

*Algorithmic
Art
Workbook #10
Flowfields
and Files*



Table of Contents

MIT Licence	4
Workbook #10 Quick Content Summary	5
p5.js Versions	6
Changing the Version	7
An Infinite Loop	8
Module E Unit #5: perlin flowfield	10
Sketch E5.1 starting point	11
Sketch E5.2 vectors not pixels	13
Sketch E5.3 random squares	15
Sketch E5.4 perlin fill	17
Sketch E5.5 vector lines	19
Sketch E5.6 rotate	21
Sketch E5.7 random angle	23
Sketch E5.8 perlin angle	26
Sketch E5.9 third dimension	28
Sketch E5.10 particle class	30
Sketch E5.11 pos, vel and acc	32
Sketch E5.12 the force	34
Sketch E5.13 particle	36
Sketch E5.14 it is there honestly!	38
Sketch E5.15 many particles	41
Sketch E5.16 random particles	44
Sketch E5.17 falling off the edge	47
Sketch E5.18 dim the lines	50
Sketch E5.19 better particles	54
Sketch E5.20 array of vectors	58
Sketch E5.21 following the vectors	61
Sketch E5.22 the index value	65
Sketch E5.23 index force	69
Sketch E5.24 may the force be with you	73
Sketch E5.25 mad dash	77
Sketch E5.26 the magnitude of it all	81
Sketch E5.27 zoff off	85
Sketch E5.28 no lines	89
Sketch E5.29 nice visual	93
Sketch E5.30 alpha stroke	97
Sketch E5.31 the previous position	101
Sketch E5.32 updating everything	105
Sketch E5.33 edges before show	109

Challenges	113
Module E Unit #6: file management	115
Accessing the index.html file	116
File List	117
Inside the index.html File	118
But what do they mean?	119
Adding a file	120
Create a File	121
Referencing the file	122
Adding the code	123
A photo	124
Back to the files	125
Drop or browse	126
Renaming the file	127
What next?	128



MIT Licence

Copyright ©2026 Warren George

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NON-INFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

Brief summary:

Permissions (what you can do)

Commercial use
Modification
Distribution
Private use

Limitations (what is not covered)

Liability
Warranty



Workbook #10 Quick Content Summary

One of the ways you can develop very interesting effects is to use particles which have a particular property. There are various forms of random, and Perlin Noise is one that can be used to great effect.

With p5.js, you can upload files whether they are images, music, videos, fonts, or even 3D objects. You can also break down your code into more manageable chunks; for instance, if you create a class, then you might put that in a separate file (think of it as a folder or tab). This keeps your code very tidy rather than one long column of lines of code. It does also mean you can find the important bits quickly.

The only slight downside is that you have to come out of one file and jump into another. Although for the most part I have kept classes in the main sketch rather than separating them out, you will find when you look at the examples of other coders in p5.js that they use the file system extensively. That is why I have introduced the example at the end of this workbook.

Although I asked you to keep the Firework Display sketch safe, you can always use the Flowfields sketch instead with a slight adaptation. In fact, I would recommend that you have a go at doing just that.

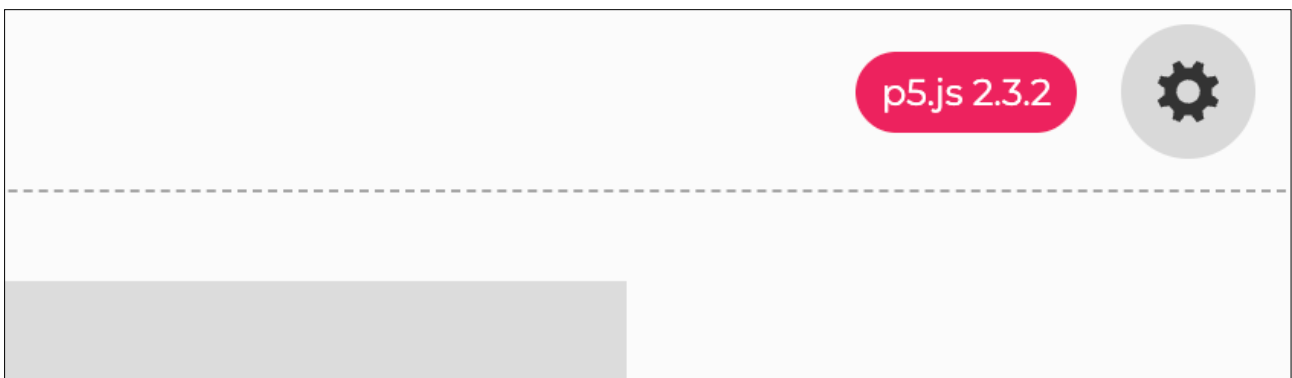


p5.js Versions

The latest version of p5.js (as of September 2026) is **2.3.3**. This has a few bugs which are being ironed out as we speak. So by the time you get to this point you may well find that my advice is irrelevant but just in case it is not I will make the following suggestions.

I suggest that you use version **2.2.3** because I have tested the code on that version and seems to work fine. You can easily switch versions. If in doubt then try different ones if a problem arises. This is especially true if you copy an older example from the internet.

Figure 1: version selection

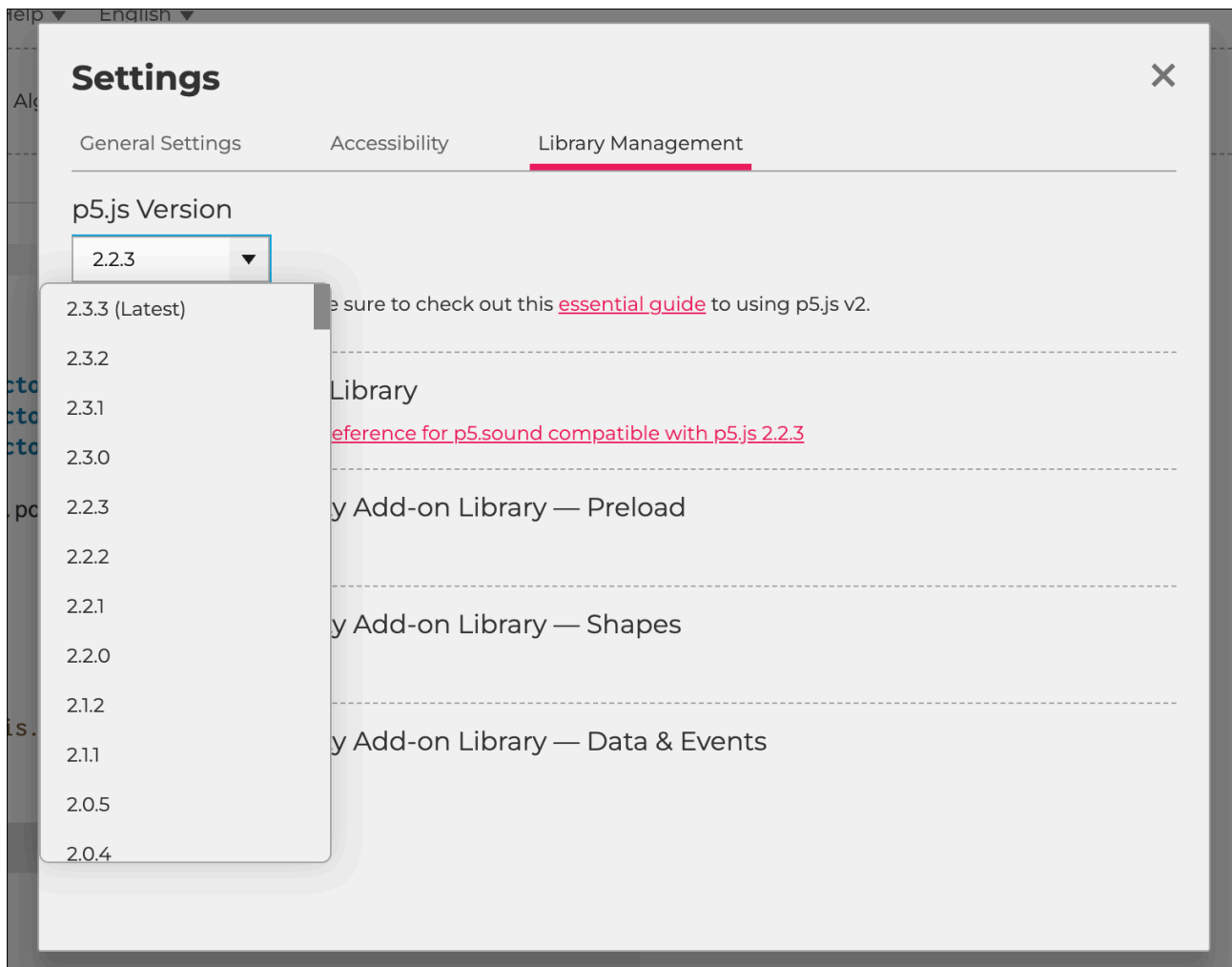




Changing the Version

You get a drop down menu like this shown in Fig.2:

Figure 2: changing the version

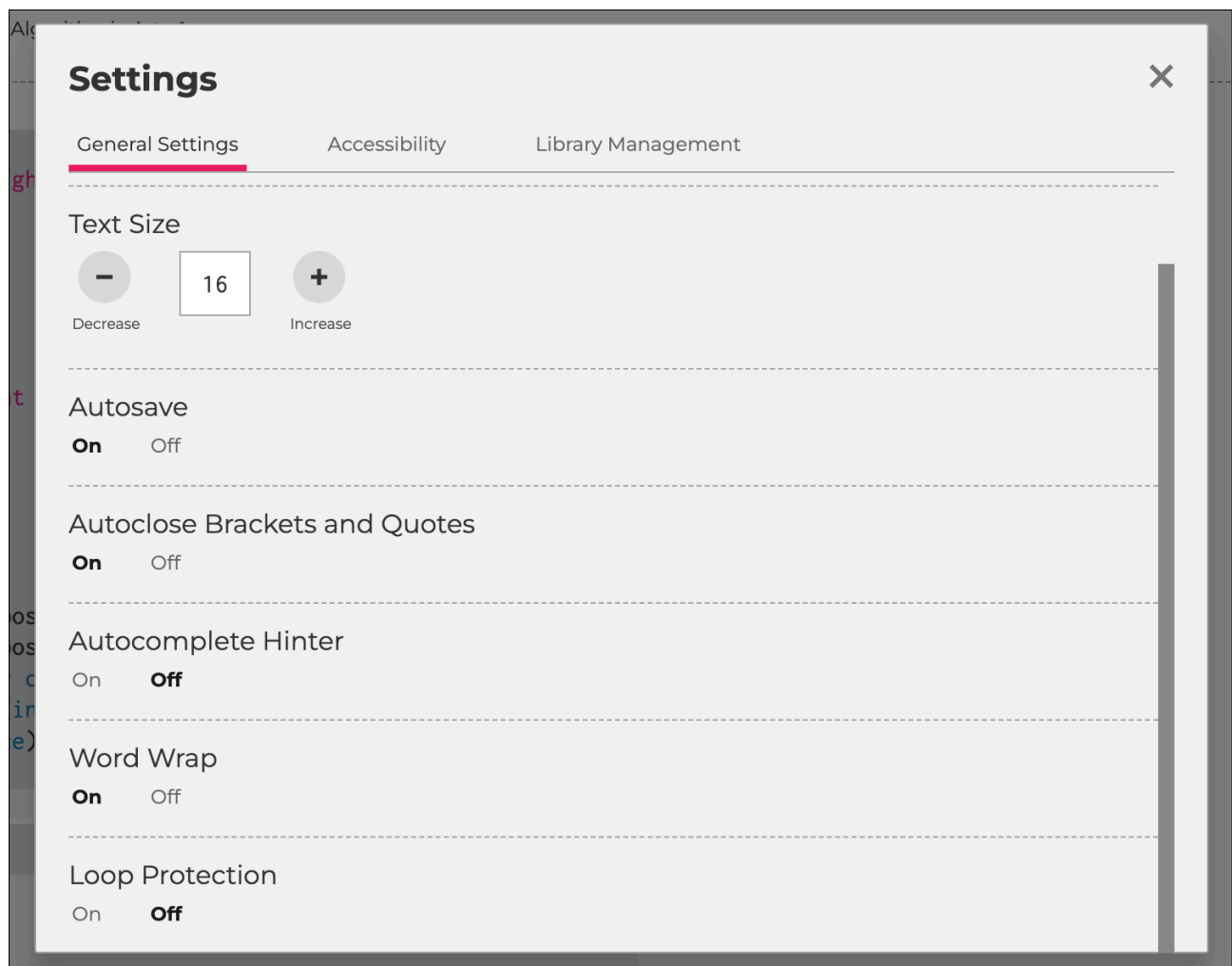


An Infinite Loop

If you get an **infinite loop detected** warning then you need to go to **Settings** (cog symbol top right), scroll down to Loop Protection, then turn this off and continue unless of course you have accidentally created an infinite loop that is.

A few sketches using perlin noise and pixels can create this problem. So I just want to alert you to the fact so that you are aware.

Figure 3: loop protection





Algorithmic
Art

Module E

Unit #5

Perlin

Flowfield



Module E Unit #5: perlin flowfield

We are going to replace pixels with arrows with Perlin noise to show a flowfield. We start where we left off with 2D Perlin noise and instead of a grayscale value we will have a vector which will point in a direction according to Perlin noise.

This is another long unit which covers a lot of concepts and ideas. This will take you step by step through creating a perlin



Sketch E5.1 starting point

! Our initial sketch.

We will start with familiar code from **module A unit #2**, Perlin 2D. Saves a bit of time, assuming you have completed that unit.

```
let xoff = 0
let yoff = 0
let inc = 0.01
let index
let val

function setup()
{
  createCanvas(400, 400)
  pixelDensity(1)
}

function draw()
{
  loadPixels()
  for (y = 0; y < height; y++)
  {
    xoff = 0
    for (x = 0; x < width; x++)
    {
      index = (x + y * width) * 4
      val = noise(xoff, yoff) * 255
      pixels[index + 0] = val
      pixels[index + 1] = val
      pixels[index + 2] = val
      pixels[index + 3] = 255
      xoff += inc
    }
    yoff += inc
  }
  updatePixels()
  noLoop()
```

```
}
```

Notes

This is what we had before.

Figure E5.1





Sketch E5.2 vectors not pixels

But we want vectors, not pixels. So we also don't want a vector for every pixel. So we will space them out with a variable, which we will call `scl` (short for scale). We will need columns (`cols`) and rows (`rows`). To get the right number of rows and columns, we divide the width and height by `scl` and also use `floor` so we have an integer number.

We can also get rid of all the references to `pixels` (`//` commented out and highlighted in blue).

```
let xoff = 0
let yoff = 0
let inc = 0.01
let index
let val

let scl = 20
let cols
let rows

function setup()
{
  createCanvas(400, 400)
  cols = floor(width / scl)
  rows = floor(height / scl)
  pixelDensity(1)
}

function draw()
{
  // loadPixels()
  for (y = 0; y < height; y++)
  {
    xoff = 0
    for (x = 0; x < width; x++)
    {
      index = (x + y * width) * 4
      val = noise(xoff, yoff) * 255
      // pixels[index + 0] = val
      // pixels[index + 1] = val
    }
  }
}
```

```

    // pixels[index + 2] = val
    // pixels[index + 3] = 255

    xoff += inc
  }
  yoff += inc
}

// updatePixels()

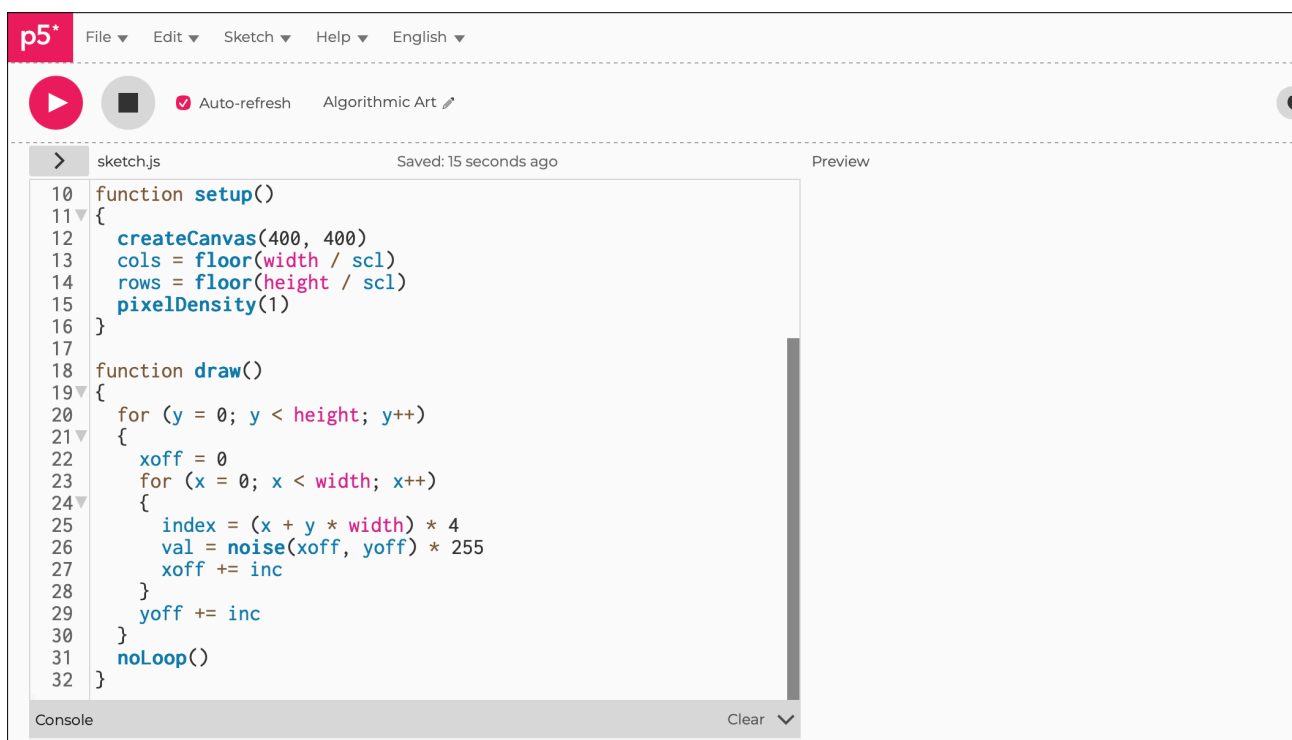
noLoop()
}

```

Notes

Nothing to show for it.

