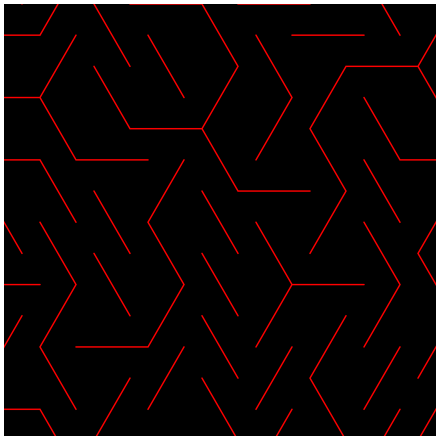
P5.JS ACTIVITY GUIDE

Hexagonal Grids

10 print but in hexagons!



An introduction to making patterns in hexagonal grids in p5.js. Square grids have 2 directions, hexagons have 3!

A zine by Tanvi Kumar

0 Give the pattern a home

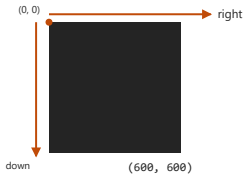
Open editor.p5js.org. Replace the starter code with this sketch, then press **Play**.

```
let x = 600; // canvas width and height

function setup() {
  createCanvas(x, x);
  noLoop();
}

function draw() {
  background("black");
}
```

setup() runs once. **draw()** paints the canvas; **noLoop()** stops it repeating. A black square is the right first result.



x = 600 is the canvas size.

The top-left corner is (0, 0).

Both sides are 600 pixels long.

Positive y goes down.

x sets both the width and height of our square canvas. **createCanvas(x, x)** makes it 600 pixels wide and 600 pixels tall.

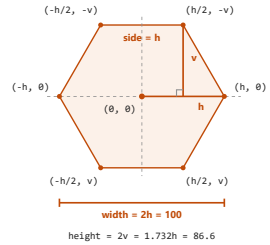
> **Tip**
> Try changing the value of **x** if you want a smaller or larger canvas!

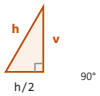
1 The Six Corners

Assume **h** is the side of a hexagon, with a size of 50. Let's find its corners! Add these lines below **x**, above **setup()**:

```
let h = 50; // hexagon side length
let v = 1.732 * h / 2; // half the hexagon height
```

This **flat-top hexagon** is centered at **(0, 0)**. Each corner is **h** away from the center; **v** is half its height.





Where does 1.732 come from?
Split the hexagon into triangles.

$v^2 + (h/2)^2 = h^2$

$v = \sqrt{3} \times h / 2$

$\sqrt{3} \approx 1.732$

Negative **y** is above the center, so the top corners use **-v**. Positive **y** is below it.

Connect each corner to the opposite corner. You get three lines through the center: two diagonals and one horizontal. We can randomly choose a line for every hexagon to make our piece!

2 Choose one of three lines

Add this function **below draw()**, not inside it.

```
function drawPattern(i, j) {
  push();
  translate(i, j);
  stroke("red"); // change the line color
  strokeWeight(2); // change the line thickness
  let r = random();
  if (r > 0.67) {
    line(h/2, -v, -h/2, v);
  } else if (r > 0.33) {
    line(-h/2, -v, h/2, v);
  } else {
    line(-h, 0, h, 0);
  }
  pop();
}
```







push() saves the drawing state. **translate(i, j)** moves the local origin to the cell's center, so every endpoint is an offset from **(i, j)**. **pop()** restores the state.

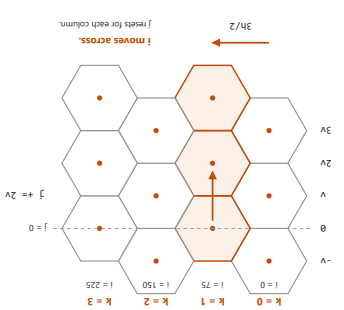
random() gives a number between 0 and 1. Above 0.67, we draw one diagonal. Between 0.33 and 0.67, we draw the other. At 0.33 or below, we draw a horizontal line: about a **1/3 chance per line!**

> **Tip**
> Temporarily add **drawPattern(x/2, x/2);** inside **draw()** to try one tile. Press Play a few times, then remove that call before moving on. The hexagon outlines are just guides; the sketch draws only the lines.

