The Joy of Coding Algorithmic Art Workbook #5 Classes

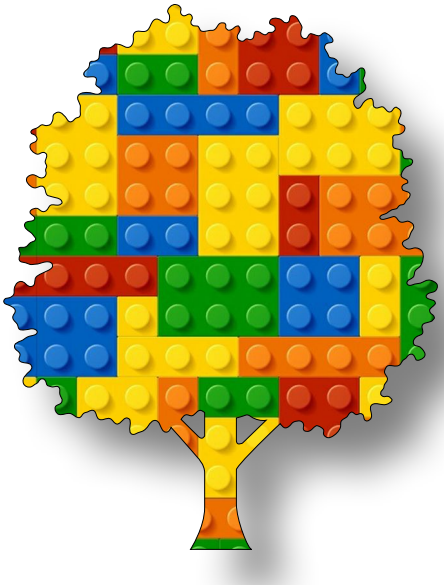


Table of Contents

MIT Licence

Brief summary:

Permissions (what you can do)

Commercial use
Modification
Distribution
Private use

Limitations (what is not covered)

Liability
Warranty

Workbook #5 Quick Content Summary

This unit looks a bit academic as well as the last one covering classes as well as more in-depth work on arrays. I hope it will become clear to you as you work through this unit how powerful some of these concepts are and what you could do with them.

In coding there are ever increasing layers of complexity. They are worth learning as they provide more ways to create and develop ideas that you may have.

The code is in the yellow boxes, any new code is highlighted in blue, so you don't have to type out the whole thing again each time. Set up an account (highly recommended) so you can save your work as you go along rather than starting again each time. It will also remind you where you were up to.

Most browsers work but Chrome is best as there is some functionality missing in some browsers. I suspect it has been tested on Chrome to guarantee it works in Chrome browser.

The Joy of Coding

Algorithmic Art

Module D Unit #1

functions and classes

Module D Unit #1: functions, objects and classes

This next section looks at coding with functions, objects, and classes. It demonstrates the different ways you can code the same effect using different approaches. The context we will use is of a vehicle or vehicles moving either across the canvas.

We can draw the vehicle with a `show()` function and its movement with a `move()` function. These are simple examples to highlight the differences to give you a flavour of what that might look like.

Sketch D1.1 starting sketch

We start a new sketch, as usual.

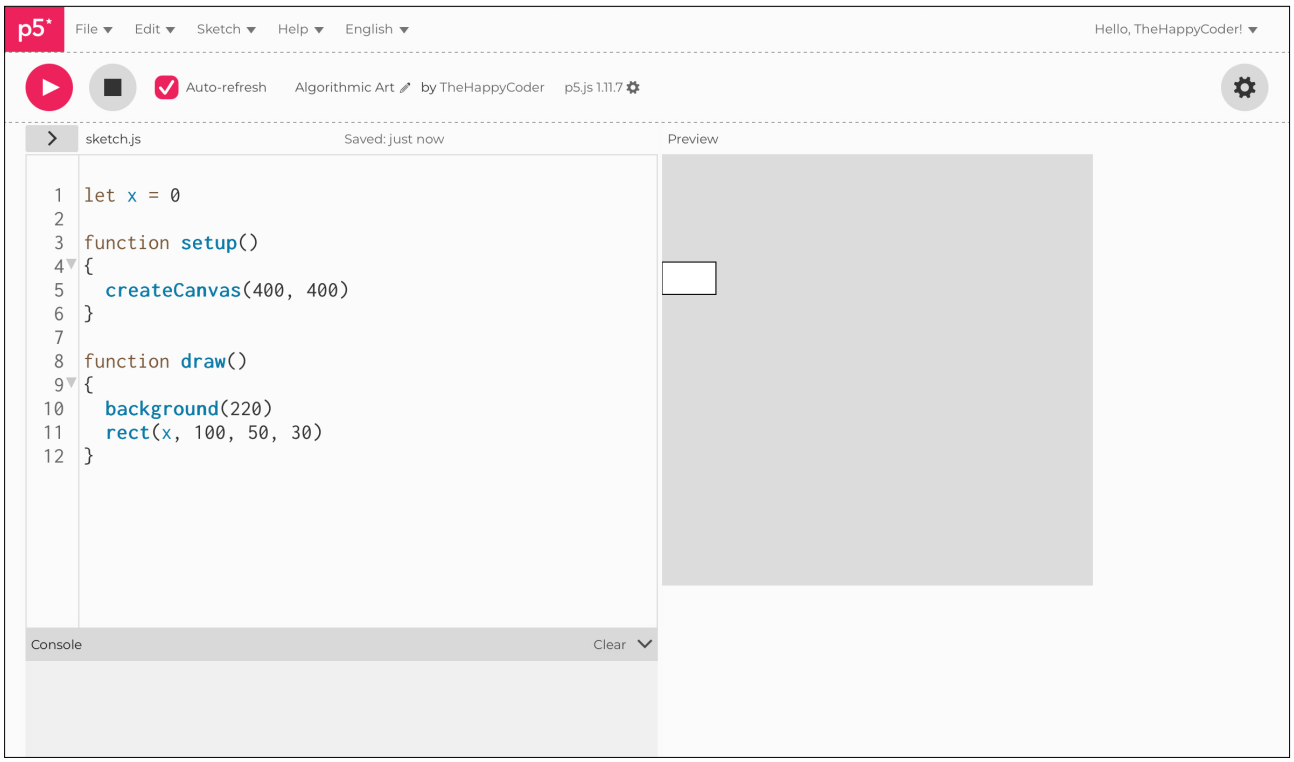
```
function setup()
{
  createCanvas(400, 400)
}

function draw()
{
  background(220)
}
```

Sketch D1.2 a single car

We create our car as a simple rectangle, starting at the left-hand edge of the canvas.

```
let x = 0

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

function draw()
{
  background(220)
  rect(x, 100, 50, 30)
}
```

Figure D1.2


p5*
File
Edit
Sketch
Help
English
Hello, TheHappyCoder!
Auto-refresh
Algorithmic Art
by TheHappyCoder
p5.js 1.11.7
sketch.js
Saved: just now
Preview
let x = 0
function setup()
{
createCanvas(400, 400)
}
function draw()
{
background(220)
rect(x, 100, 50, 30)
}
Console
Clear

Sketch D1.3 moving the car

Now we start the car moving across the canvas.

```
let x = 0

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

function draw()
{
  background(220)
  rect(x, 100, 50, 30)
  x += 1
}
```

Notes

It moves slowly across the canvas and disappears from the right-hand edge of the canvas, never to be seen again.

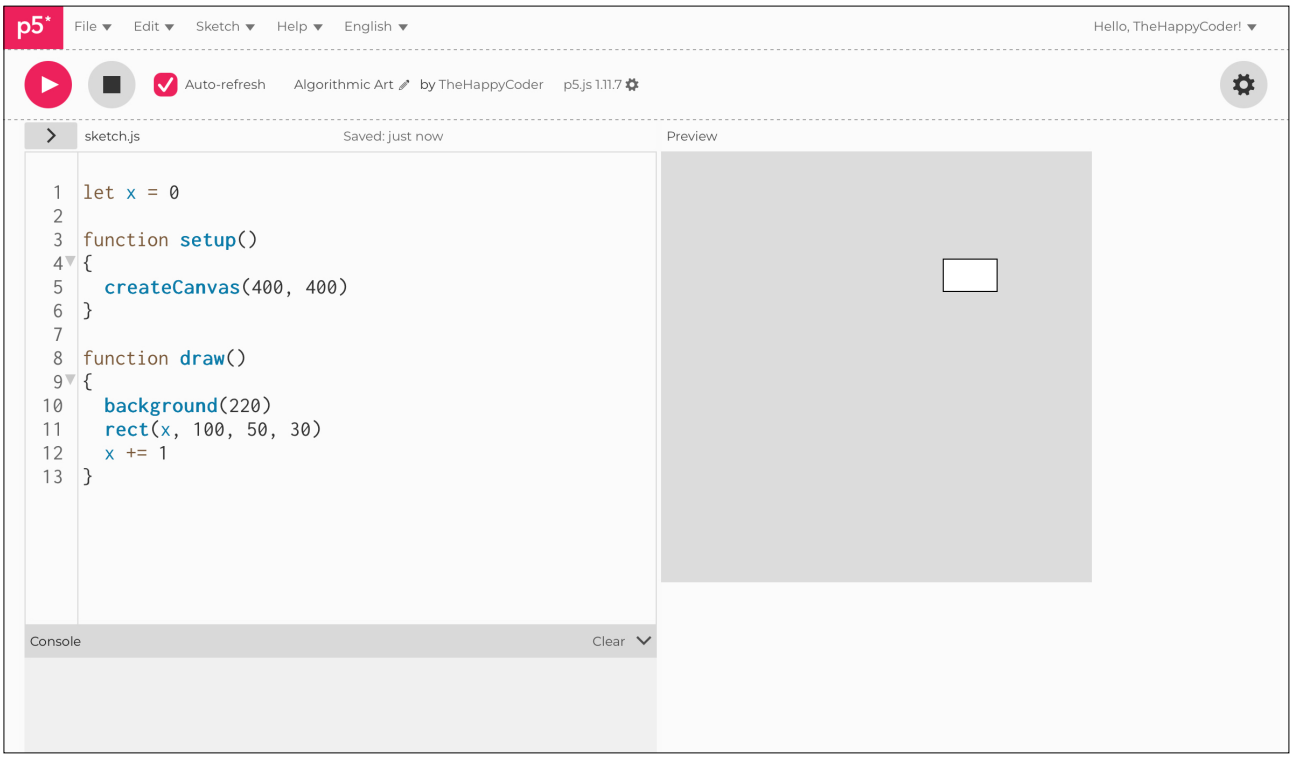
Challenge

Make it move faster.

Code Explanation

`x += 1`	This adds 1 to x on each iteration.

Figure D1.3



Sketch D1.4 it reappears

The car now reappears on the left-hand edge and off it goes again.

```
let x = 0

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

function draw()
{
  background(220)
  rect(x, 100, 50, 30)
  x += 1
  if (x > width)
  {
    x = -50
  }
}
```

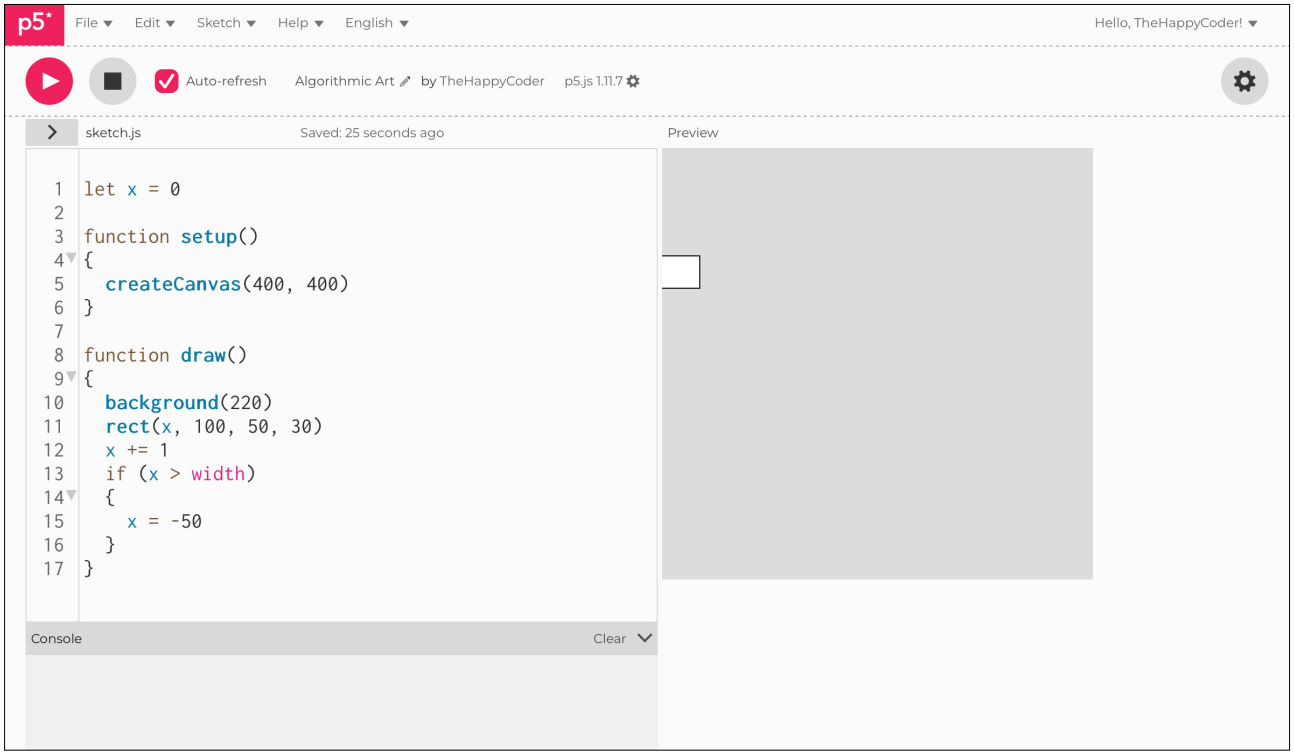
Notes

We have `x` as `-50`, so it looks like it is seamlessly continuous.

Code Explanation

`if (x > width)`	Checks to see if it has reached the edge of the canvas.
`x = -50`	Returns the x value to -50 if the car has gone off the edge of the canvas.

Figure D1.4



The car as a Function

We can express the same thing as before, but this time we use functions, two of them to describe the car and to describe the motion. This means we have the `setup()` function as before, we keep the `draw()` function (empty for now), and add the other two functions called `show()` and `move()`, putting a lot of the stuff in those new functions.

I am using a very simple example here, but bear with me as we will build on this concept when we introduce classes later.

Sketch D1.5 function single car

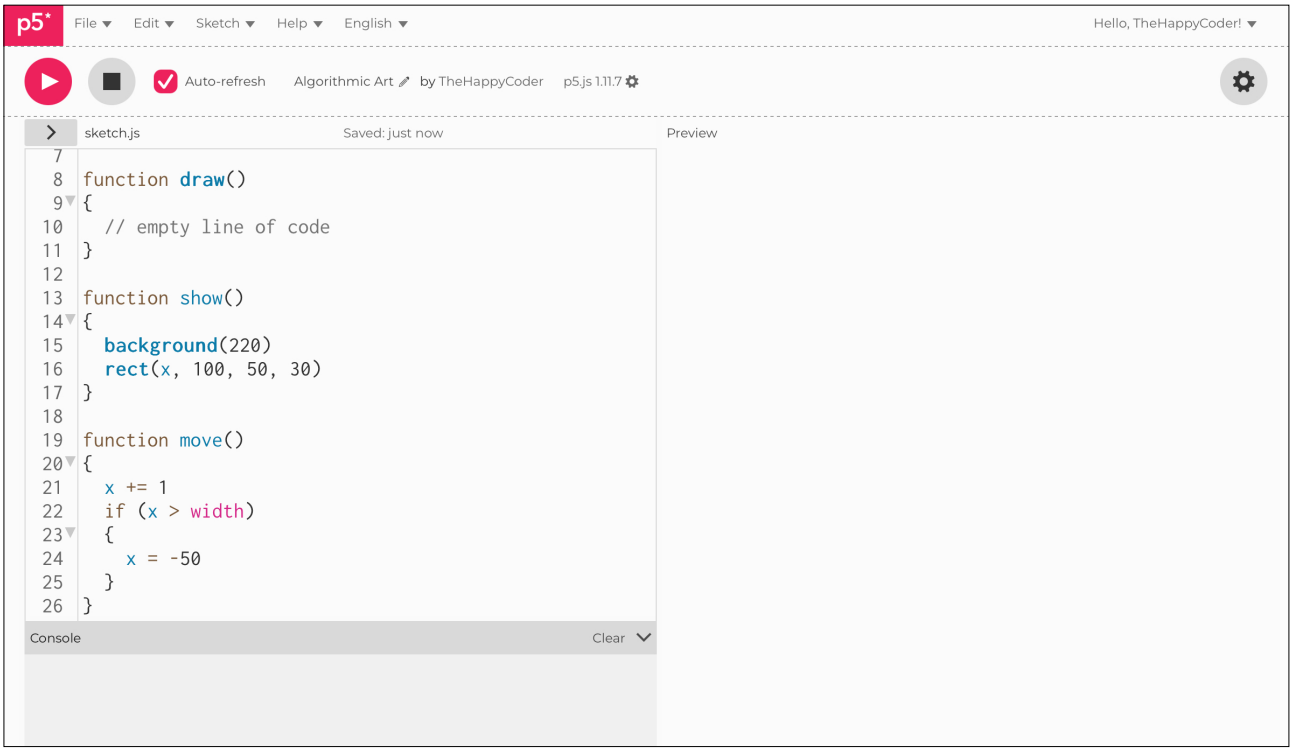
We have moved the code that was in the `draw()` function and split it between the two new functions: `show()` and `move()`. We have used the same code as in the previous sketch, just rearranged it.

```
let x = 0

function setup()
{
  createCanvas(440, 400)
}

function draw()
{
  // empty line of code
}

function show()
{
  background(220)
  rect(x, 100, 50, 30)
}

function move()
{
  x += 1
  if (x > width)
  {
    x = -50
  }
}
```

Notes

Nothing to see. You can just cut and paste to save time; however, you will notice that you get nothing, not even a canvas.

Figure D1.5



Sketch D1.6 function single car

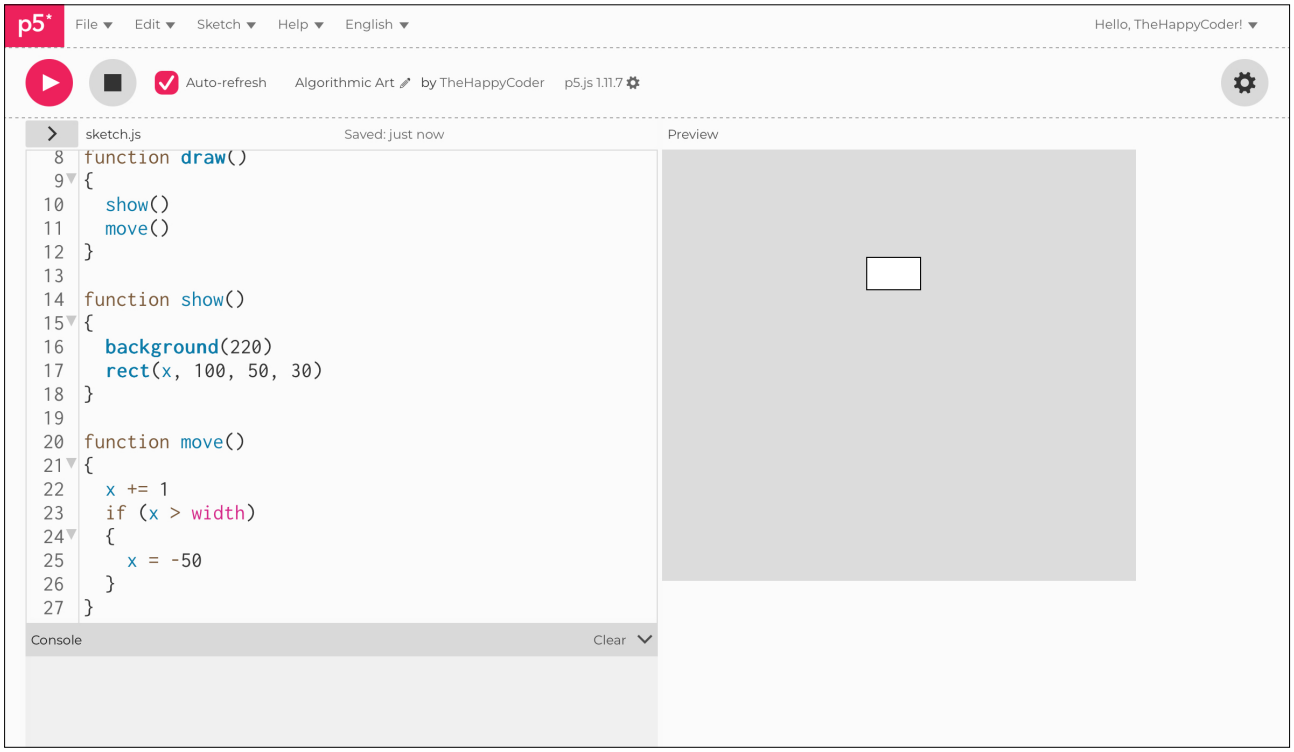
To get the two new functions to do anything, we need to call them from inside the `draw()` function, and we do it as shown below.

```
let x = 0

function setup()
{
  createCanvas(440, 400)
}

function draw()
{
  show()
  move()
}

function show()
{
  background(220)
  rect(x, 100, 50, 30)
}

function move()
{
  x += 1
  if (x > width)
  {
    x = -50
  }
}
```

Notes

Now we are back where we started, but let's not stop there; there is yet another way we can do this even before we introduce classes.

Figure D1.6



The Car as an Object using Functions

This exercise is another way of doing the same thing. I include it because it shows the concept of objects in relation to functions. We could easily create two cars, but it would mean doubling all the code for each car. This is another reason where classes come into their own, but we are getting ahead of ourselves here.

Sketch D1.7 car as an object

! Start a new sketch (highlighted differences to basic sketch)

We have added the car as an object; notice the similarity to our earlier sketch, but now we have to give it a name. In this case, we call it car.

```
let car = {x: 0}

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

function draw()
{
  background(220)
  rect(car.x, 100, 50, 30)
  car.x += 1
  if (car.x > width)
  {
    car.x = -50
  }
}
```

Notes

Everything behaves just as before. Look at the code carefully and see how you code the car as an object rather than just as a rectangle. Below is a more detailed explanation of the code; it isn't as scary as it might look.

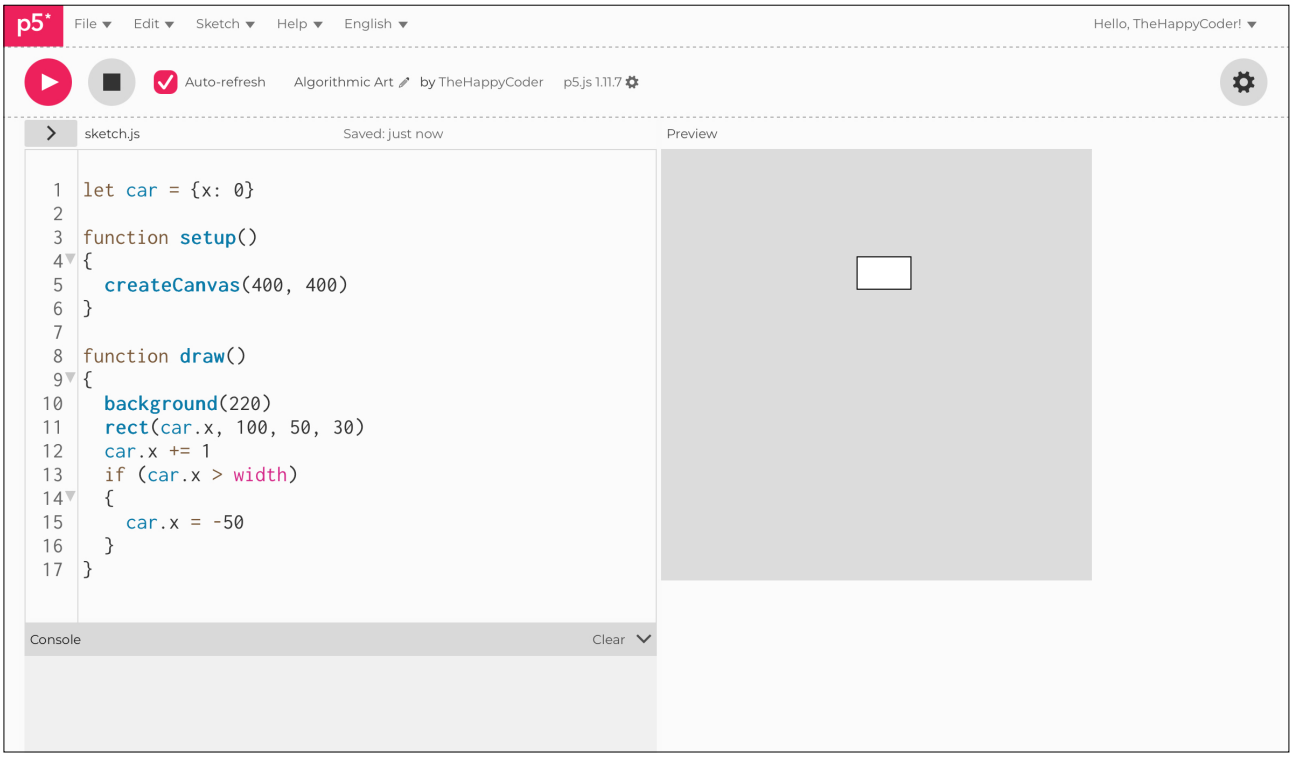
Challenges

1. Give it a `y` component
2. Introduce the `show()` and `move()` functions like we did previously (if struggling see next sketch)

Code Explanation

`let car = {x: 0}`	We initialise the x component of the car object to 0.
`rect(car.x, 100, 50, 30)`	The x component of the car object.
`car.x += 1`	Incrementing the x component by 1 on each iteration.
`if (car.x > width)`	Check when the car has gone off the edge of the canvas.
`car.x = -50`	The x component is re-initialised to -50.

Figure D1.7



Sketch D1.8 alternative car object

Now we can use the functions `show()` and `move()` as well as introducing a `y` component. The background can go into `draw()` or `show()`.

```
let car = {x: 0, y: 100}

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

function draw()
{
  background(220)
  show()
  move()
}

function show()
{
  rect(car.x, car.y, 50, 30)
}

function move()
{
  car.x += 1
  if (car.x > width)
  {
    car.x = -50
  }
}
```

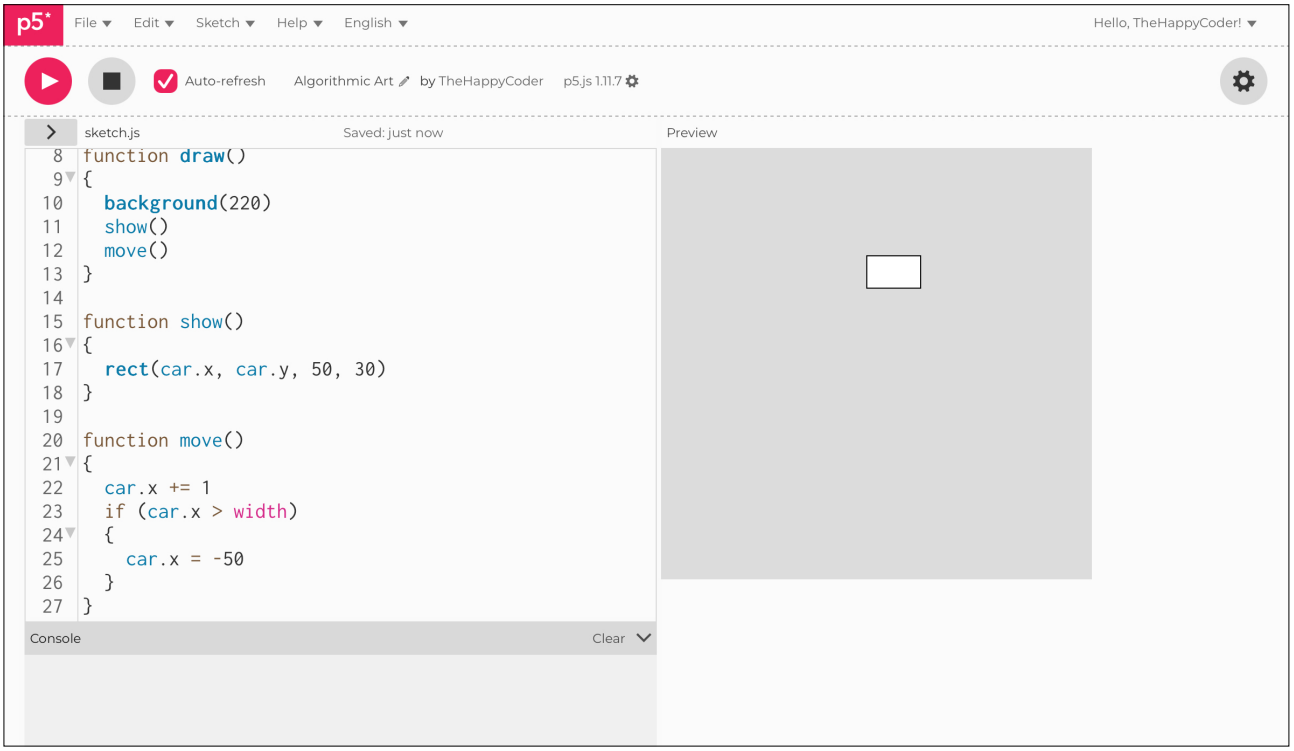
Notes

Looks quite elegant in my opinion; it should be exactly the same as before.

Code Explanation

`let car = {x: 0, y: 100}`	Adding the y component to the car object.
`rect(car.x, car.y, 50, 30)`	Adding the y component to the rectangle drawn.

Figure D1.8



Introduction to Classes

Using classes is a common way for coders to organise their code. It is not essential, as you could do the same thing without using classes, but it is a very powerful and useful approach and one worth investing the time in understanding.

It does take a bit of getting used to. I will try to illustrate this with a simple example. Imagine you have a template (or blueprint) to build a car. You, as a consumer, want some choice. The colour, the number of doors, engine size, interior style, and so on. A class is like the basic template. When you order a new car, they don't ask if you want doors, seats, a steering wheel, windows, etc. They come as standard.

A class will have the basics and the options. So that when they make 10,000 cars, they can all be slightly different depending on what the customer wants. This is a very limited comparison, but you will see that you can create lots of cars that all behave slightly differently. In our first example, we will do just that with a sort of car.

In the diagram below (`fig.1`), you will see that the class is given a name. It is usual to start the class name with a capital letter. Also, there are three functions in the example below. You can have as many functions as you like and can call them anything you like. You can see the `show()` and `move()` functions we had before, but you can have any number of functions.

The first function I use is called the `constructor()` function. This is just the usual name given to it. This is where we hold the information about any car we are going to build. Because it is a sort of template (or blueprint) where we can make as many cars as we want, we prefix any variable with the word `this`; for instance, the colour would be `this.colour`, or the starting position will be `this.x` and `this.y`, and so on.

The basic structure of the main sketch is demonstrated in `fig.2`. Where you create the car or cars from the class and call the functions from within the class.

Figure 1: class structure

```
class Car
{
  constructor()
  {
    // this.something
  }

  show()
  {
    // what it looks like
  }

  move()
  {
    // how it will move/change
  }
}
```

Figure 2: main elements in sketch

```
let car

function setup()
{
  createCanvas(400, 400)
  car = new Car()
}

function draw()
{
  background(220)
  car.show()
  car.move()
}
```

Sketch D1.9 the constructor function

! New sketch

We start with our basic sketch and create a class called Car. In that class, we have a `constructor()` function. This function has four elements that give us details about the car: its `colour`, its `x` position, its `y` position, and its `velocity`. This first example will not reveal the power of using classes but a very gentle introduction to creating a class.

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

function draw()
{
  background(220)
}
```

```
class Car
{
  constructor()
  {
    this.colour = 255
    this.x = 0
    this.y = 100
    this.velocity = 1
  }
}
```

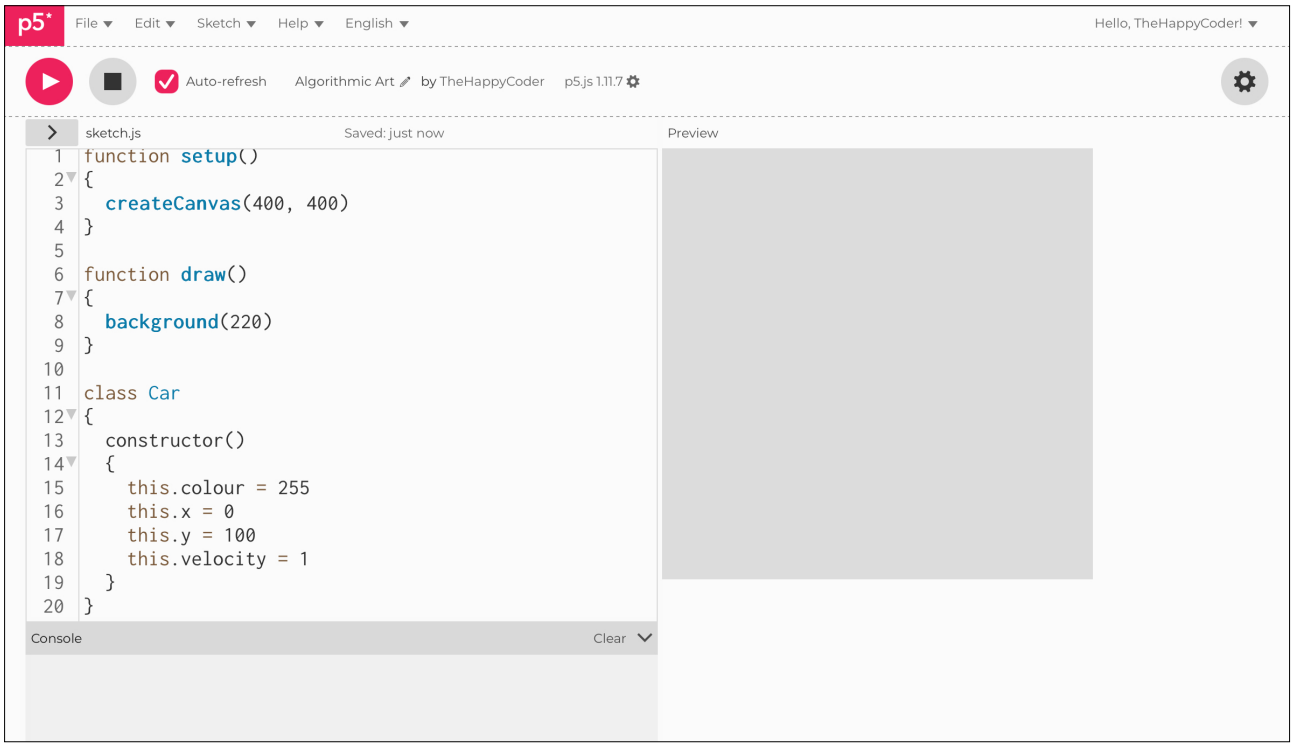
Notes

When we give attributes to an object in a class, we always use `this.` before the attribute. There is nothing to see at this point.

Code Explanation

`this.colour = 255`	For any car, we define its colour.
`this.x = 0`	For any car, we define its x position.
`this.y = 100`	For any car, we define its y position.
`this.velocity = 1`	For any car, we define its velocity.

Figure D1.9



Sketch D1.10 the show() function

In the show() function, we will describe what the car will look like.

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

function draw()
{
  background(220)
}

class Car
{
  constructor()
  {
    this.colour = 255
    this.x = 0
    this.y = 100
    this.velocity = 1
  }

  show()
  {
    fill(this.colour)
    rect(this.x, this.y, 50, 30)
  }
}
```

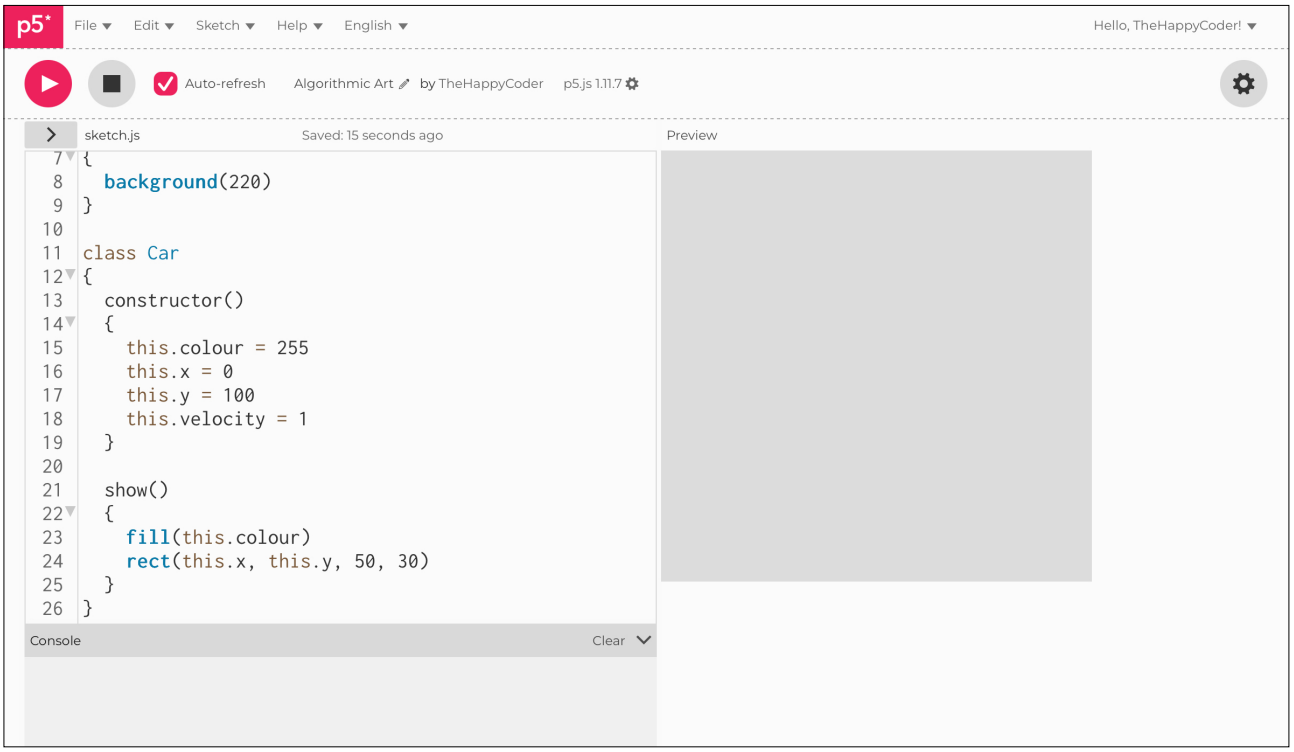
Notes

It pulls the information from the `constructor()` function. Still nothing to see yet.