Figure E5.2





Sketch E5.3 random squares

To make sure everything is working ok we will use those variables we have just created to draw a square and fill it randomly.

```
let xoff = 0
let yoff = 0
let inc = 0.01
let index
let val
let scl = 20
let cols
let rows

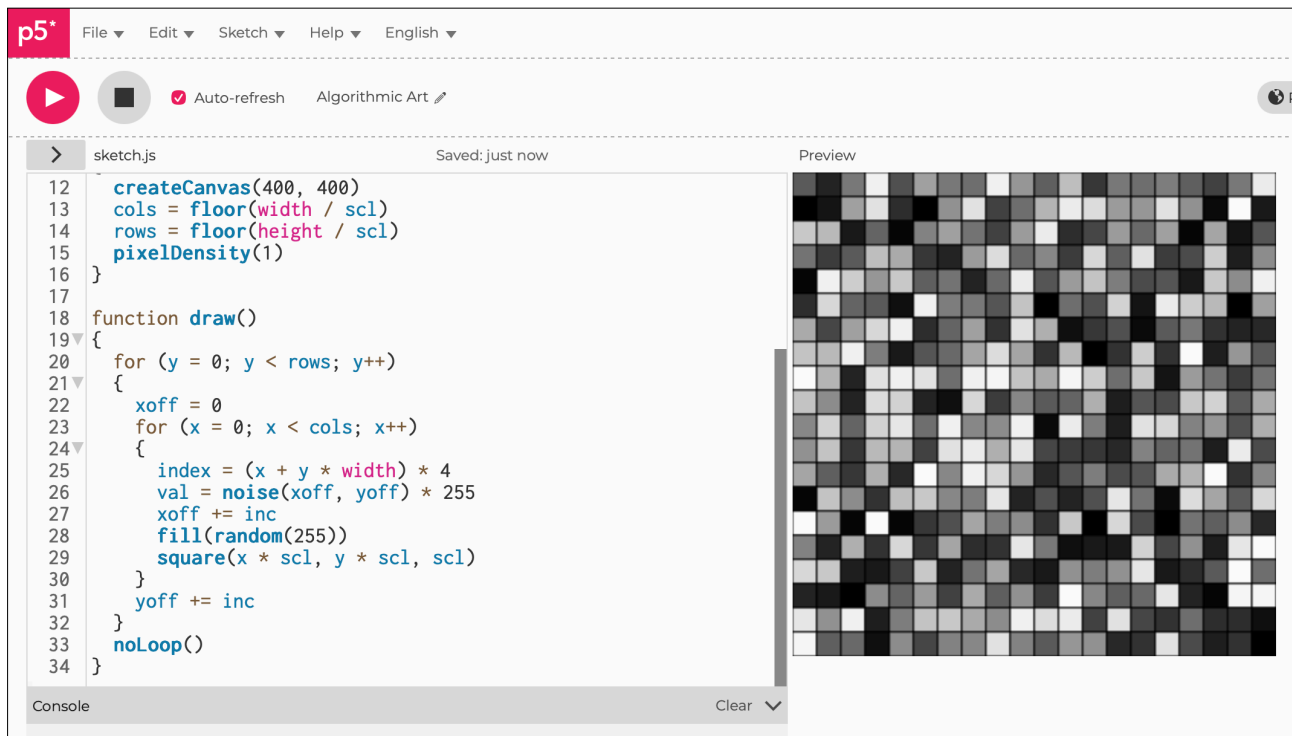
function setup()
{
  createCanvas(400, 400)
  cols = floor(width / scl)
  rows = floor(height / scl)
  pixelDensity(1)
}

function draw()
{
  for (y = 0; y < rows; y++)
  {
    xoff = 0
    for (x = 0; x < cols; x++)
    {
      index = (x + y * width) * 4
      val = noise(xoff, yoff) * 255
      xoff += inc
      fill(random(255))
      square(x * scl, y * scl, scl)
    }
    yoff += inc
  }
  noLoop()
}
```

Notes

Just randomly filled squares. The `pixelDensity()` makes it a little bit blurry.

Figure E5.3





Sketch E5.4 perlin fill

Instead of randomly filling the squares with greyscale, let's change it to Perlin. We already have the Perlin value as `val` and make the `inc` (increment) a little bigger.

```
let xoff = 0
let yoff = 0
let inc = 0.1
let index
let val
let scl = 20
let cols
let rows

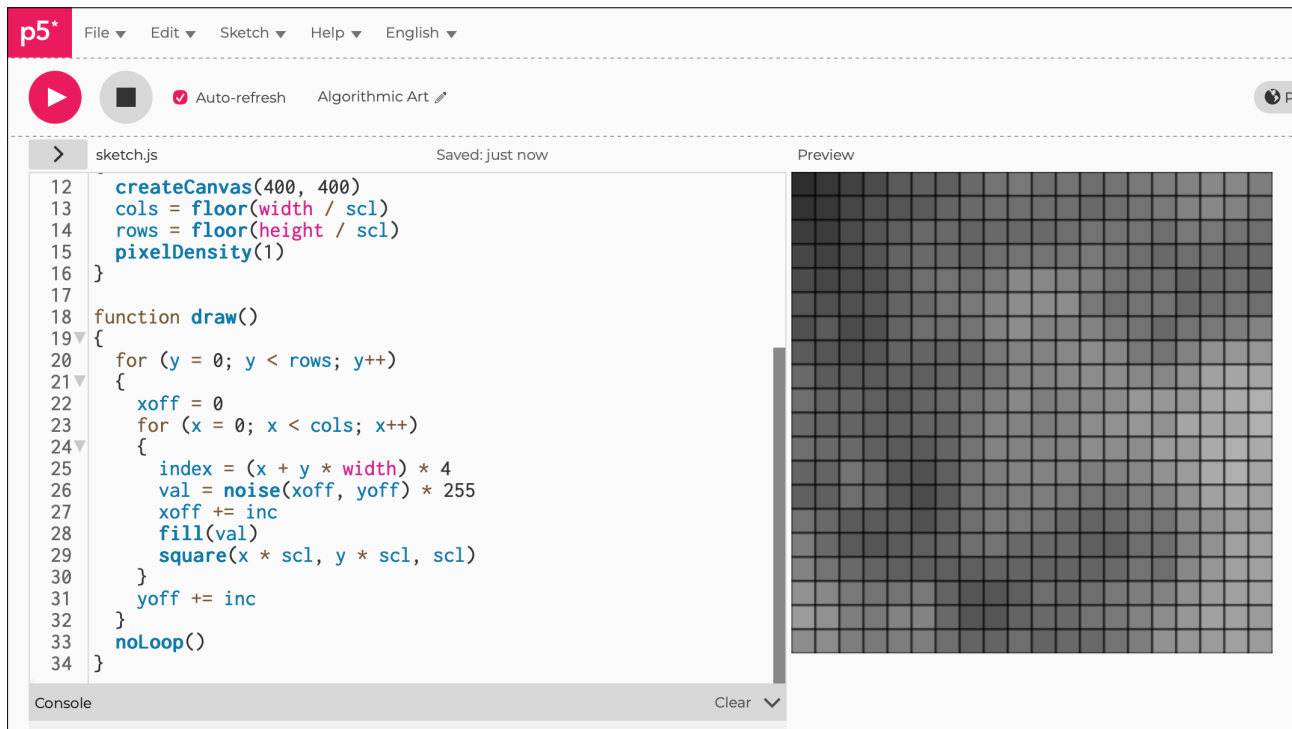
function setup()
{
  createCanvas(400, 400)
  cols = floor(width / scl)
  rows = floor(height / scl)
  pixelDensity(1)
}

function draw()
{
  for (y = 0; y < rows; y++)
  {
    xoff = 0
    for (x = 0; x < cols; x++)
    {
      index = (x + y * width) * 4
      val = noise(xoff, yoff) * 255
      xoff += inc
      fill(val)
      square(x * scl, y * scl, scl)
    }
    yoff += inc
  }
  noLoop()
}
```

Notes

The grey scaling is linked to the `Perlin noise()` value.

Figure E5.4





Sketch E5.5 vector lines

Instead of colouring in a square, we are going to draw a vector as a line, removing the `fill()` and `square()`.

```
let xoff = 0
let yoff = 0
let inc = 0.1
let index
let val
let scl = 20
let cols
let rows

function setup()
{
  createCanvas(400, 400)
  cols = floor(width / scl)
  rows = floor(height / scl)
  pixelDensity(1)
}

function draw()
{
  background(220)
  for (y = 0; y < rows; y++)
  {
    xoff = 0
    for (x = 0; x < cols; x++)
    {
      index = (x + y * width) * 4
      val = noise(xoff, yoff) * 255
      xoff += inc

      push()
      translate(x * scl, y * scl)
      line(0, 0, scl, 0)
      pop()
      // fill(val)
      // square(x * scl, y * scl, scl)
```

```

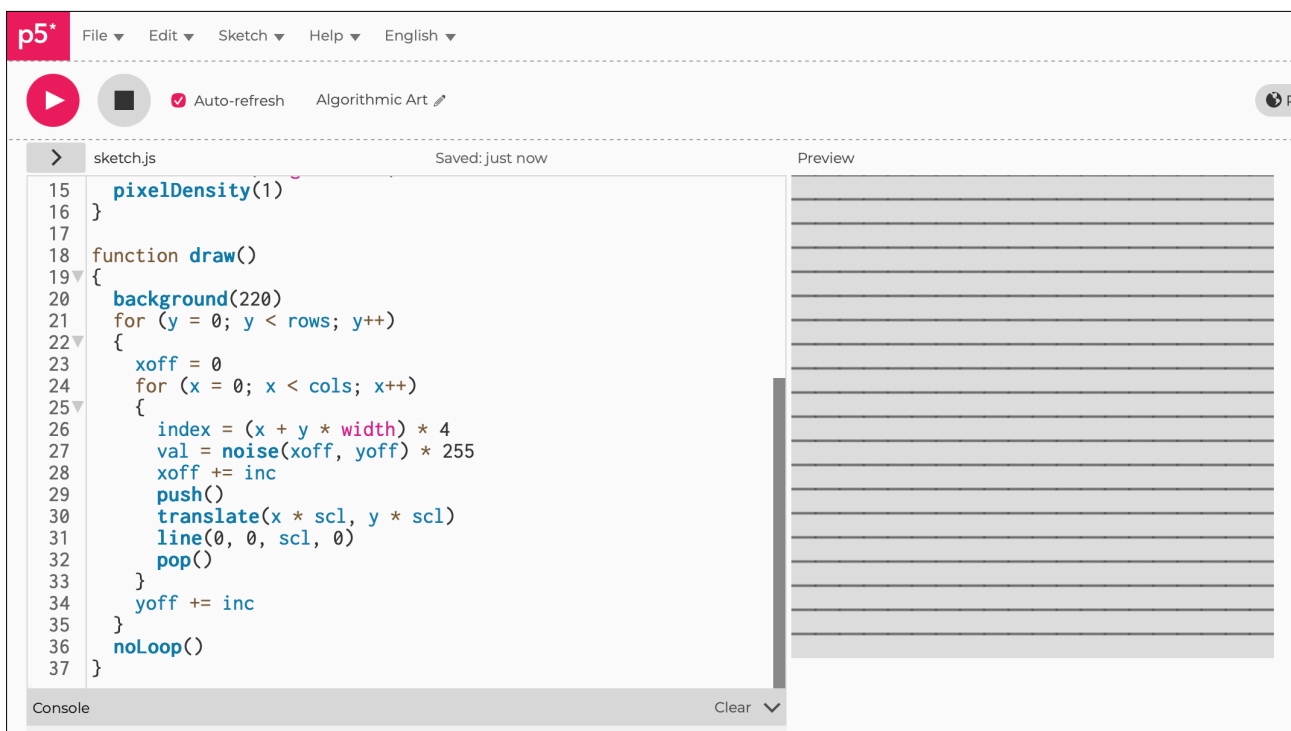
    }
    yoff += inc
  }
  noLoop()
}

```

Notes

We aren't making use of the `noise()` yet.

Figure E5.5





Sketch E5.6 rotate

We are going to rotate a vector from an angle using a p5 function called: `p5.Vector.fromAngle()` and using the `heading()` function. We will rotate by `0.5 radians`, which is `30° degrees`.

```
let xoff = 0
let yoff = 0
let inc = 0.1
let index
let val
let scl = 20
let cols
let rows

function setup()
{
  createCanvas(400, 400)
  cols = floor(width / scl)
  rows = floor(height / scl)
  pixelDensity(1)
}

function draw()
{
  background(220)
  for (y = 0; y < rows; y++)
  {
    xoff = 0
    for (x = 0; x < cols; x++)
    {
      index = (x + y * width) * 4
      val = noise(xoff, yoff) * 255
      xoff += inc

      let v = p5.Vector.fromAngle(0.5)
      push()
      translate(x * scl, y * scl)
      rotate(v.heading())
      line(0, 0, scl, 0)
```

```

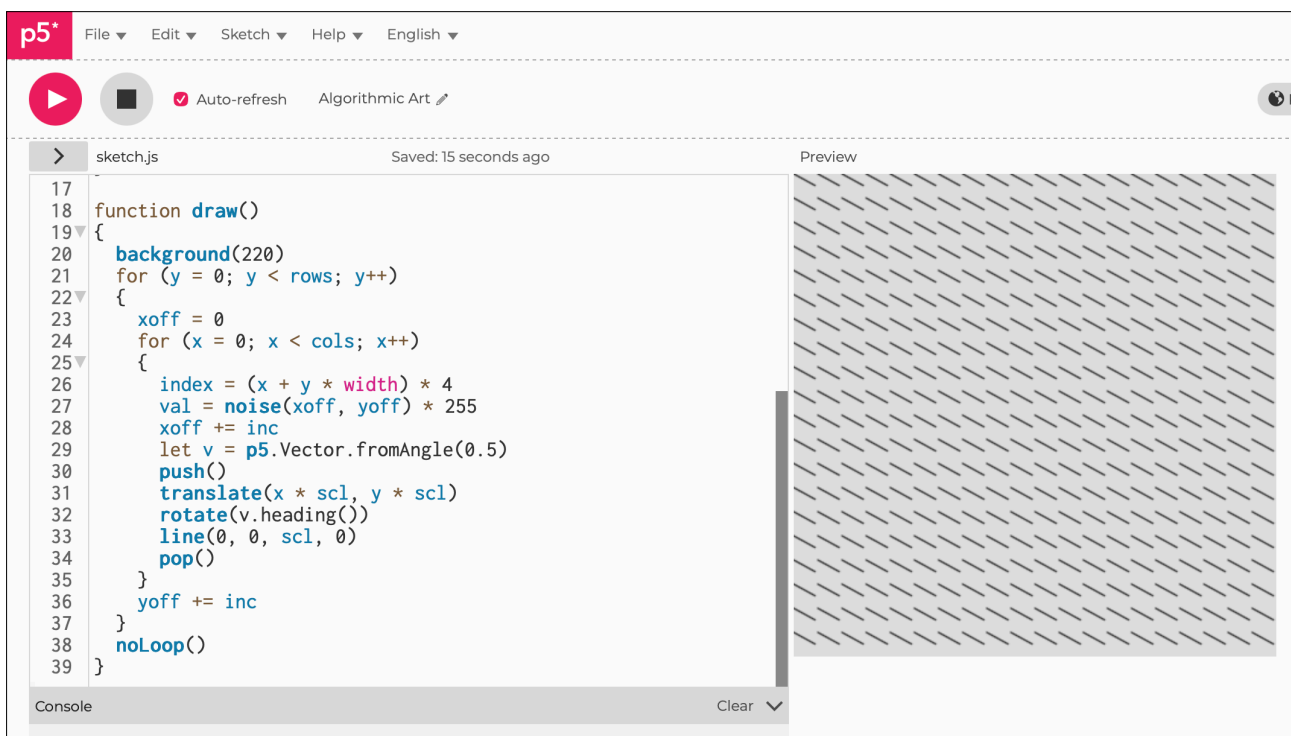
    pop()
  }
  yoff += inc
}
noLoop()
}

```

Notes

All angled nicely.

Figure E5.6





Sketch E5.7 random angle

Instead of a fixed angle, let us use `random()` from `0` to `2π`.

```
let xoff = 0
let yoff = 0
let inc = 0.1
let index
let val
let scl = 20
let cols
let rows

function setup()
{
  createCanvas(400, 400)
  cols = floor(width / scl)
  rows = floor(height / scl)
  pixelDensity(1)
}

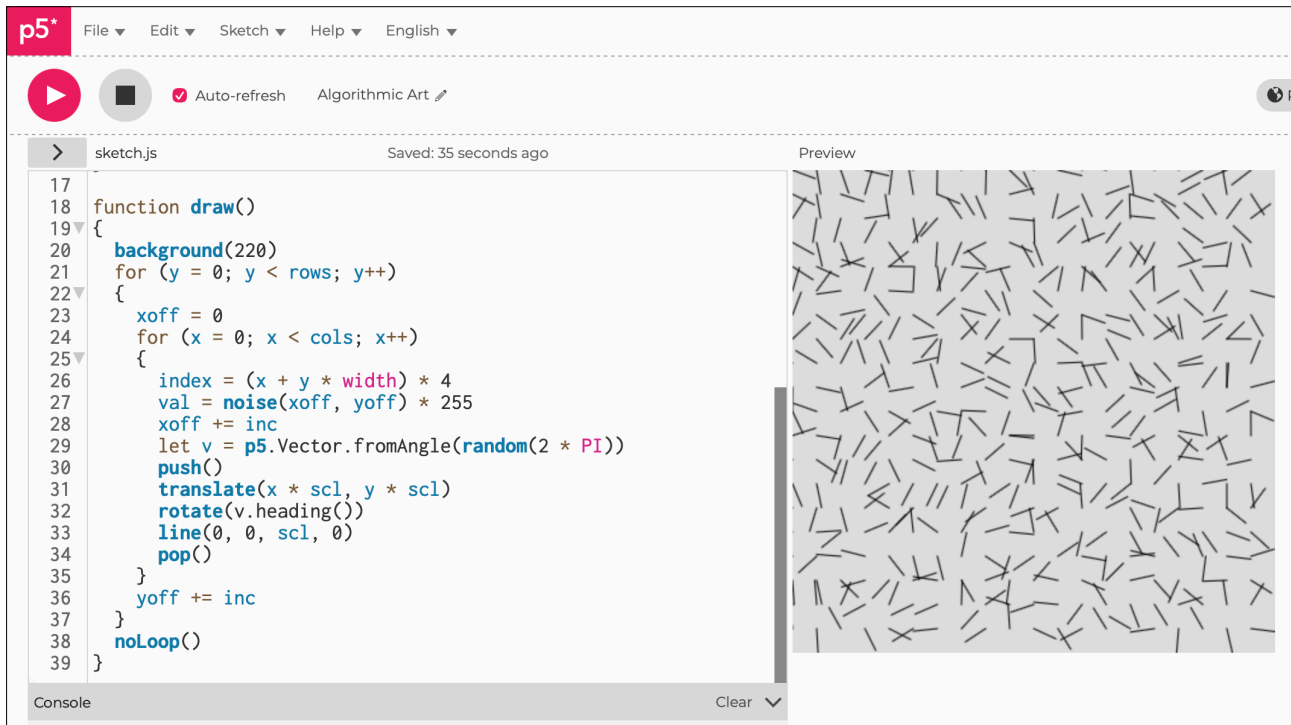
function draw()
{
  background(220)
  for (y = 0; y < rows; y++)
  {
    xoff = 0
    for (x = 0; x < cols; x++)
    {
      index = (x + y * width) * 4
      val = noise(xoff, yoff) * 255
      xoff += inc
      let v = p5.Vector.fromAngle(random(2 * PI))
      push()
      translate(x * scl, y * scl)
      rotate(v.heading())
      line(0, 0, scl, 0)
      pop()
    }
  }
}
```

```
    yoff += inc  
}  
noLoop()  
}
```

Notes

They are randomly arranged.

