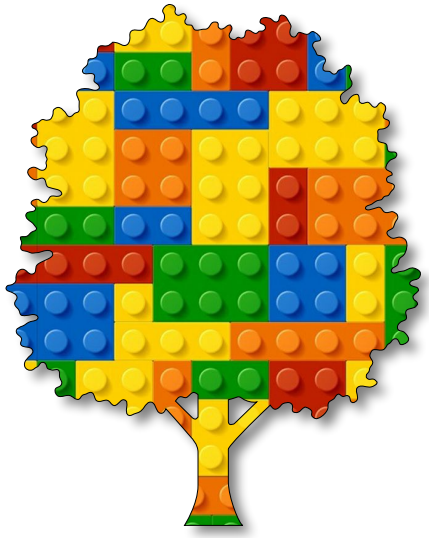
Algorithmic Art

Module A

Unit #2

your first circle



Contents

Module A Unit #2 your first circle

Introduction to your first circle

This may not seem very challenging, but using the `circle()` function means I can introduce you to many coding concepts that you will use in your journey into creative coding.

Key concepts:

- drawing a circle
- names of colours
- variables
- random
- operators
- while() loop
- for() loop

Sketch A2.1 your first sketch

First of all, you need to delete the default code and type the following. It is really just the same, but I want to arrange it in a more intuitive way. The semi-colons aren't essential, and the first curly bracket {} doesn't have to start on that line (although it is common to do so).

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

function draw()
{
  background('lightgrey')
}
```

🗒️ Notes

Your screen should look something like the image below.

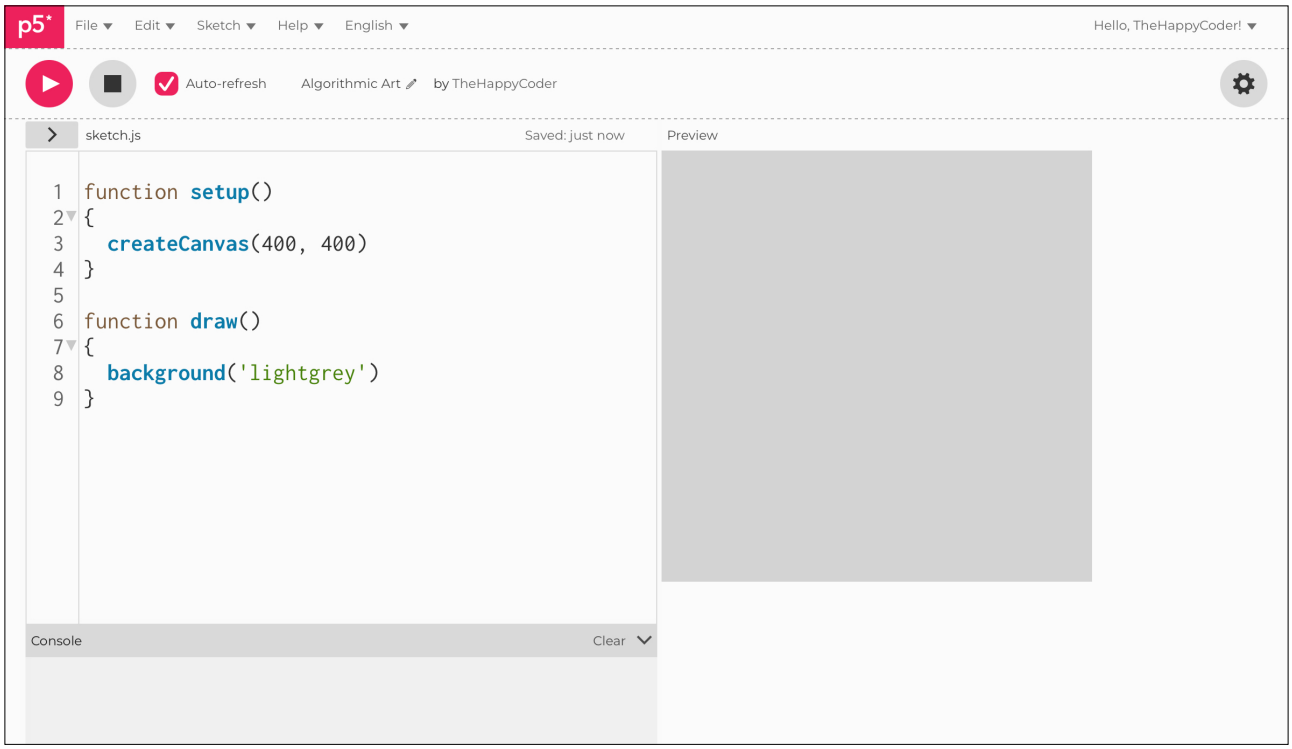
🌻 Challenge

Change the size of the canvas from
`createCanvas(400, 400)` to `createCanvas(600, 200)`

🛠️ Code Explanation

`function setup()`	Everything in the first function only happens once
`{ }`	All the code goes inside the curly brackets (or braces)
`createCanvas(400, 400)`	We are creating a canvas just like an artist ready to paint. This canvas is 400 pixels wide by 400 pixels high.
`function draw()`	This function operates a continuous loop
`background('lightgrey')`	We give the background a colour. In this instance it is a light grey colour (more on colours later)

Figure A2.1


p5*
File
Edit
Sketch
Help
English
Hello, TheHappyCoder!
Auto-refresh
Algorithmic Art
by TheHappyCoder
sketch.js
Saved: just now
Preview
1 function setup()
2 {
3 createCanvas(400, 400)
4 }
5
6 function draw()
7 {
8 background('lightgrey')
9 }
Console
Clear

Sketch A2.2 a circle

When we add or change a line (or lines) of code, they will be highlighted blue. We are going to add your first shape, a circle.

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

function draw()
{
  background('lightgrey')
  circle(200, 200, 100)
}
```

🗒️ Notes

Because the background and circle are in the `draw()` function, the programme code is drawing them continuously. First, the background, and then the circle.

