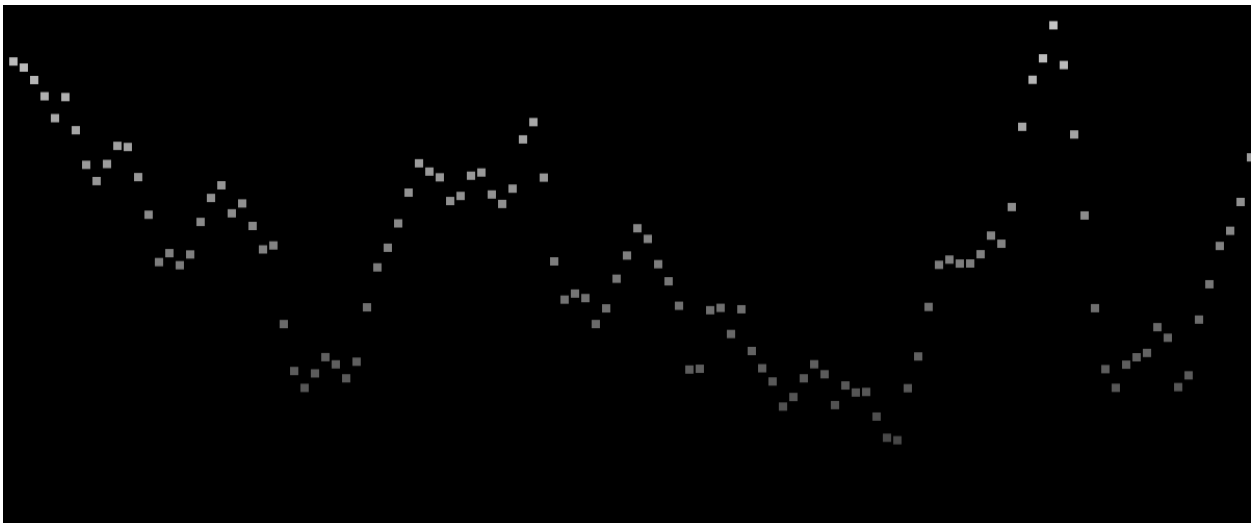


Week 5 - Technical Exercise in p5

This week we will introduce the p5 noise() function with a little bit of input from the mouse. We are returning to our row of rectangles we saw in week 3 and adjusting our code to produce the above!#

Step 1: Variables

Declare 2 variables at the top of your sketch.

```
let xpos = 5; // variable to store the x position of each rectangle
let xstep = 5; // variable to store the distance between the rectangle (on the x axis)
```

Step 2: Setup function

Define a setup function to create a canvas with a width of 600 and height of 400. As we will be drawing rectangles, set the rectangles drawing mode to draw the rectangles from the center.

```
/* The code in the setup function runs only once when the sketch starts */
function setup() {
  createCanvas(400, 400); // p5 function to create a canvas of the set size
  rectMode(CENTER); // sets the rectangle position values to be the center
}
```

Next define a draw function and set the background to black.

Step 3: Draw function

```
/* The code in the draw function is looped on every frame until closed */
function draw() {
  background(0); // draws the background a set colour


}
```

Step 4: For loop

Now we will re-use the for loop we used in week 3 to draw a row of rectangles half way up our screen. Let recap. A for loop is used to loop around some code a specific amount of times. The amount is defined on the top line of the loop. The code to be looped resides inside the body of the loop, inside the curly brackets.

Inside your draw function define a for loop using a new variable `i` :

The top line of a for loop uses the keyword `for`, then has three sections inside brackets that are **separated by commas:**

In the first section `i` is declared and initialised to `0` —> `let i = 0`

The second section determines when the loop stops, it uses the condition that `i` must be less than the width of the canvas divided by the distance between the rectangles (which calculates how many rectangles are drawn) —> `i < width/xstep`

The third section specifies that on every loop `i` must increase by 1 —> `i++`

```
for(let i = 0; i < width/xstep, i++) {

  ......

}
```

Step 5: Draw some rectangles

Inside the for loop we want to draw each rectangle in a row halfway up the screen. Firstly we will set the fill colour to `255` so it is white. We will use the variable `xstep` to set the size of the rectangle.

