Contents

Unit #43 Perlin Noise
Sketch A19.43.1 random
Sketch A19.43.2 perlin
Sketch A19.43.3 perlin offset
Sketch A19.43.4 x and y perlin
Sketch A19.43.5 graphing random
Sketch A19.43.6 graphing perlin noise
Sketch A19.43.7 a moving perlin graph
Sketch A19.43.8 random pixels
Sketch A19.43.9 2D perlin noise

Unit #44 Random mover
Sketch A19.44.1 random walker
Sketch A19.44.2 vector addition
Sketch A19.44.3 a random vector
Sketch A19.44.4 normalise vector
Sketch A19.44.5 accelerating a vector
Sketch A19.44.6 accelerating towards the mouse

Unit #45 Forces
Sketch A19.45.1 gravity and wind
Sketch A19.45.2 mass and acceleration
Sketch A19.45.3 friction
Sketch A19.45.4 drag
Sketch A19.45.5 gravitational attraction

Introduction

This explores how to simulate random movement that might be found in nature.
Unit #43 Perlin Noise

Content
Sketch A19.43.1 random
Sketch A19.43.2 perlin
Sketch A19.43.3 perlin offset
Sketch A19.43.4 x and y perlin
Sketch A19.43.5 graphing random
Sketch A19.43.6 graphing perlin noise
Sketch A19.43.7 a moving perlin graph
Sketch A19.43.8 random pixels
Sketch A19.43.9 2D perlin noise

Introduction
Random and perlin noise random. Two very different types of random.
A simple random function

```
function setup() {
  createCanvas(400, 400)
}

function draw() {
  background(220)
  let x = random(width)
  circle(x, 200, 50)
}
```

The circle just positions itself randomly.
Notes
This is just the basic random function. Returning a random value for x between 0 and the width. This is far from smooth, the perlin noise function will help to improve the motion of the circle as you will see in the next sketch. Random has a normal distribution whereas perlin is around a fixed value.
Instead of random let’s use perlin noise which is linked to the previous random value unlike random()

Although it isn’t moving in the draw loop() it moves randomly relatively to the previous position

Notes
This is a basic perlin noise sketch, it is a random number generator giving a number value between 0 and 1. So every time you refresh the sketch it moves the circle across the canvas. The value 100 is the time stamp at the point at which the sketch is run.

Challenges
1. Try different values for noise
2. Draw a few circles at those values

```javascript
function setup()
{
  createCanvas(400, 400)
}

function draw()
{
  background(220)
  let x = map(noise(100), 0, 1, 0, width)
  circle(x, 200, 50)
}
```
Smooth random

<table>
<thead>
<tr>
<th>sketch.js</th>
</tr>
</thead>
<tbody>
<tr>
<td>let xoff = 0</td>
</tr>
</tbody>
</table>

```javascript
function setup()
{
    createCanvas(400, 400)
}

function draw()
{
    background(220)
    let x = map(noise(xoff), 0, 1, 0, width)
    xoff += 0.01
    circle(x, 200, 50)
}
```

You will see a nice smooth movement of the circle

**Notes**
No we add an offset (xoff) to increment the perlin value along the value line by a small amount (0.01). This is still random but it relates to the previous value rather than just returning a random value. It creates a nice smooth

**Challenges**
1. Change the xoff value, increasing and decreasing it
Perlin in the x and y direction

```
sketch.js

let xoff = 0
let yoff = 100

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

function draw()
{
    background(220)
    let x = map(noise(xoff), 0, 1, 0, width)
    let y = map(noise(yoff), 0, 1, 0, height)
    xoff += 0.01
    yoff += 0.01
    circle(x, y, 50)
}
```

Like a floating balloon

**Notes**
This gives the circle movement in all directions.

**Challenges**
What would happen if xoff and yoff were the same initial value?
A simple line graph demonstrating randomness

```
function setup()
{
  createCanvas(600, 600)
}

function draw()
{
  background(220)
  noFill()
  beginShape()
  for(let x = 0; x < width; x++)
  {
    let y = random(height)
    vertex(x, y)
  }
  endShape()
  noLoop()
}
```

Looks quite brutal
Notes
This is what random looks like. Drawing points (vertices) at those random points and connecting them together.