Code Explanation

`fill(this.colour)`	This will fill it with white (255).
`rect(this.x, this.y, 50, 30)`	Creates a rectangle rect(0, 100, 50, 30).

Figure D1.10



Sketch D1.11 the move() function

Next, we describe how the car is going to move with the `move()` function inside the `Car` class.

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

function draw()
{
  background(220)
}

class Car
{
  constructor()
  {
    this.colour = 255
    this.x = 0
    this.y = 100
    this.velocity = 1
  }

  show()
  {
    fill(this.colour)
    rect(this.x, this.y, 50, 30)
  }

  move()
  {
    this.x += this.velocity
    if (this.x > width)
    {
      this.x = -50
    }
  }
```

```
}
```

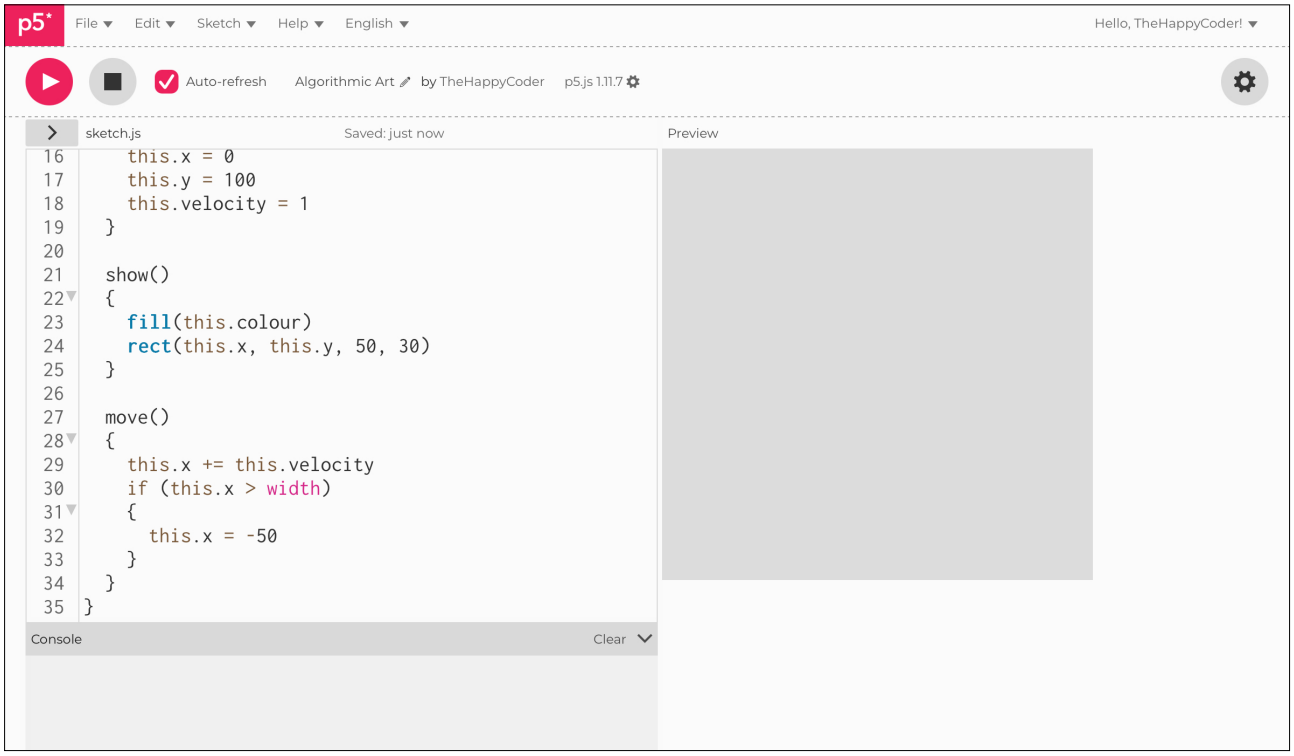
Notes

This is exactly the same as with the previous examples of a moving car. However, we created a variable for the `velocity` rather than just having a value (1). This allows us to alter it later. As before, still nothing to see.

Code Explanation

`this.x += this.velocity`	For each car, we add the velocity.
`if (this.x > width)`	If a car reaches the edge of the canvas...
`this.x = -50`	Return that car back to the left-hand edge.

Figure D1.11



Sketch D1.12 creating a car

To create a `car`, we first give this car a name. Then, in `setup()`, we create a `new Car` from the class as a template. We currently have fixed values such as `colour`, `x`, `y`, and `velocity`.

```
let car

function setup()
{
  createCanvas(400, 400)
  car = new Car()
}

function draw()
{
  background(220)
}

class Car
{
  constructor()
  {
    this.colour = 255
    this.x = 0
    this.y = 100
    this.velocity = 1
  }

  show()
  {
    fill(this.colour)
    rect(this.x, this.y, 50, 30)
  }

  move()
  {
    this.x += this.velocity
    if (this.x > width)
```

```
    {
      this.x = -50
    }
  }
}
```

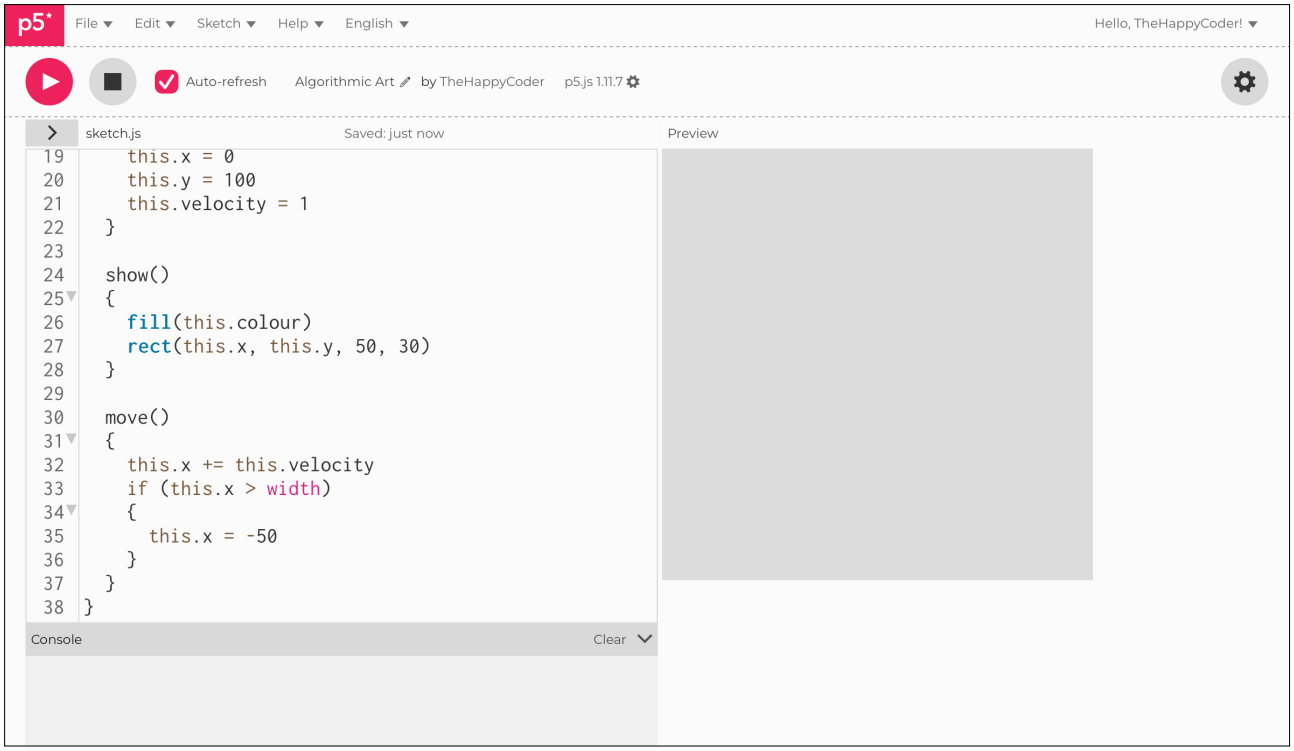
Notes

Be aware that the variable name for the car is a lowercase `c`, and the name of the class is an uppercase `C`. They both have the same name, which I admit is a little confusing, but they are totally separate entities. One is a variable name, the other is a class name. Still nothing to see here.

Code Explanation

`car = new Car()`	Creates a new car

Figure D1.12



Sketch D1.13 to see it and move it

In order to see the car, we have to call the `show()` function, and to move the car, we have to call the `move()` function, both in the `draw()` function. We ascribe these two functions to the new car we have created, called `car`.

```
let car

function setup()
{
  createCanvas(400, 400)
  car = new Car()
}

function draw()
{
  background(220)
  car.show()
  car.move()
}

class Car
{
  constructor()
  {
    this.colour = 255
    this.x = 0
    this.y = 100
    this.velocity = 1
  }

  show()
  {
    fill(this.colour)
    rect(this.x, this.y, 50, 30)
  }

  move()
  {
```

```
    this.x += this.velocity
    if (this.x > width)
    {
      this.x = -50
    }
  }
}
```

Notes

Finally, we get to see the car and watch it move.

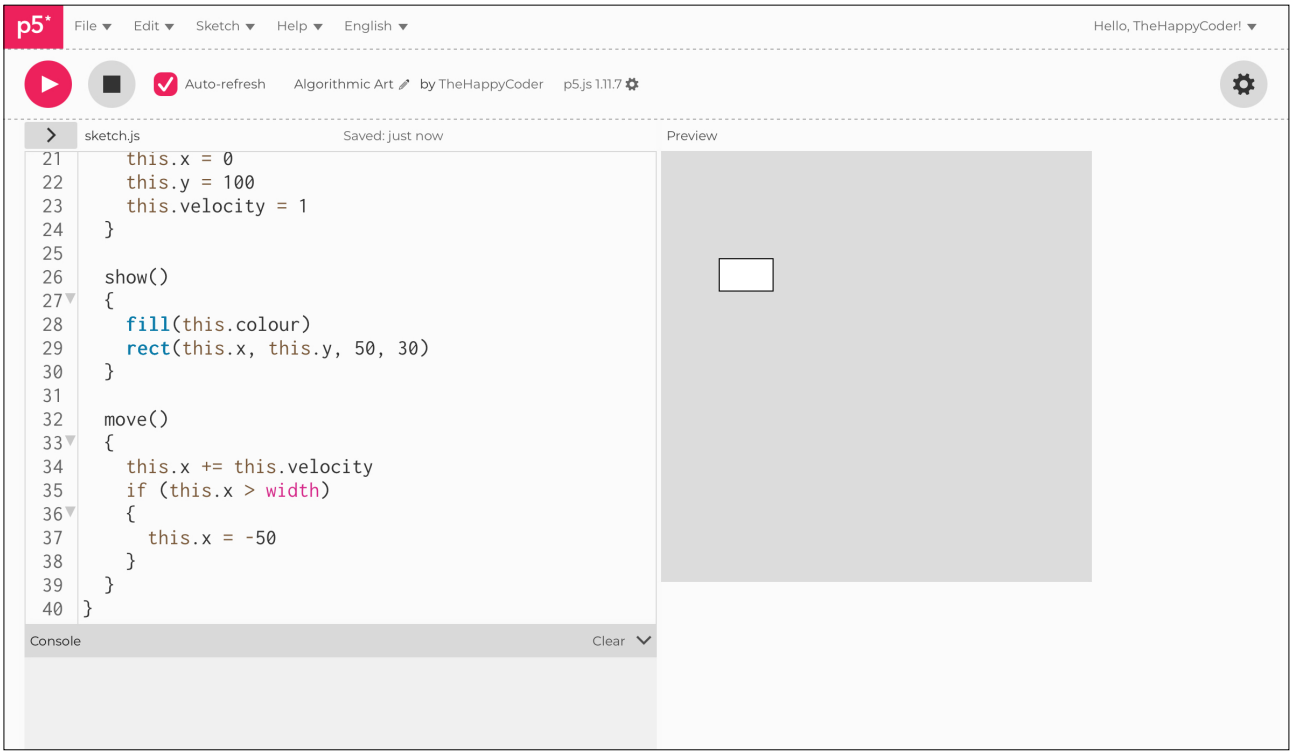
Challenges

1. Change the `colour`
2. Change the `x` value
3. Change the `y` value
4. Change the `velocity`
5. Change the name of the variable to `myCar`
6. Change the name of the `class`
7. Change the name of the `constructor()`, `show()` and `move()` functions

Code Explanation

`car.show()`	For this car, we show it according to the show() function.
`car.move()`	For this car, we move it according to the move() function.

Figure D1.13



The Power of Classes

In the following sections, we will consider what we can do with classes which makes all the trouble of creating them worthwhile. This is evident when we want hundreds of them, where each one can be created separately, independently.

Sketch D1.14 car attributes

When we create the car, we can specify its attributes rather than hard-code them in the `constructor()` function. We have given the car the same values as before. They become the arguments in the `constructor()` function: `colour`, `x`, `y`, and `velocity`. This is more like a template where you can now specify what you want.

```
let car

function setup()
{
  createCanvas(400, 400)
  car = new Car(255, 0, 100, 1)
}

function draw()
{
  background(220)
  car.show()
  car.move()
}

class Car
{
  constructor(colour, x, y, velocity)
  {
    this.colour = colour
    this.x = x
    this.y = y
    this.velocity = velocity
  }

  show()
  {
    fill(this.colour)
    rect(this.x, this.y, 50, 30)
  }

  move()
```

```
  {
    this.x += this.velocity
    if (this.x > width)
    {
      this.x = -50
    }
  }
}
```

Notes

The result is exactly the same as before because we have specified the same features of our car. The beauty of this is that we can create a second (or more) car with different features.

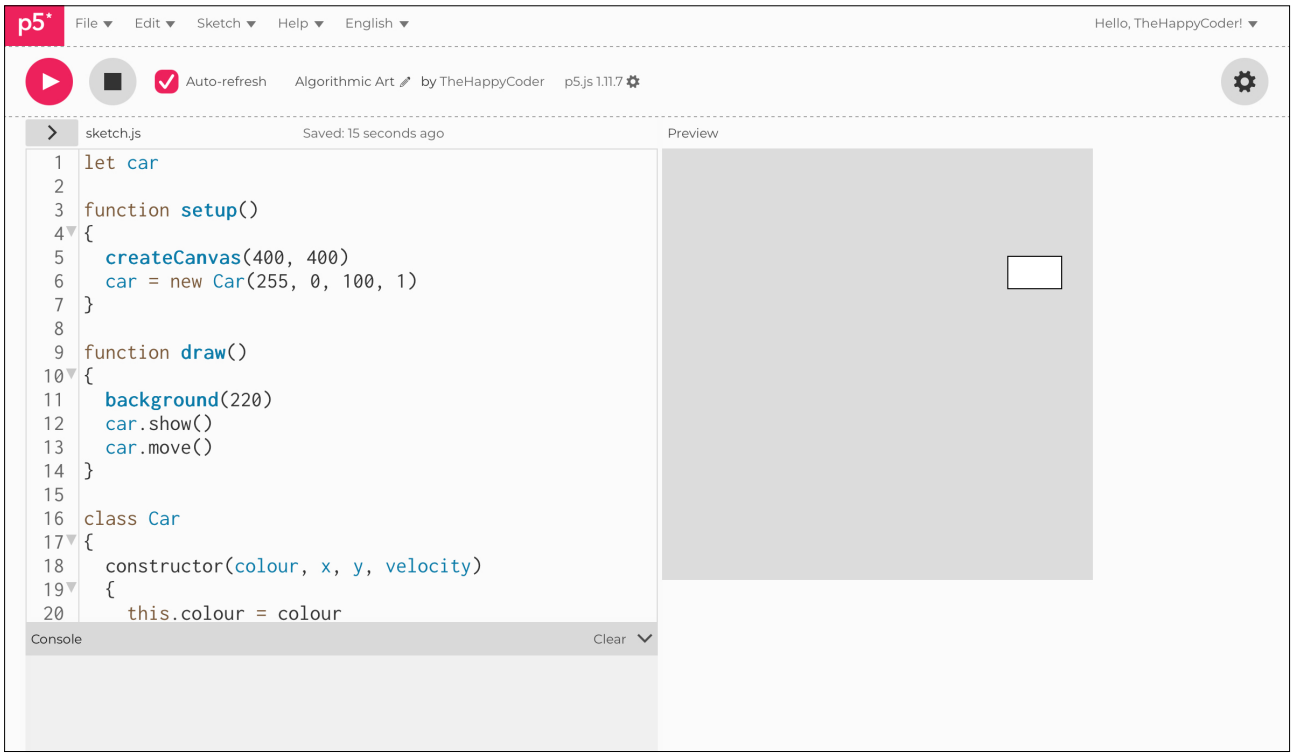
Challenge

Change the values/features of the car.

Code Explanation

`car = new Car(255, 0, 100, 1)`	We give this car some attributes.
`constructor(colour, x, y, velocity)`	The attributes are received as arguments in the constructor() function.
`this.colour = colour`	This car has the colour argument.
`this.x = x`	This car has the x position argument.
`this.y = y`	This car has the y position argument.
`this.velocity = velocity`	This car has the velocity argument.

Figure D1.14



Sketch D1.15 a second car

We add a second car and give it different features.

```
let car
let car2

function setup()
{
  createCanvas(400, 400)
  car = new Car(255, 0, 100, 1)
  car2 = new Car(55, 0, 300, 2)
}

function draw()
{
  background(220)
  car.show()
  car.move()
  car2.show()
  car2.move()
}

class Car
{
  constructor(colour, x, y, velocity)
  {
    this.colour = colour
    this.x = x
    this.y = y
    this.velocity = velocity
  }

  show()
  {
    fill(this.colour)
    rect(this.x, this.y, 50, 30)
  }
```

```
  move()
  {
    this.x += this.velocity
    if (this.x > width)
    {
      this.x = -50
    }
  }
}
```

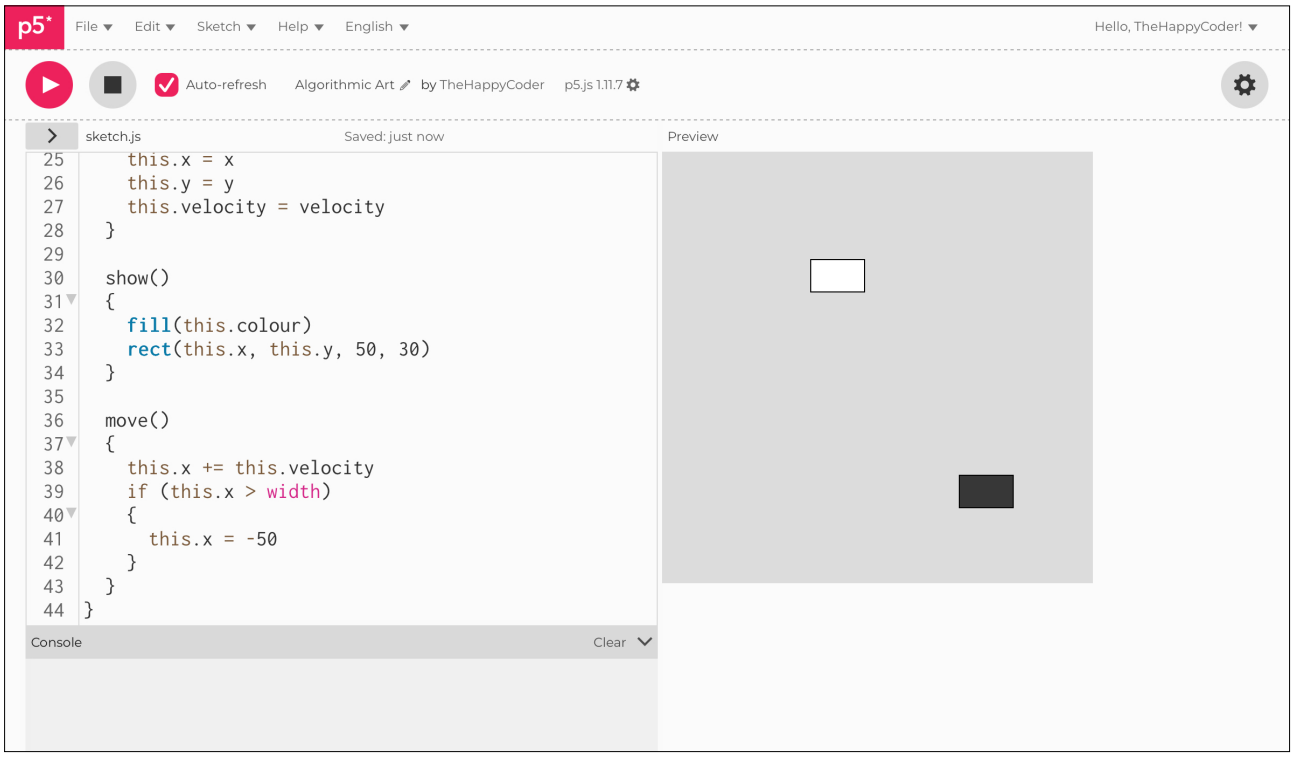
Notes

With just a few lines of code, we have created a second car. You can see the simple logic.

Challenge

Add a third car.

Figure D1.15



Sketch D1.16 lots and lots of cars

Here is a quick peek at what we could do with loops to draw lots of cars. We have covered `arrays` and `for()` loops before. We create an array of cars and cycle through them with random values for all the features (except `x`). We then cycle through the array of cars, show and move them. All this is done in the `setup()` and `draw()` functions; we don't touch the `Car` class!

```
let car = []

function setup()
{
  createCanvas(400, 400)
  for (let i = 0; i < 10; i++)
  {
    car[i] = new Car(random(255), 0, random(400), random(1, 5))
  }
}

function draw()
{
  background(220)
  for (let i = 0; i < car.length; i++)
  {
    car[i].show()
    car[i].move()
  }
}

class Car
{
  constructor(colour, x, y, velocity)
  {
    this.colour = colour
    this.x = x
    this.y = y
    this.velocity = velocity
  }

  show()
```

```
  {
    fill(this.colour)
    rect(this.x, this.y, 50, 30)
  }

  move()
  {
    this.x += this.velocity
    if (this.x > width)
    {
      this.x = -50
    }
  }
}
```

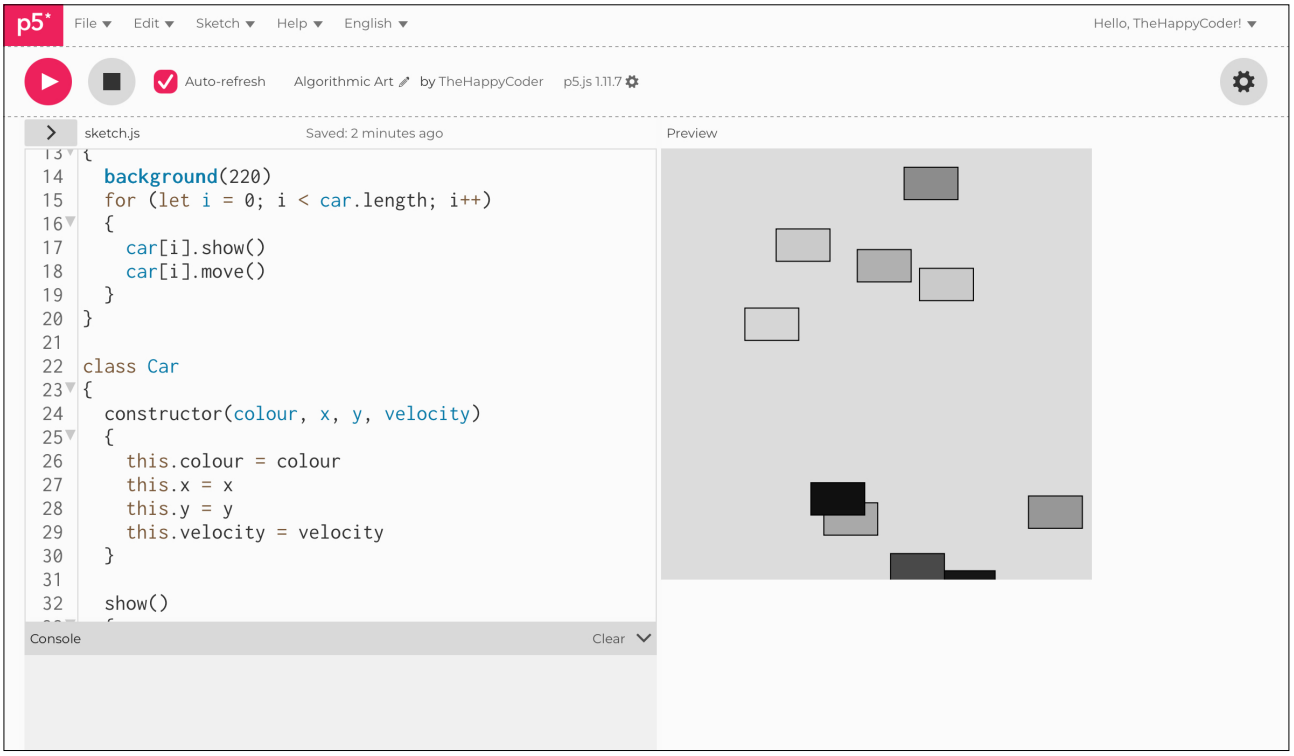
Notes

I think that is pretty elegant!

Challenge

Just have a play.

Figure D1.16



The Joy of Coding Algorithmic Art Module D Unit #2 in another class

Module D Unit #2: another class

This is another simple example of using classes to create something. The previous unit introduced classes as well as comparing them with functions and objects. Here we add a few more features to create an explosion of balls cascading or raining down. We will draw on this when we create a fireworks display.

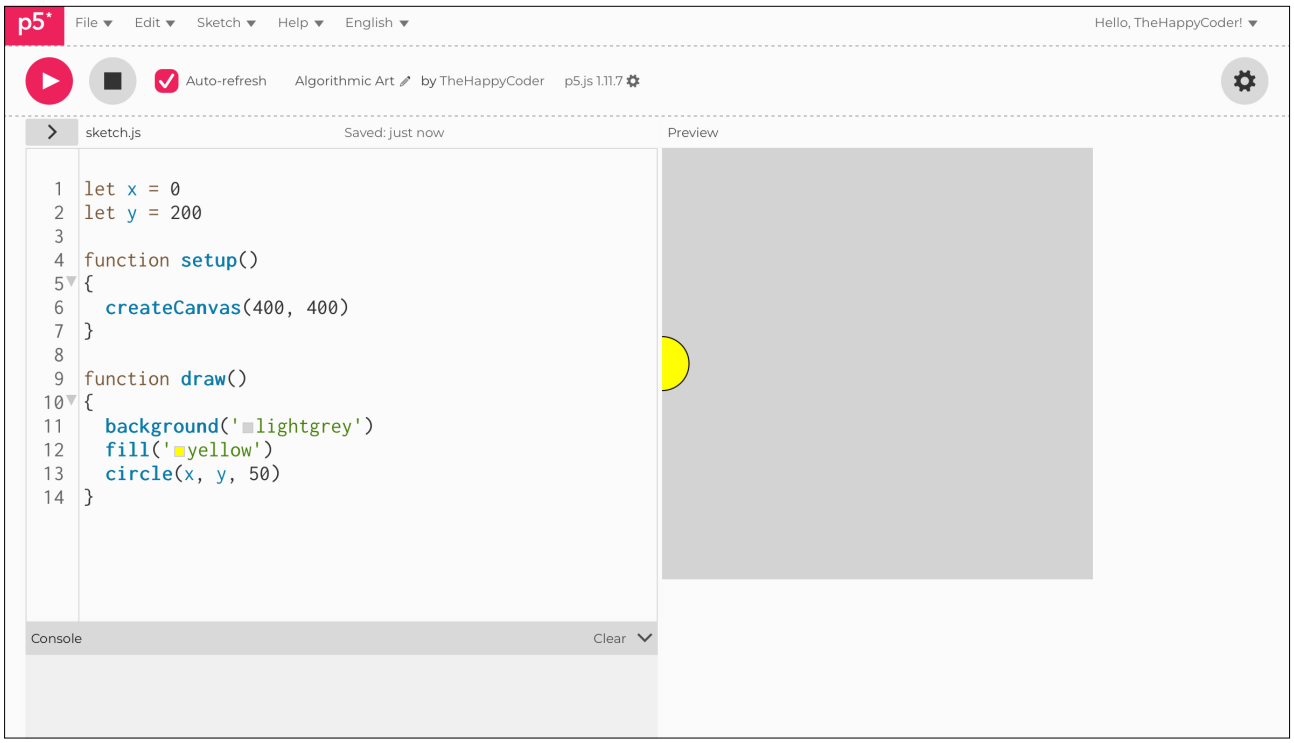
Sketch D2.1 starting again

! Starting a new sketch

A circle (ball) at (x, y) coordinates.

```
let x = 0
let y = 200

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

function draw()
{
  background('lightgrey')
  fill('yellow')
  circle(x, y, 50)
}
```

Figure D2.1


p5*
File
Edit
Sketch
Help
English
Hello, TheHappyCoder!
Auto-refresh
Algorithmic Art
by TheHappyCoder
p5.js 1.11.7
sketch.js
Saved: just now
Preview
let x = 0
let y = 200
function setup()
{
createCanvas(400, 400)
}
function draw()
{
background('lightgrey')
fill('yellow')
circle(x, y, 50)
}
Console
Clear

Sketch D2.2 a single ball

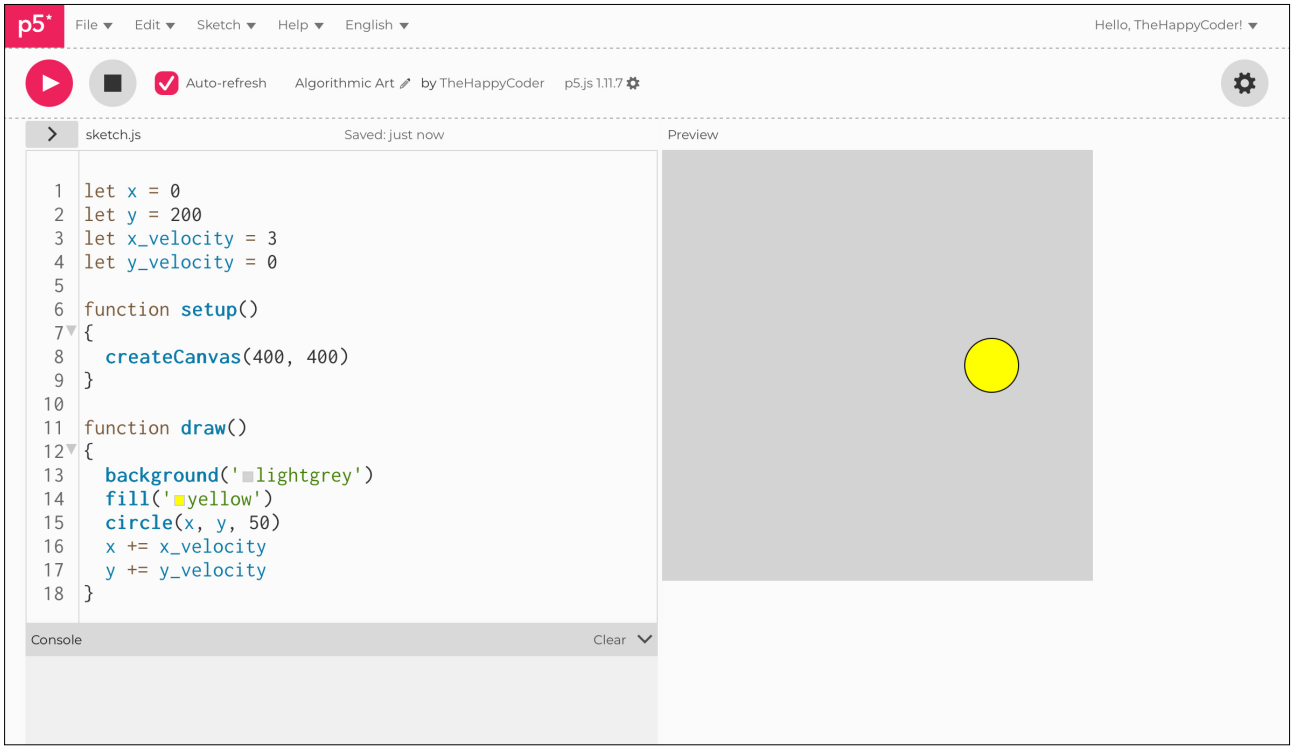
Now a ball moving across the canvas with a x velocity of 3 and a y velocity of 0.

```
let x = 0
let y = 200
let x_velocity = 3
let y_velocity = 0

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

function draw()
{
  background('lightgrey')
  fill('yellow')
  circle(x, y, 50)
  x += x_velocity
  y += y_velocity
}
```

Notes

Another naming convention is to use an underscore if splitting workday and letters in a variable name.

Figure D2.2



Sketch D2.3 adding a bit of gravity

We give the y_velocity a bit of movement upwards (hence the negative value), but by adding a bit of gravity, it causes the ball to fall downwards to the ground after a while.

```
let x = 0
let y = 200
let x_velocity = 3
let y_velocity = -3
let gravity = 0.07

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

function draw()
{
  background('lightgrey')
  fill('yellow')
  circle(x, y, 50)
  x += x_velocity
  y += y_velocity
  y_velocity += gravity
}
```

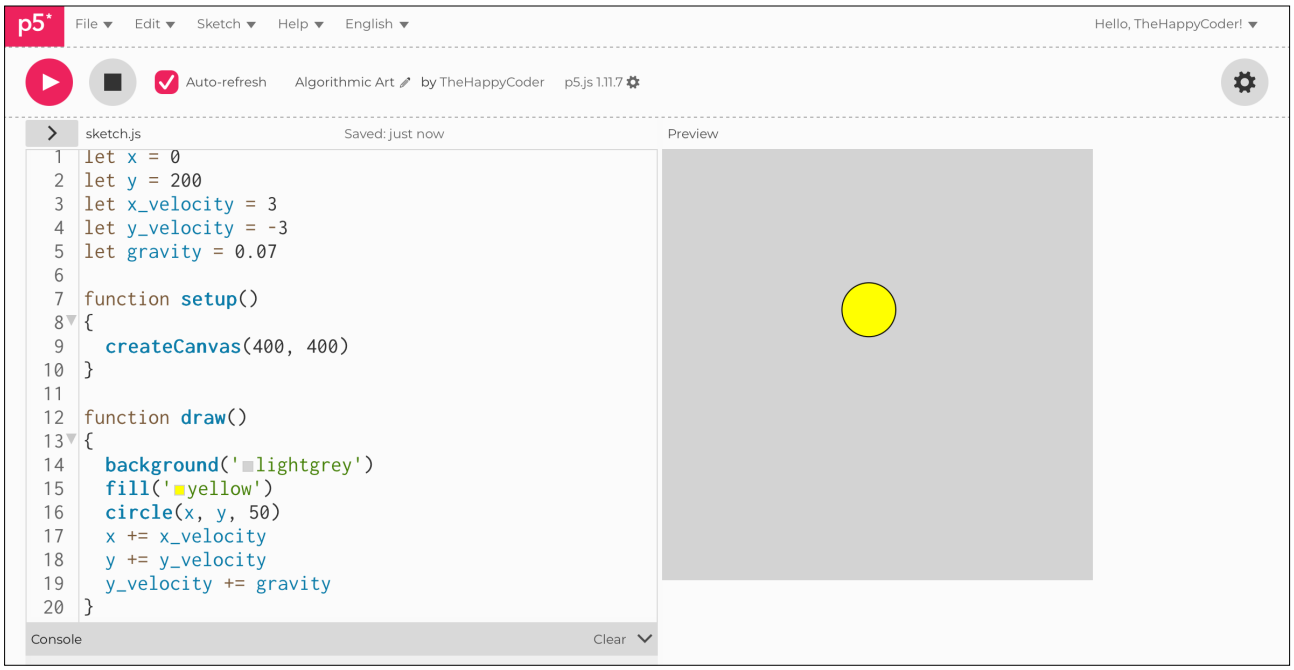
Notes

The ball goes up briefly and then starts to fall downwards once the velocity becomes positive.

Challenges

1. Try different `y` velocities and `gravity` values.
2. Wind could be a variable in the `x` direction.

Figure D2.3



Sketch D2.4 creating a ball class

We have just built the basic structure for the class. We have our `constructor()` function; we will also need a `show()` function and a `move()` function. I want to emphasise now that the names of these functions are made up; you can call each one of them whatever names you want. It is convention to have a constructor function, but the others can have other names and often do.

