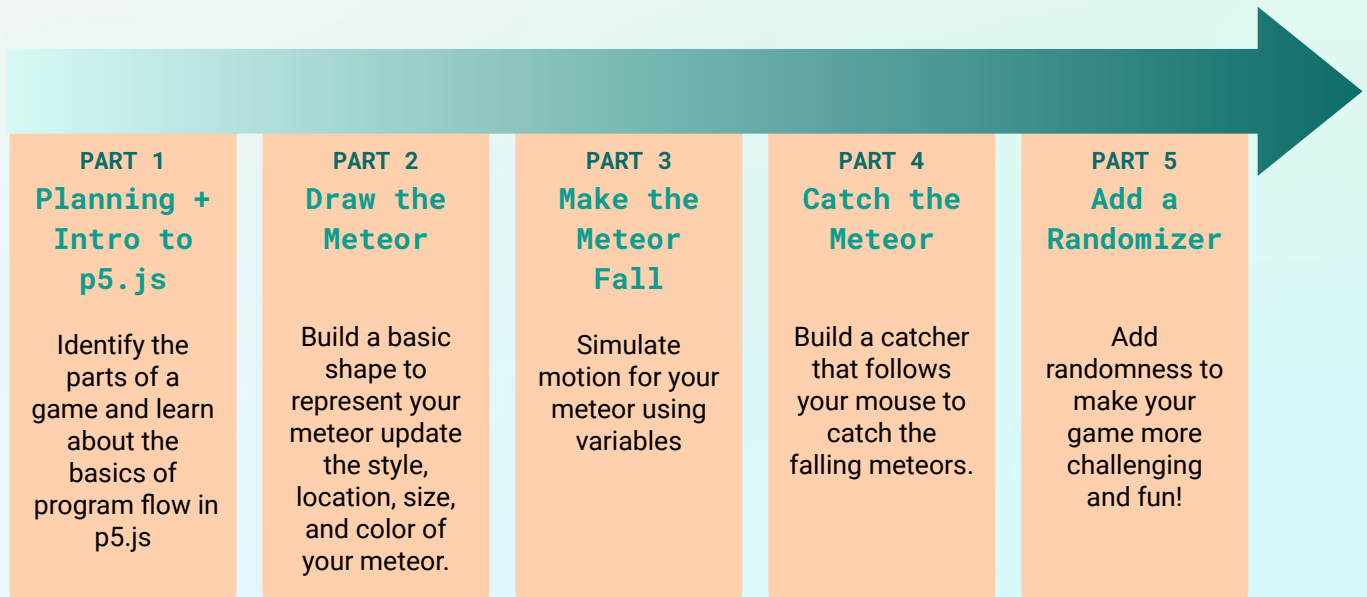
Learning Goals

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](#)



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](https://www.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](https://www.pixelles.org/).

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](https://behaviourinteractive.com/) 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](https://www.pixelles.org/) 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.



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!

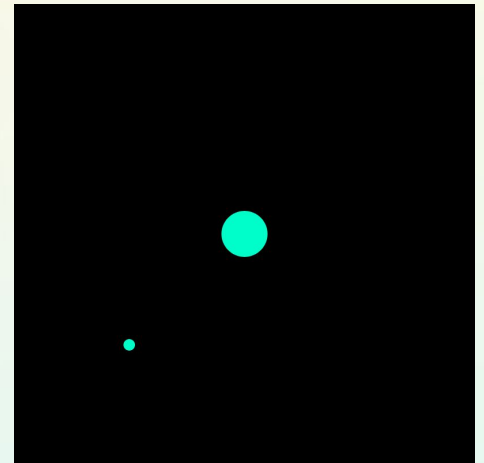
Step 1: Add the Catcher (5-10 mins)

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.

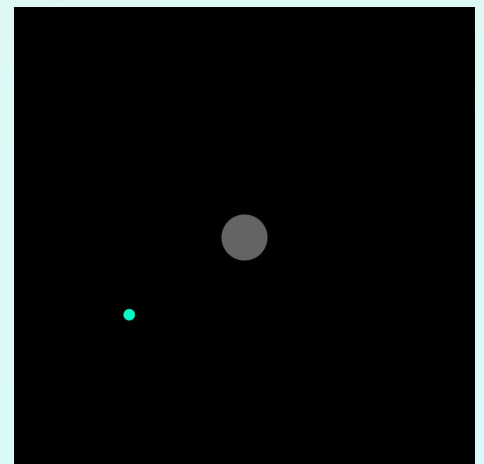
You should see a teal circle in the middle of the canvas:



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.