For the x coordinate of the rectangle we will use the same formula as we did originally:

`xpos + (xstep*i)`

This takes the starting x position, which is defined as `5` at the top of the sketch, and adds the distance between each rectangle `xstep` multiplied by `i` . As `i` increasing by 1 on every loop the x position is adjusted on every loop across the canvas.

The y coordinate starts as the height of the canvas divided by 2 to sit half way up the screen.

```
 for(let i = 0; i < width/xstep, i++) {

  fill(255);
  rect(xpos + (xstep*i), height/2, xstep, xstep);

}
```

Step 6: Noise

Here we will use the noise function to set the y position of each rectangle.

The noise function is used in a similar way to random as it outputs numbers that are unpredictable. However, unlike random, the noise function generates smoother gradients of output using a function called Perlin noise, named after its inventor, Ken Perlin. Perlin noise was invented for applications which required semi-random variations with continuous and smooth trajectories.

For the reference page on noise() see https://p5js.org/reference/#/p5/noise

To use noise we must specify at least the x coordinate in the noise space. Don't worry too much about what that means for now- all we need to know is that the function gives us a smooth sequence of values based on the value given. If we give it the value `i` , which goes from 0, 1, 2, 3 to the amount of rectangles, the noise function will give us a noisy sequence of values related to those incrementing numbers.

`noise(i)`

If you want a more detailed description of noisy values, you can watch Daniel Shiffmans video: https://www.youtube.com/watch?v=YcdldZ1E9gU

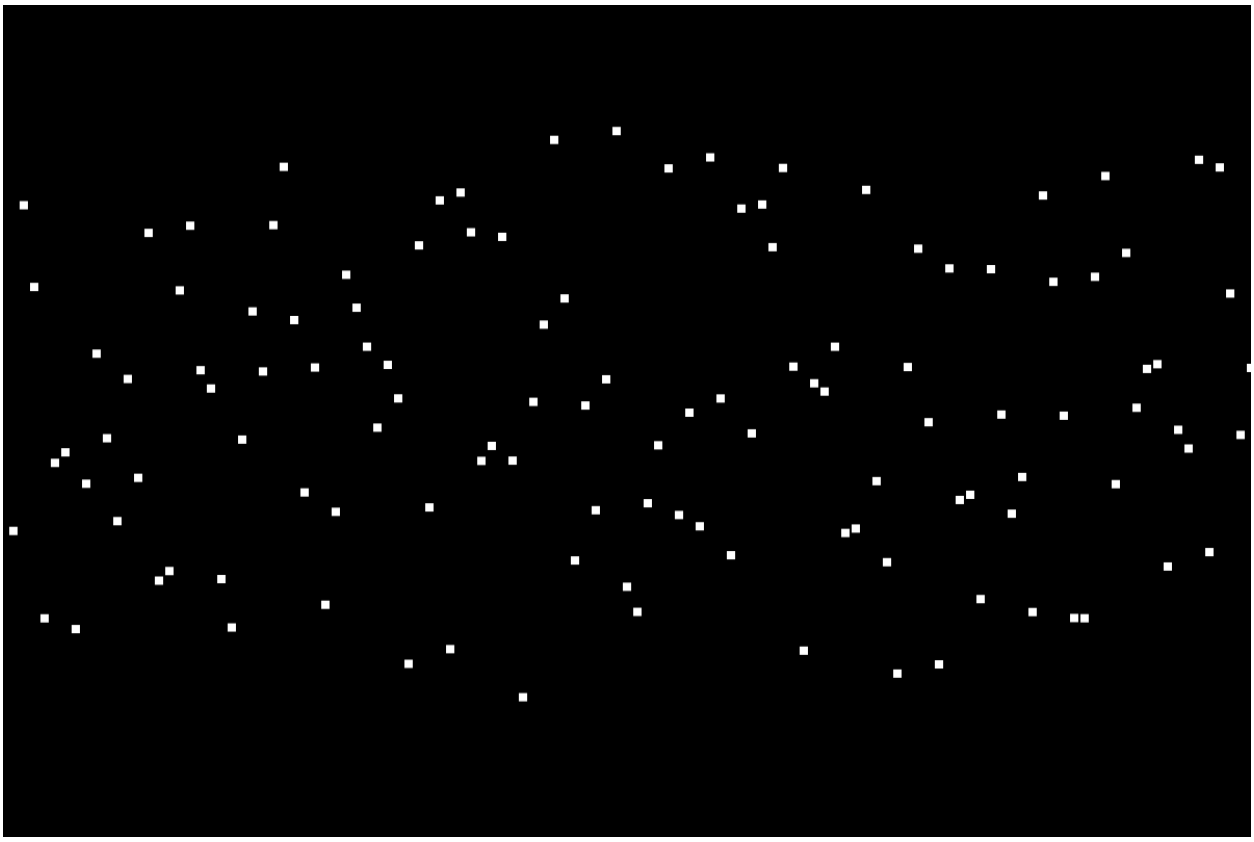
Another aspect of this function is that it gives us values between 0 and 1. So will require scaling if used inside our 600 by 400 canvas- using the map function. The reference page for the map function is here: https://p5js.org/reference/#/p5/map