```
let x = 0
let y = 200
let x_velocity = 3
let y_velocity = -3
let gravity = 0.07

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

function draw()
{
  background('lightgrey')
  fill('yellow')
  circle(x, y, 50)
  x += x_velocity
  y += y_velocity
  y_velocity += gravity
}
```

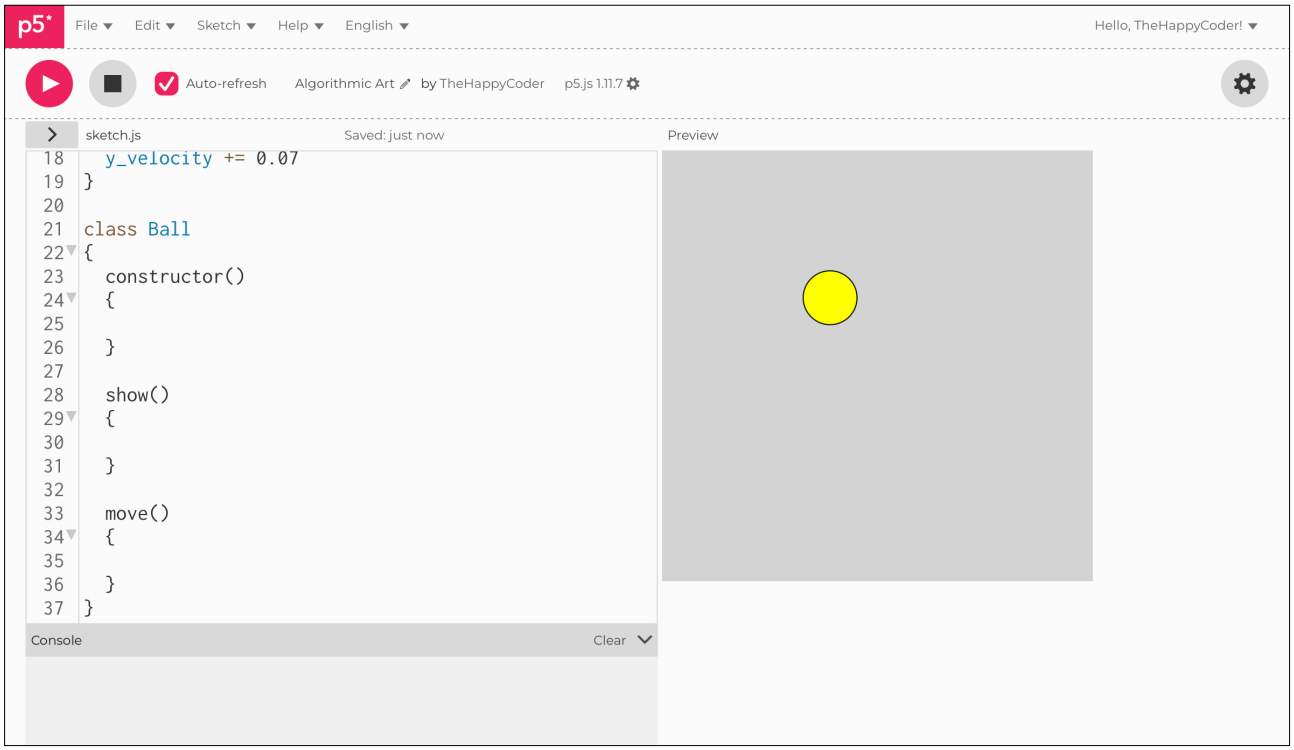
```
class Ball
{
  constructor()
  {

  }

  show()
  {

  }
```

```
  move()
  {

  }
}
```

Notes

Nothing new will happen.

Figure D2.4



Sketch D2.5 the constructor() function

! Remove the commented lines of code.

First, we will fill in the `constructor()` function using the variables highlighted. At the moment, you will get an error message if you try to run it now.

```
// let x = 0
// let y = 200
// let x_velocity = 3
// let y_velocity = -3
// let gravity = 0.07

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

function draw()
{
  background('lightgrey')
  fill('yellow')
  circle(x, y, 50)
  x += x_velocity
  y += y_velocity
  y_velocity += gravity
}

class Ball
{
  constructor()
  {
    this.x = 0
    this.y = 200
    this.x_velocity = 3
    this.y_velocity = -3
    this.gravity = 0.07
  }
```

```
  show()
  {

  }

  move()
  {

  }
}
```

Notes

The `this._____` means it is referring to a specific ball, not necessarily to all of them. You should get an error message re: the `x` variable.

Sketch D2.6 the show() function

! Remove commented out lines of code.

We can now fill in the `show()` function.

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

function draw()
{
  background('lightgrey')
  // fill('yellow')
  // circle(x, y, 50)
  x += x_velocity
  y += y_velocity
  y_velocity += gravity
}

class Ball
{
  constructor()
  {
    this.x = 0
    this.y = 200
    this.x_velocity = 3
    this.y_velocity = -3
    this.gravity = 0.07
  }

  show()
  {
    fill('yellow')
    circle(this.x, this.y, 50)
  }

  move()
```

```
  {


  }
}
```

Notes

The `x` and `y` coordinates become `this.x` and `this.y` for each ball we create. Still an error message, but don't worry.

Sketch D2.7 the move() function

! Remove the commented lines of code.

And finally, the `move()` function.

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

function draw()
{
  background('lightgrey')
  // x += x_velocity
  // y += y_velocity
  // y_velocity += gravity
}

class Ball
{
  constructor()
  {
    this.x = 0
    this.y = 200
    this.x_velocity = 3
    this.y_velocity = -3
    this.gravity = 0.07
  }

  show()
  {
    fill('yellow')
    circle(this.x, this.y, 50)
  }

  move()
  {
    this.x += this.x_velocity
```

```
    this.y += this.y_velocity
    this.y_velocity += this.gravity
  }
}
```

Notes

You shouldn't get an error message now, but neither will you see anything. We need to create the ball next and call those functions.

Sketch D2.8 can we have our ball back, please

We have finally got our ball back, phew!

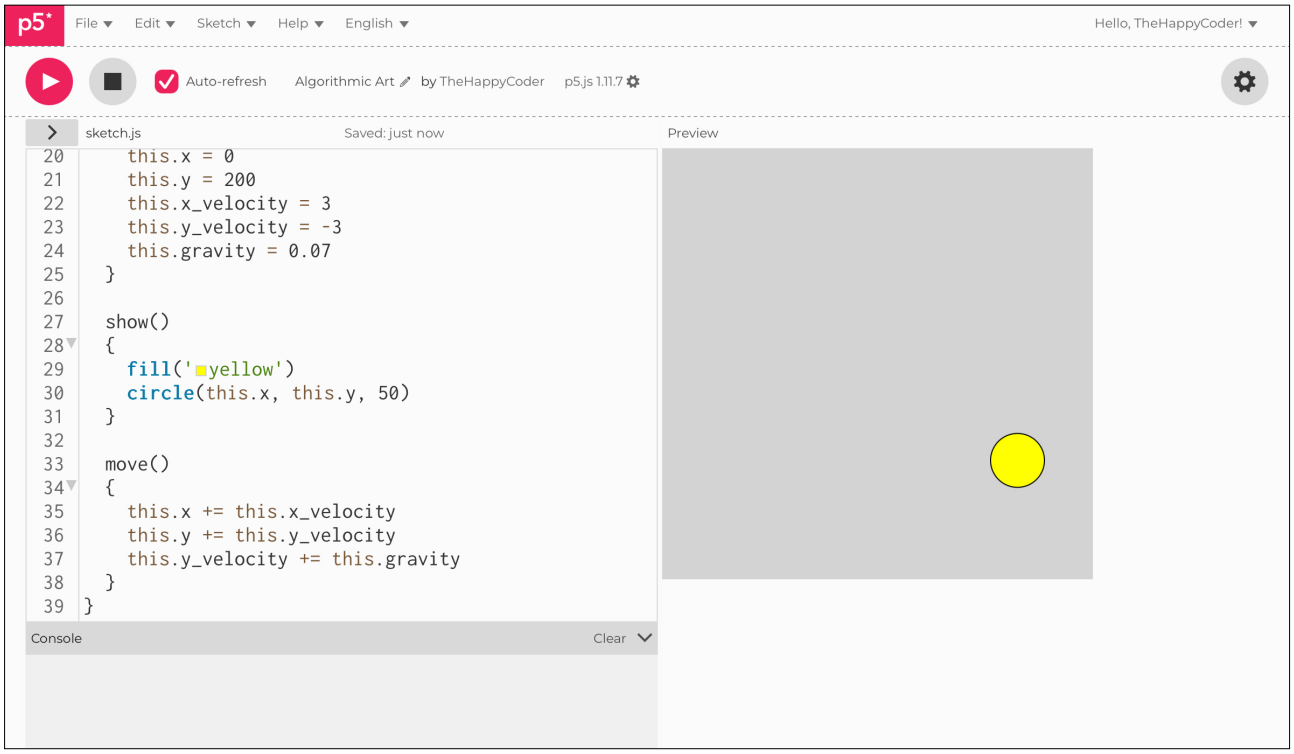
```
let ball

function setup()
{
  createCanvas(400, 400)
  ball = new Ball()
}

function draw()
{
  background('lightgrey')
  ball.show()
  ball.move()
}

class Ball
{
  constructor()
  {
    this.x = 0
    this.y = 200
    this.x_velocity = 3
    this.y_velocity = -3
    this.gravity = 0.07
  }

  show()
  {
    fill('yellow')
    circle(this.x, this.y, 50)
  }

  move()
  {
    this.x += this.x_velocity
```

```
    this.y += this.y_velocity
    this.y_velocity += this.gravity
  }
}
```

Notes

Everything should now be as before if all is well.

Figure D2.8



Sketch D2.9 many balls

! Remove and replace `ball = new Ball()` with the `for()` loop.

We can have more than one ball, so let's have `20` and for that we create an array to store them by pushing each new ball into that array, warning you that you will get an error message.

```
let ball
let balls = []
let number = 20

function setup()
{
  createCanvas(400, 400)
  for (let i = 0; i < number; i++)
  {
    balls.push(new Ball())
  }
}

function draw()
{
  background('lightgrey')
  ball.show()
  ball.move()
}

class Ball
{
  constructor()
  {
    this.x = 0
    this.y = 200
    this.x_velocity = 3
    this.y_velocity = -3
    this.gravity = 0.07
  }

  show()
```

```
  {
    fill('yellow')
    circle(this.x, this.y, 50)
  }

  move()
  {
    this.x += this.x_velocity
    this.y += this.y_velocity
    this.y_velocity += this.gravity
  }
}
```

Notes

We have broken the sketch, but have no fear, we will fix that. Error message!

Sketch D2.10 pulling the balls out

We need to pull each of the balls from the array and `show()` and `move()` each one. We use the `for...of` loop.

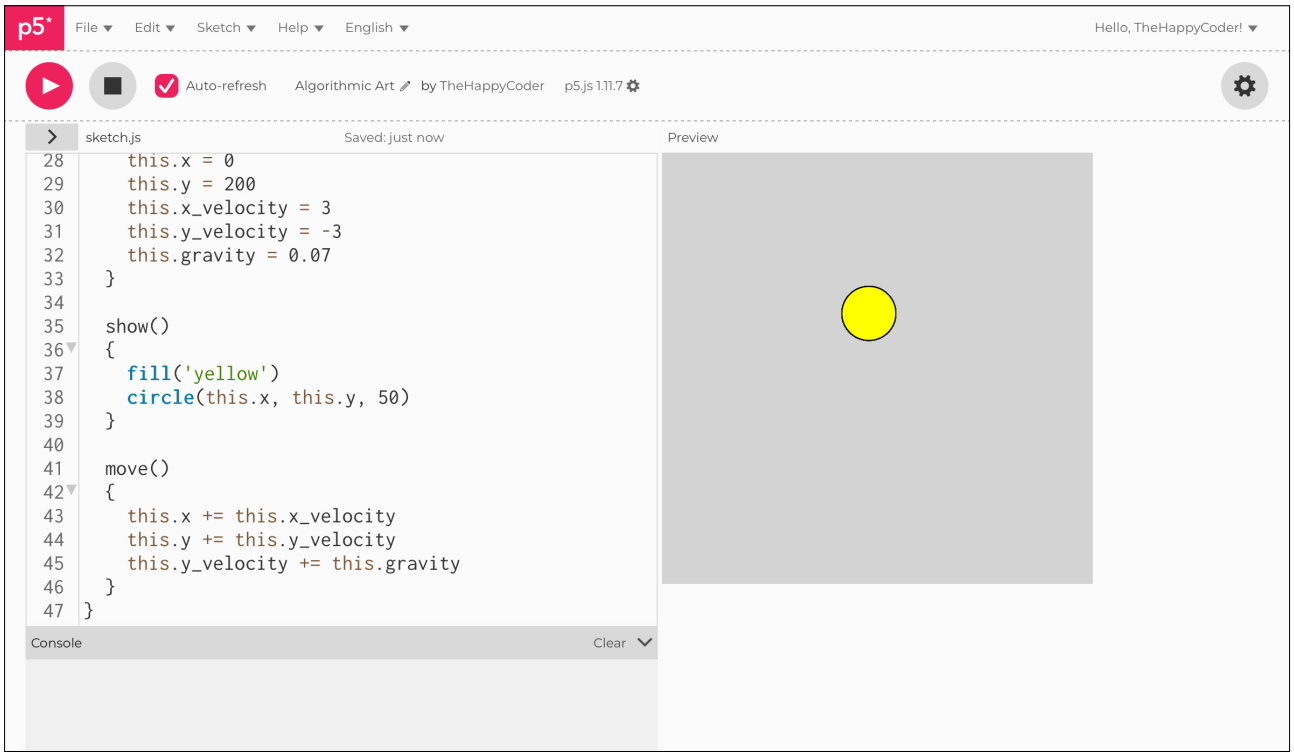
```
let ball
let balls = []
let number = 20

function setup()
{
  createCanvas(400, 400)
  for (let i = 0; i < number; i++)
  {
    balls.push(new Ball())
  }
}

function draw()
{
  background('lightgrey')
  for (ball of balls)
  {
    ball.show()
    ball.move()
  }
}

class Ball
{
  constructor()
  {
    this.x = 0
    this.y = 200
    this.x_velocity = 3
    this.y_velocity = -3
    this.gravity = 0.07
  }
```

```
  show()
  {
    fill('yellow')
    circle(this.x, this.y, 50)
  }

  move()
  {
    this.x += this.x_velocity
    this.y += this.y_velocity
    this.y_velocity += this.gravity
  }
}
```

Notes

We have fixed the sketch, but there were supposed to be 20 balls, not just one! To see them, we need to mix them up a bit; they are occupying the same space.

Figure D2.10



Sketch D2.11 random velocity

To mix it up, we can introduce some randomness to their velocities for both x and y.

```
let ball
let balls = []
let number = 20

function setup()
{
  createCanvas(400, 400)
  for (let i = 0; i < number; i++)
  {
    balls.push(new Ball())
  }
}

function draw()
{
  background('lightgrey')
  for (ball of balls)
  {
    ball.show()
    ball.move()
  }
}

class Ball
{
  constructor()
  {
    this.x = 0
    this.y = 200
    this.x_velocity = 3 * random(1, 2)
    this.y_velocity = -3 * random(-1, 1)
    this.gravity = 0.07
  }

  show()
```

```
  {
    fill('yellow')
    circle(this.x, this.y, 50)
  }

  move()
  {
    this.x += this.x_velocity
    this.y += this.y_velocity
    this.y_velocity += this.gravity
  }
}
```

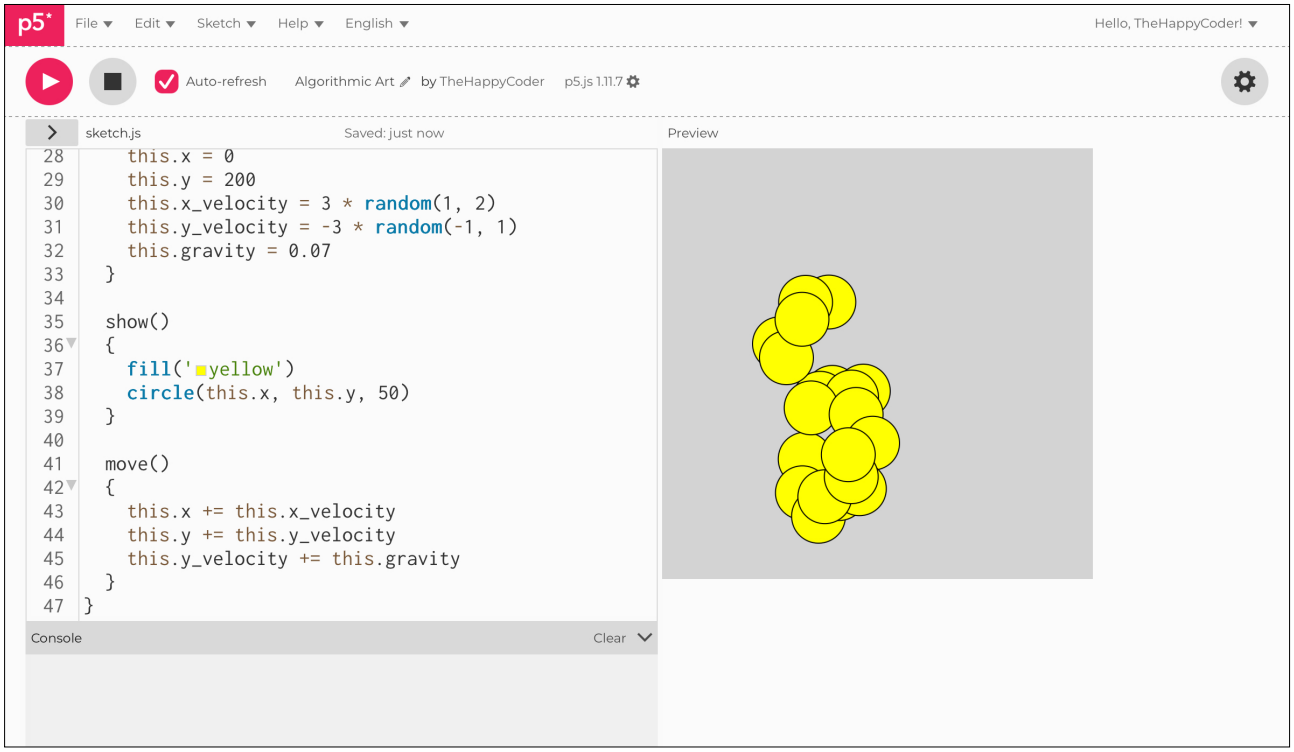
Notes

They can now wander off in different directions.

Challenge

Try other random values.

Figure D2.11



Sketch D2.12 refactoring

A bit of refactoring for a slightly different effect.

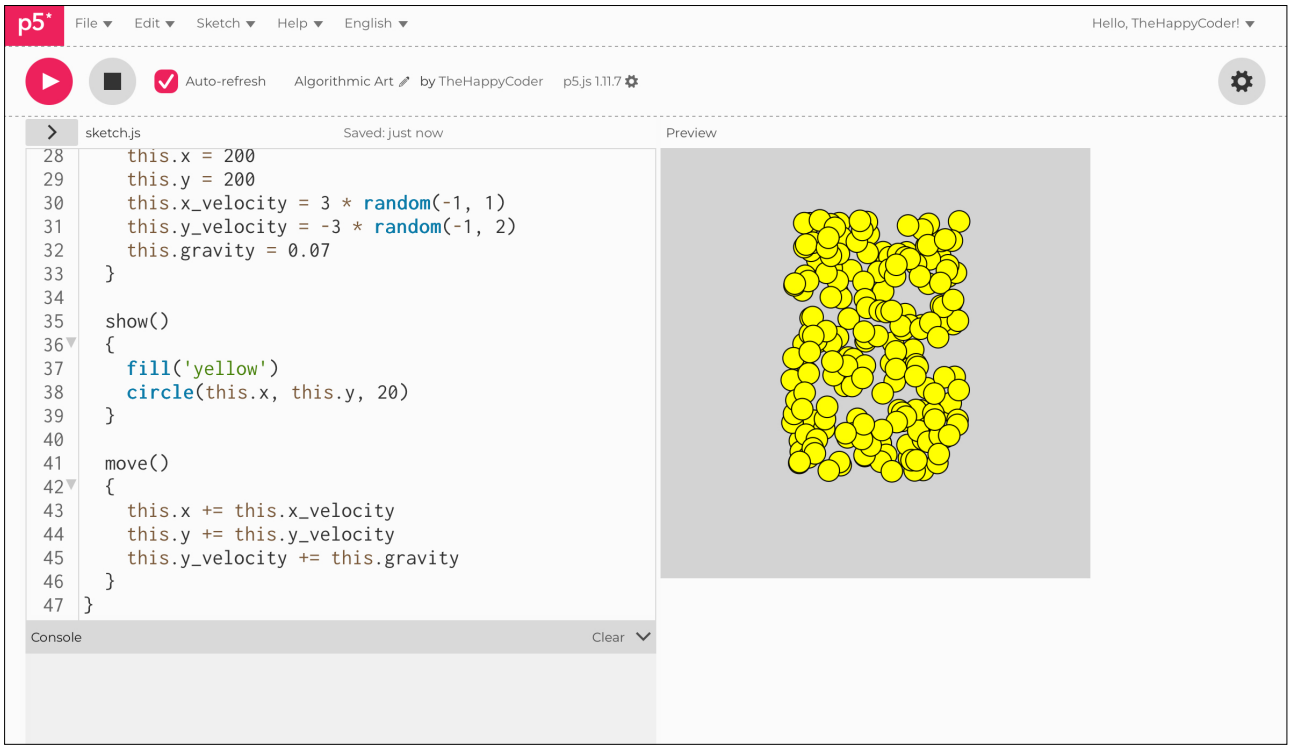
```
let ball
let balls = []
let number = 200

function setup()
{
  createCanvas(400, 400)
  for (let i = 0; i < number; i++)
  {
    balls.push(new Ball())
  }
}

function draw()
{
  background('lightgrey')
  for (ball of balls)
  {
    ball.show()
    ball.move()
  }
}

class Ball
{
  constructor()
  {
    this.x = 200
    this.y = 200
    this.x_velocity = 3 * random(-1, 1)
    this.y_velocity = -3 * random(-1, 2)
    this.gravity = 0.07
  }

  show()
```

```
  {
    fill('yellow')
    circle(this.x, this.y, 20)
  }

  move()
  {
    this.x += this.x_velocity
    this.y += this.y_velocity
    this.y_velocity += this.gravity
  }
}
```

Notes

We will have 200 smaller balls emitting from the centre; you could have something like a firework-type display.

Figure D2.12



The Joy of Coding Algorithmic Art

Module D Unit #3 array of objects

Module D Unit #3: array of objects

In this unit, we will mash together arrays and classes with a dash of mouse functions.

Recap: Arrays are extremely useful and powerful in so many ways. An array is a list of elements inside square brackets separated by a comma. The numbering of each element starts at zero; they are called the index number. Let's create an array called num and put some numbers in it.

```
let num = [42, 56, 63, 79, 88]
```

There are five numbers (elements) in the array. They are, in order: the first one is 42, the second one is 56, the third one is 63, the fourth one is 79, and the fifth one is 88.

However, to reference the first element (42), we say it is at index 0. The second element (56) is at index 1, the third (63) is index 2, the fourth (79) is index 3, and the fifth (88) is at index 4. This may seem a bit silly, but it is the way it is, so you will need to get used to it.

Sketch D3.1 bubbles

! Create a new sketch

We are going to make an array of objects. We start with an empty array called `bubbles`.

```
let bubbles = []


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

function draw()
{
  background(220)
}
```

Sketch D3.2 classy bubbles

We are going to put bubble objects in that array, but first we need to create a Bubble class to create some bubbles.

```
let bubbles = []

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

function draw()
{
  background(220)
}
```

```
class Bubble
{
  constructor()
  {

  }

  move()
  {

  }

  show()
  {

  }
}
```

Notes

Nothing to see here.

Sketch D3.3 time for an argument

We need three arguments for the bubble: the x, y (coordinates), and r (radius). The radius will be randomly generated. The x and y positions will be created with a click or drag of the mouse.

```
let bubbles = []

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

function draw()
{
  background(220)
}

class Bubble
{
  constructor(x, y, r)
  {
    this.x = x
    this.y = y
    this.r = r
  }

  move()
  {

  }

  show()
  {
    circle(this.x, this.y, this.r)
  }
}
```

Notes

We are not moving anything because we haven't drawn a bubble. So there is nothing to see just yet; all will be revealed soon.

Sketch D3.4 drawing the bubbles

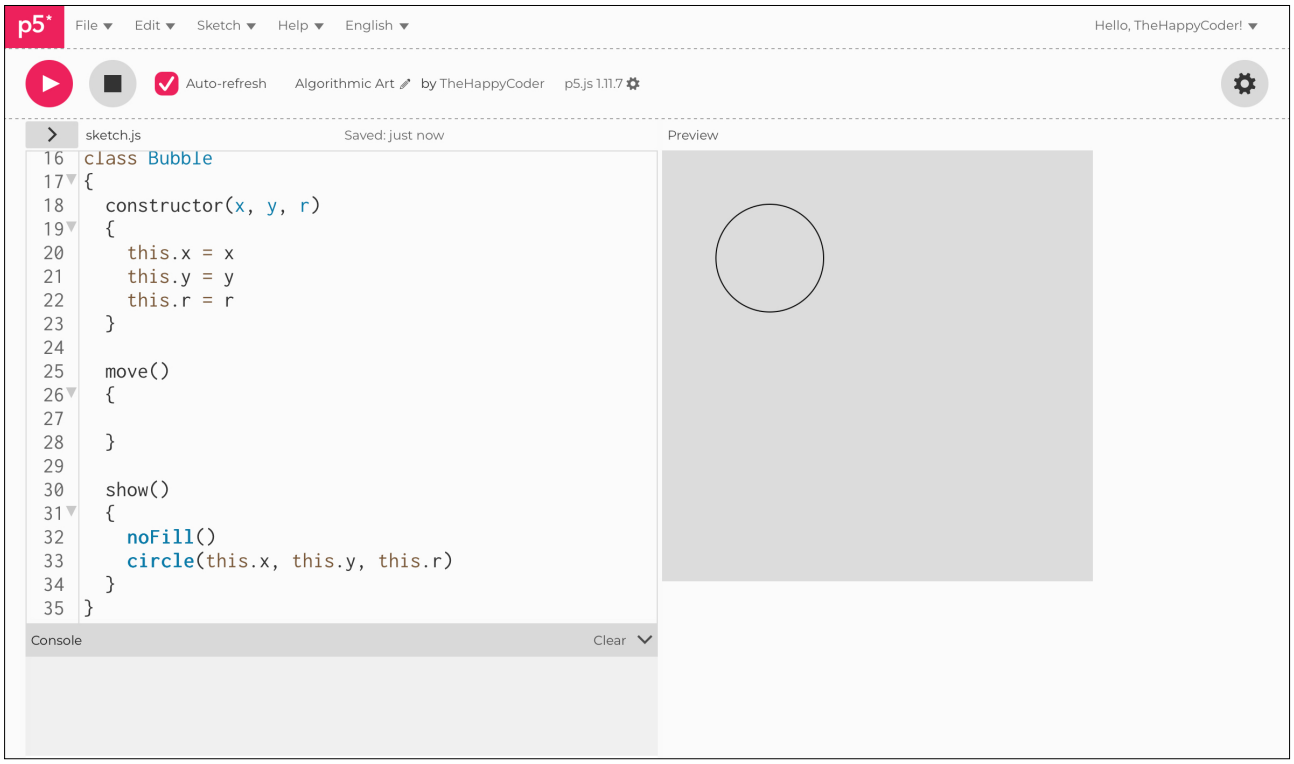
Let us draw one bubble to start with.

```
let bubbles = []
let bubble

function setup()
{
  createCanvas(400, 400)
  bubble = new Bubble(100, 100, 100)
}

function draw()
{
  background(220)
  bubble.show()
}

class Bubble
{
  constructor(x, y, r)
  {
    this.x = x
    this.y = y
    this.r = r
  }

  move()
  {

  }

  show()
  {
    noFill()
    circle(this.x, this.y, this.r)
  }
}
```

Notes

We finally have something to see, our first bubble.

Figure D3.4



Sketch D3.5 random motion

Let's move it around in a random manner.

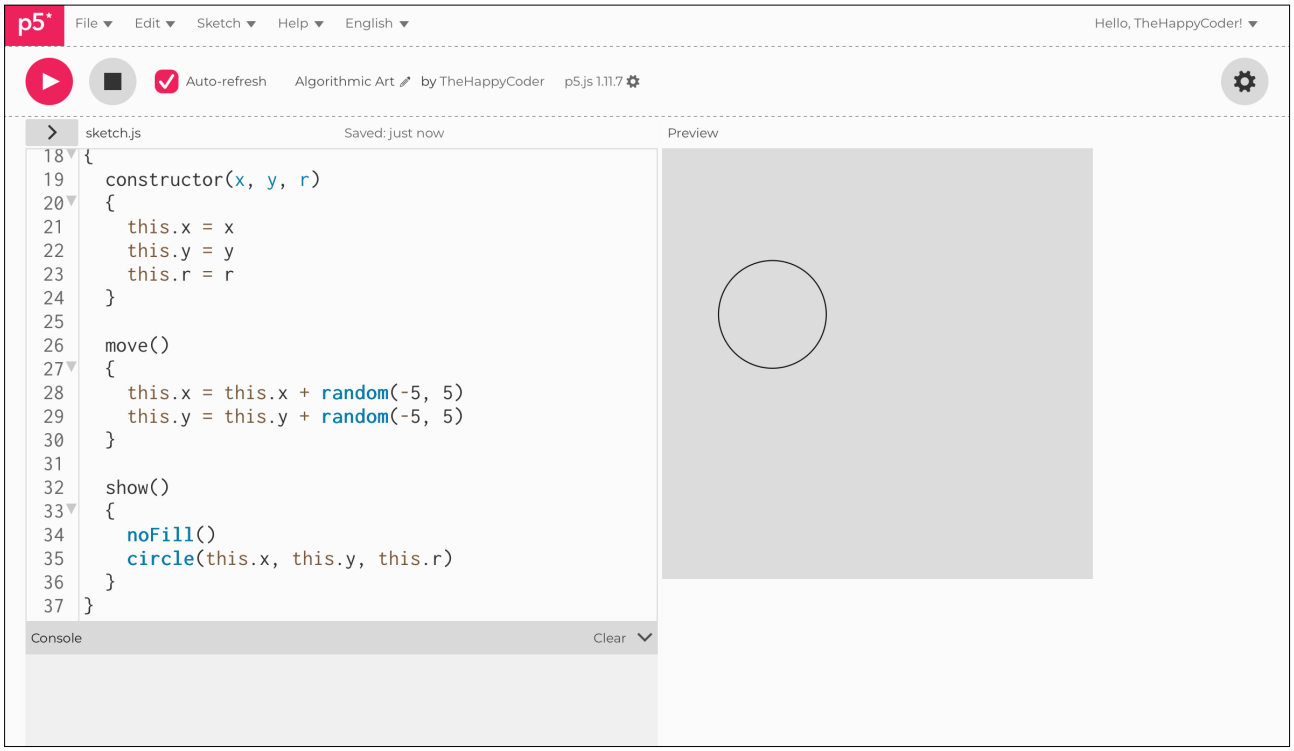
```
let bubbles = []
let bubble

function setup()
{
  createCanvas(400, 400)
  bubble = new Bubble(100, 100, 100)
}

function draw()
{
  background(220)
  bubble.show()
  bubble.move()
}

class Bubble
{
  constructor(x, y, r)
  {
    this.x = x
    this.y = y
    this.r = r
  }

  move()
  {
    this.x = this.x + random(-5, 5)
    this.y = this.y + random(-5, 5)
  }

  show()
  {
    noFill()
    circle(this.x, this.y, this.r)
```

```
  }
}
```

Notes

What you should see is a circle wobbling around the canvas (or wandering off it eventually).

Figure D3.5



Sketch D3.6 many bubbles

We can create many of them with a nice `for()` loop and put them into the `bubbles` array. Then we have another `for()` loop to pull them out. Here we are just creating and drawing one bubble for the moment.

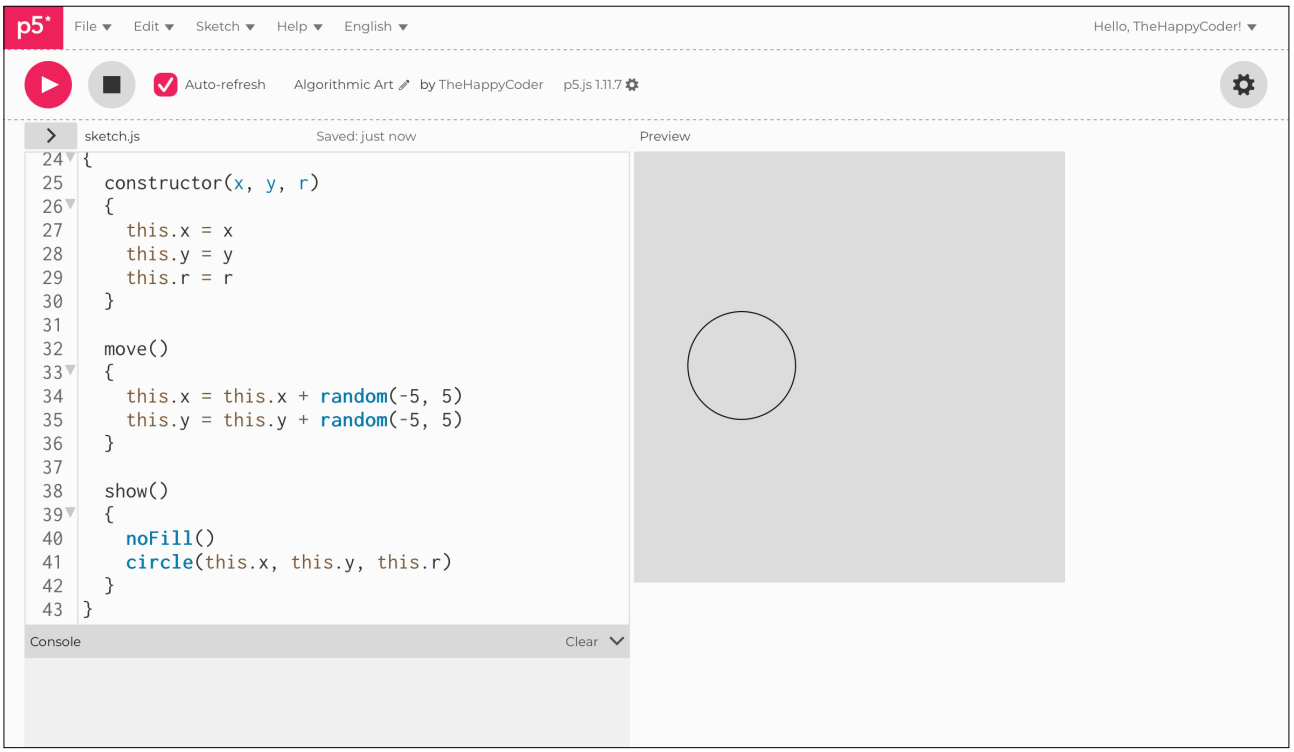
```
let bubbles = []
let bubble

function setup()
{
  createCanvas(400, 400)
  for (let i = 0; i < 1; i++)
  {
    bubbles[i] = new Bubble(100, 100, 100)
  }
}

function draw()
{
  background(220)
  for (let i = 0; i < 1; i++)
  {
    bubbles[i].show()
    bubbles[i].move()
  }
}

class Bubble
{
  constructor(x, y, r)
  {
    this.x = x
    this.y = y
    this.r = r
  }

  move()
  {
```

```
    this.x = this.x + random(-5, 5)
    this.y = this.y + random(-5, 5)
  }

  show()
  {
    noFill()
    circle(this.x, this.y, this.r)
  }
}
```

Notes

We have not really changed anything; you should have the same effect.

Figure D3.6



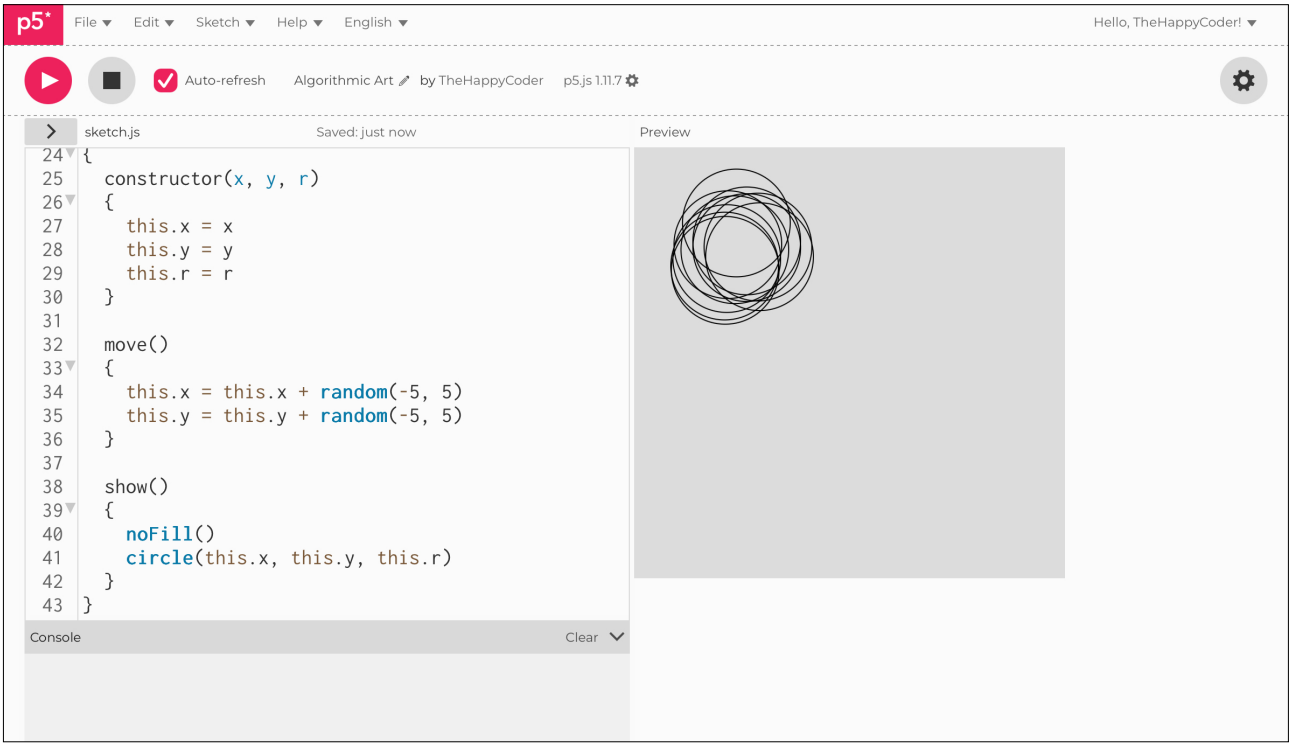
Sketch D3.7 more bubbles

We only drew one `bubble` above. There is only one `bubble` in the array of `bubbles`. We can change that by increasing it from `1` to `10`. In the second `for()` loop, we can call the length of the array rather than hardcode it just in case we change the length of the array.

