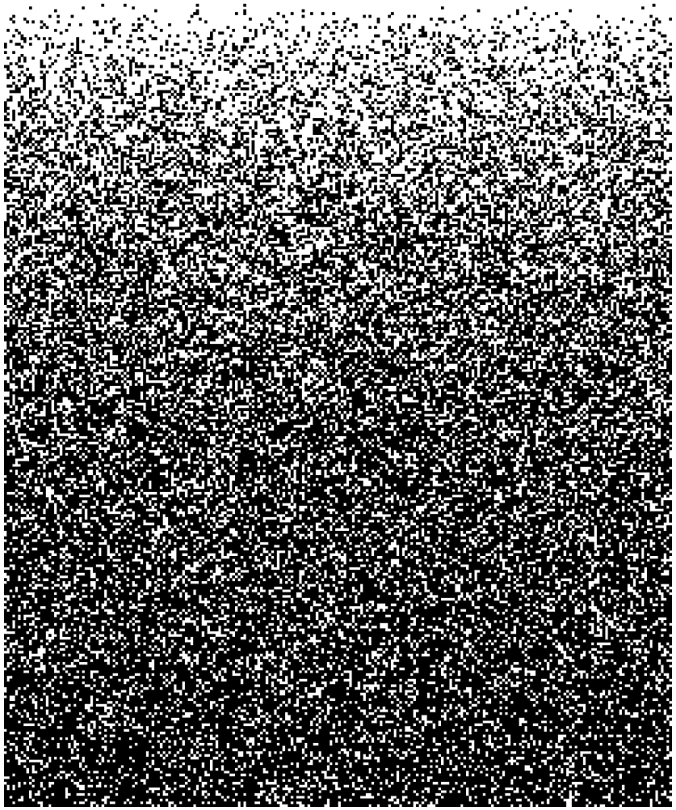
Splash of Code

Horizon Dust

Learn JavaScript by Making Computer Art



Joel Dare

Going On

If you've gotten this far, and have a complete working image, here are a few ideas about how you could make it your own. Save your work before you start to experiment.

Try setting the *likely* variable to a different value. Set it to a whole number such as 125. Experiment with other values and see how they change the image.

Try changing the size of the *topEdge* variable and see how that affects the finished image.

Try changing the color of your image. Give "red" and "orange" a try as color options.

For an even bigger challenge, set the stroke style just before you draw the rectangle. Pick a random number between 0 and 1 and assign it to a variable called *color*. If the value of color is 0, set *ctx.strokeStyle* and *ctx.fillStyle* to "red". Otherwise, set them to "orange".

likeliness will remain the same as we move across the page drawing dots but will reduce slightly as we move up the page to draw another row of dots.

```
    // Reduce likeliness
    likely--;
```

Finally, we have our random number function. This makes the code for picking a random number a little smaller and easier to read wherever *rand()* was used above.

```
// Pick a random number
function rand(min, max) {
      max = max + 1;
      let rand = Math.random();
      return Math.floor((rand * max) + min);
}
```

That's it. Now we should be able to refresh the page and see how it looks. Because we're using randomness the image will look different each time it's drawn. When you're happy with the result you can save it by right clicking on the image and selecting *Save Image As*.

Introduction

Splash of Code is a series of short zines. In each book you'll make an art piece by typing code from these pages into your computer. No previous experience, special tools, or skills are necessary. You'll be working in JavaScript, a language that works in your web browser. Along the way you'll start to absorb some computer programming techniques. In the end you'll create an art piece that's fun to display and discuss.

Our goal is to create a simple, abstract, image inspired by dust over the horizon at sunset. I'll print mine from an old black and white laser printer on standard paper. If you want to print yours, any printer will do.

Remember that you don't need to understand everything you're entering. Just relax and copy the code. You should end up with a great piece of artwork you can be proud of.

Getting Started

You'll need a computer with a web browser. Nothing else is required. I've created a basic HTML file for you on *CodePen*, a tool that lets you write code online. Open the URL below to get started.

```
https://codepen.io/codazoda/pen/aPVGGx
```

This book includes both a *Complete Code* listing and a *Code Walk Through*. The walk through explains each of the small sections in more detail. You can start at either section. If you want to get right to work, start with the *Complete Code*. If you'd rather read a little more about it, start at the *Code Walk Through* and read it all the way through. Once you're ready to start, turn to the *Complete Code* section and retype the code into CodePen.

The project consists of two files. An HTML file that you won't need to change and a JavaScript file. You'll type your work into the JavaScript file.

As you work on this project you'll probably run into typos that create errors. If the image doesn't draw at all, or in a way that you expect, double check that you typed everything correctly and then look at the output in the developer console to see if there are any errors. Because each browser is different, you'll want to search the internet for instructions on how to open the developer console in your particular web browser or refer to the instructions in the *How to Code* zine.

