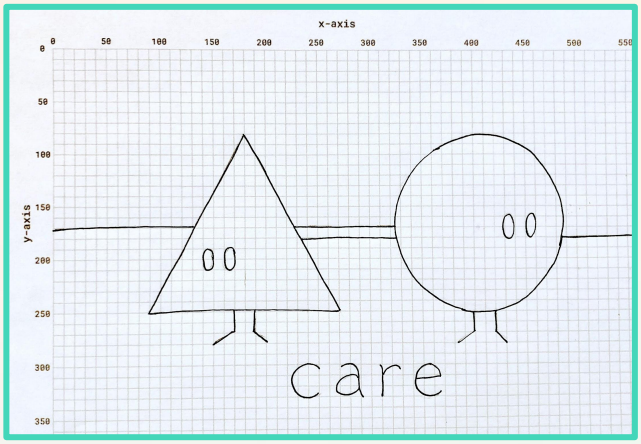
Girls Who Code At Home

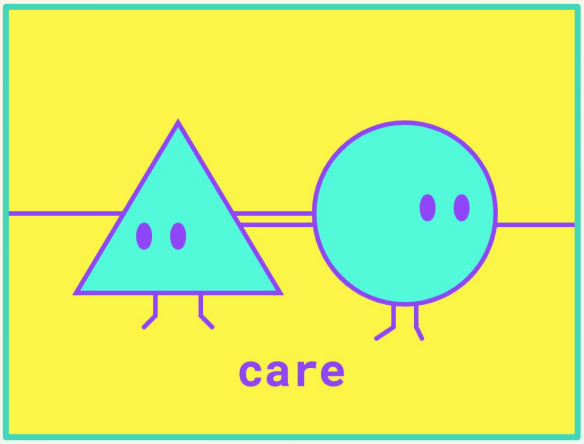
Digital Art Rules

Part 1

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 in your own words how to use rules as part of a creative process.
- ❏ plot shapes, lines, and text on a coordinate system.
- ❏ brainstorm and prototype a paper version of your digital artwork.

Materials

- ➔ Paper
- ➔ Pencil
- ➔ Markers, colored pencils, or crayons
- ➔ Timer (phone, clock, microwave, etc)
- ➔ Straight edge like a ruler or piece of paper
- ➔ Digital Art Rules Part 1 Reference Guide

Prior Knowledge

No prior knowledge is needed to complete this activity.

Women in Tech Spotlight: Ari Melenciano

Source: ITP NYU

Ari Melenciano is an artist, designer, creative technologist, researcher, and educator who is passionate about exploring the relationships between various forms of design and sentient (or sensorial) experiences.

Ari has been fascinated with technology since she was young, but it wasn't until she enrolled in NYU's Interactive Telecommunications Graduate Program (ITP) that she discovered how to merge her love of art and technology. With technology, she found a way to expand what was possible through art. While there she developed her art practice of combining electronics and new media art (i.e. art made using digital tools) through various projects like interactive sound sculptures and custom synthesizers she made with circuits. Her latest sound sculpture collection is called *Electro•Negro•Synesthesyo*. When she made them, "[Ari] was thinking about Black cultural artifacts that are often negatively stigmatized, but are very beautiful to [her], and imagined their presence in the future" (Source: Curbed).

During her time at ITP, she founded Afrotectopia, a social institution fostering interdisciplinary innovation at the intersections of art, design, technology, Black culture, and activism. It is a festival, an experimental school, a fellowship, and a space for people of color to create new media art, share resources, and imagine new futures for themselves. For Ari and her collaborators, code and design are tools to dream new possibilities outside of the societal systems that created racial disparity and denied access to Black people.

Check out this video (start at 1:00:39 to 1:05:00) to watch Ari discuss her work in her own words. Want to learn more? You can visit Ari's personal website or listen to this episode of the createCanvas podcast.

Reflect

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

PURPOSE

Ari uses art, design, and technology to imagine new worlds and futures where a long history of racial discrimination doesn't limit possibilities and access for Black people. What is a piece of technology or work of art you have seen that inspires you to dream of a more equitable world?

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

Step 1: Question the systems (8-10 mins)

Over this activity and the next, you will be working towards designing a piece of digital art using p5.js, a web-based coding environment that uses JavaScript. We will learn more about p5.js in Part 2. For now, let's concentrate on brainstorming your piece of art. But where should we begin? How should we start?

Let's do a short exercise to compare two approaches to drawing. First, we'll notice how it feels to make something when anything is possible. Next, we'll reflect on how it feels to make something when you have a few guidelines. This is just a warmup, so don't worry about being perfect. Instead, pay attention to how you feel as you complete each exercise.

Exercise 1 (2-3 mins)

Set a timer for **2 minutes**. You can sketch anything you want in the box below. GO!

Without thinking too much about it, take a minute to write down a few words that describe how you felt during those two minutes.

Remember: All feelings are valid. It's always important to be honest with yourself about how you feel in a creative process!

_____

_____

_____

Exercise 2 (2-3 mins)

Set a timer for **2 minutes**. Use the rules below to help you sketch. Again, this is practice - it doesn't have to be perfect. GO!

- ➔ Your theme is dream.
- ➔ Use 3 circles.
- ➔ Use 1 color.
- ➔ Use as many lines as you want.

Without thinking too much about it, take a minute to write down a few words that describe how you felt during those two minutes.

_____

_____

_____

Reflect (1-3 mins)

Compare your descriptions from Exercises 1 and 2. You can jot down notes if it's helpful, discuss with a friend, or simply think about each question individually.

- ➔ Did anything about one or both exercises surprise you?
- ➔ Was one easier or more enjoyable than the other? Why or why not?
- ➔ Was one more frustrating or confusing? Why or why not?

As you get to know yourself as an artist, designer, and programmer, it's just as important to think about *how* you create work as it is to brainstorm what you are making and why. For this activity, we are not going to give you a prompt for your piece of digital art. Instead, you will build your own set of design rules that will help you generate a piece of digital art: your own mini-world. This is one process you can use over and over again when you start a new project or when you are feeling a creative block.

We will also discuss the similar ways designers, artists, and programmers use rules in their projects or work. In fact, you'll discover a new world called **creative coding** where artists use programming as a tool to express themselves.

Step 2: Get inspired (15-17 mins)

What can art and design do? (3-4 mins)

Let's step back for a moment. Think about this question:

`What can art and design do?`

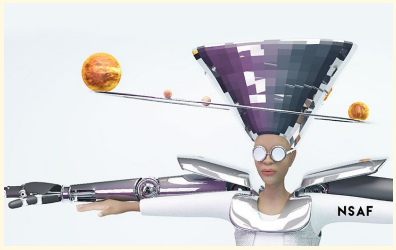
Art and design can help us solve problems, offer moments of beauty, spread messages or ideas, express emotions and identities, and more. They create opportunities for us to imagine new worlds. Check out a few examples of worlds the artists below have created:

Not the Only One by Stephanie Dinkins

A voice-interactive AI (artificial intelligence) that tells the memoir of one Black family across generations. Unlike many AI systems, the artist designed and trained this AI to make decisions based on the needs and ideals of Black and brown people. The memoir changes and evolves based on the data it receives.

Iyapo Repository by Salome Asega and Ayo Okunseinde

A collection of future artifacts created to affirm and project the future of people of African descent. The artists hold workshops in which they ask participants of African descent to generate ideas about the future they envision, then build models of their artifacts to add to a growing archive.



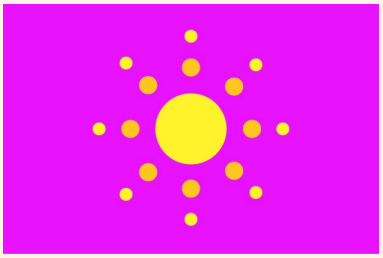
NeuroSpeculative AfroFeminism by Hyphen Labs

Uses VR to imagine a future where Black women pioneer brain research and cognitive improvement in a NeuroCosmetology lab, a reimagined Black hair salon.

Designers and artists use systems to make their work. Sounds cool, but vague. For this activity, instead we might ask, *what does it mean to draw within a system*? When we design or make art, we are making a set of intentional choices that reflect how we see or how we want to see the world. You could call these choices rules.

Visual artists and designers use a set of design elements like color, lines, shapes, text, and more to communicate an idea or feeling. They use rules to figure out how those elements work together to send the desired message or emotion. Let's examine a few digitally created images:

"This image feels a bit menacing to me. I think it is intense because the shapes and colors look like teeth."



"This image feels happy and bright to me because the circles look like a sun and the colors are warm and vibrant."

"This image feels calming like a sunset. I like how the triangles and lines create a horizon."

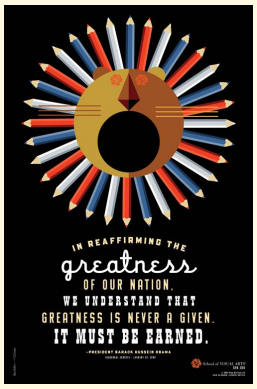
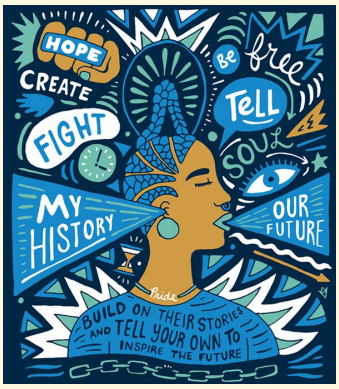
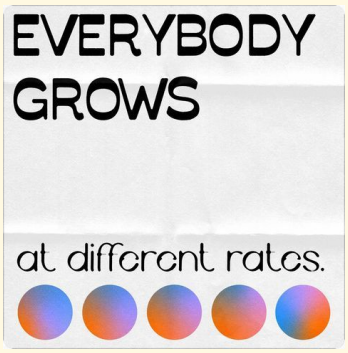
Sometimes, many experts agree on a set of rules that they think everyone should use because it creates a common language and set of expectations. Other times, communities and individuals will create their own rules to make a statement about new ideas or critique old processes. It's up to you as the artist and designer to decide which rules best fit the world you want to make, the story you are trying to tell, or the feeling you want to convey. In the next section, we'll check out how a few artists do this in their own work.

Analyze a piece of digital art (10-12 mins)

Let's explore some examples and analyze them as designed systems. Identifying **precedents** is a key part of the design process. Precedents are works other people have made that inspires or informs your own project. You might even find yourself using similar techniques or remixing their ideas, which is great! If you do this, remember to always attribute them by citing them as part of your project. *Remember: whether art or code, it's never cool to reproduce someone else's work and not give them credit.*

Choose a designer from the list on the next page that you like and analyze their work using the questions below the list. Keep in mind that the digital art we code in p5.js will be much simpler since this is a beginner activity. As you learn more about what you can do with that tool, you will be able to do much more - we promise!

Check the Reference Guide on pg 1-2 for an example.

Gail Anderson	Jen White Johnson	Andrea Pippins
		
Sophia Yeshi	**Kemunto**	**Charity Ekpo**
		

First, start from your overall impression or experience with the work. Set a timer for **3 minutes** then answer the questions below:

Briefly describe the piece in a couple sentences or bullet points. *What do you notice? What sticks out to you?*	
How does this work make you feel? *What emotions does it bring up?*	
What is the message? *What is it trying to communicate?*	

Next, set the timer for **4 minutes** to analyze the design elements:

What colors do they use? *What adjectives would you use to describe the color palette?*	
What shapes do they use? *Where are they in space? Are they in the corners or center? Are they touching too far apart? Which shapes are big? Which are small?*	
Do they use lines? *Are they thick or thin? Wavy or straight?*	
Do they use text? *If yes, how? For example, why do you think they chose that font? Is the text big or small?*	

Finally, set a timer for **2 minutes** and reflect on the system:

What are three rules this artist might have used to create this piece of work? *What guidelines do you think they followed?*	**1.**
	2.
	3.

