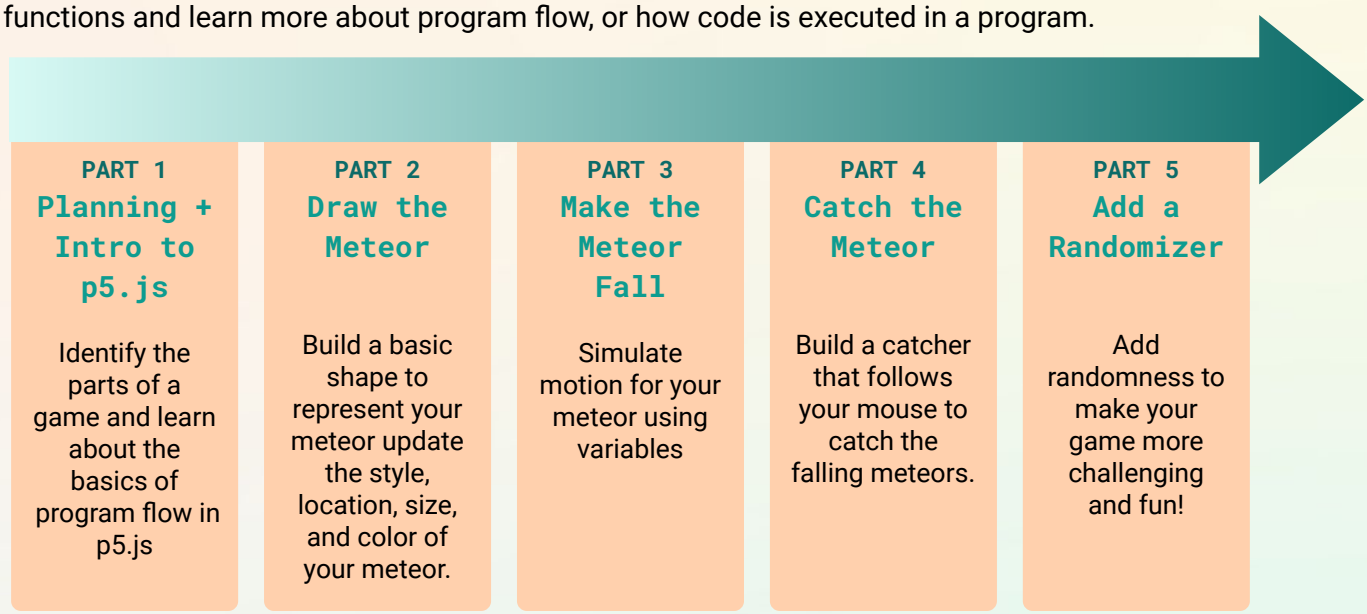
Girls Who Code At Home

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?

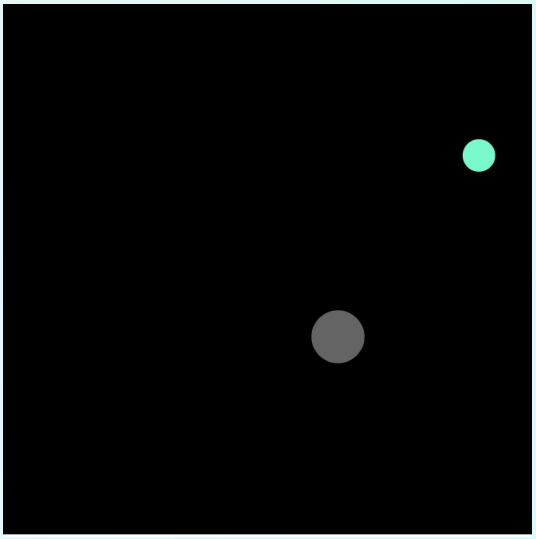
Example

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.

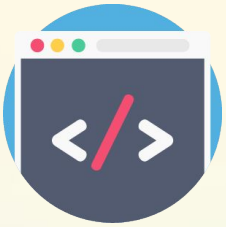
```
// Starter pseudocode
Declare any variables

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.

