The Joy of Coding Algorithmic Art Workbook #2 Colour and Movement

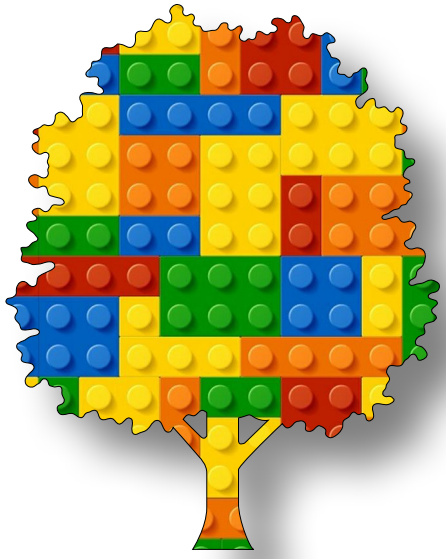


Table of Contents

MIT Licence

Workbook #2 Quick Content Summary

Assuming you have completed workbook #1, we are now going to get shapes moving and rotating. A few new concepts will be introduced as we go along. Also we are going to have a peek at some of the colour options available to us.

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 B Unit #1 bounce & rotate

Module B Unit #1: bounce and rotate

In this unit, we will get shapes moving and create patterns with some simple algorithms. We will start with moving a circle across the canvas, only to reappear on the other side. Then, we will look at bouncing it off the edge of the canvas. Finally, we will look at rotation, angular velocity, and the acceleration of a spinning baton.

Key Concepts:

angular velocity
angular acceleration
using the OR comparison
bouncing off an edge

Sketch B1.1 starting sketch

! Start a new sketch

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

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

Notes

Our simple starting sketch

Challenge

You can add some colour or different size canvas

Sketch B1.2 draw a circle

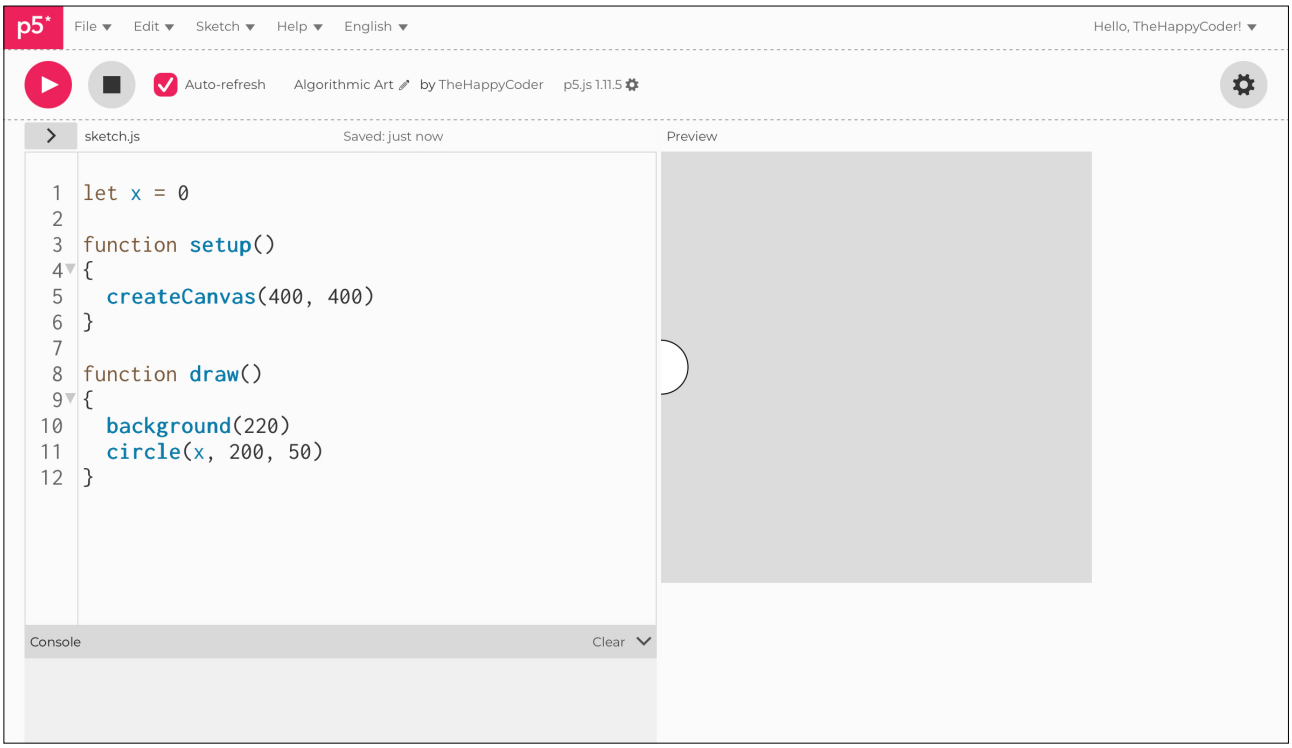
We will create a variable for the x co-ordinate, and draw a circle.

```
let x = 0

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

function draw()
{
  background(220)
  circle(x, 200, 50)
}
```

Notes

Our circle starts on the left hand edge of the canvas.

Figure B1.2



Sketch B1.3 increment

Now we can increment the variable x and make it move slowly across the canvas, one pixel at a time.

```
let x = 0

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

function draw()
{
  background(220)
  circle(x, 200, 50)
  x = x + 1
}
```

Notes

You will notice that it goes off the canvas (eventually), never to be seen again.

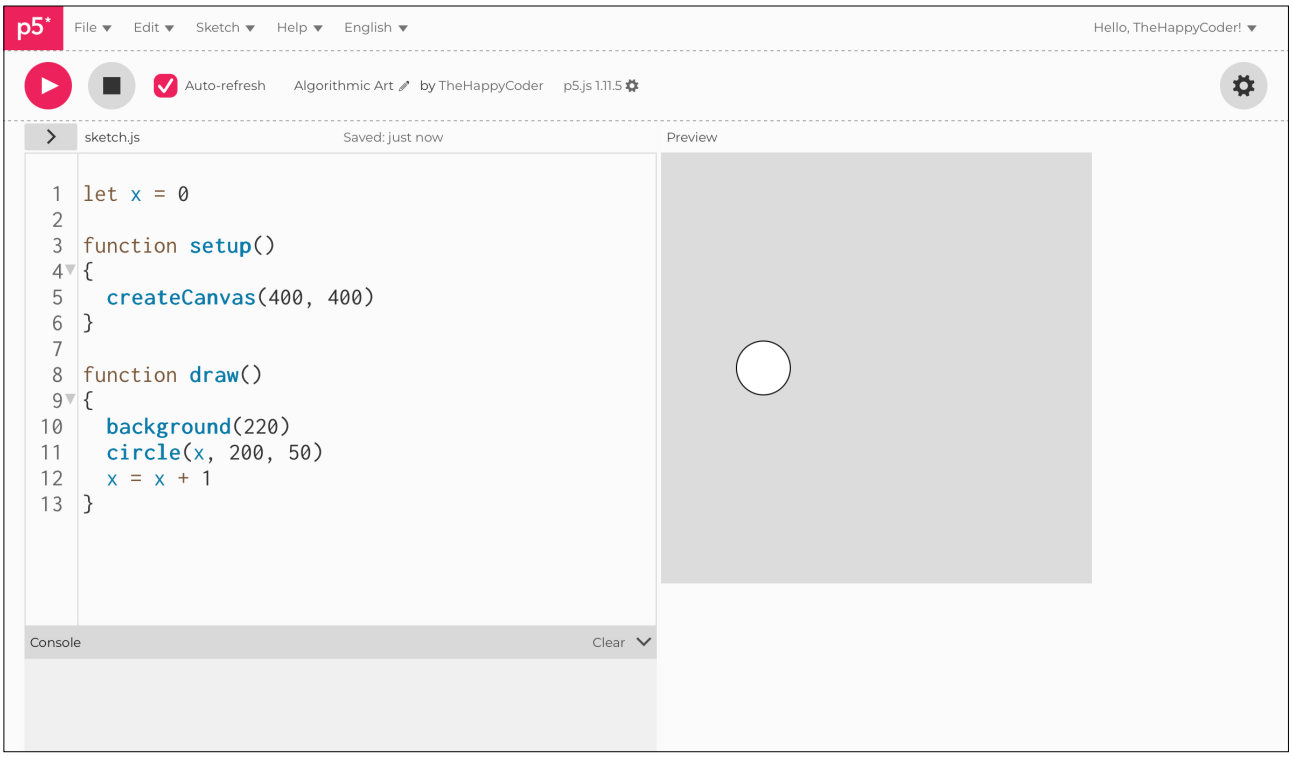
Challenge

How do you think you could make it move faster?

Code Explanation

`x = x + 1`	Adds 1 to the value of x on each iteration.

Figure B1.3



Sketch B1.4 a short cut

We can simplify the increment and also increase it from one pixel to five.

```
let x = 0

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

function draw()
{
  background(220)
  circle(x, 200, 50)
  x += 5
}
```

Notes

Moves a lot faster now.

Code Explanation

`x += 5`	Same as `x = x + 5`.

Sketch B1.5 using an if statement

The problem with the previous sketch is that the circle just wanders off the edge of the canvas. If we want it to start again once it has reached the right-hand edge, then an `if()` statement is what you need. In simple terms: if the circle is at the right-hand edge, start again.

```
let x = 0

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

function draw()
{
  background(220)
  circle(x, 200, 50)
  x += 5
  if (x == 400)
  {
    x = 0
  }
}
```

Notes

This uses an if statement. Notice that when it gets to the right-hand edge, `x` has the value `400` (`x == 400`). It then resets the value of `x` to `0`, which returns it to the start.