All of the examples above were created by artists and designers who are Black and African American women or female-identifying. If you like their work, find them on IG and give them a shoutout! If you want to learn more about, amplify the work of, and/or participate in design and art communities of color, here are three resources to get started:

- ➔ The Black Experience in Graphic Design: 1968 and 2020. from Print Magazine
- ➔ Where Are All the Black Designers? A conference to celebrate Black voices in design and address pressing issues in the design industry. The site has free recorded talks.
- ➔ Blacks Who Design. Blacks Who Design highlights all of the inspiring Black designers in the industry. The goal is to inspire new designers, encourage people to diversify their feeds, and discover amazing individuals to join your team.

Step 3: Write your rules (8-10 mins)

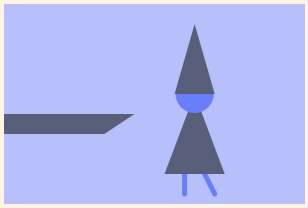
We are going to build a rule set using design elements, ideas, and emotions to guide your piece of digital art. Every designer - and programmer! - works within a set of rules to create a design or code a piece of software. Rules can be helpful because they help rein us in when the world feels too big (e.g. describe a dream without using words). Rules can also be oppressive when they prevent people from being able to express themselves (e.g. don't create art about your gender).

The rules we will use are meant to expand your creativity, not limit it. We will try to stick to the rules you choose for the rest of this activity, but if you want to change one later on, that's ok! Just keep in mind that you will have to code it later.

Observe rules in action (2-3 mins)

Check out the sketches below. List 1-2 rules that you observe at play in each sketch.

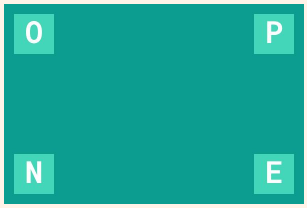
Example



Rules:

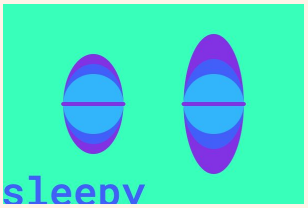
- ➔ 3 colors: gray, purple, and light purple
- ➔ 2 triangles
- ➔ 1 quadrilateral
- ➔ Feeling: anticipatory

Sketch 1



Rules:

Sketch 2



Rules:

Reflect on the following questions. You can write down your answers if it's helpful.

- ➔ Why did that rule stick out to you?
- ➔ Do you think that rules are helpful? Why or why not?

Create your rule set (6-7 mins)

Fill in the template below to create your rules. For now, we are limiting the rules based on what you will learn in Part 2. In the future, you can be much more free form with your creative rules!

Choose a keyword. *This can be a theme, idea, or message.*	
Choose an emotion. *How do you want someone to feel when they view your piece?*	

Choose up to 10 shapes from the following list. *You can use a shape multiple times or you don't have to use shapes at all. Keep in mind that you will have to program each one.*	____ Rectangle ____ Square ____ Circle ____ Oval ____ Triangle ____ Quadrilateral *A four-sided figure.*
Choose the number of lines you want to use, if any.	
Choose the words you want to include, if any. *It can be a sentence or one word or one word written multiple times.*	
Choose your colors. *We recommend sticking to about 3-5 colors. Try this color picking tool or a palette tool like Coolors.*	

Step 4: Brainstorm your design (7-10 mins)

Start sketching! (6 mins)

Time to put pencil to paper! In this step, you are going to use your rules to create three sketches in the boxes below. You will have up to two minutes to create each sketch. If you do not use all of your rules, that is OK! Remember: this is not about perfection - it is about the process. You will have a chance to refine one of them later on.

- ❏ Grab a pencil or pen and a timer.
- ❏ Review your rule set.
- ❏ Set a timer for **2 minutes**. You can draw the whole time, or stop when you're done.
- ❏ Start sketching in box 1!
- ❏ Repeat these steps for boxes 2 and 3.

Sketch 1 | Sketch 2 | Sketch 3

Refine one sketch (1-3 mins)

Pick the sketch you like the most. You can leave it as is or take a few minutes to refine it in the box to the right.

Phew, that was a lot of creative energy! Let's take a break to explore the rules we'll need as *programmers* when we translate our sketch to the screen in Part 2.

Step 5: Meet the grid (8-10 mins)

Let's say your friend gives you an instruction to draw a circle on a piece of paper. You could either just draw the circle or ask for more information. If you ask for more information, you might inquire: where on the paper? How big? What color? A perfect circle or more of an oval? To answer the question of where, your friend might say "A third of the way from the left side and three-fourths of the way down towards the bottom." That kind of instruction might work if you are talking to a human and don't need to be specific. But it won't work for a computer. Computers want you to be specific and precise with the rules you give them.

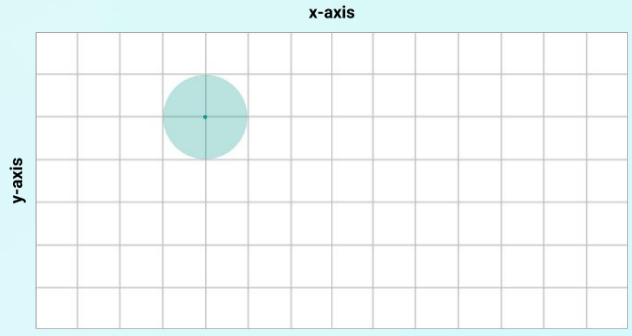
The coordinate system (3-4 mins)

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

Consider the coordinate plane to the right.

- ❏ What instructions would you give to a computer to draw the circle in that location?

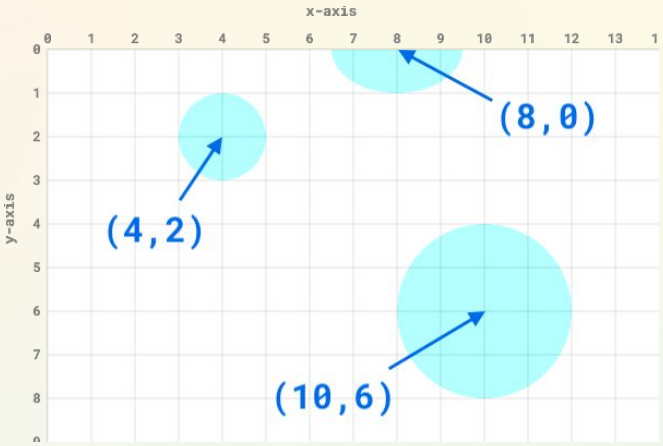
Check your ideas in the Reference Guide on pg 3.



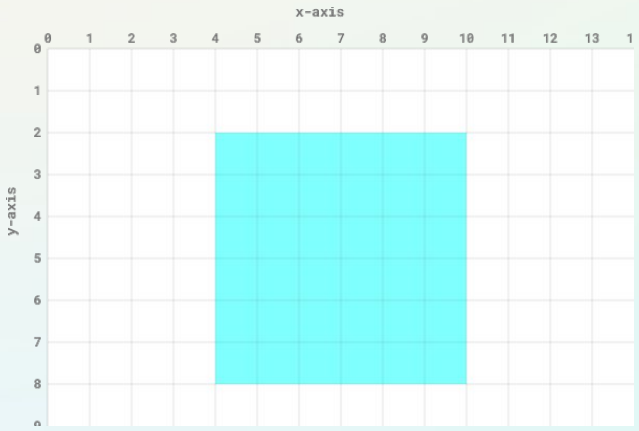
Pixel perfect (4-5 mins)

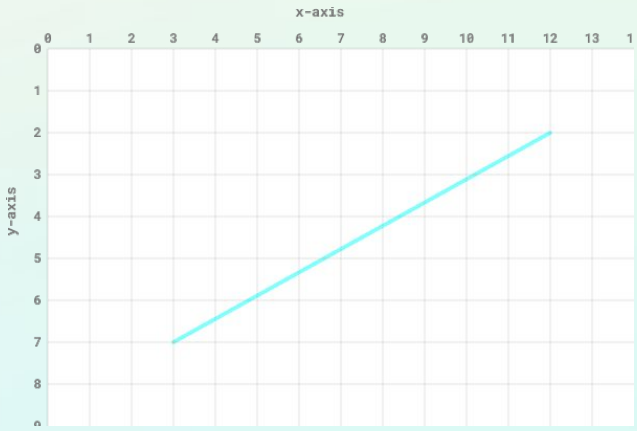
Each pixel on your screen has a unique address in the coordinate system. In order to draw pixels to the screen, we must give our program the x coordinate (i.e. the location on the x-axis) and the y coordinate (i.e. the location on the y-axis) of the pixel.

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



What information would you need to give a computer to display the following shapes? Write it in the box below each grid, then check your answers.

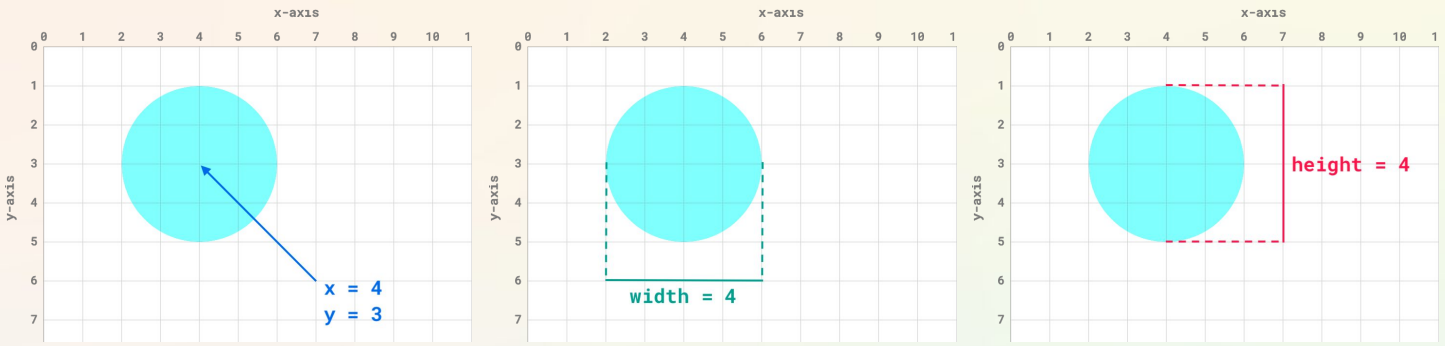




Check your code in the Reference Guide on pg 4.

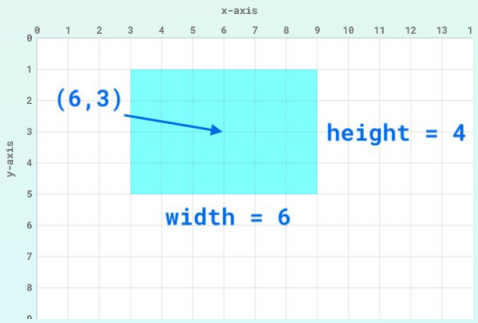
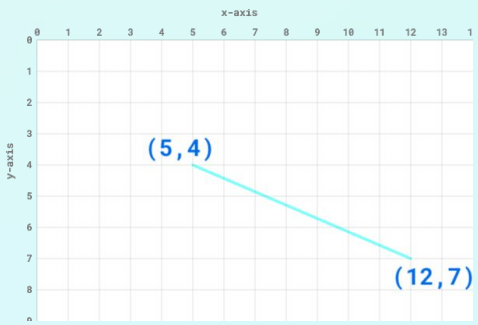
Step 6: Practice drawing shapes (7-9 mins)