The map function is used to scale a value from one range to another range. In this case we want to map `noise(i)` from a range of 0 to 1 to a range of 0 to the height of the canvas. The map function take 5 arguments. The first is the value to be mapped, the second 2 are the original range and the 4th and 5th are the range the value should be mapped to.

`map(noise(i), 0, 1, 0, height);`

Let's add this to our rectangles y position starting by creating a variable `ypos` and setting it to our mapped noise function.

```
for (let i = 0; i < width/xstep; i++) {
  fill(255);
  ypos = map(noise(i), 0, 1, 0, height); // calculate y position with noise
  rect(xpos + (xstep*i), ypos, xstep, xstep); // draw a rectangle
 }
```



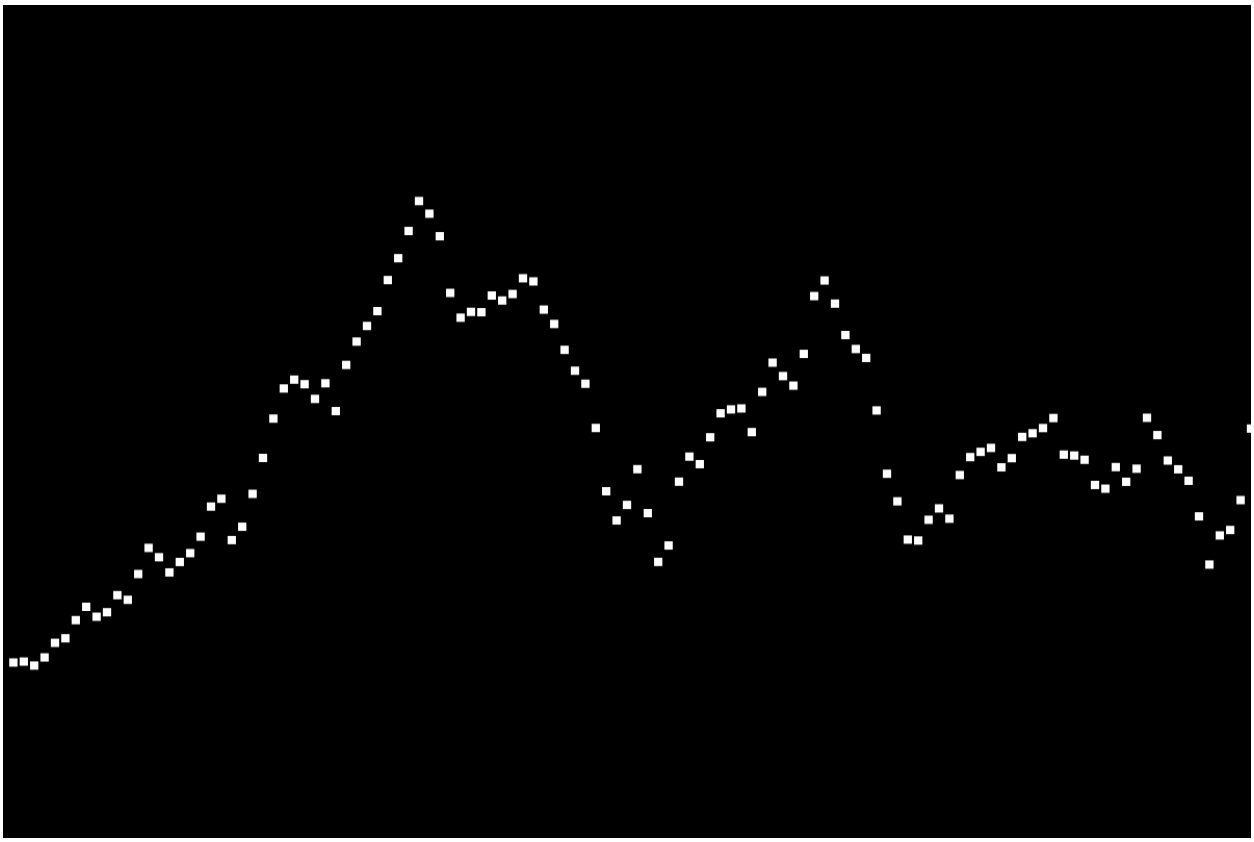
Looks a little too noisy!