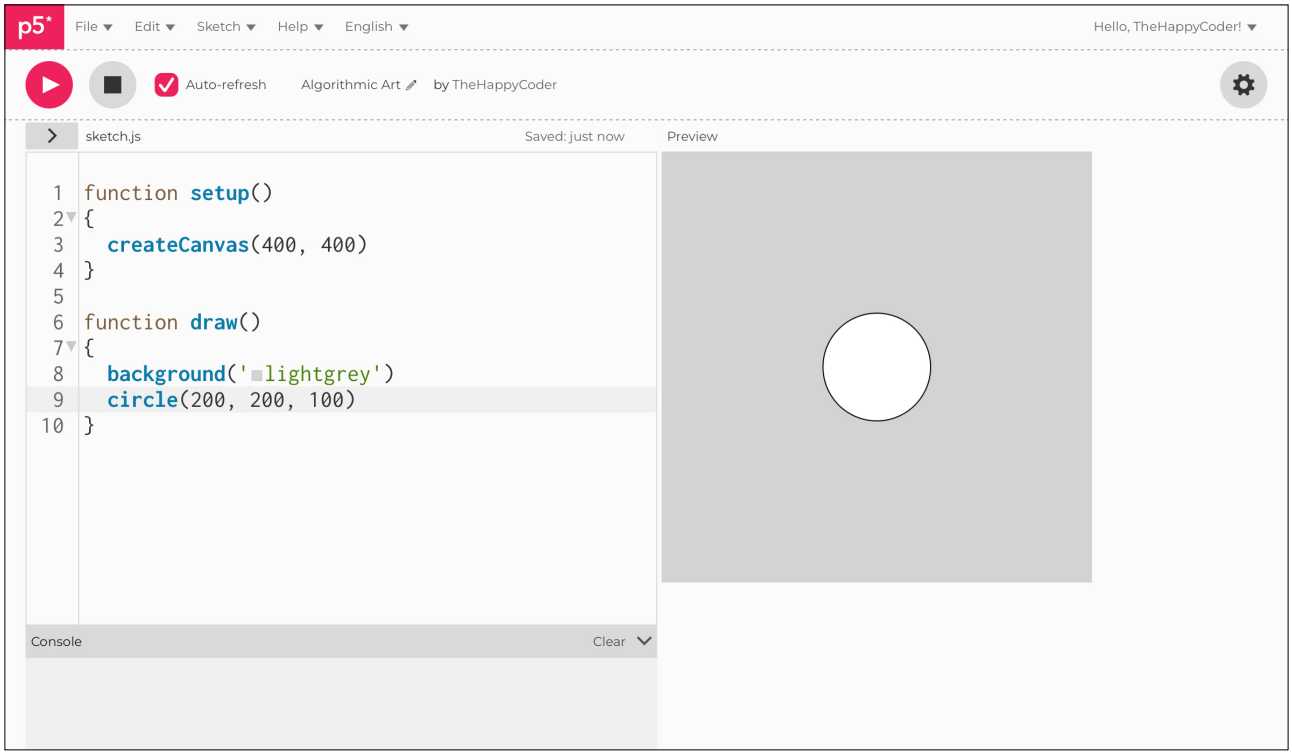
🌻 Challenges

1. Change the co-ordinates from: `circle(200, 200, 100)` to `circle(100, 300, 100)`
2. Change the circle diameter from: `circle(100, 300, 100)` to `circle(100, 300, 50)`

🛠️ Code Explanation

`circle(200, 200, 100)`	The centre of the circle is at position 200 pixels from the left hand edge of the canvas and 200 pixels from the top of the canvas. The diameter is 100 pixels.

Figure A2.2


p5*
File
Edit
Sketch
Help
English
Hello, TheHappyCoder!
Auto-refresh
Algorithmic Art
by TheHappyCoder
sketch.js
Saved: just now
Preview
1 function setup()
2 {
3 createCanvas(400, 400)
4 }
5
6 function draw()
7 {
8 background('lightgrey')
9 circle(200, 200, 100)
10 }
Console
Clear

Sketch A2.3 adding another circle

We can add more circles (and other shapes)

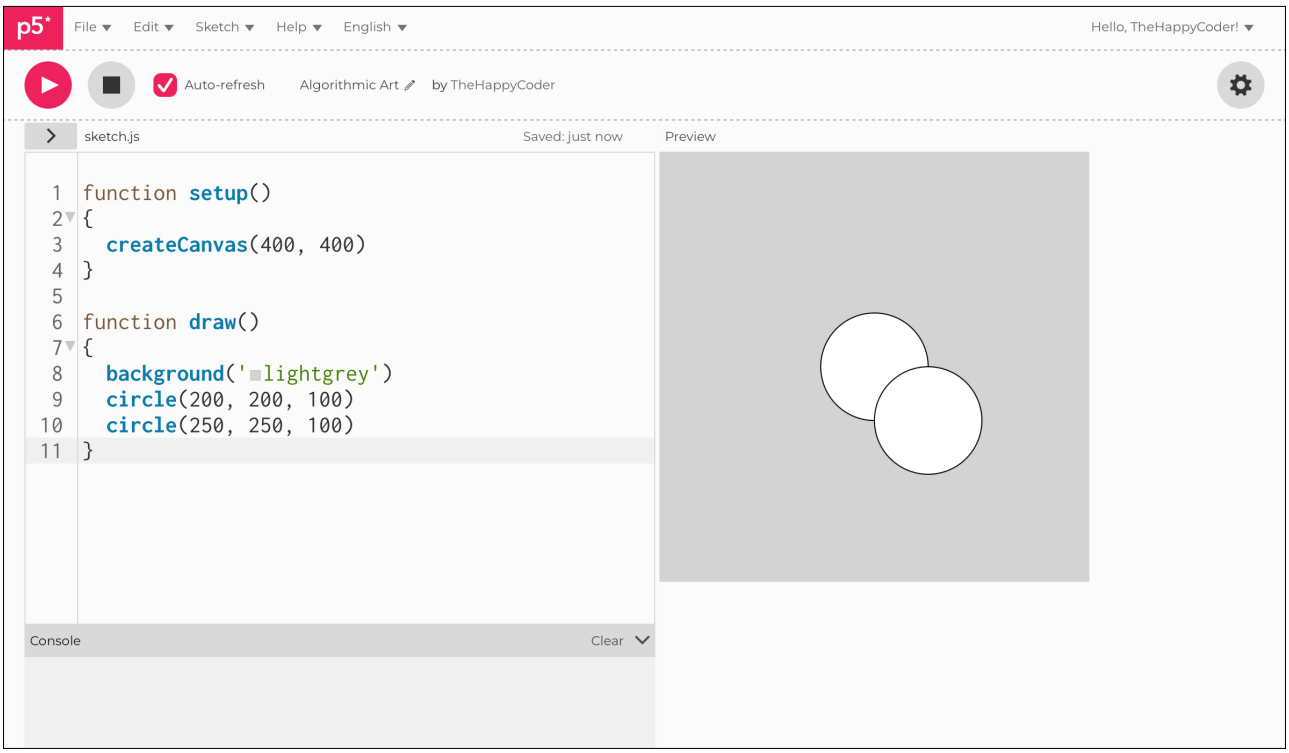
```
function setup()
{
  createCanvas(400, 400)
}

function draw()
{
  background('lightgrey')
  circle(200, 200, 100)
  circle(250, 250, 100)
}
```

Notes

Notice that the second circle overlaps on top of the first circle. The programme works from top to bottom, one line of code at a time. In this instance, it draws the background first, then the first circle, then the second circle, and then repeats.

Figure A2.3



Sketch A2.4 making a simple pattern

We will change the position of the second circle and add three more circles.

