Ethan
developer notes & tutorials

How to draw a hexagonal grid on HTML Canvas

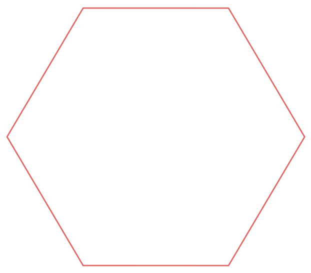
July 23, 2020

In this article, we are going to learn how to get a perfect hexagon grid using JavaScript to draw on an HTML canvas. We first need to know a bit of trigonometry to solve this problem as it is necessary for all the calculations for the coordinate points composing a regular polygon.

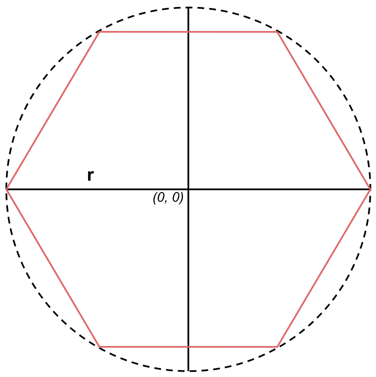
Contents

The Basics

First of all, we introduce a regular hexagon that is composed of six equal sides.

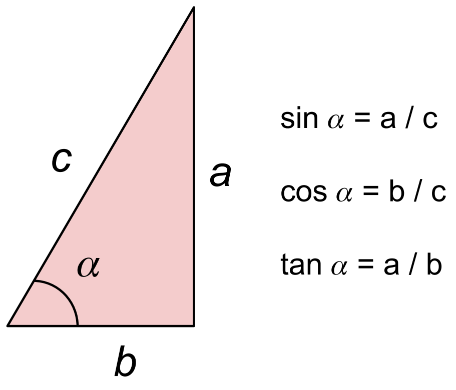


Any regular polygon can be inscribed within a circumference of radius **r**

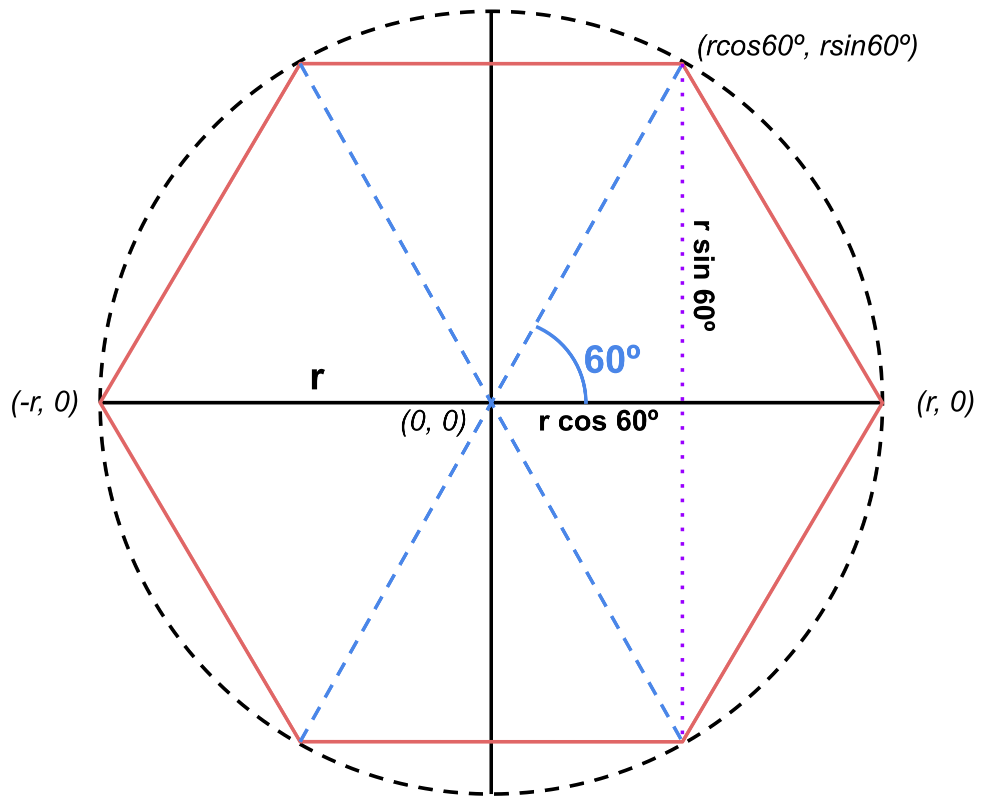


So each of its vertexes intersects with the circumference. Drawing from the premise that the centre of the circumference is the point of