Challenges
Remove the noLoop()
Compare the previous with perlin noise

```javascript
let yoff = 0

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

function draw() {
    background(220)
    noFill()
    beginShape()
    for(let x = 0; x < width; x++) {
        let y = map(noise(yoff), 0, 1, 0, height)
        vertex(x, y)
        yoff += 0.01
    }
    endShape()
    noLoop()
}
```
Notes
You can now see what the difference is between graphing random and perlin noise.
The graph now moves serenely across the canvas

```javascript
let start = 0

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

function draw()
{
  background(220)
  noFill()
  beginShape()
  let yoff = start
  for(let x = 0; x < width; x++)
  {
    let y = map(noise(yoff), 0, 1, 0, height)
    vertex(x, y)
    yoff += 0.01
  }
  endShape()
  start += 0.01
}
```
Notes
Very similar to the static perlin graph. In this we just increment the start value for the y offset.

Challenges
1. Remove the noFill() for a nice effect
2. Change the colours and add some alpha
Sketch A19.43.8 random pixels

Pure randomness

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

function draw() {
  loadPixels()
  background(220)
  noFill()
  for (let x = 0; x < width; x++) {
    for (let y = 0; y < height; y++) {
      let index = (x + y * width) * 4
      let r = random(255)
      pixels[index + 0] = r
      pixels[index + 1] = r
      pixels[index + 2] = r
      pixels[index + 3] = 255
    }
  }
  updatePixels()
  noLoop()
}
```
Notes
Creating a simple random pixel canvas with. Two nested loops to work through each x and y. [see the section about pixel arrays for further understanding]. The pixelDensity of 1 is because you might have a High Density monitor that has more than a red, green, blue and alpha for each pixel.

Challenges
Take the noLoop() off
A more realistic image of random

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

function draw()
{
  loadPixels()
  background(220)
  noFill()
  let yoff = 0
  for (let x = 0; x < width; x++)
  {
    let xoff = 0
    for (let y = 0; y < height; y++)
    {
      let index = (x + y * width) * 4
      let r = noise(xoff, yoff) * 255
      pixels[index + 0] = r
      pixels[index + 1] = r
      pixels[index + 2] = r
      pixels[index + 3] = 255
      xoff += 0.01
    }
    yoff += 0.01
  }
```
A very pleasing smoothing of the random pixels

Notes
This uses perlin noise to generate a randomness which is more like blurring where the random value is closer to the ones around it rather than separate from.

Challenges
Now do it with three colours... see below
function setup()
{
    createCanvas(400, 400)
    pixelDensity(1)
}

function draw()
{
    loadPixels()
    background(220)
    noFill()
    let yoffr = 0
    let yoffg = 10
    let yoffb = 100
    for (let x = 0; x < width; x++)
    {
        let xoffr = 0
        let xoffg = 10
        let xoffb = 100
        for (let y = 0; y < height; y++)
        {
            let index = (x + y * width) * 4
            let r = noise(xoffr, yoffr) * 255
            let g = noise(xoffg, yoffg) * 255
            let b = noise(xoffb, yoffb) * 255
            pixels[index + 0] = r
            pixels[index + 1] = g
            pixels[index + 2] = b
            pixels[index + 3] = 255
            xoffr += 0.01
            xoffg += 0.01
            xoffb += 0.01
```java
} yoffr += 0.01
yoffg += 0.01
yoffb += 0.01

updatePixels()
```
Unit #44 A Random Mover

Content
Sketch A19.44.1 random walker
Sketch A19.44.2 vector addition
Sketch A19.44.3 a random vector
Sketch A19.44.4 normalise vector
Sketch A19.44.5 accelerating a vector
Sketch A19.44.6 accelerating towards the mouse

Introduction

How can you make random movement look more natural and less random, yet retain a randomness, using vectors.
Part 1: Add two more files...

mover.js
attractor.js

index.html

```html
<!DOCTYPE html>
<html lang="en">
  <head>
    <script src="https://cdnjs.cloudflare.com/ajax/libs/p5.js/1.2.0/p5.js"></script>
    <script src="https://cdnjs.cloudflare.com/ajax/libs/p5.js/1.2.0/addons/p5.sound.min.js"></script>
    <link rel="stylesheet" type="text/css" href="style.css">
    <meta charset="utf-8" />
  </head>
  
  <body>
    <script src="sketch.js"></script>
    <script src="mover.js"></script>
    <script src="attractor.js"></script>
  </body>
</html>
```
Part 2: The main sketch

```javascript
let mover

function setup()
{
createCanvas(400, 400)
mover = new Mover(width/2, height/2)
background(220)
}

function draw()
{
mover.update()
mover.show()
}```
Part 3: In mover.js