```
let bubbles = []
let bubble

function setup()
{
  createCanvas(400, 400)
  for (let i = 0; i < 10; i++)
  {
    bubbles[i] = new Bubble(100, 100, 100)
  }
}

function draw()
{
  background(220)
  for (let i = 0; i < bubbles.length; i++)
  {
    bubbles[i].show()
    bubbles[i].move()
  }
}

class Bubble
{
  constructor(x, y, r)
  {
    this.x = x
    this.y = y
    this.r = r
  }

  move()
```

```
  {
    this.x = this.x + random(-5, 5)
    this.y = this.y + random(-5, 5)
  }

  show()
  {
    noFill()
    circle(this.x, this.y, this.r)
  }
}
```

Figure D3.7



Sketch D3.8 spacing them out

We started each one in the same spot. Let's space them along a line using the loop. We start at 10 and multiply the index i by 40.

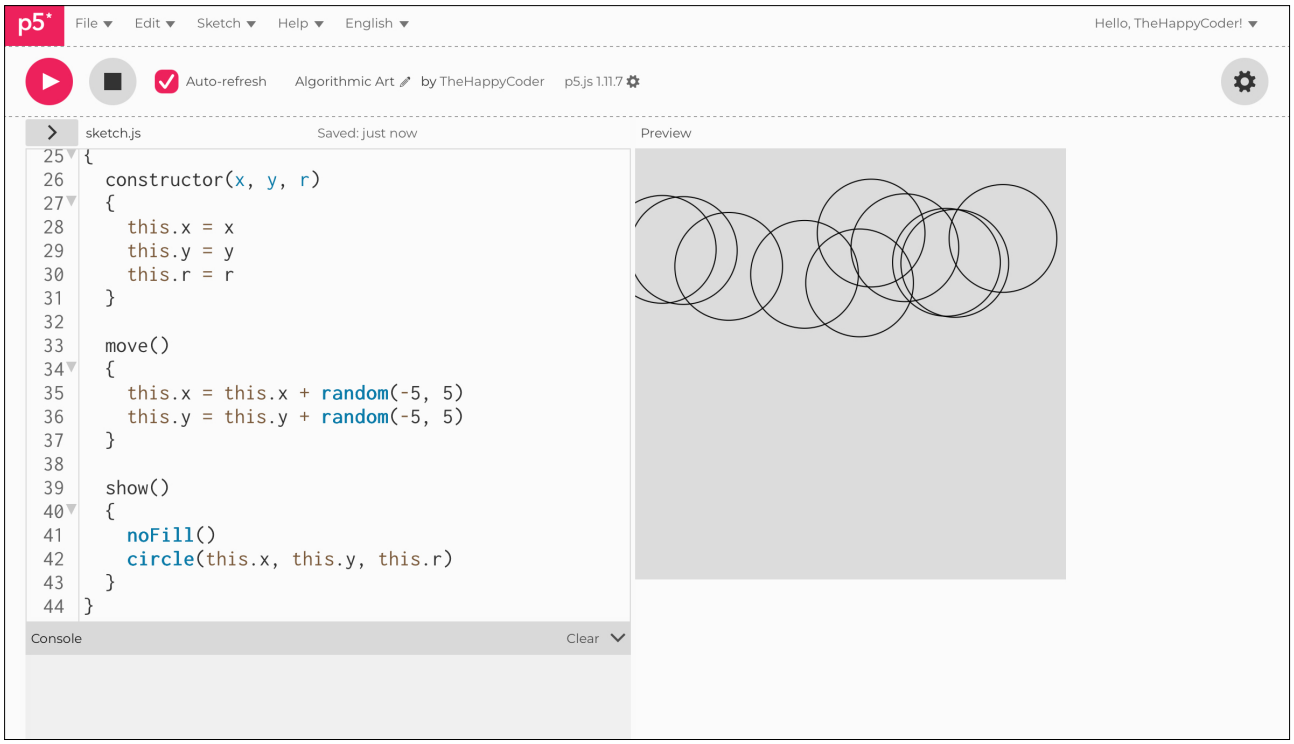
```
let bubbles = []
let bubble

function setup()
{
  createCanvas(400, 400)
  for (let i = 0; i < 10; i++)
  {
    let x = 10 + 40 * i
    bubbles[i] = new Bubble(x, 100, 100)
  }
}

function draw()
{
  background(220)
  for (let i = 0; i < bubbles.length; i++)
  {
    bubbles[i].show()
    bubbles[i].move()
  }
}

class Bubble
{
  constructor(x, y, r)
  {
    this.x = x
    this.y = y
    this.r = r
  }

  move()
  {
```

```
    this.x = this.x + random(-5, 5)
    this.y = this.y + random(-5, 5)
  }

  show()
  {
    noFill()
    circle(this.x, this.y, this.r)
  }
}
```

Notes

Notice we haven't touched anything in the class itself.

Figure D3.8



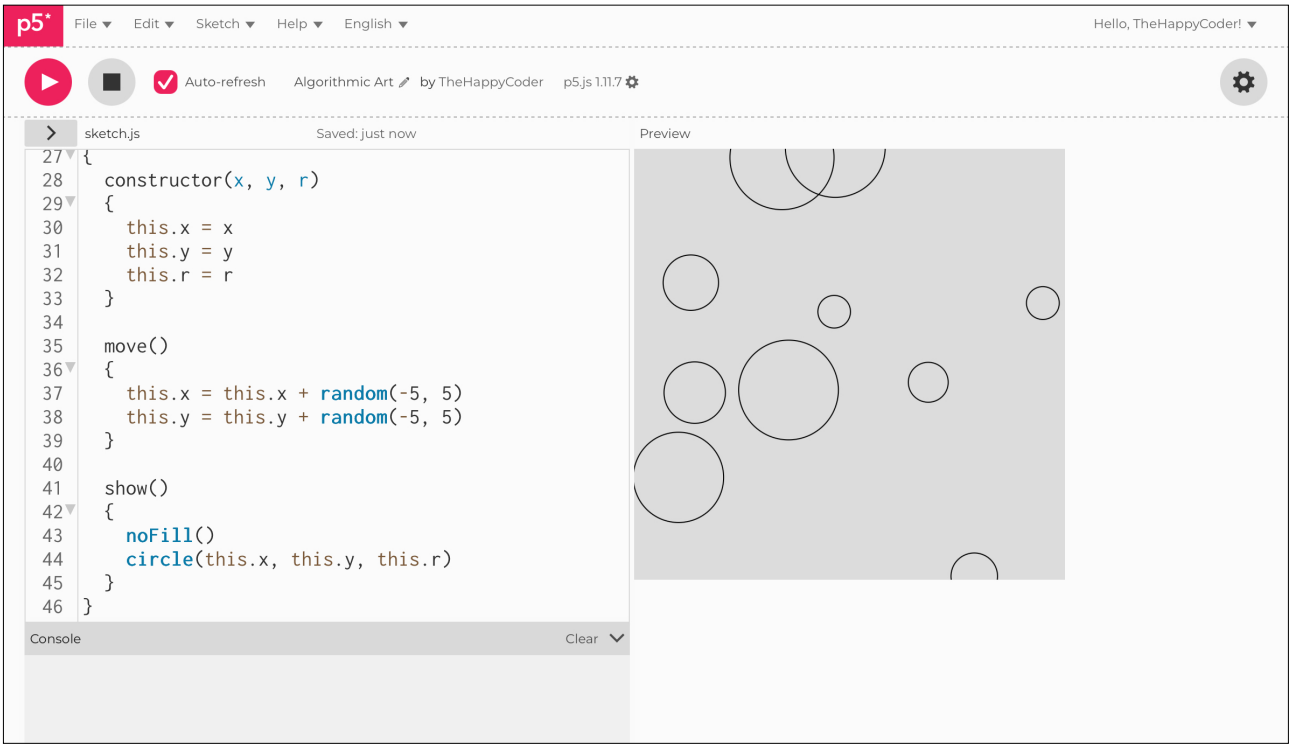
Sketch D3.9 random

Let us create a set of bubbles of random positions and sizes.

```
let bubbles = []
let bubble

function setup()
{
  createCanvas(400, 400)
  for (let i = 0; i < 10; i++)
  {
    let x = random(width)
    let y = random(height)
    let r = random(20, 100)
    bubbles[i] = new Bubble(x, y, r)
  }
}

function draw()
{
  background(220)
  for (let i = 0; i < bubbles.length; i++)
  {
    bubbles[i].show()
    bubbles[i].move()
  }
}

class Bubble
{
  constructor(x, y, r)
  {
    this.x = x
    this.y = y
    this.r = r
  }

  move()
```

```
  {
    this.x = this.x + random(-5, 5)
    this.y = this.y + random(-5, 5)
  }

  show()
  {
    noFill()
    circle(this.x, this.y, this.r)
  }
}
```

Figure D3.9



Sketch D3.10 the shimmering fog

That is all well and good, but what can we do to make it more interesting? With a few tweaks, we can do something quite slightly unexpected.

```
let bubbles = []
let bubble

function setup()
{
  createCanvas(400, 400)
  for (let i = 0; i < 1000; i++)
  {
    let x = random(width)
    let y = random(height)
    let r = random(20, 100)
    bubbles[i] = new Bubble(x, y, r)
  }
}

function draw()
{
  background(220)
  for (let i = 0; i < bubbles.length; i++)
  {
    bubbles[i].show()
    bubbles[i].move()
  }
}

class Bubble
{
  constructor(x, y, r)
  {
    this.x = x
    this.y = y
    this.r = r
  }
```

```
  move()
  {
    this.x = this.x + random(-3, 3)
    this.y = this.y + random(-3, 3)
  }

  show()
  {
    noStroke()
    fill(51, 10)
    circle(this.x, this.y, this.r)
  }
}
```

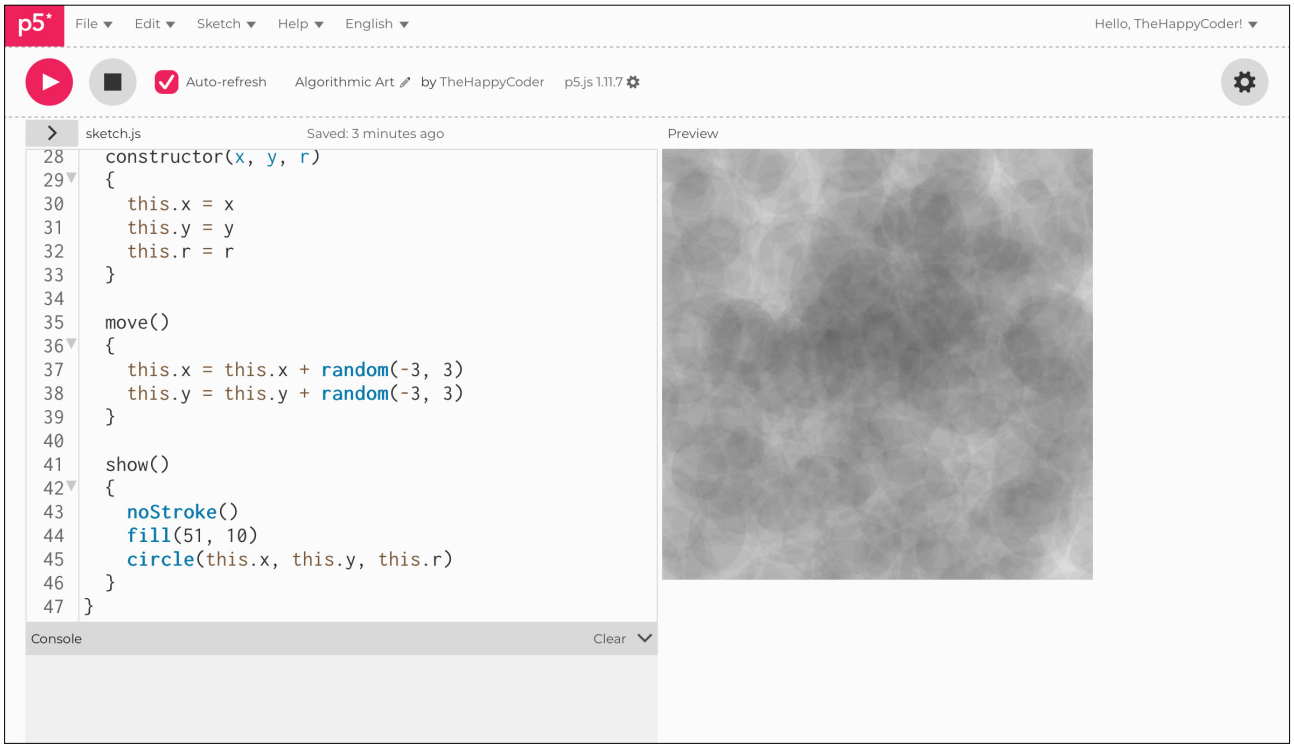
Notes

Sometimes it doesn't take much.

Challenge

Play with it.

Figure D3.10



Sketch D3.11 simpler times

Let's go back to a simpler sketch. A bit of deleting and a couple of lines of code will bring you back to a familiar starting point.

! Remove the lines of code highlighted in blue and commented out (//), and add the `noFill()` in the `show()` function.

```
let bubbles = []
let bubble

function setup()
{
  createCanvas(400, 400)
  // for (let i = 0; i < 1000; i++)
  // {
  //   let x = random(width)
  //   let y = random(height)
  //   let r = random(20, 100)
  //   bubbles = new Bubble(x, y, r)
  // }
}

function draw()
{
  background(220)
  for (let i = 0; i < bubbles.length; i++)
  {
    bubbles[i].show()
    bubbles[i].move()
  }
}

class Bubble
{
  constructor(x, y, r)
  {
    this.x = x
    this.y = y
    this.r = r
```

```
  }

  move()
  {
    this.x = this.x + random(-3, 3)
    this.y = this.y + random(-3, 3)
  }

  show()
  {
    // noStroke()
    // fill(51, 10)
    noFill()
    circle(this.x, this.y, this.r)
  }
}
```

Notes

A blank grey canvas.

Sketch D3.12 this is what you get

You should have this...

```
let bubbles = []
let bubble

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

function draw()
{
  background(220)
  for (let i = 0; i < bubbles.length; i++)
  {
    bubbles[i].show()
    bubbles[i].move()
  }
}

class Bubble
{
  constructor(x, y, r)
  {
    this.x = x
    this.y = y
    this.r = r
  }

  move()
  {
    this.x = this.x + random(-3, 3)
    this.y = this.y + random(-3, 3)
  }

  show()
  {
```

```
    noFill()
    circle(this.x, this.y, this.r)
  }
}
```

Notes

Nothing should happen! No bubbles, we want to draw bubbles where we click the mouse.

Sketch D3.13 mouse click

Creating a new bubble at the click of a mouse, we create a function called `mousePressed()` to do this.

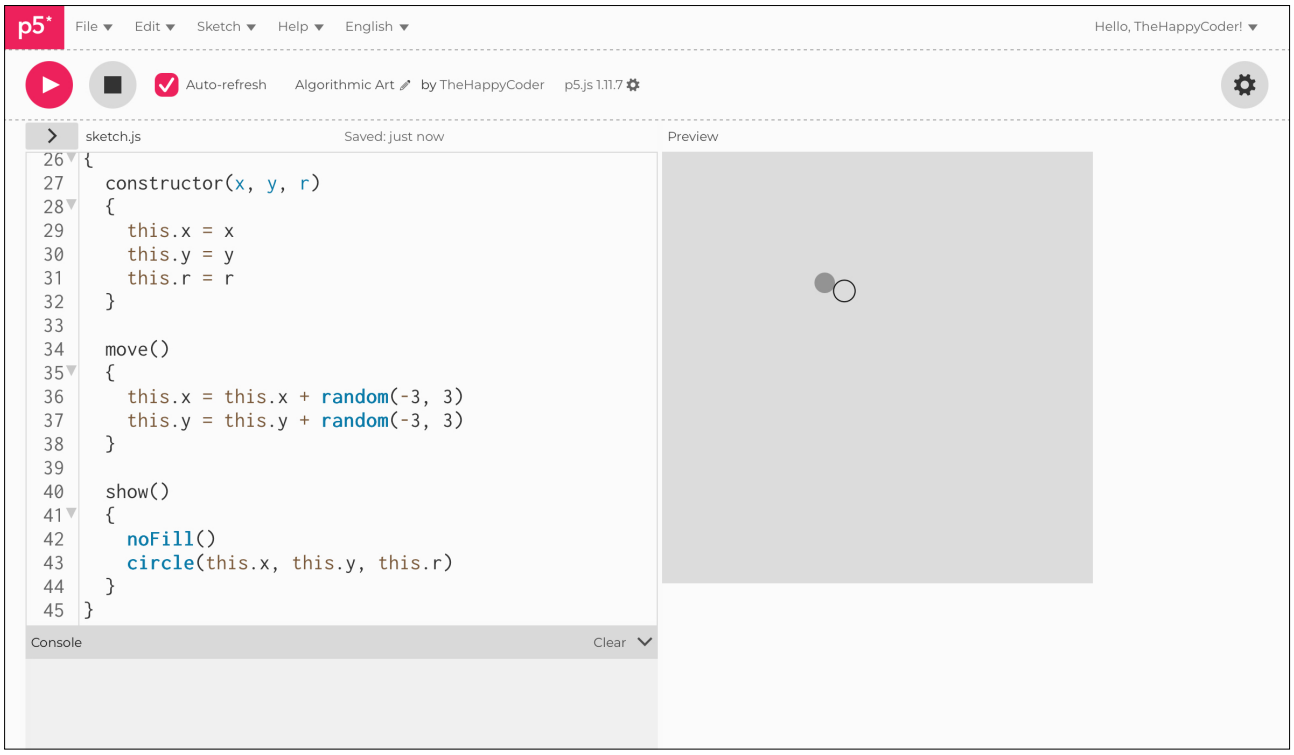
```
let bubbles = []
let bubble

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

function mousePressed()
{
  bubble = new Bubble(mouseX, mouseY, 20)
  bubbles[0] = bubble
}

function draw()
{
  background(220)
  for (let i = 0; i < bubbles.length; i++)
  {
    bubbles[i].show()
    bubbles[i].move()
  }
}

class Bubble
{
  constructor(x, y, r)
  {
    this.x = x
    this.y = y
    this.r = r
  }

  move()
```

```
  {
    this.x = this.x + random(-3, 3)
    this.y = this.y + random(-3, 3)
  }

  show()
  {
    noFill()
    circle(this.x, this.y, this.r)
  }
}
```

Notes

We create a new bubble every time we click on the canvas, but only one at a time. We want to keep each one, not discard the last one.

Figure D3.13



Sketch D3.14 when push comes to shove

We use a `push()` function to push each `bubble` into the array. Now we will keep all the `bubbles` we create with the mouse.

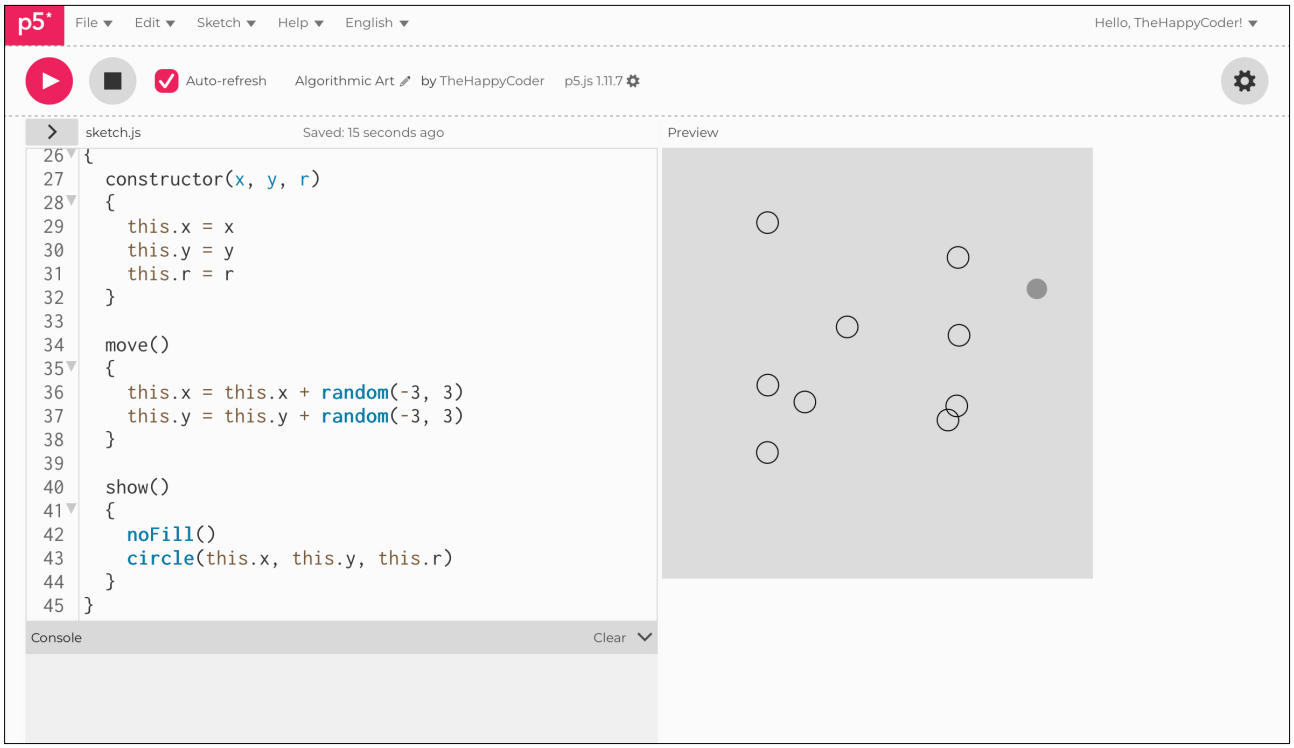
```
let bubbles = []
let bubble

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

function mousePressed()
{
  bubble = new Bubble(mouseX, mouseY, 20)
  bubbles.push(bubble)
}

function draw()
{
  background(220)
  for (let i = 0; i < bubbles.length; i++)
  {
    bubbles[i].show()
    bubbles[i].move()
  }
}

class Bubble
{
  constructor(x, y, r)
  {
    this.x = x
    this.y = y
    this.r = r
  }

  move()
```

```
  {
    this.x = this.x + random(-3, 3)
    this.y = this.y + random(-3, 3)
  }

  show()
  {
    noFill()
    circle(this.x, this.y, this.r)
  }
}
```

Notes

Now when you click on the canvas, each bubble remains on the canvas because they have been added to the array of bubbles.

Figure D3.14



Sketch D3.15 a bit of a drag

Finally, using `mouseDragged()` rather than just `mousePressed()`, we can make lots of `bubbles` as we click and drag the mouse.

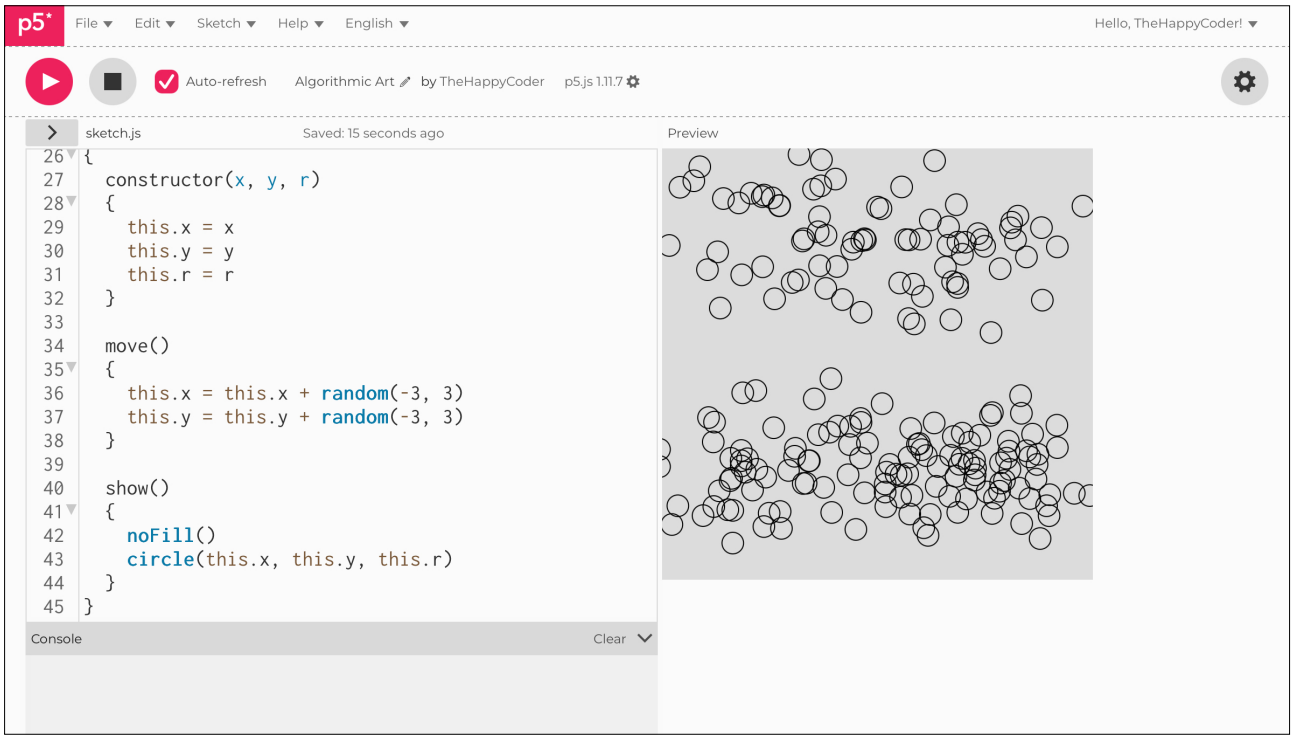
```
let bubbles = []
let bubble

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

function mouseDragged()
{
  bubble = new Bubble(mouseX, mouseY, 20)
  bubbles.push(bubble)
}

function draw()
{
  background(220)
  for (let i = 0; i < bubbles.length; i++)
  {
    bubbles[i].show()
    bubbles[i].move()
  }
}

class Bubble
{
  constructor(x, y, r)
  {
    this.x = x
    this.y = y
    this.r = r
  }

  move()
```

```
  {
    this.x = this.x + random(-3, 3)
    this.y = this.y + random(-3, 3)
  }

  show()
  {
    noFill()
    circle(this.x, this.y, this.r)
  }
}
```

Challenges

1. What could you create with a few tweaks?
2. How would you create random shapes?

Figure D3.15



The Joy of Coding Algorithmic Art Module D Unit #4 push and splice

Module D Unit #4: push and splice

Having a bit of a deeper dive into arrays and then some fun stuff. The first part, we use the `console.log()` to see inside the array, and then we are back on the `bubbles`.

We will also introduce the adding of more functions to a class, in this case, the `Bubble` class.

Sketch D4.1 recap

! A new sketch

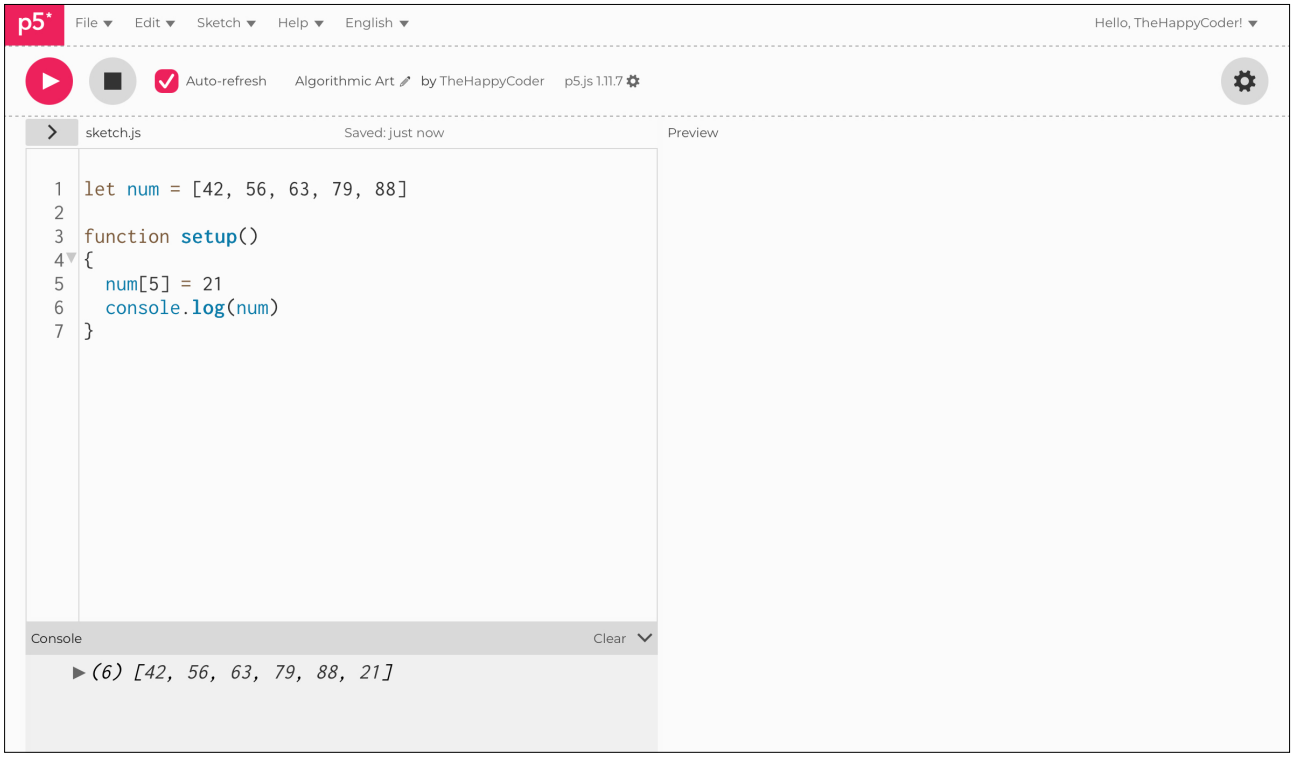
A simple way to add an element to an array is as follows: we are saying that element [5] is 21.

```
let num = [42, 56, 63, 79, 88]

function setup()
{
  num[5] = 21
  console.log(num)
}
```

Notes

You will need to make sure you know how long your array is; this is challenging when you might be adding or removing elements all the time. There is a better way.

Figure D4.1

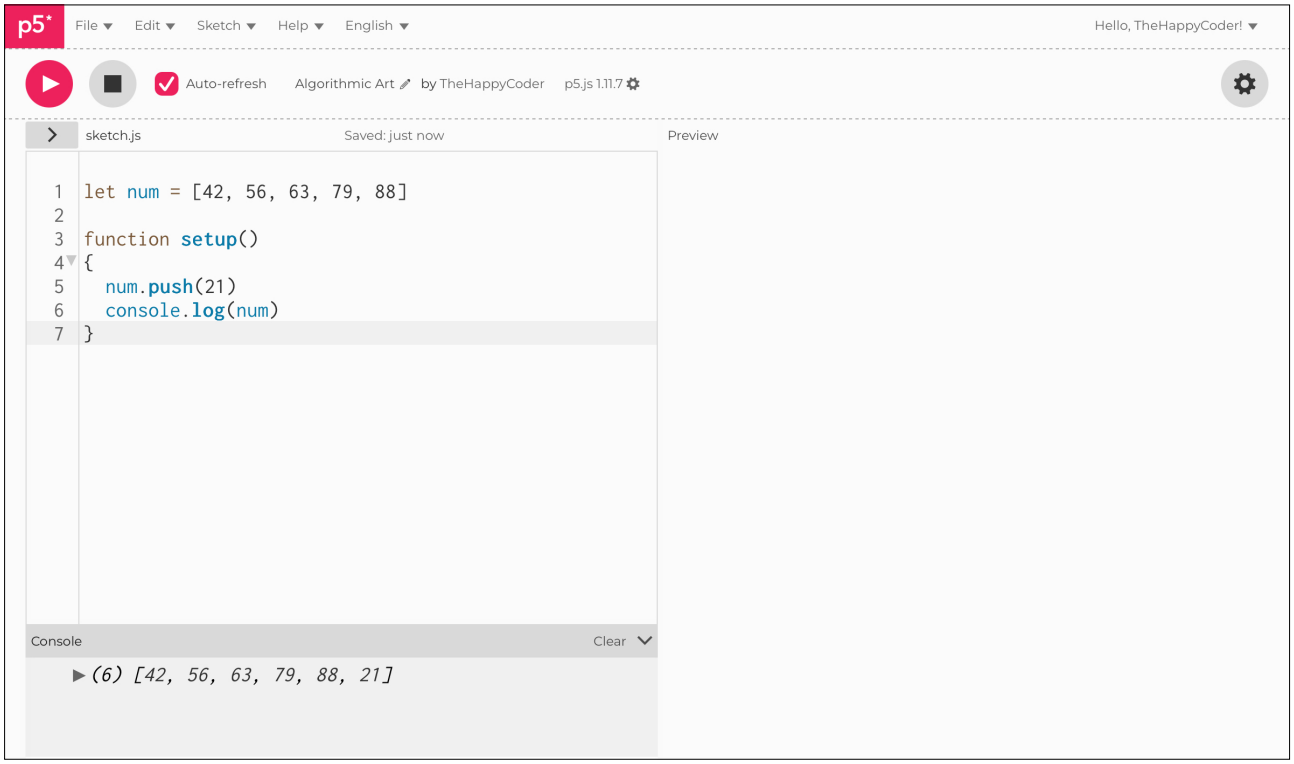


Sketch D4.2 push()

Using the push() function, it simply adds another element onto the end of the array, as we saw in the previous sketch.

```
let num = [42, 56, 63, 79, 88]

function setup()
{
  num.push(21)
  console.log(num)
}
```

Figure D4.2

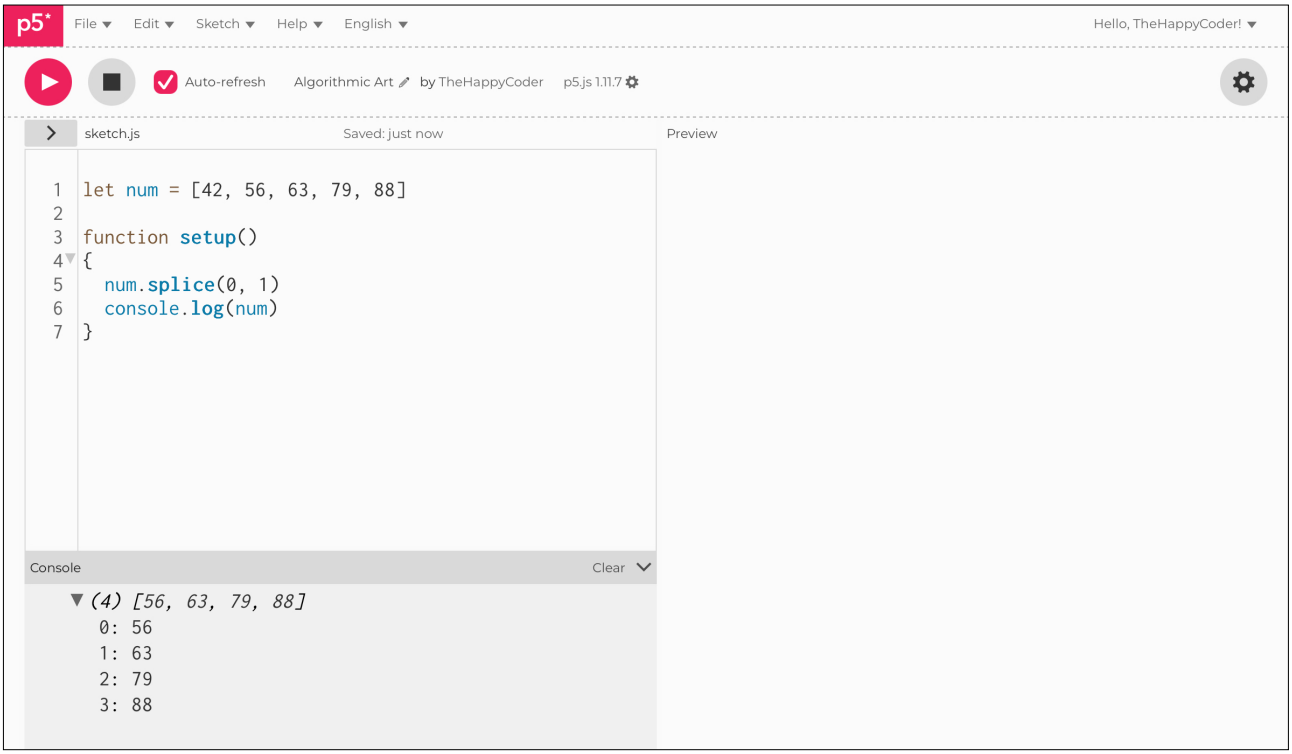


Sketch D4.3 splice() up your life

But what if we want to delete an element from an array? We use a function called `splice()`. It has two arguments: the first is the index, and the second tells you how many to delete (starting at the index). We want to delete the first one in the array, so it is `splice(0, 1)`.