origin *(0,0)* we can easily calculate the most-right and most-left vertexes are *(**r**, 0)* and *(-**r**,0)* respectively, however, what are the positions of the rest of the points? Here is where trigonometry comes into play.

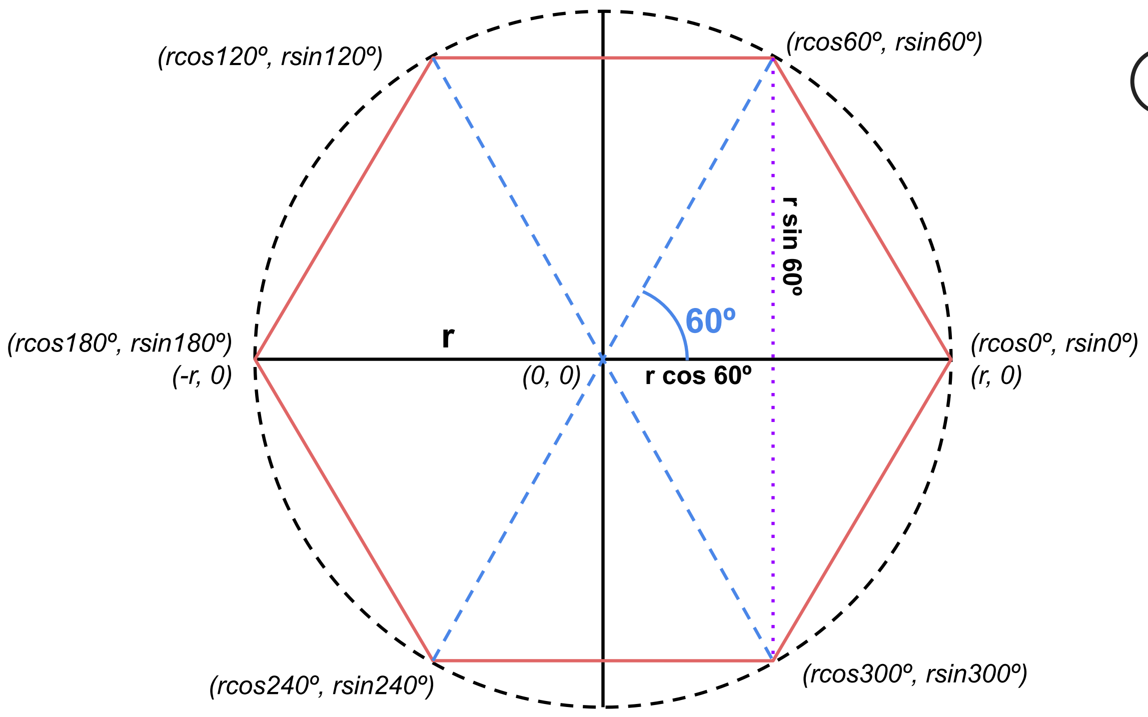
Given any right triangle, the following trigonometric functions applies:



It is very useful to know any side of the triangle if you know one of its other sides and the angle it forms. For this case, the angle formed by each vertex with the horizontal axis is equal by dividing the circumference by the number of sides (360º / 6 = **60º**) and we also know that the hypotenuse is equal to the radius of the circumference **r**. From the first equation we can say that a = c *sinα and b = c* cosα. In summary, putting all together the second vertex coordinates are ***(rcos60º,rsin60º)***.



