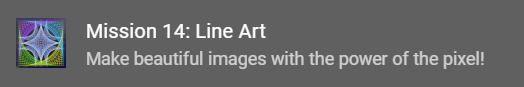
**Mission 14:**

**Line Art**

**Student Workbook**





**Digital Artistry**

In this mission you will discover the magic of computer graphics. You will use loops to create beautiful and interesting art.

Go to the Mission 14 Log and fill out the Pre-Mission preparation.

* In previous missions, you learned how to draw with lines, circles and rectangles. How do you think you can use loops to create art?

**Mission 14: Line Art**



**Digital Artistry**

This mission will lead you on a journey   
to discover the magic of computer graphics.

You will make beautiful visual art with just a few lines of code.

**Pixel Power**

It all starts with a pixel drawn on the screen. But as you’ve seen, things get much more interesting when you loop your code to create patterns of logic, sounds and light!

**for the Win!**

As you complete this mission you will gain a   
mastery of the **for** loop, a versatile tool to   
have in your coding toolbox.

* Ready to *visualize* a range of colorful pixels streaming across your LCD screen?

**Mission 14: Get started**

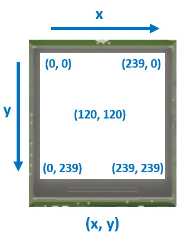
* Go to <https://make.firialabs.com/> and log in.



* Go to Mission 14



* Click and start Mission 14.

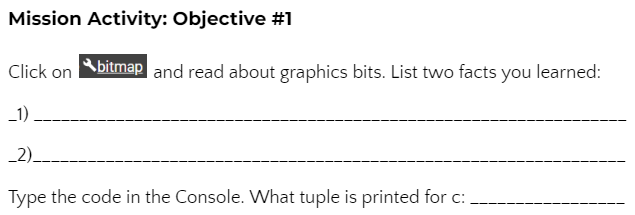
**Objective #1: Pixel Power**

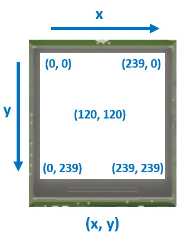
You have already drawn on the screen using bitmap functions.

* display.draw\_rect()
* display.draw\_text()
* display.draw\_circle()
* display.draw\_line()

**DO THIS:**

Click on

Go to your Mission Log and write down two facts about Bitmaps.

**Objective #1: Pixel Power**

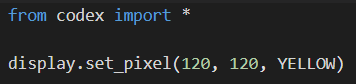
Another basic graphic is the single pixel,   
which other shapes are made from:

The CodeX LCD screen contains 240x240   
pixels.

* 240 pixels from left to right: x = 0 to 239
* 240 pixels from top to bottom: y = 0 to 239

**Setting Pixels**

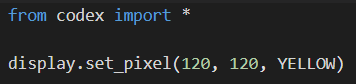
You can set (or draw) a pixel anywhere on the screen.

This code draws a YELLOW pixel at x=120 and y=120:

**Objective #1: Pixel Power**

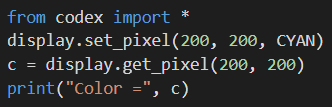
**DO THIS:**

* Create a new file named **PixelPlay**
* Type in the code below.
* Then **draw 6 more pixels** at different locations on the screen with different colors.
  + Guess where they should appear before running the code.
  + Are they where you expected them to be?
* Run the code.



**Objective #1: Pixel Power**

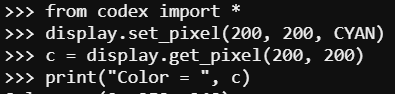
**Reading Pixels**

* Your code can write pixel colors to the display, but it can also read them back!
* The following code sets a pixel on the screen, and then reads the tuple of the color.



**DO THIS:**

* Open the Console
* Type in the 4 lines of code directly into the Console (not your program)
* You know from working with RGB colors that CYAN is (0, 255, 255). Is this the tuple that was printed?
* Go to your **Mission Log** and write down the tuple value of **c**.
* Then you can close the Console.