```javascript
mover.js

class Mover {
    constructor(x, y) {
        this.pos = createVector(x, y)
    }
    update() {
        this.pos.x = this.pos.x + random(-5, 5)
        this.pos.y = this.pos.y + random(-5, 5)
    }
    show() {
        strokeWeight(4)
        point(this.pos.x, this.pos.y)
    }
}
```
Notes
A fairly simple self explanatory use of a class for object orientated programming.

Challenges
1. Add a constraint to stop it falling off the edge of the canvas by adding the following lines of code in the update() section of the walker class

```
this.pos.x = constrain(this.pos.x, 0, width)
this.pos.y = constrain(this.pos.y, 0, height)
```

2. Use perlin noise for the size of the point
3. Have other shapes e.g circle
Using the add function to a vector in this case we add velocity top its position (in the x and y direction Remove the lines of code in the update() and show() functions

```javascript
mover.js

class Mover
{
    constructor(x, y)
    {
        this.pos = createVector(x, y)
        this.vel = createVector(1, -1)
    }
    update()
    {
        this.pos.add(this.vel)
    }
    show()
    {
        strokeWeight(4)
        point(this.pos.x, this.pos.y)
    }
}
```
Notes
Here we are going to add vectors, adding a velocity component to a positional vector using .add(). See the reference page for p5.Vector for more vector maths

Challenge
Change the values of the velocity vector e.g.
this.vel = createVector(-3, 0.1)
Part 1: Drawing a line using a vector

```javascript
function setup()
{
    createCanvas(400, 400)
    background(220)
    strokeWeight(4)
}

function draw()
{
    translate(width/2, height/2)
    let v = createVector(100, 0)
    line(0, 0, v.x, v.y)
}
```
Part 2: Randomising the line

```javascript
function setup() {
    createCanvas(400, 400)
    background(220)
    strokeWeight(4)
}

function draw() {
    translate(width/2, height/2)
    let v = createVector(random(-100, 100), random(-100, 100))
    line(0, 0, v.x, v.y)
}
```
Part 3: Using random2D() with p5.Vector maths and multiplying using .mult()

```
function setup()
{
    createCanvas(400, 400)
    background(220)
    strokeWeight(4)
}

function draw()
{
    translate(width/2, height/2)
    let v = p5.Vector.random2D()
    v.mult(100)
    line(0, 0, v.x, v.y)
}
```

Notes
Notice that the length is uniform
Part 4: randomising the length it is multiplied

<table>
<thead>
<tr>
<th>sketch.js</th>
</tr>
</thead>
<tbody>
<tr>
<td>function setup()</td>
</tr>
<tr>
<td>{</td>
</tr>
<tr>
<td>createCanvas(400, 400)</td>
</tr>
<tr>
<td>background(220)</td>
</tr>
<tr>
<td>strokeWeight(4)</td>
</tr>
<tr>
<td>}</td>
</tr>
</tbody>
</table>

function draw()  
{            
  translate(width/2, height/2)  
  let v = p5.Vector.random2D()  
  v.mult(random(100), random(100))  
  line(0, 0, v.x, v.y)  
}

Notes
Introduction to .random2D() and .mult() from p5.Vector maths.

Challenge
1. Add some flour and some alpha to the line  
2. Could you add something like this to the random walker sketch 4:2-2 (vector addition)
Part 1: Use `sub` with `p5.Vector` maths to subtract two vectors. Then use `.mag()`, `.div()`, `.mag()` and `.mult()` to normalise then fix the magnitude of the line.

```javascript
function setup() {
  createCanvas(400, 400)
  strokeWeight(4)
}

function draw() {
  background(220)
  let pos = createVector(width/2, height/2)
  let mouse = createVector(mouseX, mouseY)
  let v = p5.Vector.sub(mouse, pos)
  let m = v.mag()
  v.div(m)
  v.mult(100)
  translate(width/2, height/2)
  line(0, 0, v.x, v.y)
}
```
Part 2: Use the .\texttt{normalize}() function instead of .\texttt{mag}() and .\texttt{div}(), notice the American spelling!

Note: remove \texttt{let m = v.mag() v.div(m)}