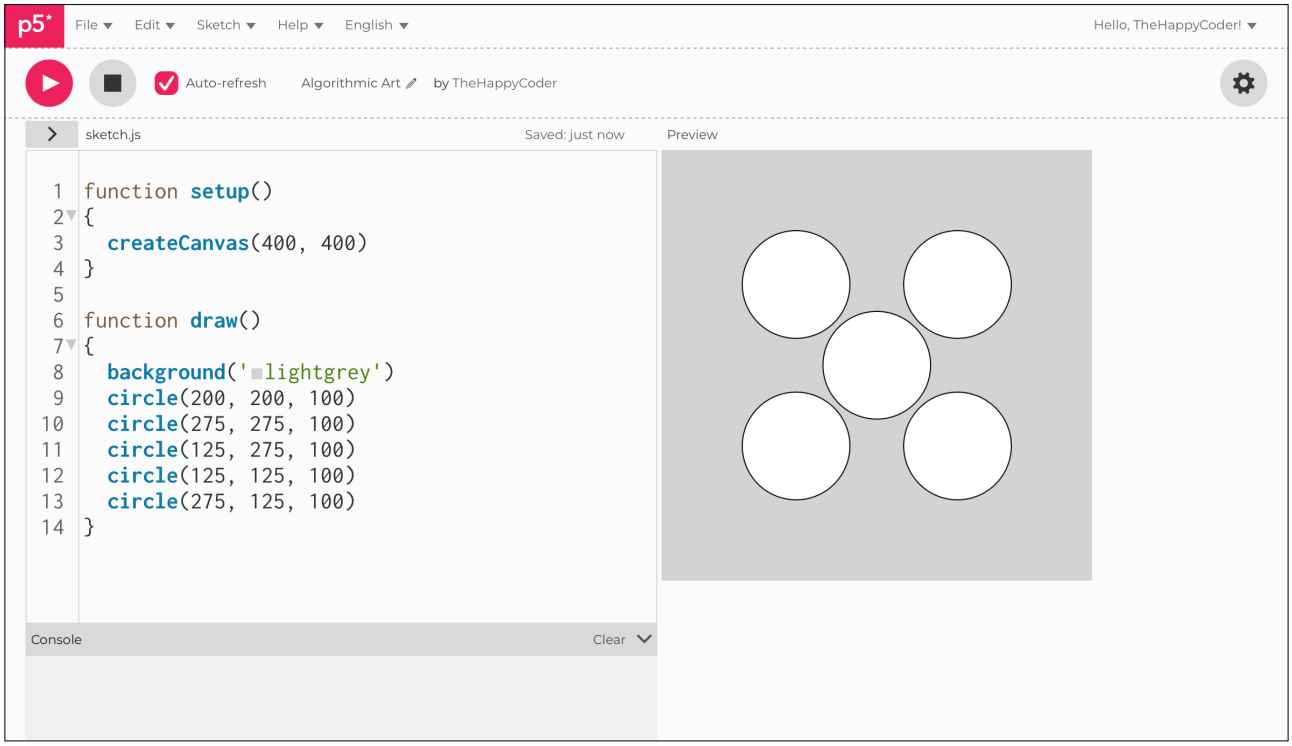
```
function setup()
{
  createCanvas(400, 400)
}

function draw()
{
  background('lightgrey')
  circle(200, 200, 100)
  circle(275, 275, 100)
  circle(125, 275, 100)
  circle(125, 125, 100)
  circle(275, 125, 100)
}
```

Notes

You might notice in the image below that next to `Auto-refresh`, the box is ticked and is highlighted in red. This is useful (as well as dangerous) so that you don't have to keep pressing the run (play) button. It can be dangerous if you are using `for()` loops (more on that later) where you can accidentally get into an infinite loop and crash the programme.

Figure A2.4


p5*
File
Edit
Sketch
Help
English
Hello, TheHappyCoder!
Auto-refresh
Algorithmic Art
by TheHappyCoder
sketch.js
Saved: just now
Preview
1 function setup()
2 {
3 createCanvas(400, 400)
4 }
5
6 function draw()
7 {
8 background('lightgrey')
9 circle(200, 200, 100)
10 circle(275, 275, 100)
11 circle(125, 275, 100)
12 circle(125, 125, 100)
13 circle(275, 125, 100)
14 }
Console
Clear

Sketch A2.5 adding some colour

Before we move onto other shapes and key concepts, we can explore colour. To start with, we can use the names of colours such as `red`, `green`, `blue`, or `orange`, for example (there are more but a limited number). To do this, we use the `fill()` function. First off, let's colour them `red`.

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

function draw()
{
  background('lightgrey')
  fill('red')
  circle(200, 200, 100)
  circle(275, 275, 100)
  circle(125, 275, 100)
  circle(125, 125, 100)
  circle(275, 125, 100)
}
```

🗒️ Notes

You will notice that in the code, you get a little red box. This just illustrates the colour that you are requesting. There are 140 named colours. You can see them all at: https://www.w3schools.com/colors/colors_names.asp or just search for JavaScript colour names

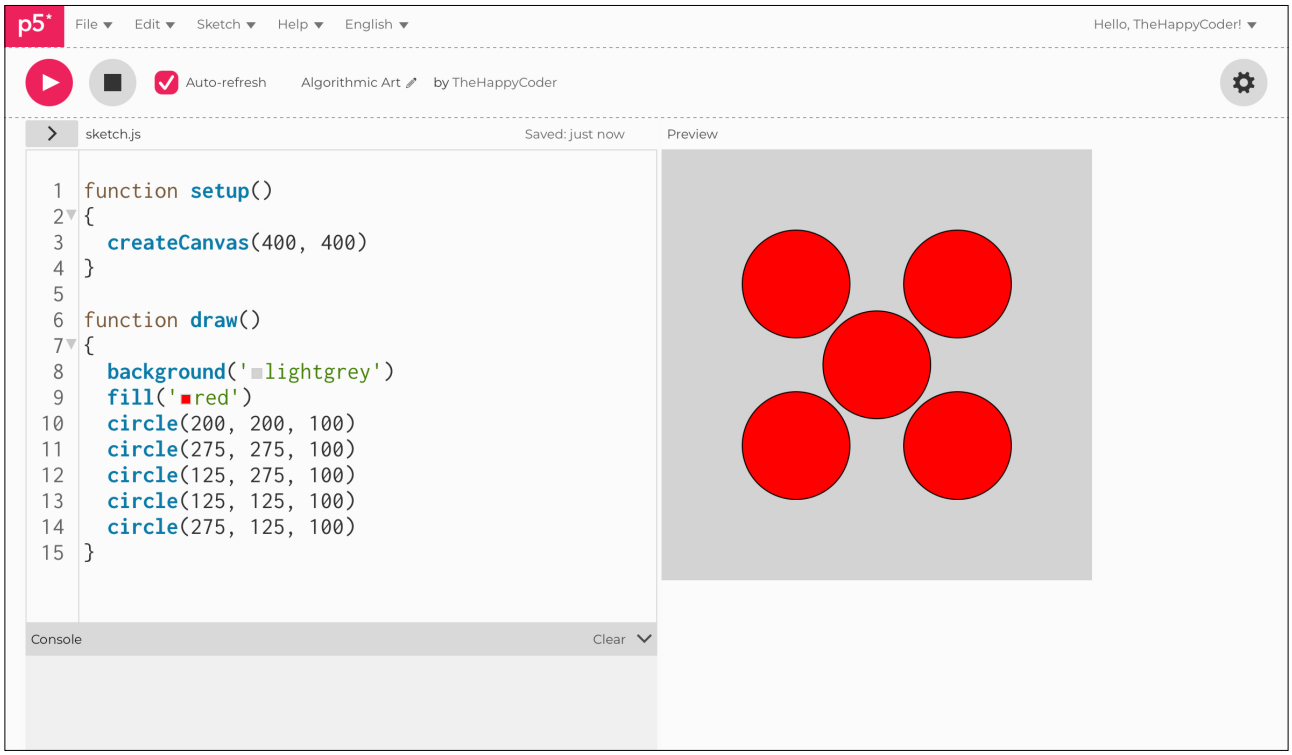
🌻 Challenge

Try other names of colours

🛠️ Code Explanation

`fill('red')`	This is the function to fill any shape any colour you chose. If you use the name of a colour then you have to put it in speech marks (single or double)

Figure A2.5


p5*
File
Edit
Sketch
Help
English
Hello, TheHappyCoder!
Auto-refresh
Algorithmic Art
by TheHappyCoder
sketch.js
Saved: just now
Preview
1 function setup()
2 {
3 createCanvas(400, 400)
4 }
5
6 function draw()
7 {
8 background('lightgrey')
9 fill('red')
10 circle(200, 200, 100)
11 circle(275, 275, 100)
12 circle(125, 275, 100)
13 circle(125, 125, 100)
14 circle(275, 125, 100)
15 }
Console
Clear

Sketch A2.6 different colours

We will give each one a different colour.

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

function draw()
{
  background('lightgrey')
  fill('red')
  circle(200, 200, 100)
  fill('green')
  circle(275, 275, 100)
  fill('blue')
  circle(125, 275, 100)
  fill('yellow')
  circle(125, 125, 100)
  fill('purple')
  circle(275, 125, 100)
}
```