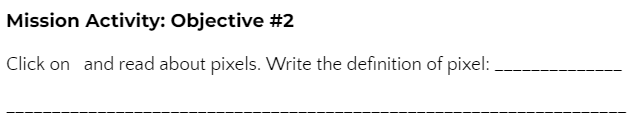
**Mission Quiz: Pixel Basics**

Test your skills by **taking the quiz**.

**Objective #2: Line Up!**

**DO THIS:**

* Click on to add it to your toolbox.
* Go to your Mission Log and write the definition of “pixel”.

****

**Objective #2: Line Up!**

Now that you have mastered pixels …

* Seriously, you pretty much know all there is to know about them!

**Line Up!** What do you call a bunch of pixels in a row?

Wait for it ….. **A LINE!**

So what are you waiting for?

* The LCD is 240 pixels wide.
* Just copy and paste display.set\_pixel() 240 times, right?

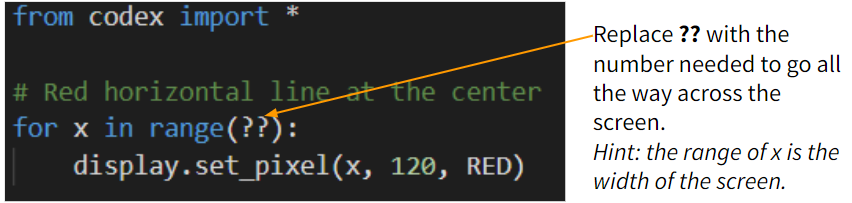
Don’t you dare! That is what loops are made for!



**DO THIS:**

Delete ALL the set\_pixel() commands. Replace them with a loop.

* A single lovely for loop is all you need to achieve the goal!
* Run the code.



**Objective #3: Two Axes to Grind**

**Add a Vertical Axes**

* You have a nice ***horizontal*** line. Adding a ***vertical*** line to match will create a perfect reference for drawing additional line art.

**Getting Centered**

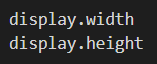
* Your code currently uses “magic numbers” like 240 for the display width, and 120 for the center.

**What is a “magic number”?**

* A number that just appears in code with no explanation.
* Other programmers may have no idea what they mean.
* And if something changes, like a size, the number won’t work and must be changed.

You can eliminate a magic number by using a ***variable*** or ***constant*** instead.

**Avoiding a “magic number”**

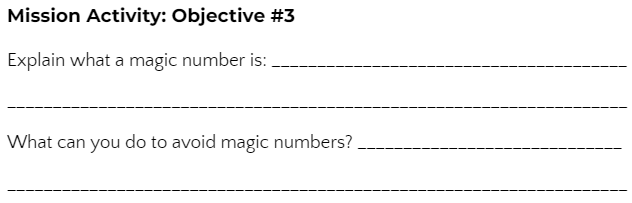
Python has a function that can tell you the display width and height.

* Use the function as the range in the **for** loop.
* Use the functions to define **x\_center** and **y\_center**.
* Then use the variables in the **display.set\_pixel()** functions

**Objective #3: Two Axes to Grind**

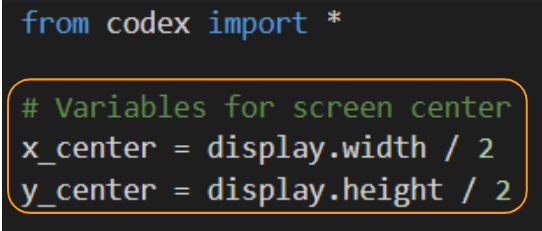
**DO THIS:**

* Go to your Mission Log and write the answers to the questions.





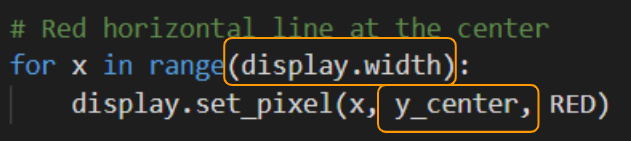
**DO THIS:**

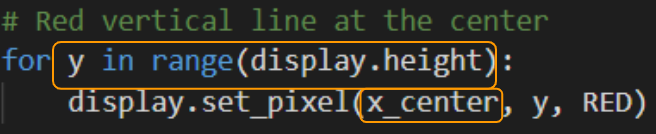
* Assign variables for the center of the screen.
  + Use **display.width** and **display.height**
  + Just divide the width and height in half!