```
sketch.js
function setup()
{
  createCanvas(400, 400)
  strokeWeight(4)
}

function draw()
{
  background(220)
  let pos = createVector(width/2, height/2)
  let mouse = createVector(mouseX, mouseY)
  let v = p5.Vector.sub(mouse, pos)
  v.normalize()
  v.mult(100)
  translate(width/2, height/2)
  line(0, 0, v.x, v.y)
}
```
Part 3: There is an even simpler way of achieving the same result by replacing `.normalize()` and `.mult()` with `.setMag()`

Note: remove

```javascript
v.normalize()
```

```javascript
v.mult(100)
```

---

### Notes

To normalise means to make something between 0 and 1, or in this case to make whatever magnitude (or length) to unit 1 and then effectively fixing it to some other value consistently e.g. 100

You can also chain things if you want to, e.g. `v.normalize().mult(100)` rather than have them on separate lines.
Part 1: Setting up the sketch.js

```javascript
let mover

function setup() {
    createCanvas(400, 400)
    mover = new Mover(width/2, height/2)
    background(220)
    strokeWeight(4)
}

function draw() {
    mover.update()
    mover.show()
}
```

Notes
(NoC 2 video 1.6 acceleration vector)
Part 2: The mover.js looks like this

```javascript
class Mover {
  constructor(x, y) {
    this.pos = createVector(x, y)
    this.vel = p5.Vector.random2D()
    this.vel.mult(random(3))
  }

  update() {
    this.pos.add(this.vel)
  }

  show() {
    point(this.pos.x, this.pos.y)
  }
}
```
Part 3: Introducing acceleration and making it a random value. The acceleration is added to the velocity which is added to the position. To stop (or limit) the amount of acceleration we set the magnitude to 0.01 in this case. Using .setMag(0.01)

```javascript
mover.js

class Mover {
  constructor(x, y) {
    this.pos = createVector(x, y)
    this.vel = p5.Vector.random2D()
    this.vel.mult(random(3))
    this.acc = p5.Vector.random2D()
    this.acc.setMag(0.01)
  }

  update() {
    this.pos.add(this.vel)
    this.vel.add(this.acc)
  }

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

www.TheHappyCoder.org
"of" 39 82
Section A19"
You will get a curved line similar to so, each time you will get a different curve because of the random2D
Part 4: Adding the .limit() function

Note: remove `this.acc.setMag(0.01)`

move `this.acc = p5.Vector.random2D()` into `update`

```javascript
mover.js

class Mover {
  constructor(x, y) {
    this.pos = createVector(x, y)
    this.vel = p5.Vector.random2D()
    this.vel.mult(random(3))
  }

  update() {
    this.acc = p5.Vector.random2D()
    this.pos.add(this.vel)
    this.vel.add(this.acc)
    this.vel.limit(2)
  }

  show() {
    point(this.pos.x, this.pos.y)
  }
}
```
Move the background into draw. The .sub() function calculates the difference between the mouse and the circle and returns the value. This has the effect of a force acting on the circle.

```javascript
mover.js

class Mover {
  constructor(x, y) {
    this.pos = createVector(x, y)
    this.vel = p5.Vector.random2D()
    this.vel.mult(random(3))
  }

  update() {
    let mouse = createVector(mouseX, mouseY)
    this.acc = p5.Vector.sub(mouse, this.pos)
    this.acc.setMag(1)
    this.pos.add(this.vel)
    this.vel.add(this.acc)
    this.vel.limit(5)
  }

  show() {
    point(this.pos.x, this.pos.y)
  }
}
```
Challenges
1. Try different values for setMag() and limit()
2. Try point, with colour and background() in setup to draw trails.
3. Try and introduce perlin noise to the acceleration.
Unit #45 Forces

Content

Sketch A19.45.1  gravity and wind
Sketch A19.45.2  mass and acceleration
Sketch A19.45.3  friction
Sketch A19.45.4  drag
Sketch A19.45.5  gravitational attraction

Introduction

Applying forces gravity, friction, drag and attraction
Part 1: Applying gravity as a force (using the random mover sketches)

```javascript
let mover

function setup()
{
    createCanvas(400, 400)
    mover = new Mover(width/2, height/2)
}

function draw()
{
    background(220)
    let gravity = createVector(0, 0.2)
    mover.applyForce(gravity)
    mover.update()
    mover.show()
    mover.edges()
}
```