Challenges

1. What happens when `x = x + 7`?
2. What could you do about that? Hint: change `x == 400` to `x >= 400`.

Code Explanation

`if(x == 400)`	If x is exactly 400, then make it equal to zero (`x = 0`).

Sketch B1.6 some velocity

We need another variable; we will call it `velocity`. We can use this variable to good effect. We have also made it so that the `==` (is equal to) symbol is now `>=` (is greater than or equals to).

```
let x = 0
let velocity = 7

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

function draw()
{
  background(220)
  circle(x, 200, 50)
  x += velocity
  if(x >= 400)
  {
    x = 0
  }
}
```

Notes

The benefit of having the greater than or equals to (>=) symbol is that you can move it as fast as you want. If we didn't change the condition, it would be moving too fast and would effectively jump over the edge of the canvas.

Challenges

1. What happens if you leave the conditional as ==?
2. Can you make it bounce off the right-hand edge?

Code Explanation

`let velocity = 7`	The velocity is initialised to 7.
`x += velocity`	Increment the position of the circle by that amount.
`if(x >= 400)`	If x is greater than or equal to 400, then set x to zero.

Sketch B1.7 bounce off the edge

In this sketch, we go one step better. Instead of starting all over again, we can use an `if()` statement to make the circle bounce off the right-hand edge. To do this, we need another variable called `velocity`, which we can use to change the direction. The variable name `velocity` is one we have just made up.

```
let x = 0
let velocity = 7

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

function draw()
{
  background(220)
  circle(x, 200, 50)
  x += velocity
  if(x >= 400)
  {
    velocity = -velocity
  }
}
```

Notes

This time, rather than adding `1`, we have created a variable called `velocity`, which we start by giving it the value `7`. This has the same effect of adding `7` to `x` each time. When the circle reaches the edge and `x` is greater than or equal to `400` (`x >= 400`), then `7` becomes `-7`, and so it subtracts `7` each time from `x`. At this point, it disappears off the canvas again. The benefit of having the greater than or equals to (`>=`) symbol is that you can move it as fast as you want.

Challenge

Can you make it bounce off the left-hand edge as well? See next sketch.

Code Explanation

`velocity = -velocity`	This reverses the direction.

Sketch B1.8 bounce from every side

Not satisfied with bouncing off one side, how about both edges? To do this, we need a logic OR comparison. The direction will change whether it reaches the right-hand or left-hand edge of the canvas. You will need to locate the | symbols (called pipes ||) on your keyboard; you may find that with some keyboards they are two vertical lines, one on top of the other.

```
let x = 0
let velocity = 7

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

function draw()
{
  background(220)
  circle(x, 200, 50)
  x += velocity
  if (x >= 400 || x <= 0)
  {
    velocity = -velocity
  }
}
```

Notes

You should have the circle bouncing off each side. This `if()` statement has two conditions, the first one is for the right-hand edge `x >= 400`, and the second is the left-hand edge `x <= 0`. The two vertical lines (`||`) are the symbols for `OR`, which means if either is true. The symbols are called pipes for some reason.

Challenge

How could you make the circle get bigger towards the centre and smaller as it approaches the edges? It is doable but requires some mental gymnastics.

Code Explanation

`if (x >= 400 || x <= 0)`	If x is greater than or equal to 400, OR x is less than or equal to 0, then reverse the direction.

Sketch B1.9 bigger and smaller

Making the circle expand and contract as it moves across the canvas.

```
let x = 0
let velocity = 7
let diameter = 0

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

function draw()
{
  background(220)
  circle(x, 200, diameter)
  x += velocity
  if (x >= 400 || x <= 0)
  {
    velocity = -velocity
  }
  if (x <= 200)
  {
    diameter = x / 2
  }
  if (x >= 200)
  {
    diameter = (400 - x) / 2
  }
}
```

Notes

This is a good time to work through the maths here. `x` is the variable that is changing from `0` to `400` as it moves across the canvas and vice versa on the way back.

Challenges

1. Try some other values.
2. Experiment with other shapes.

Code Explanation

`diameter = x / 2`	The diameter is dependent on this value in the left-hand half of the canvas.
`diameter = (400 - x) / 2`	The diameter is dependent on this value in the right-hand half of the canvas.

Spinning baton

In this section, we are going to create a baton (simply a line with a circle at each end) and try to spin it. We will introduce angular velocity and angular acceleration to the baton.

Angular velocity is the measure of the sped of rotation whereas the angular acceleration is how much that speed is increasing or decreasing. There is a limit to how fast we can accelerate something because of the nature of the screen frame rate.

Sketch B1.10 new sketch

! Starting a new sketch

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

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

Sketch B1.11 translate the line

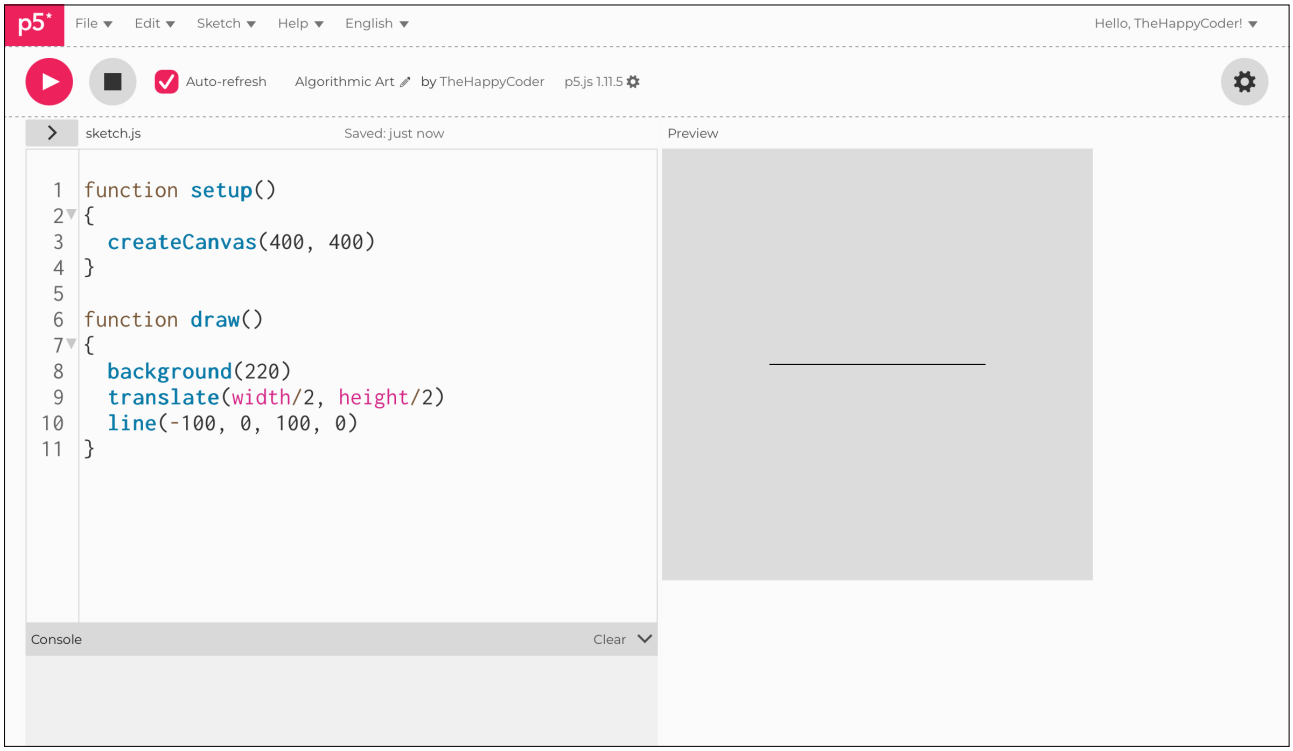
We translate the origin to the centre of the canvas and draw a line so that it straddles the centre.

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

function draw()
{
  background(220)
  translate(width/2, height/2)
  line(-100, 0, 100, 0)
}
```

Notes

This puts the origin in the centre of the canvas, the co-ordinates (the four arguments) of the line reflect this. If you are a little confused, I suggest sketching it out.

Figure B1.11



Sketch B1.12 adding the circles

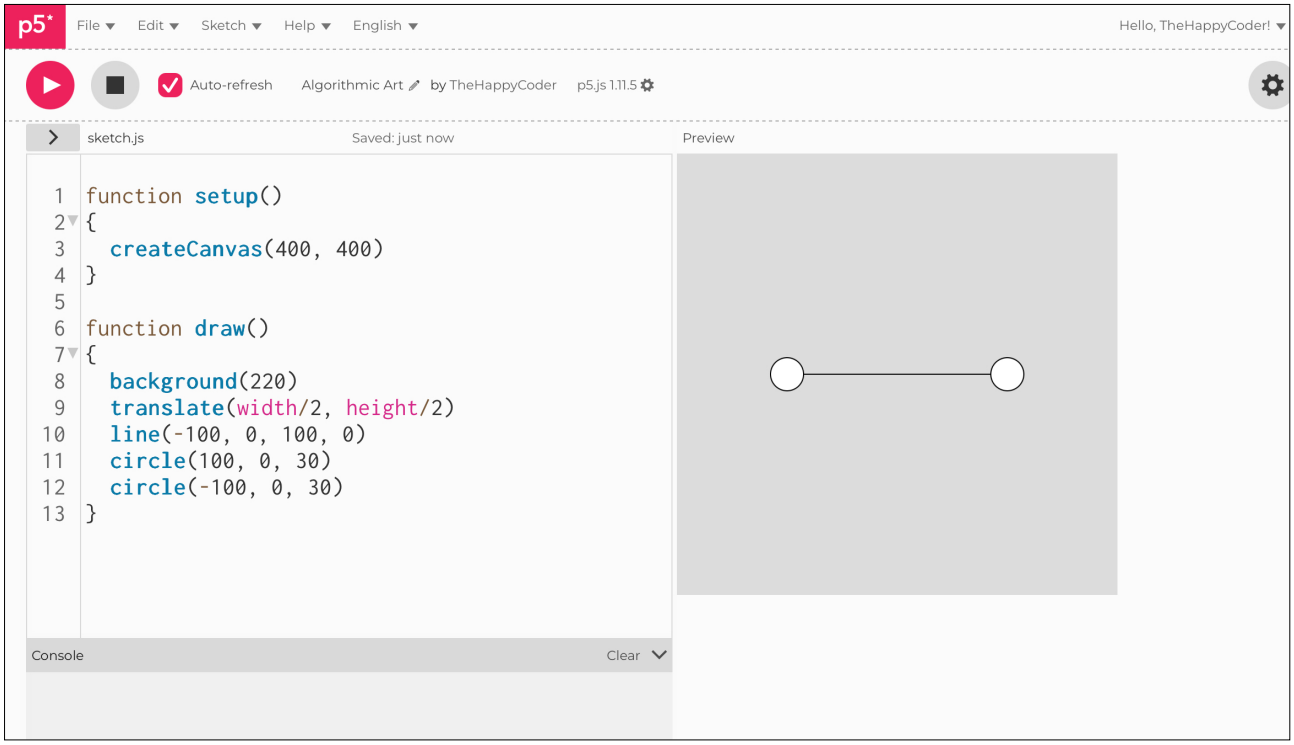
We will add a circle to each end of the line.

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

function draw()
{
  background(220)
  translate(width/2, height/2)
  line(-100, 0, 100, 0)
  circle(100, 0, 30)
  circle(-100, 0, 30)
}
```