```
let num = [42, 56, 63, 79, 88]


function setup()
{
  num.splice(0, 1)
  console.log(num)
}
```

Figure D4.3



Sketch D4.4 splicing a bubble

! New bubble sketch, I have highlighted the major changes.

To delete a bubble, we create a function called `keyPressed()` which will activate when you press a key on your keyboard. Also, I have reduced the random wobble so that you can see how they are deleted every time you hit a key.

! Remember to click and draw some bubbles on the canvas first.

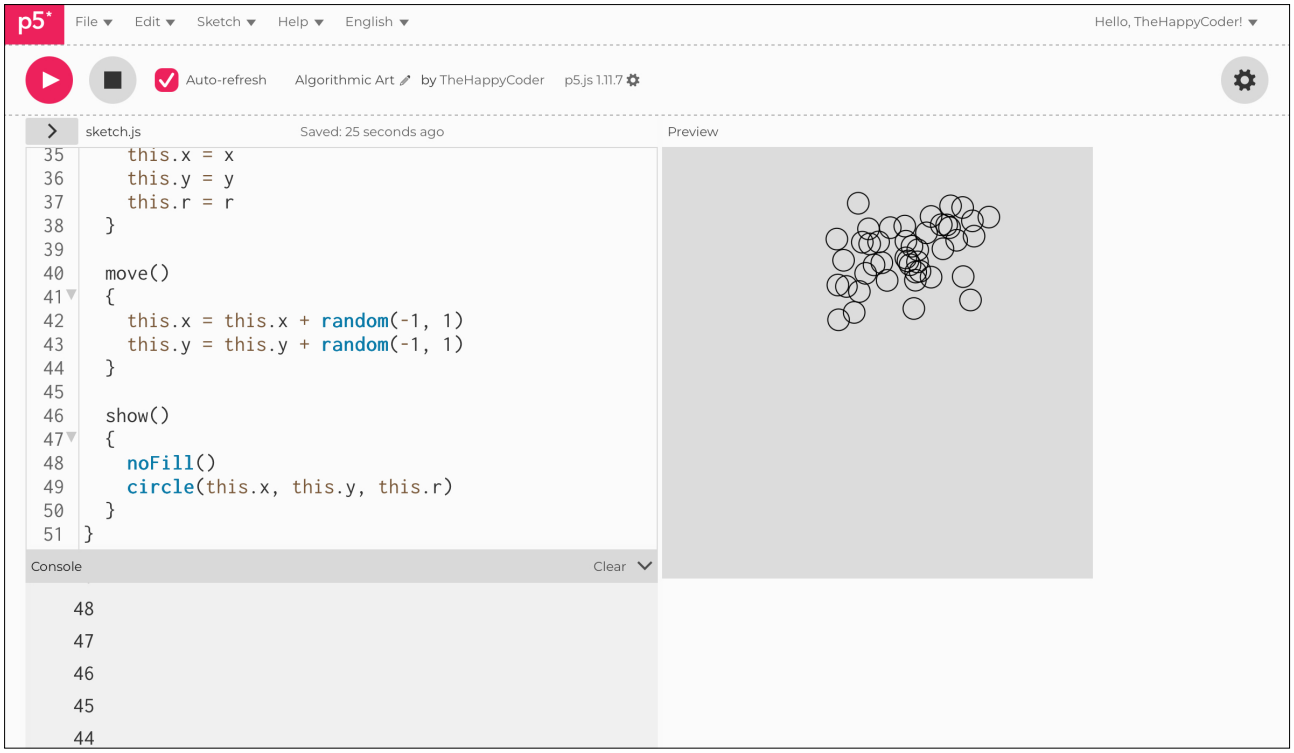
```
let bubbles = []
let bubble

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

function mouseDragged()
{
  bubble = new Bubble(mouseX, mouseY, 20)
  bubbles.push(bubble)
}

function keyPressed()
{
  bubbles.splice(0, 1)
  console.log(bubbles.length)
}

function draw()
{
  background(220)
  for (let i = 0; i < bubbles.length; i++)
  {
    bubbles[i].show()
    bubbles[i].move()
  }
}
```

```
class Bubble
{
  constructor(x, y, r)
  {
    this.x = x
    this.y = y
    this.r = r
  }

  move()
  {
    this.x = this.x + random(-1, 1)
    this.y = this.y + random(-1, 1)
  }

  show()
  {
    noFill()
    circle(this.x, this.y, this.r)
  }
}
```

Notes

I have included a `console.log()` so that you can see how many are in the array and see the number go down. Every time you press a key, you should see a bubble disappear, the oldest or first in the array.

Figure D4.4



Sketch D4.5 oversized array

Something else we can do is delete them if the array is over a certain size. For this, we remove the `keyPressed()` function completely. Here, the array is never bigger than `50` bubbles.

```
let bubbles = []
let bubble

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

function mouseDragged()
{
  bubble = new Bubble(mouseX, mouseY, 20)
  bubbles.push(bubble)
}

function draw()
{
  background(220)
  for (let i = 0; i < bubbles.length; i++)
  {
    bubbles[i].show()
    bubbles[i].move()
  }
  if (bubbles.length > 50)
  {
    bubbles.splice(0, 1)
  }
}

class Bubble
{
  constructor(x, y, r)
  {
    this.x = x
```

```
    this.y = y
    this.r = r
  }

  move()
  {
    this.x = this.x + random(-1, 1)
    this.y = this.y + random(-1, 1)
  }

  show()
  {
    noFill()
    circle(this.x, this.y, this.r)
  }
}
```

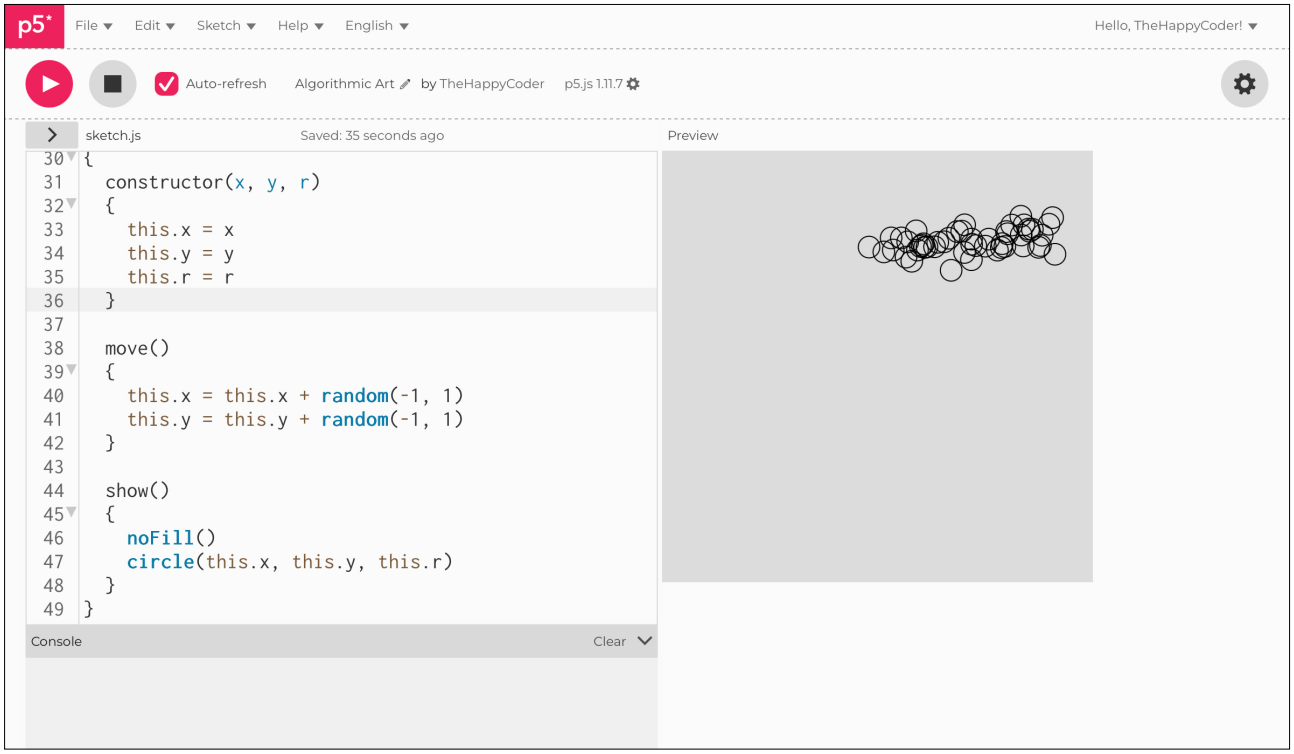
Notes

What you get is what looks like a trail following you, which never gets too long.

Challenge

Have them delete themselves if they wander off the canvas.

Figure D4.5



Sketch D4.6 just roll over

Changing the colour of the bubble when the mouse rolls over the bubble. First, we will create a function called `changeColour()` in the bubble class. It will have an argument called `bright`, which is set to `0` by default.

```
let bubbles = []
let bubble

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

function mouseDragged()
{
  bubble = new Bubble(mouseX, mouseY, 20)
  bubbles.push(bubble)
}

function draw()
{
  background(220)
  for (let i = 0; i < bubbles.length; i++)
  {
    bubbles[i].show()
    bubbles[i].move()
  }
  if (bubbles.length > 50)
  {
    bubbles.splice(0, 1)
  }
}

class Bubble
{
  constructor(x, y, r)
  {
    this.x = x
```

```
    this.y = y
    this.r = r
    this.brightness = 0
  }

  move()
  {
    this.x = this.x + random(-1, 1)
    this.y = this.y + random(-1, 1)
  }

  show()
  {
    noFill()
    circle(this.x, this.y, this.r)
  }

  changeColour(bright)
  {
    this.brightness = bright
  }
}
```

Notes

Nothing is yet changing, just building the sketch.

Sketch D4.7 just a short dist()

Next, we want a function that measures when the mouse is over the bubble. We create a function in the bubble class called `rollover()`. It receives two arguments, `px` and `py`, which are the `mouse X` and `mouseY` co-ordinates (which we will get to in a moment).

The function `dist()` measures the distance between two sets of co-ordinates; in this case, it is the distance between the centre of the bubble, `this.x`, `this.y`, and the `mouseX` and `mouseY` positions.

```
let bubbles = []
let bubble

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

function mouseDragged()
{
  bubble = new Bubble(mouseX, mouseY, 20)
  bubbles.push(bubble)
}

function draw()
{
  background(220)
  for (let i = 0; i < bubbles.length; i++)
  {
    bubbles[i].show()
    bubbles[i].move()
  }
  if (bubbles.length > 50)
  {
    bubbles.splice(0, 1)
  }
}

class Bubble
```

```
{
  constructor(x, y, r)
  {
    this.x = x
    this.y = y
    this.r = r
    this.brightness = 0
  }

  move()
  {
    this.x = this.x + random(-1, 1)
    this.y = this.y + random(-1, 1)
  }

  show()
  {
    noFill()
    circle(this.x, this.y, this.r)
  }

  changeColour(bright)
  {
    this.brightness = bright
  }

  rollover(px, py)
  {
    let d = dist(px, py, this.x, this.y)
  }
}
```

Notes

Still nothing new to see.

Sketch D4.8 inside the bubble

The radius of the bubble is `r` (`this.r`), so if `px` and `py` are less than the radius, we know that the mouse is inside the bubble. We use a simple boolean, `true` or `false`.

```
let bubbles = []
let bubble

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

function mouseDragged()
{
  bubble = new Bubble(mouseX, mouseY, 20)
  bubbles.push(bubble)
}

function draw()
{
  background(220)
  for (let i = 0; i < bubbles.length; i++)
  {
    bubbles[i].show()
    bubbles[i].move()
  }
  if (bubbles.length > 50)
  {
    bubbles.splice(0, 1)
  }
}

class Bubble
{
  constructor(x, y, r)
  {
    this.x = x
    this.y = y
```

```
    this.r = r
    this.brightness = 0
  }

  move()
  {
    this.x = this.x + random(-1, 1)
    this.y = this.y + random(-1, 1)
  }

  show()
  {
    noFill()
    circle(this.x, this.y, this.r)
  }

  changeColour(bright)
  {
    this.brightness = bright
  }

  rollover(px, py)
  {
    let d = dist(px, py, this.x, this.y)
    if (d < this.r/2)
    {
      return true
    }
    else
    {
      return false
    }
  }
}
```

Notes

Still the same old...

Sketch D4.9 true colour

We want to fill the bubble with whatever colour is true (255) or false (0), the default, so we use the `fill()` function with a little bit of alpha.

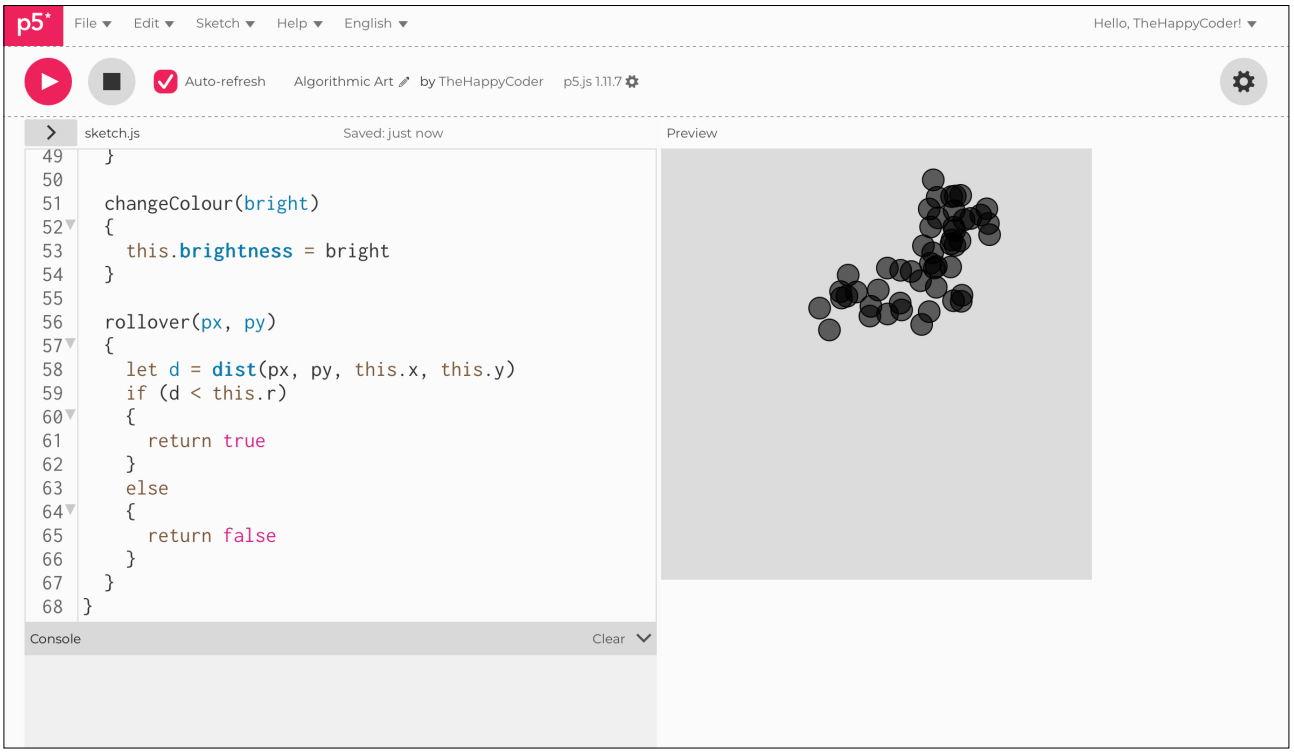
```
let bubbles = []
let bubble

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

function mouseDragged()
{
  bubble = new Bubble(mouseX, mouseY, 20)
  bubbles.push(bubble)
}

function draw()
{
  background(220)
  for (let i = 0; i < bubbles.length; i++)
  {
    bubbles[i].show()
    bubbles[i].move()
  }
  if (bubbles.length > 50)
  {
    bubbles.splice(0, 1)
  }
}

class Bubble
{
  constructor(x, y, r)
  {
    this.x = x
    this.y = y
```

```
    this.r = r
    this.brightness = 0
  }

  move()
  {
    this.x = this.x + random(-1, 1)
    this.y = this.y + random(-1, 1)
  }

  show()
  {
    fill(this.brightness, 150)
    circle(this.x, this.y, this.r)
  }

  changeColour(bright)
  {
    this.brightness = bright
  }

  rollover(px, py)
  {
    let d = dist(px, py, this.x, this.y)
    if (d < this.r/2)
    {
      return true
    }
    else
    {
      return false
    }
  }
}
```

Notes

You get something, what you get is the default dark grey (black with alpha) bubbles. We are not finished yet.

Figure D4.9



Sketch D4.10 array of bubbles check

We need to check through the array of bubbles to see if any of the bubbles contain the mouseX or mouseY. We use an if() statement for this job.

```
let bubbles = []
let bubble

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

function mouseDragged()
{
  bubble = new Bubble(mouseX, mouseY, 20)
  bubbles.push(bubble)
}

function draw()
{
  background(220)
  for (let i = 0; i < bubbles.length; i++)
  {
    if (bubbles[i].rollover(mouseX, mouseY))
    {
      bubbles[i].changeColour(255)
    }
    else
    {
      bubbles[i].changeColour(0)
    }
    bubbles[i].show()
    bubbles[i].move()
  }
  if (bubbles.length > 50)
  {
    bubbles.splice(0, 1)
  }
```

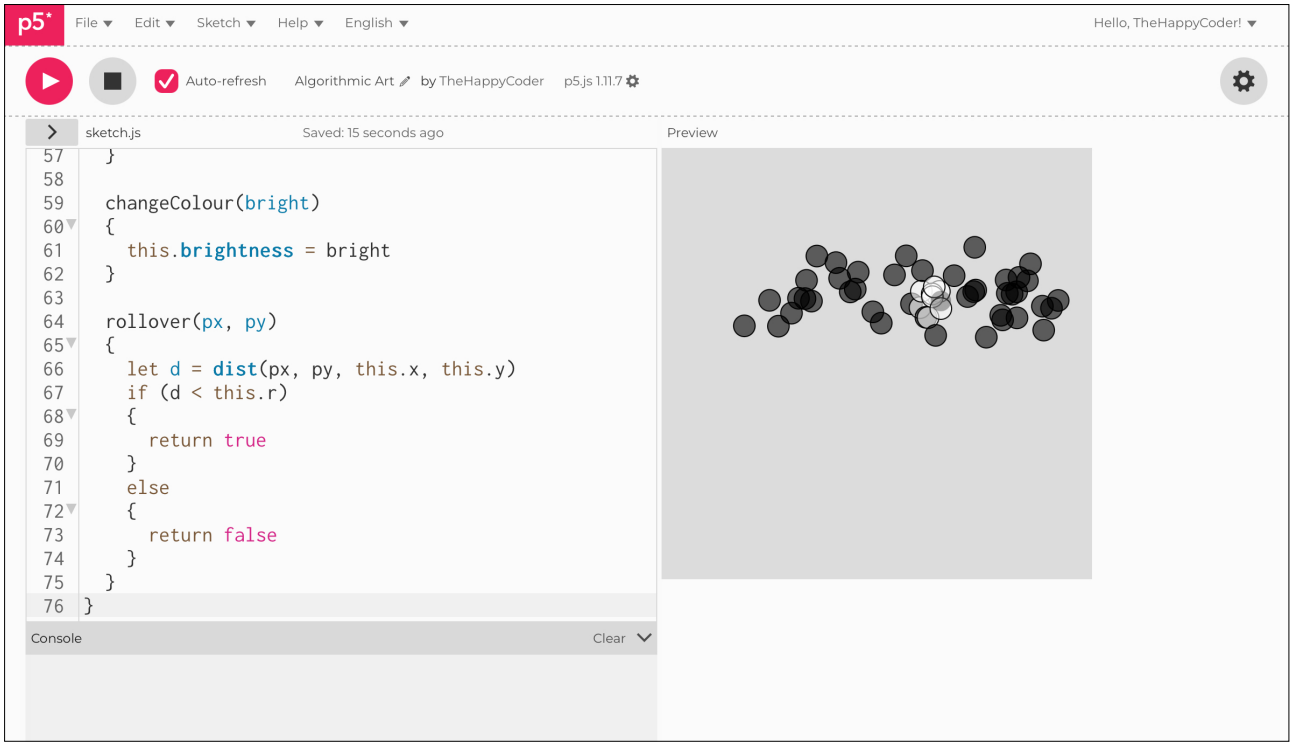
```
}

class Bubble
{
  constructor(x, y, r)
  {
    this.x = x
    this.y = y
    this.r = r
    this.brightness = 0
  }

  move()
  {
    this.x = this.x + random(-1, 1)
    this.y = this.y + random(-1, 1)
  }

  show()
  {
    fill(this.brightness, 150)
    circle(this.x, this.y, this.r)
  }

  changeColour(bright)
  {
    this.brightness = bright
  }

  rollover(px, py)
  {
    let d = dist(px, py, this.x, this.y)
    if (d < this.r/2)
    {
      return true
    }
    else
    {
```

```
      return false
    }
  }
}
```

Notes

Now the bubble changes to white when you hover or roll over it.

Figure D4.10



Sketch D4.11 click of a mouse

! Remove `mouseDragged()` and all its components.

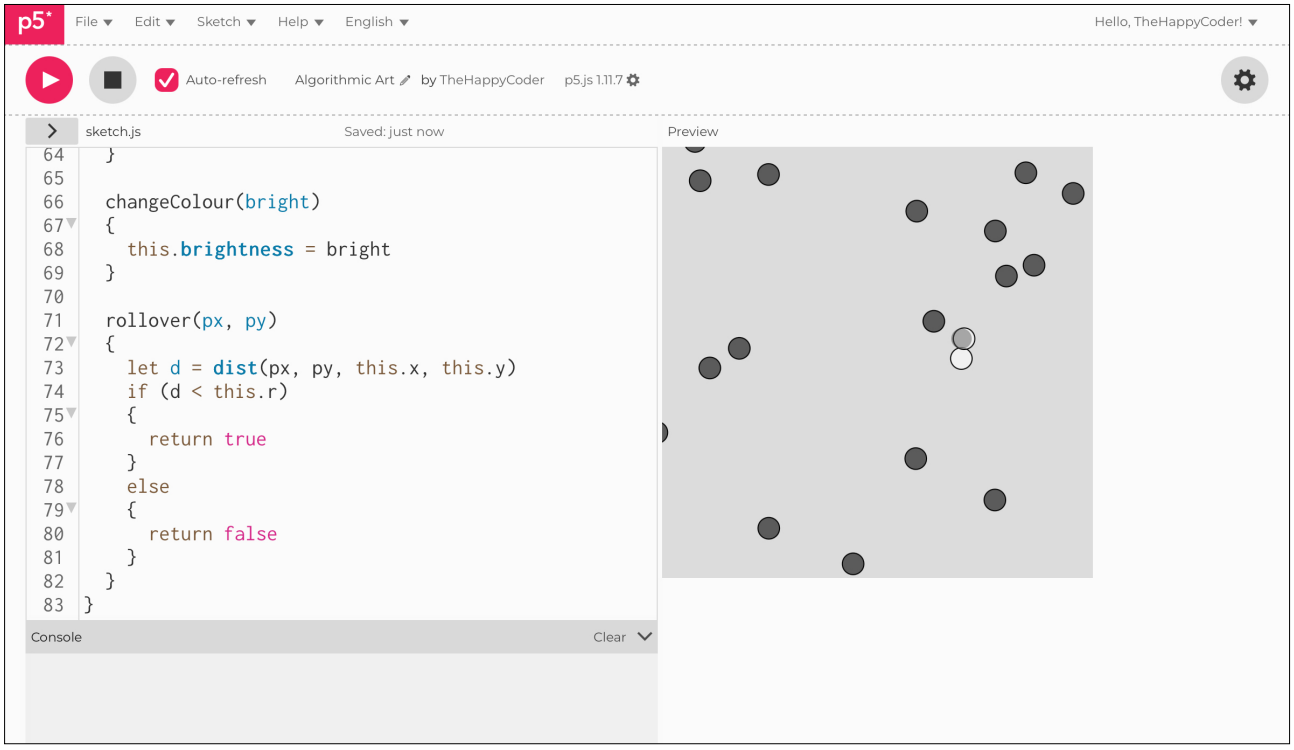
We want to delete a bubble with the click of a mouse. We use the `splice()` function for this. It removes one or more elements from an array and returns those spliced values. First, let us create `20` randomly distributed bubbles and change the colour when we roll over them.

```
let bubbles = []
let bubble

function setup()
{
  createCanvas(400, 400)
  for (let i = 0; i < 20; i++)
  {
    let x = random(width)
    let y = random(height)
    bubble = new Bubble(x, y, 20)
    bubbles.push(bubble)
  }
}

// function mouseDragged()
// {
//   bubble = new Bubble(mouseX, mouseY, 20)
//   bubbles.push(bubble)
// }

function draw()
{
  background(220)
  for (let i = 0; i < bubbles.length; i++)
  {
    if (bubbles[i].rollover(mouseX, mouseY))
    {
      bubbles[i].changeColour(255)
    }
    else
```

```
    {
      bubbles[i].changeColour(0)
    }
    bubbles[i].show()
    bubbles[i].move()
  }
  if (bubbles.length > 50)
  {
    bubbles.splice(0, 1)
  }
}

class Bubble
{
  constructor(x, y, r)
  {
    this.x = x
    this.y = y
    this.r = r
    this.brightness = 0
  }

  move()
  {
    this.x = this.x + random(-1, 1)
    this.y = this.y + random(-1, 1)
  }

  show()
  {
    fill(this.brightness, 150)
    circle(this.x, this.y, this.r)
  }

  changeColour(bright)
  {
    this.brightness = bright
  }
```

```
  rollover(px, py)
  {
    let d = dist(px, py, this.x, this.y)
    if (d < this.r/2)
    {
      return true
    }
    else
    {
      return false
    }
  }
}
```

Figure D4.11



Sketch D4.12 mousePressed()

Now we introduce `mousePressed()` and when there is a rollover and the mouse is pressed, we delete that `bubble` with `splice()`. The `mousePressed()` function has a `for()` loop that checks the array from the end and works backwards to the beginning to see if any of the bubbles have been clicked on.

The reason for doing it backwards is because of the index numbering. If you delete through starting at the beginning of the array, you will then change the indexing of the whole array, even though you are still working through the array one element at a time. It is possible you will miss an element otherwise.

```
let bubbles = []
let bubble

function setup()
{
  createCanvas(400, 400)
  for (let i = 0; i < 20; i++)
  {
    let x = random(width)
    let y = random(height)
    bubble = new Bubble(x, y, 20)
    bubbles.push(bubble)
  }
}

function mousePressed()
{
  for (let i = bubbles.length - 1; i >= 0; i--)
  {

  }
}

function draw()
{
  background(220)
  for (let i = 0; i < bubbles.length; i++)
  {
```

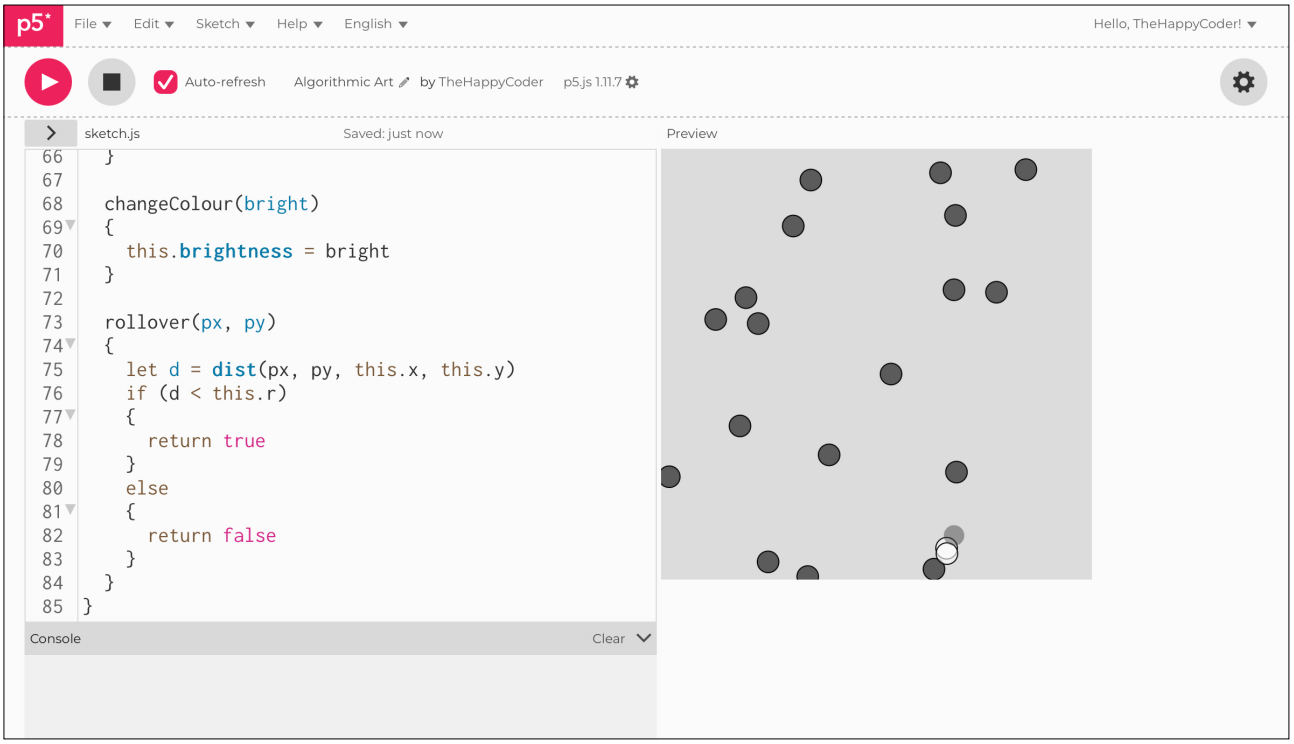
```
    if (bubbles[i].rollover(mouseX, mouseY))
    {
      bubbles[i].changeColour(255)
    }
    else
    {
      bubbles[i].changeColour(0)
    }
    bubbles[i].show()
    bubbles[i].move()
  }
  if (bubbles.length > 50)
  {
    bubbles.splice(0, 1)
  }
}

class Bubble
{
  constructor(x, y, r)
  {
    this.x = x
    this.y = y
    this.r = r
    this.brightness = 0
  }

  move()
  {
    this.x = this.x + random(-1, 1)
    this.y = this.y + random(-1, 1)
  }

  show()
  {
    fill(this.brightness, 150)
    circle(this.x, this.y, this.r)
  }
```

```
  changeColour(bright)
  {
    this.brightness = bright
  }

  rollover(px, py)
  {
    let d = dist(px, py, this.x, this.y)
    if (d < this.r/2)
    {
      return true
    }
    else
    {
      return false
    }
  }
}
```

Notes

Nothing new is happening just yet. We are just checking; we have an empty `for()` loop at the moment.

Figure D4.12



Sketch D4.13 mouse delete

Now we can delete a bubble when it is clicked.

```
let bubbles = []
let bubble

function setup()
{
  createCanvas(400, 400)
  for (let i = 0; i < 20; i++)
  {
    let x = random(width)
    let y = random(height)
    bubble = new Bubble(x, y, 20)
    bubbles.push(bubble)
  }
}

function mousePressed()
{
  for (let i = bubbles.length - 1; i >= 0; i--)
  {
    if (bubbles[i].rollover(mouseX, mouseY))
    {
      bubbles.splice(i, 1)
    }
  }
}

function draw()
{
  background(220)
  for (let i = 0; i < bubbles.length; i++)
  {
    if (bubbles[i].rollover(mouseX, mouseY))
    {
      bubbles[i].changeColour(255)
    }
```

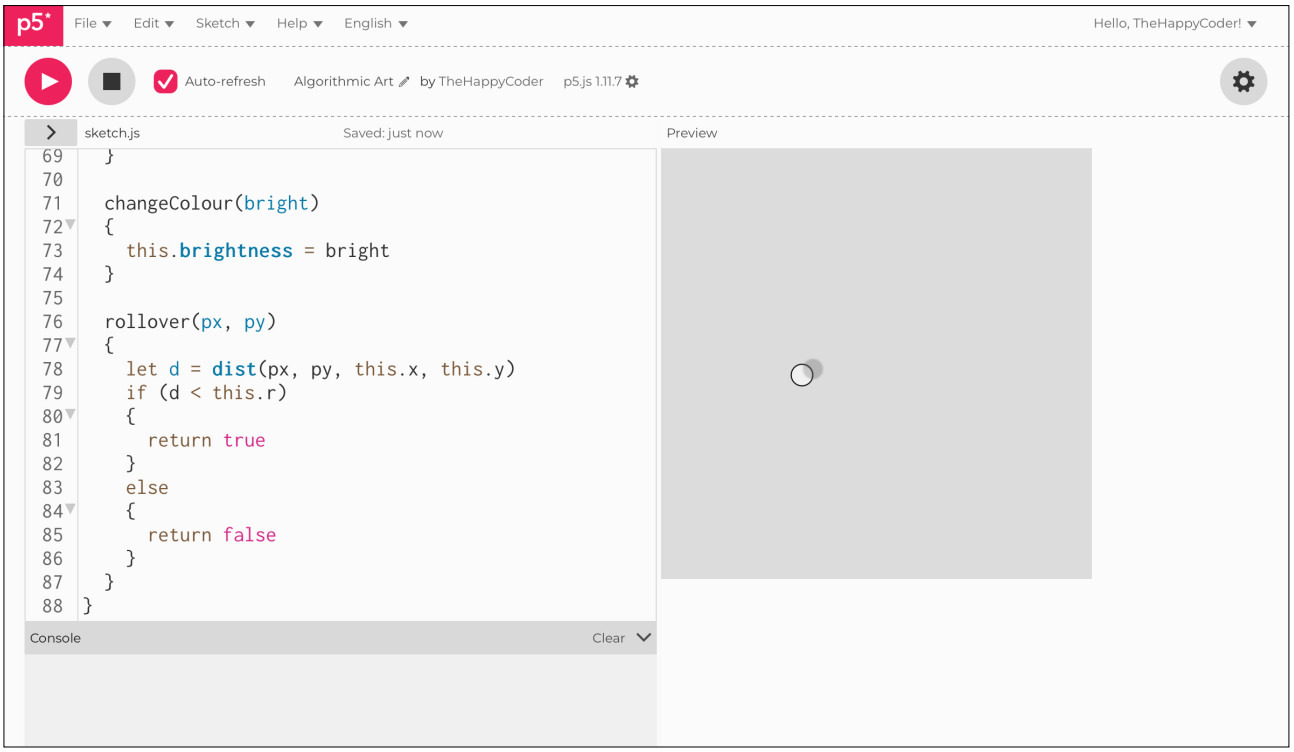
```
    else
    {
      bubbles[i].changeColour(0)
    }
    bubbles[i].show()
    bubbles[i].move()
  }
  if (bubbles.length > 50)
  {
    bubbles.splice(0, 1)
  }
}

class Bubble
{
  constructor(x, y, r)
  {
    this.x = x
    this.y = y
    this.r = r
    this.brightness = 0
  }

  move()
  {
    this.x = this.x + random(-1, 1)
    this.y = this.y + random(-1, 1)
  }

  show()
  {
    fill(this.brightness, 150)
    circle(this.x, this.y, this.r)
  }

  changeColour(bright)
  {
    this.brightness = bright
```

```
  }

  rollover(px, py)
  {
    let d = dist(px, py, this.x, this.y)
    if (d < this.r/2)
    {
      return true
    }
    else
    {
      return false
    }
  }
}
```

Notes

You should see the bubble change colour still when you hover over it, but then when you click on it, it should disappear.

Figure D4.13



The Joy of Coding Algorithmic Art

Module D Unit #5 the overlap

Module D Unit #5: the overlap

We will cover more aspects of arrays where we can add and remove elements from an array. Keep hold of what you did in the last unit as we will use it again and build on it.

Sketch D5.1 removing some code

! Using the sketch from the previous unit.

We want to change the colour of the bubbles if they overlap each other rather than when we move the mouse. We start by deleting the following lines of code (// in blue).