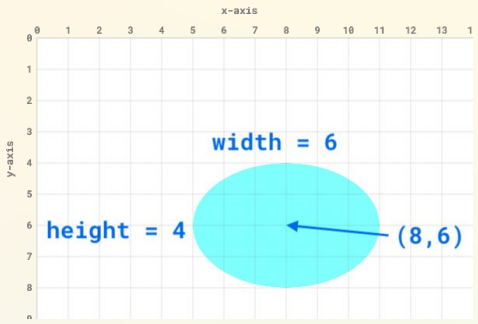
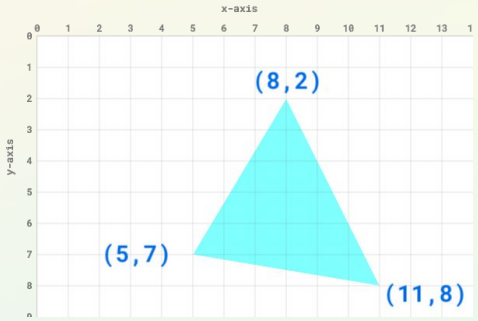
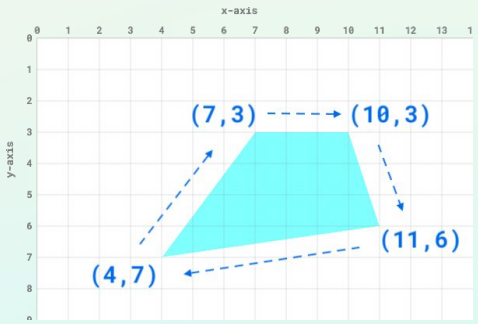
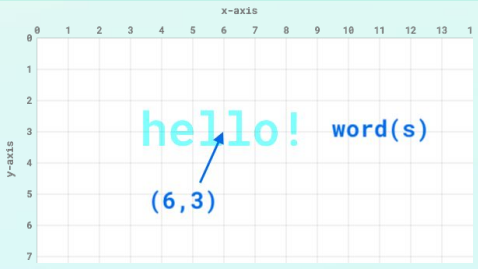
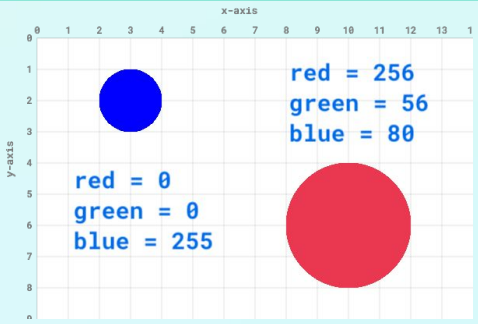
Luckily for us, the creators of p5.js included a set of pre-programmed instructions that we can use to perform specific tasks. This means that instead of drawing each pixel of a circle, we can use these pre-programmed instructions to do most of the work for us. We only need to give it a few key pieces of data: the x coordinate, the y coordinate, the width, and the height.



Shape reference (3-4 mins)

Review the values you need to know to draw the following shapes in p5.js. Keep these in mind as you translate your sketch to the grid in Step 6 and record the values for each shape in Step 7.

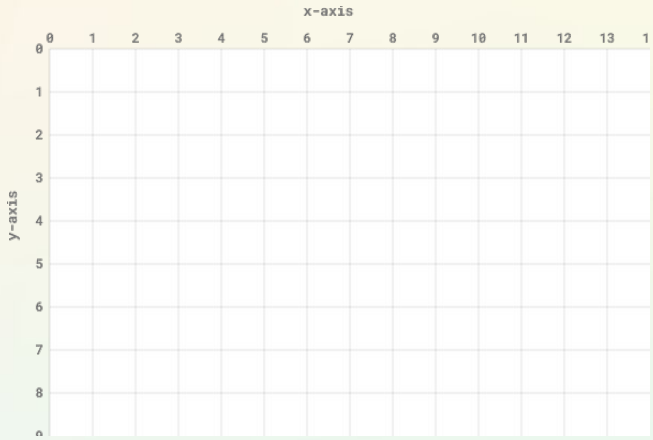
ELEMENT	VALUES	EXAMPLE
Line	➔ x and y coordinates of starting point ➔ x and y coordinates of ending point	
Rectangle or square	➔ x and y coordinates of the center point ➔ width ➔ height *Note: To create a square, the values of width and height should be equal.*	

ELEMENT	VALUES	EXAMPLE
Circle or oval	➔ x and y coordinates of the center point ➔ width ➔ height *Note: To create a circle, the values of width and height should be equal.*	
Triangle	➔ x and y coordinates of 1st point ➔ x and y coordinates of 2nd point ➔ x and y coordinates of 3rd point *Note: It doesn't matter which order.*	
Quadrilateral *A quadrilateral is a four sided shape.*	➔ x and y coordinates of 1st point ➔ x and y coordinates of 2nd point ➔ x and y coordinates of 3rd point ➔ x and y coordinates of 4th point *Note: You should list all points in a clockwise or counterclockwise motion.*	
Text	➔ word or words ➔ x and y coordinates to set the **center** of the text's location	
Color *We will use RGB color mode. It uses combinations of red, green, and blue light to create a range of digital colors.*	➔ red value from 0 to 255 (R) ➔ green value from 0 to 255 (G) ➔ blue value from 0 to 255 (B) *Note: Try this color picking tool or a palette tool like Coolors.*	

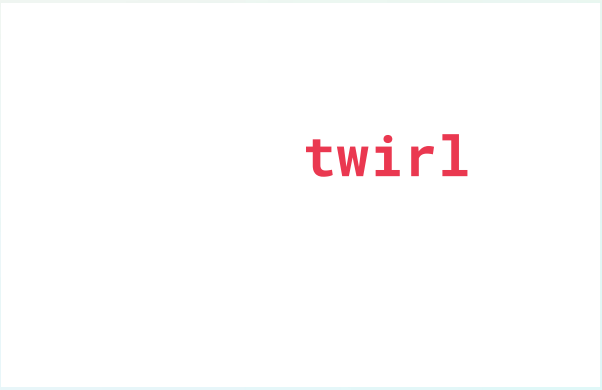
Practice (3-4 mins)

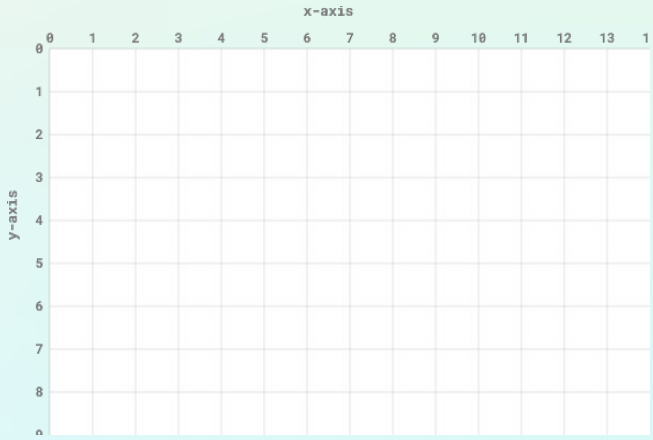
Use the information above to practice drawing shapes on a grid. You can estimate the location or you can be precise and use a ruler.

- ❏ **Draw the triangle on the grid.** First, label the triangle with the information you would need to give a program. Next, use that information to translate it to the grid.



- ❏ **Draw the text on the grid.** First, label the text with the information you would need to give a program. Next, use that information to translate it to the grid.

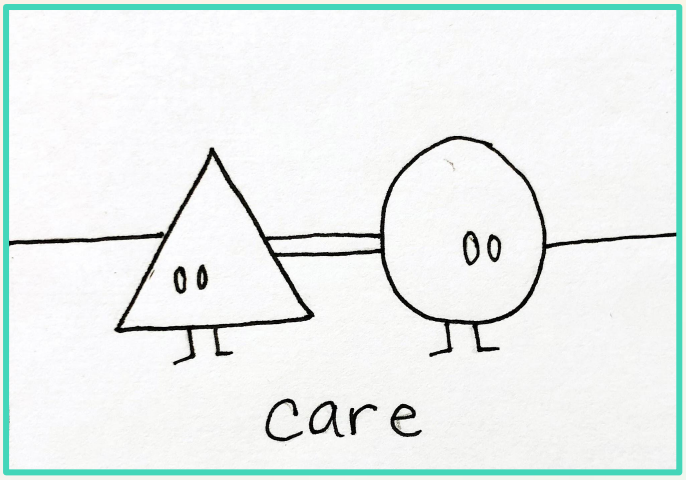




Check your code in the Reference Guide on pg 5.

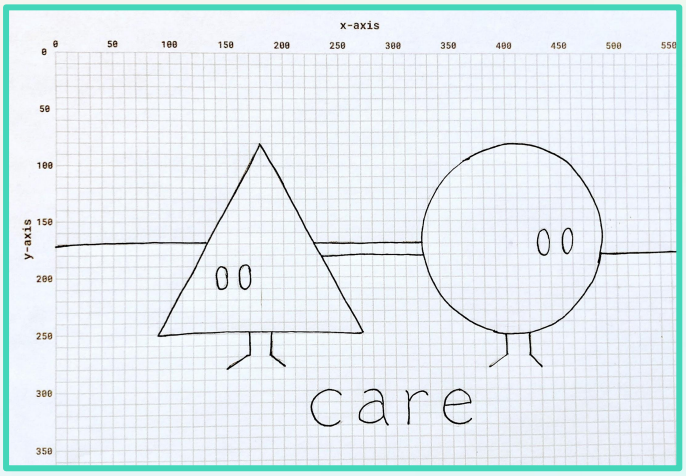
Step 7: Translate your sketch to the grid (5-15 mins)

Now that we've had some practice with the coordinate system, it's time to draw your sketch onto the grid on the next page. Keep in mind that it's ok if it's not exact! You can always make adjustments in Part 2. Right now, our goal is to get the basic layout.



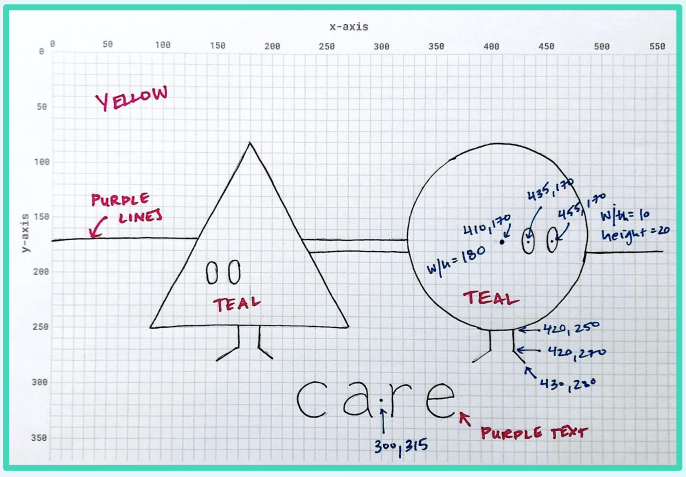
Estimate the elements' location.

First, think about the general location of each element. If you have one element in a central location, you can try starting with that one. This will make it easier to figure out the placement of the other elements.



Draw your elements.

Next, finish drawing your shapes, lines, and/or text on the grid. Plot the points of each shape, line, and/or text. You should also include any additional information you will need like width or height.



Add some color!

Finally, use your color palette to label each element with the color you want to use. If you have a wide assortment of markers or colored pencils, you can also just color it in!


x-axis
0
50
100
150
200
250
300
350
400
450
500
550
y-axis
0
50
100
150
200
250
300
350

Step 8: Write your instructions (5-7 mins)

We have all the information we need to start programming your piece of digital art! Before we break, let's take one final step to make our lives a bit easier later on when we translate it to p5.js. Record the data you need for each shape, line, and text with corresponding colors in the table below. Use the table in Step 6 if you need a refresher.

ELEMENT	VALUES	COLOR
Circle	*Center = (410, 170) width = 180 height = 180*	*Teal* *R = 82, G = 250, B = 218*

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

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

Stay tuned for Digital Art Rules Part 2!