**Objective #3: Two Axes to Grind**

**DO THIS:**

* Modify the **for** loop to use **display.width** and **y\_center**
* Add a new **for** loop just like the first one, but for a vertical line.
  + Use y and display.height for the range.
  + Use x\_center in the display function.
* Run the code. You WILL get an ERROR!





**Objective #4: Bug Fix**

Variables used for a location, like **x** and **y** must be integers, or data type **int**

* You don’t always know something until you try it!
* Then, you can just fix the bug.
* Be fearless – try stuff!

You can eliminate the error by **converting** the division problem to an integer, or data type **int**.

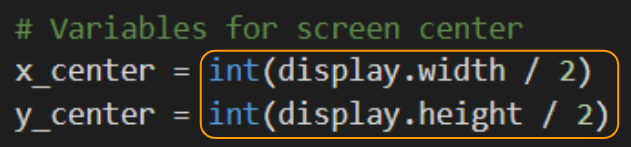


When you use division (/), the answer is automatically a float, or decimal number.

* **display.width / 2** = 120.0
* This is a float!
* We need 120, not 120.0
* Use the conversion function **int()** to convert it to an integer!

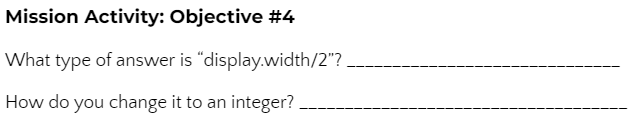
**Objective #4: Bug Fix**

**DO THIS:**

* Fix the bug by converting the **x\_center** and **y\_center** variables to integers
* Run the Code.

**DO THIS:**

* Go to your Mission Log and write the answers to the questions.



**Objective #5: Graphical Grid**

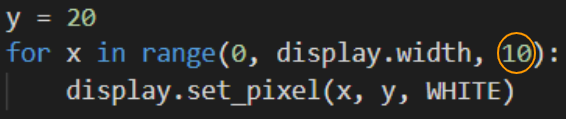
The X and Y axes (horizontal and vertical lines) will help with symmetry and balance as you create artistic designs.

* But it is still difficult to judge   
  scale at a glance.
* Creating a grid of dots can help   
  to clearly see pixel spacing over the whole screen.

**Dot dot dot …**

You want to create a grid of dots. Each “dot” is just a single pixel.

You can draw a line of white dots, only showing every 10th pixel, by using a step in the **for** loop.

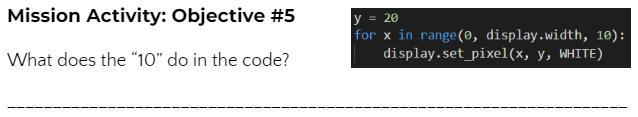


Go ahead – try it! Change the step and run the code again, to see what happens.

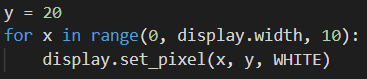
**Objective #5: Graphical Grid**

**DO THIS:**

* Go to your Mission Log and write the answer to the question.



**Objective #5: Graphical Grid**



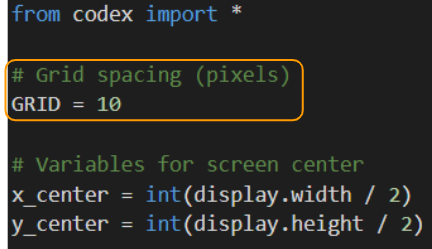
Notice how the **step parameter** of **range(start, stop, step)** advances **x** by **10** every loop.

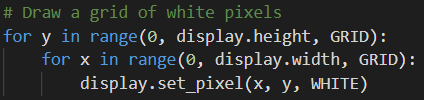
**Enter the Matrix:** You will need more than a single line of dots to complete the objective.

* A grid that covers the whole screen!
* This is a bunch of lines, drawn top to bottom.