Figure E5.7





Sketch E5.8 perlin angle

Rather than pure random, we incorporate Perlin noise. We change the variable name from `val` to something more relevant (`angle`). We multiply by 2π (360°) and then draw the vector to that angle. Every time you refresh it, it changes, but you can see the Perlin noise effect.

```
let xoff = 0
let yoff = 0
let inc = 0.1
let index
let angle
let scl = 20
let cols
let rows

function setup()
{
  createCanvas(400, 400)
  cols = floor(width / scl)
  rows = floor(height / scl)
  pixelDensity(1)
}

function draw()
{
  background(220)
  for (y = 0; y < rows; y++)
  {
    xoff = 0
    for (x = 0; x < cols; x++)
    {
      index = (x + y * width) * 4
      angle = noise(xoff, yoff) * (2 * PI)
      xoff += inc
      let v = p5.Vector.fromAngle(angle)
      push()
      translate(x * scl, y * scl)
      rotate(v.heading())
```

```

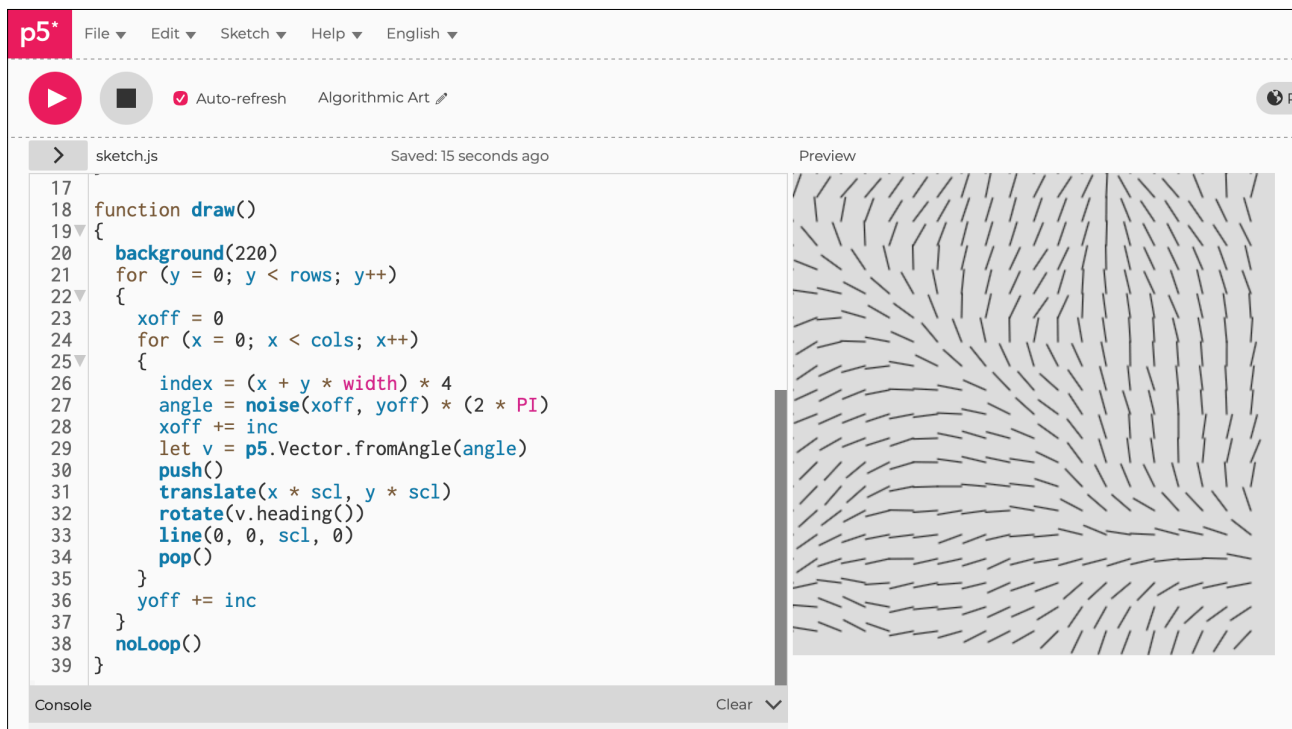
    line(0, 0, scl, 0)
    pop()
  }
  yoff += inc
}
noLoop()
}

```

Notes

The angles now have something in common with each other.

Figure E5.8





Sketch E5.9 third dimension

Let's create a third dimension for time and call it **zoff**. We will change the time in small increments. This will animate as if we have slices of time. Each slice is connected to the previous one through Perlin noise. We need to make a few alterations to have a slow animated Perlin noise effect. You can play with the parameters.

```
let xoff = 0
let yoff = 0
let zoff = 0
let inc = 0.1
let index
let angle
let scl = 20
let cols
let rows

function setup()
{
  createCanvas(400, 400)
  cols = floor(width / scl)
  rows = floor(height / scl)
  pixelDensity(1)
}

function draw()
{
  background(220)
  yoff = 0
  for (y = 0; y < rows; y++)
  {
    xoff = 0
    for (x = 0; x < cols; x++)
    {
      index = (x + y * width) * 4
      angle = noise(xoff, yoff, zoff) * (2 * PI)
      xoff += inc
      let v = p5.Vector.fromAngle(angle)
      push()
```

```
    translate(x * scl, y * scl)
    rotate(v.heading())
    line(0, 0, scl, 0)
    pop()
}
yoff += inc
zoff += 0.0005
}
// noLoop()
}
```

Notes

You now get a very pleasing, slowly moving, choreographed set of vectors.



Sketch E5.10 particle class

Now we add the Particle class. We have added a `constructor()` function to the `Particle` class.

```
let xoff = 0
let yoff = 0
let zoff = 0
let inc = 0.1
let index
let angle
let scl = 20
let cols
let rows

function setup()
{
  createCanvas(400, 400)
  cols = floor(width / scl)
  rows = floor(height / scl)
  pixelDensity(1)
}

function draw()
{
  background(220)
  yoff = 0
  for (y = 0; y < rows; y++)
  {
    xoff = 0
    for (x = 0; x < cols; x++)
    {
      index = (x + y * width) * 4
      angle = noise(xoff, yoff, zoff) * (2 * PI)
      xoff += inc
      let v = p5.Vector.fromAngle(angle)
      push()
      translate(x * scl, y * scl)
      rotate(v.heading())
```

```
    line(0, 0, scl, 0)
    pop()
}
yoff += inc
zoff += 0.0005
}
// noLoop()
}
```

```
class Particle
{
  constructor()
  {
    this.pos = createVector(0, 0)
    this.vel = createVector(0, 0)
    this.acc = createVector(0, 0)
  }
}
```

Notes

We have created three vectors for the **position**, the **velocity**, and the **acceleration**.



Sketch E5.11 pos, vel and acc

We will add in the **position**, **velocity**, and **acceleration** to the **move()** function and reset the acceleration to zero.

```
let xoff = 0
let yoff = 0
let zoff = 0
let inc = 0.1
let index
let angle
let scl = 20
let cols
let rows

function setup()
{
  createCanvas(400, 400)
  cols = floor(width / scl)
  rows = floor(height / scl)
  pixelDensity(1)
}

function draw()
{
  background(220)
  yoff = 0
  for (y = 0; y < rows; y++)
  {
    xoff = 0
    for (x = 0; x < cols; x++)
    {
      index = (x + y * width) * 4
      angle = noise(xoff, yoff, zoff) * (2 * PI)
      xoff += inc
      let v = p5.Vector.fromAngle(angle)
      push()
      translate(x * scl, y * scl)
      rotate(v.heading())
```

```

        line(0, 0, scl, 0)
        pop()
    }
    yoff += inc
    zoff += 0.0005
}
// noLoop()
}

class Particle
{
    constructor()
    {
        this.pos = createVector(0, 0)
        this.vel = createVector(0, 0)
        this.acc = createVector(0, 0)
    }

    move()
    {
        this.vel.add(this.acc)
        this.pos.add(this.vel)
        this.acc.mult(0)
    }
}

```

Notes

Nothing is changing just yet.



Sketch E5.12 the force

A force is applied to an acceleration

```
let xoff = 0
let yoff = 0
let zoff = 0
let inc = 0.1
let index
let angle
let scl = 20
let cols
let rows

function setup()
{
  createCanvas(400, 400)
  cols = floor(width / scl)
  rows = floor(height / scl)
  pixelDensity(1)
}

function draw()
{
  background(220)
  yoff = 0
  for (y = 0; y < rows; y++)
  {
    xoff = 0
    for (x = 0; x < cols; x++)
    {
      index = (x + y * width) * 4
      angle = noise(xoff, yoff, zoff) * (2 * PI)
      xoff += inc
      let v = p5.Vector.fromAngle(angle)
      push()
      translate(x * scl, y * scl)
      rotate(v.heading())
      line(0, 0, scl, 0)
```

```

        pop()
    }
    yoff += inc
    zoff += 0.0005
}
// noLoop()
}

class Particle
{
    constructor()
    {
        this.pos = createVector(0, 0)
        this.vel = createVector(0, 0)
        this.acc = createVector(0, 0)
    }

    move()
    {
        this.vel.add(this.acc)
        this.pos.add(this.vel)
        this.acc.mult(0)
    }

    applyForce(force)
    {
        this.acc.add(force)
    }
}

```

Notes

Still nothing new.



Sketch E5.13 particle

Now, to draw the particle.

```
let xoff = 0
let yoff = 0
let zoff = 0
let inc = 0.1
let index
let angle
let scl = 20
let cols
let rows

function setup()
{
  createCanvas(400, 400)
  cols = floor(width / scl)
  rows = floor(height / scl)
  pixelDensity(1)
}

function draw()
{
  background(220)
  yoff = 0
  for (y = 0; y < rows; y++)
  {
    xoff = 0
    for (x = 0; x < cols; x++)
    {
      index = (x + y * width) * 4
      angle = noise(xoff, yoff, zoff) * (2 * PI)
      xoff += inc
      let v = p5.Vector.fromAngle(angle)
      push()
      translate(x * scl, y * scl)
      rotate(v.heading())
      line(0, 0, scl, 0)
```

```

        pop()
    }
    yoff += inc
    zoff += 0.0005
}
// noLoop()
}

class Particle
{
    constructor()
    {
        this.pos = createVector(0, 0)
        this.vel = createVector(0, 0)
        this.acc = createVector(0, 0)
    }

    show()
    {
        point(this.pos.x, this.pos.y)
    }

    move()
    {
        this.vel.add(this.acc)
        this.pos.add(this.vel)
        this.acc.mult(0)
    }

    applyForce(force)
    {
        this.acc.add(force)
    }
}

```

Notes

Still nothing to see here. We need to go to the main sketch and reference it there.



Sketch E5.14 it is there honestly!

Let us draw the particle. You may not be able to see it very well, but it is there, honest. We create a particle array and put one particle in it at **index [0]**.

```
let xoff = 0
let yoff = 0
let zoff = 0
let inc = 0.1
let index
let angle
let scl = 20
let cols
let rows

let particles = []

function setup()
{
  createCanvas(400, 400)
  cols = floor(width / scl)
  rows = floor(height / scl)
  pixelDensity(1)
  particles[0] = new Particle()
}

function draw()
{
  background(220)
  yoff = 0
  for (y = 0; y < rows + 1; y++)
  {
    xoff = 0
    for (x = 0; x < cols + 1; x++)
    {
      index = (x + y * width) * 4
      angle = noise(xoff, yoff, zoff) * (2 * PI)
      xoff += inc
      let v = p5.Vector.fromAngle(angle)
      push()
```

```

        translate(x * scl, y * scl)
        rotate(v.heading())
        line(0, 0, scl, 0)
        pop()
    }
    yoff += inc
    zoff += 0.0005
}

```

```

particles[0].show()
particles[0].move()

```

```

}

```

```

class Particle

```

```

{
    constructor()
    {
        this.pos = createVector(0, 0)
        this.vel = createVector(0, 0)
        this.acc = createVector(0, 0)
    }

```

```

    show()
    {
        point(this.pos.x, this.pos.y)
    }

```

```

    move()
    {
        this.vel.add(this.acc)
        this.pos.add(this.vel)
        this.acc.mult(0)
    }

```

```

    applyForce(force)
    {
        this.acc.add(force)
    }

```

```

}

```

Notes

Still not a lot to see.



Sketch E5.15 many particles

Let's make 100 particles with a `for()` loop.

```
let xoff = 0
let yoff = 0
let zoff = 0
let inc = 0.1
let index
let angle
let scl = 20
let cols
let rows
let particles = []

function setup()
{
  createCanvas(400, 400)
  cols = floor(width / scl)
  rows = floor(height / scl)
  pixelDensity(1)
  for (let i = 0; i < 100; i++)
  {
    particles[i] = new Particle()
  }
}

function draw()
{
  background(220)
  yoff = 0
  for (y = 0; y < rows + 1; y++)
  {
    xoff = 0
    for (x = 0; x < cols + 1; x++)
    {
      index = (x + y * width) * 4
      angle = noise(xoff, yoff, zoff) * (2 * PI)
      xoff += inc
```

```

    let v = p5.Vector.fromAngle(angle)
    push()
    translate(x * scl, y * scl)
    rotate(v.heading())
    line(0, 0, scl, 0)
    pop()
  }
  yoff += inc
  zoff += 0.0005
}

for (let i = 0; i < particles.length; i++)
{
  particles[i].show()
  particles[i].move()
}

}

class Particle
{
  constructor()
  {
    this.pos = createVector(0, 0)
    this.vel = createVector(0, 0)
    this.acc = createVector(0, 0)
  }

  show()
  {
    point(this.pos.x, this.pos.y)
  }

  move()
  {
    this.vel.add(this.acc)
    this.pos.add(this.vel)
    this.acc.mult(0)
  }
}

```

```
    applyForce(force)
  {
    this.acc.add(force)
  }
}
```

Notes

Just be patient.



Sketch E5.16 random particles

We can check if this is working in the Particle class and give them a random velocity.

```
let xoff = 0
let yoff = 0
let zoff = 0
let inc = 0.1
let index
let angle
let scl = 20
let cols
let rows
let particles = []

function setup()
{
  createCanvas(400, 400)
  cols = floor(width / scl)
  rows = floor(height / scl)
  pixelDensity(1)
  for (let i = 0; i < 100; i++)
  {
    particles[i] = new Particle()
  }
}

function draw()
{
  background(220)
  yoff = 0
  for (y = 0; y < rows + 1; y++)
  {
    xoff = 0
    for (x = 0; x < cols + 1; x++)
    {
      index = (x + y * width) * 4
      angle = noise(xoff, yoff, zoff) * (2 * PI)
      xoff += inc
```

```

    let v = p5.Vector.fromAngle(angle)
    push()
    translate(x * scl, y * scl)
    rotate(v.heading())
    line(0, 0, scl, 0)
    pop()
  }
  yoff += inc
  zoff += 0.0005
}
for (let i = 0; i < particles.length; i++)
{
  particles[i].show()
  particles[i].move()
}
}

class Particle
{
  constructor()
  {
    this.pos = createVector(random(width), random(height))
    this.vel = p5.Vector.random2D()
    this.acc = createVector(0, 0)
  }

  show()
  {
    point(this.pos.x, this.pos.y)
  }

  move()
  {
    this.vel.add(this.acc)
    this.pos.add(this.vel)
    this.acc.mult(0)
  }
}

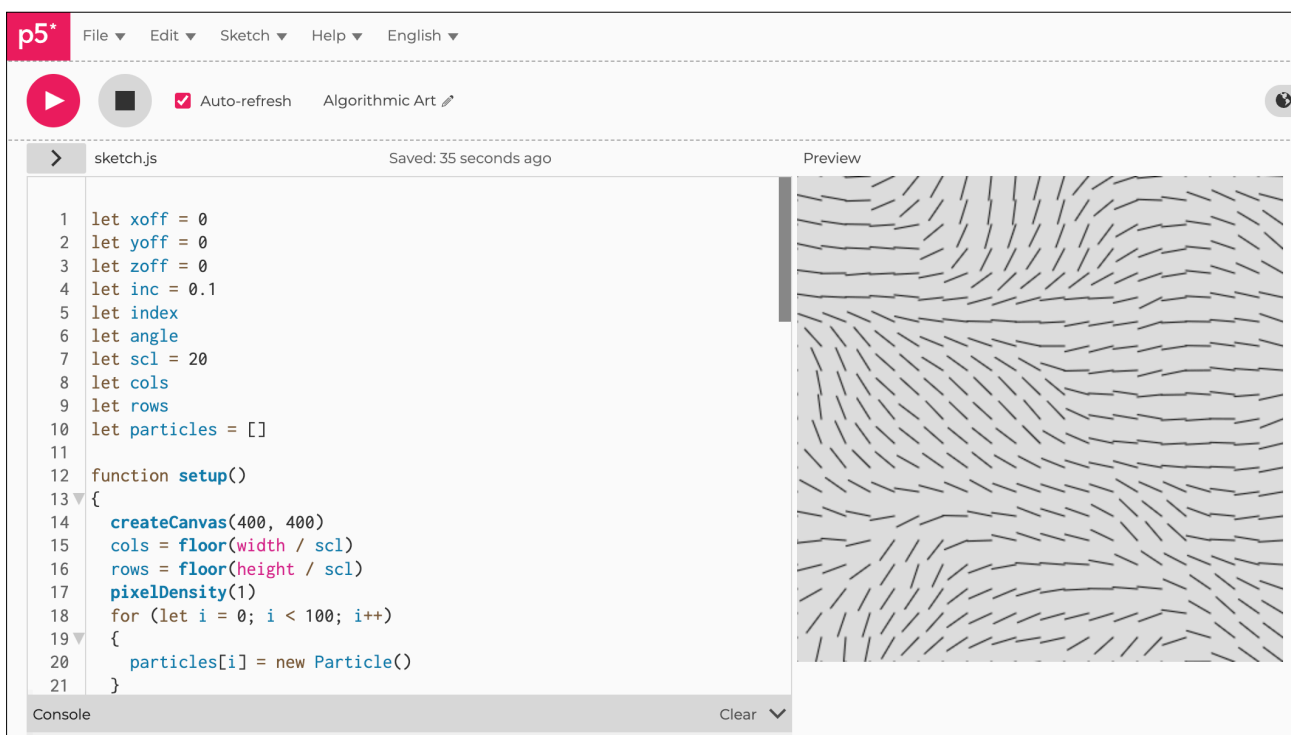
```

```
    applyForce(force)
  {
    this.acc.add(force)
  }
}
```

Notes

You should see them if you look closely.