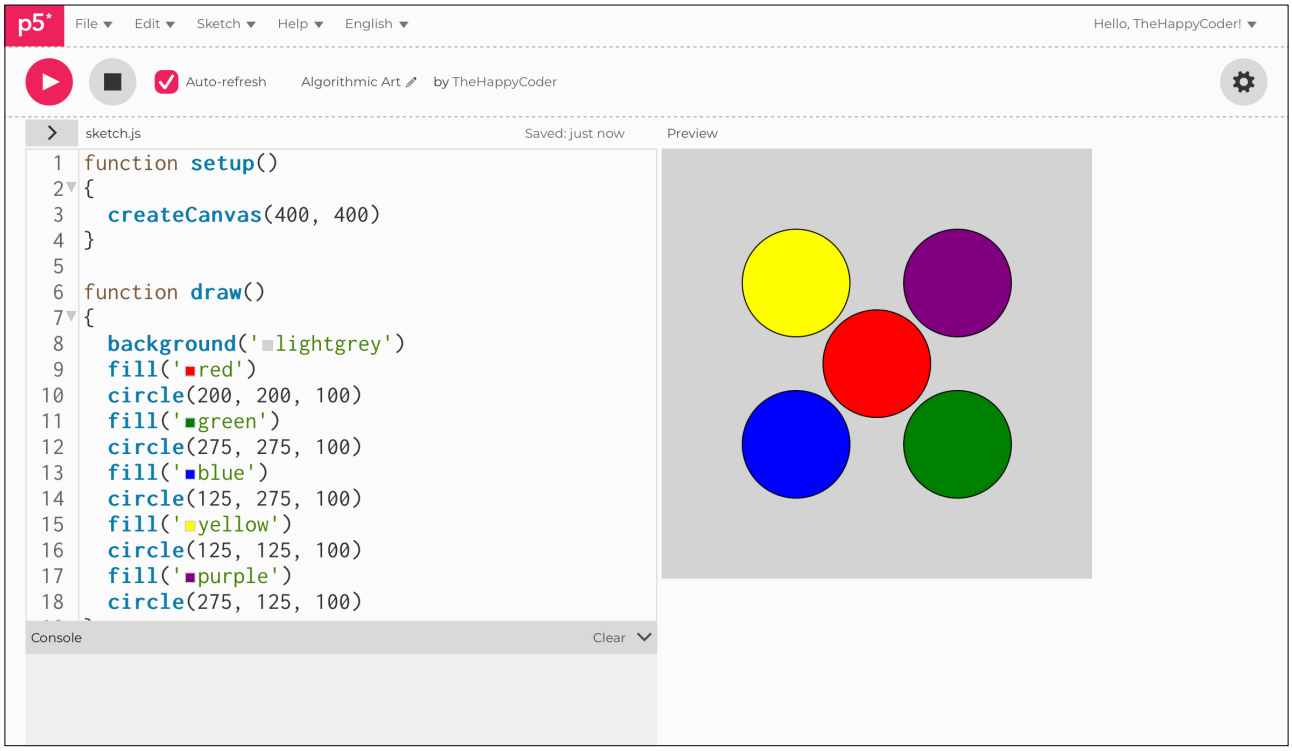
🗒️ Notes

To get each separate colour to fill the next circle, we have to put the `fill()` function in between the circles. The code runs in a linear fashion; it goes step by step, one line at a time, starting at the top and then works its way down till it reaches the bottom. Then, with the `draw()` function, it goes to the top and starts all over again.

🌻 Challenge

Try other colours and see which colours work and which don't

Figure A2.6


p5*
File
Edit
Sketch
Help
English
Hello, TheHappyCoder!
Auto-refresh
Algorithmic Art
by TheHappyCoder
sketch.js
Saved: just now
Preview
1 function setup()
2 {
3 createCanvas(400, 400)
4 }
5
6 function draw()
7 {
8 background('lightgrey')
9 fill('red')
10 circle(200, 200, 100)
11 fill('green')
12 circle(275, 275, 100)
13 fill('blue')
14 circle(125, 275, 100)
15 fill('yellow')
16 circle(125, 125, 100)
17 fill('purple')
18 circle(275, 125, 100)
Console
Clear

Sketch A2.7 new sketch

❗ Start a brand new sketch.

Writing the code for five circles is not too arduous, but what if you want to draw a hundred or a thousand circles? We are going to need a bit of help. Programming is all about efficiency, and this usually means writing code in the fewest lines as possible. Delete the previous code and write this.

❗ Notice that the `background()` goes into the `setup()` function.

