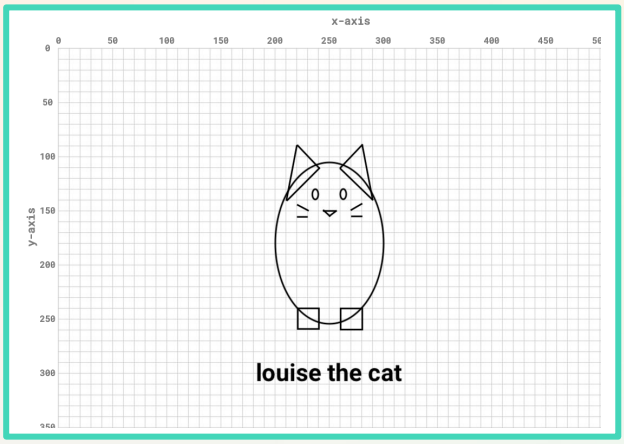
Girls Who Code At Home

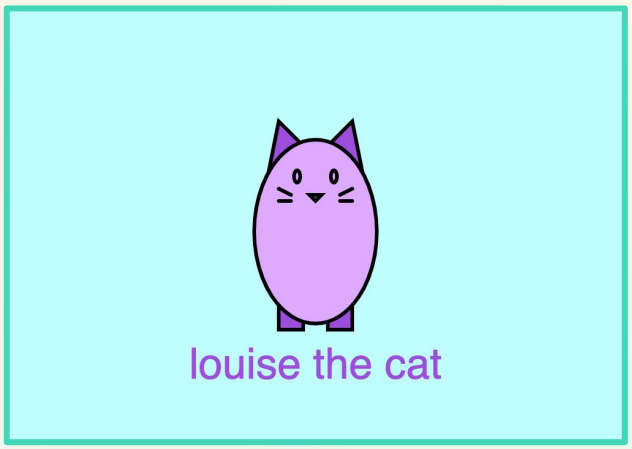
Digital Art Rules

Part 2

Activity Overview

Are you an artist or designer who is interested in exploring code? Or maybe you have some experience with code and want to explore creative processes. Or perhaps you want to explore both! Either way, this project is for you! Over two activities, we will learn the basics of p5.js, a JavaScript library made for beginners and creative coders, by creating a piece of digital art.





Part 1 is all about planning, brainstorming, and paper prototyping. We will work on developing a set of rules to guide your artwork, then draw it on a grid. **Part 2** is focused on introducing you to the basics of p5.js so you can translate your analog drawing to a digital sketch. Along the way, you will practice using the design process and get to know a range of Black and African American women and female-identifying designers and artists.

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.
- ❏ translate your physical drawing to a digital environment using code.

Materials

- ➔ p5.js Online Editor
- ➔ Your drawing from Part 1
- ➔ Your instructions from Part 1
- ➔ p5.js Reference
- ➔ Digital Art Rules Part 2 Reference Guide

Prior Knowledge

- ➔ You should have completed Digital Art Rules Part 1 before beginning this activity.
- ➔ It is still possible to complete Part 2 using an example image and the Reference Guide. We have included key concepts you will need to cover or review from Part 1.

Women in Tech Spotlight: Kelechi Anyadiegwu

Source: African Business Central

In the Shona language of Zimbabwe, "Zuvaa" translates to sunshine. When Kelechi first heard it, she immediately knew that it would be perfect for her company because it represented her brand's positivity, pride, and the inner light that shines in African fashion. Founded in 2013, Zuvaa curates designs from over 80 vendors and operates as a pop-up shop and online retailer for African prints and designs. Since its founding, Zuvaa has become a dominating force in the fashion industry.

After graduating from college, Kelechi returned home to pursue her dream of becoming an entrepreneur. Currently based in Atlanta, Kelechi runs Zuvaa as its founder and CEO.

Watch this video about Kelechi's journey to learn more about her pathway to becoming a fashion tech entrepreneur, and her vision for African fashion in the United States of America!

Reflect

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

PURPOSE

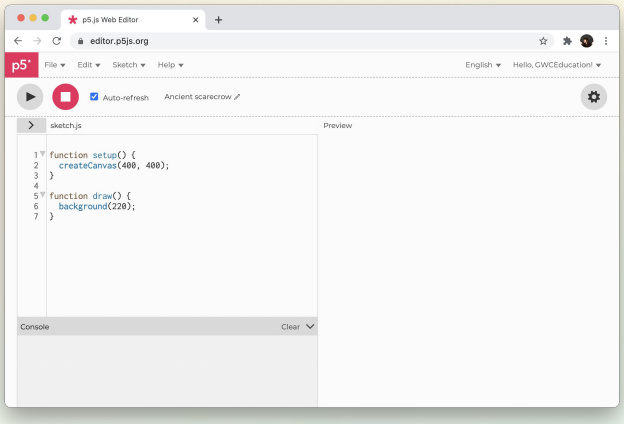
There were many African clothing retailers when Zuvaa entered the market. Despite the competition, what made Kelechi stick with her company?

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

Step 1: Meet p5.js! (10-15 mins)

We have now reached the digital portion of our journey! In this step, you'll meet p5.js and create an account.

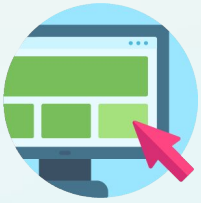
P5.js (or just p5) allows you to create interactive art for web browsers. It is a tool for **creative coding** - projects that use code for expression instead of just functionality. P5.js is a library for JavaScript, a programming language that allows you to add interactivity on the web. Being a library 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.