Using `noise(i)` means that `i` , which is 0, 1, 2, 3 to the amount of rectangles, quite actually a large difference for function that operates between 0 and 1. We can adjust this by factoring the `i` when we use it as an argument in the noise function: `noise(i/10)`

I've made a new variable factor to store this so I can find it easier later:

`factor = i/10;`

```
for (let i = 0; i < width/xstep; i++) {
  factor = i/10;
  ypos = map(noise(factor), 0, 1, 0, height); // calculate y position with noise

  rect(xpos + (xstep*i), ypos, xstep, xstep); // draw a rectangle
}
```



Step 7: Use random() to change the fill

To make the sketch a little more interesting by changing the fill of each rectangle to a random value on each draw.

To do this I'll add random(255) to the fill argument. This randomly give a value between 0 and 255.

```
fill(random(255));
```

Step 8: mouseX and mouseY

Here we are going to introduce some simple user input by taking the mouse position to set the fill of rectangles. p5 makes the mouse position available using the `mouseX` and `mouseY` values.

The mouse values will be from within the canvas, so `mouseX` will be between 0 and the width of the canvas (0 to 600 in this case). So this is another case where we can use the `map()` function, as the fill value is between 0 and 255:

```
fill(map(mouseX, 0, width, 0, 255));
```

As we don't want the rectangles to completely disappear, we can change the resulting range to be mapped 100 to 255.

```
fill(map(mouseX, 0, width, 100, 255));
```

Run it a test it out.

Play

By now you should have lots to play with. Here is the final code- with the comment `//PLAY` on each line where you can see how adjusting the numbers could be interesting to start you off.