**Notes**
NoC 2 video 2.1 simulating forces: gravity & wind
Part 2: Applying gravity as a force mover class

```javascript
class Mover {
  constructor(x, y) {
    this.pos = createVector(x, y)
    this.vel = createVector(0, 0)
    this.acc = createVector(0, 0)
  }

  edges() {
    if(this.pos.y >= height) {
      this.pos.y = height
      this.vel.y *= -1
    }
  }

  applyForce(force) {
    this.acc = force
  }

  update() {
    this.vel.add(this.acc)
    this.pos.add(this.vel)
    this.acc.set(0, 0)
  }
}
```
You should have nice bouncing ball
Part 3: Adding wind to gravity as a force only when the mouse is pressed

```javascript
let mover

function setup()
{
    createCanvas(600, 600)
    mover = new Mover(width/2, height/2)
}

function draw()
{
    background(220)
    if(mouseIsPressed)
    {
        let wind = createVector(0.1, 0)
        mover.applyForce(wind)
    }
    let gravity = createVector(0, 0.2)
    mover.applyForce(gravity)
    mover.update()
    mover.show()
    mover.edges()
}
```
Part 4: click on the canvas. Applying wind and gravity as forces by using the `.add()` function in the `applyForce()` function, this adds all the forces, otherwise the last one in the programme overrides the previous.

```javascript
mover.js

class Mover {

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

  edges() {
    if(this.pos.y >= height) {
      this.pos.y = height
      this.vel.y *= -1
    }
  }

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

  update() {
    this.vel.add(this.acc)
    this.pos.add(this.vel)
  }

www.TheHappyCoder.org
```
this.acc.set(0, 0)
}

show()
{
  circle(this.pos.x, this.pos.y, 20)
}
}
Challenges
1. Create an edge for the right hand side
2. And now for the right
3. Have it bounce on the edge of the circle rather than the centre
4. Could you introduce the arrow keys to blow it in either direction
Part 1: Making two spheres A and B, sphere A has mass = 2 and sphere B has mass = 4. The force has to be applied using the p5 static function p5.Vector(). The weight of each sphere is calculated by gravity multiplied by mass.

```javascript
let moverA
let moverB

function setup() {
  createCanvas(600, 600)
  moverA = new Mover(300, 300, 2)
  moverB = new Mover(300, 300, 4)
}

function draw() {
  background(0)
  if(mouseIsPressed) {
    let wind = createVector(0.1, 0)
    moverA.applyForce(wind)
    moverB.applyForce(wind)
  }
  let gravity = createVector(0, 0.2)
  let weightA = p5.Vector.mult(gravity, moverA.mass)
  let weightB = p5.Vector.mult(gravity, moverB.mass)
  moverA.applyForce(weightA)
  moverB.applyForce(weightB)
  moverA.update()
```
moverA.show()
moverA.edges()
moverB.update()
moverB.show()
moverB.edges()
Part 2: The mass of the sphere is represented by the area, hence the radius is the square root of the mass and is multiplied by an arbitrary number to make it more visible. The constructor has an extra argument m for mass. The new force f is calculated using a static function from p5.Vector().

```javascript
class Mover
{
    constructor(x, y, m)
    {
        this.pos = createVector(x, y)
        this.vel = createVector(0, 0)
        this.acc = createVector(0, 0)
        this.mass = m
        this.r = sqrt(this.mass) * 20
    }

    edges()
    {
        if(this.pos.y >= height)
        {
            this.pos.y = height
            this.vel.y *= -1
        }
        if(this.pos.x >= width)
        {
            this.pos.x = width
            this.vel.x *= -1
        }
        if(this.pos.x <= 0)
        {
            this.pos.x = 0
        }
    }
}
```
this.vel.x *= -1
}

applyForce(force)
{
  let f = p5.Vector.div(force, this.mass)
  this.acc.add(f)
}

update()
{
  this.vel.add(this.acc)
  this.pos.add(this.vel)
  this.acc.set(0, 0)
}

show()
{
  fill(0, 100)
  stroke(0)
  circle(this.pos.x, this.pos.y, this.r)
}
Part 1: This has a friction applied to the surface between the bottom edge and the sphere. So when you add the wind and the sphere reaches the bottom edge the friction it exerts stops it rolling.

My suggestion is start the sketch all over again - too many changes.
Part 2: Using classical physics to explain the nature of friction. \( F(\text{friction}) = -1\mu N \text{velocity} \). The velocity is normalised to 1. \( \mu \) is the coefficient of friction (0.1) and \( N \) is the mass of the sphere. The friction only applies when the distance (diff) between the sphere and the bottom edge is less than 1 pixel. The use of \( \text{this.r}/2 \) is to make it just touch the edges.