Screenshot of the p5.js Web Editor

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)



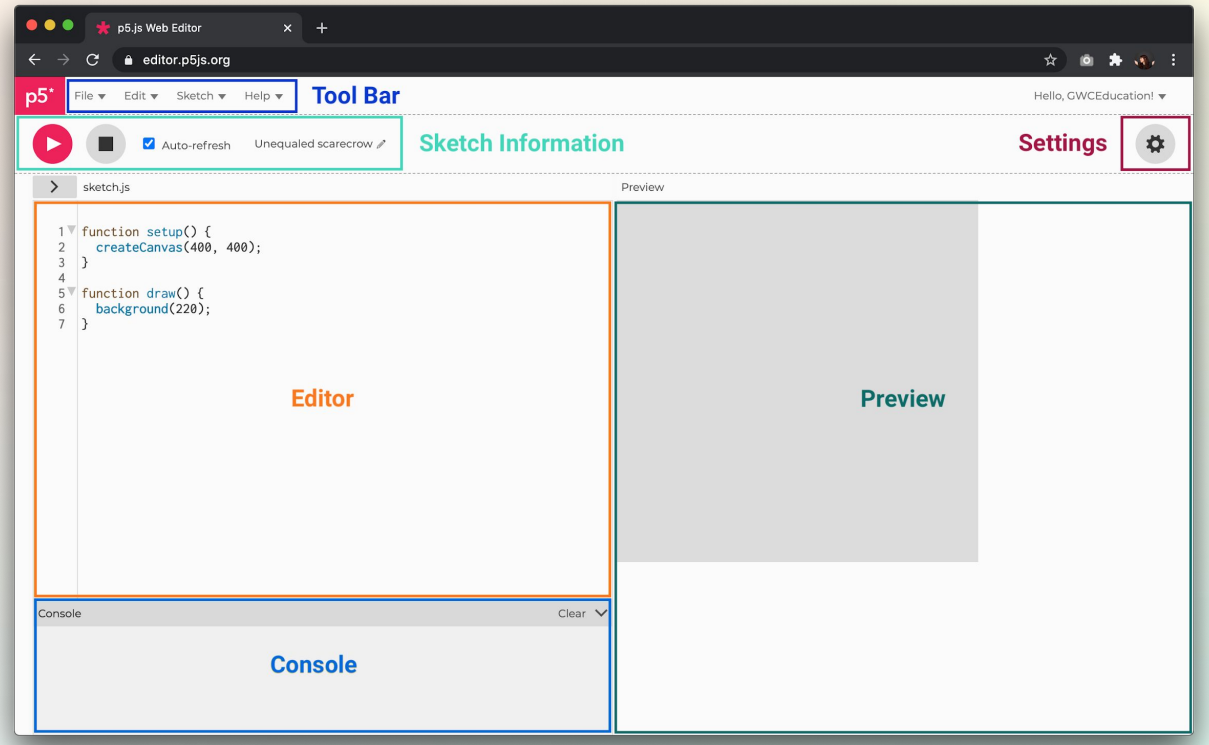
There are two ways you can use p5.js: the online web editor or a text editor and copy of p5.js that you download to your local computer. The easiest way to get started with p5 is the online editor. This allows you to write code and run your program in a web browser. In this tutorial, we are only going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

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.

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.

- **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 2: 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.

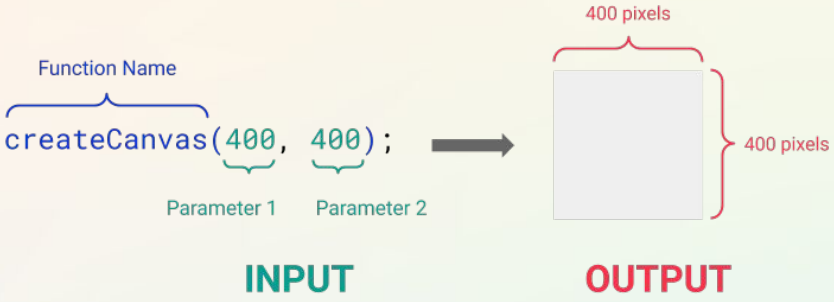
- ➔ Single line comments use a double forward slash, `//`.
- ➔ Multiline comments use a forward slash and asterisk, `/*`, to open it and an asterisk and forward slash, `*/`, to close it.

```
// This is a single line comment

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

Step 3: Write your first function (8-12 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.



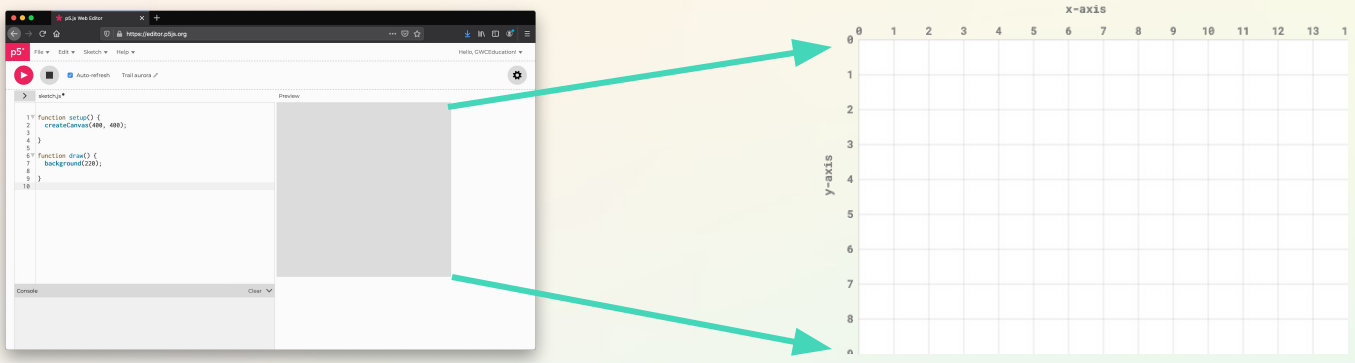
Set your canvas size (5-7 mins)

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:

FUNCTION	DESCRIPTION
`createCanvas(width, height);`	➔ `createCanvas`: The function name. *To learn more, see the `createCanvas()` entry in the 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. ➔ `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.

If this sounds a lot like the coordinate plane or grid we created in Part 1, your intuitions are correct! The top of the canvas is our x-axis and the left side is our y-axis.

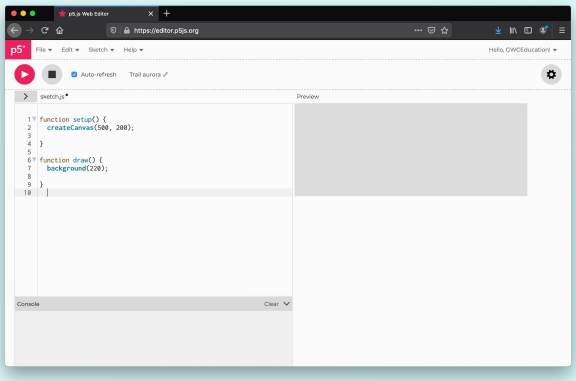


If you need a refresher on the coordinate system, see the Reference Guide pg 2.

We need to change the parameter values to resize the canvas based on the size of your digital artwork. Use the drawing you created in Part 1 or the example for reference.

- ❏ **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. Gray isn't that much fun though. Let's use this newfound *function*-al knowledge to change the background color to the one you picked.



Set your background color (2-5 mins)

The background() function 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.

FUNCTION	DESCRIPTION
`background(redValue, greenValue, blueValue);`	➔ `background`: The function name. *To learn more, see the background() entry in the 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. ➔ `redValue`: The red value between 0 and 255. ➔ `greenValue`: The green value between 0 and 255. ➔ `blueValue:` The blue value between 0 and 255. ➔ `;`: All lines of code in JavaScript must end with a semicolon.

We'll be using RGB color mode. It uses combinations of red, green, and blue light to create a range of digital colors. You can assign a value to each color value - red, green, and blue - between 0 to 255. For example, (255, 0, 0) would be red, (0, 0, 0) would be black, (78, 205, 196) would be teal, and (255, 255, 255) would be white.

= (255,0,0) = (0,0,0) = (78,205,196) = (255,255,255)

Use the `background()` function to add color to your canvas:

- ❑ Check your drawing or instructions from Part 1 to remember the background color you chose.
- ❑ Find the RGB values for the color. You can use a tool like color pickers or Coolors if you need help determining the values.
- ❑ Call the `background()` function inside `draw()` and add the RGB values (remember that location is important!).

Step 4: Learn about program flow (5-10 mins)

In the last step, we learned about two functions, but we put them in two different locations:

```
function setup() {
  createCanvas(400, 300);
}

function draw() {
  background(13, 156, 144);
}
```

We know how to give our program instructions, but how do we know where to 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. Think about baking your favorite dessert - like cookies. First you get out all the ingredients, then you measure them, mix them, put the dough on a cookie sheet, bake them, and eat them. These steps happen *sequentially* - you can't perform these steps out of order. For example, you can't eat the cookies before you measure the ingredients.

`My Overly Simplified Cookie Recipe`

1. Get out all the ingredients.
2. Measure the ingredients.
3. Mix them together.
4. Drop the dough on a cookie sheet
5. Bake the cookies.
6. Eat all of them.

There are two core functions in p5.js that determine *when* and *how often* your code runs: `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.

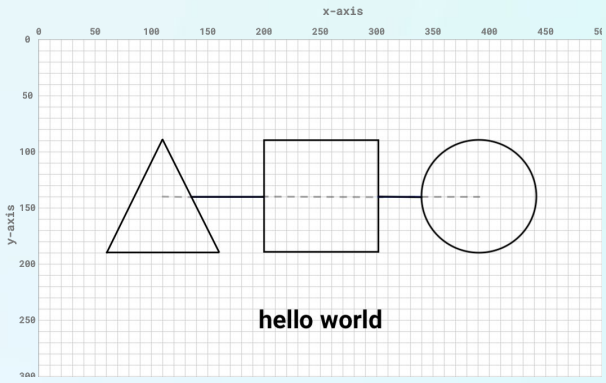
If you are only creating static or still sketches with no movement or interaction, then you can put all your code in `setup()`. But if you are animating a shape or want to listen for a mouse click, you will likely need to put most of your code in `draw()`. Since you will be using `draw()` a fair amount as you work more and more with p5.js, we will put most of our code there.

Step 5: Practice translating an example image (12-15 mins)

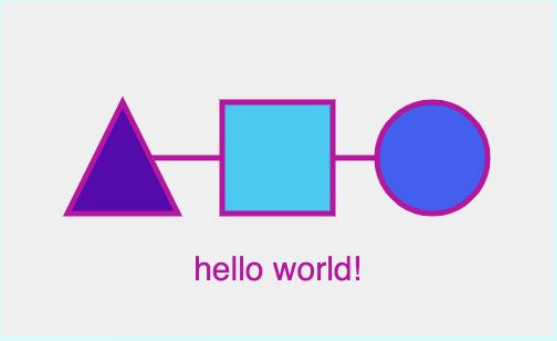
The last batch of knowledge you need to create your digital artwork are the functions that will render your design elements: shape, line, color, and text.

SHAPE	LINE	COLOR	TEXT
`rect();`	`line();`	`background();`	`text();`
`rectMode();`	`strokeWeight();`	`fill();`	`textSize();`
`ellipse();`	`noStroke();`	`stroke();`	`textAlign();`
`triangle();`			
`quad();`			

In this step, you will practice using these functions by translating the example drawing on the left into the p5.js digital sketch on the right. We will pay special attention to the sequencing or order in which we write our functions.



Example Drawing

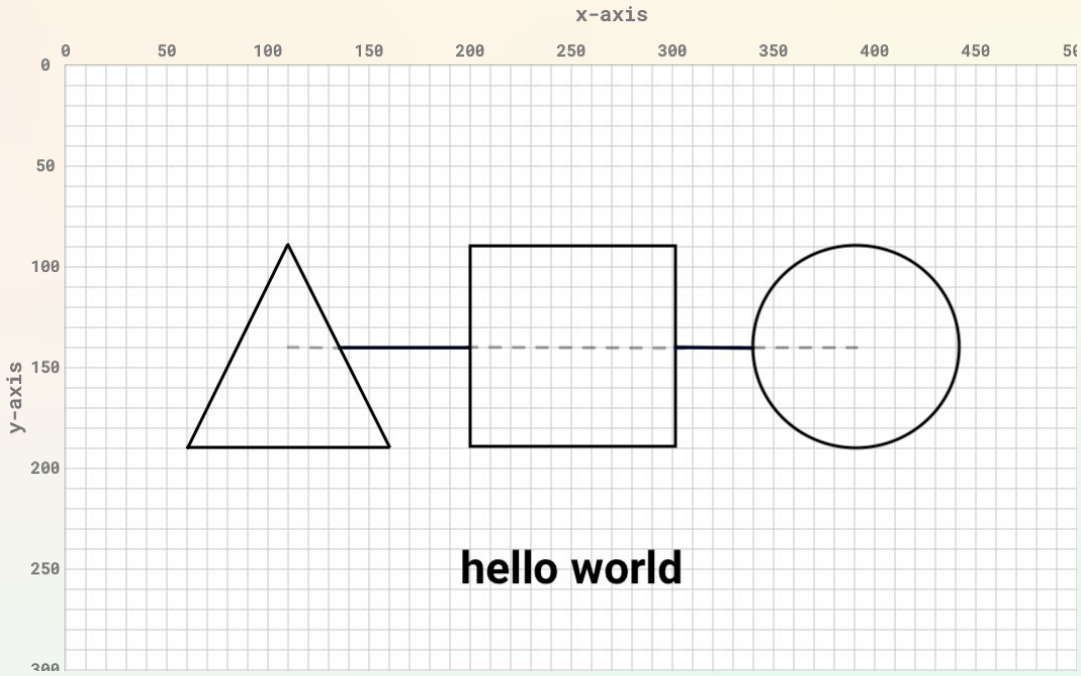


P5.js digital sketch

Planning (3-5 mins)

Review the example drawing

Our drawing is on a grid with a width of 500 pixels and a height of 300 pixels. Each square on the grid represents 10 pixels.



Review the example instructions

Read through the instruction set below to get a sense of the data you need to complete the drawing. We will remind you of the individual instructions as we program each shape.

ELEMENT	VALUES	COLOR
Square	Center = (250,140) Width = 100 Height = 100	Light blue: R = 76, G = 201, B = 240
Circle	Center = (390,140) Width = 100 Height = 100	Dark blue: R = 67, G = 97, B = 238
Triangle	Point 1 = (110,90) Point 2 = (160,190) Point 3 = (60,190)	Purple: R = 86, G = 11, B = 173
Line	Point 1 = (110,140) Point 2 = (390,140)	Light purple: R = 181, G = 23, B = 158
Text	Text = “hello world!” Center = (250,250)	Light purple: R = 181, G = 23, B = 158

Make a plan

Once you've reviewed them, let's come up with a general plan for how we will program the sketch:

Program the lines and shapes

- ➔ Draw the shape or line in the desired location. Test your sketch.
- ➔ Make stylistic changes to the shapes and/or lines if necessary. Test.
- ➔ Fill the shape or line with the given color. Test..
- ➔ Repeat for the next shape or line.

Program the text

- ➔ Draw the text to the screen in the desired location. Test your sketch.
- ➔ Make any stylistic changes to the text if necessary. Test.
- ➔ Change the color of the text. Test.

Review the starter code

Finally, open this starter code sketch and make a copy. Rename it to something descriptive that you will remember. Our starter code only contains a few lines. In `setup()`, we set the canvas size to 500 pixels in width and 300 pixels in height. In `draw()`, we set the background color to a light gray by giving it a value of 240. We only need to use one value if the color is in grayscale with 0 being black and 255 being white.