Notes

Notices that the co-ordinates take into account the translation; this is so we can rotate about the origin.

Figure B1.12



Sketch B1.13 angle of rotation

We add a variable called `angle` and rotate it about that angle. We will keep it in radians for now.

```
let angle = 0

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

function draw()
{
  background(220)
  translate(width/2, height/2)
  rotate(angle)
  line(-100, 0, 100, 0)
  circle(100, 0, 30)
  circle(-100, 0, 30)
}
```

Notes

Notice that there is no movement just yet; to do that, we need to increment the angle.

Code Explanation

`rotate(angle)`	Rotates the value of the variable angle in radians.

Sketch B1.14 start it spinning

We can now spin the baton at a nice, steady rate by incrementing the `angle` by `0.05` every iteration.

```
let angle = 0

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

function draw()
{
  background(220)
  translate(width/2, height/2)
  rotate(angle)
  line(-100, 0, 100, 0)
  circle(100, 0, 30)
  circle(-100, 0, 30)
  angle += 0.05
}
```

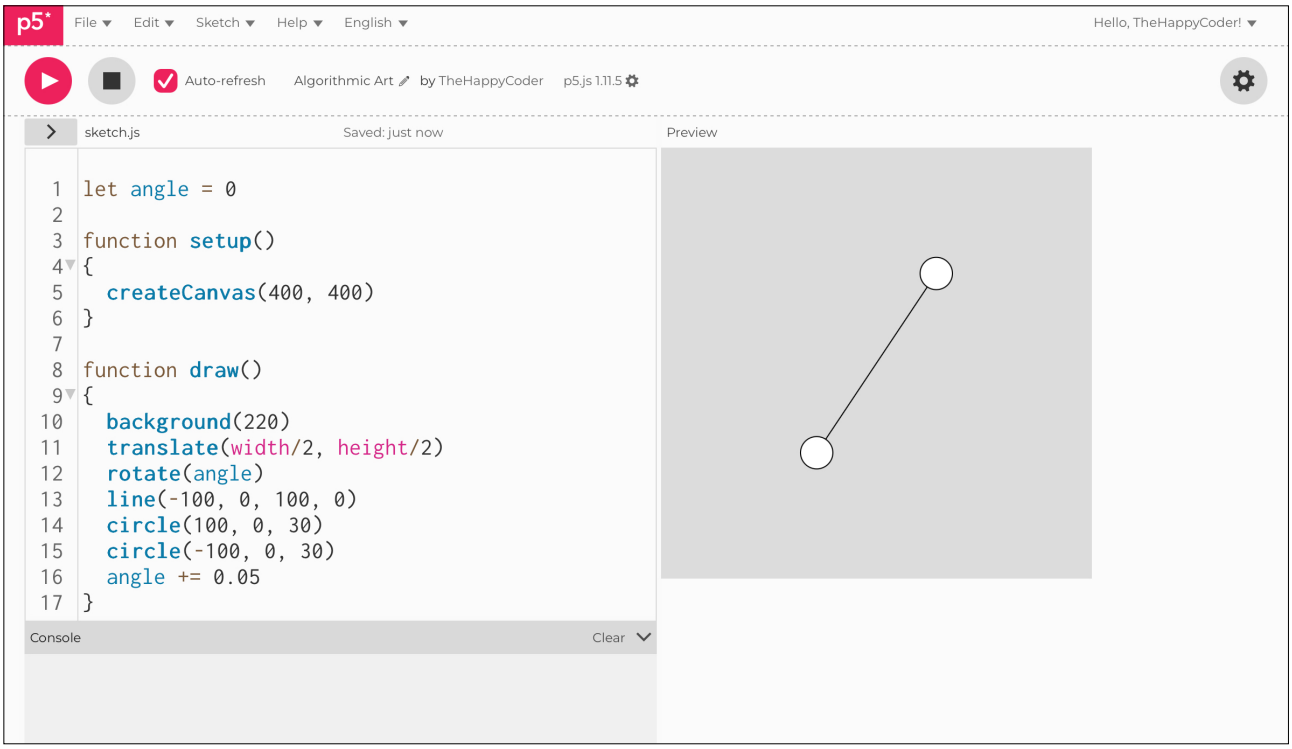
Notes

The baton rotates at `0.05` radians per iteration in the `draw()` loop.

Challenge

Change the value from `0.05`.

Figure B1.14



Sketch B1.15 velocity

We replace `increment` with a value for `velocity`.

```
let angle = 0
let velocity = 0.05

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

function draw()
{
  background(220)
  translate(width/2, height/2)
  rotate(angle)
  line(-100, 0, 100, 0)
  circle(100, 0, 30)
  circle(-100, 0, 30)
  angle += velocity
}
```

Notes

Nothing has changed.

Sketch B1.16 acceleration

We add an acceleration variable; this means that the velocity is incremented by that amount each iteration (hence the very small number).

```
let angle = 0
let velocity = 0.05
let acceleration = 0.0005

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

function draw()
{
  background(220)
  translate(width/2, height/2)
  rotate(angle)
  line(-100, 0, 100, 0)
  circle(100, 0, 30)
  circle(-100, 0, 30)
  angle += velocity
  velocity += acceleration
}
```

Notes

What you should see (if you give it a minute or two) is the baton spinning faster and faster. Acceleration is the rate of change of velocity. So the velocity is increasing gradually.

Challenge

Try: `let velocity = -0.1`.

The Joy of Coding Algorithmic Art Module B Unit #2 orbit and oscillate

Module B Unit #2: orbit and oscillate

Using sine and cosine, we will move in a circular motion, followed by an introduction to simple harmonic motion (SHM). With these simple algorithms, we can create some interesting patterns.

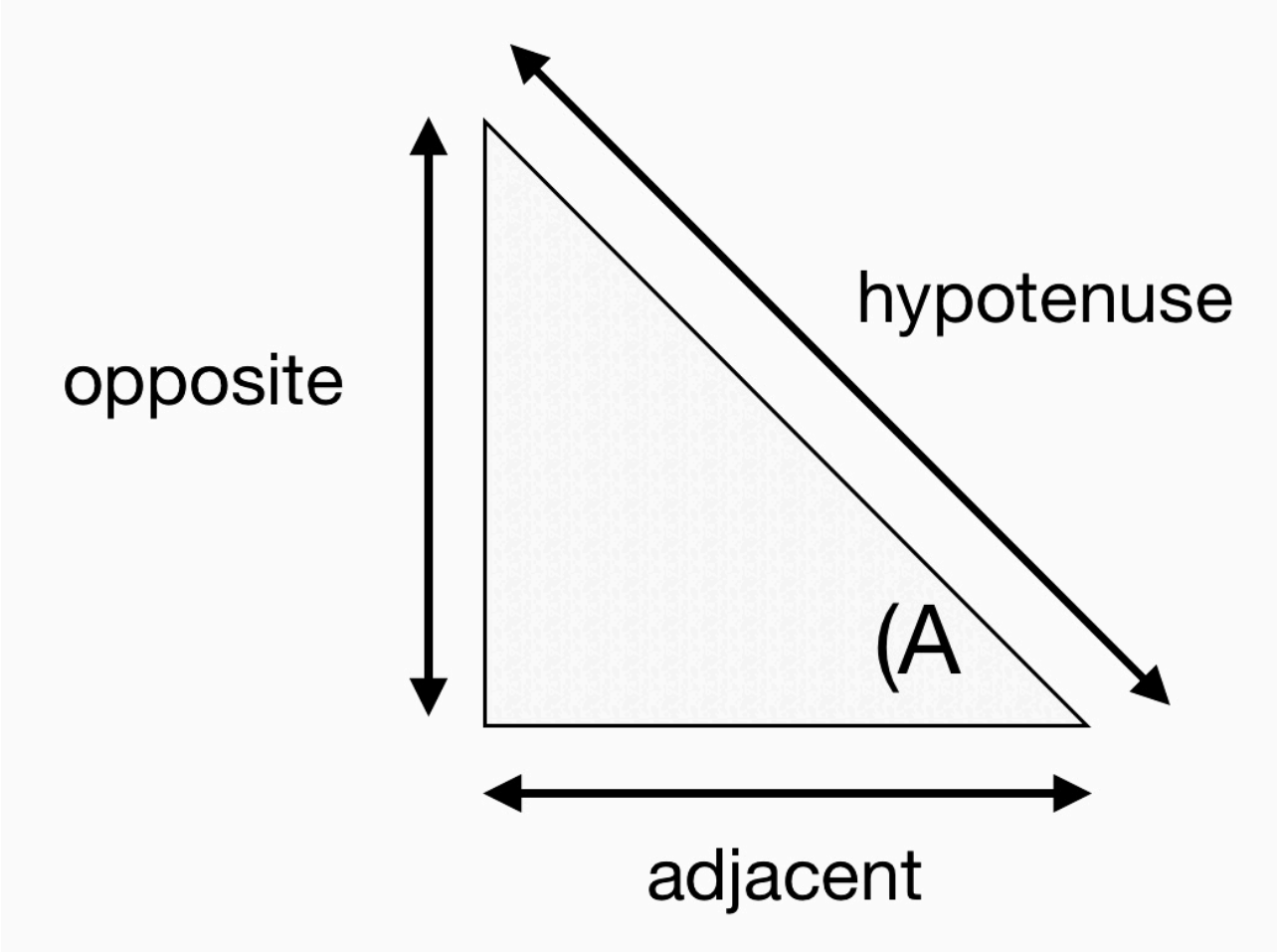
Key concepts:

polar co-ordinates
sine and cosine
oscillation
simple harmonic motion

In this section, we will look at angles with cosine (cos) and sine (sin). This is a little bit of maths. The sine of an angle in a right-handed triangle is the length of the opposite side divided by the hypotenuse. See fig. 1 below.

```
sine(angle)   = opposite / hypotenuse
cosine(angle) = adjacent / hypotenuse
```

Figure 1: A triangle labelled relative to angle (A)

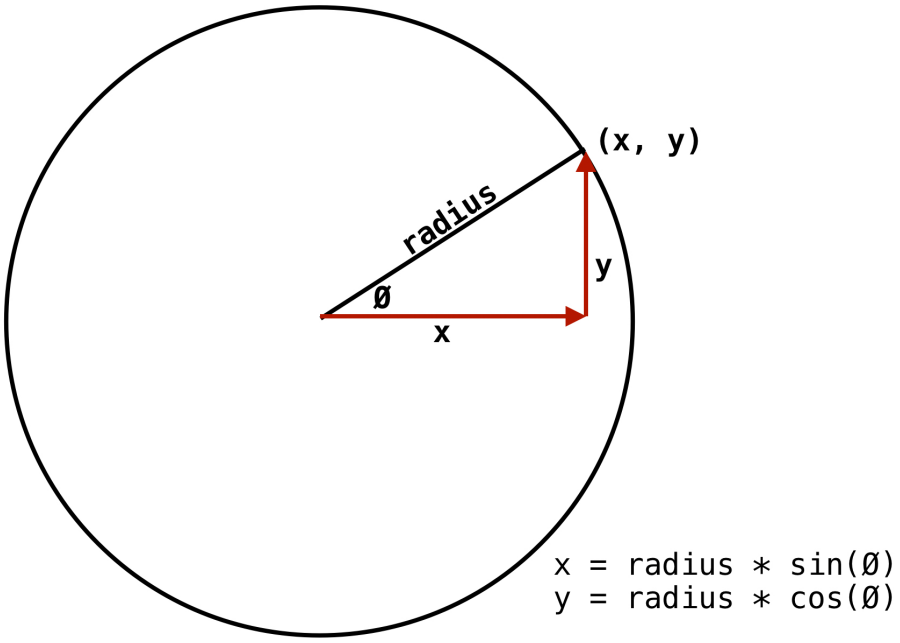


We are going to move a circle in a large circular motion. This may seem easy, but we will need the above maths to work out the co-ordinates of the circle. Without going into lots of maths, the equation for the co-ordinates from the centre point (0, 0) is:

```
x = radius * sin(angle)
y = radius * cos(angle)
```

See fig.2 below where Ø is the angle.

Figure 2: coordinates of a point on a circle



Sketch B2.1 a simple circle

! Start a new sketch

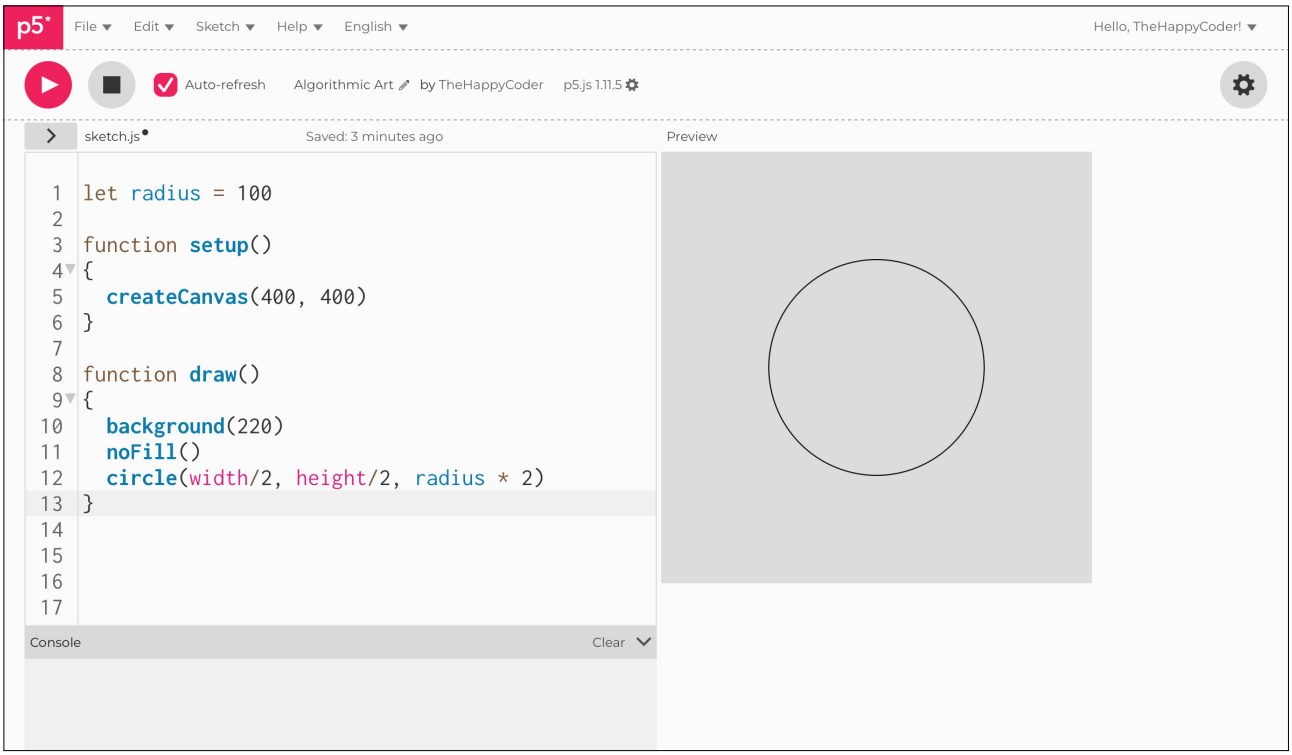
I have drawn a circle in the centre of the canvas with a diameter of `200` (`radius * 2`).

```
let radius = 100

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

function draw()
{
  background(220)
  noFill()
  circle(width/2, height/2, radius * 2)
}
```

Notes

What we are going to do is have a small circle follow the path of the large circle.

Figure B2.1



Sketch B2.2 creating the variables

We want three variables: an `angle` and the `x` and `y` co-ordinates to draw our circle.

```
let radius = 100
let angle = 0
let x
let y

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

function draw()
{
  background(220)
  noFill()
  circle(width/2, height/2, radius * 2)
}
```

Notes

Nothing yet to show for it, though.

Sketch B2.3 the co-ordinates

We put in the co-ordinates of a point on the circle.

```
let radius = 100
let angle = 0
let x
let y

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

function draw()
{
  background(220)
  noFill()
  circle(width/2, height/2, radius * 2)
  x = radius * sin(angle)
  y = radius * cos(angle)
}
```

Notes

This is not correct, as you will see in the next sketch.

Code Explanation

`x = radius * sin(angle)`	The x-coordinate on a circle at that angle with a radius.
`y = radius * cos(angle)`	The y-coordinate on a circle at that angle with a radius.

Sketch B2.4 draw the circle

Here we can add the small (yellow) circle that we want to follow the path of the large circle.

```
let radius = 100
let angle = 0
let x
let y

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

function draw()
{
  background(220)
  noFill()
  circle(width/2, height/2, radius * 2)
  x = radius * sin(angle)
  y = radius * cos(angle)
  fill('yellow')
  circle(x, y, 20)
}
```