<table>
<thead>
<tr>
<th>mover.js</th>
</tr>
</thead>
<tbody>
<tr>
<td>class Mover</td>
</tr>
<tr>
<td>{</td>
</tr>
<tr>
<td>constructor(x, y, m)</td>
</tr>
<tr>
<td>{</td>
</tr>
<tr>
<td>this.pos = createVector(x, y)</td>
</tr>
<tr>
<td>this.vel = createVector(0, 0)</td>
</tr>
<tr>
<td>this.acc = createVector(0, 0)</td>
</tr>
<tr>
<td>this.mass = m</td>
</tr>
<tr>
<td>this.r = sqrt(this.mass) * 20</td>
</tr>
<tr>
<td>}</td>
</tr>
<tr>
<td>edges()</td>
</tr>
<tr>
<td>{</td>
</tr>
<tr>
<td>if(this.pos.y &gt;= height - this.r/2)</td>
</tr>
<tr>
<td>{</td>
</tr>
<tr>
<td>this.pos.y = height - this.r/2</td>
</tr>
<tr>
<td>this.vel.y *= -1</td>
</tr>
<tr>
<td>}</td>
</tr>
<tr>
<td>if(this.pos.x &gt;= width - this.r/2)</td>
</tr>
<tr>
<td>{</td>
</tr>
<tr>
<td>this.pos.x = width - this.r/2</td>
</tr>
<tr>
<td>this.vel.x *= -1</td>
</tr>
<tr>
<td>}</td>
</tr>
<tr>
<td>if(this.pos.x &lt;= 0 + this.r/2)</td>
</tr>
<tr>
<td>{</td>
</tr>
<tr>
<td>this.pos.x = 0 + this.r/2</td>
</tr>
<tr>
<td>this.vel.x *= -1</td>
</tr>
<tr>
<td>}</td>
</tr>
<tr>
<td>}</td>
</tr>
</tbody>
</table>
```javascript
this.pos.x = 0 + this.r/2
this.vel.x *= -1
}
}

friction()
{
    let diff = height - (this.pos.y + this.r)
    if(diff < 1)
    {
        let friction = this.vel.copy()
        friction.normalize()
        friction.mult(-1)
        let mu = 0.1
        let normal = this.mass
        friction.setMag(mu * normal)
        this.applyForce(friction)
    }
}

applyForce(force)
{
    let f = p5.Vector.div(force, this.mass)
    this.acc.add(f)
}

update()
{
    this.vel.add(this.acc)
    this.pos.add(this.vel)
    this.acc.set(0, 0)
}
```
show()
{
    fill(0, 100)
    stroke(0)
    circle(this.pos.x, this.pos.y, this.r)
}
}
Part 3: Creating an array of 10 spheres. The line of code for(let mover of movers) enables all the mover to have all the array of movers objects

<table>
<thead>
<tr>
<th>sketch.js</th>
</tr>
</thead>
<tbody>
<tr>
<td>let movers = []</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>function setup()</th>
</tr>
</thead>
<tbody>
<tr>
<td>{</td>
</tr>
<tr>
<td>createCanvas(400, 400)</td>
</tr>
<tr>
<td>for(let i = 0; i &lt; 10; i++)</td>
</tr>
<tr>
<td>{</td>
</tr>
<tr>
<td>movers[i] = new Mover(random(width), 100, random(1, 8))</td>
</tr>
<tr>
<td>}</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>function draw()</td>
</tr>
<tr>
<td>---</td>
</tr>
<tr>
<td>{</td>
</tr>
<tr>
<td>background(220)</td>
</tr>
<tr>
<td>for(let mover of movers)</td>
</tr>
<tr>
<td>{</td>
</tr>
<tr>
<td>if(mouseIsPressed)</td>
</tr>
<tr>
<td>{</td>
</tr>
<tr>
<td>let wind = createVector(0.1, 0)</td>
</tr>
<tr>
<td>mover.applyForce(wind)</td>
</tr>
<tr>
<td>}</td>
</tr>
<tr>
<td>let gravity = createVector(0, 0.2)</td>
</tr>
<tr>
<td>let weight = p5.Vector.mult(gravity, mover.mass)</td>
</tr>
<tr>
<td>mover.applyForce(weight)</td>
</tr>
<tr>
<td>mover.friction()</td>
</tr>
<tr>
<td>mover.update()</td>
</tr>
<tr>
<td>mover.show()</td>
</tr>
<tr>
<td>mover.edges()</td>
</tr>
</tbody>
</table>
The mover.js sketch remains the same. Which is why object orientated programming is so powerful.
Part 1: This takes the effect of \( \rho \) the density of the fluid. It is dependant on the velocity squared.