```
let bubbles = []
let bubble

function setup()
{
  createCanvas(400, 400)
  for (let i = 0; i < 20; i++)
  {
    let x = random(width)
    let y = random(height)
    bubble = new Bubble(x, y, 20)
    bubbles.push(bubble)
  }
}

function draw()
{
  background(220)
  for (let i = 0; i < bubbles.length; i++)
  {
    // if (bubbles[i].rollover(mouseX, mouseY))
    // {
    //   bubbles[i].changeColour(255)
    // }
    // else
    // {
    //   bubbles[i].changeColour(0)
    // }
    bubbles[i].show()
    bubbles[i].move()
  }
  // if (bubbles.length > 50)
```

```
  // {
  //   bubbles.splice(0, 1)
  // }
}

class Bubble
{
  constructor(x, y, r)
  {
    this.x = x
    this.y = y
    this.r = r
    this.brightness = 0
  }

  move()
  {
    this.x = this.x + random(-1, 1)
    this.y = this.y + random(-1, 1)
  }

  show()
  {
    fill(this.brightness, 150)
    circle(this.x, this.y, this.r)
  }

  changeColour(bright)
  {
    this.brightness = bright
  }

//   rollover(px, py)
//   {
//     let d = dist(px, py, this.x, this.y)
//     if (d < this.r/2)
//     {
//       return true
```

```
//     }
//     else
//     {
//       return false
//     }
//   }
}
```

Notes

We want the bare bones.

Sketch D5.2 this is what is left

This is what we have left.

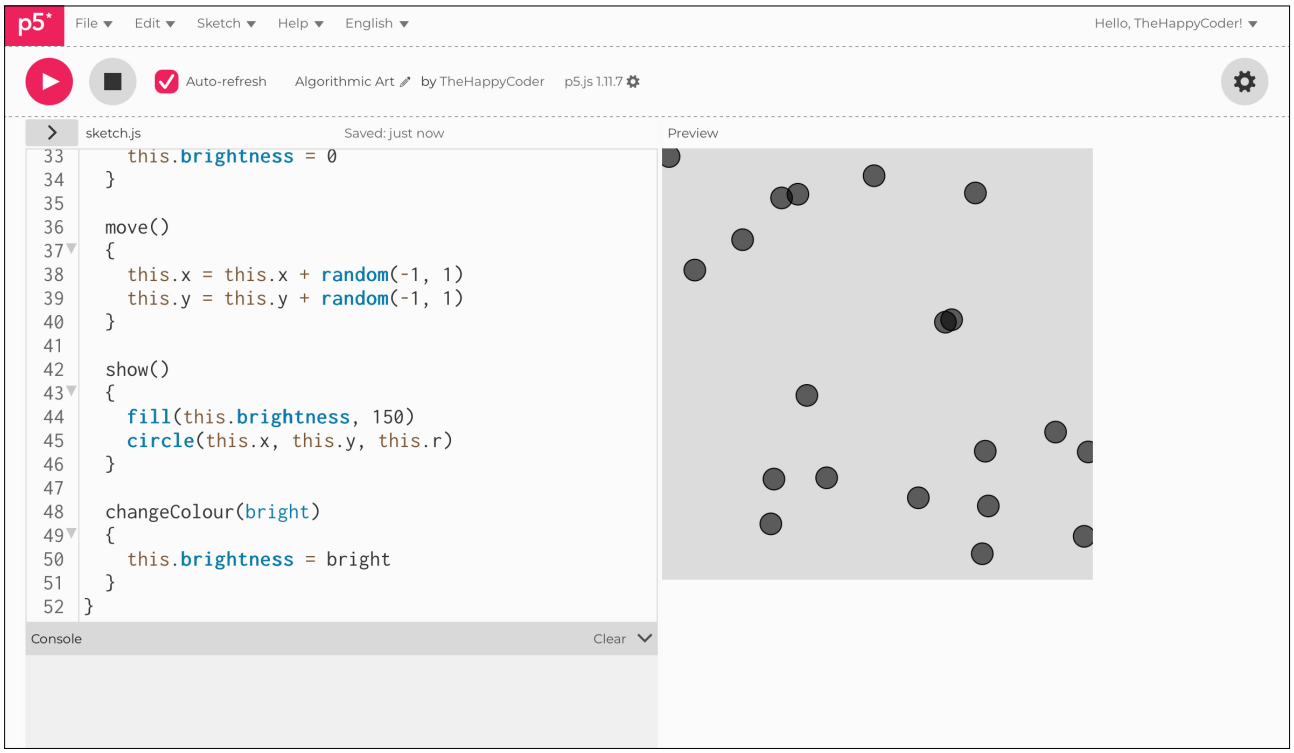
```
let bubbles = []
let bubble

function setup()
{
  createCanvas(400, 400)
  for (let i = 0; i < 20; i++)
  {
    let x = random(width)
    let y = random(height)
    bubble = new Bubble(x, y, 20)
    bubbles.push(bubble)
  }
}

function draw()
{
  background(220)
  for (let i = 0; i < bubbles.length; i++)
  {
    bubbles[i].show()
    bubbles[i].move()
  }
}

class Bubble
{
  constructor(x, y, r)
  {
    this.x = x
    this.y = y
    this.r = r
    this.brightness = 0
  }
```

```
  move()
  {
    this.x = this.x + random(-1, 1)
    this.y = this.y + random(-1, 1)
  }

  show()
  {
    fill(this.brightness, 150)
    circle(this.x, this.y, this.r)
  }

  changeColour(bright)
  {
    this.brightness = bright
  }
}
```

Notes

We get our 20 wandering bubbles.

Figure D5.2



Sketch D5.3 the intersection

This is where it gets tricky. We want to check if a particular bubble in the array is overlapping with another bubble (but not itself). Before we do that bit, we need to have a function that checks for the intersection of the two bubbles, for instance, whether the distance between the centres is less than the addition of the two radii. If you draw two circles touching, their distance centre to centre is equal to the radius of one plus the radius of the other.

We have two bubbles, one bubble is `this.___` and the other bubble is `other.___`.

```
let bubbles = []
let bubble

function setup()
{
  createCanvas(400, 400)
  for (let i = 0; i < 20; i++)
  {
    let x = random(width)
    let y = random(height)
    bubble = new Bubble(x, y, 20)
    bubbles.push(bubble)
  }
}

function draw()
{
  background(220)
  for (let i = 0; i < bubbles.length; i++)
  {
    bubbles[i].show()
    bubbles[i].move()
  }
}

class Bubble
{
  constructor(x, y, r)
  {
```

```
    this.x = x
    this.y = y
    this.r = r
    this.brightness = 0
  }

  intersects(other)
  {
    let d = dist(this.x, this.y, other.x, other.y)
    return d < this.r + other.r
  }

  move()
  {
    this.x = this.x + random(-1, 1)
    this.y = this.y + random(-1, 1)
  }

  show()
  {
    fill(this.brightness, 150)
    circle(this.x, this.y, this.r)
  }

  changeColour(bright)
  {
    this.brightness = bright
  }
}
```

Notes

We create another function in the Bubbles class called `intersect()` and we have an argument called `other` (which we will come to shortly). We return the distance `d` only if it is less than the two radii. That means they are overlapping or intersecting. We haven't created the `other` array just yet, but we will do so next.

Code Explanation

`return d < this.r + other.r`	We only want d if it less than than both radii combined

Sketch D5.4 the other bubble

We create another array called other.

```
let bubbles = []
let bubble

function setup()
{
  createCanvas(400, 400)
  for (let i = 0; i < 20; i++)
  {
    let x = random(width)
    let y = random(height)
    bubble = new Bubble(x, y, 20)
    bubbles.push(bubble)
  }
}

function draw()
{
  background(220)
  for (let i = 0; i < bubbles.length; i++)
  {
    bubbles[i].show()
    bubbles[i].move()
    for (let other of bubbles)
    {

    }
  }
}

class Bubble
{
  constructor(x, y, r)
  {
    this.x = x
    this.y = y
```

```
    this.r = r
    this.brightness = 0
  }

  intersects(other)
  {
    let d = dist(this.x, this.y, other.x, other.y)
    return d < this.r + other.r
  }

  move()
  {
    this.x = this.x + random(-1, 1)
    this.y = this.y + random(-1, 1)
  }

  show()
  {
    fill(this.brightness, 150)
    circle(this.x, this.y, this.r)
  }

  changeColour(bright)
  {
    this.brightness = bright
  }
}
```

Notes

The `let...of` allows us to effectively copy the `bubbles` array and call it `other`.

Sketch D5.5 not the same

We need to check that we are not comparing the same bubble.

```
let bubbles = []
let bubble

function setup()
{
  createCanvas(400, 400)
  for (let i = 0; i < 20; i++)
  {
    let x = random(width)
    let y = random(height)
    bubble = new Bubble(x, y, 20)
    bubbles.push(bubble)
  }
}

function draw()
{
  background(220)
  for (let i = 0; i < bubbles.length; i++)
  {
    bubbles[i].show()
    bubbles[i].move()
    for (let other of bubbles)
    {
      if (bubbles[i] !== other)
      {


      }
    }
  }
}

class Bubble
{
  constructor(x, y, r)
```

```
  {
    this.x = x
    this.y = y
    this.r = r
    this.brightness = 0
  }

  intersects(other)
  {
    let d = dist(this.x, this.y, other.x, other.y)
    return d < this.r + other.r
  }

  move()
  {
    this.x = this.x + random(-1, 1)
    this.y = this.y + random(-1, 1)
  }

  show()
  {
    fill(this.brightness, 150)
    circle(this.x, this.y, this.r)
  }

  changeColour(bright)
  {
    this.brightness = bright
  }
}
```

Notes

This is a neat bit of code that compares any element from the `other` array with any element from the `bubbles` array. It uses the `!==` (not the same) comparison.

Code Explanation

`if (bubbles[i] !== other)`	Cycles through all the elements in the bubbles array, looking for the ones that are NOT in the other array.

Sketch D5.6 do we have intersection?

We also need to know if it intersects. So now we have the full condition: it cannot be comparing itself to itself (`bubble !== other`) and it also must (`&&`) intersect.

```
let bubbles = []
let bubble

function setup()
{
  createCanvas(400, 400)
  for (let i = 0; i < 20; i++)
  {
    let x = random(width)
    let y = random(height)
    bubble = new Bubble(x, y, 20)
    bubbles.push(bubble)
  }
}

function draw()
{
  background(220)
  for (let i = 0; i < bubbles.length; i++)
  {
    bubbles[i].show()
    bubbles[i].move()
    for (let other of bubbles)
    {
      if (bubbles[i] !== other && bubbles[i].intersects(other))
      {


      }
    }
  }
}

class Bubble
{
```

```
  constructor(x, y, r)
  {
    this.x = x
    this.y = y
    this.r = r
    this.brightness = 0
  }

  intersects(other)
  {
    let d = dist(this.x, this.y, other.x, other.y)
    return d < this.r + other.r
  }

  move()
  {
    this.x = this.x + random(-1, 1)
    this.y = this.y + random(-1, 1)
  }

  show()
  {
    fill(this.brightness, 150)
    circle(this.x, this.y, this.r)
  }

  changeColour(bright)
  {
    this.brightness = bright
  }
}
```

Notes

This may seem very confusing, but it is extremely logical. Work through it slowly, reading it out loud as if it is a sentence; sometimes it makes more sense that way.

Sketch D5.7 false by default

We are now going to add a boolean expression for overlapping. By default, the boolean will be called `false` and only change that to `true` when it isn't the same AND it intersects.

```
let bubbles = []
let bubble

function setup()
{
  createCanvas(400, 400)
  for (let i = 0; i < 20; i++)
  {
    let x = random(width)
    let y = random(height)
    bubble = new Bubble(x, y, 20)
    bubbles.push(bubble)
  }
}

function draw()
{
  background(220)
  for (let i = 0; i < bubbles.length; i++)
  {
    bubbles[i].show()
    bubbles[i].move()
    let overlapping = false
    for (let other of bubbles)
    {
      if (bubbles[i] !== other && bubbles[i].intersects(other))
      {
        overlapping = true
      }
    }
  }
}
```

```
class Bubble
{
  constructor(x, y, r)
  {
    this.x = x
    this.y = y
    this.r = r
    this.brightness = 0
  }

  intersects(other)
  {
    let d = dist(this.x, this.y, other.x, other.y)
    return d < this.r + other.r
  }

  move()
  {
    this.x = this.x + random(-1, 1)
    this.y = this.y + random(-1, 1)
  }

  show()
  {
    fill(this.brightness, 150)
    circle(this.x, this.y, this.r)
  }

  changeColour(bright)
  {
    this.brightness = bright
  }
}
```

Notes

A boolean variable is just either `true` or `false`. It is either overlapping or it is not; we start off `false` (not overlapping), until it is, then it is `true`.

Sketch D5.8 use the info

We need to do something with that information. Change the colour.

```
let bubbles = []
let bubble

function setup()
{
  createCanvas(400, 400)
  for (let i = 0; i < 20; i++)
  {
    let x = random(width)
    let y = random(height)
    bubble = new Bubble(x, y, 20)
    bubbles.push(bubble)
  }
}

function draw()
{
  background(220)
  for (let i = 0; i < bubbles.length; i++)
  {
    bubbles[i].show()
    bubbles[i].move()
    let overlapping = false
    for (let other of bubbles)
    {
      if (bubbles[i] !== other && bubbles[i].intersects(other))
      {
        overlapping = true
      }
    }
    if (overlapping)
    {
      bubbles[i].changeColour(255)
    }
    else
```

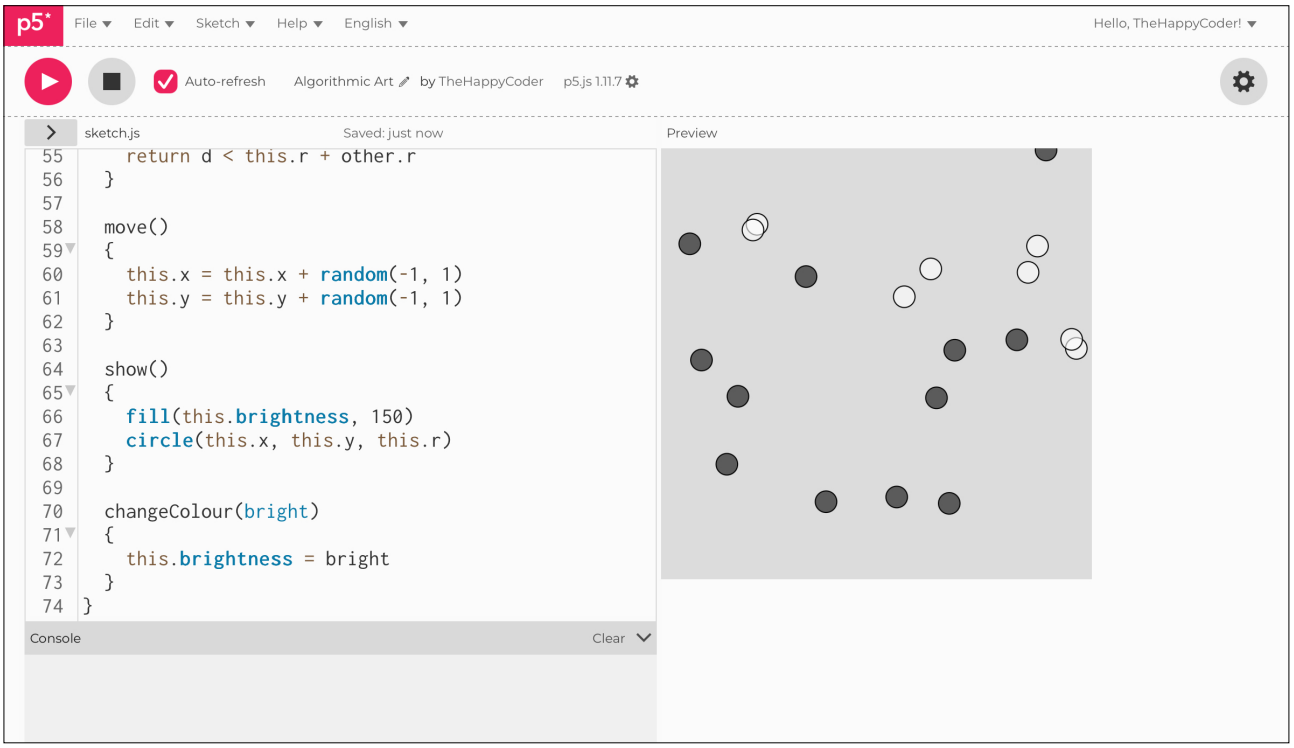
```
    {
      bubbles[i].changeColour(0)
    }
  }
}

class Bubble
{
  constructor(x, y, r)
  {
    this.x = x
    this.y = y
    this.r = r
    this.brightness = 0
  }

  intersects(other)
  {
    let d = dist(this.x, this.y, other.x, other.y)
    return d < this.r + other.r
  }

  move()
  {
    this.x = this.x + random(-1, 1)
    this.y = this.y + random(-1, 1)
  }

  show()
  {
    fill(this.brightness, 150)
    circle(this.x, this.y, this.r)
  }

  changeColour(bright)
  {
    this.brightness = bright
  }
```

```
}
```

Notes

If overlapping is `true`, then colour it white (255); otherwise, it is `false` and so colour it black (0). However, if you look closely, there is something wrong; they aren't overlapping but are still changing. This is because we have set the radius to 20, but when we draw the circle, it is the diameter.

Figure D5.8



Sketch D5.9 simple change

Making a very simple change.

```
let bubbles = []
let bubble

function setup()
{
  createCanvas(400, 400)
  for (let i = 0; i < 20; i++)
  {
    let x = random(width)
    let y = random(height)
    bubble = new Bubble(x, y, 20)
    bubbles.push(bubble)
  }
}

function draw()
{
  background(220)
  for (let i = 0; i < bubbles.length; i++)
  {
    bubbles[i].show()
    bubbles[i].move()
    let overlapping = false
    for (let other of bubbles)
    {
      if (bubbles[i] !== other && bubbles[i].intersects(other))
      {
        overlapping = true
      }
    }
    if (overlapping)
    {
      bubbles[i].changeColour(255)
    }
    else
```

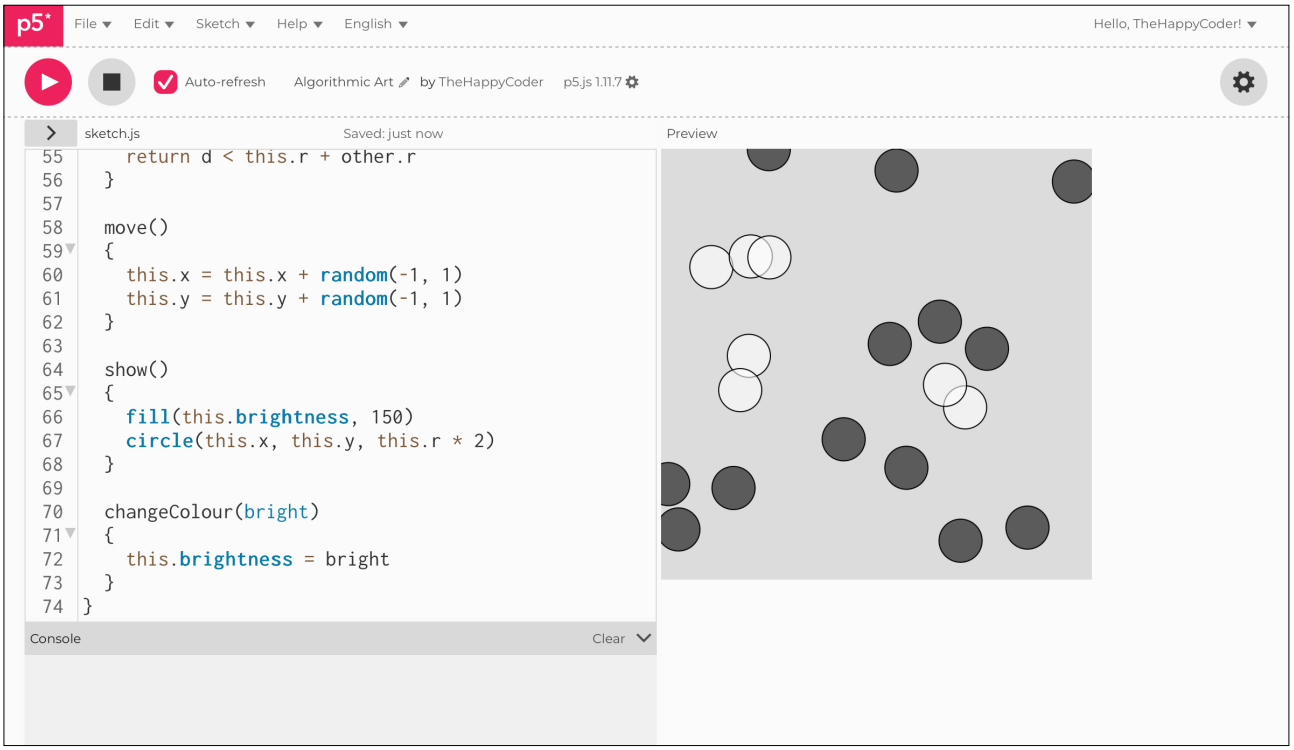
```
    {
      bubbles[i].changeColour(0)
    }
  }
}

class Bubble
{
  constructor(x, y, r)
  {
    this.x = x
    this.y = y
    this.r = r
    this.brightness = 0
  }

  intersects(other)
  {
    let d = dist(this.x, this.y, other.x, other.y)
    return d < this.r + other.r
  }

  move()
  {
    this.x = this.x + random(-1, 1)
    this.y = this.y + random(-1, 1)
  }

  show()
  {
    fill(this.brightness, 150)
    circle(this.x, this.y, this.r * 2)
  }

  changeColour(bright)
  {
    this.brightness = bright
  }
```

```
}
```

Notes

Double the radius; there is a reason for doing this.

Figure D5.9



Sketch D5.10 random sizes

The reason for doing it that way is so that you could have randomly sized bubbles.

```
let bubbles = []
let bubble

function setup()
{
  createCanvas(400, 400)
  for (let i = 0; i < 20; i++)
  {
    let x = random(width)
    let y = random(height)
    let r = random(10, 30)
    bubble = new Bubble(x, y, r)
    bubbles.push(bubble)
  }
}

function draw()
{
  background(220)
  for (let i = 0; i < bubbles.length; i++)
  {
    bubbles[i].show()
    bubbles[i].move()
    let overlapping = false
    for (let other of bubbles)
    {
      if (bubbles[i] !== other && bubbles[i].intersects(other))
      {
        overlapping = true
      }
    }
    if (overlapping)
    {
      bubbles[i].changeColour(255)
    }
```

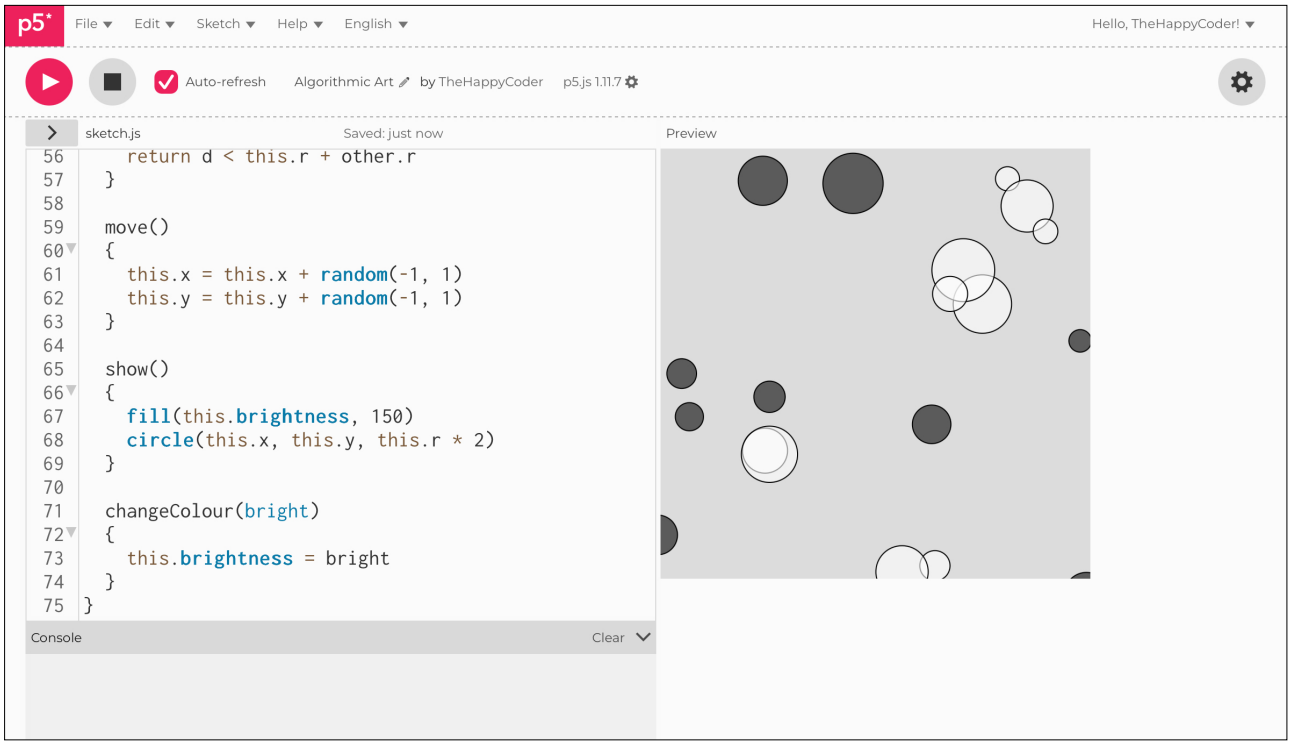
```
    else
    {
      bubbles[i].changeColour(0)
    }
  }
}

class Bubble
{
  constructor(x, y, r)
  {
    this.x = x
    this.y = y
    this.r = r
    this.brightness = 0
  }

  intersects(other)
  {
    let d = dist(this.x, this.y, other.x, other.y)
    return d < this.r + other.r
  }

  move()
  {
    this.x = this.x + random(-1, 1)
    this.y = this.y + random(-1, 1)
  }

  show()
  {
    fill(this.brightness, 150)
    circle(this.x, this.y, this.r * 2)
  }

  changeColour(bright)
  {
    this.brightness = bright
```

```
  }
}
```

Notes

Now it works for any size of radius for any bubble.

Figure D5.10



The Joy of Coding Algorithmic Art

Module D Unit #6 pixel arrays

Module D Unit #6: pixel arrays

A canvas is a grid of pixels, a `pixel` array. Each with its own colour. Each pixel is made up of four channels: `red`, `green`, `blue`, and `alpha`. We use this when we are colouring in our shapes in `RGB` mode.

We can get all the pixel information and also alter it in our code. For that, we use two functions: `loadPixels()` and `updatePixels()`. There is still quite a bit happening between these two functions.

We have a grid where we can allocate an (`x`, `y`) co-ordinate for each pixel. We use nested `for()` loops to scan all the pixels and update them.

If you are using a Mac or a Retina display, you will need to incorporate a function called `pixelDensity()`; weird things happen.

We need to get our head round how the pixel array is arranged; otherwise, it will seem a bit confusing. The array holds the four values for each pixel. One pixel may have something like this:

```
let pixel = [132, 231, 5, 255]
```

Where `132` is the red value, `231` is the green value, `5` is the blue value, and `255` is the alpha value. If we have two pixels, then the array looks like this:

```
let pixel = [132, 231, 5, 255, 65, 21, 200, 157]
```

The last four elements of the array are the red value (`65`), green value (`21`), blue value (`200`) and the alpha (`157`) of the second pixel and so on. So the pixel array holds information about each pixel in blocks of four, hence why we have to leap over each block in the loops.

So if you have a canvas of (`400`, `400`) with each pixel having four channels then the length of the pixel array is `400 x 400 x 4 = 640,000` elements in the array even though there are only `160,000` pixels.

We have another challenge to get the index value for a particular pixel. To do this we need a simple formula. The first row is straightforward. We just add `x` and `y` where `y` is zero. However when `y = 1` for the second row `x` has already reached `400` (canvas width). So now the formula is `x` + (`y` times `width`). The code will look like this:

```
let index[i] = x + (y * width)
```

This means that the index value for the pixel on the second row is:

```
0 + (1 * 400) = 400
```

This is correct and the next one is:

```
1 + (1 * 400) = 401
```

...and so on. However, when we are looping through to get a particular value from the array, for example, the red value for a pixel, we need to jump four elements in the array, so to get all the red values, or green, or blue, or alpha, we now use the formula below:

```
let index[i] = x + (y * width) * 4
```

This may seem a bit laboured but hopefully will help as we work through this shortly. Just remember what the array actually represents (one long list of values for every channel and every pixel) and how you are accessing it with the nested `for()` loops. It is all very logical.

Sketch D6.1 starting sketch

Our starting sketch.

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

function draw()
{
  background(220)
}
```

Sketch D6.2 loading the pixels

Introducing the two functions. The first gets all the pixels on the canvas with an array of 640,000 elements. This does nothing except store that information. Next, we will access a pixel and do something to it.

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

function draw()
{
  background(220)
  loadPixels()
  updatePixels()
}
```

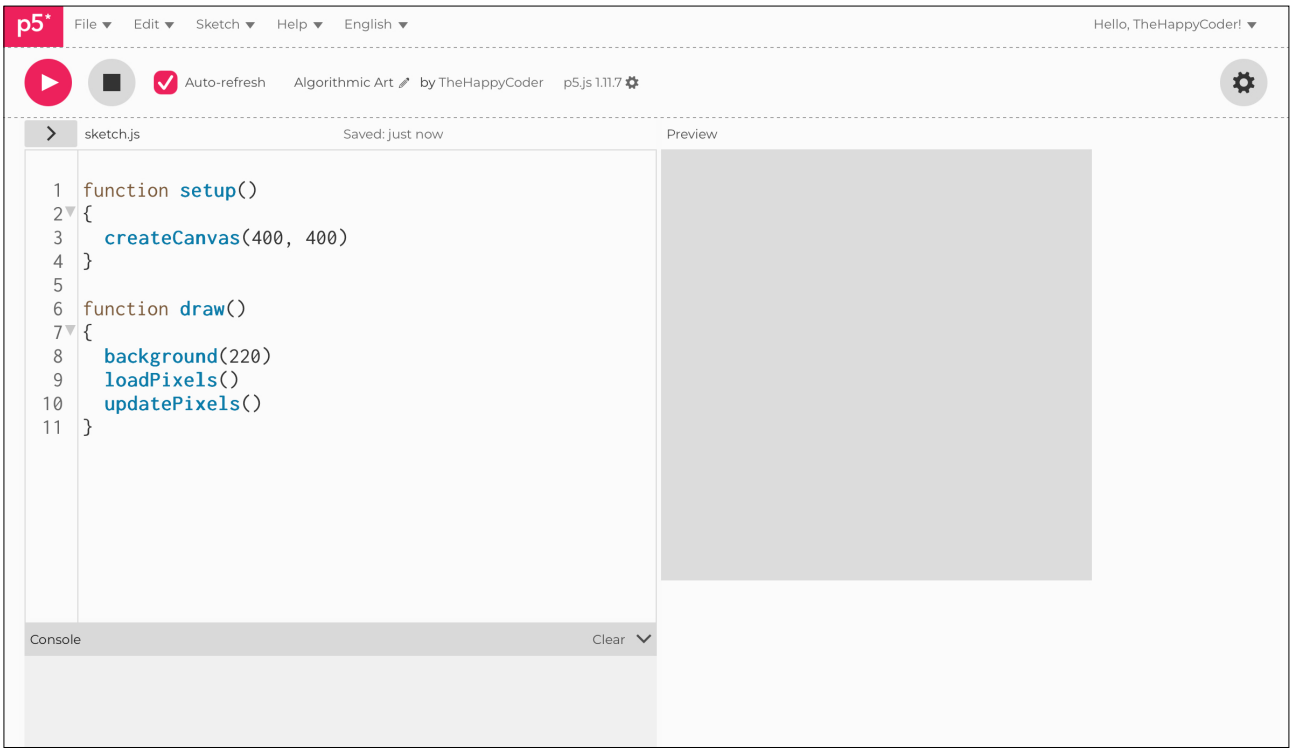
Notes

All we have done is load the pixel array and updated the array, which means we have done nothing.

Code Explanation

`loadPixels()`	Loads all the pixels (elements) in the canvas.
`updatePixels()`	This would update any changes to the pixel array.

Figure D6.2



Sketch D6.3 change the pixel

To see what we are doing here, we will change the first pixel because we know where each channel is in the array. Index`[0]` is red, `index[1]` is green, `index[2]` is blue, and `index[3]` is the alpha. What we have done here is change the very first pixel, making it red. You may struggle to see this, but it is there if you zoom in (see Figure D6.3b).

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

function draw()
{
  background(220)
  loadPixels()
  pixels[0] = 255
  pixels[1] = 0
  pixels[2] = 0
  pixels[3] = 255
  updatePixels()
}
```

Notes

We loaded and changed the first four elements of the pixel array, and then updated the array to draw the results on the canvas.

Challenges

1. Make some other changes, play with the values.
2. Can you make it blue?

Code Explanation

`pixels[0] = 255`	The very first element is the red component, giving it 255.
`pixels[1] = 0`	The second element is the green component value of 0.
`pixels[2] = 0`	The third element is the blue, which is also 0.
`pixels[3] = 255`	The fourth is the alpha of a value of 255.

Figure D6.3a

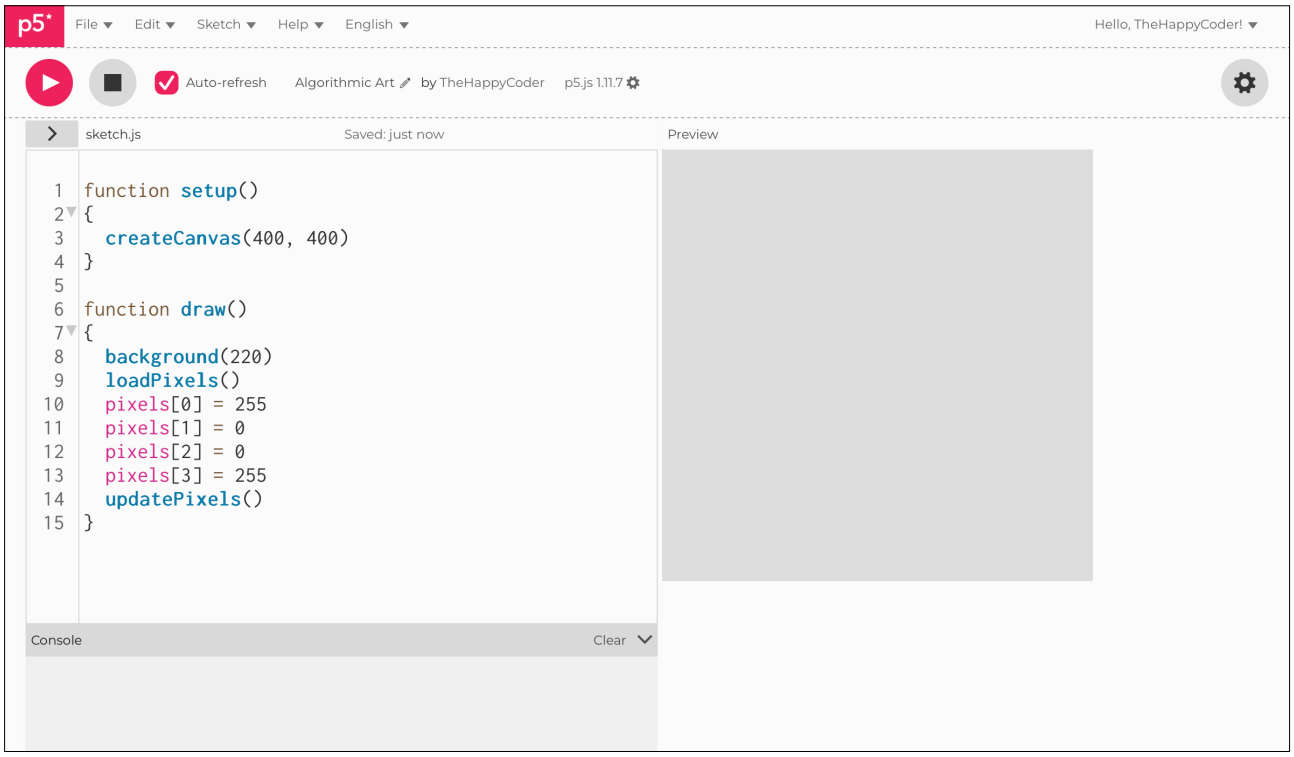


Figure D6.3b: a red pixel (zoomed in)

Sketch D6.4 a row of pixels

In this instance, we are just going to make all the pixels in the first row red. We need to jump every `four` elements in the array, and so we need to multiply the `width` by `four` (there are four times as many elements as pixels). Also, I needed to bring in the `pixelDensity()`, otherwise, it may not work.