Girls Who Code At Home

Digital Art Rules Part 1

Reference Guide

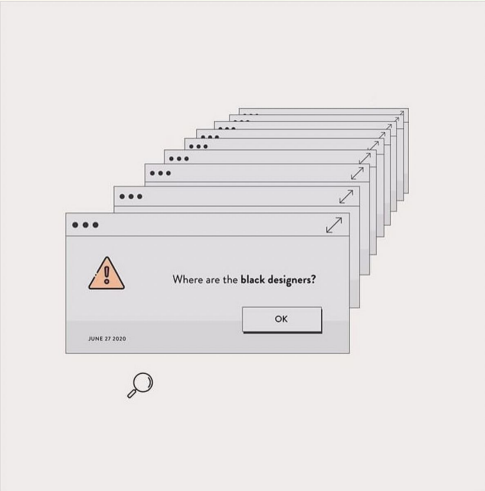
Digital Art Rules Part 1 Reference Guide

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

Step 2: Get inspired

Analyze a piece of digital art (10-12 mins)

If you are having trouble getting started, check out how we analyzed Samantha Morales' poster from *Where are all the Black designers?*



First, start from your overall impression or experience with the work. Set a timer for **3 minutes** then answer the questions below:

Briefly describe the piece in a couple sentences or bullet points. *What do you notice? What sticks out to you?*	These types of alert windows usually pop up when something has gone wrong on a computer. It also looks like a design from a much older operating system.
How does this work make you feel? *What emotions does it bring up?*	Frustrated, angry
What is the message? *What is it trying to communicate?*	The lack of Black designers is a problem that has been around for a while and we all need to be alerted to.

Next, set the timer for **4 minutes** to analyze the design elements:

What colors do they use? *What adjectives would you use to describe the color palette?*	Gray, black, yellow, tan
What shapes do they use? *Where are they in space? Are they in the corners or center? Are they touching too far apart? Which shapes are big? Which are small?*	They use a series of overlapping rectangles that get bigger as they move to the front or foreground. There are also circles in the left corner of each rectangle. The front rectangle has a triangle with an exclamation mark on the left and a smaller rectangle on the right.
Do they use lines? *Are they thick or thin? Wavy or straight?*	There are thin lines with arrows in the top right of each box.
Do they use text? *If yes, how? For example, why do you think they chose that font? Is the text big or small?*	Yes, they use text inside of the front box to ask a question: "Where are all the black designers?" The words black designers are in bold. There is also the work "OK" in the button box.

Finally, set a timer for **2 minutes** and reflect on the system:

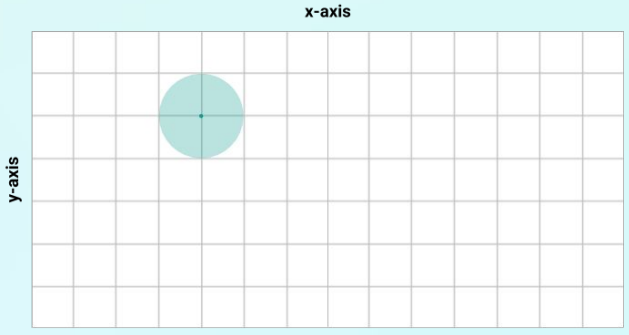
What are three rules this artist might have used to create this piece of work? *What guidelines do you think they followed?*	**1.** Use a neutral color palette (gray and tan) with one highlight color (yellow).
	2. Use rectangles, a triangle, and circles.
	3. Repeat the same shapes.

Step 5: Meet the grid

The coordinate system (3-4 mins)

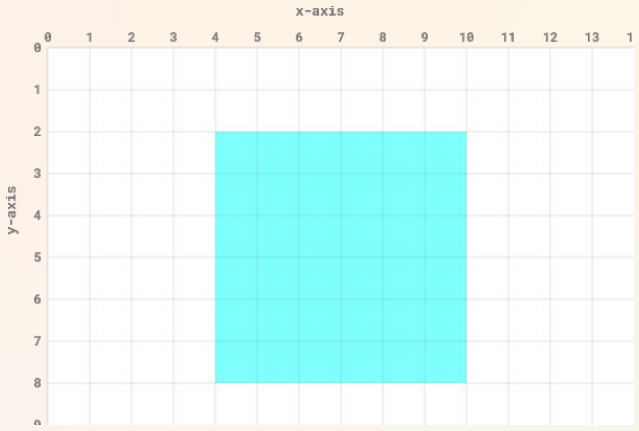
There are lots of different ways you can write this! Here are two versions:

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

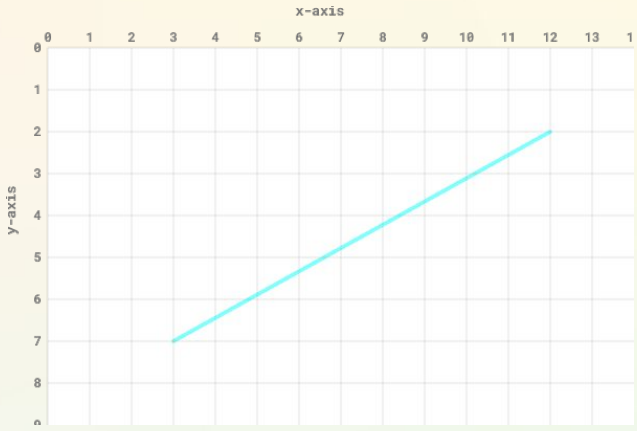


Pixel perfect (4-5 mins)

Here is the information would you need to give a computer to display the following shapes:



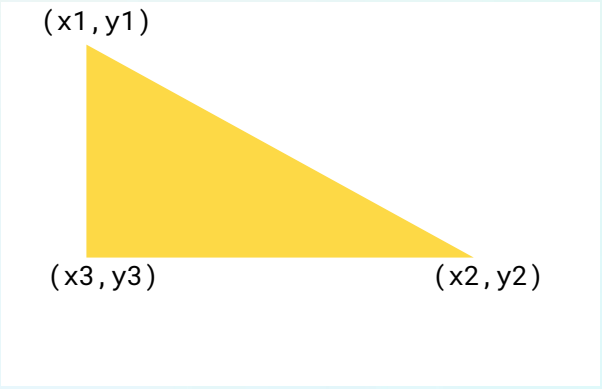
- ➔ x and y coordinates of the center point
- ➔ width
- ➔ height

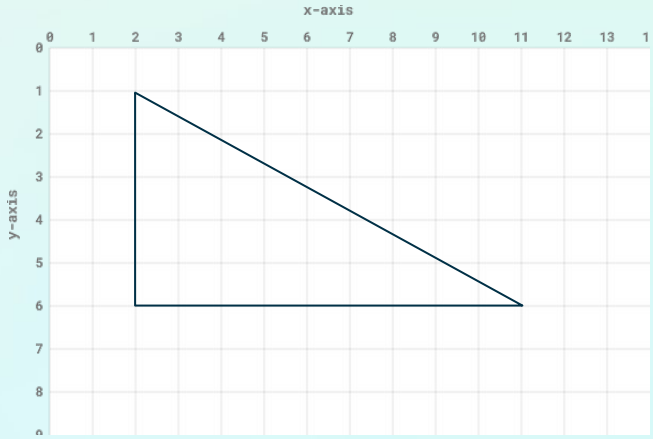


- ➔ x and y coordinates of starting point
- ➔ x and y coordinates of ending point

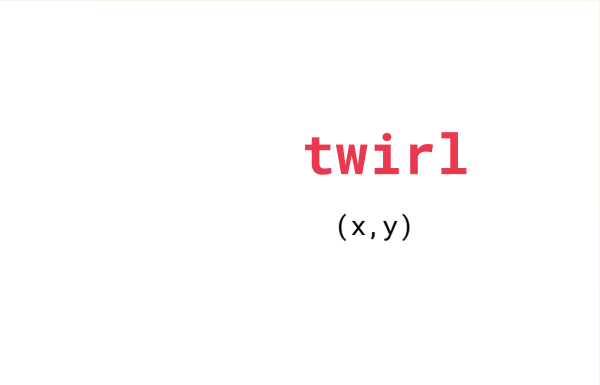
Practice (3-4 mins)

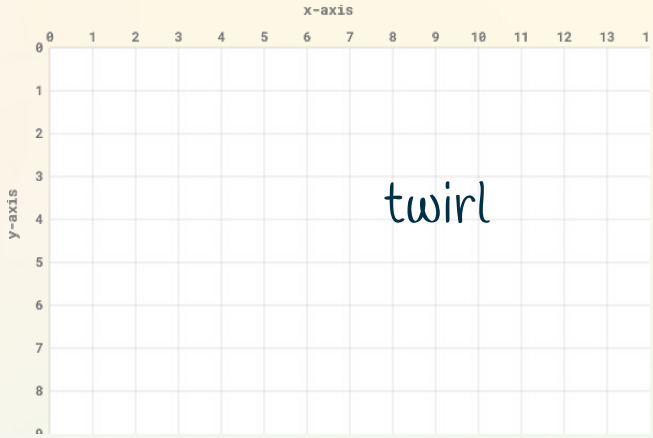
❏ **Draw the triangle on the grid.** First, label the triangle with the information you would need to give a program. Next, use that information to translate it to the grid.





❏ **Draw the text on the grid.** First, label the text with the information you would need to give a program. Next, use that information to translate it to the grid.





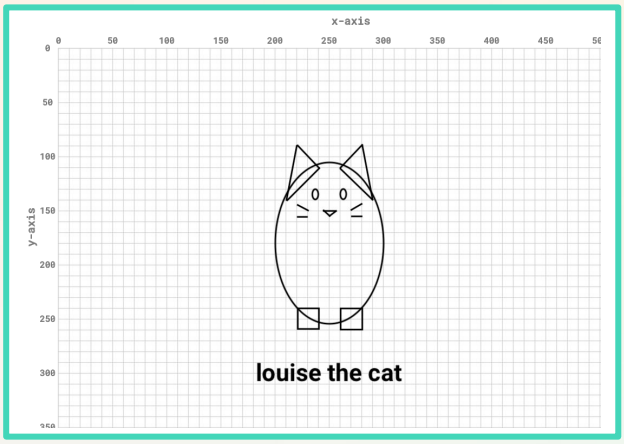
Girls Who Code At Home

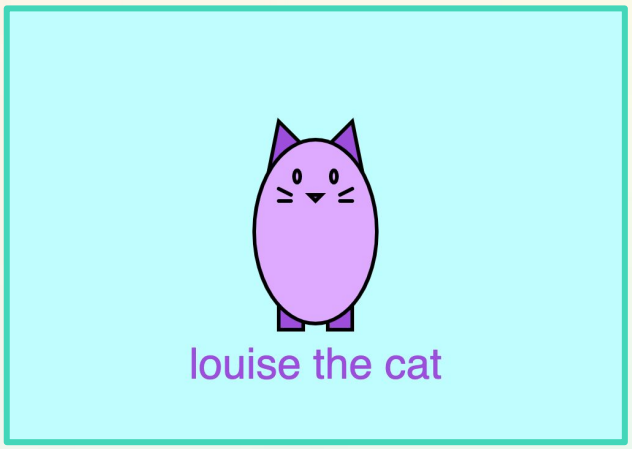
Digital Art Rules

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

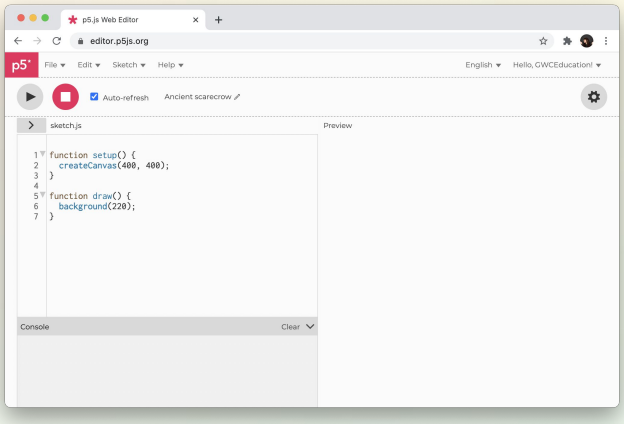
There were many African clothing retailers when Zuvaa entered the market. Despite the competition, what made Kelechi stick with her company?

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.