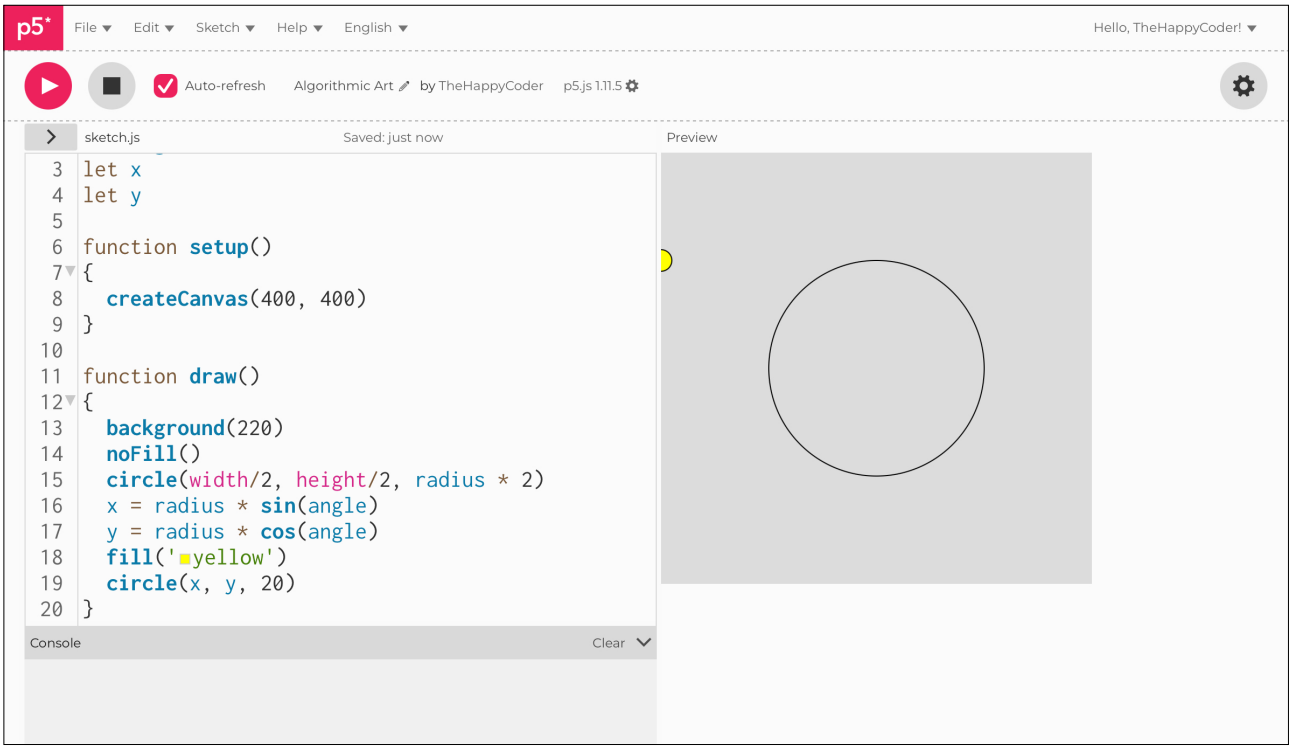
Notes

We have a problem, the yellow circle is not on the path of the larger circle. This is because we need to translate the origin of the coordinates to the centre of the canvas (the large circle). Remember the diagram (fig.2).

Challenge

Can you work out how to do that?

Figure B2.4



Sketch B2.5 translate

When we translate the co-ordinates for the origin, we shift everything to where we need it, similar to rotating the baton.

```
let radius = 100
let angle = 0
let x
let y

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

function draw()
{
  background(220)
  noFill()
  circle(width/2, height/2, radius * 2)
  translate(width/2, height/2)
  x = radius * sin(angle)
  y = radius * cos(angle)
  fill('yellow')
  circle(x, y, 20)
}
```

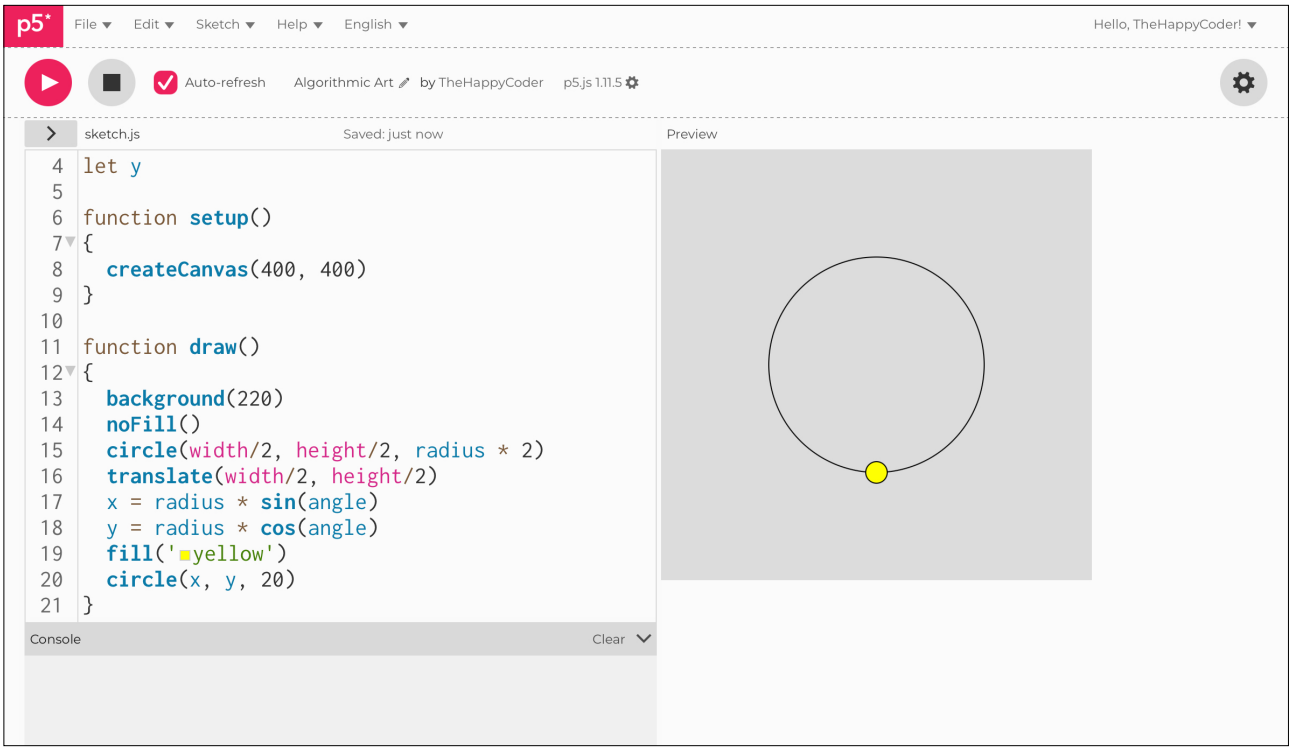
Notes

It should sit nicely on the large circle.

Challenges

1. Change the value of the angle.
2. What happens if you translate before drawing the large circle?

Figure B2.5



Sketch B2.6 moving it

Our final step is to get the yellow circle moving. We add another variable called `velocity` (angular) and increment the `angle` according to that variable.

```
let radius = 100
let angle = 0
let x
let y
let velocity = 0.05

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

function draw()
{
  background(220)
  noFill()
  circle(width/2, height/2, radius * 2)
  translate(width/2, height/2)
  x = radius * sin(angle)
  y = radius * cos(angle)
  fill('yellow')
  circle(x, y, 20)
  angle += velocity
}
```

Notes

You should see the yellow circle orbiting the big circle. We give it an initial angular velocity of `0.05`.

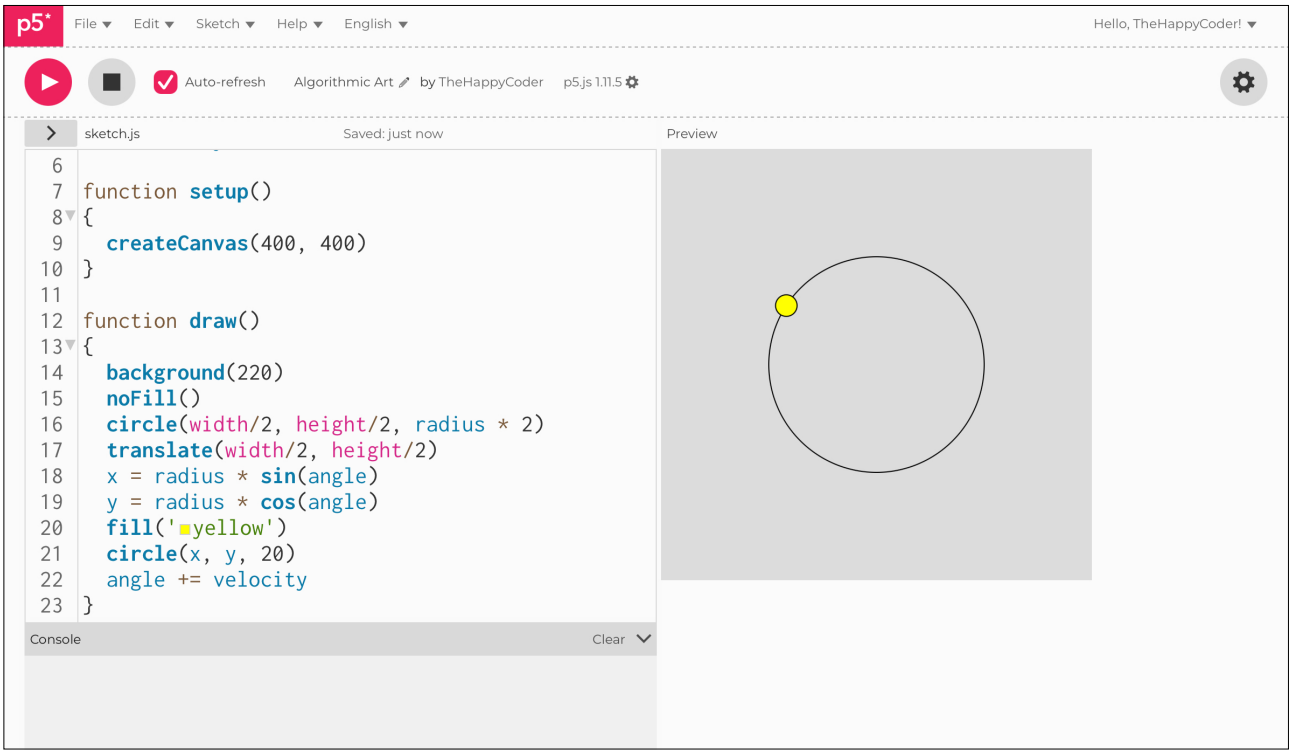
Challenges

1. Try other values for the velocity.
2. What happens if you give it a negative velocity?
3. How would you have more than one circle?