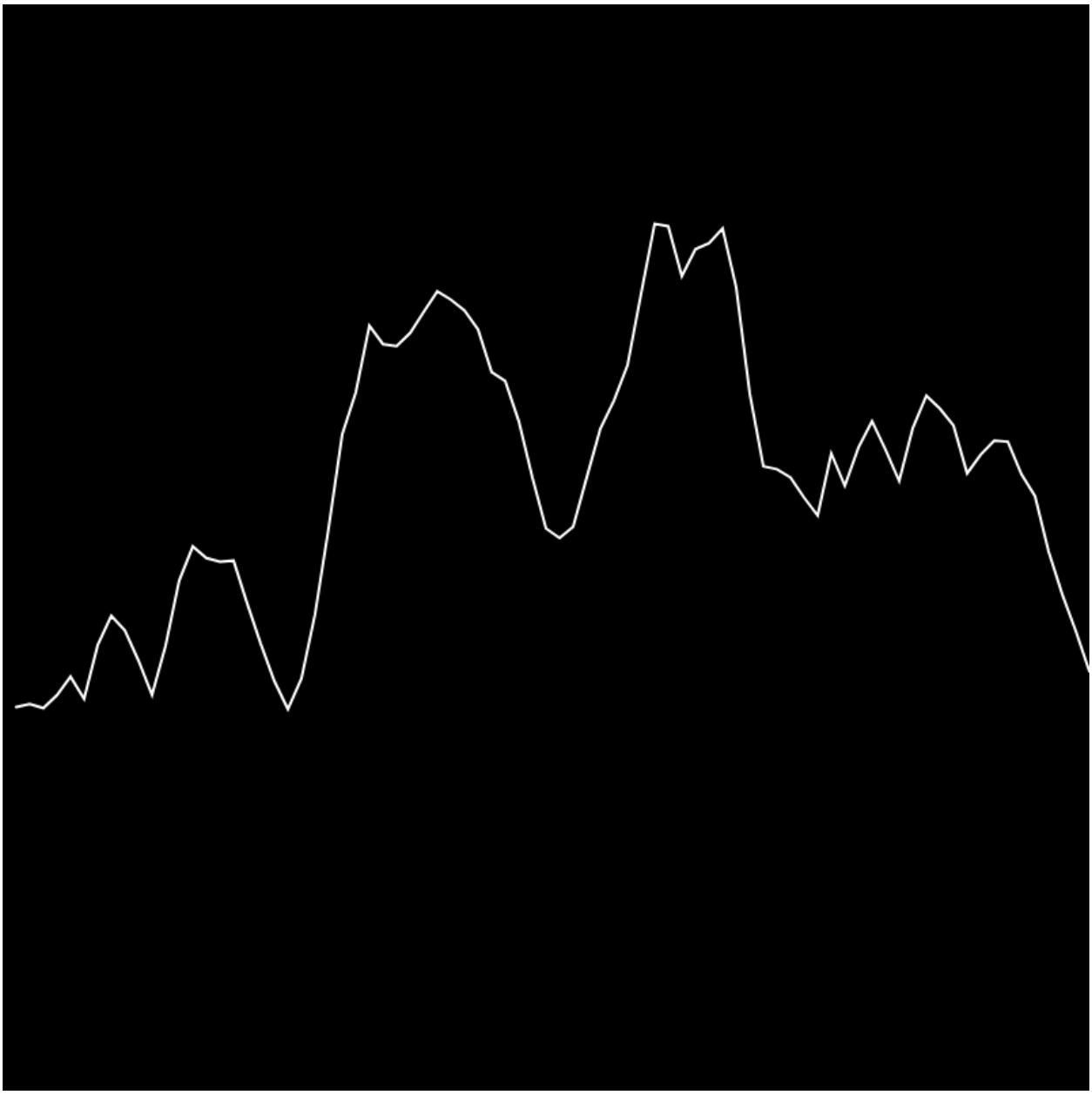
If you want to go deeper, try changing the `fill()` to be set with the `noise()` function.

```
let xpos = 5; //PLAY
let xstep = 5; //PLAY


function setup() {
  createCanvas(600, 400);
  rectMode(CENTER);
}


function draw() {
  background(0); //PLAY


  for (let i = 0; i < width/xstep; i++) {

    fill(map(mouseX, 0, width, 0, 255)); //PLAY

    factor = i/10; //PLAY
    ypos = map(noise(factor), 0, 1, 0, height);

    rect(xpos + (xstep*i), ypos, xstep, xstep);

  }

}
```

For a fully line by line commented code, you can find this on the git repo in week 5.

Coding Challenge

This weeks challenge is to adapt this sketch to create the following image.



Hint: One way is to use the beginShape() | endShape() commands with the for loop above but adapted with vertex()….