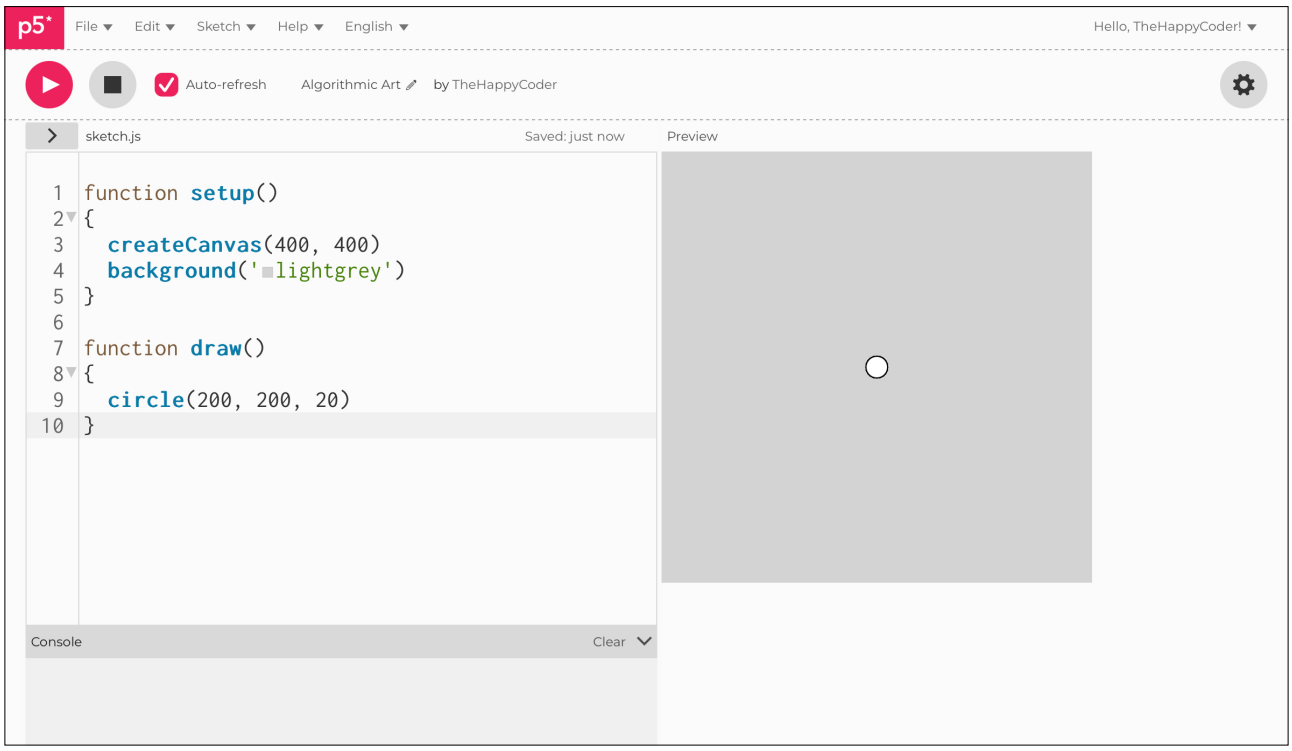
```
function setup()
{
  createCanvas(400, 400)
  background('lightgrey')
}

function draw()
{
  circle(200, 200, 20)
}
```

🗒️ Notes

We have drawn a small (20-pixel diameter) circle in the centre of the canvas, but before we move onto the next sketch, I need to introduce an important concept: `variables`.

Figure A2.7


p5*
File
Edit
Sketch
Help
English
Hello, TheHappyCoder!
Auto-refresh
Algorithmic Art
by TheHappyCoder
sketch.js
Saved: just now
Preview
1 function setup()
2 {
3 createCanvas(400, 400)
4 background('lightgrey')
5 }
6
7 function draw()
8 {
9 circle(200, 200, 20)
10 }
Console
Clear

Introduction to variables

Variables are very useful for storing data that we may want to access or change later. Variables can be named with a single letter or a name. They can have numbers in them but must never start with a number. Usually, the name of the variables has some relevant meaning.

For instance, if you want to have a variable for speed, you would be best to use the word `speed` rather than just `s` because later on in a long list of code, you may forget what `s` represents. This is especially the case if you have a lot of variables. At the same time, don't make them too long; otherwise, the code will look very messy and difficult to read by anyone else but you.

In this next example, we are going to give the coordinates for the circle names `x` and `y`. So that we can alter them in a later sketch. Variables are very powerful and extremely useful.

To use a variable like `x` and `y`, then we need to define them. We use the key word `let`. You can just define it or initialise it, for example:

`let x`	this defines x as a variable
`let x = 10`	this gives x an initial value of 10

Now in the sketch below we are going to create a variable for the `x-coordinate` (which is `200`) and the `y` co-ordinate (which is `300`) of the circle by calling them `let x` and `let y`.

One other important detail, `scope`. It is always a good idea to declare variables at the beginning of a sketch, that way they are available everywhere. If you only declare them between two curly brackets they only live between those curly brackets and nowhere else. You will see that I do both so it is usually OK but something to bear in mind when generating many functions and lines of code.

Mostly we will be using numeric values (integers and floats) but just occasionally we will be using strings (words or letters).

Sketch A2.8 adding some variables

We have replaced the x value and y value with variables of that name. This draws the same small circle in the centre of the canvas.

```
let x = 200
let y = 200

function setup()
{
  createCanvas(400, 400)
  background('lightgrey')
}

function draw()
{
  circle(x, y, 20)
}
```

🗒️ Notes

This demonstrates how using variables can be very useful, especially when there are many shapes and lots of movement. If you are familiar with data types `let` is a float (also called real) by default. The line of code `circle(x, y, 20)` is just the same as `circle(200, 200, 20)`.

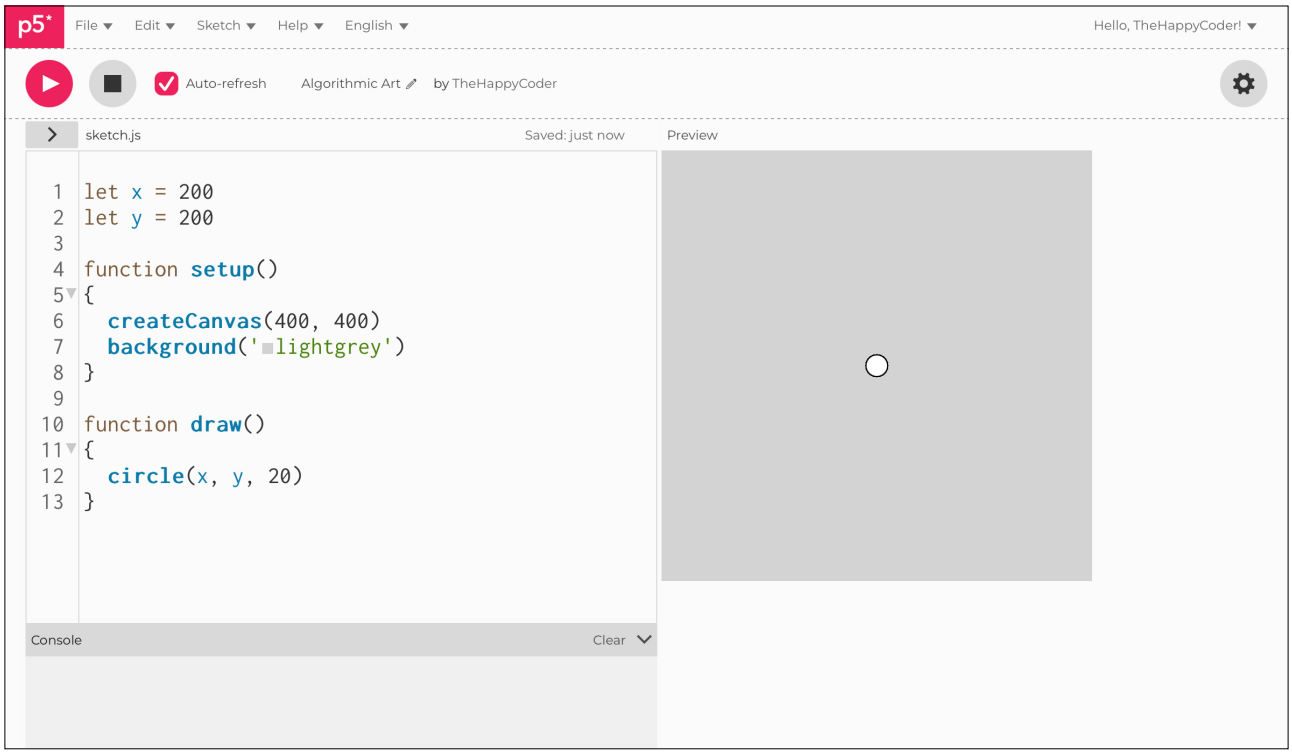
🌻 Challenges

1. Use different initialised values for the x and y variable
2. Create another variable for the diameter

🛠️ Code Explanation

`let x = 200`	Declares the variable x and initialises it to a value of 200
`let y = 200`	Declares the variable y and initialises it to a value of 200
`circle(x, y, 20)`	The circle is drawn at the initialised x and y values (200, 200)

Figure A2.8


p5*
File
Edit
Sketch
Help
English
Hello, TheHappyCoder!
Auto-refresh
Algorithmic Art
by TheHappyCoder
sketch.js
Saved: just now
Preview
let x = 200
let y = 200
function setup()
{
createCanvas(400, 400)
background('lightgrey')
}
function draw()
{
circle(x, y, 20)
}
Console
Clear

Introduction to Random

Now we will be making shapes move and introduce random. Different functions as loops will be explored. This is where we can create a random number. It is not really random but for most purposes it is random enough. We use the function `random()` and it works like this...