**Objective #5: Graphical Grid**

**DO THIS:**

* No magic numbers!
* Define a CONSTANT for the step parameter (grid spacing)
* Use two loops – an inner and outer loop – to draw the grid
  + The **inner loop** draws a single horizontal line
  + The **outer loop** repeats the horizontal line down the height of the screen
  + *Note: If you have code in your program from page 14 (experimenting with step) you can delete it first.*
* Run the code.





**Mission Quiz: Graphics Ranger**

Test your skills by **taking the quiz**.

**Objective #6: Keep It Simple**

You have transformed your screen into a fantastic canvas for graphical artistry!

* But before you move on, you should neaten it up a bit.

**Line Drawing Function**

* Drawing a line pixel by pixel is awesome, but CodeX has a **built-in function** that draws a line faster and simpler
* And it can do diagonal lines, too!

Simplify your code by replacing the for loops with display.draw\_line().

The built-in function looks like this:



It requires the starting location (x1, y1) and the ending location (x2, y2) and the color.

**Objective #6: Keep It Simple**

Think about the horizontal and vertical lines in your code   
right now, drawn with for loops and pixels.

* What would the starting and ending locations be for each line?
  + **Horizontal line:** x goes from 0 to display.width,   
    y is at y\_center
  + **Vertical line:** y goes from 0 to display.height,   
    x is at x\_center
* What would the commands look like to draw the two lines?

**Frame it Up!**

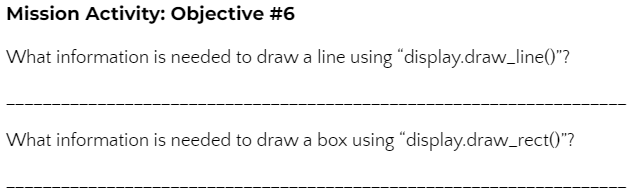
You can also draw a blue box around the edges to create a border for your art. You used **display.draw\_rect()** several times in Mission 13.

* What would the starting location be for the box?
  + Do you think x and y start at (0, 0)
* What are the width and height of the box?
  + Would you use display.width and display.height?

**Objective #6: Keep It Simple**

**DO THIS:**

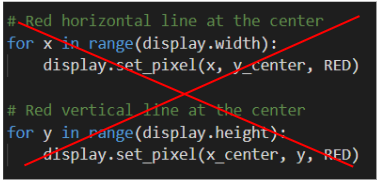
* Go to your Mission Log and write the answers to the questions.

****



**DO THIS:**

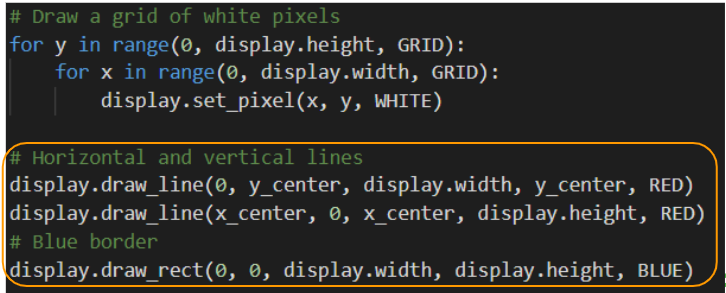
* Save to a New File!
* Use the **File → Save As** menu to create a new file called **LineArt**
* In your new **LineArt** file, **delete the code** that drew the red horizontal and vertical lines.

****

**Objective #6: Keep It Simple**

**DO THIS:**

* Draw the horizontal and vertical lines using **display.draw\_line()**
* Draw a border box using **display.draw\_rect()**
* Run the code.

****

**Objective #7: Get Artistic!**

You are working with straight lines. How artistic can you get?

* Well, you might be surprised!
* Straight lines can get downright curvy!

Look at the graphic in CodeSpace Objective 7.

* Notice as **Y** moves ***down***, **X** moves to the ***right***!
* Remember – on CodeX, **y** values **increase** when **going down**!

**Whoa! String Art :-)**

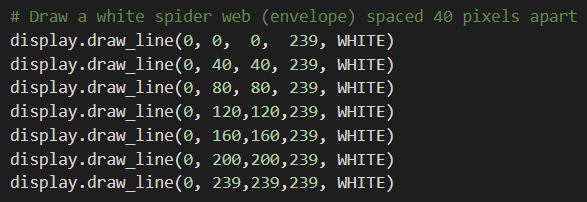
Watch the animation in CodeSpace Objective 7. The lines make a curve. That is called an envelope.