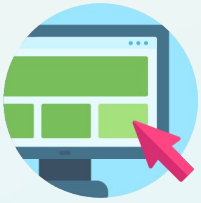
P5.js (or just p5) allows you to create interactive art for web browsers. It is a tool for **creative coding** - projects that use code for expression instead of just functionality. P5.js is a library for JavaScript, a programming language that allows you to add interactivity on the web. Being a library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



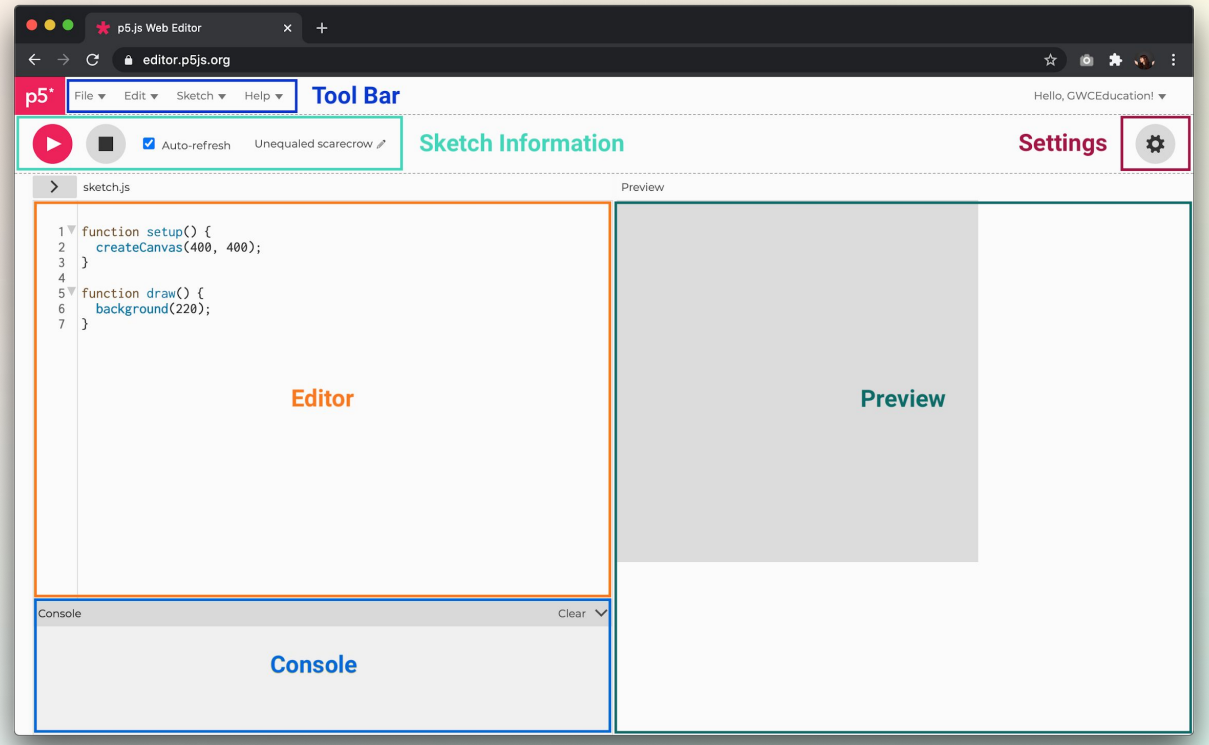
There are two ways you can use p5.js: the online web editor or a text editor and copy of p5.js that you download to your local computer. The easiest way to get started with p5 is the online editor. This allows you to write code and run your program in a web browser. In this tutorial, we are only going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

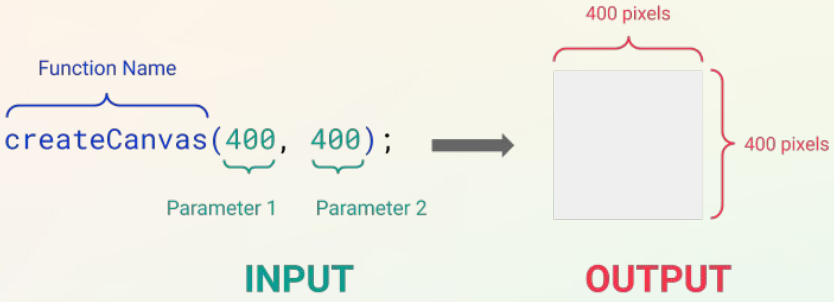
At the top of your sketch, you will include a comment that gives basic information about the sketch. Use **code comments** to remind yourself of how something works, the reasoning for a decision, or a follow up task.

- ➔ 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.



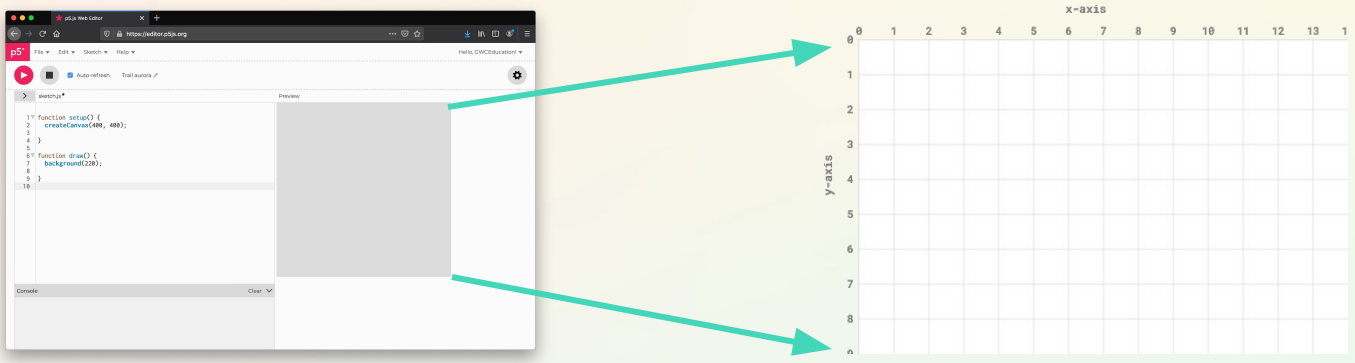
Set your canvas size (5-7 mins)

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.

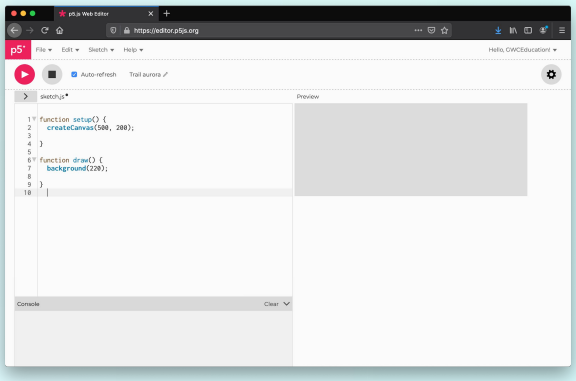


If you need a refresher on the coordinate system, see the Reference Guide pg 2.

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.

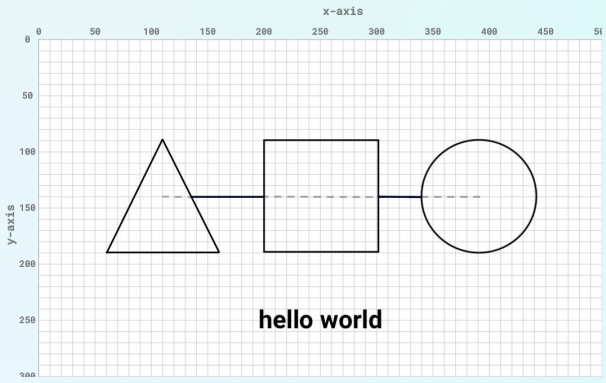
If you are only creating static or still sketches with no movement or interaction, then you can put all your code in `setup()`. But if you are animating a shape or want to listen for a mouse click, you will likely need to put most of your code in `draw()`. Since you will be using `draw()` a fair amount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

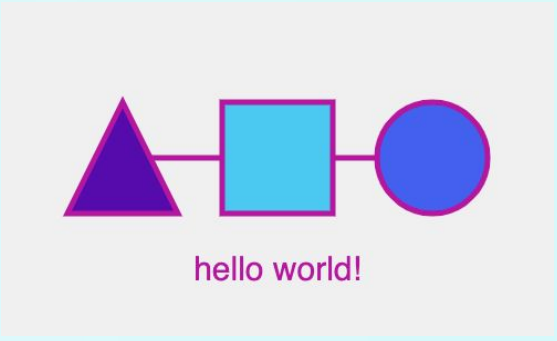
The last batch of knowledge you need to create your digital artwork are the functions that will render your design elements: shape, line, color, and text.

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

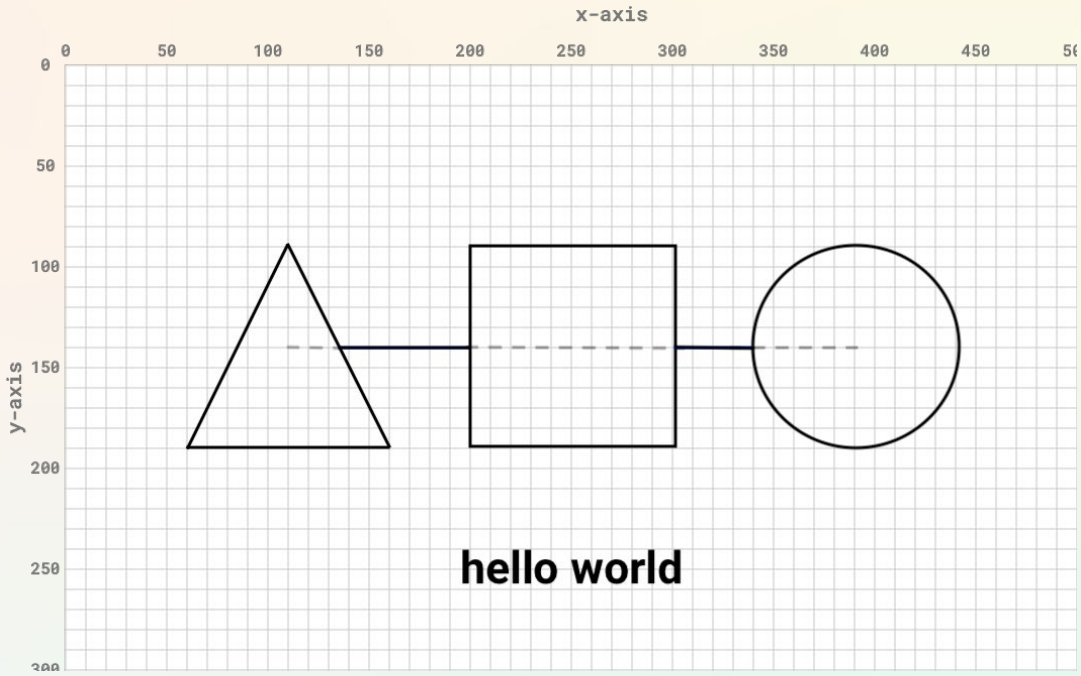


P5.js digital sketch

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.

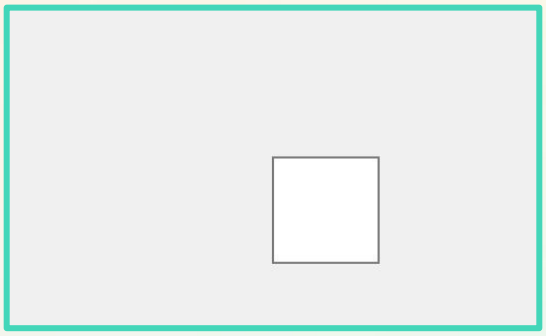
Use the instructions and the function reference table to add this line of code to your sketch:

- ❏ **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:

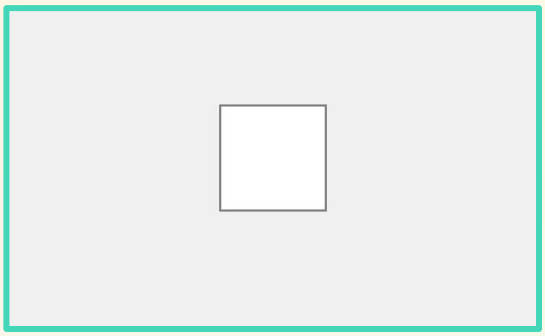


But it's not in the right location! By default, p5.js sets the location coordinates in the upper left corner, not the center. We can use the `rectMode()` function to change how p5 interprets the location coordinates we give the `rect()` function.