```
// Only runs once
function setup() {
  createCanvas(500, 300);
}

// Runs over and over in a loop
function draw() {
  // Set the background
  background(240);
}
```

Starter Code Sketch

Program the square (5-7 mins)

Let's draw the square first using the `rect()` function. This function allows you to draw a rectangle on the canvas (remember that a square is a rectangle!).

FUNCTION	DESCRIPTION
`rect(x, y, width, height);`	➔ `rect`: The function name. *To learn more, see the rect() entry in the 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 of the rectangle. ➔ `y`: The y-coordinate of the rectangle. ➔ `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.

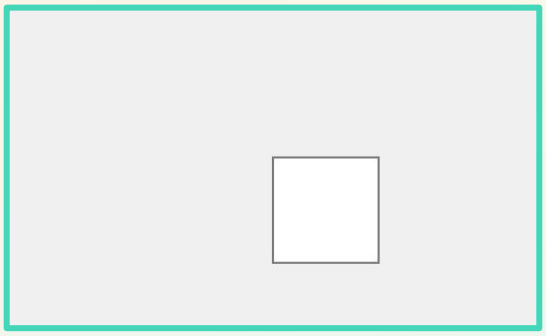
Use the instructions and the function reference table to add this line of code to your sketch:

- ❏ **Add a square to your canvas using the `rect()` function.**
- ❏ **Press play to test it.**

Instructions: Values

- ➔ Center = (250,140)
- ➔ Width = 100 Height = 100

When you run your code, a white square with a black outline should display in the lower right:

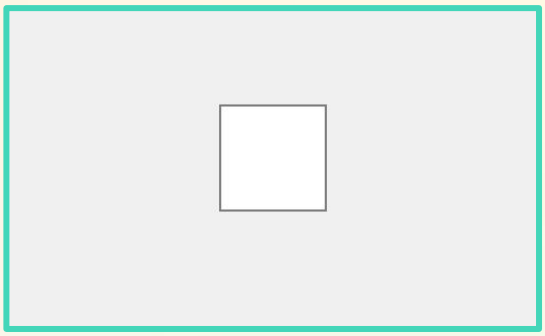


But it's not in the right location! By default, p5.js sets the location coordinates in the upper left corner, not the center. We can use the `rectMode()` function to change how p5 interprets the location coordinates we give the `rect()` function.

FUNCTION	DESCRIPTION
`rectMode(CENTER);`	➔ `rect`: The function name. *To learn more, see the rectMode() entry in the 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. ➔ `CENTER`: Interprets the first two parameters as the shape's center point. Note that this is in all caps and is case sensitive. ➔ `;`: All lines of code in p5.js must end with a semicolon.

- ❏ **Add the `rectMode()` function**. Place it **before** the `rect()` function. Since our program reads sequentially, we need to tell p5 how to interpret the `rect()` parameters *before* we call or use the `rect()` function.
- ❏ **Press the play button to test it.**

The center of your square should now be the same as the center of the canvas.



Time to add color using the `fill()` function in RGB mode. Similar to the `rectMode()` function, the `fill()` function should come *before* our shape. We need to tell p5 the color we want to paint our shape before we draw it. You can think of it like an actual paint brush - we can't paint anything until we add color to our brush!

FUNCTION	DESCRIPTION
`fill(redValue, greenValue, blueValue);`	➔ `fill`: The function name. *To learn more, see the fill() entry in the 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. ➔ `redValue`: The red value between 0 and 255. ➔ `greenValue`: The green value between 0 and 255. ➔ `blueValue:` The blue value between 0 and 255. ➔ `;`: All lines of code in p5.js must end with a semicolon.

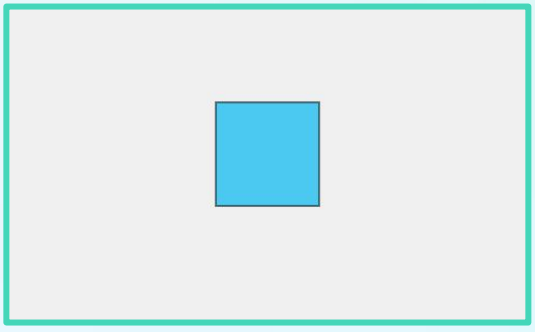
Use the instructions and the function reference table to add this line of code to your sketch:

- ❑ **Add the `fill()` function before the `rect()` function.**
- ❑ **Press the play button to test it.**

Instructions: Color

➔ Light blue: R = 76, G = 201, B = 240

Your square should turn a light blue color:



Check your code in the Reference Guide on pg 3.

Program the circle (2-3 mins)

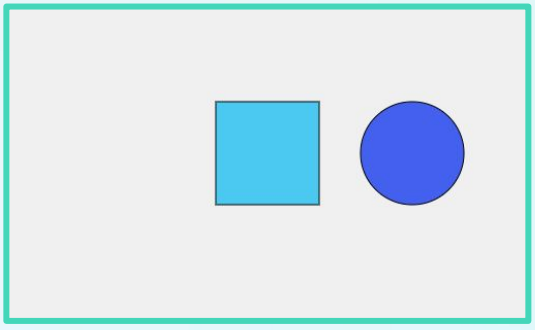
One shape down! Now onto the circle. Our star function here is the `ellipse()` function. It allows you to draw an ellipse - a fancy word for oval - on the canvas.

FUNCTION	DESCRIPTION
`ellipse(x, y, width, height);`	➔ `ellipse`: The function name. Ellipse is another word for oval. *To learn more, see the ellipse() entry in the 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 instructions below and the function reference table to add these lines of code to your sketch:

- ❑ **Draw the circle in the correct location.**
- ❑ **Add color using the `fill()` function.** Remember that the order of your code matters!
- ❑ **Press the play button to test it.**

Your circle should be to the right of the square and have a dark blue color:



Instructions: Values

- ➔ Center = (390,140)
- ➔ Width = 100 Height = 100

Instructions: Color

- ➔ Dark blue: R = 67, G = 97, B = 238

Check your code in the Reference Guide on pg 4.

Program the triangle (2-3 mins)

We can use the `triangle()` function to draw our last shape to the canvas.

FUNCTION	DESCRIPTION
`triangle(x1, y1, x2, y2, x3, y3);`	➔ `triangle`: The function name. To learn more, see the *`triangle()` entry in the 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. ➔ `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. ➔ `x3`: The x-coordinate of the third point. ➔ `y3`: The y-coordinate of the third point. ➔ `,`: 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 instructions below and the function reference table to add these lines of code to your sketch:

- ❑ **Tell p5.js the color of the triangle using the `fill()` function.** Remember that the order of your code matters!
- ❑ **Draw the triangle in the correct location.**
- ❑ **Press the play button to test it.**

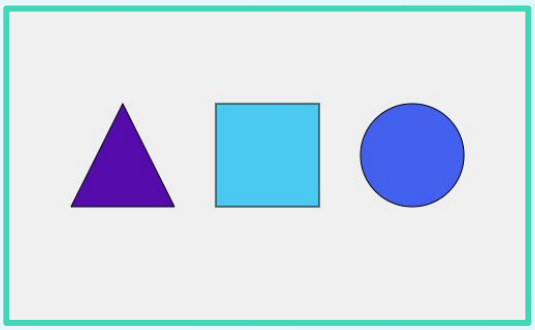
Instructions: Values

- ➔ Point 1 = (110,90)
- ➔ Point 2 = (160,190)
- ➔ Point 3 = (60,190)

Instructions: Color

- ➔ Purple: R = 86, G = 11, B = 173

Your triangle should be to the left of the square and have a purple color:



Check your code in the Reference Guide on pg 4.

Program the line (5-7 mins)

Next, we'll draw the line that goes through the middle of all three shapes using the `line()` function.

FUNCTION	DESCRIPTION
`line(x1, y1,x2, y2);`	➔ `line`: The function name. *To learn more, see the `line()` entry in the 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. ➔ `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. ➔ `,`: 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.

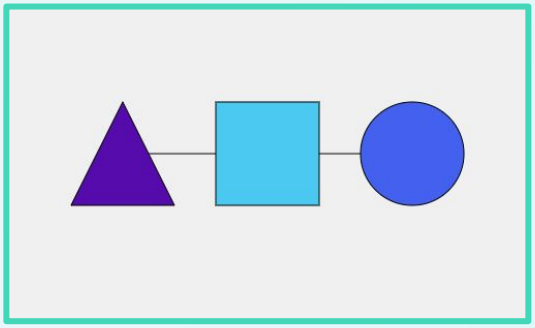
Use the instructions below and the function reference table to add this line of code to your sketch:

- ❑ **Draw one line that touches all shapes.** The line should not be visible on the front of the shapes. Remember that the order of your code matters!
- ❑ **Press the play button to test it.**

Instructions: Values

- ➔ Point 1 = (110,140)
- ➔ Point 2 = (390,140)

You should see a line from the triangle to the circle that only displays **behind** the shapes. If your line is in front of the shapes, try moving the `line()` before the shapes in your code.



From an aesthetic perspective, the thin black lines don't really do much for the composition or visual arrangement of the image. Let's change that. We can add color to our line with the `stroke()` function and change the thickness of our line using `strokeWeight()`.

FUNCTION	DESCRIPTION
`stroke(redValue, greenValue, blueValue);`	➔ `stroke`: The function name. *To learn more, see the stroke() entry in the 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. ➔ `redValue`: The red value between 0 and 255. ➔ `greenValue`: The green value between 0 and 255. ➔ `blueValue:` The blue value between 0 and 255. ➔ `;`: All lines of code in p5.js must end with a semicolon.
`strokeWeight(weight);`	➔ `strokeWeight`: The function name. *To learn more, see the strokeWeight() entry in the 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. ➔ `weight:` The width of the stroke used for lines, points and the border around shapes. All widths are set in units of pixels. ➔ `;`: All lines of code in p5.js must end with a semicolon.