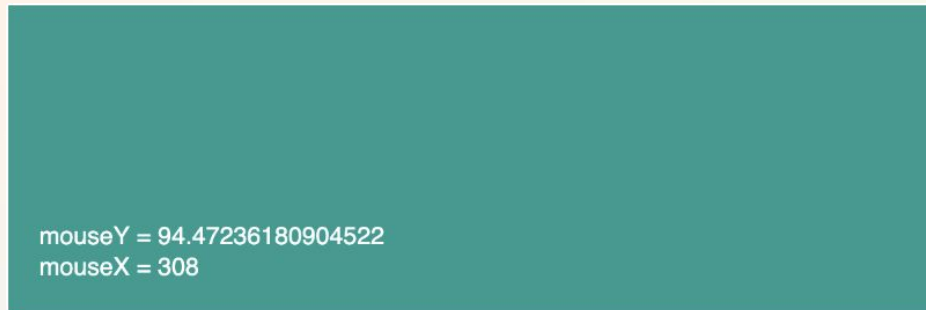
You should see the color of the catcher change to a semi-transparent white. In the image below, the catcher may appear gray. Since it is semi-transparent, some of the background color leaks through.



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.



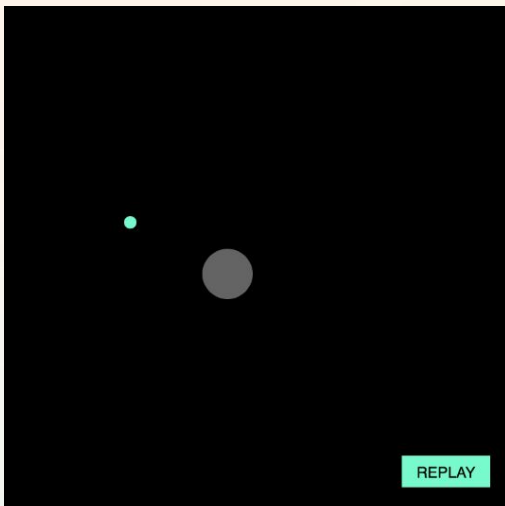
We can use `mouseX` and `mouseY` to attach the catcher to the mouse. This means that the x and y position of the mouse becomes the x and y position of the catcher. Let's update our catcher:

- ❑ 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)

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).

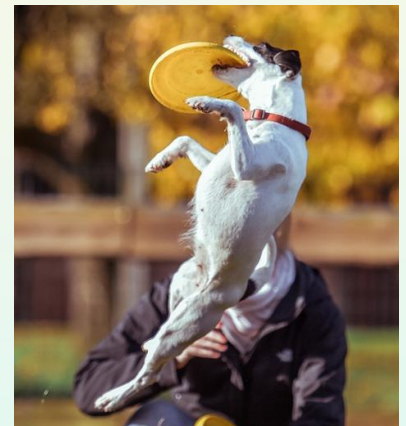
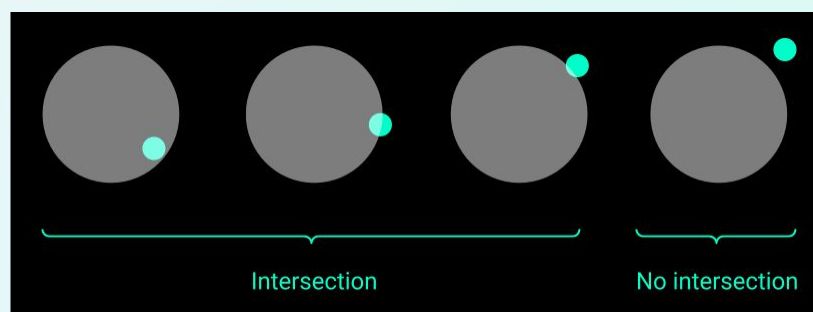


Image Source: [Pixabay](https://www.pixabay.com/)

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!

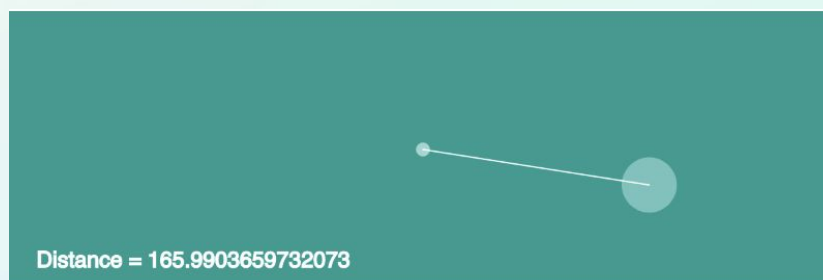
Step 6: Calculating distance (10-15 mins)

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
<code>dist(x1, y1, x2, y2);</code>	<ul style="list-style-type: none">→ <code>dist</code>: The function name.→ <code>()</code>: 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.→ <code>x1</code>: The x coordinate of the first point.→ <code>y1</code>: The y coordinate of the first point.→ <code>x2</code>: The x coordinate of the second point.→ <code>y2</code>: The y coordinate of the second point.→ <code>;</code>: 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.”

Step 6: Calculating distance (cont.)

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.

Step 6: Calculating distance (cont.)

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
<pre>print('Distance = ' + distance);</pre>	<ul style="list-style-type: none">→ 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.

The screenshot shows the p5.js IDE interface. On the left, the 'sketch.js' file is open, displaying the following code:

```
> sketch.js
48 fill(255, 255, 255, 100);
49 ellipse(mouseX, mouseY, catcherDiameter, catcherDiameter);
50
51 // Determine the distance between meteor and the catcher
52 distance = dist(meteorX, meteorY, mouseX, mouseY);
53
54 // Print the value of distance
55 print('Distance = ' + distance);
56
57 }
58
```

On the right, the 'Preview' window shows a dark gray circle (the catcher) on a black background. A small light blue dot (the meteor) is positioned near the top-left of the catcher circle.