3 Walk the hexagon centers

A square grid stacks rows in line. In a hexagonal grid, we shift **alternate columns by half a cell's height**. We do this to tile the hexagons.

Let's see the math needed to navigate (or walk) this grid.



i moves right
The center's horizontal coordinate.

j moves down
The center's vertical coordinate.

k counts
The column number: 0, 1, 2, 3...

Across: a neighboring center is **h + h/2 = 3h/2** away. With **h = 50**, that is **75** pixels.

Down: centers in one column are a full height, **2v = 86.6** pixels, apart. Adjacent columns differ by **v = 43.3** pixels vertically.

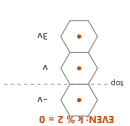
> **Tip**
> Try tracing the centers with your finger: follow **j** down one column, then move **i** right by 75 and start again at the top of the next column.

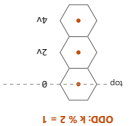
4 Turn the map into loops

% gives the remainder. For odd **k**, **k % 2** is **1** (true): start **j** at **0**. For even **k**, it is **0** (false): start at **-v**.

The inner loop finishes a column. Then **i += 3*h/2** moves right, **k++** counts the next column, and a new **j** starts at that column's top.

Why go outside? Starting at **-v** and continuing to **x + h / x + 2*v** draws past the edges. The canvas clips the excess, avoiding empty strips.






```
  }
}
for (let i = 0, k = 0;
     i < x + h; i += 3*h/2, k++) {
  if (k % 2) {
    for (let j = 0;
         j < x + 2*v; j += 2*v) {
      drawPattern(i, j);
    }
  } else {
    for (let j = -v;
         j < x + 2*v; j += 2*v) {
      drawPattern(i, j);
    }
  }
}
```

Inside **draw()**, we walk the grid and draw our pattern.

> **Tip**
> Try making both starting values of **j** zero and notice how the columns line up. Then undo the change to restore the honeycomb.

5 The whole sketch

```
let x = 600; // canvas width and height
let h = 50; // hexagon side length
let v = 1.732 * h / 2; // half the hexagon height

function setup() {
  createCanvas(x, x);
  noLoop();
}

function draw() {
  background("black");
  stroke("red"); // change the line color
  strokeWeight(2); // change the line thickness
  let r = random();

  if (r > 0.67) {
    line(h/2, -v, -h/2, v);
  } else if (r > 0.33) {
    line(-h/2, -v, h/2, v);
  } else {
    line(-h, 0, h, 0);
  }
  pop();
}

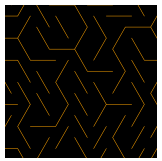
function drawPattern(i, j) {
  push();
  translate(i, j);
  stroke("red"); // change the line color
  strokeWeight(2); // change the line thickness
  let r = random();

  if (r > 0.67) {
    line(h/2, -v, -h/2, v);
  } else if (r > 0.33) {
    line(-h/2, -v, h/2, v);
  } else {
    line(-h, 0, h, 0);
  }
  pop();
}

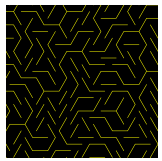
for (let i = 0, k = 0;
     i < x + h; i += 3*h/2, k++) {
  if (k % 2) {
    for (let j = 0;
         j < x + 2*v; j += 2*v) {
      drawPattern(i, j);
    }
  } else {
    for (let j = -v;
         j < x + 2*v; j += 2*v) {
      drawPattern(i, j);
    }
  }
}
```


6 Same rules, different patterns

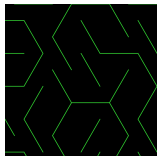
Try different values of **x** and **h**. A larger canvas fits more hexagons; a smaller side length packs them closer together.



x = 600, h = 50
Our starting pattern



x = 600, h = 30
Smaller hexagons, more lines



x = 400, h = 50
A smaller canvas, fewer cells

> **Tip**
> Remove **noLoop();** and add **frameRate(1);** inside **setup()**. Watch a different artwork be created every second!

We drew lines. You could draw more complex random things to create other fun patterns!

Enjoy exploring with p5!