Click this link to learn more about [envelopes](https://en.wikipedia.org/wiki/Envelope_(mathematics)) in geometry. You don’t need to understand the math! Scroll through the page and look at some of the string art examples.

**Objective #7: Get Artistic!**

**DO THIS:**

* Draw several lines to create some line art (after the border.
  + Notice the **start x** is always **0**, and the **ending y** is always **239**.
  + Increasing the **start y** and **ending x** is how you slide the line down the screen.
  + The spacing inside the parameters is for clarity and doesn’t impact the run.
  + With all this repetition, you are probably thinking … shouldn’t we use a loop?
  + The answer is YES. We will do that next. For this objective, type in all the lines.
* Run the code.



**Objective #8: Automate Your Art**

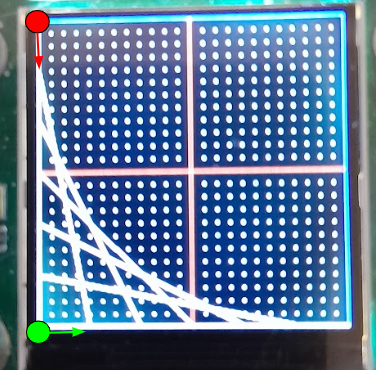
Hey, that is a pretty cool display!

* But it would be nice to automate some of those “magic numbers” and reduce the lines of code

The web follows this pattern:

* First line starts at (0, 0) and ends at (0, 239)
* The next line moves **y** down (+40) –and–   
  moves **x** across (+40)
* Then repeat the **y+40** and **x+40** many times

It may help to track the lines with your fingers.

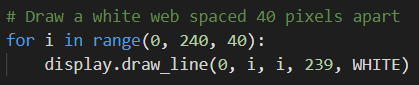
* Use your left finger for the start of the line.
* Use your right finger for the end of the line.
* Start with the first line (it is vertical against the border)
* Follow each line by moving down (left) and over (right)

**Objective #8: Automate Your Art**

Now try doing the webbing with a **for** loop.

Did you notice that the middle two numbers (**start y** and   
**end x**) were always the same? You can use a variable for that.

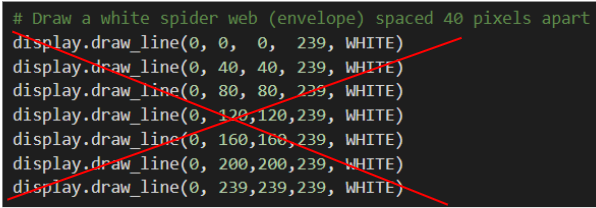
The numbers increase by **40**, so use that as your **step** parameter. Your loop could look like this:

****

Can you identify the start y, end x and step?



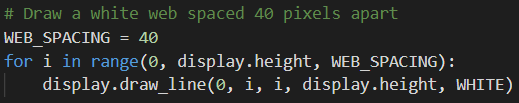
**DO THIS:**

* Delete ALL the lines of code that draw the white webbing.

**Objective #8: Automate Your Art**

**DO THIS:**

* To avoid a magic number, define a constant for **WEB\_SPACING** and use **display.height** or **display.width** when needed.
* Replace the code with a **for** loop.
* Run the code.

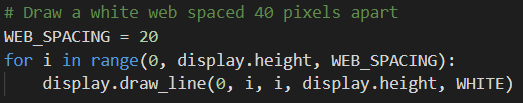




**DO THIS:**

* Change the value of WEB\_SPACING to something less than 40.

Run the code again and see what happens.

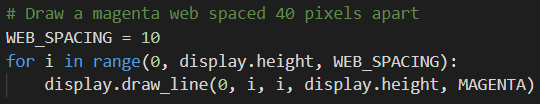


**Objective #9: A Splash of Color!**

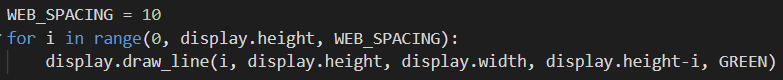
Take some time to experiment with the code you have now.