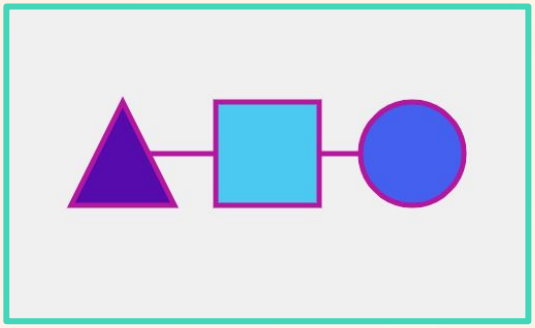
Use the instructions and the function reference table to add these lines of code to your sketch:

- ❑ **Change the line color.**
- ❑ **Increase the thickness of the line.** There is no parameter for the stroke weight in our instructions, so try changing the value until you find a thickness you like.
- ❑ **Press the play button to test it.**

Instructions: Color

➔ Light purple: R = 181, G = 23, B = 158

A thicker, light purple line should display on the canvas:



Check your code in the Reference Guide on pg 5.

Program the text (6-8 mins)

We have reached our final step - the text! There are a lot of things you can do with words and letters. Let's start with the basics - drawing text to the screen with the `text()` function.

JAVASCRIPT	DESCRIPTION
`text(string, x, y);`	➔ `text`: The function name. *To learn more, see the `text()` entry in the 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. ➔ `string`: A string is a series of text characters that can be words, letters, or symbols. They must be surrounded by either single-quotation marks(') or double-quotation marks("). ➔ `x`: The x-coordinate of the text location. ➔ `y`: The y-coordinate of the text location. ➔ `,`: 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.

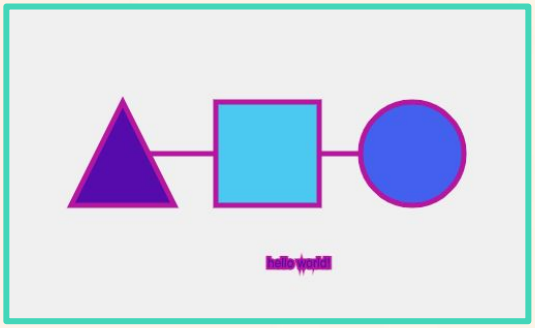
Use the instructions below and the function reference table to add this line of code to your sketch:

- ❑ **Add a message underneath the shapes in the center of the canvas.**
- ❑ **Press the play button to test it.**

Instructions: Values

- ➔ Text = "hello world!"
- ➔ Center = (250,250)

Your message should display as text on the canvas and appear something like this:



Check your code in the Reference Guide on pg 6.

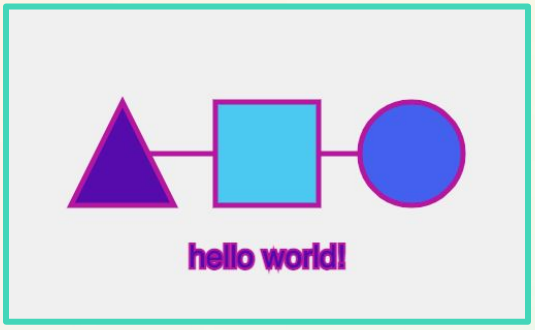
We have text on the screen, but that definitely does not resemble the text on our drawing. It's too small to read, off-center, has an outline, and is the wrong color. Let's tackle the size and alignment first. The `textSize()` function sets the size in pixels and `textAlign()` sets the text alignment.

FUNCTION	DESCRIPTION
`textSize(size);`	➔ `textSize`: The function name. *To learn more, see the `textSize()` entry in the 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. ➔ `size`: The size of the letters in units of 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.
`textAlign(horizontalAlign);`	➔ `textAlign`: The function name. *To learn more, see the `textAlign()` entry in the 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. ➔ `horizontalAlign`: Set the horizontal alignment using LEFT, CENTER, or RIGHT. Note that these values are in all caps and case sensitive. ➔ `,`: 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 function reference tables to add these lines of code to your sketch:

- ❑ **Set the size to 30.**
- ❑ **Align the text to center.**
- ❑ **Press the play button to test it.**

Your message should display in a larger size and be centered on the canvas:



Now let's fix the outline and color. So far we have used functions to add elements to our screen, but there are also functions that remove elements. We can call the `noStroke()` function to disable all lines and outlines that come **after** it.

FUNCTION	DESCRIPTION
`noStroke();`	➔ `noStroke`: The function name. *To learn more, see the `noStroke()` entry in the 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. ➔ `,`: 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.

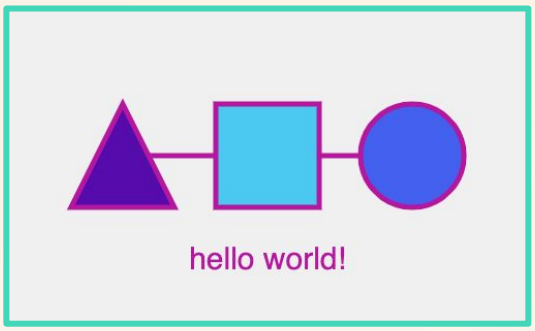