`random(20)`	will give you a number between 0 and 19 (but not 20)
`random(12, 37)`	would give you a value between 12 and 37 (but not 37)

Before we go any further we need to look at notations used with coding which are also used in maths. Just read quickly through the next section to get a feel for the notations and what they mean. As you use them they will make more sense, I include them here and now so that you have a reference.

Symbols we use... operators

❗ Please just skim read through these; they are for reference and you don't need to learn them just yet. I include them now without any context so that you are aware of them. You may find them a useful reference later on.

In coding (as well as in maths), we use notation, symbols rather than words. Most of the maths used in coding is fairly basic; some symbols, however, may be unfamiliar. We are going to quickly cover the following:

1 Arithmetic Operators

Simple mathematical operators you will use frequently. They are used throughout all coding and should be quite familiar except for, perhaps, `modulus`.

+	Addition (2 + 4) will give you 6
-	Subtraction (6 - 3) will give you 3
/	Division (8 / 2) will give you 4
*	Multiplication (3 * 5) will give you 15
%	Modulus gives you the remainder (10 % 8) will give you 2
=	Equals (not equals to) 2 + 4 = 6

2 Comparison Operators

In short, these are conditional statements where a condition needs to be met. These are often used with `while()` loops

==	means equal to (5 == 5) is TRUE, (6 == 5) is FALSE
<	means less than (6 < 8) is TRUE, (8 < 6) is FALSE,
<=	means less than or equal to (6 <= 8) is TRUE, (6 <= 6) is also TRUE
>	means greater than (8 > 6) is TRUE, (6 > 8) is FALSE,
>=	means greater than or equal to (8 >= 6) is TRUE, (6 >= 6) is also TRUE
!=	means not equal to (6 != 5) is TRUE (5 != 5) is FALSE

3 Logical Operators

These are used with `if()` statements. The `if()` statement can be similar to the `while()` loop; if something is true or a condition is met, then do something, for example: `if(x < 100 && y > 50)` means if x is `less than` 100 `AND` y is `greater than` 50.

&&	means AND (x < 100 && y > 50)

`||`	means OR `(x < 100 || y y > 50)`
`!`	Means NOT `(x < 100 ! y y > 50)`

4 Assignment Operators

They are used often in coding (some much more than others). It is more obvious when you see them in a meaningful context, which often goes for all coding.

`++`	means increasing by 1 (x++) x is incremented by 1 each time
`--`	mean reducing or subtracting by 1 (y–) y is decremented by 1 each time
`+=`	means addition `(x += 10)` or `(x += y)` same as `x = x + y`
`-=`	means subtracting `(x -= 10)` or `(x -= y)` same as `x = x - y`
`*=`	means multiplying `(x *= 10)` or `(x *= y)` same as `x = x * 10`
`/=`	means division `(x /= 2)` or `(x /= y)` same as `x = x / 10`

5 Maths Functions

When using the mathematical notation, these can be very useful. Here are just a few

`floor()`	Calculates the closest integer value that is less than or equal to the value of a number
`abs()`	Calculates the absolute value of a number. A number's absolute value is its distance from zero on the number line. -5 and 5 are both five units away from zero, so calling **abs(-5)** and **abs(5)** both return 5. The absolute value of a number is always positive.
`round()`	Calculates the integer closest to a number. For example, **round(133.8)** returns the value 134. The second parameter, **decimals**, is optional. It sets the number of decimal places to use when rounding. For example, **round(12.34, 1)** returns 12.3. **It is** zero by default.
`sq()`	Calculates the square of a number. Squaring a number means multiplying the number by itself. For example, **sq(3)** evaluates 3 x 3 which is 9. The **sq(-3)** evaluates -3 x -3 which is also 9. Multiplying two negative numbers produces a positive number. The value returned by **sq()** is always positive.
`squrt()`	Calculates the square root of a number. A number's square root can be multiplied by itself to produce the original number. For example, **sqrt(9)** returns 3 because 3 x 3 = 9. The **sqrt()** always returns a positive value. **sqrt()** doesn't work with negative arguments such as **sqrt(-9)**.
`pow`	Calculates exponential expressions such as 2^3. For example, **pow(2, 3)** evaluates the expression 2 x 2 x 2. **pow(2, -3)** evaluates 1 ÷ (2 x 2 x 2).
`ceil()`	Calculates the closest integer value that is greater than or equal to a number. For example, calling **ceil(9.03)** and **ceil(9.97)** both return the value 10.
`exp()`	Calculates the value of Euler's number e (2.71828...) raised to the power of a number.

Sketch A2.9 drawing random circles

In this sketch, you will draw circles in random positions on the canvas. The function `draw()` is a continuous loop and will draw them forever. You don't need to give `x` and `y` an initial value, but it is good practice to do so.

```
let x = 200
let y = 200

function setup()
{
  createCanvas(400, 400)
  background('lightgrey')
}

function draw()
{
  x = random(400)
  y = random(400)
  circle(x, y, 20)
}
```

🗒️ Notes

A reminder that in computer code it starts counting from 0, not 1, and so it stops when it gets to the 400th number, which is 399. Just something to remember.