Once you've created an image that you're happy with you can save it by right clicking on the image and selecting *Save Image As*.

Because we're going to draw rectangles we'll want to set a stroke (or outline) style and a fill style. In both cases we'll set them to *black*.

```
// Set the stoke style
ctx.strokeStyle = "black";
ctx.fillStyle = "black";
```

We're going to start at the bottom of the canvas and work our way up. We're going to draw random dots (actually rectangles) across the page and as we move up the number of dots we draw will decrease, making each line of random dots a little lighter than the line drawn before it. Let's setup a few variables and then create a *for* loop.

```
// Loop from the bottom to the top
var y = 0;
var shouldDraw = 0;
// Figure out the top edge size (20%)
var topEdge = (height / dotSize) * 0.20;
// How likely are we to draw a pixel
var likely = height / dotSize;
for(y=height; y>0; y=y-dotSize) {
    // Drawing happens here
}
```

Next we jump inside the for loop and we draw a rectangle at the random position we select.

```
    // Pick a random number
    shouldDraw = rand(0, likely);
    if(shouldDraw >= topEdge) {
        ctx.fillRect(x, y, dotSize, dotSize);
    }
```

Moving back outside the for loop, we want to reduce the likeliness that we'll draw a dot. As we move up the page each line will be less and less likely to draw a random dot at the current location. The

Code Walk Through

In this section we'll discuss how the code works but we'll jump around a bit. Read this section all the way through and use the *Complete Code* listing as your reference when typing the code.

First we'll set up some page parameters. We'll set a resolution of 300 pixels per inch, set the width to 4 inches, and set the height to 6 inches. We'll also set a dot size of 10 pixels. This is how large we'll draw each dot or rectangle on the image.

```
// Setup some params
const resolution = 300;
var width = 4 * resolution;
var height = 6 * resolution;
var dotSize = 10;
```

Now we'll grab the canvas element from our HTML page and assign it to the *myCanvas* variable.

```
// Grab the canvas element from the page
var myCanvas = document.getElementById("myCanvas");
```

Next we set the canvas width and height to the values we calculated earlier.

```
// Set the canvas width and height
myCanvas.width = width;
myCanvas.height = height;
```

Now we grab the context of the canvas to draw on and we save it in a variable that we call *ctx*.

```
// Grab the 2D context of the canvas to draw on
var ctx = myCanvas.getContext("2d");
```

Pixels, Points, Dots, and Inches

When we're drawing on the canvas in JavaScript we typically work in pixels. You can't really draw an image in inches because we're drawing on the computer screen and different screens have pixels of different physical sizes. If you were to draw an image that was 200 pixels wide and measure it with a ruler you'd find that it's a different size on your computer than it is on my computer.

When we print something we typically print a bunch of dots in an inch on the page. This is called the print *resolution*. Let's say that your printer is capable of printing 300 dots per inch. If we want to print an image that's 4 inches wide, we could make the image 4 x 300 pixels, or 1200 pixels, wide.

When you print that image you can still print it any size you like. If you use your printers default settings it will probably stretch to the full size of the page, minus the margins. When that happens the printed image is no longer 300 dots per inch. Your printer did some interpolation and increased the size of the image.

It can all get a bit confusing. Keep in mind that software on your computer can stretch an image, you're printer can stretch it, and the browser can stretch it, just to name a few. Sometimes it's easiest just to create it at a "high enough" resolution like 300 or 600 dots per inch and let the image viewing software figure it out.

Complete Code

```
// Setup some params
const resolution = 300;
var width = 4 * resolution;
var height = 6 * resolution;
var dotSize = 10;

// Grab the canvas element from the page
var myCanvas = document.getElementById("myCanvas");

// Set the canvas width and height
myCanvas.width = width;
myCanvas.height = height;

// Grab the 2D context of the canvas to draw on
var ctx = myCanvas.getContext("2d");

// Set the stoke style
ctx.strokeStyle = "black";
ctx.fillStyle = "black";

// Loop from the bottom to the top
var y = 0;
var shouldDraw = 0;
// Figure out the top edge size (20%)
var topEdge = (height / dotSize) * 0.20;
// How likely are we to draw a pixel
var likely = height / dotSize;
for(y=height; y>0; y=y-dotSize) {
    // Loop from the left to the right
    for(x=0; x<=width; x=x+dotSize) {
        // Pick a random number
        shouldDraw = rand(0, likely);
        if(shouldDraw >= topEdge) {
            ctx.fillRect(x, y, dotSize, dotSize);
        }
    }
    // Reduce likeliness
    likely--;
}

// Pick a random number
function rand(min, max) {
    max = max + 1;
    let rand = Math.random();
    return Math.floor((rand * max) + min);
}
```