* Art is all about experimentation and creativity!

**Try changing the color:**

****

Add to your code by going from lower left to upper right:



Is your brain hurting yet?

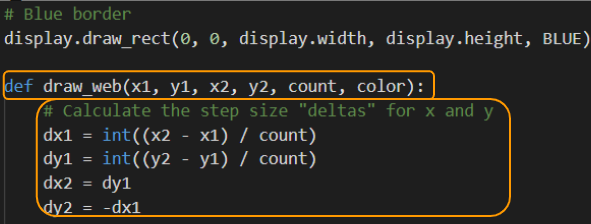
* It can be a little mind-bending trying to work out how to sweep lines across the screen just the way you want to.

We can simplify the approach!

* Create a function that draws webbing.
* It will use variables for “delta”, which means “change”

**Objective #9: A Splash of Color!**

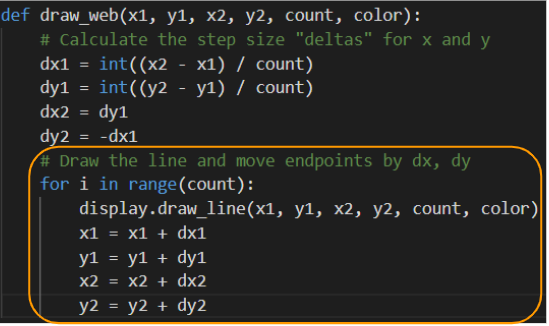
**DO THIS:**

* Create a function that will draw a web.
  + It is very similar to **display.draw\_line()** but uses one more parameter, **count**.
  + Count is the number of lines it will draw.
  + The function can go under the grid and border.
* Calculate the changes in the x and y locations.
  + This is a lot of math! Don’t worry about it if it doesn’t make sense.
  + Just know that a delta is a small change, and this math does all the work.

**Objective #9: A Splash of Color!**

**DO THIS:**

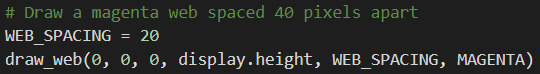
* Now use your values to draw a line, and make the changes for the next line.



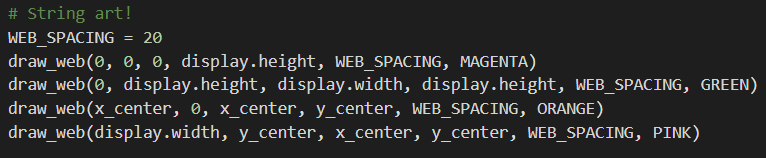
**Objective #9: A Splash of Color!**

**DO THIS:**

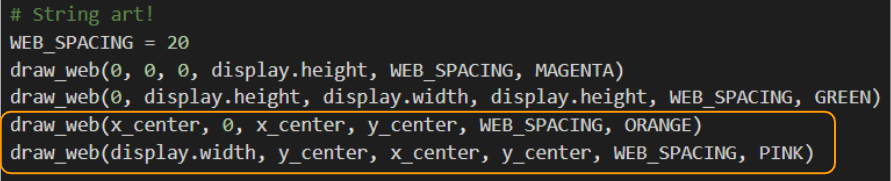
* First delete the for loop that draws the MAGENTA web.
* Replace the code by calling the function.
* Run the program.



* Now delete the code for the GREEN webbing.
* Replace the code by calling the function.
* Run the program



* Draw two more webs.
* Run the program.



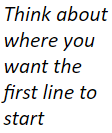
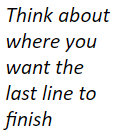
**Objective #9: A Splash of Color!**

**DO THIS:**

* Draw at least two more webs!
* You can use your own creativity, or:
  + Try finishing the inner webs by filling in the ??



* + Try finishing the outer webs by filling in the ??



**Mission Complete**

You have completed the thirteenth mission. 

**Do this:**

* Read your “Completed Mission” message
* Complete your Mission 14 Log
  + Post-Mission Reflection
* Get ready for your next mission!

**Wait! Before you go … Clear the CodeX**

Go to FILE -- BROWSE FILES

Select the “**Clear**” file and open it

Run the program to clear the CodeX

**Okay. Now you can go.**