Figure E5.16





Sketch E5.17 falling off the edge

To stop them disappearing off the edge of the canvas, we will create an `edge()` function boundary in the Particle class. In the `setup()` part of the main sketch.

```
let xoff = 0
let yoff = 0
let zoff = 0
let inc = 0.1
let index
let angle
let scl = 20
let cols
let rows
let particles = []

function setup()
{
  createCanvas(400, 400)
  cols = floor(width / scl)
  rows = floor(height / scl)
  pixelDensity(1)
  for (let i = 0; i < 100; i++)
  {
    particles[i] = new Particle()
  }
}

function draw()
{
  background(220)
  yoff = 0
  for (y = 0; y < rows + 1; y++)
  {
    xoff = 0
    for (x = 0; x < cols + 1; x++)
    {
      index = (x + y * width) * 4
      angle = noise(xoff, yoff, zoff) * (2 * PI)
```

```

    xoff += inc
    let v = p5.Vector.fromAngle(angle)
    push()
    translate(x * scl, y * scl)
    rotate(v.heading())
    line(0, 0, scl, 0)
    pop()
  }
  yoff += inc
  zoff += 0.0005
}
for (let i = 0; i < particles.length; i++)
{
  particles[i].show()
  particles[i].move()
}
}

class Particle
{
  constructor()
  {
    this.pos = createVector(random(width), random(height))
    this.vel = p5.Vector.random2D()
    this.acc = createVector(0, 0)
  }

  show()
  {
    point(this.pos.x, this.pos.y)
  }

  move()
  {
    this.vel.add(this.acc)
    this.pos.add(this.vel)
    this.acc.mult(0)
  }
}

```

```
}

applyForce(force)
{
  this.acc.add(force)
}

edges()
{
  if (this.pos.x > width)
  {
    this.pos.x = 0
  }
  if (this.pos.x < 0)
  {
    this.pos.x = width
  }
  if (this.pos.y > height)
  {
    this.pos.y = 0
  }
  if (this.pos.y < 0)
  {
    this.pos.y = height
  }
}
}
```

Notes

No change, do you know why? I bet you do.



Sketch E5.18 dim the lines

To dim the lines a bit, make the following adjustments (as well as adding the `edges()` function). Remove the `pixelDensity()`.

```
let xoff = 0
let yoff = 0
let zoff = 0
let inc = 0.1
let index
let angle
let scl = 20
let cols
let rows
let particles = []

function setup()
{
  createCanvas(400, 400)
  cols = floor(width / scl)
  rows = floor(height / scl)
  // pixelDensity(1)
  for (let i = 0; i < 100; i++)
  {
    particles[i] = new Particle()
  }
}

function draw()
{
  background(220)
  yoff = 0
  for (y = 0; y < rows + 1; y++)
  {
    xoff = 0
    for (x = 0; x < cols + 1; x++)
    {
      index = (x + y * width) * 4
      angle = noise(xoff, yoff, zoff) * (2 * PI)
```

```

    xoff += inc
    let v = p5.Vector.fromAngle(angle)
    push()
    translate(x * scl, y * scl)
    rotate(v.heading())
    strokeWeight(1)
    stroke(0, 50)
    line(0, 0, scl, 0)
    pop()
  }
  yoff += inc
  zoff += 0.0005
}
for (let i = 0; i < particles.length; i++)
{
  particles[i].show()
  particles[i].move()
  particles[i].edges()
}
}

class Particle
{
  constructor()
  {
    this.pos = createVector(random(width), random(height))
    this.vel = p5.Vector.random2D()
    this.acc = createVector(0, 0)
  }

  show()
  {
    point(this.pos.x, this.pos.y)
  }

  move()
  {
    this.vel.add(this.acc)
  }
}

```

```
    this.pos.add(this.vel)
    this.acc.mult(0)
  }

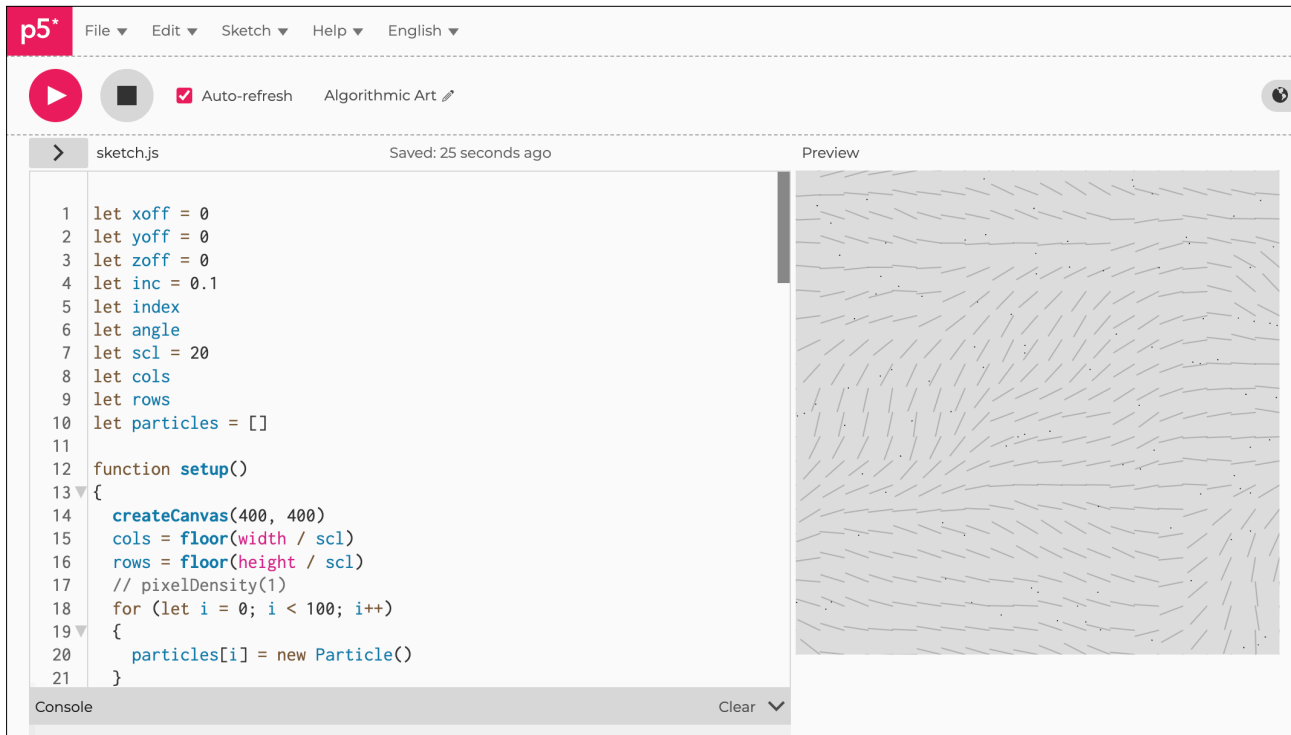
  applyForce(force)
  {
    this.acc.add(force)
  }

  edges()
  {
    if (this.pos.x > width)
    {
      this.pos.x = 0
    }
    if (this.pos.x < 0)
    {
      this.pos.x = width
    }
    if (this.pos.y > height)
    {
      this.pos.y = 0
    }
    if (this.pos.y < 0)
    {
      this.pos.y = height
    }
  }
}
```

Notes

You can see the particles flowing much better.

Figure E5.18





Sketch E5.19 better particles

In the Particle class, I want to see the particles clearly. If you are using the latest version you will get some distortion, so I recommend that if that is the case you switch to version **2.2.3** now if not already using it.

```
let xoff = 0
let yoff = 0
let zoff = 0
let inc = 0.1
let index
let angle
let scl = 20
let cols
let rows
let particles = []

function setup()
{
  createCanvas(400, 400)
  cols = floor(width / scl)
  rows = floor(height / scl)
  for (let i = 0; i < 100; i++)
  {
    particles[i] = new Particle()
  }
}

function draw()
{
  background(220)
  yoff = 0
  for (y = 0; y < rows + 1; y++)
  {
    xoff = 0
    for (x = 0; x < cols + 1; x++)
    {
      index = (x + y * width) * 4
      angle = noise(xoff, yoff, zoff) * (2 * PI)
```

```

    xoff += inc
    let v = p5.Vector.fromAngle(angle)
    push()
    translate(x * scl, y * scl)
    rotate(v.heading())
    strokeWeight(1)
    stroke(0, 50)
    line(0, 0, scl, 0)
    pop()
  }
  yoff += inc
  zoff += 0.0005
}
for (let i = 0; i < particles.length; i++)
{
  particles[i].show()
  particles[i].move()
  particles[i].edges()
}
}

class Particle
{
  constructor()
  {
    this.pos = createVector(random(width), random(height))
    this.vel = p5.Vector.random2D()
    this.acc = createVector(0, 0)
  }

  show()
  {
    strokeWeight(3)
    point(this.pos.x, this.pos.y)
  }

  move()
  {

```

```
    this.vel.add(this.acc)
    this.pos.add(this.vel)
    this.acc.mult(0)
}

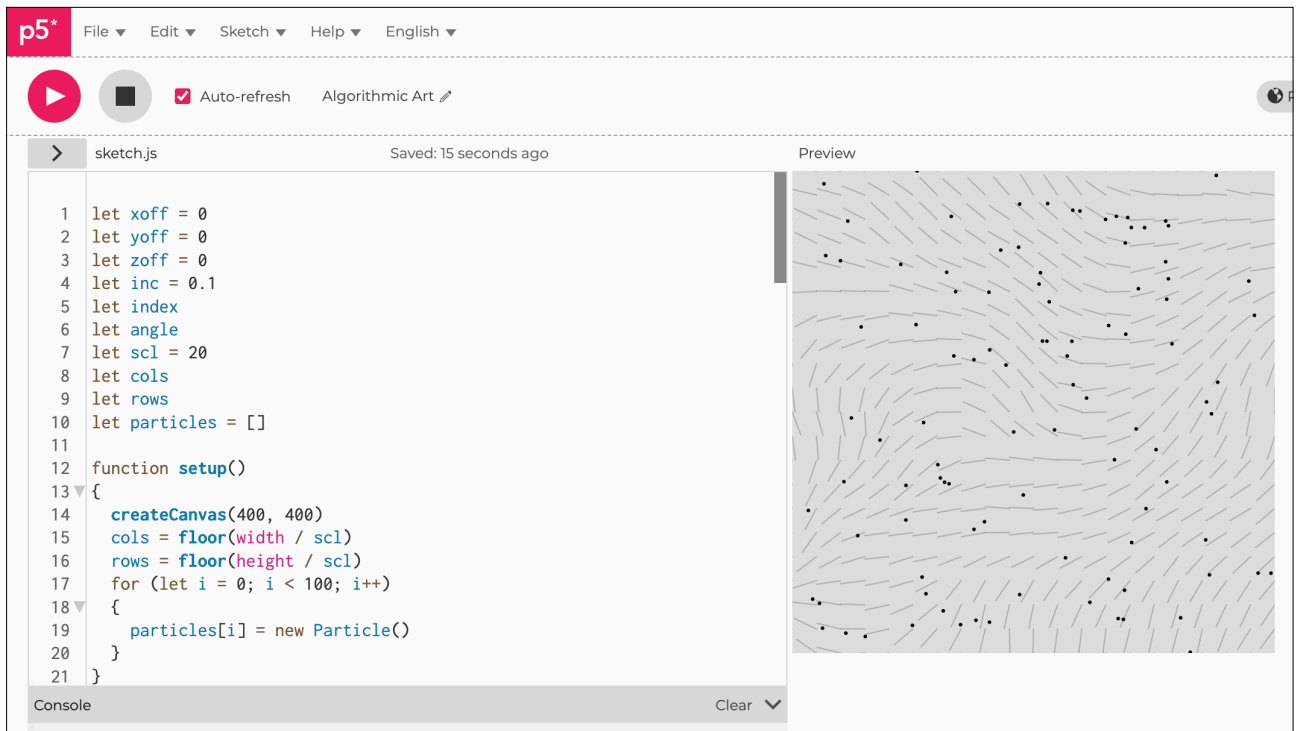
applyForce(force)
{
    this.acc.add(force)
}

edges()
{
    if (this.pos.x > width)
    {
        this.pos.x = 0
    }
    if (this.pos.x < 0)
    {
        this.pos.x = width
    }
    if (this.pos.y > height)
    {
        this.pos.y = 0
    }
    if (this.pos.y < 0)
    {
        this.pos.y = height
    }
}
}
```

Notes

The points aren't moving according to `perlin noise()`.

Figure E5.19





Sketch E5.20 array of vectors

We want to store the values of the vectors in an array and then attribute them to the particles nearest to them. We create an empty array called `flowfield[]`. We use the formula for calculating the index and change the width to `cols` (and remove the times 4). Now we can fill the array with those vectors `v`.

```
let xoff = 0
let yoff = 0
let zoff = 0
let inc = 0.1
let index
let angle
let scl = 20
let cols
let rows
let particles = []
let flowfield = []

function setup()
{
  createCanvas(400, 400)
  cols = floor(width / scl)
  rows = floor(height / scl)
  flowfield = new Array(cols * rows)
  for (let i = 0; i < 100; i++)
  {
    particles[i] = new Particle()
  }
}

function draw()
{
  background(220)
  yoff = 0
  for (y = 0; y < rows + 1; y++)
  {
    xoff = 0
    for (x = 0; x < cols + 1; x++)
```

```

{
  index = (x + y * cols)
  angle = noise(xoff, yoff, zoff) * (2 * PI)
  xoff += inc
  let v = p5.Vector.fromAngle(angle)
  flowfield[index] = v

  push()
  translate(x * scl, y * scl)
  rotate(v.heading())
  strokeWeight(1)
  stroke(0, 50)
  line(0, 0, scl, 0)
  pop()
}
yoff += inc
zoff += 0.0005
}
for (let i = 0; i < particles.length; i++)
{
  particles[i].show()
  particles[i].move()
  particles[i].edges()
}
}

class Particle
{
  constructor()
  {
    this.pos = createVector(random(width), random(height))
    this.vel = p5.Vector.random2D()
    this.acc = createVector(0, 0)
  }

  show()
  {
    strokeWeight(3)
    point(this.pos.x, this.pos.y)
  }
}

```

```
}

move()
{
  this.vel.add(this.acc)
  this.pos.add(this.vel)
  this.acc.mult(0)
}

applyForce(force)
{
  this.acc.add(force)
}

edges()
{
  if (this.pos.x > width)
  {
    this.pos.x = 0
  }
  if (this.pos.x < 0)
  {
    this.pos.x = width
  }
  if (this.pos.y > height)
  {
    this.pos.y = 0
  }
  if (this.pos.y < 0)
  {
    this.pos.y = height
  }
}
}
```

Notes

Nothing new happening yet.



Sketch E5.21 following the vectors

We need to write a function called `follow()` in the particle class that helps us identify which particle is near which vector. It will have an argument called `vectors`. We have to find the co-ordinates of the particle according to how the vector lines are laid out.

```
let xoff = 0
let yoff = 0
let zoff = 0
let inc = 0.1
let index
let angle
let scl = 20
let cols
let rows
let particles = []
let flowfield = []

function setup()
{
  createCanvas(400, 400)
  cols = floor(width / scl)
  rows = floor(height / scl)
  flowfield = new Array(cols * rows)
  for (let i = 0; i < 100; i++)
  {
    particles[i] = new Particle()
  }
}

function draw()
{
  background(220)
  yoff = 0
  for (y = 0; y < rows + 1; y++)
  {
    xoff = 0
    for (x = 0; x < cols + 1; x++)
```

```

{
  index = (x + y * cols)
  angle = noise(xoff, yoff, zoff) * (2 * PI)
  xoff += inc
  let v = p5.Vector.fromAngle(angle)
  flowfield[index] = v
  push()
  translate(x * scl, y * scl)
  rotate(v.heading())
  strokeWeight(1)
  stroke(0, 50)
  line(0, 0, scl, 0)
  pop()
}
yoff += inc
zoff += 0.0005
}
for (let i = 0; i < particles.length; i++)
{
  particles[i].show()
  particles[i].move()
  particles[i].edges()
}
}

class Particle
{
  constructor()
  {
    this.pos = createVector(random(width), random(height))
    this.vel = p5.Vector.random2D()
    this.acc = createVector(0, 0)
  }

  show()
  {
    strokeWeight(3)
    point(this.pos.x, this.pos.y)
  }
}

```

```

}

move()
{
  this.vel.add(this.acc)
  this.pos.add(this.vel)
  this.acc.mult(0)
}

applyForce(force)
{
  this.acc.add(force)
}

edges()
{
  if (this.pos.x > width)
  {
    this.pos.x = 0
  }
  if (this.pos.x < 0)
  {
    this.pos.x = width
  }
  if (this.pos.y > height)
  {
    this.pos.y = 0
  }
  if (this.pos.y < 0)
  {
    this.pos.y = height
  }
}

```

```

follow(vectors)
{
  let x = floor(this.pos.x/scl)
  let y = floor(this.pos.y/scl)

```

```
}
```

```
}
```

Notes

Building blocks at present.



Sketch E5.22 the index value

From that, we calculate the index value from **x** and **y** using the formula.

```
let xoff = 0
let yoff = 0
let zoff = 0
let inc = 0.1
let index
let angle
let scl = 20
let cols
let rows
let particles = []
let flowfield = []

function setup()
{
  createCanvas(400, 400)
  cols = floor(width / scl)
  rows = floor(height / scl)
  flowfield = new Array(cols * rows)
  for (let i = 0; i < 100; i++)
  {
    particles[i] = new Particle()
  }
}

function draw()
{
  background(220)
  yoff = 0
  for (y = 0; y < rows + 1; y++)
  {
    xoff = 0
    for (x = 0; x < cols + 1; x++)
    {
      index = (x + y * cols)
      angle = noise(xoff, yoff, zoff) * (2 * PI)
```

```

    xoff += inc
    let v = p5.Vector.fromAngle(angle)
    flowfield[index] = v
    push()
    translate(x * scl, y * scl)
    rotate(v.heading())
    strokeWeight(1)
    stroke(0, 50)
    line(0, 0, scl, 0)
    pop()
  }
  yoff += inc
  zoff += 0.0005
}
for (let i = 0; i < particles.length; i++)
{
  particles[i].show()
  particles[i].move()
  particles[i].edges()
}
}

class Particle
{
  constructor()
  {
    this.pos = createVector(random(width), random(height))
    this.vel = p5.Vector.random2D()
    this.acc = createVector(0, 0)
  }

  show()
  {
    strokeWeight(3)
    point(this.pos.x, this.pos.y)
  }

  move()

```

```

{
  this.vel.add(this.acc)
  this.pos.add(this.vel)
  this.acc.mult(0)
}

applyForce(force)
{
  this.acc.add(force)
}

edges()
{
  if (this.pos.x > width)
  {
    this.pos.x = 0
  }
  if (this.pos.x < 0)
  {
    this.pos.x = width
  }
  if (this.pos.y > height)
  {
    this.pos.y = 0
  }
  if (this.pos.y < 0)
  {
    this.pos.y = height
  }
}

follow(vectors)
{
  let x = floor(this.pos.x/scl)
  let y = floor(this.pos.y/scl)
  let index = x + (y * cols)
}
}

```

Notes

If you accidentally run this with the latest version (2.3.4 in my case) it goes a bit weird. So make sure it is 2.2.3 for the moment although it may well be worth trying at some point in the future.