Code Explanation

`angle += velocity`	Changing the angle by that (velocity) amount.

Figure B2.6



Sketch B2.7 accelerating

Increasing the angular velocity.

```
let radius = 100
let angle = 0
let x
let y
let velocity = 0.05
let acceleration = 0.0005

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

function draw()
{
  background(220)
  noFill()
  circle(width/2, height/2, radius * 2)
  translate(width/2, height/2)
  x = radius * sin(angle)
  y = radius * cos(angle)
  fill('yellow')
  circle(x, y, 20)
  angle += velocity
  velocity += acceleration
}
```

Notes

In this, we simply accelerate the orbiting circle by increasing the velocity incrementally.

Challenges

1. What happens if you start with a negative velocity?
2. How would you go about changing the direction of rotation every time it reaches the top position?

Simple Harmonic Motion (SHM)

We have moved an object (yellow circle) scribing a circle using the co-ordinates of all the points on a circle. Using sine and cosine, we can describe other motions such as linear. This motion is called simple harmonic motion, where the oscillation is about a central point.

We can also use this to describe an ellipse (oval) as well as other effects that can be useful in creative coding.

Sketch B2.8 a line and a circle

! Start a new sketch

We are drawing a red circle at the end of the line.

```
let angle = 0
let x

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

function draw()
{
  background(220)
  translate(width/2, height/2)
  x = 150
  line(0, 0, x, 0)
  fill('red')
  circle(x, 0, 20)
}
```

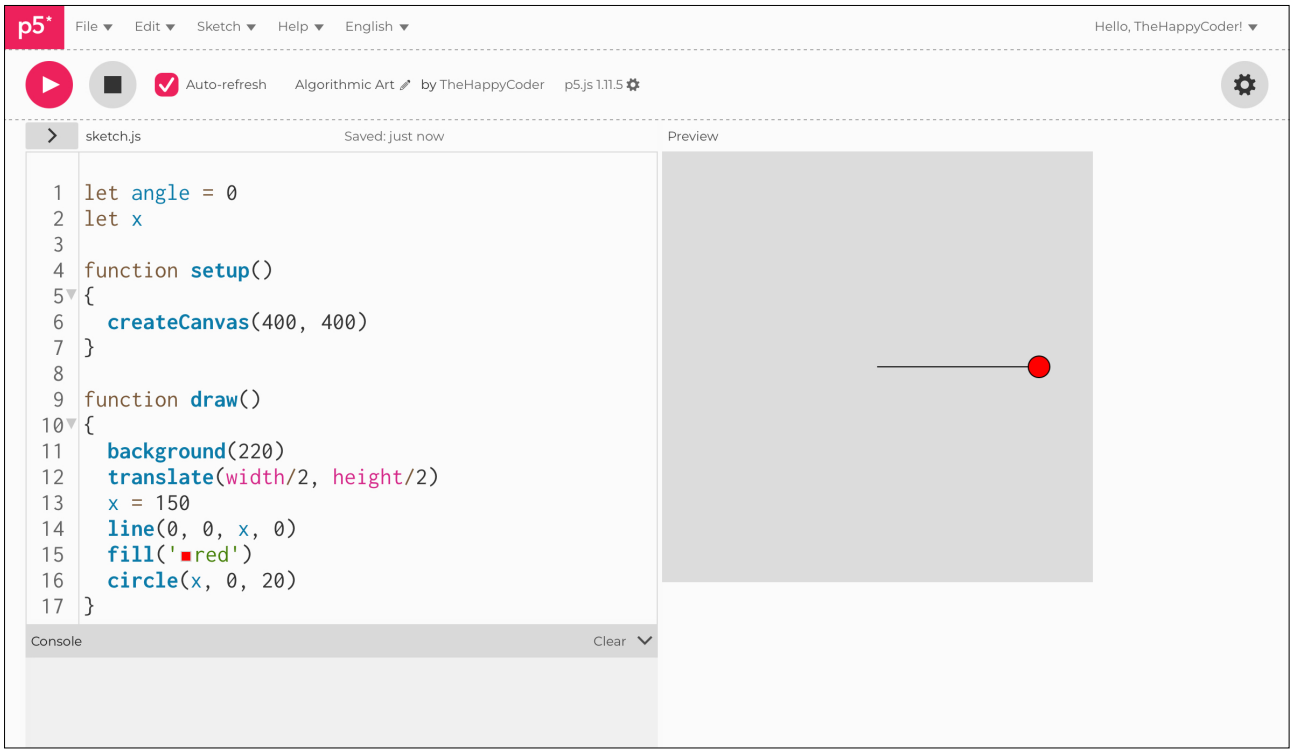
Notes

All very static to start with.

Code Explanation

`translate(width/2, height/2)`	Translates all the coordinates to the centre of the canvas.
`x = 150`	The x-coordinate is initialised to 150.
`line(0, 0, x, 0)`	Draw a line from the centre of the canvas to the x-coordinate.
`circle(x, 0, 20)`	Draw a red circle at the end of the line where y is kept at 0 (central plane of the canvas).

Figure B2.8



Sketch B2.9 amplitude

Instead of radius, we will use the variable name amplitude. We are only doing this for the x value for now.

```
let angle = 0
let x
let amplitude = 150

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

function draw()
{
  background(220)
  translate(width/2, height/2)
  x = amplitude * sin(angle)
  line(0, 0, x, 0)
  fill('red')
  circle(x, 0, 20)
}
```

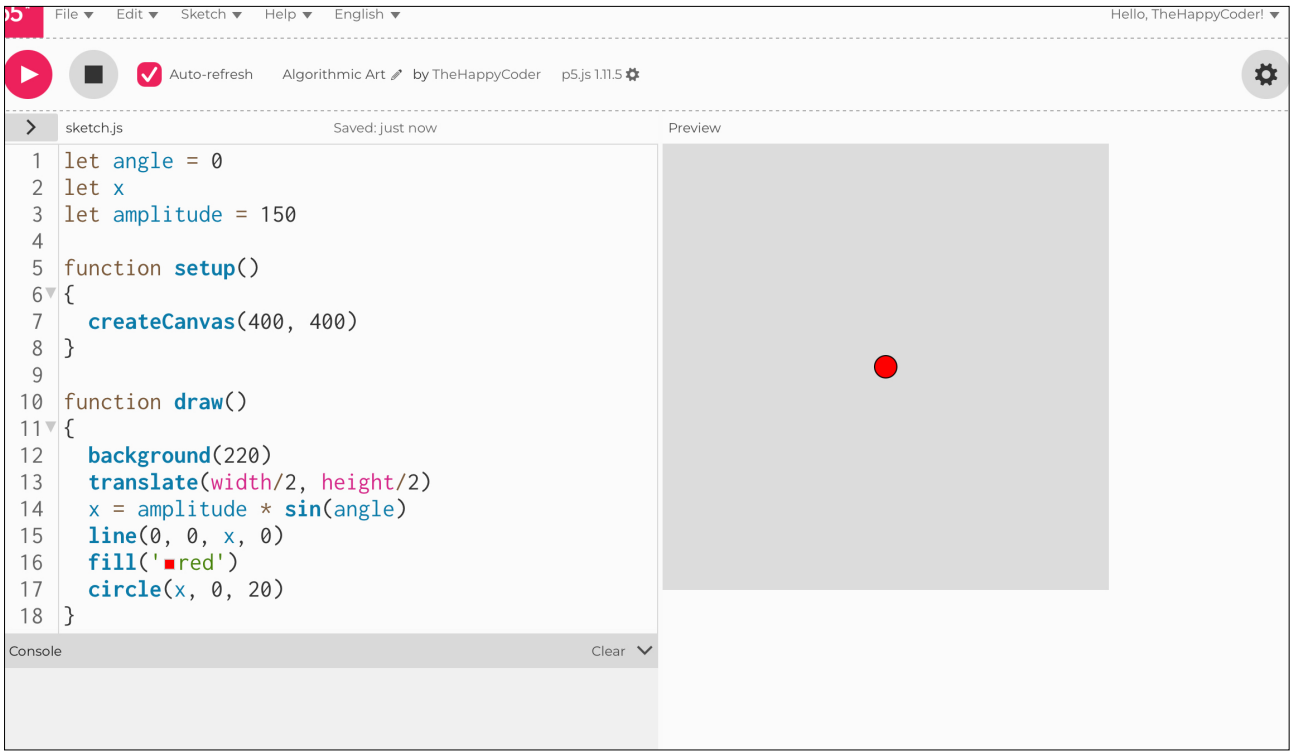
Notes

It moves the red circle to the centre; the line is still there!

Code Explanation

`x = amplitude * sin(angle)`	Defining the position of the x component of the coordinates.