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.

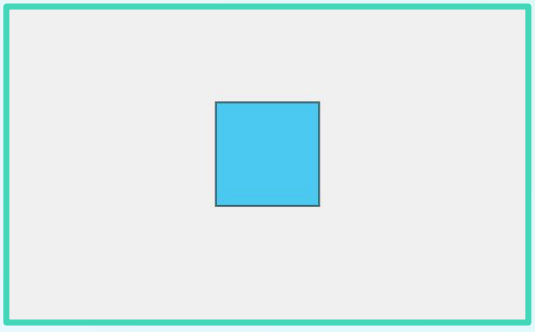
Use the instructions and the function reference table to add this line of code to your sketch:

- ❑ **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)

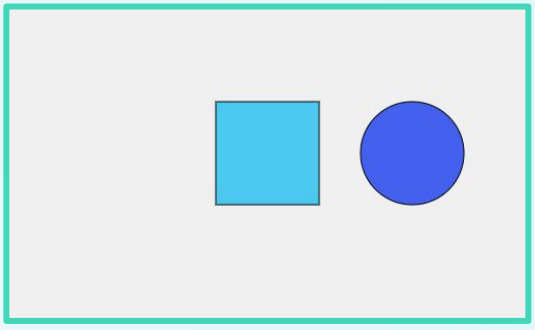
One shape down! Now onto the circle. Our star function here is the `ellipse()` function. It allows you to draw an ellipse - a fancy word for oval - on the canvas.

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.**

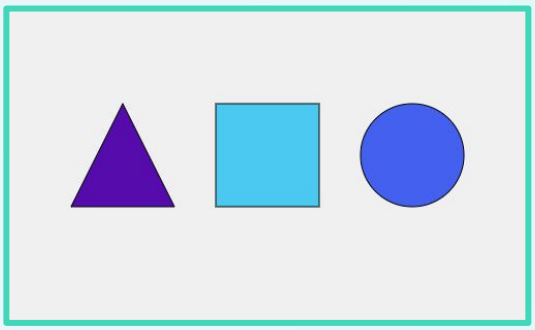
Instructions: Values

- ➔ 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.

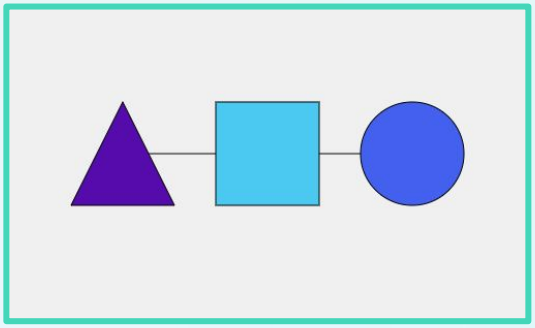
Use the instructions below and the function reference table to add this line of code to your sketch:

- ❑ **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.

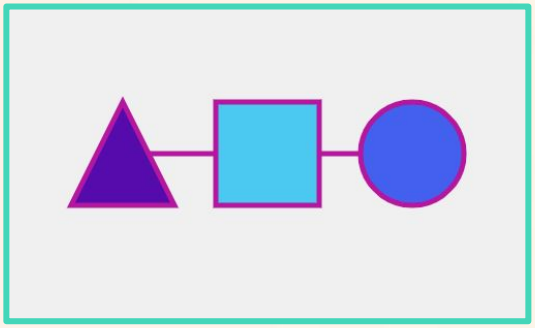
Use the instructions and the function reference table to add these lines of code to your sketch:

- ❑ **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.

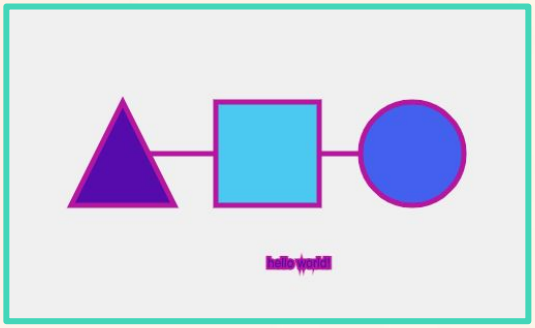
Use the instructions below and the function reference table to add this line of code to your sketch:

- ❑ **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.

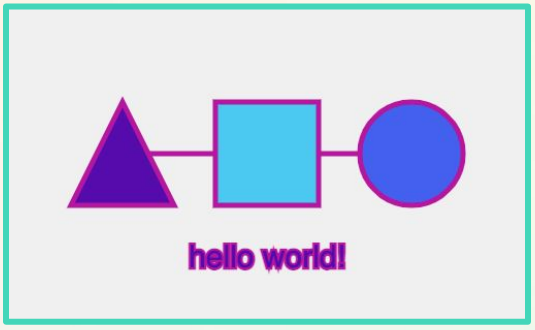
We have text on the screen, but that definitely does not resemble the text on our drawing. It's too small to read, off-center, has an outline, and is the wrong color. Let's tackle the size and alignment first. The `textSize()` function sets the size in pixels and `textAlign()` sets the text alignment.

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.

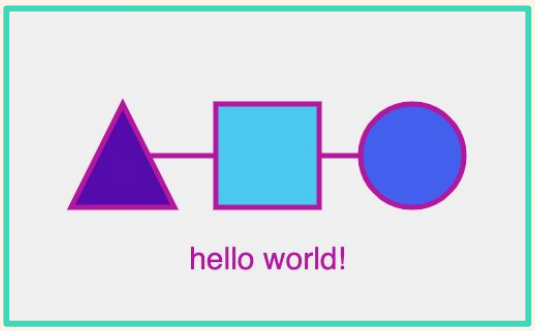
Right now the text takes the same color as the last `fill()` function we called for the triangle. We need to add another `fill()` function before `text()` to set it's color. Use the instructions and the function reference table to complete the following steps:

- ❑ **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:

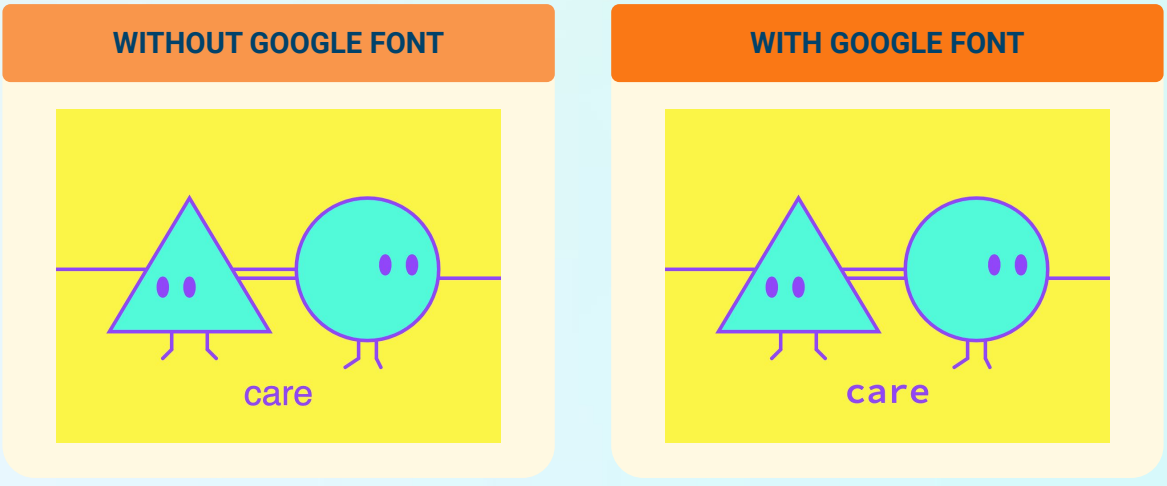
- ❏ **Is your code inside the correct curly brackets?**
- ❏ **Do you have semicolons at the end of each line of code?**
- ❏ **Did you spell the variable and function names correctly?** Remember that JavaScript is also case-sensitive!
- ❏ **Are your functions in the correct location and sequence?** Remember 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.

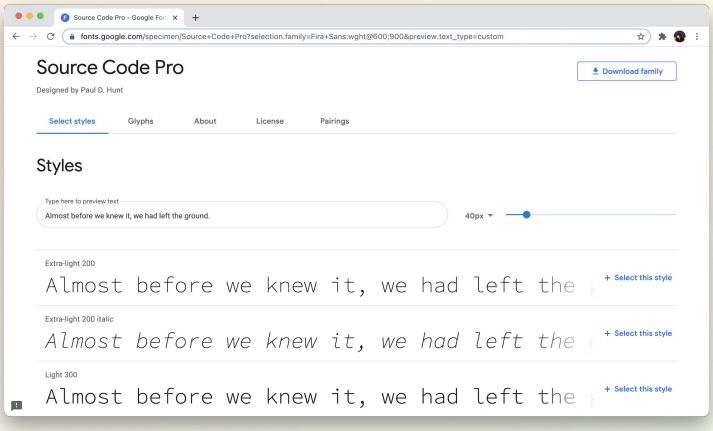
Step 7: Extensions (10-20 mins)

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.

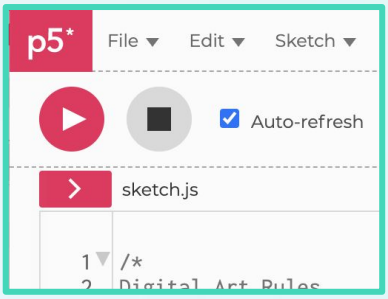


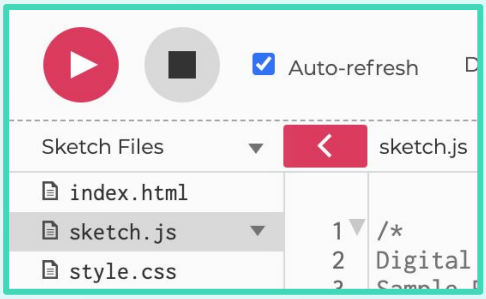
- ❏ **Select the styles.** You will notice that there are many different styles for each font. Click *Select this style* for the one you want to use.

- ❏ **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.

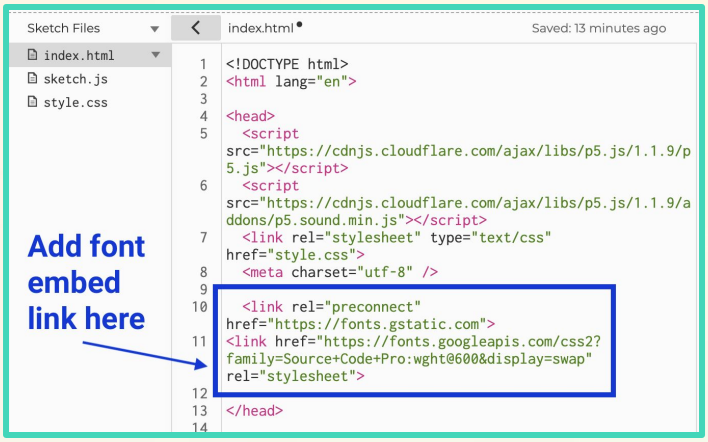
- ❏ **Add the embed link to your sketch's HTML file.** Find the small button under the play button that has a small arrow on it and click it.



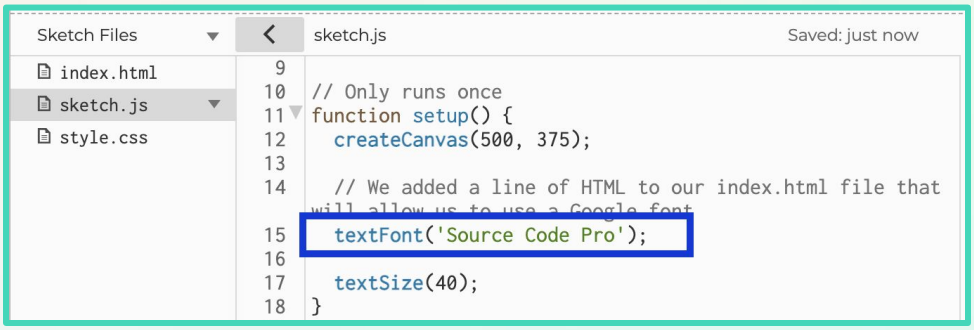


You should now see the three files that make up your p5.js sketch: an index.html file, a styles.css file, and a sketch.js file. So far, we have only been working in the sketch.js file, but to add our font, we will open the index.html file.