Below the code editor, the 'Console' window is open, showing a series of distance values printed over time:

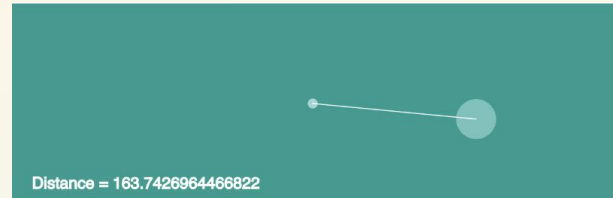
```
Distance = 16.439066914563774
Distance = 17.327723451163457
Distance = 14.212670403551895
Distance = 11.01135777277262
Distance = 10
Distance = 7.632168761236874
Distance = 6.4031242374328485
Distance = 6.020797289396148
```

Step 7: Determining intersection (5 mins)

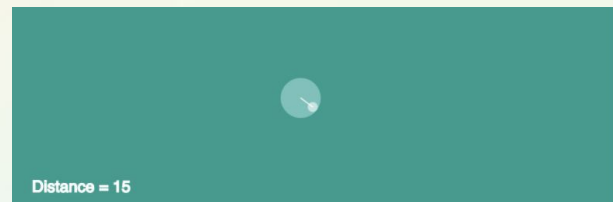
Define intersection using the distance value (2-3 mins)

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.

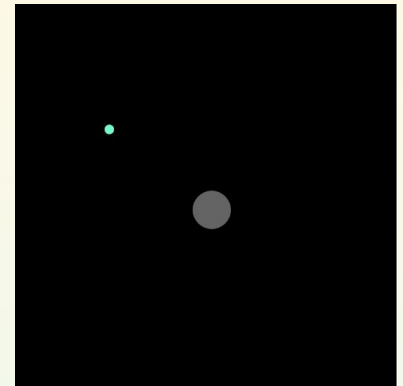
Below is an example to illustrate the syntax for a conditional statement in JavaScript (remember that p5.js is a library for JavaScript). It tells our program to draw a circle in the top left corner of the canvas if the x position of the mouse is greater than 100.

JAVASCRIPT	DESCRIPTION
<pre>if(mouseX > 100){ ellipse(0,0,50,50); }</pre>	<ul style="list-style-type: none">→ 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.

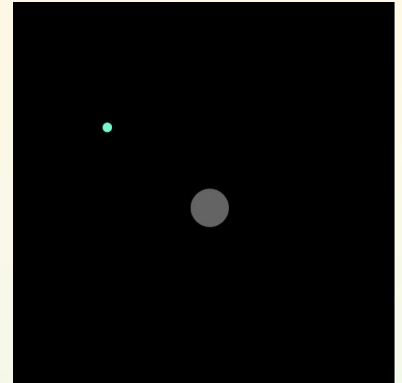


Don't forget to check your code with the Reference Guide on pg 5.

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)

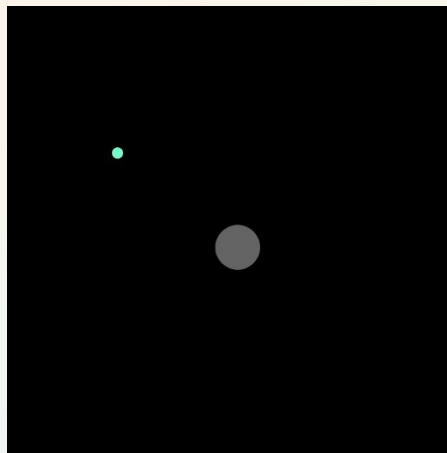
Now let's translate your pseudocode to actual code:

- ☐ 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?



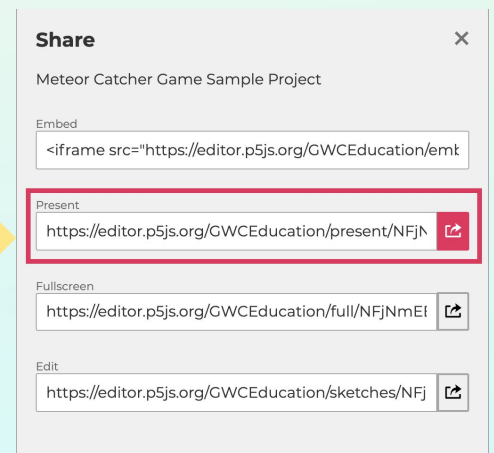
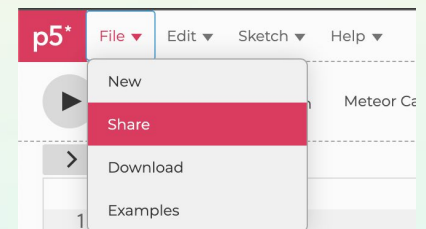
Don't forget to check your ideas with the Reference Guide on pg 7.

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!