Figure B2.9



Sketch B2.10 moving it

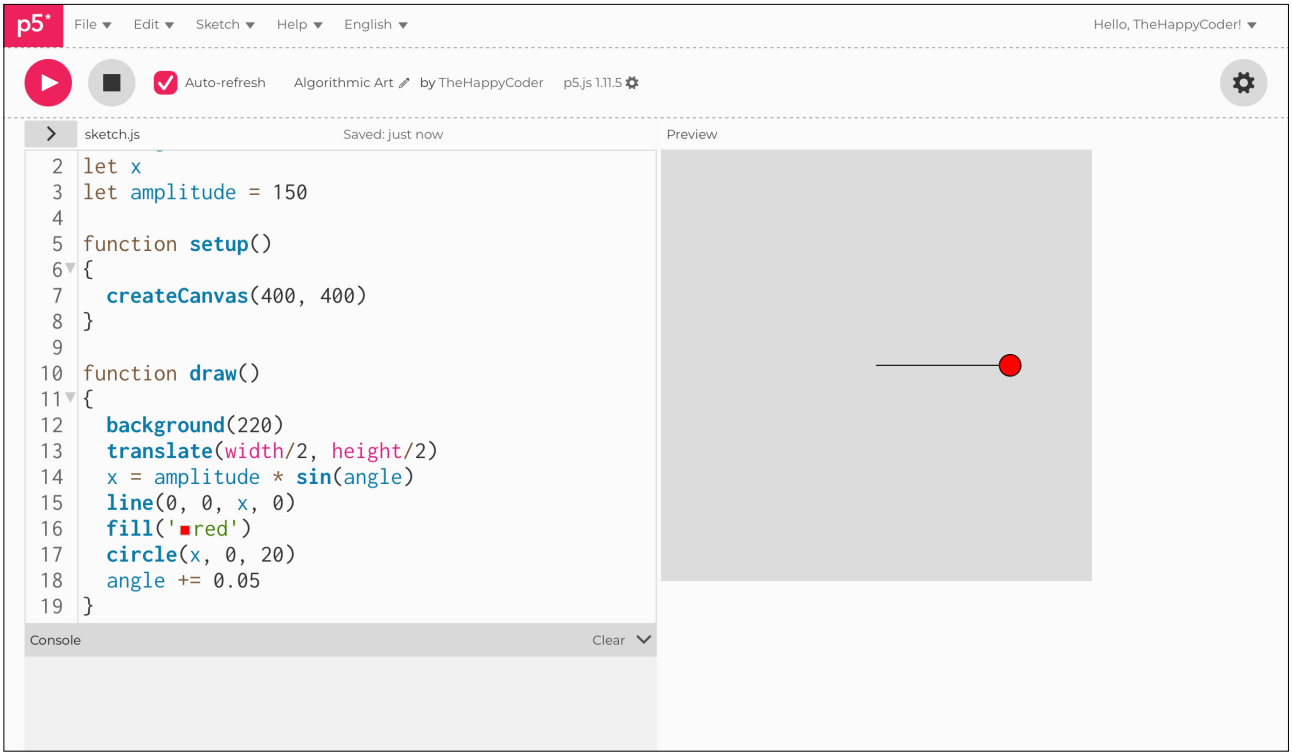
Although we are using sine wave motion, we now have a smooth, simple harmonic motion (SHM). This is similar to the motion of the circle but in a linear direction only.

```
let angle = 0
let x
let amplitude = 150

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

function draw()
{
  background(220)
  translate(width/2, height/2)
  x = amplitude * sin(angle)
  line(0, 0, x, 0)
  fill('red')
  circle(x, 0, 20)
  angle += 0.05
}
```

Notes

Now it oscillates nicely about the origin.

Figure B2.10



Sketch B2.11 an ellipse

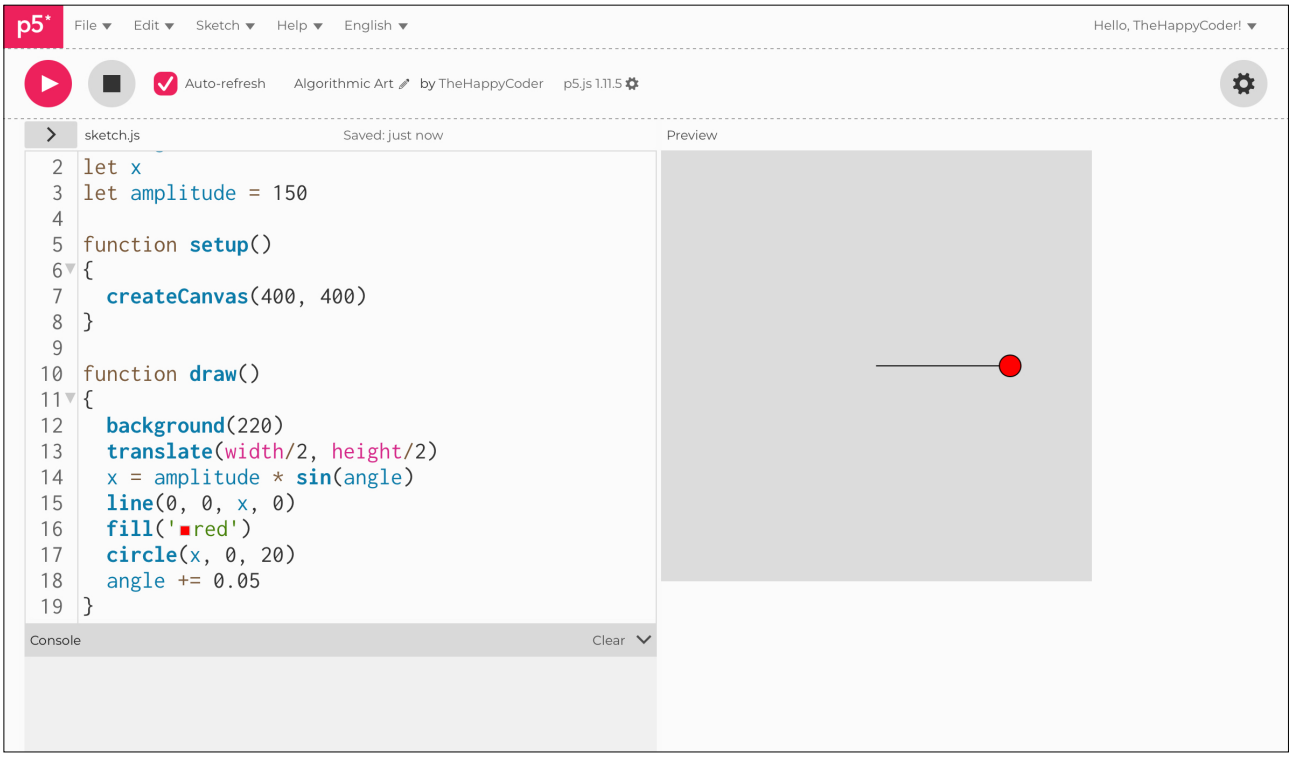
A circle or linear movement is all very well, but we can also do an ellipse. Before we make too many changes at once, let's take it one step at a time. We will change the name of the `amplitude` to `amplitudeX`.

```
let angle = 0
let x
let amplitudeX = 150

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

function draw()
{
  background(220)
  translate(width/2, height/2)
  x = amplitudeX * sin(angle)
  line(0, 0, x, 0)
  fill('red')
  circle(x, 0, 20)
  angle += 0.05
}
```

Notes

Nothing drastically changing, just renaming.

Figure B2.12



Sketch B2.12 the y component

To draw an ellipse, we need the y component, as we did with the circle. They should make sense. The difference being that we have a different amplitude (radius) for y than for x, and it is this that gives it the ellipse shape.

```
let angle = 0
let x
let y
let amplitudeX = 150
let amplitudeY = 50

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

function draw()
{
  background(220)
  translate(width/2, height/2)
  x = amplitudeX * sin(angle)
  y = amplitudeY * cos(angle)
  line(0, 0, x, y)
  fill('red')
  circle(x, y, 20)
  angle += 0.05
}
```

Notes

Adding in the y component to the circle and the line, you now have it scribing an elliptical shape.

Here we can play with many different variables. I will just show you a few and highly recommend that you play with them to see what patterns you can create. Later on, we will create a pattern called phyllotaxis, which is very similar to what has been covered here.

Sketch B2.13 moving the background

If we move the `background()` into the `setup()` function, it draws the background only once. We can use `//` at the beginning of a line of code for four reasons:

1. We can remove unwanted code for the moment but may want it later.
2. There is a problem with our code and removing a line of code means it can help us debug (find the problem) the code.
3. We can leave comments or information that is helpful for yourself or someone else looking at the code.
4. The `//` has the effect of making it invisible to the computer and will ignore anything on that line.

Here we are commenting out the `background()` and the `line()` in the `draw()` function. I have highlighted them.

```
let angle = 0
let x
let y
let amplitudeX = 150
let amplitudeY = 50

function setup()
{
  createCanvas(400, 400)
  background(220)
}

function draw()
{
  // background(220)
  translate(width/2, height/2)
  x = amplitudeX * sin(angle)
  y = amplitudeY * cos(angle)
  // line(0, 0, x, y)
  fill('red')
  circle(x, y, 20)
  angle += 0.05
}
```