Remove \texttt{mover.friction()}

```
let movers = []

function setup()
{
    createCanvas(400, 400)
    for(let i = 0; i < 10; i++)
    {
        movers[i] = new Mover(random(width), 100, random(1, 8))
    }
}

function draw()
{
    background(220)
    for(let mover of movers)
    {
        if(mouseIsPressed)
        {
            let wind = createVector(0.1, 0)
            mover.applyForce(wind)
        }
        let gravity = createVector(0, 0.2)
        let weight = p5.Vector.mult(gravity, mover.mass)
        mover.applyForce(weight)
        mover.drag()
    }
}
```
mover.update()
mover.show()
mover.edges()
}
Part 2: Instead of friction we will have drag where $c$ is the general coefficient of drag. The velocity magnitude is squared using `magSq()`

```javascript
class Mover {
    constructor(x, y, m) {
        this.pos = createVector(x, y)
        this.vel = createVector(0, 0)
        this.acc = createVector(0, 0)
        this.mass = m
        this.r = sqrt(this.mass) * 20
    }

    edges() {
        if(this.pos.y >= height - this.r/2) {
            this.pos.y = height - this.r/2
            this.vel.y *= -1
        }
        if(this.pos.x >= width - this.r/2) {
            this.pos.x = width - this.r/2
            this.vel.x *= -1
        }
        if(this.pos.x <= 0 + this.r/2) {
            this.pos.x = 0 + this.r/2
            this.vel.x *= -1
        }
    }
}
```
drag()
{
    let drag = this.vel.copy()
    drag.normalize()
    drag.mult(-1)
    let c = 0.1
    let speedSq = this.vel.magSq()
    drag.setMag(c * speedSq)
    this.applyForce(drag)
}

applyForce(force)
{
    let f = p5.Vector.div(force, this.mass)
    this.acc.add(f)
}

update()
{
    this.vel.add(this.acc)
    this.pos.add(this.vel)
    this.acc.set(0, 0)
}

show()
{
    fill(0, 100)
    stroke(0)
    circle(this.pos.x, this.pos.y, this.r)
}
Part 3: Creating a fluid and a vacuum to show the effect of drag. The mover.js sketch remains the same.

```
sketch.js

let movers = []

function setup()
{
    createCanvas(400, 400)
    for (let i = 0; i < 10; i++)
    {
        movers[i] = new Mover(random(width), 100, random(1, 8))
    }
}

function draw()
{
    background(220)
    fill(255, 75)
    noStroke()
    rect(0, height / 2, width, height / 2)
    for (let mover of movers)
    {
        if (mouseIsPressed)
        {
            let wind = createVector(0.1, 0)
            mover.applyForce(wind)
        }
        let gravity = createVector(0, 0.2)
        let weight = p5.Vector.mult(gravity, mover.mass)
        mover.applyForce(weight)
        if (mover.pos.y > height / 2)
        {
```

www.TheHappyCoder.org

68 of 82

Section A19
mover.drag()
}
mover.update()
mover.show()
mover.edges()
}

Challenge
Try different values of drag (c), e.g. 0.5 in the mover.js sketch - run it a few times to see what happens when the drag is too great.

Above the line no drag, below the line (liquid) with drag
Part 1: Add attractor.js to index.html

```html
<!DOCTYPE html>
<html lang="en">
<head>
  <script src="https://cdnjs.cloudflare.com/ajax/libs/p5.js/1.3.1/p5.js"></script>
  <script src="https://cdnjs.cloudflare.com/ajax/libs/p5.js/1.3.1/addons/p5.sound.min.js"></script>
  <link rel="stylesheet" type="text/css" href="style.css">
  <meta charset="utf-8" />
</head>
<body>
  <script src="sketch.js"></script>
  <script src="mover.js"></script>
  <script src="attractor.js"></script>
</body>
</html>
```
Part 2: The three sketches are to set the scene. Where we have the two spheres which will attract each other.

```javascript
let mover
let attractor

function setup()
{
    createCanvas(400, 400)
    mover = new Mover(width/2, height/2, 5)
    attractor = new Attractor(100, 100, 5)
}

function draw()
{
    background(220)
    mover.update()
    mover.show()
    attractor.show()
}
```
Part 3: The mover class, the sphere is white. A few bits have been removed otherwise most is the same.