Then the rest comes as a multiple of 60º as 120º, 180º, 240º, 300º and 360º which is equal to 0º again. Notice that the most-right and most-left vertex coincide with what we have expected due to sin0º = 0, cos0º = 1, cos180º = -1 and sin0º = 0. These are the resulting vertexes:



A Hexagon

As this point we can start a new project to put in practice all we have seen. In an **index.html** file we set the minimum required fields for a HTML canvas:

```html
<!DOCTYPE HTML>
<html lang="en">
  <head>
   <meta charset="UTF-8">
   <title>HexGrid</title>
  </head>
  <body>
    <canvas id="canvas" width="800" height="500"/>
    <script src="main.js"></script>
  </body>
</html>
```

And a **main.js** file:

```javascript
const canvas = document.getElementById('canvas');
const ctx = canvas.getContext('2d');

function init() {}
init();
```

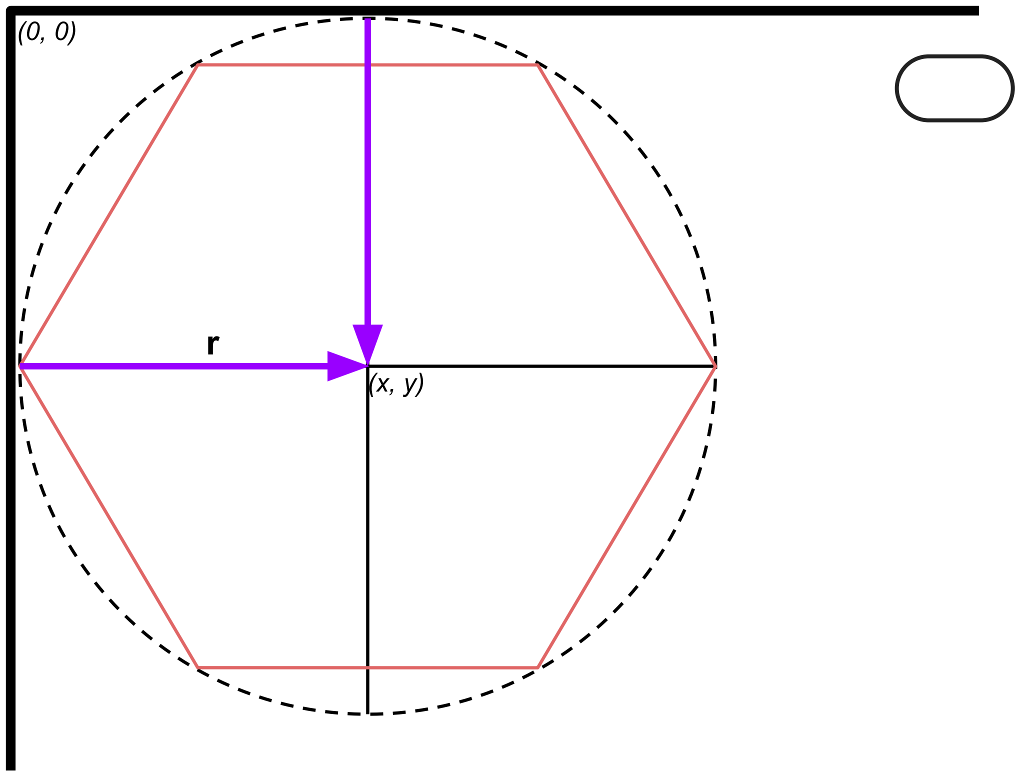
As far as we know, we are going to set up the angle and the size of the hexagon as constants. Notice the angles are needed to be expressed in radians (360º = 2π rad)

```javascript
const a = 2 * Math.PI / 6;
const r = 50;
```