❑ **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!

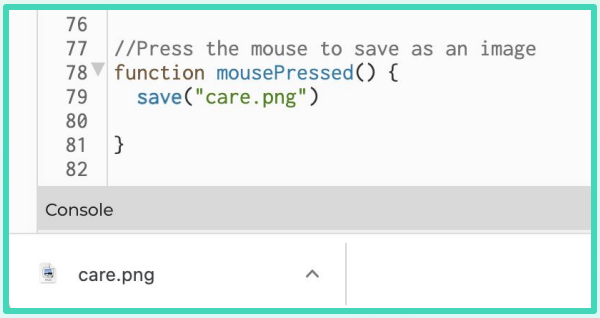
- ➔ Google Fonts
- ➔ `textFont()`
- ➔ Introduction to HTML

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**

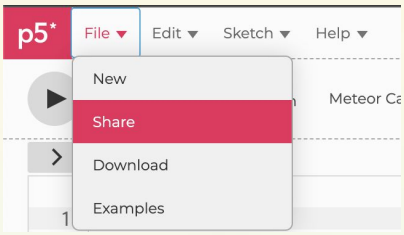
Here is a link to our solution code for extension 2. We recommend trying it yourself first and using this resource when you get really stuck or want to check your work.

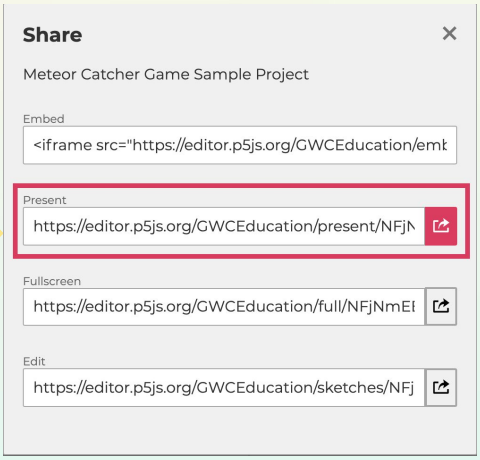
Step 7: Share Your Girls Who Code at Home Project! (5 mins)

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!

Girls Who Code At Home

Digital Art Rules Part 2

Reference Guide

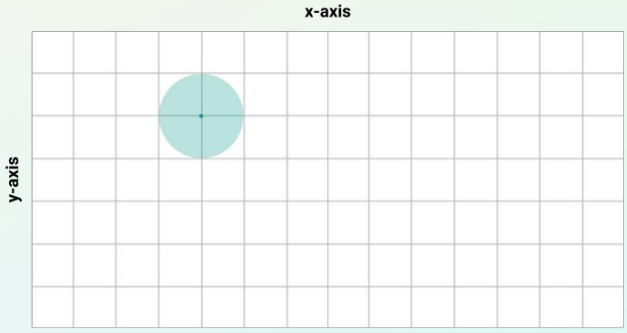
Digital Art Rules Part 2 Reference Guide

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

Step 3: Write your first function

Coordinate System Review

Let's say your friend gives you an instruction to draw a circle on a piece of paper. You could either just draw the circle or ask for more information. If you ask for more information, you might inquire: where on the paper? How big? What color? A perfect circle or more of an oval? To answer the question of where, your friend might say "A third of the way from the left side and three-fourths of the way down towards the bottom." That kind of instruction might work if you are talking to a human and don't need to be specific. But it won't work for a computer. Computers want you to be specific and precise with the rules you give them.



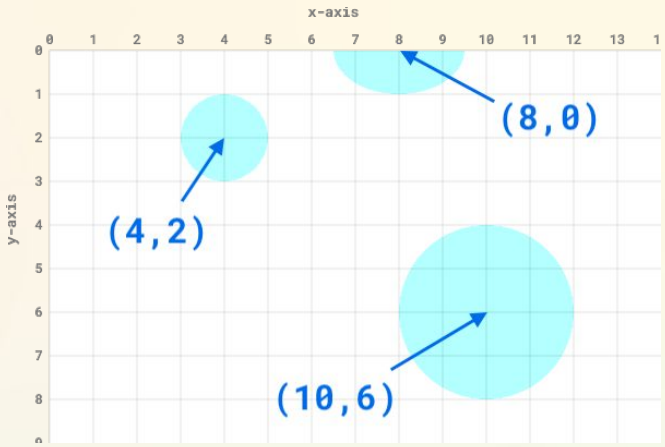
The coordinate system

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

Pixel perfect

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

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



Step 5: Practice translating an example image

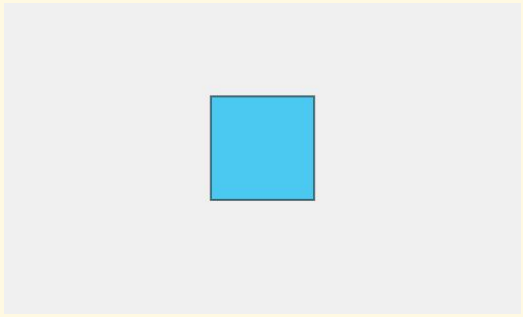
The new code added after each section is in **teal**.

Program the square

CODE

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

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

  // Draw the square
  rectMode(CENTER);
  fill(76, 201, 240);
  rect(250,140,100,100);
```

RESULT

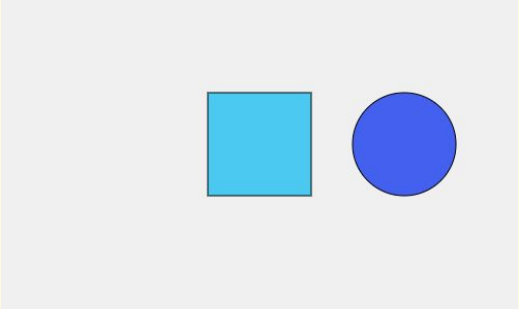


Program the circle

CODE

```
// Only draw() is included

function draw() {
  // Set the background
  background(240);

  // Draw the square
  rectMode(CENTER);
  fill(76, 201, 240);
  rect(250,140,100,100);

  // Draw the circle
  fill(67, 97, 238);
  ellipse(390,140,100,100);
```

RESULT

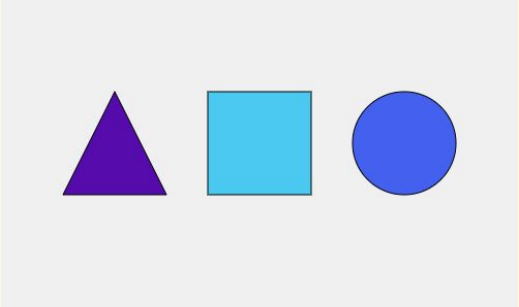


Program the triangle

CODE

```
// Only draw() is included

function draw() {
  // Set the background
  background(240);

  // Draw the square
  rectMode(CENTER);
  fill(76, 201, 240);
  rect(250,140,100,100);

  // Draw the circle
  fill(67, 97, 238);
  ellipse(390,140,100,100);

  // Draw the triangle
  fill(86, 11, 173);
  triangle(110,90,160,190,60,190);
```

RESULT

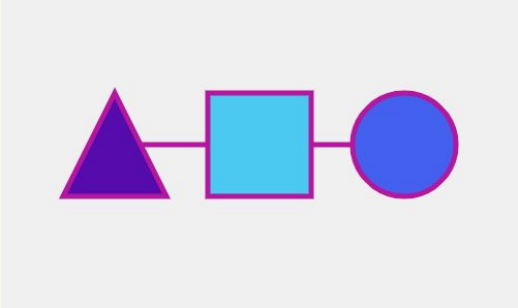


Program the line

CODE

```
// Only draw() is included

function draw() {
  // Set the background
  background(240);

  // Draw the line
  stroke(181, 23, 158);
  strokeWeight(5);
  line(110,140,390,140);

  // Draw the square
  rectMode(CENTER);
  fill(76, 201, 240);
  rect(250,140,100,100);

  // Draw the circle
  fill(67, 97, 238);
  ellipse(390,140,100,100);
```

RESULT

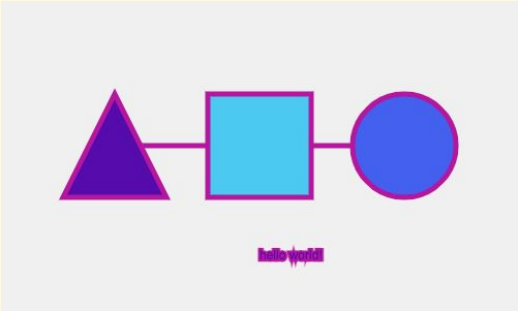


Program the text

CODE

```
// Only draw() is included

function draw() {
  // Set the background
  background(240);

  // Draw the line
  stroke(181, 23, 158);
  strokeWeight(5);
  line(110,140,390,140);

  // Draw the square
  rectMode(CENTER);
  fill(76, 201, 240);
  rect(250,140,100,100);

  // Draw the circle
  fill(67, 97, 238);
  ellipse(390,140,100,100);

  // Draw the triangle
  fill(86, 11, 173);
  triangle(110,90,160,190,60,190);

  // Draw text
  text("hello world!",250,250);
}
```

RESULT

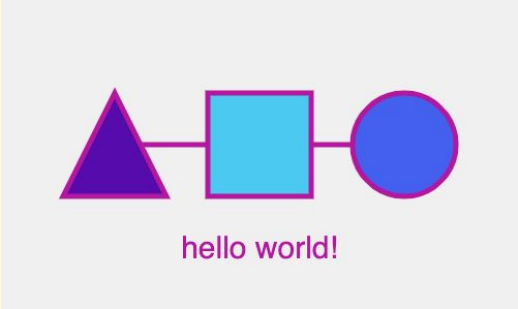


Program the text (cont.)

CODE

```
// Only draw() is included

function draw() {
  // Set the background
  background(240);

  // Draw the line
  stroke(181, 23, 158);
  strokeWeight(5);
  line(110,140,390,140);

  // Draw the square
  rectMode(CENTER);
  fill(76, 201, 240);
  rect(250,140,100,100);

  // Draw the circle
  fill(67, 97, 238);
  ellipse(390,140,100,100);

  // Draw the triangle
  fill(86, 11, 173);
  triangle(110,90,160,190,60,190);

  // Remove outlines from this
  // point forward
  noStroke();