Sketch E5.23 index force

We calculate the **force** of the vector at that index.

```
let xoff = 0
let yoff = 0
let zoff = 0
let inc = 0.1
let index
let angle
let scl = 20
let cols
let rows
let particles = []
let flowfield = []

function setup()
{
  createCanvas(400, 400)
  cols = floor(width / scl)
  rows = floor(height / scl)
  flowfield = new Array(cols * rows)
  for (let i = 0; i < 100; i++)
  {
    particles[i] = new Particle()
  }
}

function draw()
{
  background(220)
  yoff = 0
  for (y = 0; y < rows + 1; y++)
  {
    xoff = 0
    for (x = 0; x < cols + 1; x++)
    {
      index = (x + y * cols)
      angle = noise(xoff, yoff, zoff) * (2 * PI)
```

```

    xoff += inc
    let v = p5.Vector.fromAngle(angle)
    flowfield[index] = v
    push()
    translate(x * scl, y * scl)
    rotate(v.heading())
    strokeWeight(1)
    stroke(0, 50)
    line(0, 0, scl, 0)
    pop()
  }
  yoff += inc
  zoff += 0.0005
}
for (let i = 0; i < particles.length; i++)
{
  particles[i].show()
  particles[i].move()
  particles[i].edges()
}
}

class Particle
{
  constructor()
  {
    this.pos = createVector(random(width), random(height))
    this.vel = p5.Vector.random2D()
    this.acc = createVector(0, 0)
  }

  show()
  {
    strokeWeight(3)
    point(this.pos.x, this.pos.y)
  }

  move()

```

```

{
  this.vel.add(this.acc)
  this.pos.add(this.vel)
  this.acc.mult(0)
}

applyForce(force)
{
  this.acc.add(force)
}

edges()
{
  if (this.pos.x > width)
  {
    this.pos.x = 0
  }
  if (this.pos.x < 0)
  {
    this.pos.x = width
  }
  if (this.pos.y > height)
  {
    this.pos.y = 0
  }
  if (this.pos.y < 0)
  {
    this.pos.y = height
  }
}

follow(vectors)
{
  let x = floor(this.pos.x/scl)
  let y = floor(this.pos.y/scl)
  let index = x + (y * cols)
  let force = vectors[index]
}

```

}

Notes

Just doing the maths.



Sketch E5.24 may the force be with you

We then apply the **force**.

```
let xoff = 0
let yoff = 0
let zoff = 0
let inc = 0.1
let index
let angle
let scl = 20
let cols
let rows
let particles = []
let flowfield = []

function setup()
{
  createCanvas(400, 400)
  cols = floor(width / scl)
  rows = floor(height / scl)
  flowfield = new Array(cols * rows)
  for (let i = 0; i < 100; i++)
  {
    particles[i] = new Particle()
  }
}

function draw()
{
  background(220)
  yoff = 0
  for (y = 0; y < rows + 1; y++)
  {
    xoff = 0
    for (x = 0; x < cols + 1; x++)
    {
      index = (x + y * cols)
      angle = noise(xoff, yoff, zoff) * (2 * PI)
```

```

    xoff += inc
    let v = p5.Vector.fromAngle(angle)
    flowfield[index] = v
    push()
    translate(x * scl, y * scl)
    rotate(v.heading())
    strokeWeight(1)
    stroke(0, 50)
    line(0, 0, scl, 0)
    pop()
  }
  yoff += inc
  zoff += 0.0005
}
for (let i = 0; i < particles.length; i++)
{
  particles[i].show()
  particles[i].move()
  particles[i].edges()
}
}

class Particle
{
  constructor()
  {
    this.pos = createVector(random(width), random(height))
    this.vel = p5.Vector.random2D()
    this.acc = createVector(0, 0)
  }

  show()
  {
    strokeWeight(3)
    point(this.pos.x, this.pos.y)
  }

  move()

```

```

{
  this.vel.add(this.acc)
  this.pos.add(this.vel)
  this.acc.mult(0)
}

applyForce(force)
{
  this.acc.add(force)
}

edges()
{
  if (this.pos.x > width)
  {
    this.pos.x = 0
  }
  if (this.pos.x < 0)
  {
    this.pos.x = width
  }
  if (this.pos.y > height)
  {
    this.pos.y = 0
  }
  if (this.pos.y < 0)
  {
    this.pos.y = height
  }
}

follow(vectors)
{
  let x = floor(this.pos.x/scl)
  let y = floor(this.pos.y/scl)
  let index = x + (y * cols)
  let force = vectors[index]
  this.applyForce(force)
}

```

}

}

Notes

Applying the maths.



Sketch E5.25 mad dash

Let's make the **velocity** zero in the Particle class, then set a maximum speed **this.maxSpeed = 4**, and then limit the speed in the **move()** function.

```
let xoff = 0
let yoff = 0
let zoff = 0
let inc = 0.1
let index
let angle
let scl = 20
let cols
let rows
let particles = []
let flowfield = []

function setup()
{
  createCanvas(400, 400)
  cols = floor(width / scl)
  rows = floor(height / scl)
  flowfield = new Array(cols * rows)
  for (let i = 0; i < 100; i++)
  {
    particles[i] = new Particle()
  }
}

function draw()
{
  background(220)
  yoff = 0
  for (y = 0; y < rows + 1; y++)
  {
    xoff = 0
    for (x = 0; x < cols + 1; x++)
    {
      index = (x + y * cols)
```

```

    angle = noise(xoff, yoff, zoff) * (2 * PI)
    xoff += inc
    let v = p5.Vector.fromAngle(angle)
    flowfield[index] = v
    push()
    translate(x * scl, y * scl)
    rotate(v.heading())
    strokeWeight(1)
    stroke(0, 50)
    line(0, 0, scl, 0)
    pop()
  }
  yoff += inc
  zoff += 0.0005
}
for (let i = 0; i < particles.length; i++)
{
  particles[i].show()
  particles[i].move()
  particles[i].edges()
}
}

class Particle
{
  constructor()
  {
    this.pos = createVector(random(width), random(height))
    this.vel = createVector(0, 0)
    this.acc = createVector(0, 0)
    this.maxSpeed = 4
  }

  show()
  {
    strokeWeight(3)
    point(this.pos.x, this.pos.y)
  }
}

```

```
move()
{
  this.vel.add(this.acc)
  this.vel.limit(this.maxSpeed)
  this.pos.add(this.vel)
  this.acc.mult(0)
}

applyForce(force)
{
  this.acc.add(force)
}

edges()
{
  if (this.pos.x > width)
  {
    this.pos.x = 0
  }
  if (this.pos.x < 0)
  {
    this.pos.x = width
  }
  if (this.pos.y > height)
  {
    this.pos.y = 0
  }
  if (this.pos.y < 0)
  {
    this.pos.y = height
  }
}

follow(vectors)
{
  let x = floor(this.pos.x/scl)
  let y = floor(this.pos.y/scl)
```

```
    let index = x + (y * cols)
    let force = vectors[index]
    this.applyForce(force)
  }
}
```

Notes

We have come to a complete standstill. Be patient, just wait.



Sketch E5.26 the magnitude of it all

In the main sketch, we can set the magnitude of the force. You will notice that it goes a bit mad if you run it, but it is the beginning of something.

! Remove `flowfield = new Array()` (highlighted).

```
let xoff = 0
let yoff = 0
let zoff = 0
let inc = 0.1
let index
let angle
let scl = 20
let cols
let rows
let particles = []
let flowfield = []

function setup()
{
  createCanvas(400, 400)
  cols = floor(width / scl)
  rows = floor(height / scl)
  // flowfield = new Array(cols * rows)
  for (let i = 0; i < 100; i++)
  {
    particles[i] = new Particle()
  }
}

function draw()
{
  background(220)
  yoff = 0
  for (y = 0; y < rows + 1; y++)
  {
    xoff = 0
    for (x = 0; x < cols + 1; x++)
```

```

{
  index = (x + y * cols)
  angle = noise(xoff, yoff, zoff) * (2 * PI)
  xoff += inc
  let v = p5.Vector.fromAngle(angle)
  v.setMag(0.1)
  flowfield[index] = v
  push()
  translate(x * scl, y * scl)
  rotate(v.heading())
  strokeWeight(1)
  stroke(0, 50)
  line(0, 0, scl, 0)
  pop()
}
yoff += inc
zoff += 0.0005
}
for (let i = 0; i < particles.length; i++)
{
  particles[i].follow(flowfield)
  particles[i].show()
  particles[i].move()
  particles[i].edges()
}
}

class Particle
{
  constructor()
  {
    this.pos = createVector(random(width), random(height))
    this.vel = createVector(0, 0)
    this.acc = createVector(0, 0)
    this.maxSpeed = 4
  }

  show()

```

```
{
  strokeWeight(3)
  point(this.pos.x, this.pos.y)
}

move()
{
  this.vel.add(this.acc)
  this.vel.limit(this.maxSpeed)
  this.pos.add(this.vel)
  this.acc.mult(0)
}

applyForce(force)
{
  this.acc.add(force)
}

edges()
{
  if (this.pos.x > width)
  {
    this.pos.x = 0
  }
  if (this.pos.x < 0)
  {
    this.pos.x = width
  }
  if (this.pos.y > height)
  {
    this.pos.y = 0
  }
  if (this.pos.y < 0)
  {
    this.pos.y = height
  }
}
```

```

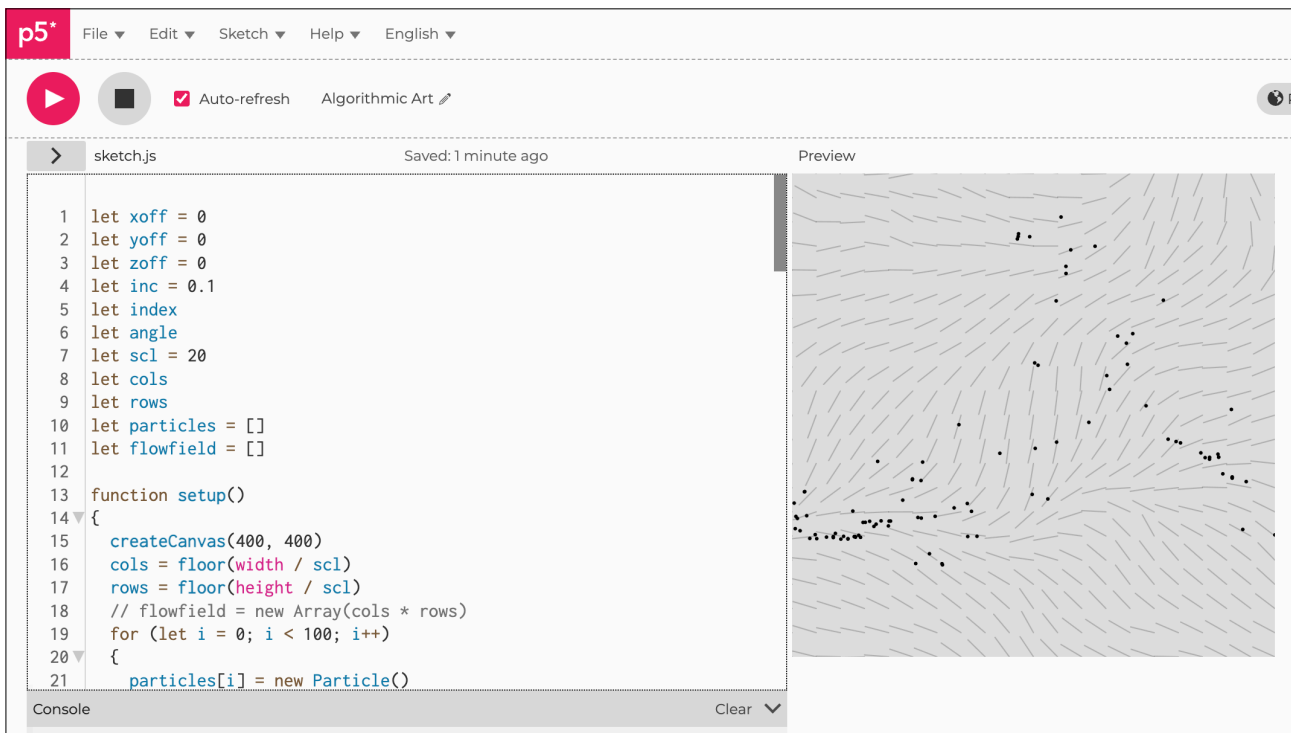
follow(vectors)
{
  let x = floor(this.pos.x/scl)
  let y = floor(this.pos.y/scl)
  let index = x + (y * cols)
  let force = vectors[index]
  this.applyForce(force)
}
}

```

Notes

You should see them start to follow the flowfield.

Figure E5.26





Sketch E5.27 zoff off

Now set the magnitude to **5** and turn off the **zoff** and you should see it very clearly.

```
let xoff = 0
let yoff = 0
let zoff = 0
let inc = 0.1
let index
let angle
let scl = 20
let cols
let rows
let particles = []
let flowfield = []

function setup()
{
  createCanvas(400, 400)
  cols = floor(width / scl)
  rows = floor(height / scl)
  for (let i = 0; i < 100; i++)
  {
    particles[i] = new Particle()
  }
}

function draw()
{
  background(220)
  yoff = 0
  for (y = 0; y < rows + 1; y++)
  {
    xoff = 0
    for (x = 0; x < cols + 1; x++)
    {
      index = (x + y * cols)
      angle = noise(xoff, yoff, zoff) * (2 * PI)
      xoff += inc
```

```

    let v = p5.Vector.fromAngle(angle)
    v.setMag(5)
    flowfield[index] = v
    push()
    translate(x * scl, y * scl)
    rotate(v.heading())
    strokeWeight(1)
    stroke(0, 50)
    line(0, 0, scl, 0)
    pop()
  }
  yoff += inc
  // zoff += 0.0005
}
for (let i = 0; i < particles.length; i++)
{
  particles[i].follow(flowfield)
  particles[i].show()
  particles[i].move()
  particles[i].edges()
}
}

class Particle
{
  constructor()
  {
    this.pos = createVector(random(width), random(height))
    this.vel = createVector(0, 0)
    this.acc = createVector(0, 0)
    this.maxSpeed = 4
  }

  show()
  {
    strokeWeight(3)
    point(this.pos.x, this.pos.y)
  }
}

```

```

move()
{
  this.vel.add(this.acc)
  this.vel.limit(this.maxSpeed)
  this.pos.add(this.vel)
  this.acc.mult(0)
}

applyForce(force)
{
  this.acc.add(force)
}

edges()
{
  if (this.pos.x > width)
  {
    this.pos.x = 0
  }
  if (this.pos.x < 0)
  {
    this.pos.x = width
  }
  if (this.pos.y > height)
  {
    this.pos.y = 0
  }
  if (this.pos.y < 0)
  {
    this.pos.y = height
  }
}

follow(vectors)
{
  let x = floor(this.pos.x/scl)
  let y = floor(this.pos.y/scl)

```

```

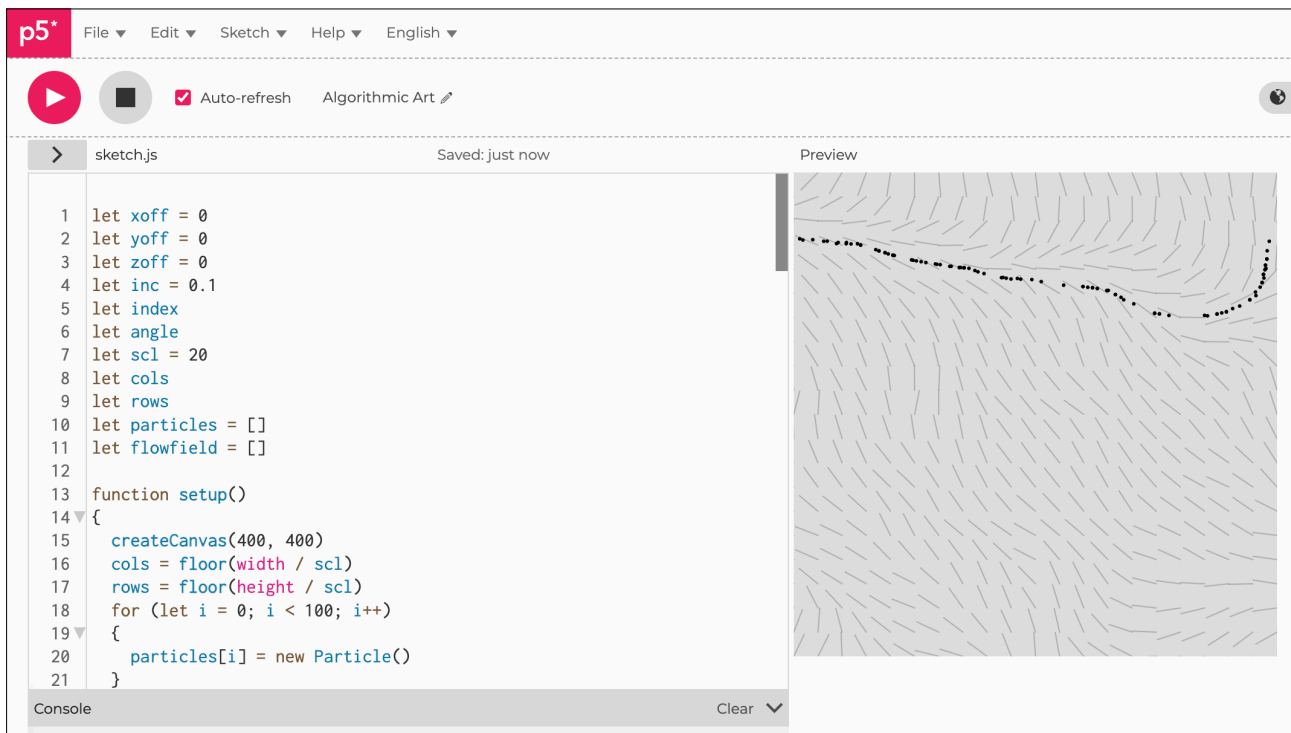
    let index = x + (y * cols)
    let force = vectors[index]
    this.applyForce(force)
  }
}

```

Notes

You can see it following an element of the flowfield.