Right now the text takes the same color as the last `fill()` function we called for the triangle. We need to add another `fill()` function before `text()` to set it's color. Use the instructions and the function reference table to complete the following steps:

- ❑ **Remove the outlines on the text.**
- ❑ **Change the color of the text.**
- ❑ **Press the play button to test it.**

Instructions: Color

➔ Light purple: R = 181, G = 23, B = 158

Your text should be a light purple with no outlines. If it displays differently from the image below, check the sequence of functions in your code.



Check your code in the Reference Guide on pg 7.

Program complete! In this step, you learned how to translate your coordinate plane drawing and written instructions to a digital image in p5.js. You learned how to use and apply shape, line, and color functions to program your sketch. In the next step, you will use this same process to translate your drawing from Part 1 to a piece of digital art!

Step 6: Translate your drawing to p5.js (15-25 mins)

We will implement the same process from the last step to program your drawing.

Make a plan (2-5 mins)

Gather your drawing and instructions to review. As you read through your instructions, think about what sequence you should use to program your elements. Remember: order matters. You can write any additional notes or comments to yourself if it's helpful.

If you did not complete Part 1, you can use one of the samples in the Reference Guide on pg 8-11.

Write your program (10-20 mins)

Use your plan to program one element at a time. Just like we practiced in Step 6, you should write a line or two of code at a time, then press the play button to test your sketch. This will allow you to catch any mistakes early instead of having to sift through multiple lines of code to find your error.

For a full list of the function reference tables, check the Reference Guide on pg 12-16.

Keep adding all of your shapes, colors, lines, and text until you're finished. If you run into a problem, check out the Debugging Tips below. **Debugging** is the word programmers and engineers use for fixing technical problems.

Debugging Tips

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:

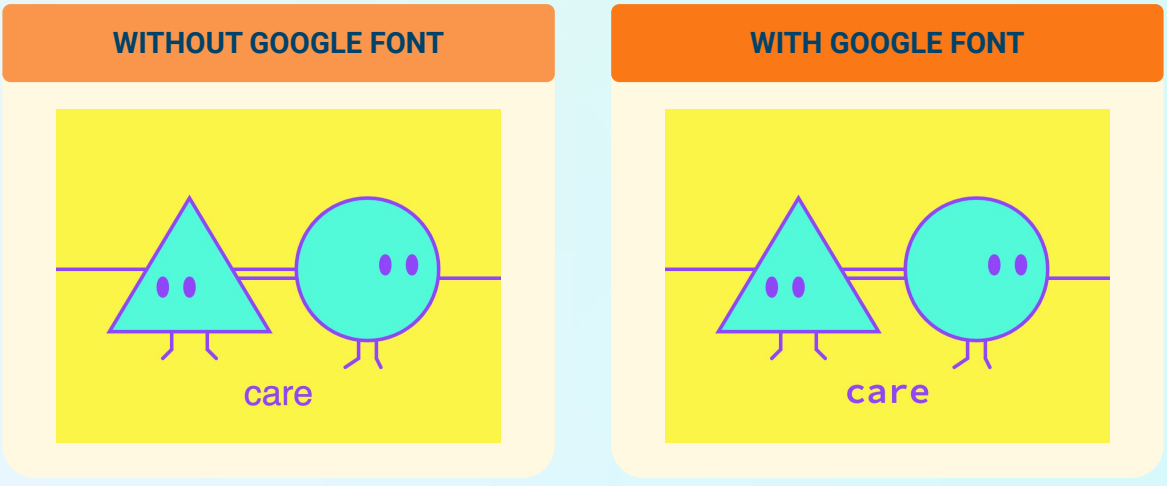
- ❏ **Is your code inside the correct curly brackets?**
- ❏ **Do you have semicolons at the end of each line of code?**
- ❏ **Did you spell the variable and function names correctly?** Remember that JavaScript is also case-sensitive!
- ❏ **Are your functions in the correct location and sequence?** Remember that order matters in program flow!
- ❏ **Do you have single or double quotation marks around the text in the functions that require it, such as the `text()` function?**
- ❏ **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?

To learn more about best practices for debugging, check out this post on debugging from the p5 community.

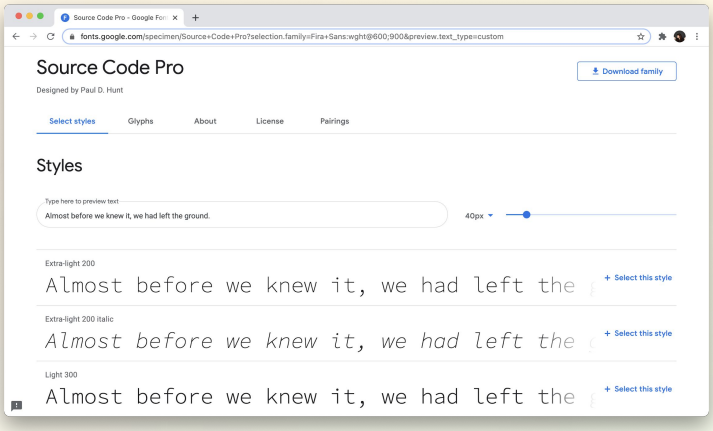
Step 7: Extensions (10-20 mins)

Extension 1: Add a new font (5-10 mins)

Right now, your text displays in a default sans serif font. You can change the font of your text in a few simple steps by embedding a link into your sketch's HTML file.



- ❏ **Choose a font from [Google fonts](fonts.google.com).** Click on the font you want to use.

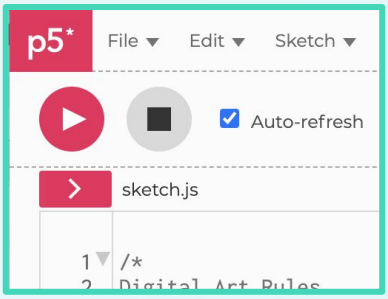


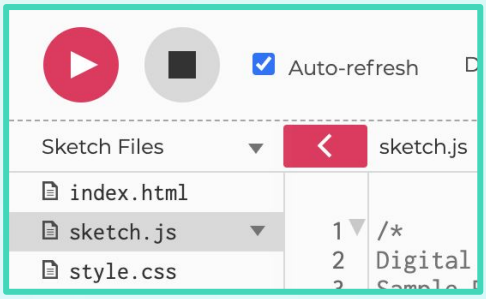
- ❏ **Select the styles.** You will notice that there are many different styles for each font. Click *Select this style* for the one you want to use.

- ❏ **Copy the HTML embed link.** In the top right corner, you will notice a button with three squares and a plus sign. Click this button to view the font you selected.