  // Draw text
  fill(181, 23, 158);
  textSize(30);
  textAlign(CENTER);
  text("hello world!",250,250);
}
```

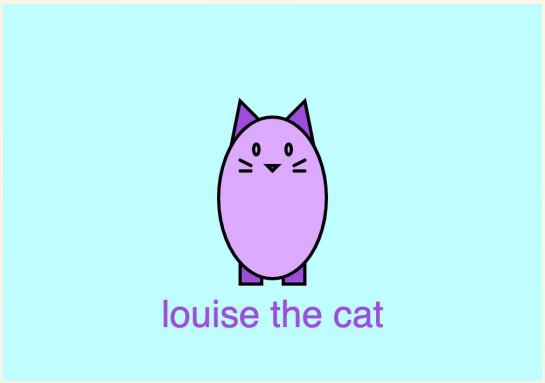
RESULT



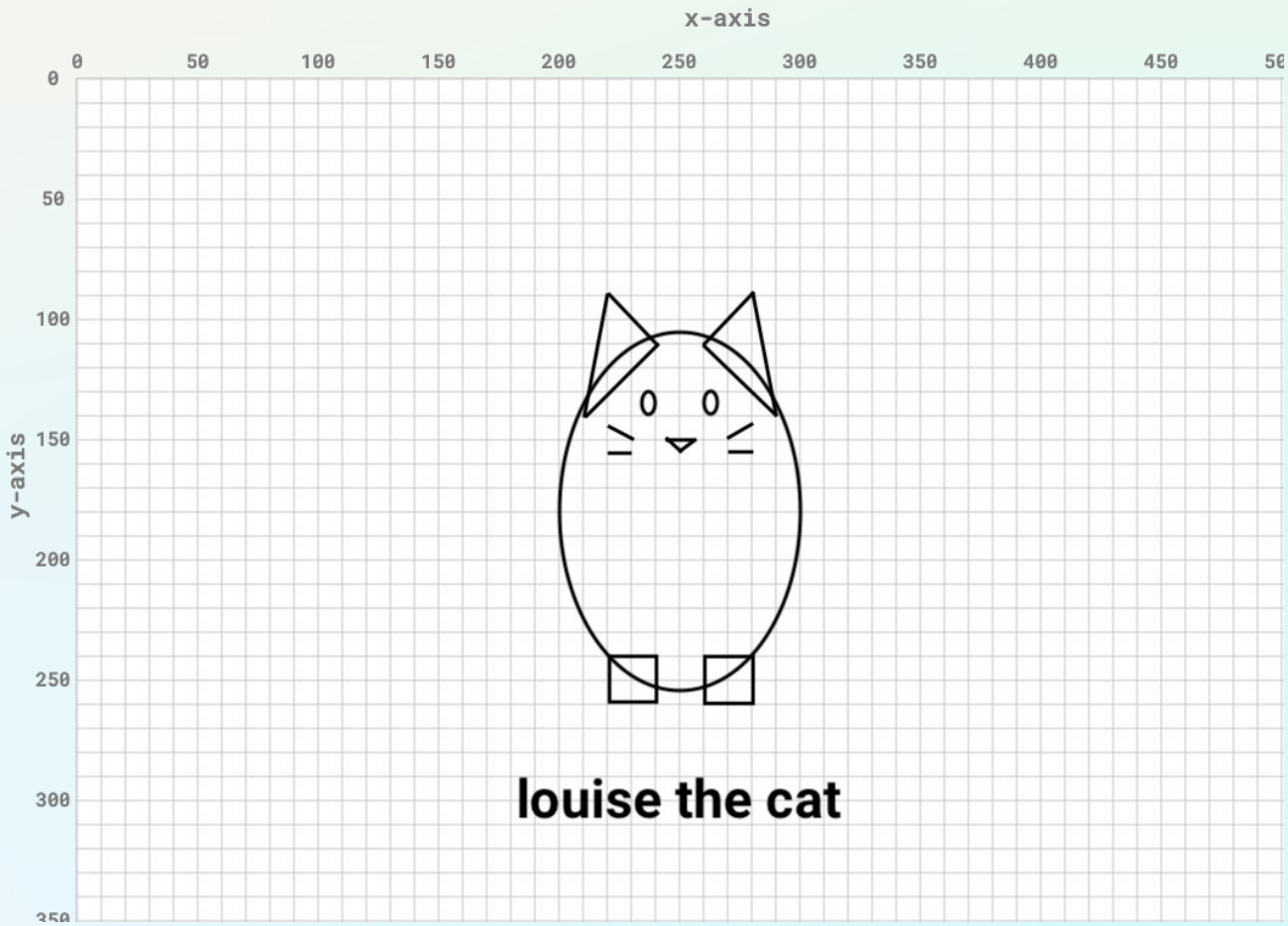
Step 6: Translate your drawing to p5.js

Sample Project: Cat (Easy)

If you don't have a drawing or instructions from Part 1, you can use this sample project to complete Part 1. This sample is easier as it includes fewer elements and none of the extensions.



Coordinate Plane Drawing



Instruction Set

ELEMENT	VALUES	COLOR
Background		Light teal: R = 192, G = 253, B = 255
Ellipse	Center = (250,180) Width = 100 Height = 150	Light purple: R = 222, G = 170, B = 255
Left triangle	Point 1 = (260,110) Point 2 = (280,90) Point 3 = (290,140)	Dark purple: R = 157, G = 78, B = 221
Right triangle	Point 1 = (240,110) Point 2 = (220,90) Point 3 = (210,140)	Dark purple
Small ellipse (left)	Center = (235,135) Width = 5 Height = 10	Light purple
Small ellipse (right)	Center = (265,135) Width = 5 Height = 10	Light purple
Small triangle	Point 1 = (245,150) Point 2 = (255,150) Point 3 = (250,155)	Dark purple
Top line (left)	Point 1 = (230,150) Point 2 = (220,145)	Black
Bottom line (left)	Point 1 = (230,155) Point 2 = (220,155)	Black
Top line (right)	Point 1 = (270,150) Point 2 = (280,145)	Black
Bottom line (right)	Point 1 = (270,155) Point 2 = (280,155)	Black
Text	Text = “louise the cat” Center = (250,300)	Dark purple

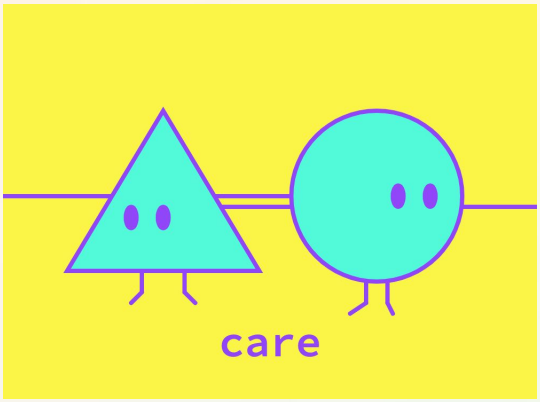
Sample Code

You should only examine this code ***after*** completing the activity.

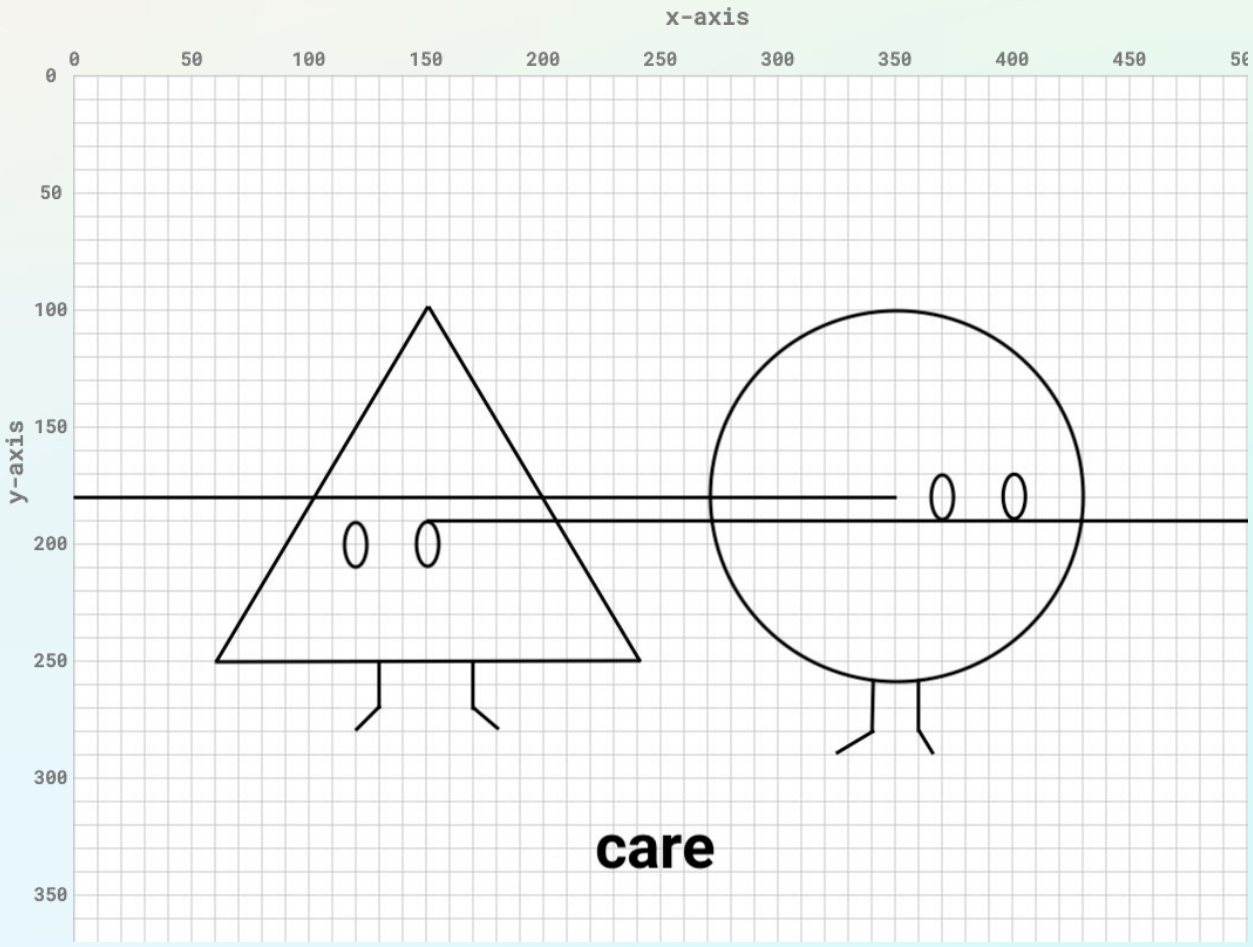
➔ Digital Art Rules Sample Project - Cat

Sample Project: Care (Challenging)

If you don't have a drawing or instructions from Part 1, you can use this sample project to complete Part 2. This sample is more challenging as it includes more elements and the extensions.



Coordinate Plane Drawing



Instruction Set

ELEMENT	VALUES	COLOR
Background		Yellow: R = 250, G = 245, B = 70
Top horizontal line	Point 1 = (0,180) Point 2 = (350,180)	Purple: R = 145, G = 70, B = 250
Bottom horizontal line	Point 1 = (150,190) Point 2 = (550,190)	Purple
Triangle	Point 1 = (110,90) Point 2 = (160,190) Point 3 = (60,190)	Teal: R = 82, G = 250, B = 218
Left ellipse (Triangle)	Center = (120,200) Width = 10 Height = 20	Purple
Right ellipse (Triangle)	Center = (150,200) Width = 10 Height = 20	Purple
Left line (Triangle leg)	Point 1 = (130,250) Point 2 = (130,270)	Purple
Left line (Triangle foot)	Point 1 = (130,270) Point 2 = (120,280)	Purple
Right line (Triangle leg)	Point 1 = (170,250) Point 2 = (170,270)	Purple
Right line (Triangle foot)	Point 1 = (170,270) Point 2 = (180,280)	Purple
Circle	Center = (350,180) Width = 160 Height = 160	Teal
Left ellipse (Circle)	Center = (370,180) Width = 10 Height = 20	Purple
Right ellipse (Circle)	Center = (400,180) Width = 10 Height = 20	Purple
Left line (Circle leg)	Point 1 = (360,260) Point 2 = (360,280)	Purple
Left line (Circle foot)	Point 1 = (360,280) Point 2 = (365,290)	Purple
Right line (Circle leg)	Point 1 = (340,260) Point 2 = (340,280)	Purple
Right line (Circle foot)	Point 1 = (340,280) Point 2 = (325,290)	Purple
Text	Text = “care” Center = (250,330)	Purple

Sample Code

You should only examine this code ***after*** completing the activity.

➔ Digital Art Rules Sample Project - Care (with Extensions)

Function Reference

SHAPES	
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.
`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.
`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.

Function Reference (cont.)

SHAPES (CONT.)	
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.
`quad(x1, y1, x2, y2, x3, y3, x4, y4);` *Note that the points should be listed clockwise or counterclockwise*	➔ **`quad`**: The function name. To learn more, see the *`quad()` 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. ➔ **`x4`**: The x-coordinate of the fourth point. ➔ **`y4`**: The y-coordinate of the fourth 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.

Function Reference (cont.)

LINE	
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.
`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.
`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.

Function Reference (cont.)

COLOR	
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.
`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.
`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.

Function Reference (cont.)

TEXT	
FUNCTION	**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.
`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.