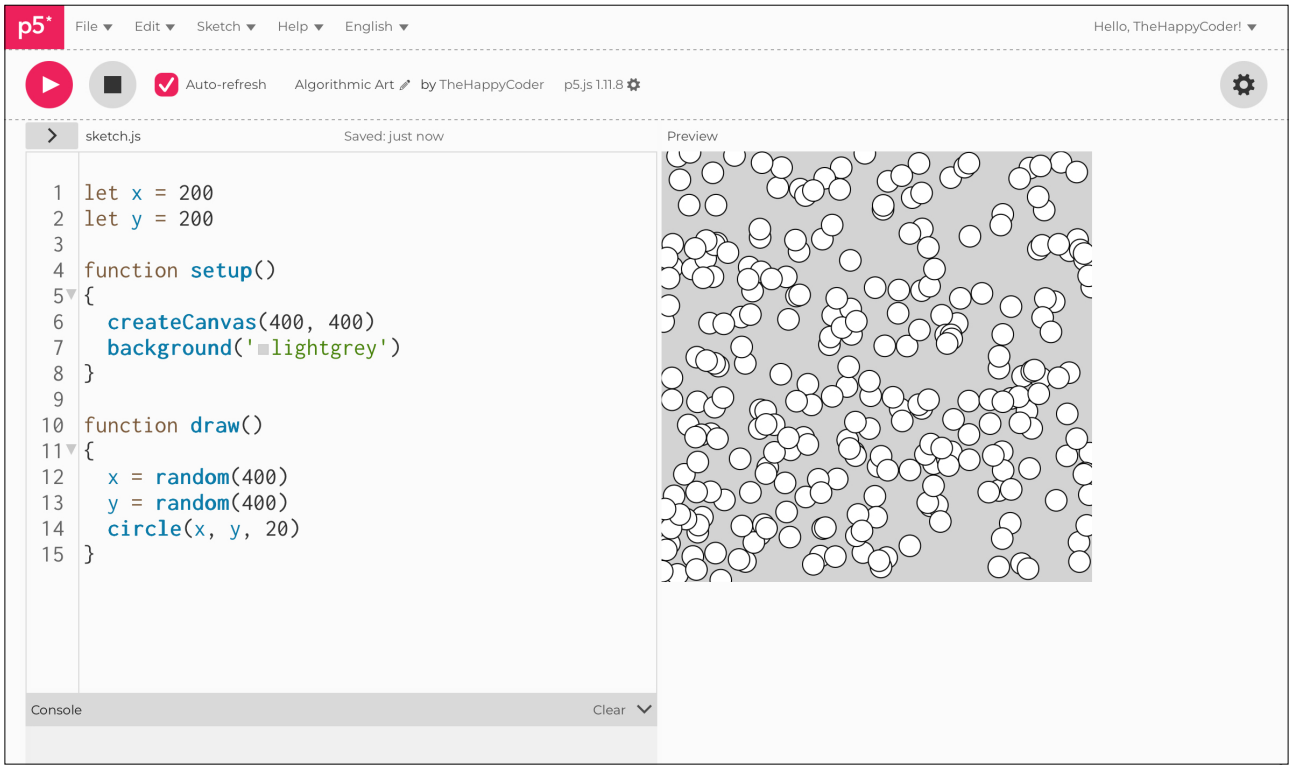
🌻 Challenge

Create a random diameter as well.

🛠️ Code Explanation

`x = random(400)`	Choses a random number from 0 to 399 for the x value
`y = random(400)`	Choses a random number from 0 to 399 for the x value

Figure A2.9


p5*
File
Edit
Sketch
Help
English
Hello, TheHappyCoder!
Auto-refresh
Algorithmic Art
by TheHappyCoder
p5.js 1.11.8
sketch.js
Saved: just now
Preview
let x = 200
let y = 200
function setup()
{
createCanvas(400, 400)
background('lightgrey')
}
function draw()
{
x = random(400)
y = random(400)
circle(x, y, 20)
}
Console
Clear

Sketch A2.10 while() loop circles

❗ delete what is in the `draw()` function, notice we have a loop within draw and so things are double indented.

What we want to do is draw `100` circles and then stop. The `while()` loop keeps checking the number of circles it has drawn. If the number is less than (`<`) `100`, it keeps going while it is `true`. Once it has drawn `100` and the next one is therefore more than `100`, it then stops because it returns `false`. To keep track of the number of circles, we add a new variable called `count`.

❗ Make sure that `Auto-refresh` is `OFF`, otherwise the programme will likely crash as you type it in. It is the tick box near the top.

```
let x = 200
let y = 200
let count = 0

function setup()
{
  createCanvas(400, 400)
  background('lightgrey')
}

function draw()
{
  while (count < 100)
  {
    x = random(400)
    y = random(400)
    circle(x, y, 20)
    count = count + 1
  }
}
```

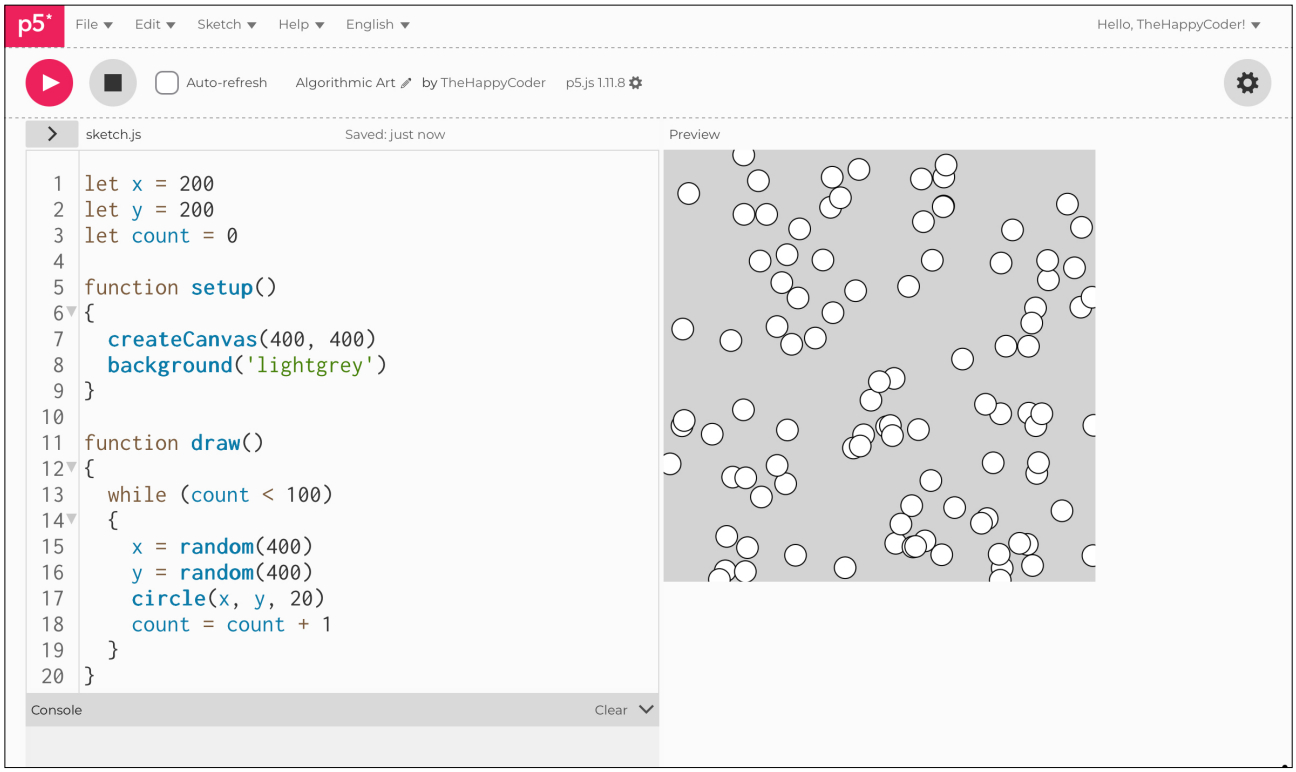
🗒️ Notes

This sketch draws exactly `100` small circles in random positions every time you press the `run` button. It makes good use of the `while()` loop function. Also uses the comparison `<` less than.