Figure E5.27





Sketch E5.28 no lines

Removing the line vectors and adding more particles gives us a better feel, as well as reducing the **force** to **1** and also making the angle of rotation larger.

```
let xoff = 0
let yoff = 0
let zoff = 0
let inc = 0.1
let index
let angle
let scl = 20
let cols
let rows
let particles = []
let flowfield = []

function setup()
{
  createCanvas(400, 400)
  cols = floor(width / scl)
  rows = floor(height / scl)
  for (let i = 0; i < 200; i++)
  {
    particles[i] = new Particle()
  }
}

function draw()
{
  background(220)
  yoff = 0
  for (y = 0; y < rows + 1; y++)
  {
    xoff = 0
    for (x = 0; x < cols + 1; x++)
    {
      index = (x + y * cols)
      angle = noise(xoff, yoff, zoff) * (4 * PI)
```

```

    xoff += inc
    let v = p5.Vector.fromAngle(angle)
    v.setMag(1)
    flowfield[index] = v

    // push()
    // translate(x * scl, y * scl)
    // rotate(v.heading())
    // strokeWeight(1)
    // stroke(0, 50)
    // line(0, 0, scl, 0)
    // pop()
  }
  yoff += inc
  zoff += 0.0003
}

for (let i = 0; i < particles.length; i++)
{
  particles[i].follow(flowfield)
  particles[i].show()
  particles[i].move()
  particles[i].edges()
}
}

class Particle
{
  constructor()
  {
    this.pos = createVector(random(width), random(height))
    this.vel = createVector(0, 0)
    this.acc = createVector(0, 0)
    this.maxSpeed = 4
  }

  show()
  {
    strokeWeight(3)
    point(this.pos.x, this.pos.y)
  }
}

```

```

}

move()
{
  this.vel.add(this.acc)
  this.vel.limit(this.maxSpeed)
  this.pos.add(this.vel)
  this.acc.mult(0)
}

applyForce(force)
{
  this.acc.add(force)
}

edges()
{
  if (this.pos.x > width)
  {
    this.pos.x = 0
  }
  if (this.pos.x < 0)
  {
    this.pos.x = width
  }
  if (this.pos.y > height)
  {
    this.pos.y = 0
  }
  if (this.pos.y < 0)
  {
    this.pos.y = height
  }
}

follow(vectors)
{
  let x = floor(this.pos.x/scl)

```

```

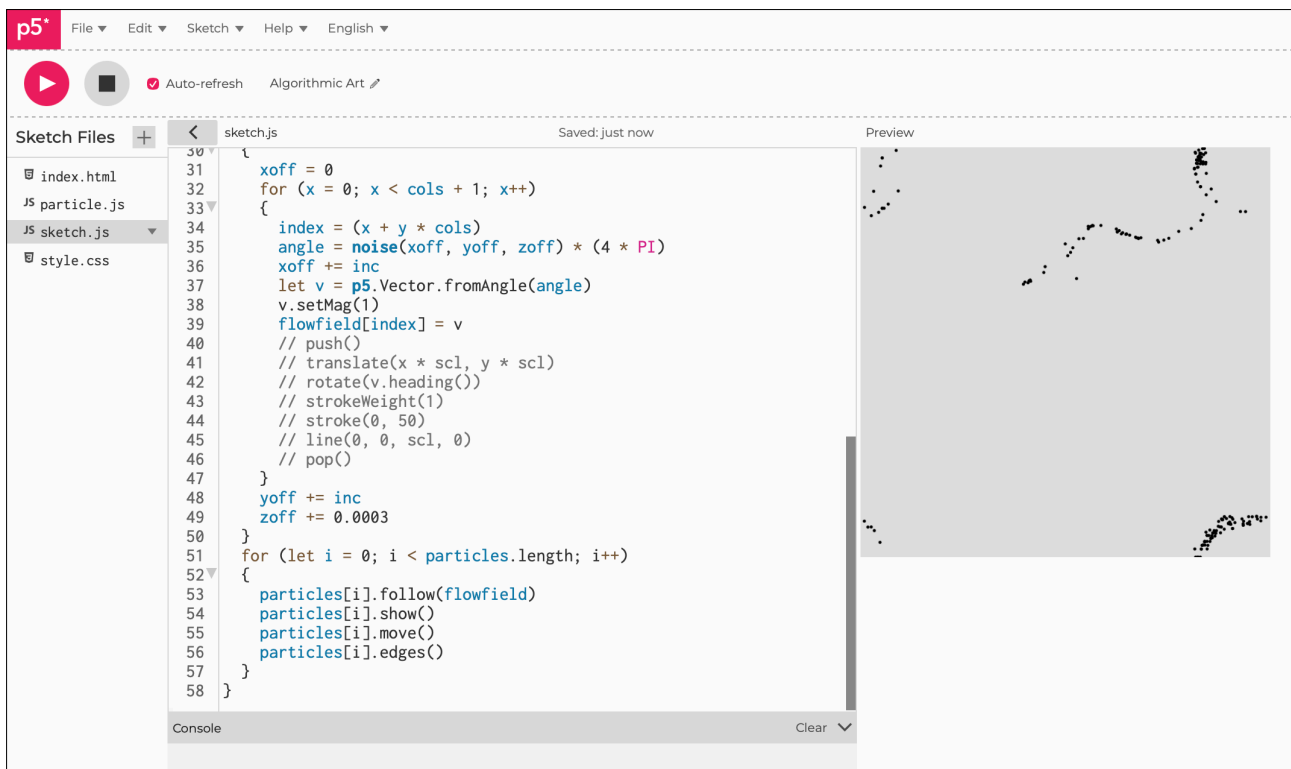
    let y = floor(this.pos.y/scl)
    let index = x + (y * cols)
    let force = vectors[index]
    this.applyForce(force)
  }
}

```

Notes

You do see some strange movement.

Figure E5.28





Sketch E5.29 nice visual

Making more amendments to get a nice visual. Move the `background()` into the `setup()` function so that it draws it only once, and increase the number of particles to `500`.

```
let xoff = 0
let yoff = 0
let zoff = 0
let inc = 0.1
let index
let angle
let scl = 20
let cols
let rows
let particles = []
let flowfield = []

function setup()
{
  createCanvas(400, 400)
  cols = floor(width / scl)
  rows = floor(height / scl)
  for (let i = 0; i < 500; i++)
  {
    particles[i] = new Particle()
  }
  background(220)
}

function draw()
{
  yoff = 0
  for (y = 0; y < rows + 1; y++)
  {
    xoff = 0
    for (x = 0; x < cols + 1; x++)
    {
      index = (x + y * cols)
```

```

    angle = noise(xoff, yoff, zoff) * (4 * PI)
    xoff += inc
    let v = p5.Vector.fromAngle(angle)
    v.setMag(1)
    flowfield[index] = v
  }
  yoff += inc
  zoff += 0.0003
}
for (let i = 0; i < particles.length; i++)
{
  particles[i].follow(flowfield)
  particles[i].show()
  particles[i].move()
  particles[i].edges()
}
}

class Particle
{
  constructor()
  {
    this.pos = createVector(random(width), random(height))
    this.vel = createVector(0, 0)
    this.acc = createVector(0, 0)
    this.maxSpeed = 4
  }

  show()
  {
    strokeWeight(3)
    point(this.pos.x, this.pos.y)
  }

  move()
  {
    this.vel.add(this.acc)
    this.vel.limit(this.maxSpeed)
  }
}

```

```

    this.pos.add(this.vel)
    this.acc.mult(0)
  }

  applyForce(force)
  {
    this.acc.add(force)
  }

  edges()
  {
    if (this.pos.x > width)
    {
      this.pos.x = 0
    }
    if (this.pos.x < 0)
    {
      this.pos.x = width
    }
    if (this.pos.y > height)
    {
      this.pos.y = 0
    }
    if (this.pos.y < 0)
    {
      this.pos.y = height
    }
  }

  follow(vectors)
  {
    let x = floor(this.pos.x/scl)
    let y = floor(this.pos.y/scl)
    let index = x + (y * cols)
    let force = vectors[index]
    this.applyForce(force)
  }
}

```

Notes

Interesting, but we can do even better.

Figure E5.29





Sketch E5.30 alpha stroke

In the Particle class, we change the drawing of the point. Making it a `strokeWeight()` of 1 and a `stroke()` with some `alpha`.

```
let xoff = 0
let yoff = 0
let zoff = 0
let inc = 0.1
let index
let angle
let scl = 20
let cols
let rows
let particles = []
let flowfield = []

function setup()
{
  createCanvas(400, 400)
  cols = floor(width / scl)
  rows = floor(height / scl)
  for (let i = 0; i < 500; i++)
  {
    particles[i] = new Particle()
  }
  background(220)
}

function draw()
{
  yoff = 0
  for (y = 0; y < rows + 1; y++)
  {
    xoff = 0
    for (x = 0; x < cols + 1; x++)
    {
      index = (x + y * cols)
      angle = noise(xoff, yoff, zoff) * (4 * PI)
```

```

    xoff += inc
    let v = p5.Vector.fromAngle(angle)
    v.setMag(1)
    flowfield[index] = v
  }
  yoff += inc
  zoff += 0.0003
}
for (let i = 0; i < particles.length; i++)
{
  particles[i].follow(flowfield)
  particles[i].show()
  particles[i].move()
  particles[i].edges()
}
}

class Particle
{
  constructor()
  {
    this.pos = createVector(random(width), random(height))
    this.vel = createVector(0, 0)
    this.acc = createVector(0, 0)
    this.maxSpeed = 4
  }

  show()
  {
    strokeWeight(1)
    stroke(0, 5)
    point(this.pos.x, this.pos.y)
  }

  move()
  {
    this.vel.add(this.acc)
    this.vel.limit(this.maxSpeed)
  }
}

```

```

    this.pos.add(this.vel)
    this.acc.mult(0)
}

applyForce(force)
{
    this.acc.add(force)
}

edges()
{
    if (this.pos.x > width)
    {
        this.pos.x = 0
    }
    if (this.pos.x < 0)
    {
        this.pos.x = width
    }
    if (this.pos.y > height)
    {
        this.pos.y = 0
    }
    if (this.pos.y < 0)
    {
        this.pos.y = height
    }
}

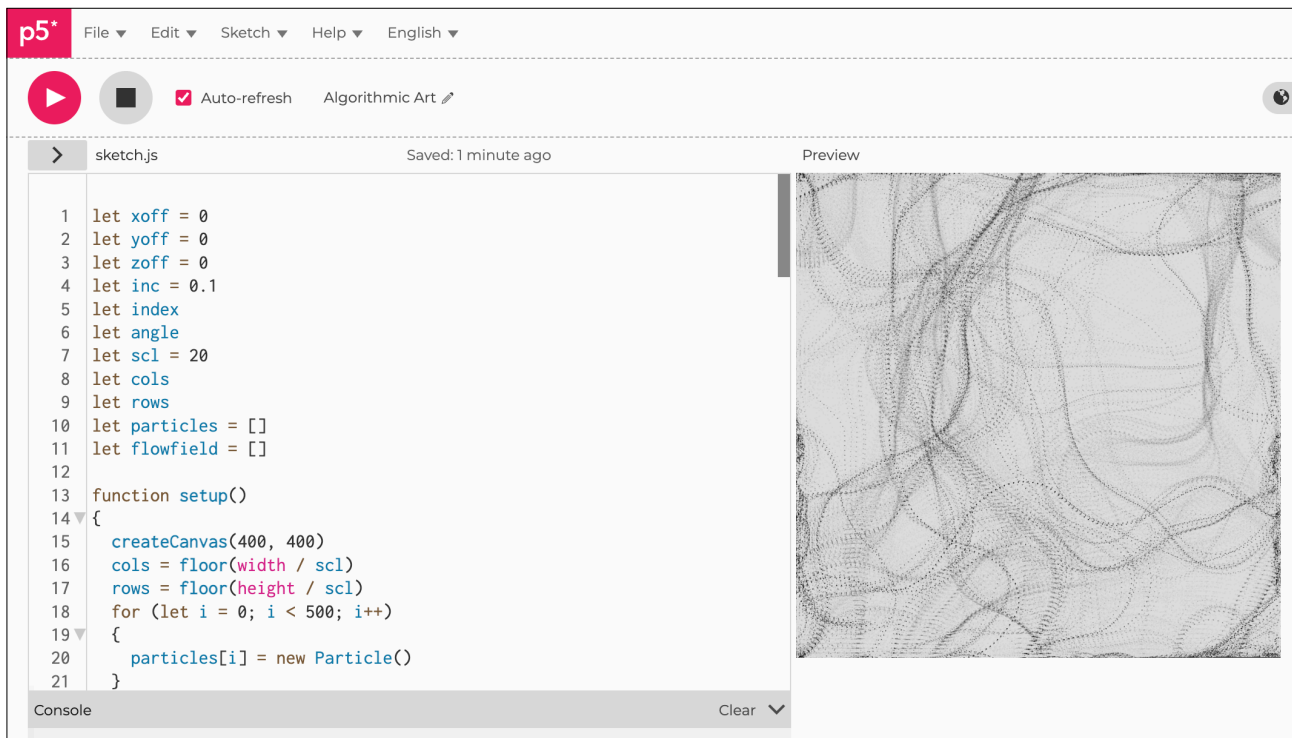
follow(vectors)
{
    let x = floor(this.pos.x/scl)
    let y = floor(this.pos.y/scl)
    let index = x + (y * cols)
    let force = vectors[index]
    this.applyForce(force)
}
}

```

Notes

You will notice that it is a bit pixelated because the point is moving faster than the frame rate. We will address this in the next part.

Figure E5.30





Sketch E5.31 the previous position

We copy the position of the pixel before it moves and call `this.prevPos` using the `copy()` function. Then, when the pixel moves, it draws a line between its position and its previous position.

```
let xoff = 0
let yoff = 0
let zoff = 0
let inc = 0.1
let index
let angle
let scl = 20
let cols
let rows
let particles = []
let flowfield = []

function setup()
{
  createCanvas(400, 400)
  cols = floor(width / scl)
  rows = floor(height / scl)
  for (let i = 0; i < 500; i++)
  {
    particles[i] = new Particle()
  }
  background(220)
}

function draw()
{
  yoff = 0
  for (y = 0; y < rows + 1; y++)
  {
    xoff = 0
    for (x = 0; x < cols + 1; x++)
    {
      index = (x + y * cols)
```

```

    angle = noise(xoff, yoff, zoff) * (4 * PI)
    xoff += inc
    let v = p5.Vector.fromAngle(angle)
    v.setMag(1)
    flowfield[index] = v
  }
  yoff += inc
  zoff += 0.0003
}
for (let i = 0; i < particles.length; i++)
{
  particles[i].follow(flowfield)
  particles[i].show()
  particles[i].move()
  particles[i].edges()
}
}

class Particle
{
  constructor()
  {
    this.pos = createVector(random(width), random(height))
    this.vel = createVector(0, 0)
    this.acc = createVector(0, 0)
    this.maxSpeed = 4
    this.prevPos = this.pos.copy()
  }

  show()
  {
    strokeWeight(1)
    stroke(0, 5)
    line(this.pos.x, this.pos.y, this.prevPos.x, this.prevPos.y)
    // point(this.pos.x, this.pos.y)
  }

  move()

```

```

{
  this.vel.add(this.acc)
  this.vel.limit(this.maxSpeed)
  this.pos.add(this.vel)
  this.acc.mult(0)
}

applyForce(force)
{
  this.acc.add(force)
}

edges()
{
  if (this.pos.x > width)
  {
    this.pos.x = 0
  }
  if (this.pos.x < 0)
  {
    this.pos.x = width
  }
  if (this.pos.y > height)
  {
    this.pos.y = 0
  }
  if (this.pos.y < 0)
  {
    this.pos.y = height
  }
}

follow(vectors)
{
  let x = floor(this.pos.x/scl)
  let y = floor(this.pos.y/scl)
  let index = x + (y * cols)
  let force = vectors[index]
}

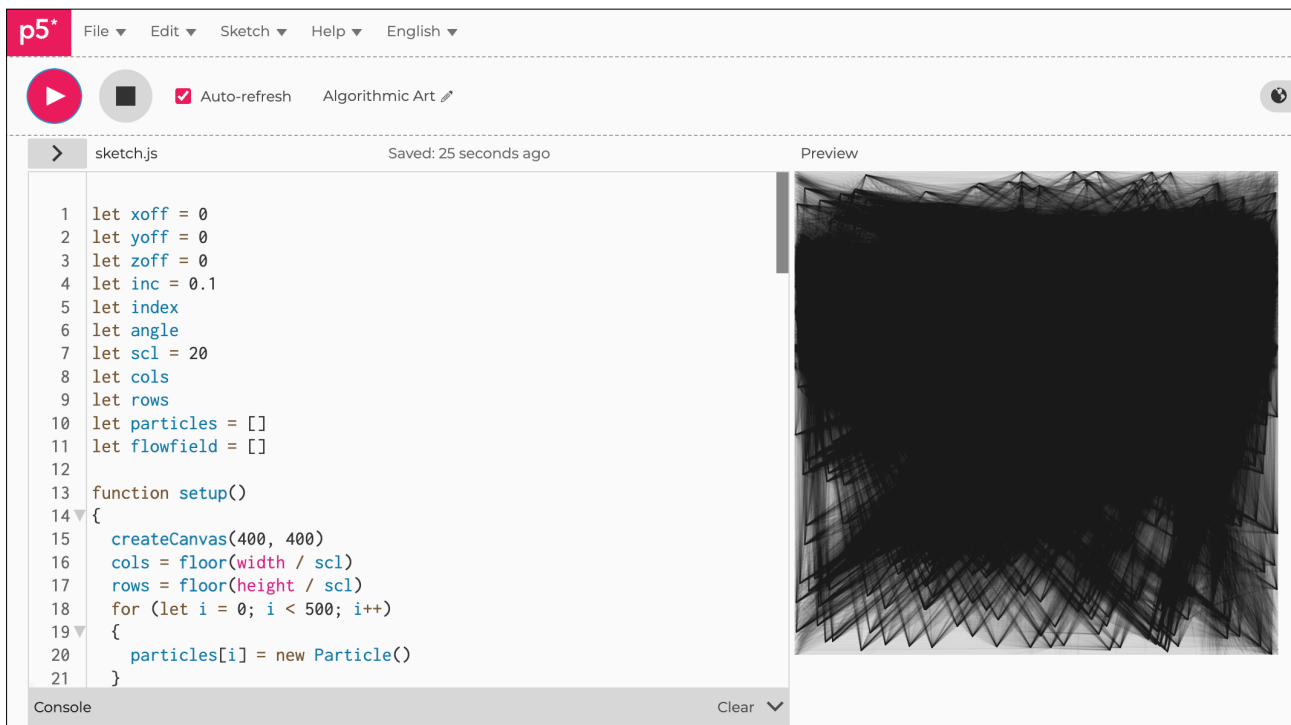
```

```
this.applyForce(force)
}
}
```

Notes

Hmmm, very quickly we have a problem because of the edges. However, we can rectify this.