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

function draw()
{
  background(220)
  loadPixels()
  for (let x = 0; x < width * 4; x += 4)
  {
    pixels[x + 0] = 255
    pixels[x + 1] = 0
    pixels[x + 2] = 0
    pixels[x + 3] = 255
  }
  updatePixels()
}
```

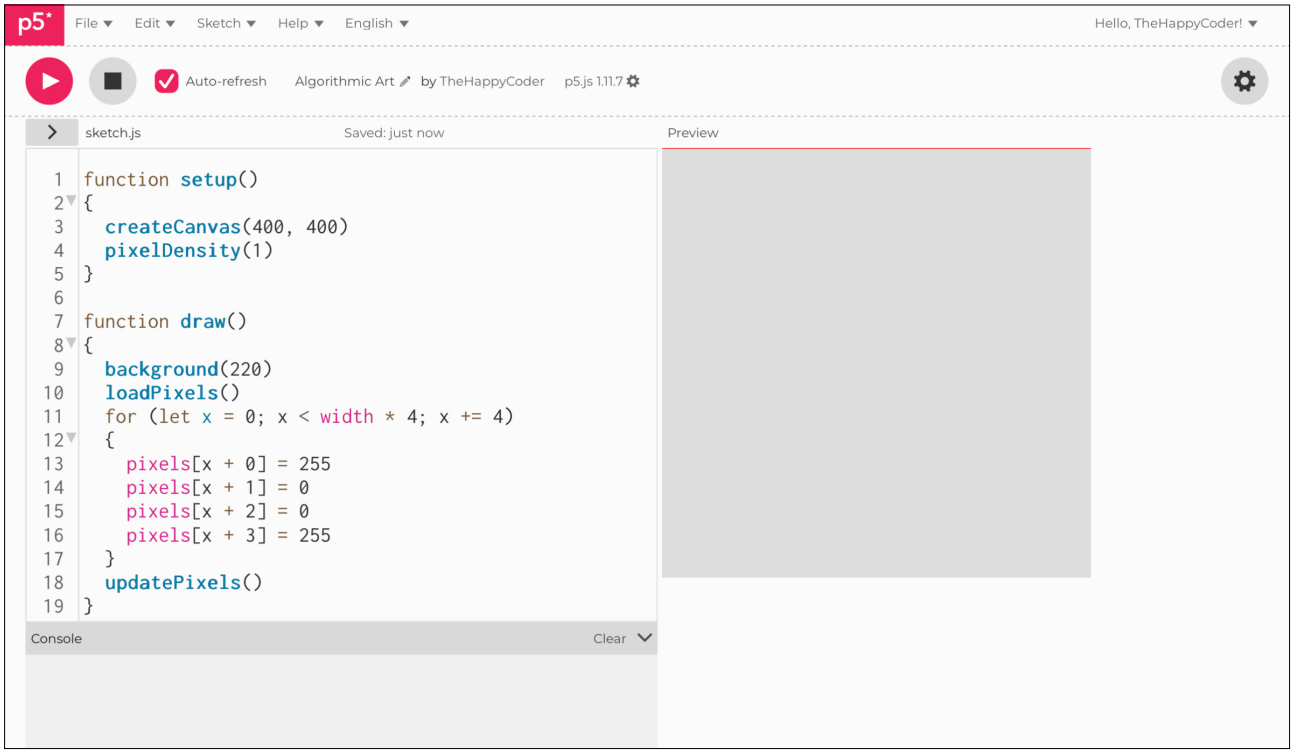
Notes

You can see what we have done: created a red line. We multiplied by four as the width is just 400.

Challenge

Try it without multiplying by four.

Figure D6.4



Sketch D6.5 every pixel array

To colour every pixel on the canvas red, we need to build the nested loop. We start with the y value as the outside of the nested loops because we want to move from top to bottom and on each row left to right. We add in the formula for the index.

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

function draw()
{
  background(220)
  loadPixels()
  for (let y = 0; y < height; y++)
  {
    for (let x = 0; x < width; x++)
    {
      let index = (x + (y * width)) * 4
      pixels[index + 0] = 255
      pixels[index + 1] = 0
      pixels[index + 2] = 0
      pixels[index + 3] = 255
    }
  }
  updatePixels()
}
```

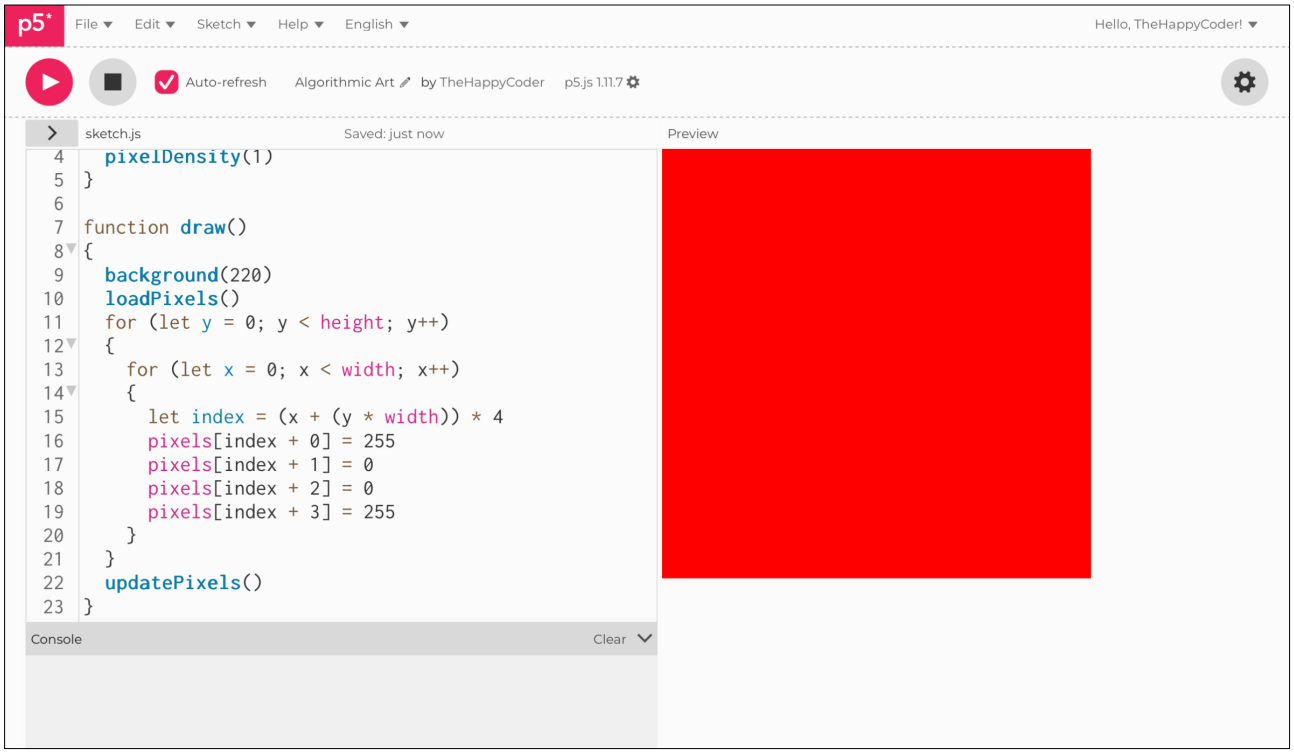
Notes

A nested loop covers the entire canvas.

Challenge

Try going from left to right rather than top to bottom.

Figure D6.5



Sketch D6.6 the pixel values

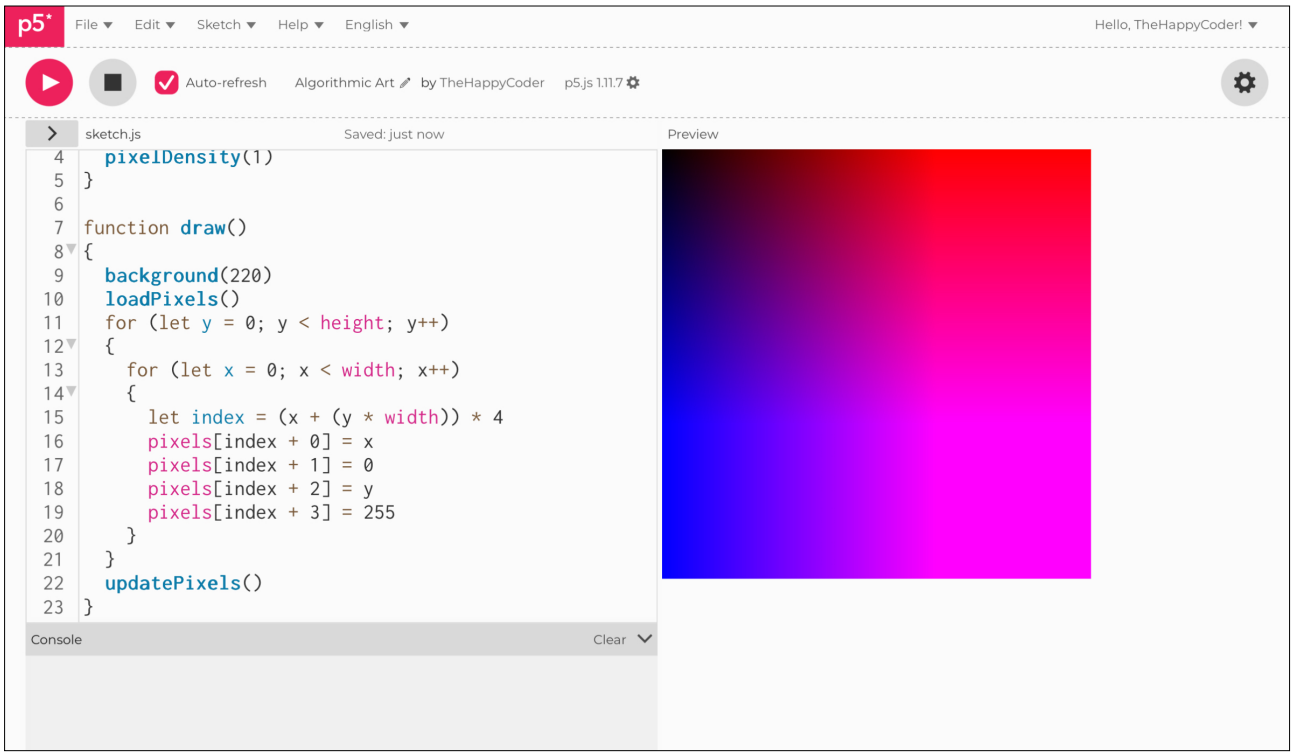
We can play with these values and do interesting stuff. Play around with the variables yourself.

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

function draw()
{
  background(220)
  loadPixels()
  for (let y = 0; y < height; y++)
  {
    for (let x = 0; x < width; x++)
    {
      let index = (x + (y * width)) * 4
      pixels[index + 0] = x
      pixels[index + 1] = 0
      pixels[index + 2] = y
      pixels[index + 3] = 255
    }
  }
  updatePixels()
}
```

Notes

You are linking the pixel element value to its position on the canvas.

Figure D6.6



Sketch D6.7 another effect

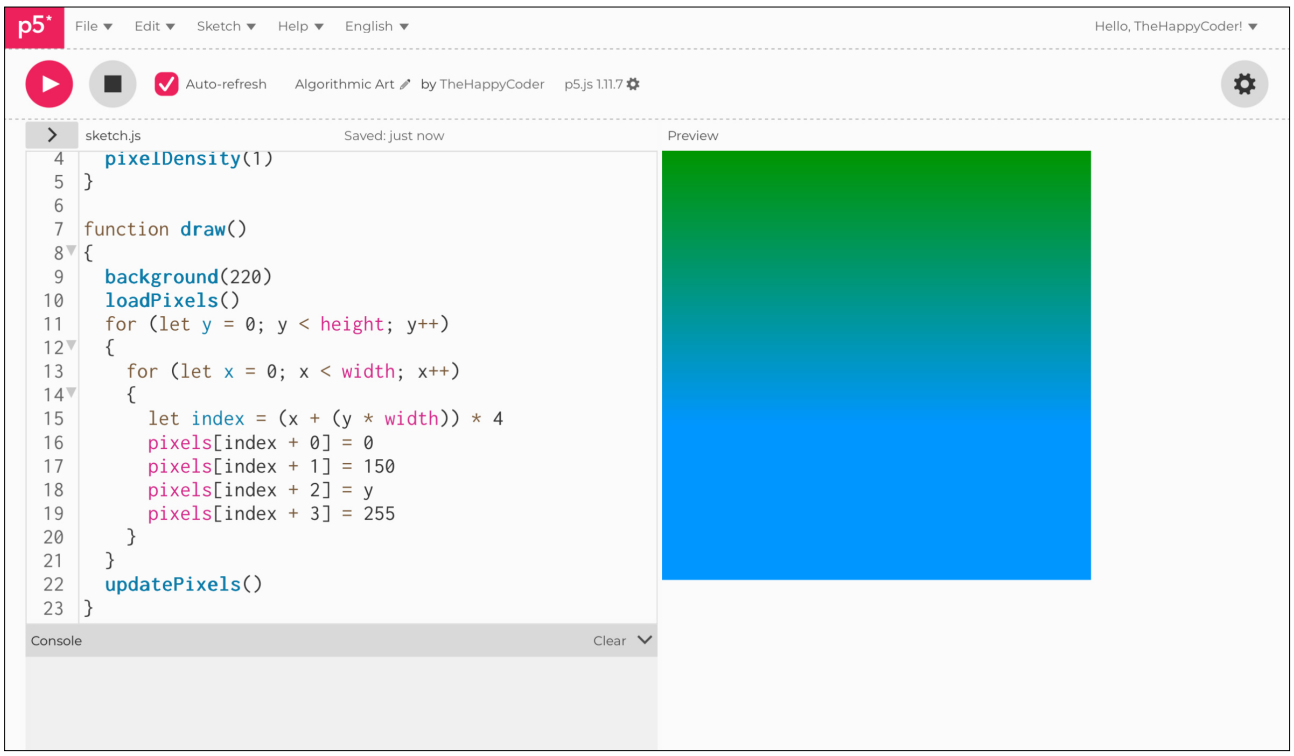
And yet more playing, incorporating the y value.

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

function draw()
{
  background(220)
  loadPixels()
  for (let y = 0; y < height; y++)
  {
    for (let x = 0; x < width; x++)
    {
      let index = (x + (y * width)) * 4
      pixels[index + 0] = 0
      pixels[index + 1] = 150
      pixels[index + 2] = y
      pixels[index + 3] = 255
    }
  }
  updatePixels()
}
```

Challenge

Consider using `mouseX` and `mouseY` (see below).

Figure D6.7

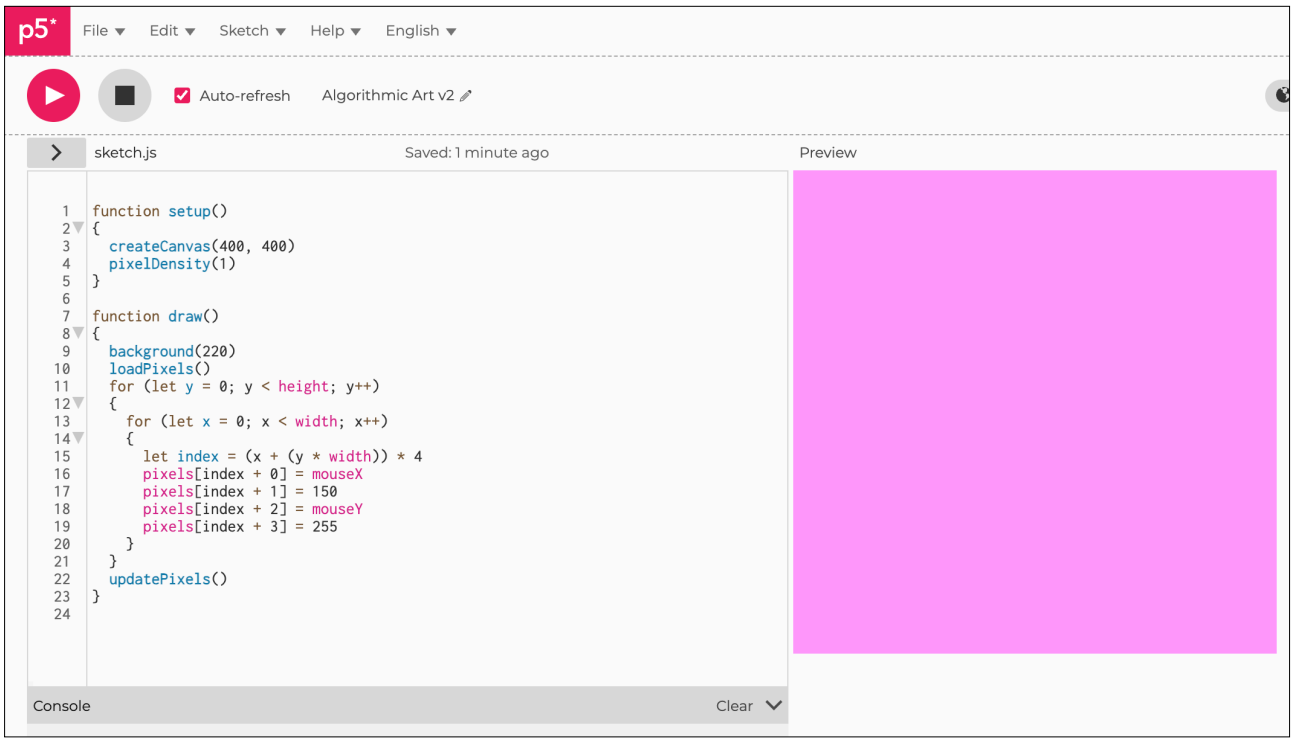


Sketch D6.8 the mouse effect

Using the mouse to determine the colour.

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

function draw()
{
  background(220)
  loadPixels()
  for (let y = 0; y < height; y++)
  {
    for (let x = 0; x < width; x++)
    {
      let index = (x + (y * width)) * 4
      pixels[index + 0] = mouseX
      pixels[index + 1] = 150
      pixels[index + 2] = mouseY
      pixels[index + 3] = 255
    }
  }
  updatePixels()
}
```

Figure D6.8


p5*
File
Edit
Sketch
Help
English
Auto-refresh
Algorithmic Art v2
sketch.js
Saved: 1 minute ago
Preview
function setup()
{
createCanvas(400, 400)
pixelDensity(1)
}
function draw()
{
background(220)
loadPixels()
for (let y = 0; y < height; y++)
{
for (let x = 0; x < width; x++)
{
let index = (x + (y * width)) * 4
pixels[index + 0] = mouseX
pixels[index + 1] = 150
pixels[index + 2] = mouseY
pixels[index + 3] = 255
}
}
updatePixels()
}
Console
Clear

The Joy of Coding Algorithmic Art

Module D Unit #7 3D arrays part 1

Module D Unit #7: 3D arrays part 1

In this unit, we are going to create a cube class and introduce 3D arrays. We will have a load of spinning cubes created every time you click on the canvas.

Sketch D7.1 new sketch

! New sketch for this unit.

We will call them cubes (rather than boxes), and because they are objects, we will create a class called Cube with a `constructor()`, `move()` and `show()` function.

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

function draw()
{
  background(220)
}

class Cube
{
  constructor()
  {

  }

  move()
  {

  }

  show()
  {

  }
}
```

Notes

This is our basic class sketch.

Sketch D7.2 mouse pressed cubes

We want to click on the canvas and put a cube there. So we need a `mousePressed()` function, an array called `cubes`, and a variable called `cube` (which is one box).

```
let cubes = []
let cube

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

function mousePressed()
{

}

function draw()
{
  background(220)
}

class Cube
{
  constructor()
  {

  }

  move()
  {

  }

  show()
  {

  }
```

```
}
```

Notes

As you may be aware, we won't see anything yet.

Challenge

Can you guess how we might create a cube where we click on the canvas?

Sketch D7.3 three arguments

We are going to create a new cube at the place where we point the mouse and click. The cube will have three arguments. The x position (mouseX), the y position (mouseY), and the z position (0). So in the constructor, we will have x, y, and z, and when we click on the canvas, we pass on those values into the array as a new object. When we create a new cube, we push it into the array (cubes[]).

```
let cubes = []
let cube

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

function mousePressed()
{
  cube = new Cube(mouseX, mouseY, 0)
  cubes.push(cube)
}

function draw()
{
  background(220)
}

class Cube
{
  constructor(x, y, z)
  {
    this.x = x
    this.y = y
    this.x = z
  }

  move()
  {

  }
```

```
  show()
  {

  }
}
```

Notes

We create a new cube and add it to the array (nothing to see, though).

Sketch D7.4 drawing the cube

Nothing will happen until we draw the cube, so in the show() function we will first have to translate to where we click on the canvas and then draw the box, but we need to loop through the array in draw(). You will notice something odd when you click on the canvas; the cube isn't where it should be. We will rectify this.

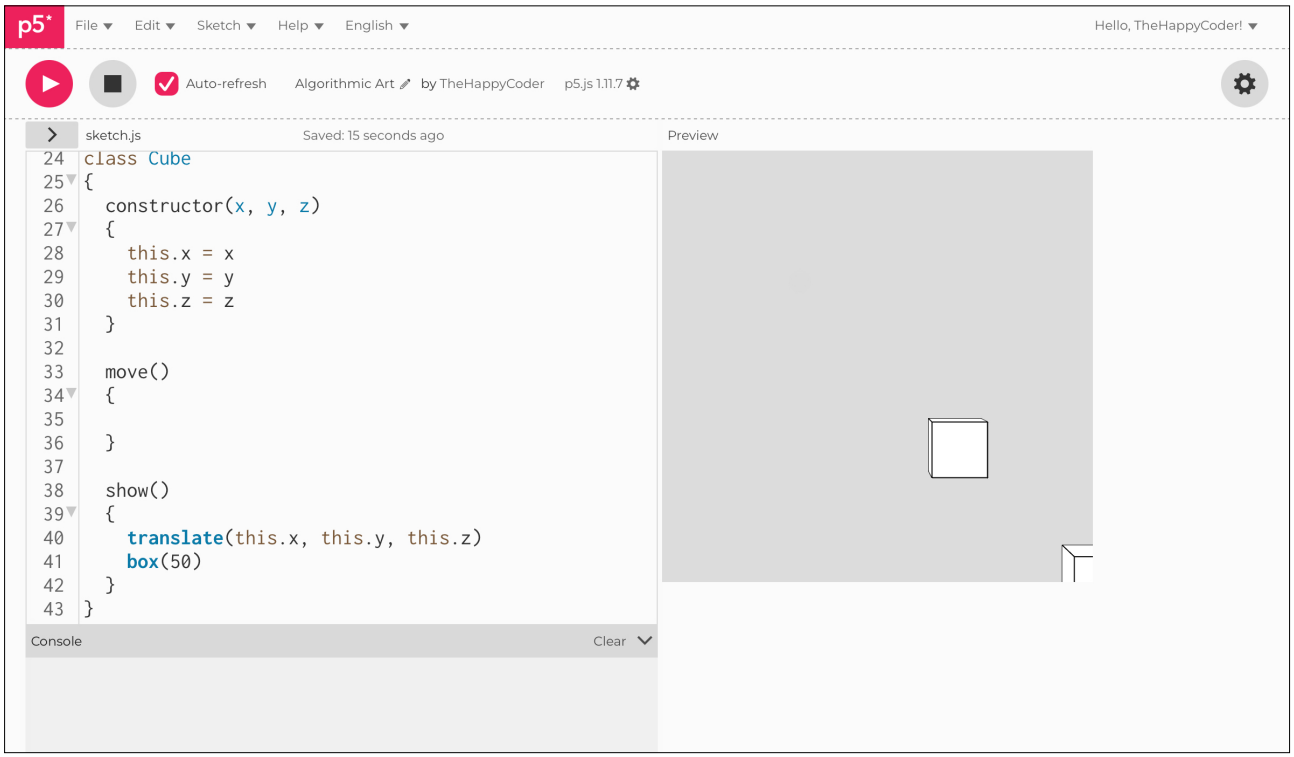
```
let cubes = []
let cube

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

function mousePressed()
{
  cube = new Cube(mouseX, mouseY, 0)
  cubes.push(cube)
}

function draw()
{
  background(220)
  for (let i = 0; i < cubes.length; i++)
  {
    cubes[i].show()
  }
}

class Cube
{
  constructor(x, y, z)
  {
    this.x = x
    this.y = y
    this.z = z
  }
```

```
  move()
  {

  }

  show()
  {
    translate(this.x, this.y, this.z)
    box(50)
  }
}
```

Notes

We can call the length of the cubes array as it grows.

Figure D7.4



Sketch D7.5 translation

Let's rectify this because the origin is at the centre of the canvas rather than the top left; we have to take that into account. Also, when we translate, we need to only translate for that particular `cube`; otherwise, we start to add the translation, and things disappear very quickly, so we use `push()` and `pop()`. Try without to see the difference.

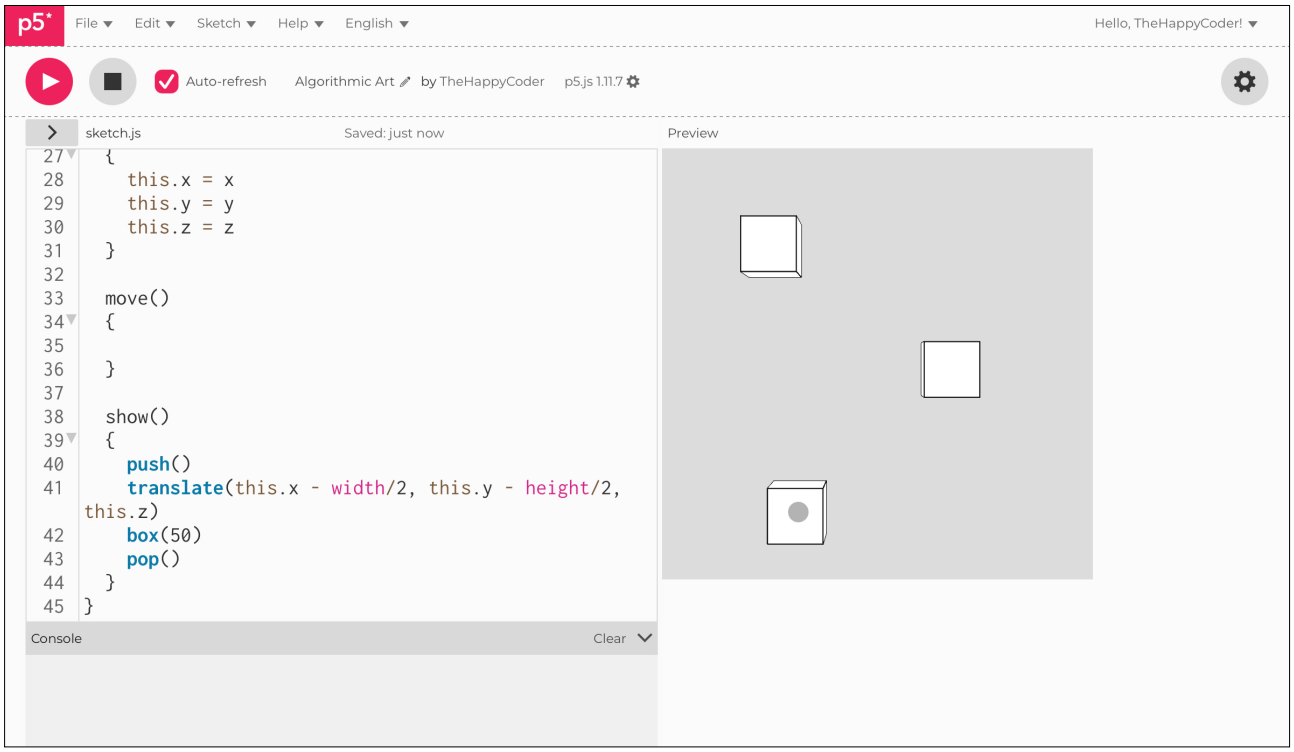
```
let cubes = []
let cube

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

function mousePressed()
{
  cube = new Cube(mouseX, mouseY, 0)
  cubes.push(cube)
}

function draw()
{
  background(220)
  for (let i = 0; i < cubes.length; i++)
  {
    cubes[i].show()
  }
}

class Cube
{
  constructor(x, y, z)
  {
    this.x = x
    this.y = y
    this.z = z
  }
```

```
  move()
  {

  }

  show()
  {
    push()
    translate(this.x - width/2, this.y - height/2, this.z)
    box(50)
    pop()
  }
}
```

Notes

That's better! Wherever you click on the canvas, you should see the cube appear at that position. Next, we will look at rotating each one individually.

Figure D7.5



Sketch D7.6 adding an angle

Using the same sketch, we add in a new `angle` variable, initialise it to `0`, and change the angle mode from radians to degrees.

```
let cubes = []
let cube
let angle = 0

function setup()
{
  createCanvas(400, 400, WEBGL)
  angleMode(DEGREES)
}

function mousePressed()
{
  cube = new Cube(mouseX, mouseY, 0)
  cubes.push(cube)
}

function draw()
{
  background(220)
  for (let i = 0; i < cubes.length; i++)
  {
    cubes[i].show()
  }
}

class Cube
{
  constructor(x, y, z)
  {
    this.x = x
    this.y = y
    this.z = z
  }
```

```
  move()
  {

  }

  show()
  {
    push()
    translate(this.x - width/2, this.y - height/2, this.z)
    box(50)
    pop()
  }
}
```

Notes

No appreciable change yet.

Sketch D7.7 angle to the object

Next, we need to add an angle to the cube object.

```
let cubes = []
let cube
let angle = 0

function setup()
{
  createCanvas(400, 400, WEBGL)
  angleMode(DEGREES)
}

function mousePressed()
{
  cube = new Cube(mouseX, mouseY, 0, angle)
  cubes.push(cube)
}

function draw()
{
  background(220)
  for (let i = 0; i < cubes.length; i++)
  {
    cubes[i].show()
  }
}

class Cube
{
  constructor(x, y, z, angle)
  {
    this.x = x
    this.y = y
    this.z = z
    this.angle = angle
  }
```

```
  move()
  {

  }

  show()
  {
    push()
    translate(this.x - width/2, this.y - height/2, this.z)
    box(50)
    pop()
  }
}
```

Notes

Still nothing new to see yet!

Sketch D7.8 rotating in all directions

We want to rotate along the x, y, and z axes.

```
let cubes = []
let cube
let angle = 0

function setup()
{
  createCanvas(400, 400, WEBGL)
  angleMode(DEGREES)
}

function mousePressed()
{
  cube = new Cube(mouseX, mouseY, 0, angle)
  cubes.push(cube)
}

function draw()
{
  background(220)
  for (let i = 0; i < cubes.length; i++)
  {
    cubes[i].show()
  }
}

class Cube
{
  constructor(x, y, z, angle)
  {
    this.x = x
    this.y = y
    this.z = z
    this.angle = angle
  }
```

```
  move()
  {

  }

  show()
  {
    push()
    translate(this.x - width/2, this.y - height/2, this.z)
    rotateX(this.angle)
    rotateY(this.angle)
    rotateZ(this.angle)
    box(50)
    pop()
  }
}
```

Notes

Getting closer.

Challenge

What do you think we need to do to get the cubes rotating?

Sketch D7.9 incrementing the angle

Now we need to increment the `angle` by one degree in the `move()` function and add it into `draw()`. You should have nice rotating boxes or cubes.

```
let cubes = []
let cube
let angle = 0

function setup()
{
  createCanvas(400, 400, WEBGL)
  angleMode(DEGREES)
}

function mousePressed()
{
  cube = new Cube(mouseX, mouseY, 0, angle)
  cubes.push(cube)
}

function draw()
{
  background(220)
  for (let i = 0; i < cubes.length; i++)
  {
    cubes[i].show()
    cubes[i].move()
  }
}

class Cube
{
  constructor(x, y, z, angle)
  {
    this.x = x
    this.y = y
    this.z = z
    this.angle = angle
```

```
  }

  move()
  {
    this.angle++
  }

  show()
  {
    push()
    translate(this.x - width/2, this.y - height/2, this.z)
    rotateX(this.angle)
    rotateY(this.angle)
    rotateZ(this.angle)
    box(50)
    pop()
  }
}
```

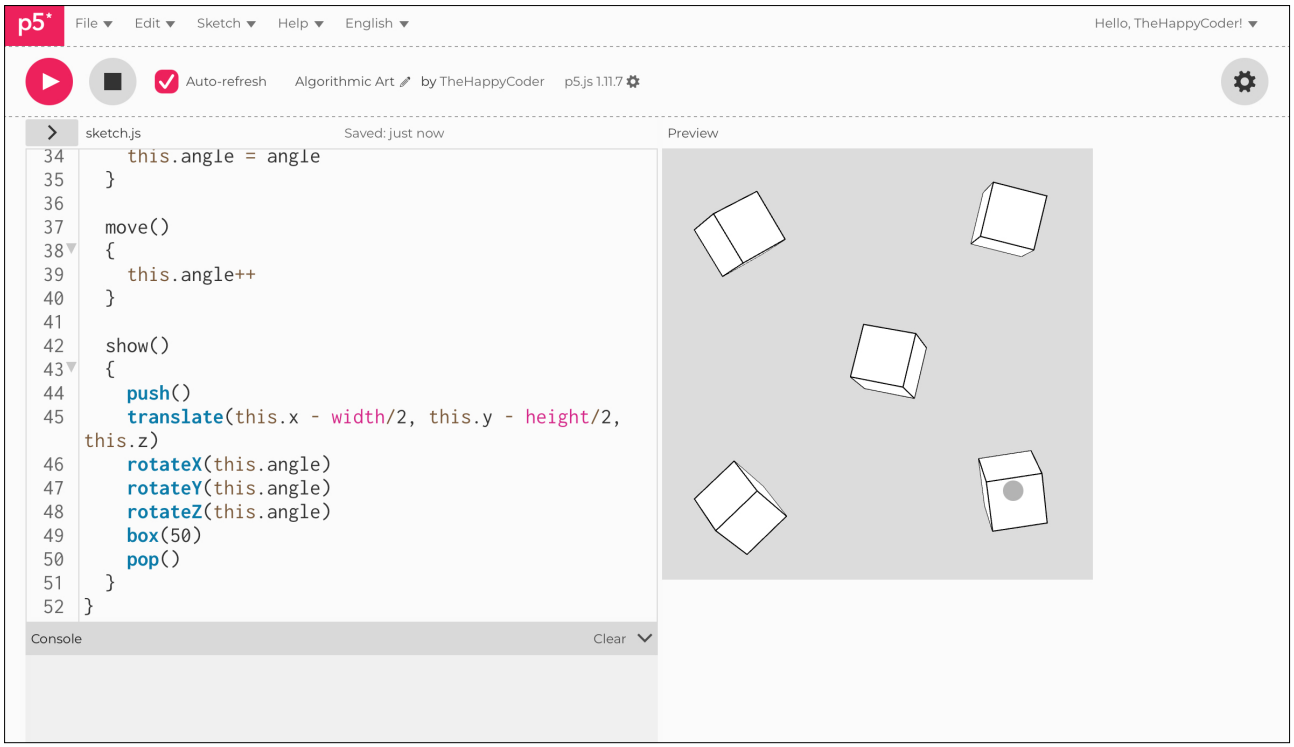
Notes

Now we have it, each one rotates individually.

Challenges

1. Add materials, lights, and their own colour.
2. Create different-sized cubes.
3. Use `mouseDragged()`.

Figure D7.9



The Joy of Coding Algorithmic Art

Module D Unit #8 3D arrays part 2

Module D Unit #8: 3D arrays part 2

We are going to develop further the concept of a 3D array and create some interesting shapes and patterns. First off is a cube of cubes, and then we will have a spiral of bubbles!

Sketch D8.1 an array of cubes

! Using much of the previous sketch.

Starting a new sketch with many of the previous elements. Making an array of cubes where we have 0 to 100 in steps of 20 for a cube size of 20, giving us five cubes. We do this with a simple for() loop.

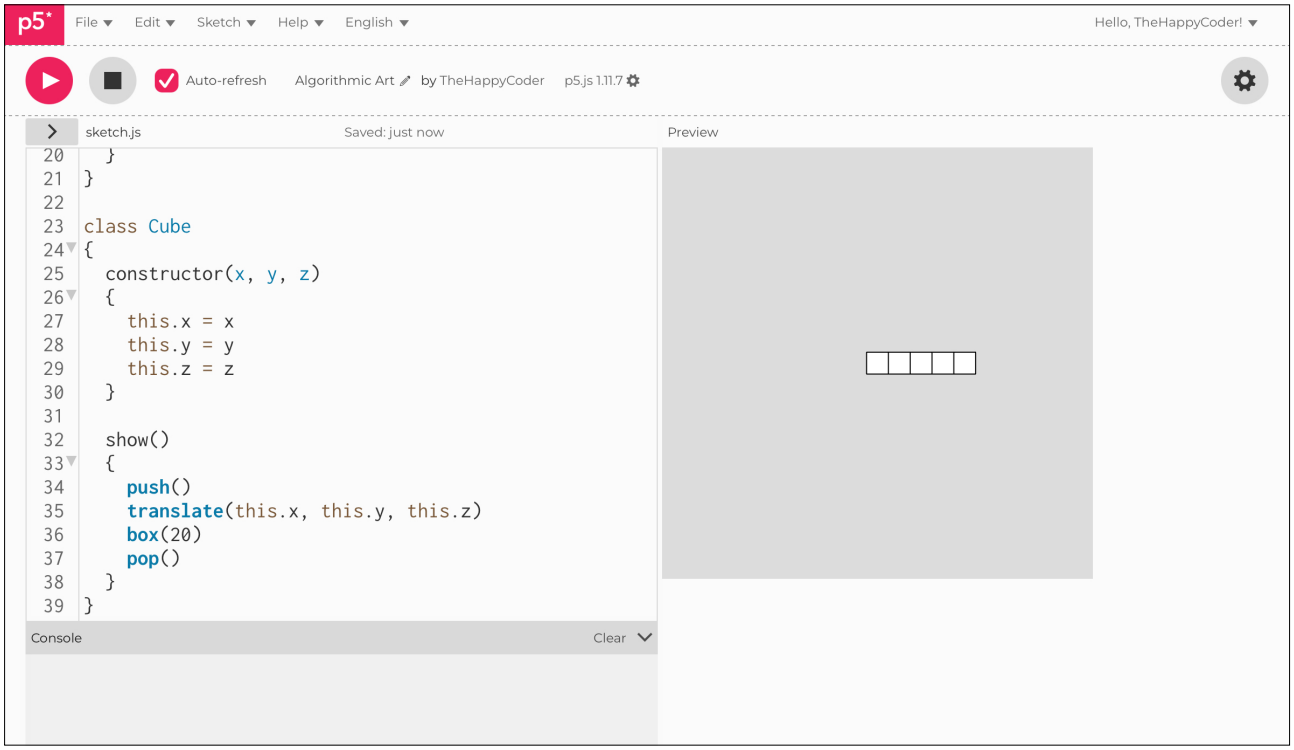
```
let cubes = []
let cube

function setup()
{
  createCanvas(400, 400, WEBGL)
  for (let i = 0; i < 100; i += 20)
  {
    cube = new Cube(i, 0, 0)
    cubes.push(cube)
  }
}

function draw()
{
  background(220)
  for (let i = 0; i < cubes.length; i++)
  {
    cubes[i].show()
  }
}

class Cube
{
  constructor(x, y, z)
  {
    this.x = x
    this.y = y
    this.z = z
  }
```

```
  show()
  {
    push()
    translate(this.x, this.y, this.z)
    box(20)
    pop()
  }
}
```

Notes

As you can see, we have a problem. We have drawn the cubes from `(0, 0)`, which is the centre of the canvas. We need to start at `-100` rather than `0`. Also, we can only see from the front; it would be nice to rotate the array of cubes to see them properly.