🛠️ Code Explanation

`while (count < 100)`	This is a while() loop which checks to see if count has reached 100
`count = count + 1`	We add 1 each time to the count varaible

Figure A2.10



Sketch A2.11 a few more adjustments

We can use the `random()` function to create a border and introduce a very common shorthand for adding each time `1` (`++`). So, `count = count + 1` becomes: `count++`.

```
let x = 200
let y = 200
let count = 0

function setup()
{
  createCanvas(400, 400)
  background('lightgrey')
}

function draw()
{
  while (count < 100)
  {
    x = random(100, 300)
    y = random(100, 300)
    circle(x, y, 20)
    count++
  }
}
```

🗒️ Notes

We have now limited the random range to between `100` and `300`; this gives us a nice empty border around the canvas.

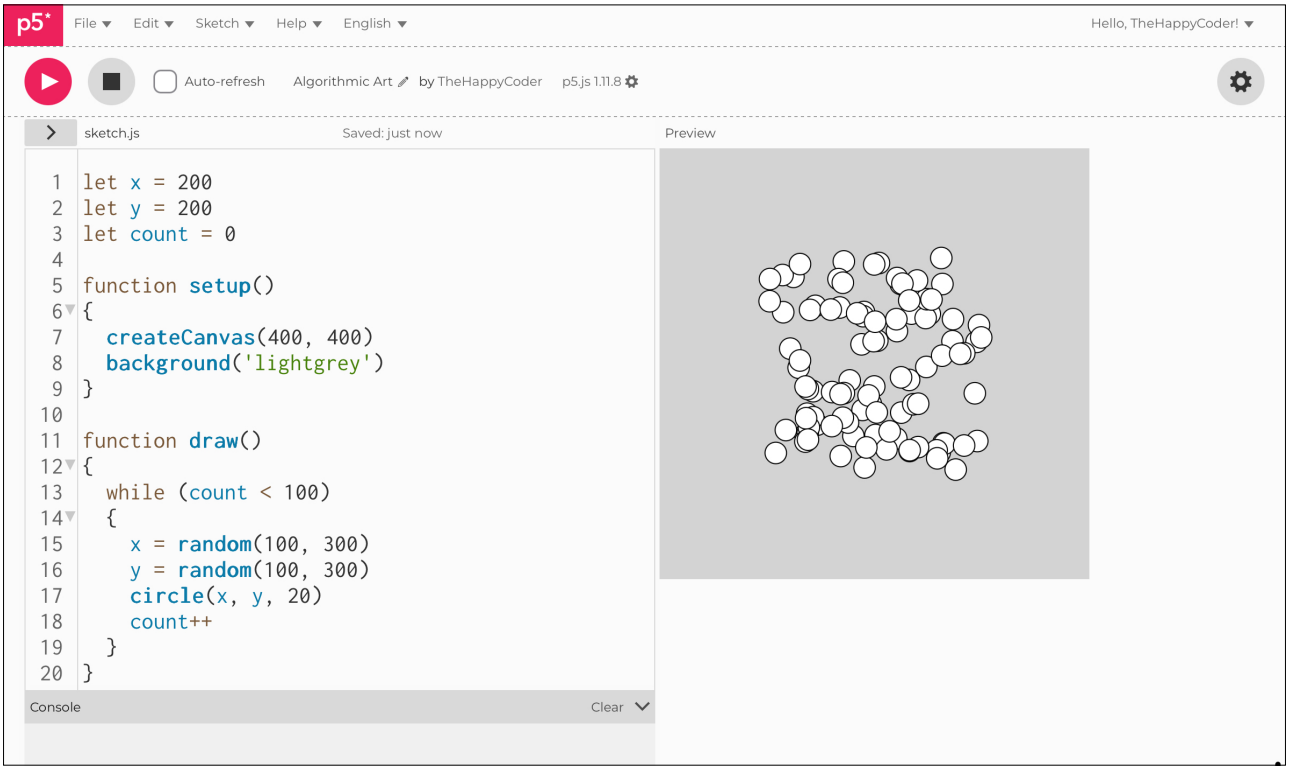
🌻 Challenge

Try different values for random.

🛠️ Code Explanation

`x = random(100, 300)`	Random number between 100 and 300
`y = random(100, 300)`	Random number between 100 and 300
`count++`	Shorthand version, adds 1 each time, the same as count = count + 1

Figure A2.11


p5*
File
Edit
Sketch
Help
English
Hello, TheHappyCoder!
Auto-refresh
Algorithmic Art
by TheHappyCoder
p5.js 1.11.8
sketch.js
Saved: just now
Preview
let x = 200
let y = 200
let count = 0
function setup()
{
createCanvas(400, 400)
background('lightgrey')
}
function draw()
{
while (count < 100)
{
x = random(100, 300)
y = random(100, 300)
circle(x, y, 20)
count++
}
}
Console
Clear

Introducing the for() loop

You have already come across the `while()` loop. There is another called the `if()` statement (coming soon) which is similar to the `while()` loop. There is a third and it is arguably the most important or the most used. It is a little bit more complicated, but as we will be using it a lot, you will get used to it and it does become very intuitive (trust me).

Our example is as follows:

```
for (let i = 0; i < 100; i++)
```

Inside the `for()` loop, brackets are three parts separated by semicolons (`;`), they are:

1️⃣ `let i = 0`
2️⃣ `i < 100`
3️⃣ `i++`

Firstly, 1️⃣ we create a variable called `i`. This is just convention; you can call it anything you like, and we give it an initial value of `0`.
Secondly, 2️⃣ we put a condition on this variable, in this case, less than `100` (`<`). The `for()` loop is true while `i` is less than `100`.
Thirdly, 3️⃣ we increment the loop, in this instance, we simply add `1` to `i` for each loop (`i++`) until `i` has the value of `100` and stops.

Sketch A2.12 using for() loops

Another way to draw 100 circles is to use a for() loop. It loops through as a sort of counter from 0 to 100 in steps of 1. We are still going to use i as the counter, but we initialise it inside the for() loop itself. The noLoop() function stops the draw() function but only after it has drawn 100 circles.

❗ Remove: all reference to the count variable, I recommend starting a new sketch, it helps memory muscle.

```
let x = 200
let y = 200

function setup()
{
  createCanvas(400, 400)
  background('lightgrey')
}

function draw()
{
  for (let i = 0; i < 100; i++)
  {
    x = random(100, 300)
    y = random(100, 300)
    circle(x, y, 20)
  }
  noLoop()
}
```

🗒️ Notes

Draws 100 circles in random positions once between 100 and 300 using the for() loop. For (let i = 0; i < 100; i++) draws 100 circles in random positions once between 100 and 300 using the for() loop. There are three parts to it:
Part 1: let i = 0;
Part 2: i < 100;
Part 3: i++
Persevere with the for() loop as it will appear a lot in the future. Notice we have the semicolons separating the three parts.

🌻 Challenges

1. Change the number from 100 to either 10 or 1000.
2. Take out the noLoop() function.

🛠️ Code Explanation

for()	This is the for() loop, it will count to 10 and stop (actually will then start counting again. The draw function is a loop so it is a loop within a loop
let i = 0	This declares a variable called i and initialises it to 0
i < 100	This bit checks to see if i is still less than 100
i++	This adds 1 to the i variable every time it does a loop
noLoop()	Stops the draw() loop once the condition has been met

Figure A2.12