Figure E5.31





Sketch E5.32 updating everything

We create another function to update the previous position.

```
let xoff = 0
let yoff = 0
let zoff = 0
let inc = 0.1
let index
let angle
let scl = 20
let cols
let rows
let particles = []
let flowfield = []

function setup()
{
  createCanvas(400, 400)
  cols = floor(width / scl)
  rows = floor(height / scl)
  for (let i = 0; i < 500; i++)
  {
    particles[i] = new Particle()
  }
  background(220)
}

function draw()
{
  yoff = 0
  for (y = 0; y < rows + 1; y++)
  {
    xoff = 0
    for (x = 0; x < cols + 1; x++)
    {
      index = (x + y * cols)
      angle = noise(xoff, yoff, zoff) * (4 * PI)
      xoff += inc
```

```

    let v = p5.Vector.fromAngle(angle)
    v.setMag(1)
    flowfield[index] = v
  }
  yoff += inc
  zoff += 0.0003
}
for (let i = 0; i < particles.length; i++)
{
  particles[i].follow(flowfield)
  particles[i].show()
  particles[i].move()
  particles[i].edges()
}
}

class Particle
{
  constructor()
  {
    this.pos = createVector(random(width), random(height))
    this.vel = createVector(0, 0)
    this.acc = createVector(0, 0)
    this.maxSpeed = 4
    this.prevPos = this.pos.copy()
  }

  show()
  {
    strokeWeight(1)
    stroke(0, 5)
    line(this.pos.x, this.pos.y, this.prevPos.x, this.prevPos.y)
    this.updatePrev()
  }

  move()
  {
    this.vel.add(this.acc)

```

```
    this.vel.limit(this.maxSpeed)
    this.pos.add(this.vel)
    this.acc.mult(0)
}
```

```
applyForce(force)
{
    this.acc.add(force)
}
```

```
updatePrev()
{
    this.prevPos.x = this.pos.x
    this.prevPos.y = this.pos.y
}
```

```
edges()
{
    if (this.pos.x > width)
    {
        this.pos.x = 0
```

```
        this.updatePrev()
```

```
    }
```

```
    if (this.pos.x < 0)
```

```
    {
```

```
        this.pos.x = width
```

```
        this.updatePrev()
```

```
    }
```

```
    if (this.pos.y > height)
```

```
    {
```

```
        this.pos.y = 0
```

```
        this.updatePrev()
```

```
    }
```

```
    if (this.pos.y < 0)
```

```
    {
```

```
        this.pos.y = height
```

```
        this.updatePrev()
```

```
    }
```

```

}

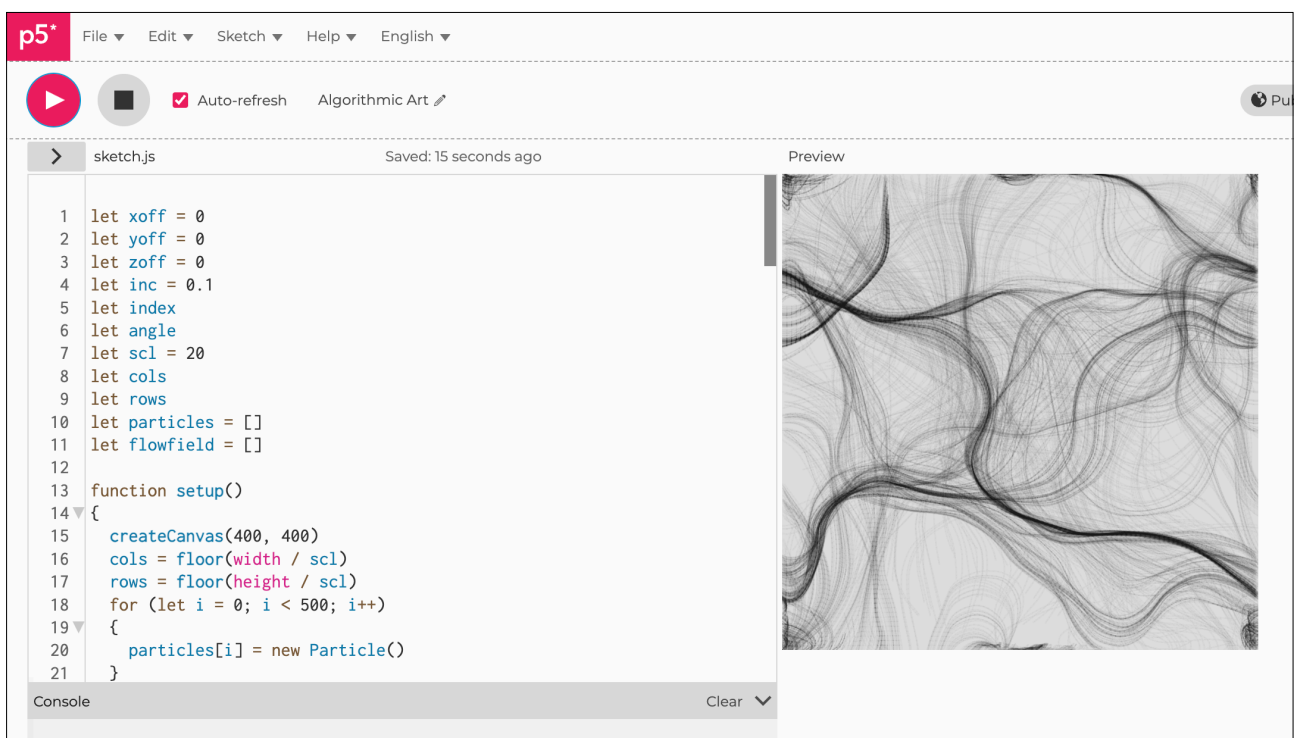
follow(vectors)
{
  let x = floor(this.pos.x/scl)
  let y = floor(this.pos.y/scl)
  let index = x + (y * cols)
  let force = vectors[index]
  this.applyForce(force)
}
}

```

Notes

A much improved response.

Figure E5.32





Sketch E5.33 edges before show

Move the `edges()` before `show()` so they are calculated first before drawing the particles/lines. Also removed the `pixelDensity()` and the drawing of lines (all commented out anyway `//`).

```
let xoff = 0
let yoff = 0
let zoff = 0
let inc = 0.1
let index
let angle
let scl = 20
let cols
let rows
let particles = []
let flowfield = []

function setup()
{
  createCanvas(400, 400)
  cols = floor(width / scl)
  rows = floor(height / scl)
  for (let i = 0; i < 300; i++)
  {
    particles[i] = new Particle()
  }
  background(220)
}

function draw()
{
  yoff = 0
  for (y = 0; y < rows; y++)
  {
    xoff = 0
    for (x = 0; x < cols; x++)
    {
      index = (x + y * cols)
```

```

    angle = noise(xoff, yoff, zoff) * (4 * PI)
    xoff += inc
    let v = p5.Vector.fromAngle(angle)
    v.setMag(1)
    flowfield[index] = v
  }
  yoff += inc
  zoff += 0.0003
}
for (let i = 0; i < particles.length; i++)
{
  particles[i].follow(flowfield)
  particles[i].edges()
  particles[i].show()
  particles[i].move()
}
}

class Particle
{
  constructor()
  {
    this.pos = createVector(random(width), random(height))
    this.vel = createVector(0, 0)
    this.acc = createVector(0, 0)
    this.maxSpeed = 4
    this.prevPos = this.pos.copy()
  }

  show()
  {
    strokeWeight(1)
    stroke(0, 5)
    line(this.pos.x, this.pos.y, this.prevPos.x, this.prevPos.y)
    this.updatePrev()
  }

  move()

```

```

{
  this.vel.add(this.acc)
  this.vel.limit(this.maxSpeed)
  this.pos.add(this.vel)
  this.acc.mult(0)
}

applyForce(force)
{
  this.acc.add(force)
}

updatePrev()
{
  this.prevPos.x = this.pos.x
  this.prevPos.y = this.pos.y
}

edges()
{
  if (this.pos.x > width)
  {
    this.pos.x = 0
    this.updatePrev()
  }
  if (this.pos.x < 0)
  {
    this.pos.x = width
    this.updatePrev()
  }
  if (this.pos.y > height)
  {
    this.pos.y = 0
    this.updatePrev()
  }
  if (this.pos.y < 0)
  {
    this.pos.y = height

```

```

    this.updatePrev()
  }
}

follow(vectors)
{
  let x = floor(this.pos.x/scl)
  let y = floor(this.pos.y/scl)
  let index = x + (y * cols)
  let force = vectors[index]
  this.applyForce(force)
}
}

```

Notes

Save all the code for the next unit where you will put the Particle class in another file.

Figure E5.33



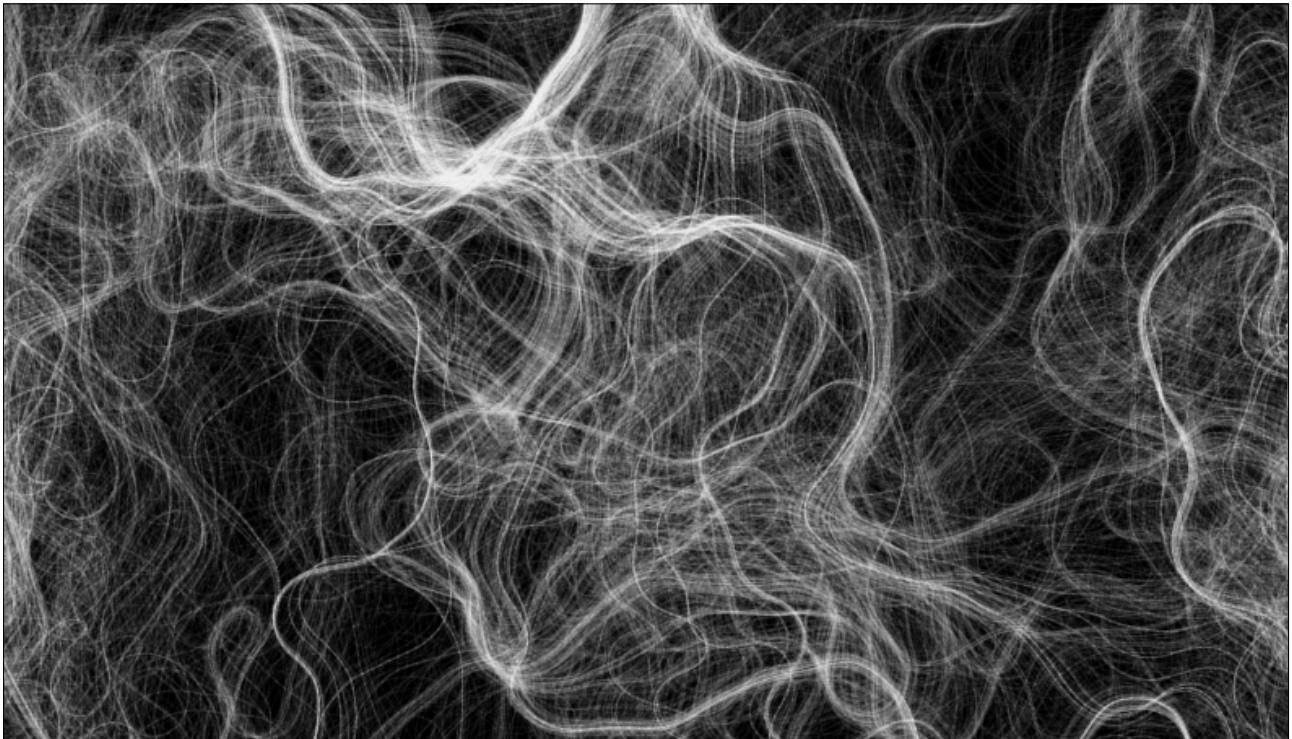


Challenges

To improve the effect, I suggest altering the following:

1. Change the max force in `setMag()` value.
2. Alter `xoff`.
3. Change the amount of `alpha`.
4. Change the `strokeWeight()` to `0.5`.
5. Reduce the `maxSpeed`.
6. Change the number of particles.
7. Background to black and the lines to white/grey.
8. Introduce coloured lines.

All the above are tweaks to the parameters; an example below of what I mean.





Algorithmic Art Module E Unit #6 File Management



Module E Unit #6: file management

Before you skip this and move on, this is an important topic covering much of what you may do next, also it is how everyone else manages their files. You can add files, add images, videos, fonts, you can add libraries such as ml5.js (an AI library) and others.

The following unit is not covering any new code as such but is designed to introduce you to what is going on in the background before you need to know and use what is going on in the background and it is very easy but you just need to practice.

The first task will be to find the files, then we will look at the index.html file. This is at the heart of what you see but needs little (if any) attention 90% of the time. It is very useful to have an understanding of where it is and what it is and what it is for.

Then we can look at creating new files and how they can be really useful. Another useful feature is uploading images that we can incorporate into our sketches. This is also how we upload other types of files such as videos, fonts etc. The finally how to add libraries that can help us to do some really interesting stuff.

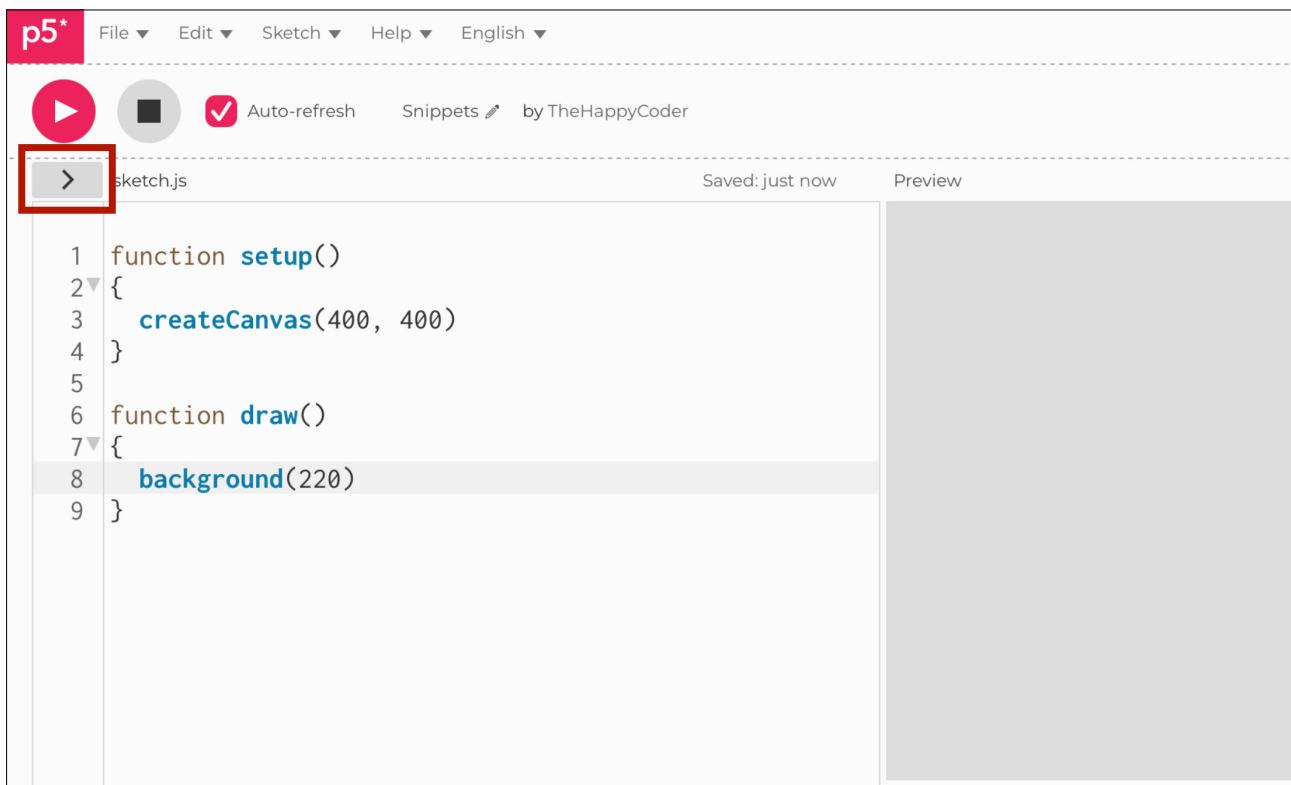


Accessing the index.html file

This is the code within the `index.html` file. The index file is the core of the editor because it is where the editor interacts with the web browser. HTML is a tag based coding language and is often used to create web sites. All this is in the web editor by default.

This is how we can access the index.html file: click on the grey box with the arrow in it as highlighted in **Fig.1**.

Figure 1: click on the arrow

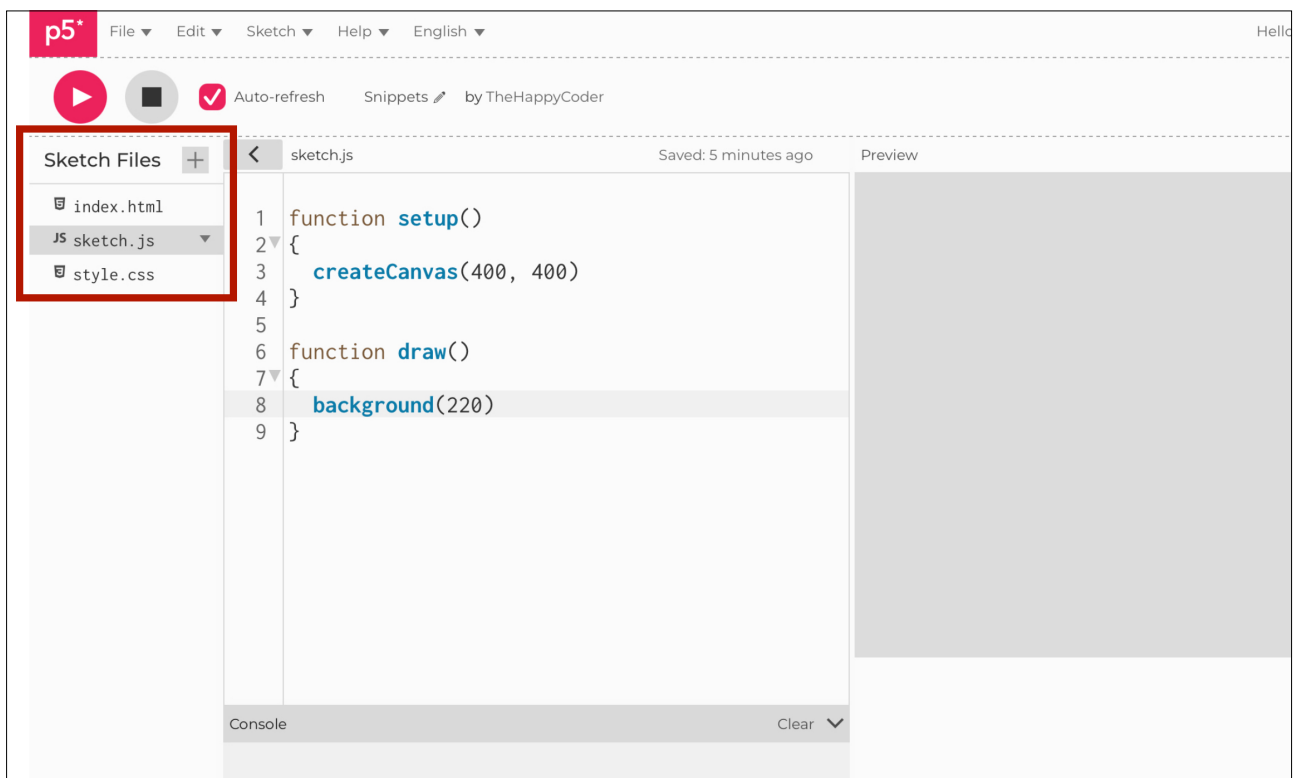




File List

This will give you a drop down menu of a number of files under the heading of **Sketch Files**. The main one you have been using is the **sketch.js**. We want the **index.html** file and so click on the **index.html** file as shown in **Fig.2**.

Figure 2: files menu





Inside the index.html File

This is what it currently looks like inside the index.html. Even as I write this workbook the latest version is 2.2.4, it is being constantly updated. I have stretched it out a bit so you can see the lines of code clearly. If you have never seen HTML code before it can look a little bit daunting, that is because it is unfamiliar.

Yet HTML code has been around for a very long time and is the backbone of websites. You will also notice that there is a CSS file that goes with it, but more on that later and you will be able to play with it in workbook #8.

Figure 3: HTML code

```
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4   <script src="https://cdn.jsdelivr.net/npm/p5@2.3.4/lib/p5.js"></script>
5   <script src="https://cdn.jsdelivr.net/npm/p5.sound@0.4.1/dist/p5.sound.min.js"></script>
6
7
8
9   <link rel="stylesheet" type="text/css" href="style.css">
10  <meta charset="utf-8" />
11
12 </head>
13 <body>
14   <main>
15   </main>
16   <script src="sketch.js"></script>
17 </body>
18 </html>
```



But what do they mean?