A side bar will pop up with information about your font and a link you can use to embed your font. Use your mouse to highlight the whole link and copy it to your clipboard.

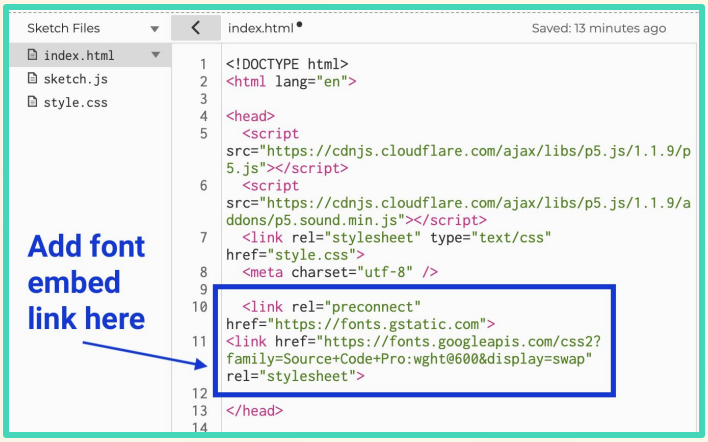
- ❏ **Add the embed link to your sketch's HTML file.** Find the small button under the play button that has a small arrow on it and click it.



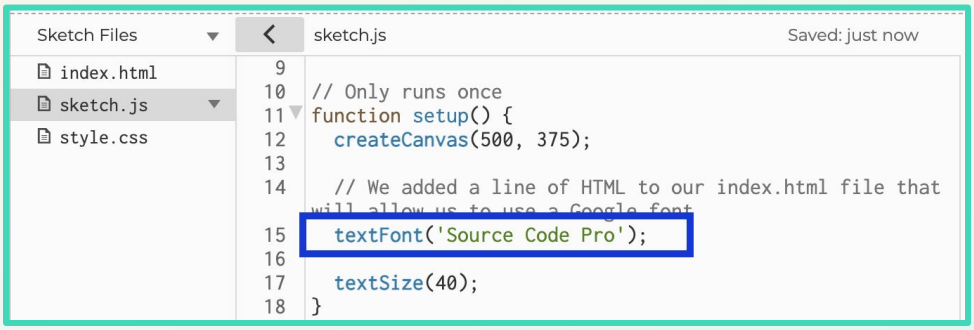


You should now see the three files that make up your p5.js sketch: an index.html file, a styles.css file, and a sketch.js file. So far, we have only been working in the sketch.js file, but to add our font, we will open the index.html file.

❑ **Click the index.html file.** Locate line 8 in the file that reads `<meta charset="utf-8" />` and paste your font embed link under that line of HTML.



❑ **Tell p5.js to use your font.** Now we need to go back to our sketch.js file and add a new function to activate the font. First, click the sketch.js file in the left sidebar. Next, use the `textFont()` function to display your new font. Include this new line of code inside `setup()`.



FUNCTION	DESCRIPTION
`textFont('font name');`	➔ `textFont`: The function name. *To learn more, see the `textFont()` entry in the 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. ➔ `'font name'`: Write the font name displayed on the Google Font webpage in single or double quotation marks. ➔ `,`: 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.

NOTE: The p5.js reference page will tell you to use the `loadFont()` function. Instead of that, we just added the font directly to our HTML.

- ❑ **Test it.** Press the play button to run your code and make sure that your font displays properly. If it doesn't, make sure you spelled the font name correctly and that your code is in the right location.

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!

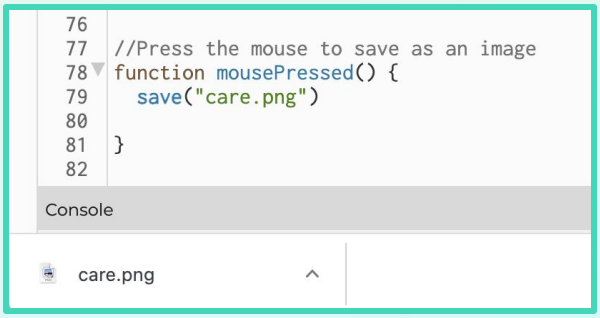
- ➔ Google Fonts
- ➔ `textFont()`
- ➔ Introduction to HTML

Here is a link to our solution code for extension 1. We recommend trying it yourself first and using this resource when you get really stuck or want to check your work.

Extension 2: Save your drawing (4-8 mins)

You might be saying to yourself, this is great and all, but how do I save my image to share it? Well, you can take a screenshot or you can get fancy with a new function: `mousePressed()`. This function runs the code inside it whenever you click the mouse inside the canvas area. Unlike our other functions, it goes outside of `setup()` and `draw()` and does not take any parameters.

- ❑ **Add the `mousePressed()` function.** Place it under the `draw()` function.
- ❑ **Tell p5.js to save your canvas.** To download our image, we can use the `save()` function with a filename and file extension. For example, we could use "myArt.png" or "digitalArtwork.jpg". Add the `save()` function with your file name and extension inside the `mousePressed()` function.
- ❑ **Test it.** Press the play button to run your code if it doesn't run automatically. Position your mouse inside the canvas and click. An image file with the name you specified should begin downloading to your machine.



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!

- ➔ `mousePressed()`
- ➔ `save()`
- ➔ Coding Train p5.js Tutorials **by Dan Shiffman**

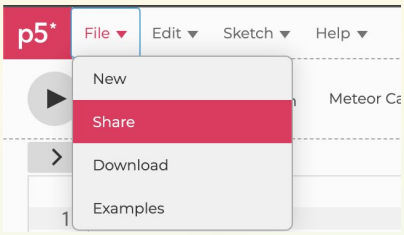
Here is a link to our solution code for extension 2. We recommend trying it yourself first and using this resource when you get really stuck or want to check your work.

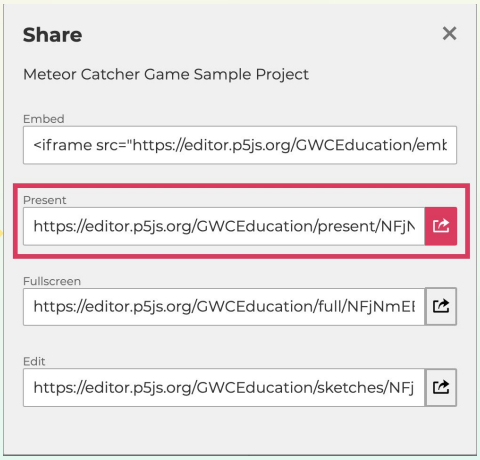
Step 7: Share Your Girls Who Code at Home Project! (5 mins)

We would love to see your work and we know others would as well. Share your sketch 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 activities!