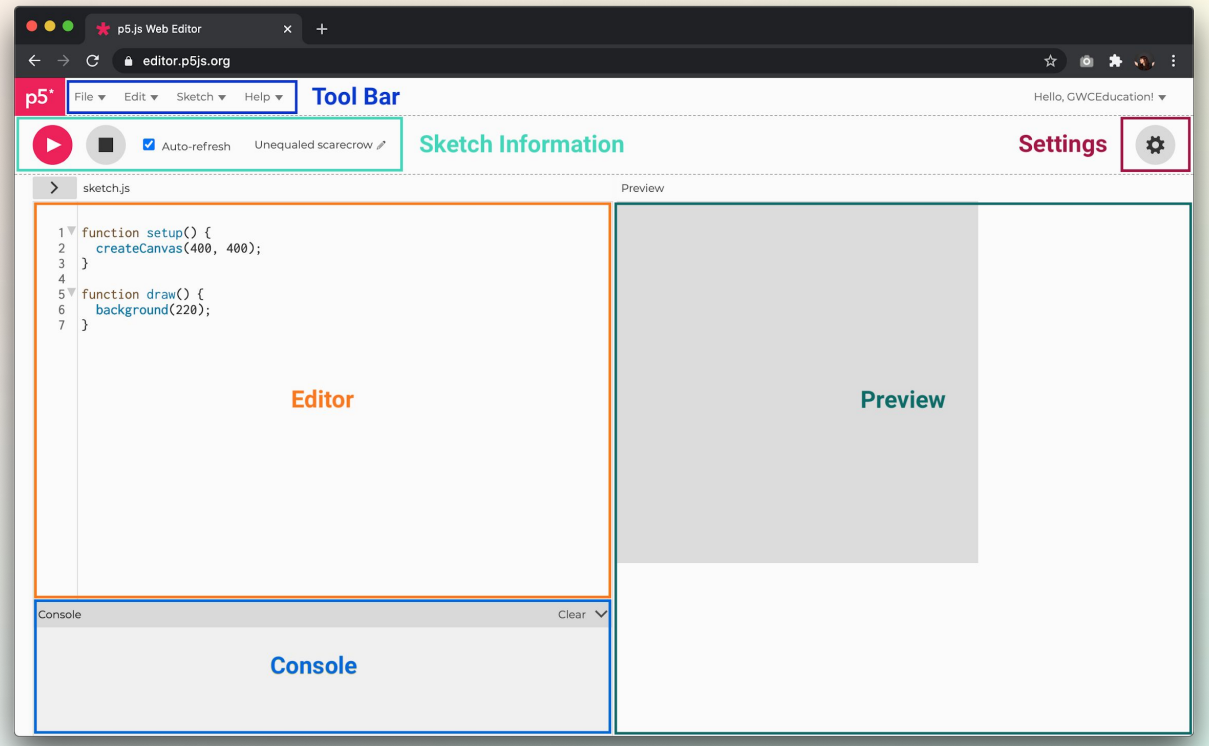
- ❏ Go to **https://editor.p5js.org/signup**.
- ❏ **Sign Up.** Fill in all the fields (username, email, password, and password confirmation) then click "Sign up" or you can choose to sign in with Google or GitHub.
- ❏ **Confirm your Email.** You will receive an email to confirm and verify your account (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.

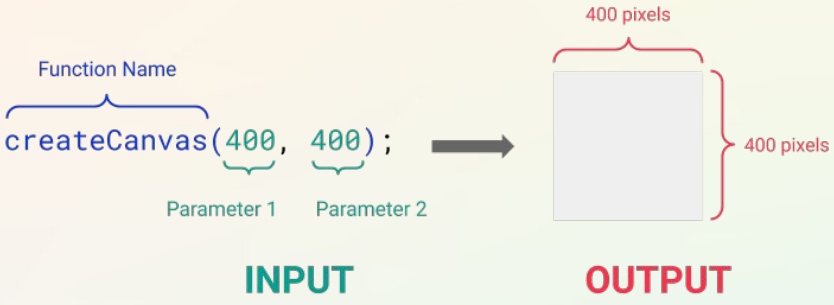
Accessibility in p5.js (2 mins)

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)

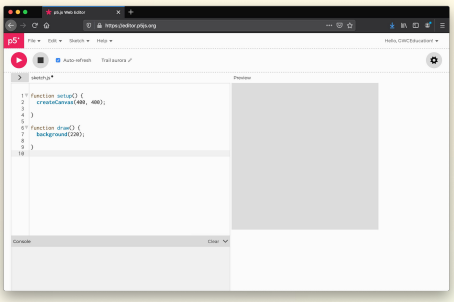
A **program** is a set of instructions you create for a computer to follow. Instead of writing the same instructions over and over, we can group instructions into chunks so we can reuse them later. These chunks are called **functions**. Functions are lines of code that perform a set of actions. You can 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:

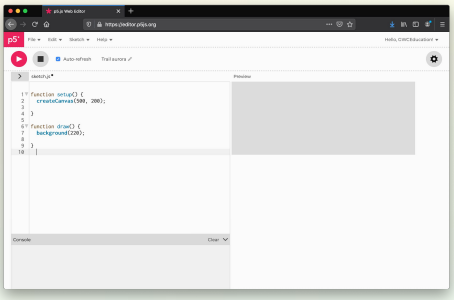
JAVASCRIPT	DESCRIPTION
`createCanvas(width, height);`	➔ `createCanvas`: The function name. ➔ `()`: We use parentheses to tell our program that it needs to call the function. Sometimes we include parameters or inputs in the function inside our parentheses. ➔ `width`: The first parameter that sets the 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:

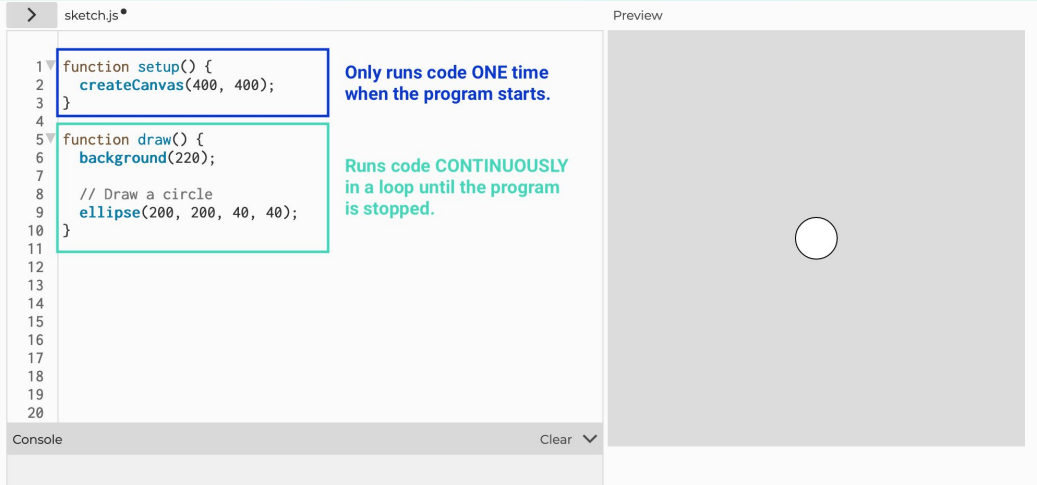
- ❏ **Open the p5 web editor and login.** You may have noticed that the sketch came prepopulated with starter code, including our friend `createCanvas()`. The default size of the canvas is 400 pixels wide and 400 pixels high.
- ❏ **Click the play button to run the code.** Notice the size of the canvas.
- ❏ **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.

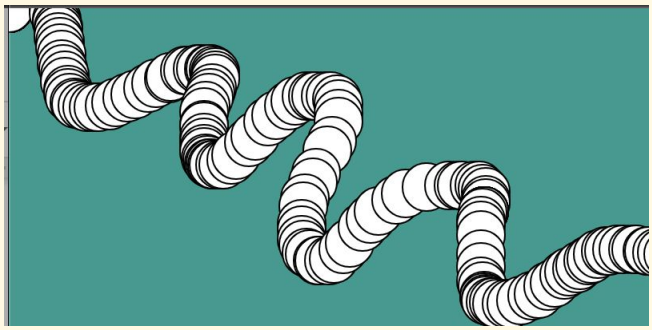
Placing the background function in the `setup()` or `draw()` of a sketch yields very different results. Consider this code:

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

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

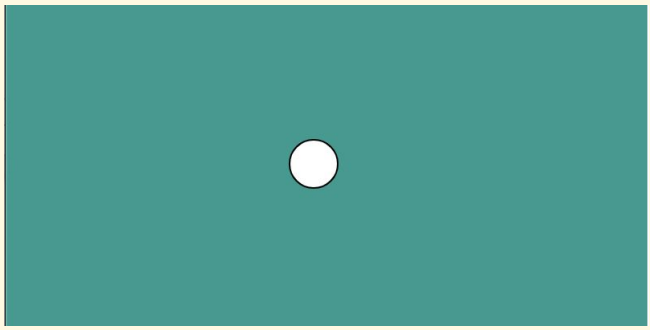
Right now, this sketch does not have a background. It creates a canvas and draws a circle or ellipse to the screen at the position of the mouse. Since the `ellipse()` function is in `draw()`, our program paints a new circle to the screen each time the program loops through, about 60 times per second,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