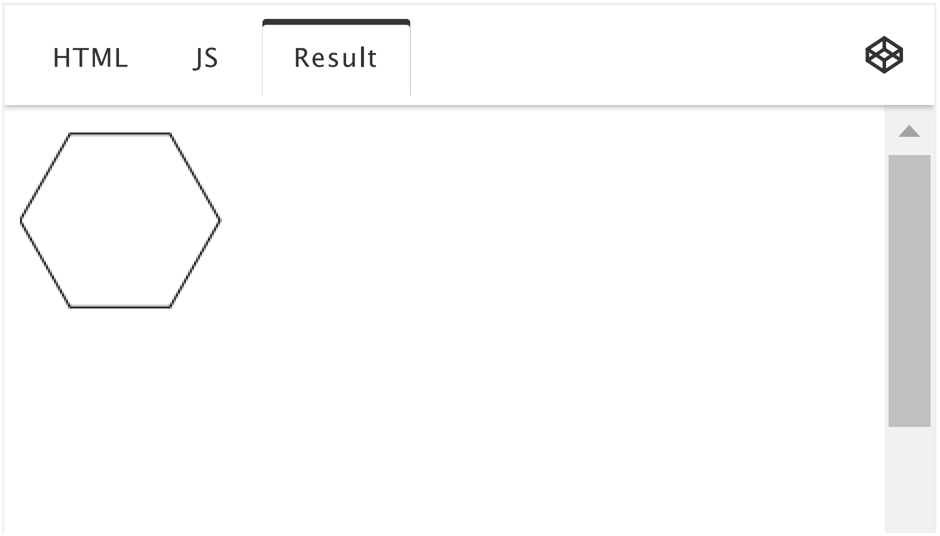
In order to draw a regular hexagon we define a function named *drawHexagon(x,y)* being *x* and *y* the center point. We are going to use a path that allows to set the coordinates before drawing them and when finished we use *stroke()* to draw only the border line. It is possible doing a *for loop* to draw a line between each vertex so the result is as follows:

```javascript
function drawHexagon(x, y) {
  ctx.beginPath();
  for (var i = 0; i < 6; i++) {
    ctx.lineTo(x + r * Math.cos(a * i), y + r * Math.sin(a * i));
  }
  ctx.closePath();
  ctx.stroke();
}
```

Before testing it, notice that the point *(0,0)* in our canvas starts on the upper left corner, so to fit the drawing we need a minimum offset of **r**.

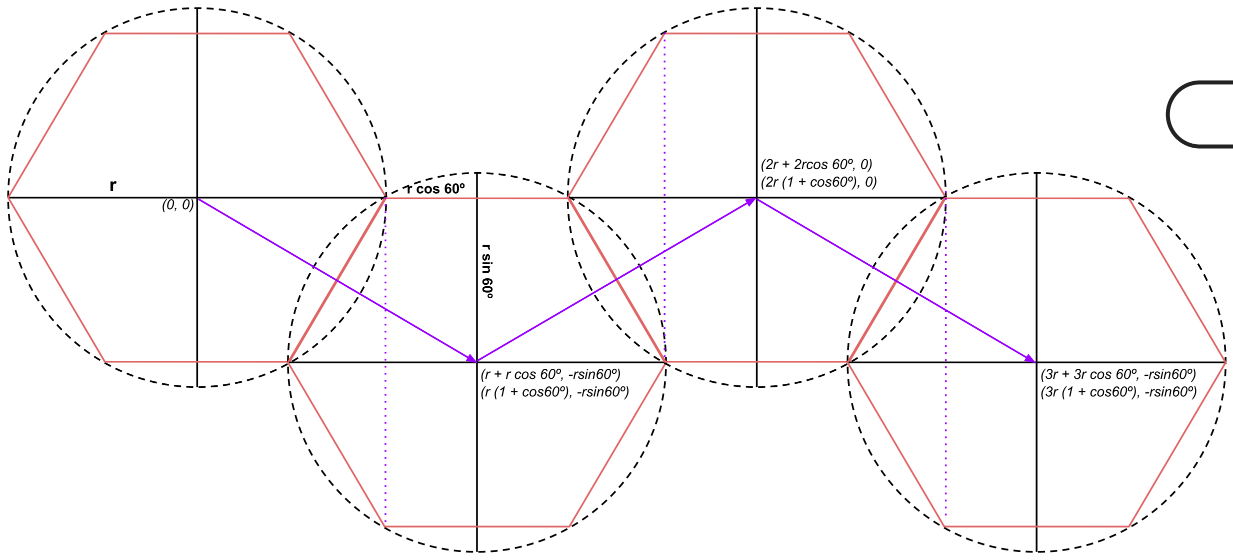


Result



A Row

Perfect! The next step is to draw a row of hexagons, like that:


r
(0, 0)
r cos 60º
r sin 60º
(2r + 2rcos 60º, 0)
(2r (1 + cos60º), 0)
(r + r cos 60º, -rsin60º)
(r (1 + cos60º), -rsin60º)
(3r + 3r cos 60º, -rsin60º)
(3r (1 + cos60º), -rsin60º)

Essentially it is important to know where the next centre is going to be located to fit perfectly with one another. First, notice how much horizontally is placed the purple arrow. It is a distance of the radius **r** plus a segment we already know as **rcos60º**. And same as vertically, a segment of **rsin60º** downwards. The procedure is always adding the same amount horizontally and alternating vertically.

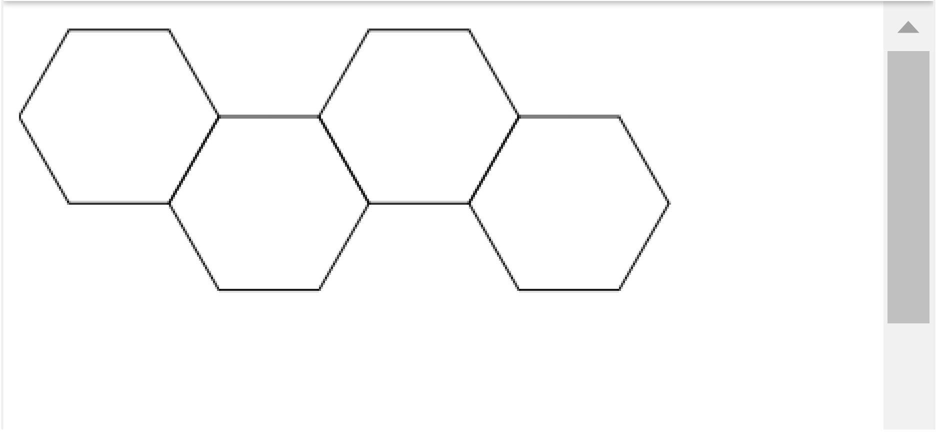
The code that allows to draw the four hexagons showed before is:

```
// 1st
x = r;
y = r;
drawHexagon(x, y);

// 2nd
x = x + r + r * Math.cos(a);
y = y + r * Math.sin(a);
drawHexagon(x, y);

// 3rd
x = x + r + r * Math.cos(a);
y = y - r * Math.sin(a);
drawHexagon(x, y);

```

```
// 4th
x = x + r + r * Math.cos(a);
y = y + r * Math.sin(a);
drawHexagon(x, y);
```

Result

HTML JS Result



We need to find the pattern that will allow making this scalable. On the one hand, *x* could be written as a increment of:

```
x = x + r + r * Math.cos(a);
```

That shortened is expressed as:

```
x += r * (1 + Math.cos(a));
```

On the other hand, *y* is altered between adding or subtracting whether it is an even or odd position:

```
y = y + r * Math.sin(a); // Even position
y = y - r * Math.sin(a); // Odd position
```

How it could be written for a general case? Let’s assign a new variable *j* that increases just as it does the position we are in. If we use this mathematical trick, we can do like an if-statement for alternating whether is an even or an odd number:

```
(-1) ** j = -1 when j is odd
(-1) ** j = 1 when j is even
```

That is exactly what we were looking for! Let’s wrap it all together, and *y* is expressed for every iteration as:

```
j++;
y = y + (-1) ** j * r * Math.sin(a);
```

That shortened is expressed as:

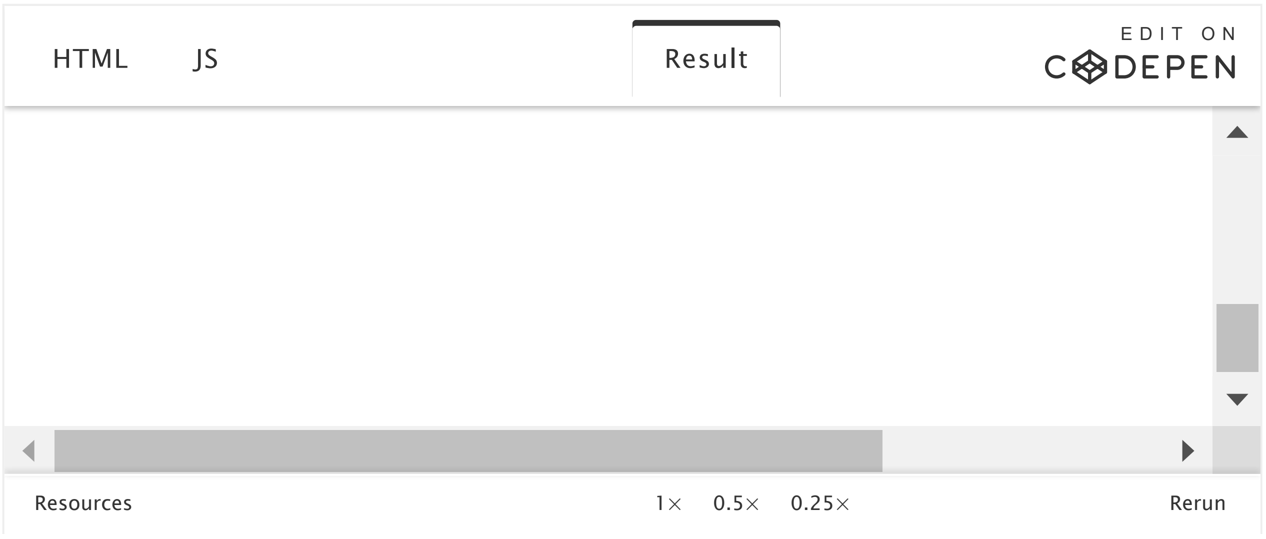
```
y += (-1) ** j++ * r * Math.sin(a);
```

Finally we arrive to the solution on how to draw many hexagons in a row as we initially intended. We define a function named *drawGrid(width,height)* that prints what we have just explained up to this point:

```
function drawGrid(width, height) {
  let y = r;
  for (let x = r, j = 0; x + r * (1 + Math.cos(a)) < width; x += r * (1 + Math.cos(a)), y += (-1) ** j++ * r * Math.sin(a)) {
    drawHexagon(x, y);
  }
}
```