Figure D8.1



Sketch D8.2 rotating the array

Now we rotate the array about the centre of the canvas. Also, we have now got ten cubes.

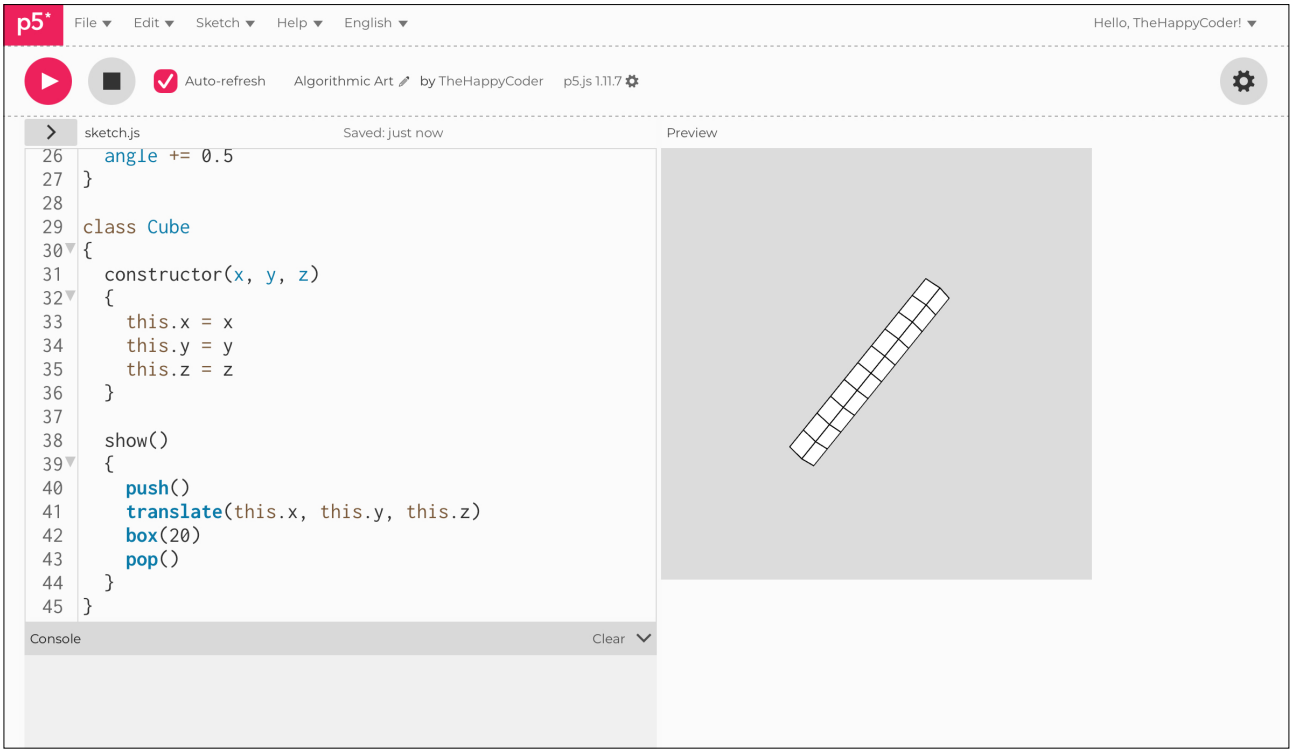
```
let cubes = []
let cube
let angle = 0

function setup()
{
  createCanvas(400, 400, WEBGL)
  angleMode(DEGREES)
  for (let i = -100; i < 100; i += 20)
  {
    cube = new Cube(i, 0, 0)
    cubes.push(cube)
  }
}

function draw()
{
  background(220)
  rotateX(angle)
  rotateY(angle)
  rotateZ(angle)
  for (let i = 0; i < cubes.length; i++)
  {
    cubes[i].show()
  }
  angle += 0.5
}

class Cube
{
  constructor(x, y, z)
  {
    this.x = x
    this.y = y
```

```
    this.z = z
  }

  show()
  {
    push()
    translate(this.x, this.y, this.z)
    box(20)
    pop()
  }
}
```

Notes

A long stick of cubes or a long cuboid, rotating.

Figure D8.2



Sketch D8.3 spaced out

If we space them out a little bit more by increasing the steps from 20 to 25, this will also mean that we have only eight cubes.

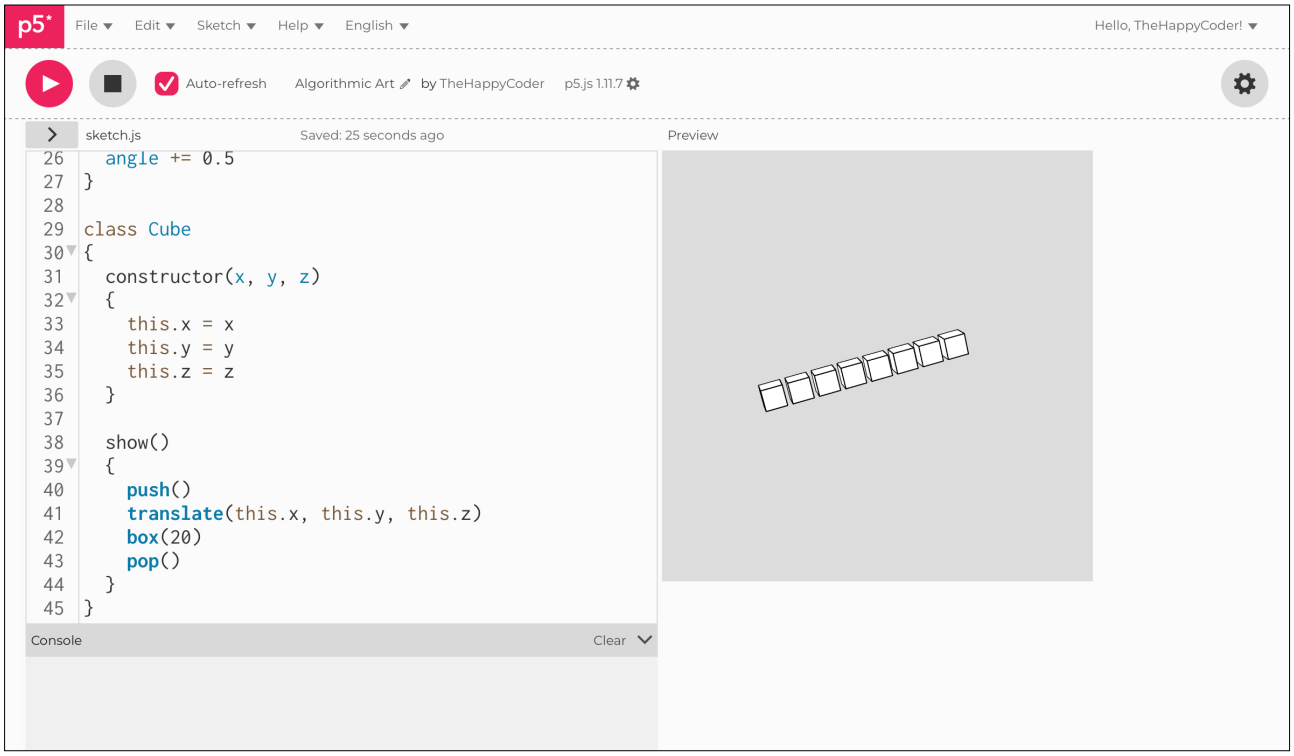
```
let cubes = []
let cube
let angle = 0

function setup()
{
  createCanvas(400, 400, WEBGL)
  angleMode(DEGREES)
  for (let i = -100; i < 100; i += 25)
  {
    cube = new Cube(i, 0, 0)
    cubes.push(cube)
  }
}

function draw()
{
  background(220)
  rotateX(angle)
  rotateY(angle)
  rotateZ(angle)
  for (let i = 0; i < cubes.length; i++)
  {
    cubes[i].show()
  }
  angle += 0.5
}

class Cube
{
  constructor(x, y, z)
  {
    this.x = x
    this.y = y
```

```
    this.z = z
  }

  show()
  {
    push()
    translate(this.x, this.y, this.z)
    box(20)
    pop()
  }
}
```

Notes

A bit more spaced out.

Figure D8.3



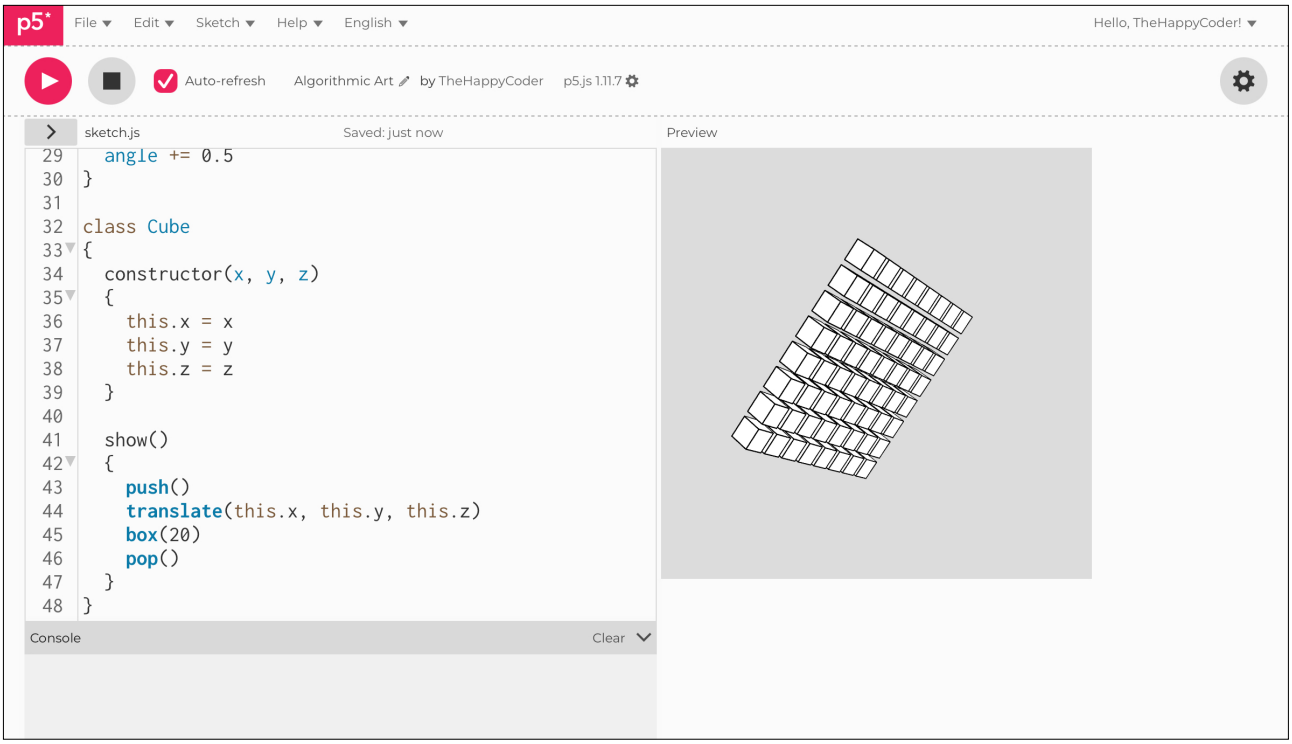
Sketch D8.4 an array in the y direction

Yet we can do even better because we can create an array in the y direction also. For that, we need a nested loop.

```
let cubes = []
let cube
let angle = 0

function setup()
{
  createCanvas(400, 400, WEBGL)
  angleMode(DEGREES)
  for (let i = -100; i < 100; i += 25)
  {
    for (let j = -100; j < 100; j += 25)
    {
      cube = new Cube(i, j, 0)
      cubes.push(cube)
    }
  }
}

function draw()
{
  background(220)
  rotateX(angle)
  rotateY(angle)
  rotateZ(angle)
  for (let i = 0; i < cubes.length; i++)
  {
    cubes[i].show()
  }
  angle += 0.5
}

class Cube
{
  constructor(x, y, z)
```

```
  {
    this.x = x
    this.y = y
    this.z = z
  }

  show()
  {
    push()
    translate(this.x, this.y, this.z)
    box(20)
    pop()
  }
}
```

Figure D8.4



Sketch D8.5 a cube of cubes

But why stop there? Let's add the third dimension, the z direction; we just add another nested loop.

```
let cubes = []
let cube
let angle = 0

function setup()
{
  createCanvas(400, 400, WEBGL)
  angleMode(DEGREES)
  for (let i = -100; i < 100; i += 25)
  {
    for (let j = -100; j < 100; j += 25)
    {
      for (let k = -100; k < 100; k += 25)
      {
        cube = new Cube(i, j, k)
        cubes.push(cube)
      }
    }
  }
}

function draw()
{
  background(220)
  rotateX(angle)
  rotateY(angle)
  rotateZ(angle)
  for (let i = 0; i < cubes.length; i++)
  {
    cubes[i].show()
  }
  angle += 0.5
}
```

```
class Cube
{
  constructor(x, y, z)
  {
    this.x = x
    this.y = y
    this.z = z
  }

  show()
  {
    push()
    translate(this.x, this.y, this.z)
    box(20)
    pop()
  }
}
```

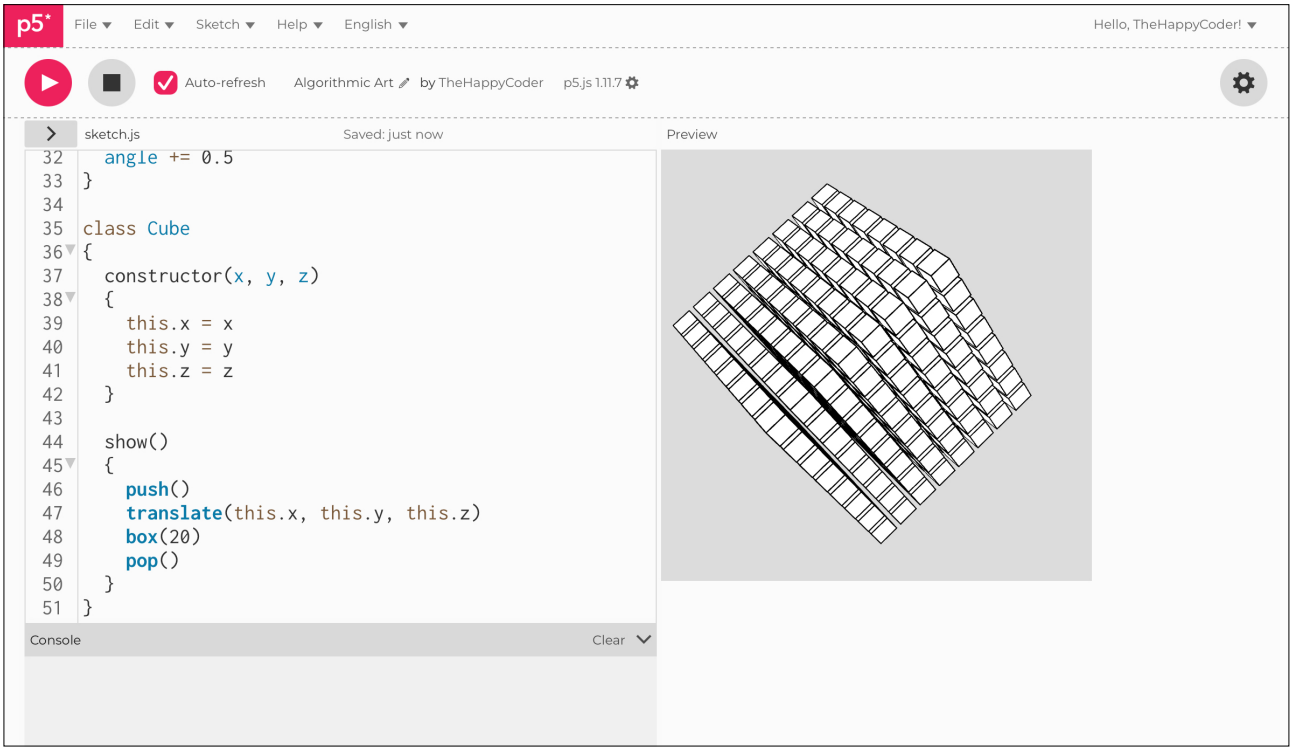
Notes

Inside the `cubes[]` array, we have a 3D array.

Challenge

Have a look inside the `cubes[]` array with `console.log()`.

Figure D8.5



Sketch D8.6 adding a splash of colour

We can add individual colours (random) to each cube. We use a function called `color(a, b, c)` with three arguments for the red, green, and blue. We add a variable (`c`) to carry this information to the new cube being created.

```
let cubes = []
let cube
let angle = 0

function setup()
{
  createCanvas(400, 400, WEBGL)
  angleMode(DEGREES)
  for (let i = -100; i < 100; i += 25)
  {
    for (let j = -100; j < 100; j += 25)
    {
      for (let k = -100; k < 100; k += 25)
      {
        let c = color(random(255), random(255), random(255))
        cube = new Cube(i, j, k, c)
        cubes.push(cube)
      }
    }
  }
}

function draw()
{
  background(220)
  rotateX(angle)
  rotateY(angle)
  rotateZ(angle)
  for (let i = 0; i < cubes.length; i++)
  {
    cubes[i].show()
  }
  angle += 0.5
```

```
}

class Cube
{
  constructor(x, y, z, c)
  {
    this.x = x
    this.y = y
    this.z = z
    this.c = c
  }

  show()
  {
    push()
    fill(this.c)
    translate(this.x, this.y, this.z)
    box(20)
    pop()
  }
}
```

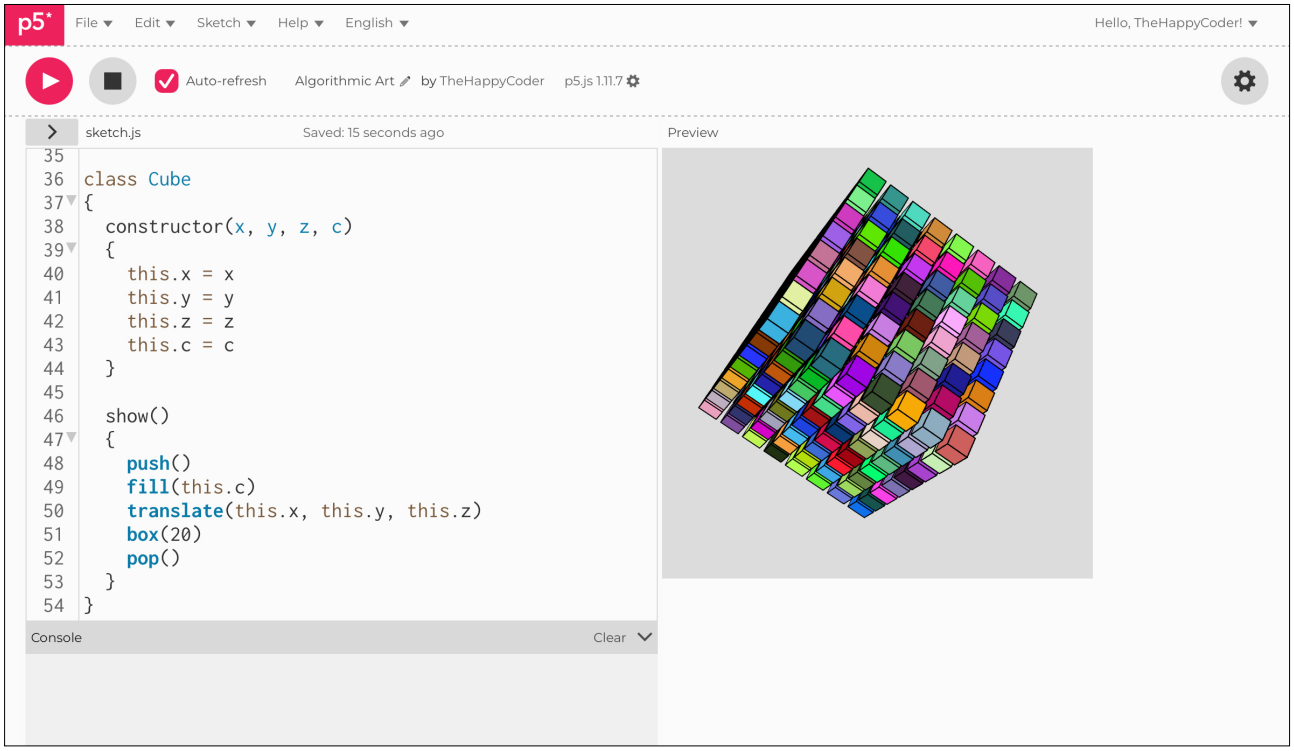
Notes

We give each individual cube its own colour.

Challenges

1. Add some `alpha` (fourth argument) to the colour to make it transparent and remove the `stroke()`.
2. Change the shape to a sphere or cylinder.

Figure D8.6



Sketch D8.7 a bubble factory

A bit of refactoring. We are going to use a sphere, so we will change the name of the class, variable, and array to something more appropriate and familiar: `bubble`. Also added `sphere(10)`, `lights()`, and `noStroke()`.

```
let bubbles = []
let bubble
let angle = 0

function setup()
{
  createCanvas(400, 400, WEBGL)
  angleMode(DEGREES)
  for (let i = -100; i < 100; i += 25)
  {
    for (let j = -100; j < 100; j += 25)
    {
      for (let k = -100; k < 100; k += 25)
      {
        let c = color(random(255), random(255), random(255))
        bubble = new Bubble(i, j, k, c)
        bubbles.push(bubble)
      }
    }
  }
}

function draw()
{
  background(220)
  rotateX(angle)
  rotateY(angle)
  rotateZ(angle)
  for (let i = 0; i < bubbles.length; i++)
  {
    bubbles[i].show()
  }
  angle += 0.5
```

```
}


class Bubble
{
  constructor(x, y, z, c)
  {
    this.x = x
    this.y = y
    this.z = z
    this.c = c
  }

  show()
  {
    push()
    fill(this.c)
    translate(this.x, this.y, this.z)
    lights()
    noStroke()
    sphere(10)
    pop()
  }
}
```

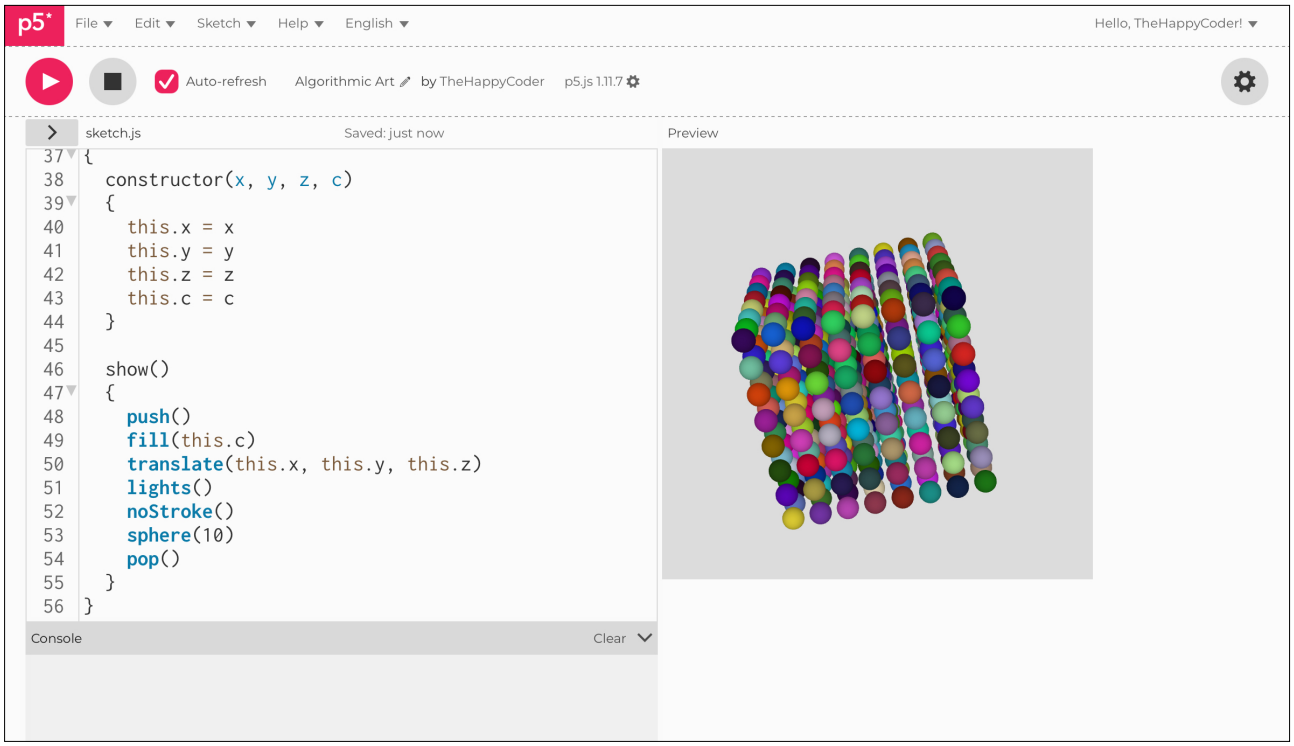
Notes

The `noStroke()` is just so that you can see the colours.

Challenge

Call the variable, class, and array anything you want.

Figure D8.7



Sketch D8.8 not a cube

Our next step is putting all those bubbles in a circle rather than a cube, but first remove all the code highlighted and commented, then change the `constructor()` function.

```
let bubbles = []
let bubble
let angle = 0

function setup()
{
  createCanvas(400, 400, WEBGL)
  angleMode(DEGREES)
  // for (let i = -100; i < 100; i += 25)
  // {
  //   for (let j = -100; j < 100; j += 25)
  //   {
  //     for (let k = -100; k < 100; k += 25)
  //     {
  //       let c = color(random(255), random(255), random(255))
  //       bubble = new Bubble(i, j, k, c)
  //       bubbles.push(bubble)
  //     }
  //   }
  // }
}

function draw()
{
  background(220)
  rotateX(angle)
  rotateY(angle)
  rotateZ(angle)
  for (let i = 0; i < bubbles.length; i++)
  {
    bubbles[i].show()
  }
  angle += 0.5
```

```
}

class Bubble
{
  constructor(x, y, z)
  {
    this.x = x
    this.y = y
    this.z = z
    // this.c = c
  }

  show()
  {
    push()
   // fill(this.c)
    translate(this.x, this.y, this.z)
    lights()
    noStroke()
    sphere(10)
    pop()
  }
}
```

Notes

Don't try running this just yet!

Sketch D8.9 now a circle

Creating a circle of bubbles.

```
let bubbles = []
let bubble
let angle = 0

function setup()
{
  createCanvas(400, 400, WEBGL)
  angleMode(DEGREES)
  for (let i = 0; i < 360; i += 30)
  {
    let x = 100 * sin(i)
    let y = 100 * cos(i)
    bubble = new Bubble(x, y, 0)
    bubbles.push(bubble)
  }
}

function draw()
{
  background(220)
  rotateX(angle)
  rotateY(angle)
  rotateZ(angle)
  for (let i = 0; i < bubbles.length; i++)
  {
    bubbles[i].show()
  }
  angle += 0.5
}

class Bubble
{
  constructor(x, y, z)
  {
    this.x = x
```

```
    this.y = y
    this.z = z
  }

  show()
  {
    push()
    translate(this.x, this.y, this.z)
    lights()
    fill('yellow')
    noStroke()
    sphere(10)
    pop()
  }
}
```

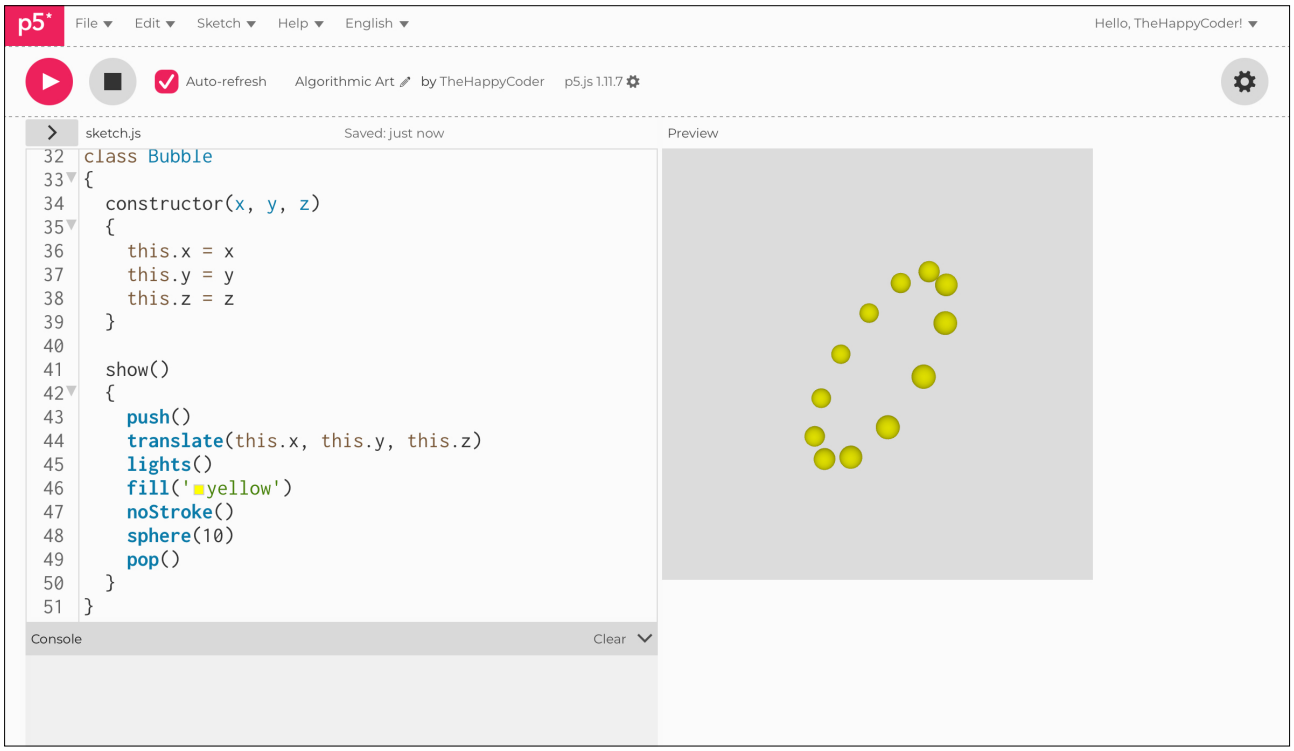
Notes

We have a nice ring of spheres.

Challenge

Add other lights or materials.

Figure D8.9



Sketch D8.10 creating a spiral

Next, let's try to make a spiral so that the array of bubbles also has a `z` component.

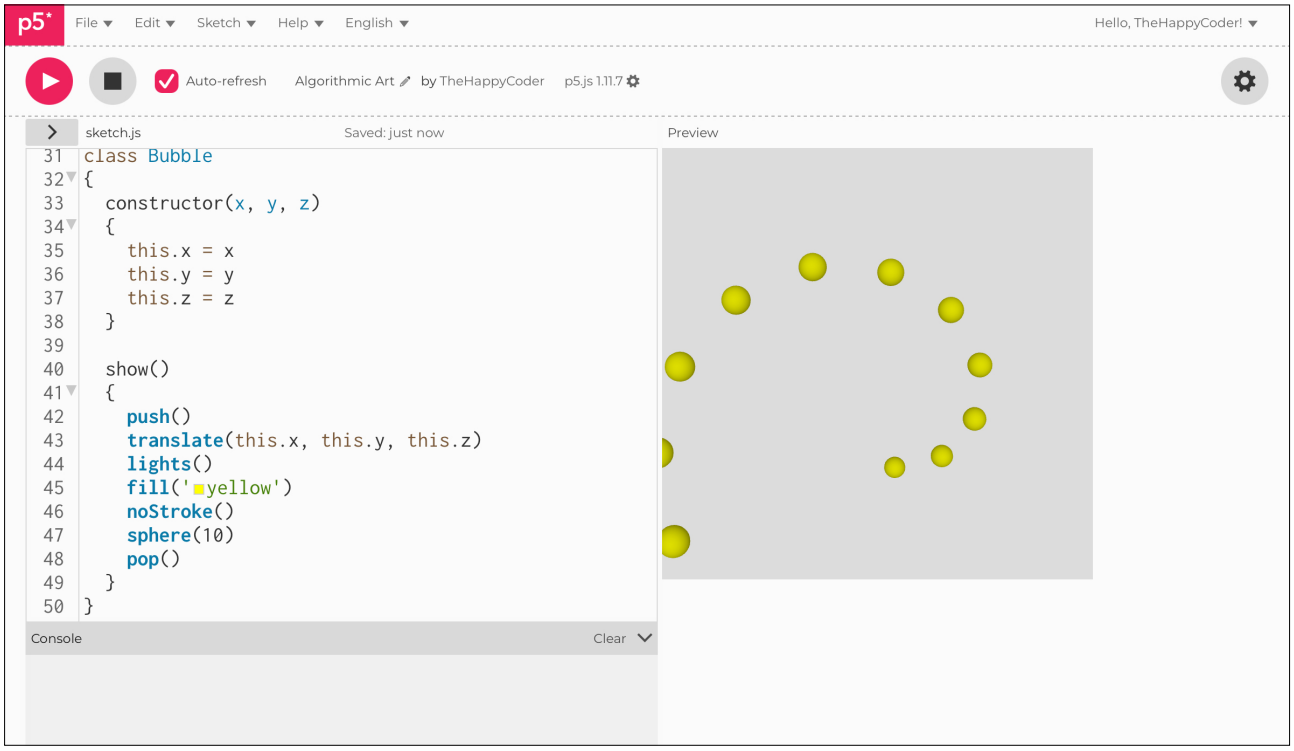
```
let bubbles = []
let bubble
let angle = 0

function setup()
{
  createCanvas(400, 400, WEBGL)
  angleMode(DEGREES)
  for (let i = 0; i < 360; i += 30)
  {
    let x = 100 * sin(i)
    let y = 100 * cos(i)
    bubble = new Bubble(x, y, i)
    bubbles.push(bubble)
  }
}

function draw()
{
  background(220)
  rotateX(angle)
  rotateY(angle)
  rotateZ(angle)
  for (let i = 0; i < bubbles.length; i++)
  {
    bubbles[i].show()
  }
  angle += 0.5
}

class Bubble
{
  constructor(x, y, z)
  {
    this.x = x
```

```
    this.y = y
    this.z = z
  }

  show()
  {
    push()
    translate(this.x, this.y, this.z)
    lights()
    fill('yellow')
    noStroke()
    sphere(10)
    pop()
  }
}
```

Notes

It's a spiral, yes, but let's see if we can improve it.

Figure D8.10



Sketch D8.11 more spirally

We can make some improvements to this to make it more of a spiral.

```
let bubbles = []
let bubble
let angle = 0

function setup()
{
  createCanvas(400, 400, WEBGL)
  angleMode(DEGREES)
  for (let i = -360; i < 360; i += 10)
  {
    let x = 100 * sin(i)
    let y = 100 * cos(i)
    bubble = new Bubble(x, y, i/4)
    bubbles.push(bubble)
  }
}

function draw()
{
  background(220)
  rotateX(angle)
  rotateY(angle)
  rotateZ(angle)
  for (let i = 0; i < bubbles.length; i++)
  {
    bubbles[i].show()
  }
  angle += 0.5
}

class Bubble
{
  constructor(x, y, z)
  {
    this.x = x
```

```
    this.y = y
    this.z = z
  }

  show()
  {
    push()
    translate(this.x, this.y, this.z)
    lights()
    fill('yellow')
    noStroke()
    sphere(10)
    pop()
  }
}
```

Notes

Much better.

Challenges

1. Can you randomise the diameter of the sphere?
2. Add random colours to the spheres.
3. Change the radius of the spiral so that it starts small and increases.
4. How would you extend it?
5. How would you rotate it about an axis?

Figure D8.11