mover.js

class Mover
{
  constructor(x, y, m)
  {
    this.mass = m
    this.pos = createVector(x, y)
    this.vel = createVector(0, 0)
    this.acc = createVector(0, 0)
    this.r = sqrt(this.mass) * 10
  }

  applyForce(force)
  {
    let f = p5.Vector.div(force, this.mass)
    this.acc.add(f)
  }

  update()
  {
    this.vel.add(this.acc)
    this.pos.add(this.vel)
    this.acc.set(0, 0)
  }

  show()
  {
    fill(255)
    circle(this.pos.x, this.pos.y, this.r)
  }
}
Part 4: Now the attractor class, the sphere is red

<table>
<thead>
<tr>
<th>attractor.js</th>
</tr>
</thead>
<tbody>
<tr>
<td>class Attractor</td>
</tr>
<tr>
<td>{</td>
</tr>
<tr>
<td>constructor(x, y, m)</td>
</tr>
<tr>
<td>{</td>
</tr>
<tr>
<td>this.pos = createVector(x, y)</td>
</tr>
<tr>
<td>this.mass = m</td>
</tr>
<tr>
<td>this.r = sqrt(this.mass) * 10</td>
</tr>
<tr>
<td>}</td>
</tr>
<tr>
<td>show()</td>
</tr>
<tr>
<td>{</td>
</tr>
<tr>
<td>fill(200, 0, 0)</td>
</tr>
<tr>
<td>circle(this.pos.x, this.pos.y, this.r)</td>
</tr>
<tr>
<td>}</td>
</tr>
<tr>
<td>}</td>
</tr>
</tbody>
</table>

The attractor in red and the mover in white
Part 5: Adding the attractor.attract() function on the mover

```javascript
let mover
let attractor

function setup()
{
    createCanvas(600, 600)
    mover = new Mover(100, 100, 5)
    attractor = new Attractor(300, 300, 5)
}

function draw()
{
    background(100)
    mover.update()
    mover.show()
    attractor.attract(mover)
    attractor.show()
}

Nothing to report
Part 6: Adding a random velocity vector

```javascript
class Mover {
    constructor(x, y, m) {
        this.mass = m
        this.pos = createVector(x, y)
        this.vel = p5.Vector.random2D()
        this.acc = createVector(0, 0)
        this.r = sqrt(this.mass) * 10
    }

    applyForce(force) {
        let f = p5.Vector.div(force, this.mass)
        this.acc.add(f)
    }

    update() {
        this.vel.add(this.acc)
        this.pos.add(this.vel)
        this.acc.set(0, 0)
    }

    show() {
        fill(255)
        circle(this.pos.x, this.pos.y, this.r)
    }
}
```
Part 7: Introducing the equation for the attraction between two bodies.

```javascript
attactor.js

class Attractor {
  constructor(x, y, m) {
    this.pos = createVector(x, y)
    this.mass = m
    this.r = sqrt(this.mass) * 10
  }

  attract(mover) {
    let force = p5.Vector.sub(this.pos, mover.pos)
    let distSq = constrain(force.magSq(), 100, 5000)
    let G = 50
    let strength = G * (this.mass * mover.mass) / distSq
    force.setMag(strength)
    mover.applyForce(force)
  }

  show() {
    fill(200, 0, 0)
    circle(this.pos.x, this.pos.y, this.r)
  }
}
```
The mover is attracted to the attractor and will loop round continuously
Part 8: Let’s add an array of movers

```javascript
let mover
let attractor
let movers = []

function setup()
{
  createCanvas(400, 400)
  for(let i = 0; i < 20; i++)
  {
    let x = random(width)
    let y = random(height)
    movers[i] = new Mover(x, y, 5)
  }
  attractor = new Attractor(100, 100, 5)
}

function draw()
{
  background(220)
  for(let mover of movers)
  {
    mover.update()
    mover.show()
    attractor.attract(mover)
  }
  attractor.show()
}
```

Now you should have a swarm of them