HTML uses tags, those are the `<` and `>` symbols. They denote the start of a set of instructions. The tags are usually in pairs, so `<html lang="en">` has to have closure with a forward slash `/` such as `</html>` at the end.

<code><!DOCTYPE html></code>	This tells the computer that it is a HTML document/code
<code><html lang="en"></code>	The language we are going to use in English
<code><head></code>	This means the next section contains metadata which just means data about data
<code><script src="https://cdn.jsdelivr.net/npm/p5@2.3.4/lib/p5.js"></script></code>	This is where we get our p5.js from. It is sent to the website we are using in our browser. This is why you don't need to download it. It also means you do need to have the internet. Notice that it is accessing the latest version 2.3.4.
<code><script src="https://cdn.jsdelivr.net/npm/p5.sound@0.4.1/dist/p5.sound.min.js"></script></code>	This is another aspect of p5.js when we want to generate or use sounds, more on that in a later workbook.
<code><link rel="stylesheet" type="text/css" href="style.css"></code>	Because the CSS file is separate the index file as to link to it.
<code><meta charset="utf-8" /></code>	This tag tells the web browser how to translate the text data of an HTML page into readable characters.
<code></head></code>	End tag
<code><body></code>	This contains all the contents of an HTML document, such as headings, paragraphs, images, hyperlinks, tables, lists, etc.
<code><main> </main></code>	Clearly this does nothing in this case but we can put paragraphs and headers in here, again see later on.
<code><script src="sketch.js"></script></code>	This refers to the sketch.js file you can see on the left-hand side list of files. If this was not tagged here your sketch would be invisible to it and therefore will not run your code.
<code></body></code>	End body tag.
<code></html></code>	End of HTML.

Challenge

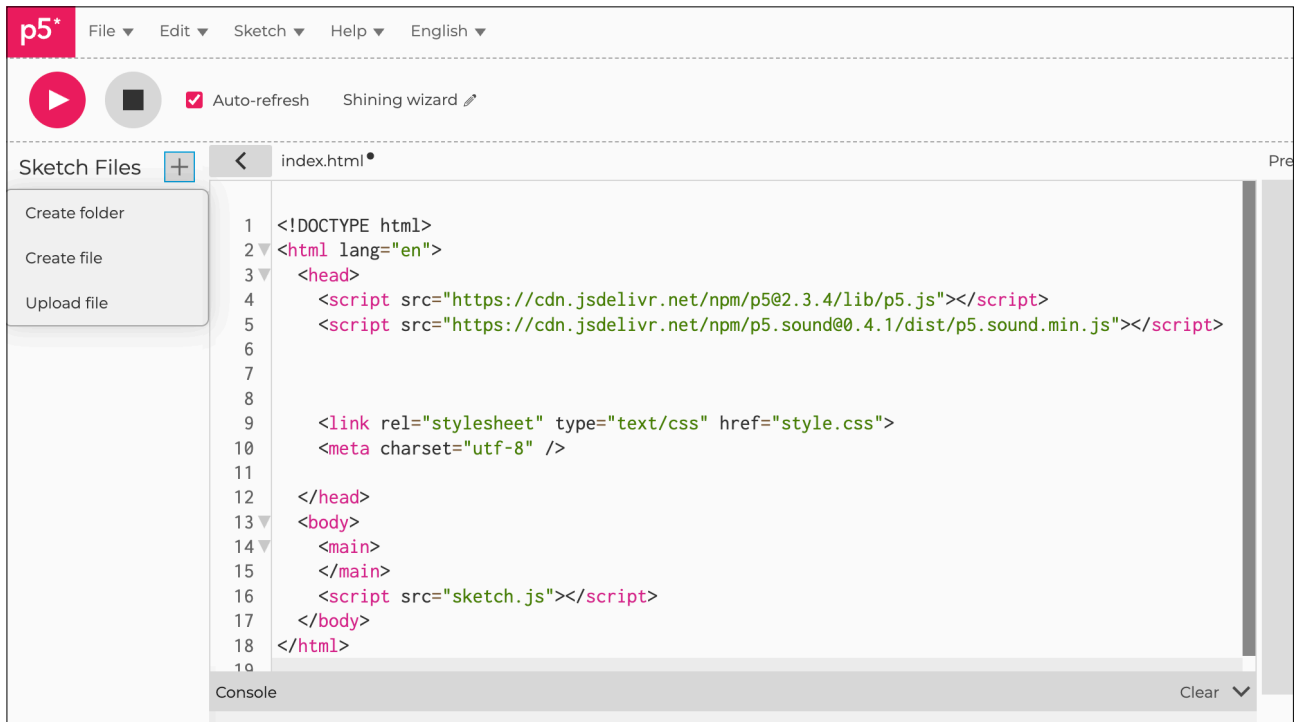
Click on `<script src="sketch.js"></script>` and then comment out (press **command** button and `/`) and now try to run your sketch.



Adding a file

If you click on the **+** symbol in the grey box next to **Sketch Files** you will get this drop down menu.

Figure 4: folder/file menu





Create a File

Click on Create File, we want to create a file to put the **Particle class**. A box will appear, type the full name of the file you want to create and this includes the format, in this case it is a JavaScript file so we want **particle.js**. Then press **Add File**. You will notice that it has been added to the list of files. The next step is important and easily forgotten.

Figure 5: naming the file





Referencing the file

Pop back to the index.html file and we now need to reference it there otherwise it will not know anything about the new file when we run the sketch. What I do is copy the **sketch.js**, paste it underneath and change the name.

Figure 6: referencing the file

The screenshot shows the p5.js IDE interface. The top menu bar includes 'File', 'Edit', 'Sketch', 'Help', and 'English'. Below the menu bar, there are icons for running (a play button) and stopping (a square button), along with a checkbox for 'Auto-refresh' and a 'Shining wizard' icon. The main workspace is divided into two panes. The left pane, titled 'Sketch Files', shows a file tree with 'index.html' selected, and other files listed below it: 'particle.js', 'sketch.js', and 'style.css'. The right pane shows the content of 'index.html' as an HTML document. The code is as follows:

```
1 <!DOCTYPE html>
2 <html lang="en">
3   <head>
4     <script src="https://cdn.jsdelivr.net/npm/p5@2.3.4/lib/p5.js"></script>
5     <script src="https://cdn.jsdelivr.net/npm/p5.sound@0.4.1/dist/p5.sound.min.js"></script>
6
7
8
9     <link rel="stylesheet" type="text/css" href="style.css">
10    <meta charset="utf-8" />
11
12  </head>
13  <body>
14    <main>
15    </main>
16    <script src="sketch.js"></script>
17    <script src="particle.js"></script>
18  </body>
19 </html>
```

At the bottom of the IDE, there is a 'Console' pane with a 'Clear' button.

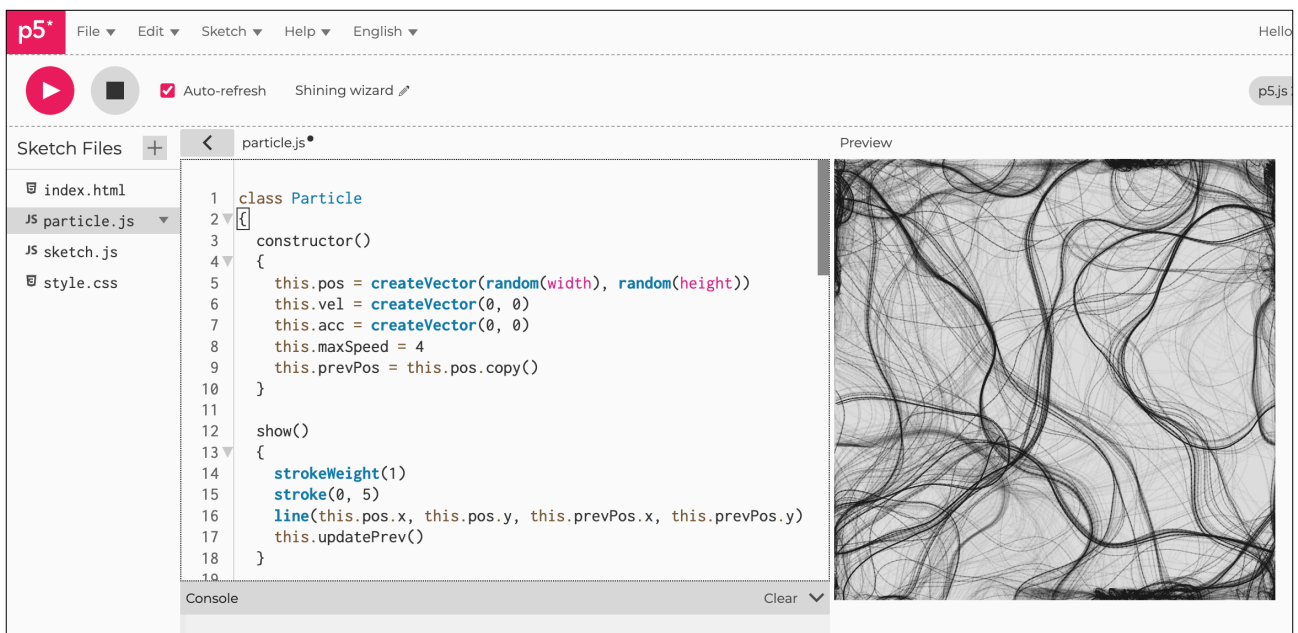


Adding the code

Go back and get the code for the Flowfield and put it in sketch.js if it isn't already there. Copy and cut out the Particle class code. Then click on the particle.js file. You will notice that it is completely empty. Paste the Particle class code in there. It should work just as before. If it doesn't then check you have copied all of the Particle class and haven't left it in the sketch.js file.

You have now created a separate file system. It is not essential but considered good practice. I haven't used it because I think it gets a bit messy when you multiply classes in a workbook.

Figure 7: the Particle class in the particle.js file





A photo

Another thing we can do is upload images, photos and videos as well as music files, 3D objects and fonts. Here we are going to upload a photo, a single image. The principle is just the same no matter type of file it is. What you need is a photo or image. This is the one I am going to use, cute isn't it?

Figure 8: my image





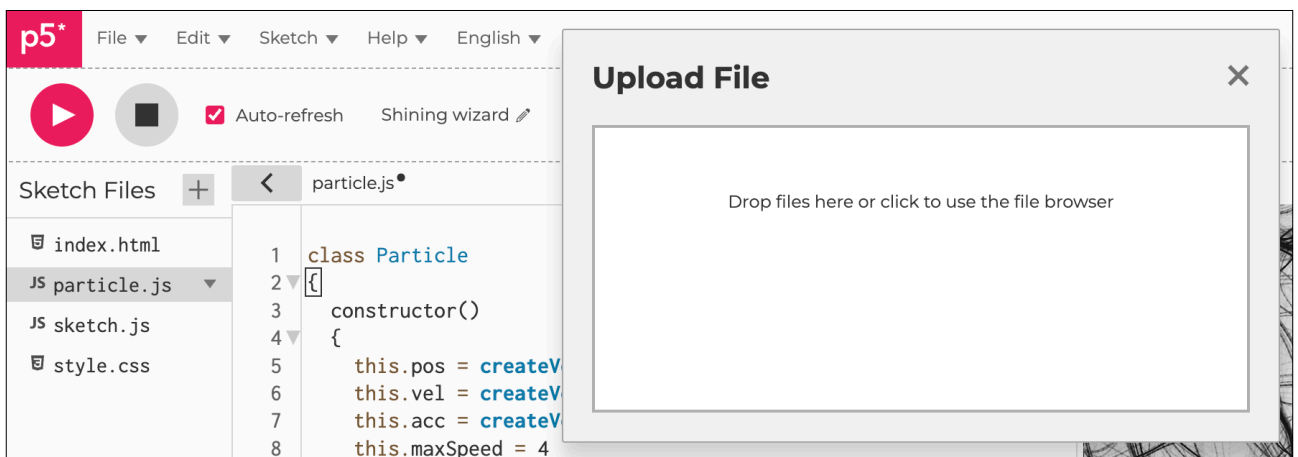
Back to the files

This time we want to upload a file, so click on the + button and click on Upload file, you will get a box

Figure 9: uploading a file



Figure 10: upload file box



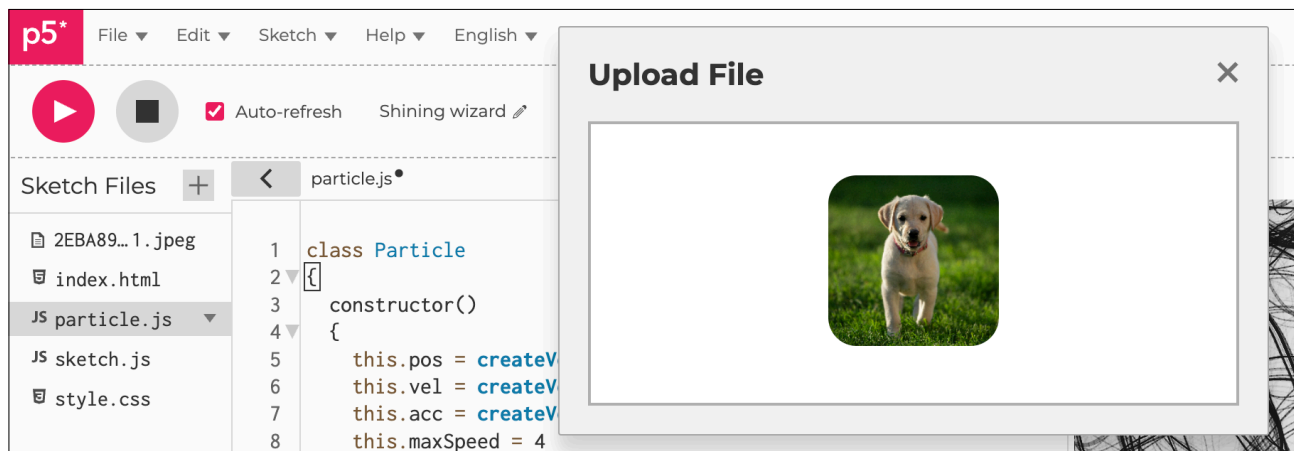


Drop or browse

There are two ways you can add an image (or any other file), if you have the image on your desktop or somewhere easily accessible then you can drag it into the box and drop it in. Otherwise you can click on the box and it will assume that you are browsing, then go to where your image is stored (photos probably) and select it.

It will then start uploading it. If successful you get a nice tick and then the file will appear on the left-hand side.

Figure 11: image uploaded





Renaming the file

If you click on the file (on the left) you will see the image you have uploaded. But in this case (as will be in most cases) the name of the file is unhelpful. The good news is that we can easily rename it. It is important that you keep the file suffix as it was in the original, for instance jpeg or jpg and so on. We are going to call ours dog for obvious reasons. Click on the little downward arrow next to the name of the file.

Figure 12: renaming

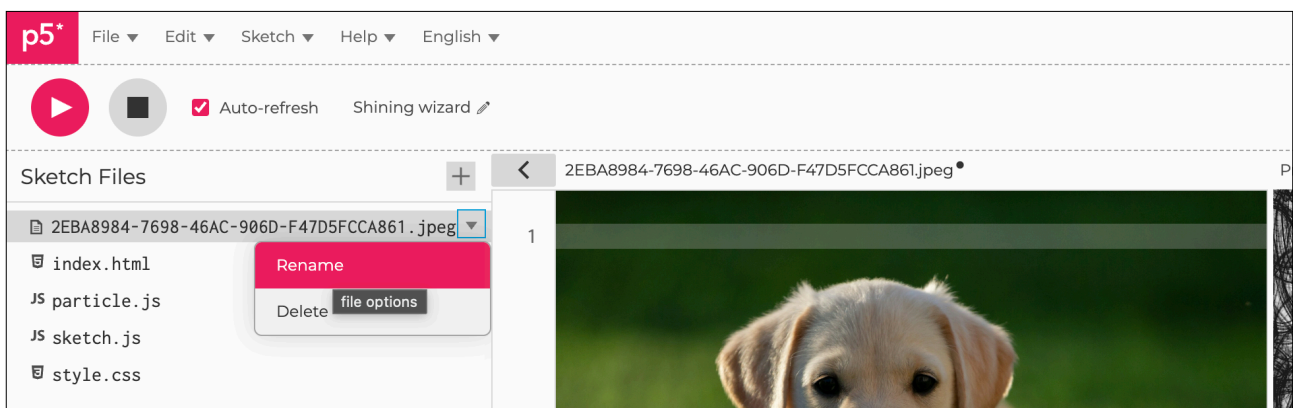
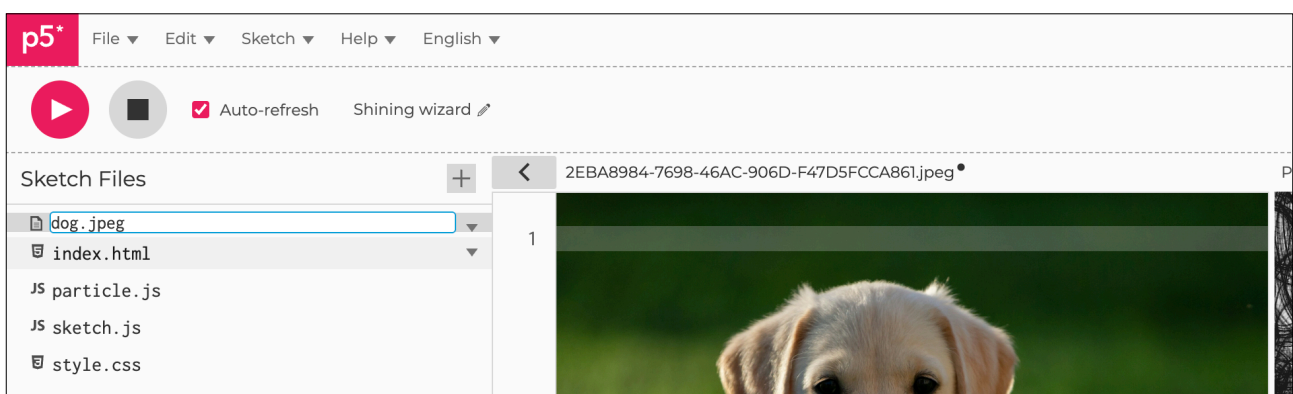


Figure 13: dog.jpeg





What next?

We have uploaded our file and so what next? We can then incorporate the file into our sketch but that is for another workbook, but it is important that you name your file carefully because you will need to reference it exactly in the sketch including it being case sensitive.

One other thing is the size of the file, there is a limit. So if your file is too big (I am unsure what it is currently) it won't upload it. I have had this issue with some large images and some videos. So be aware that you may need to adjust your expectations.

You can upload many images and when you do that you can create a folder (see the other tab option) to put all your image in. Your folder may contain cats and another folder may contain images of dogs etc. If you then use it in your code you need to reference the folder as well as the file name, for instance dogs/dog.jpeg where the folder is called dogs.

That is covering the basics and later on you will be adding other libraries as <scripts>... but that is again for another workbook.