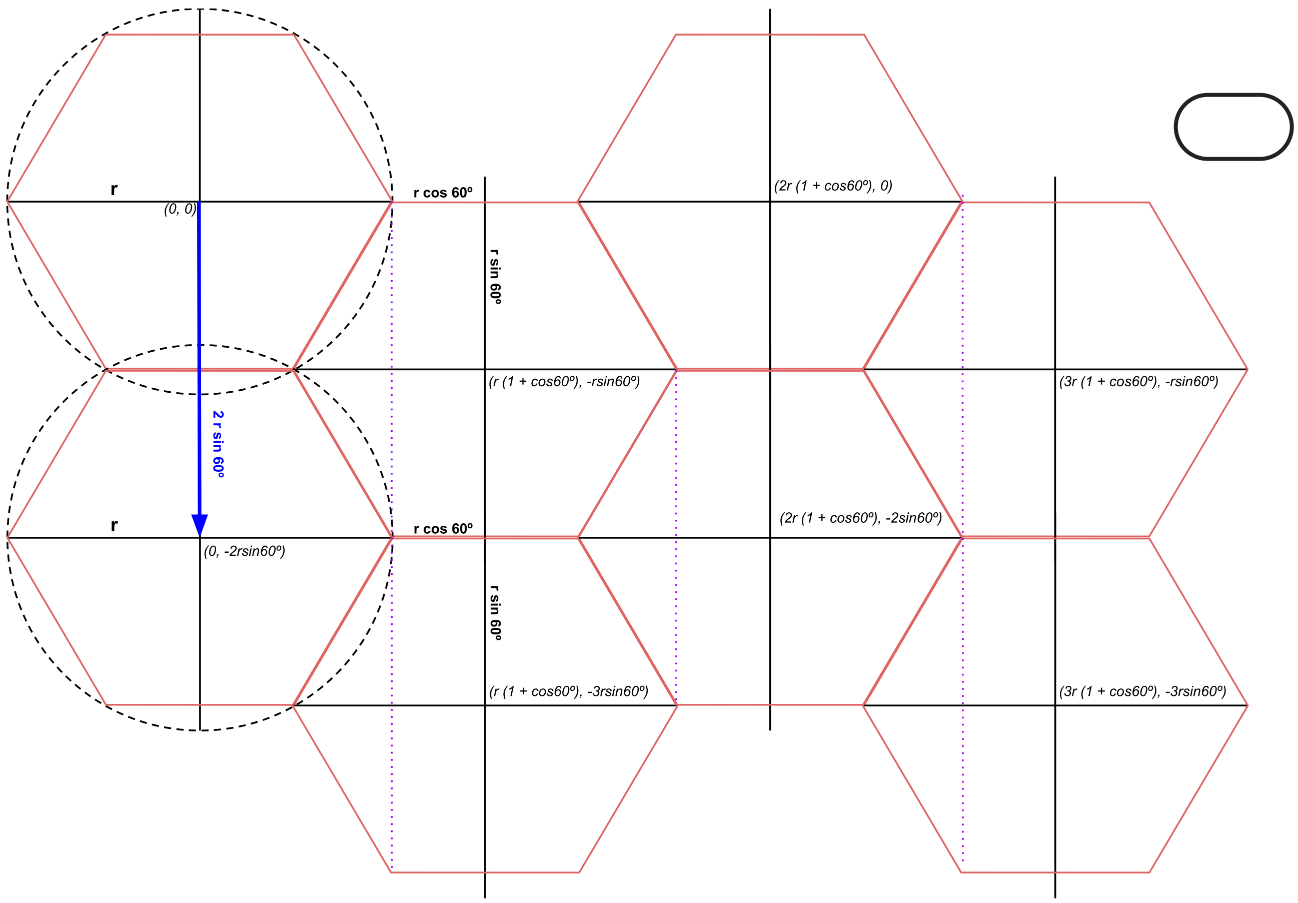
Notice whether the subsequent hexagon, in every iteration, that we are going to draw fits inside the canvas.

Result



The Grid

That is it! We are just one step away from success. All we need is to repeat the same procedure but in the row below repeatedly. But, how much lower is it from the original row? Let’s find it out:

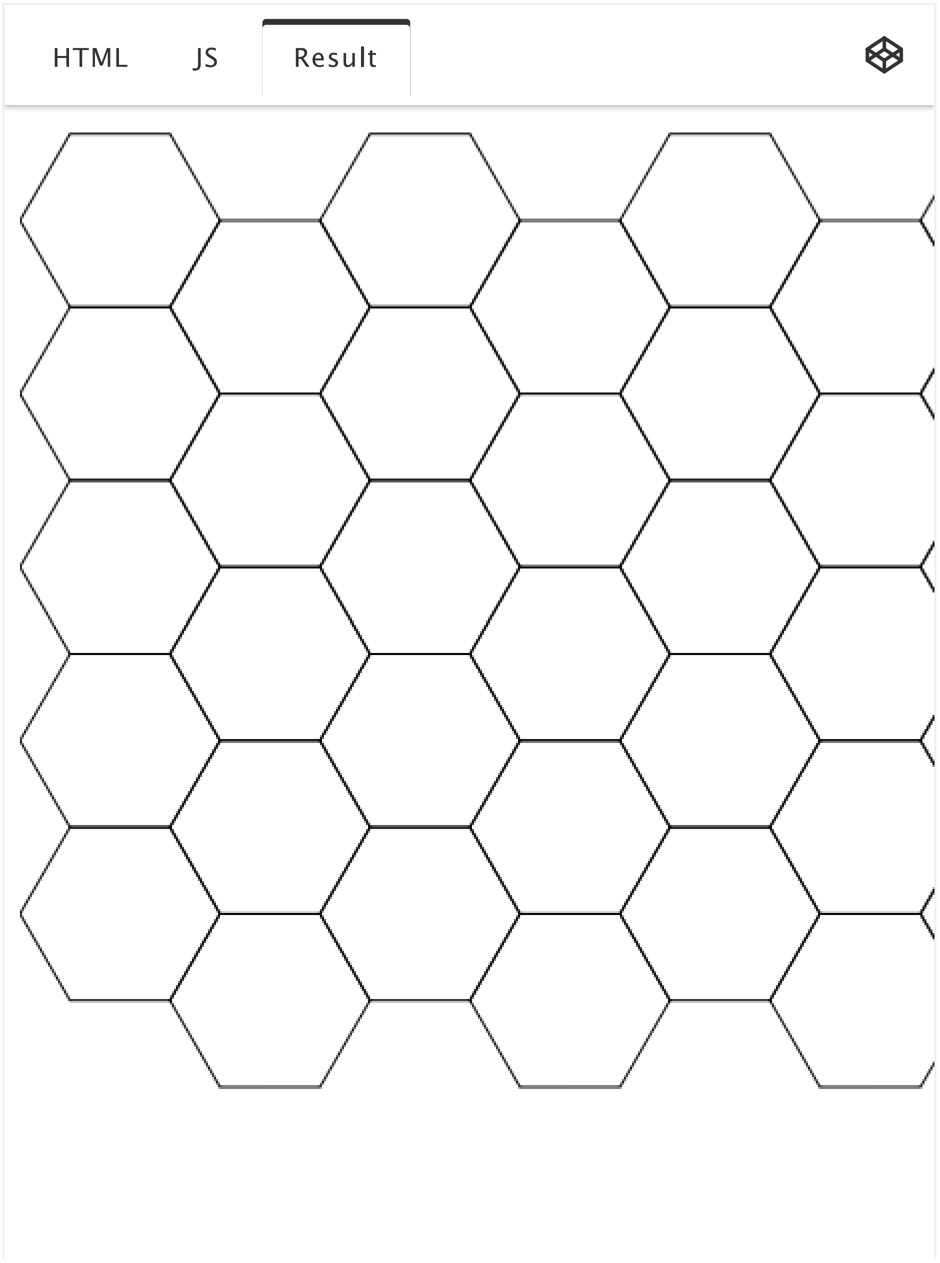

r
(0, 0)
r cos 60º
r sin 60º
(2r (1 + cos60º), 0)
(r (1 + cos60º), -rsin60º)
(3r (1 + cos60º), -rsin60º)
2 r sin 60º
r
(0, -2rsin60º)
r cos 60º
(2r (1 + cos60º), -2sin60º)
r sin 60º
(r (1 + cos60º), -3rsin60º)
(3r (1 + cos60º), -3rsin60º)

This would be the final scheme of our grid, showing the first four centres of each row to get a good view on what is going on. From the centre *(0,0)* we can see that the blue arrow takes a distance of twice the length of the hexagon height that sums up to **2rsin60º**. The rest is going to be the same taking into account this offset. We modify our function to draw many lines as the last hexagon fits in the canvas height.

```javascript
function drawGrid(width, height) {
  for (let y = r; y + r * Math.sin(a) < height; y += r * Math.sin(a)) {
    for (let x = r, j = 0; x + r * (1 + Math.cos(a)) < width; x += r * (1 + Math.cos(a)), y += (-1) ** j++ * r * Math.sin(a)) {
      drawHexagon(x, y);
    }
  }
}
```

Let's put all together and try it out!

Result



We have finally gotten a hex grid with just a few lines of code in JavaScript.

Thanks to @SergiVera for helping me write this article.

Written by **Izan Pérez Cosano**
GitHub Profile

Telecommunications and Telematics Engineer at the Polytechnic University of Catalonia. LinkedIn Profile

Creating the UNO game on JavaScript. (Part I) →