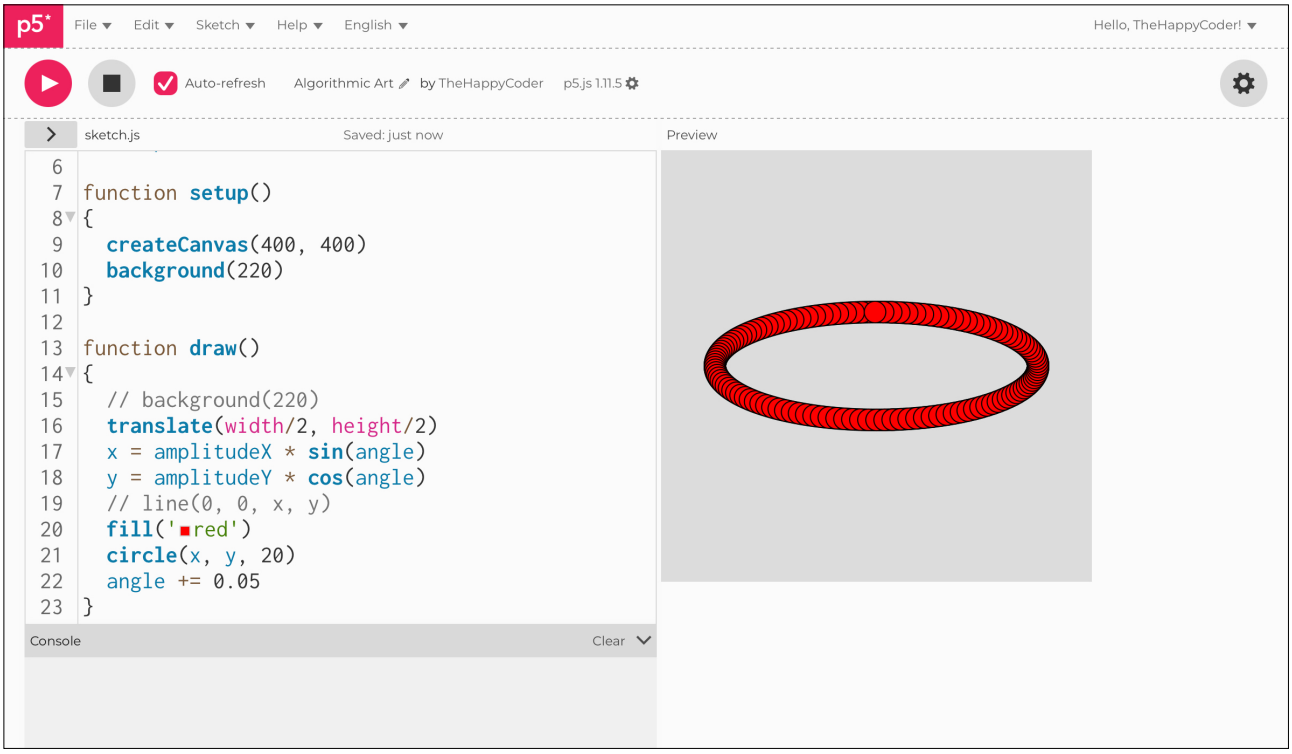
Notes

This gives an animated response as if drawing on the canvas.

Code Explanation

`// background(220)`	Ignores this line of code.
`// line(0, 0, x, y)`	Ignores this line of code as well.

Figure B2.13



Sketch B2.14 pattern (1)

! Remove the commented-out (//) lines of code, this demonstrates one use of //.

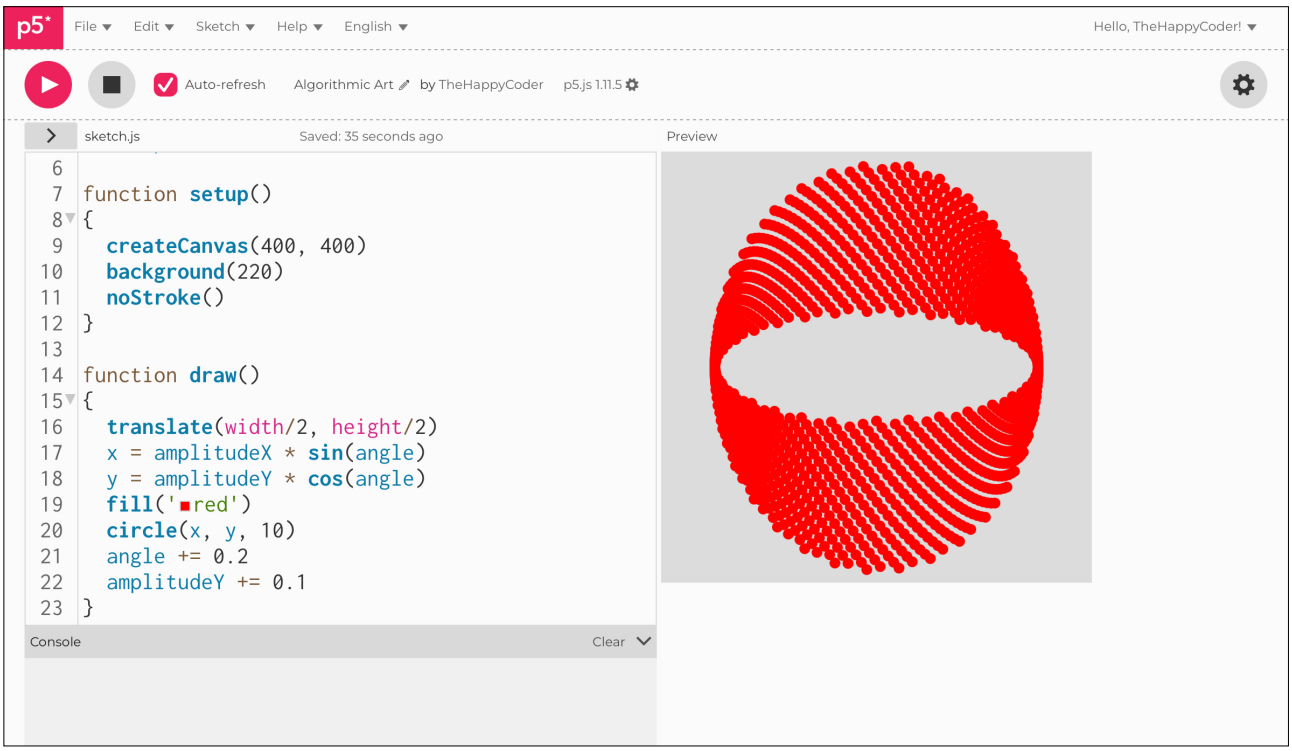
We have added a noStroke(), made the circle a bit smaller, changed the rate of angle, and most importantly, incremented the amplitude of the y component, amplitudeY.

```
let angle = 0
let x
let y
let amplitudeX = 150
let amplitudeY = 50

function setup()
{
  createCanvas(400, 400)
  background(220)
  noStroke()
}

function draw()
{
  translate(width/2, height/2)
  x = amplitudeX * sin(angle)
  y = amplitudeY * cos(angle)
  fill('red')
  circle(x, y, 10)
  angle += 0.2
  amplitudeY += 0.1
}
```

Notes

A different sort of pattern emerges.

Figure B2.14



Sketch B2.15 pattern (2)

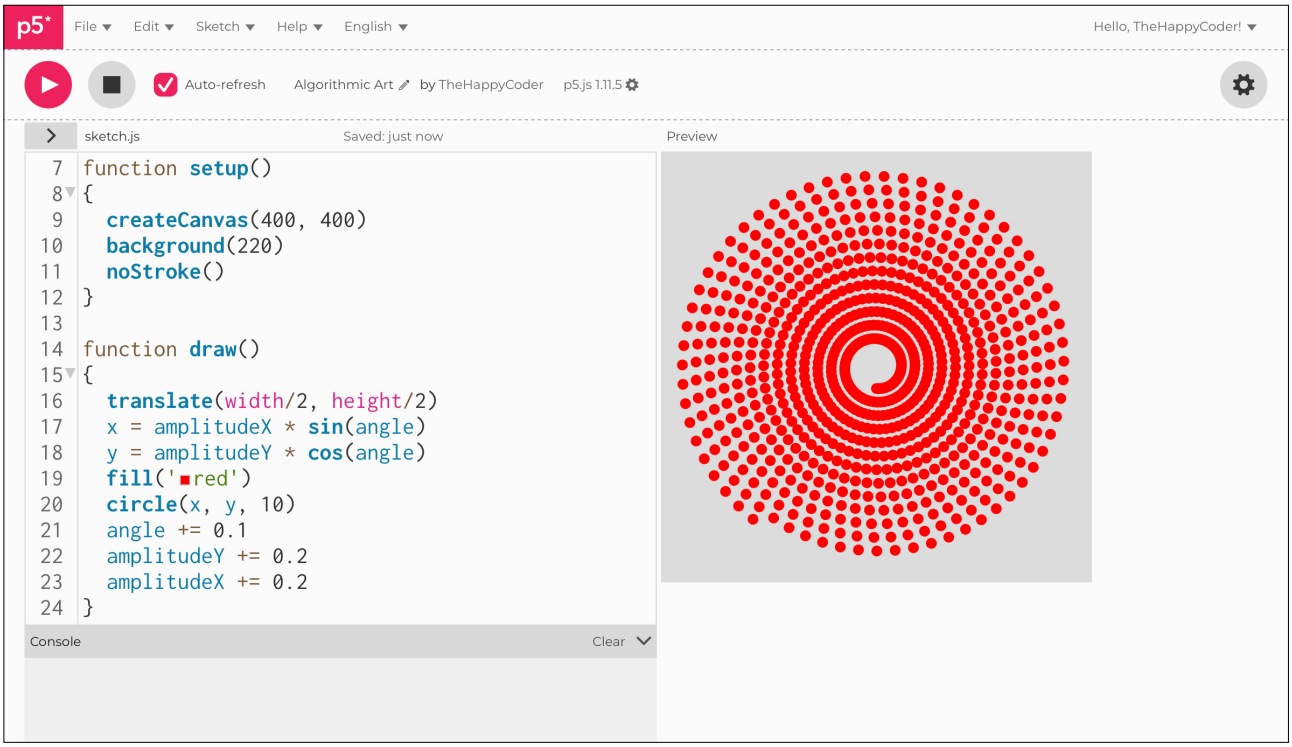
Changing the amplitude for x and y. Also, alter the starting amplitude for both of them and the angle increment.

```
let angle = 0
let x
let y
let amplitudeX = 20
let amplitudeY = 20

function setup()
{
  createCanvas(400, 400)
  background(220)
  noStroke()
}

function draw()
{
  translate(width/2, height/2)
  x = amplitudeX * sin(angle)
  y = amplitudeY * cos(angle)
  fill('red')
  circle(x, y, 10)
  angle += 0.1
  amplitudeY += 0.2
  amplitudeX += 0.2
}
```

Notes

A nice spiral effect.

Figure B2.15



Sketch B2.16 pattern (3)

Adding in the `diameter` variable and changing some of the other values.

```
let angle = 0
let x
let y
let amplitudeX = 0
let amplitudeY = 0
let diameter = 1

function setup()
{
  createCanvas(400, 400)
  background(220)
  noStroke()
}

function draw()
{
  translate(width/2, height/2)
  x = amplitudeX * sin(angle)
  y = amplitudeY * cos(angle)
  fill('red')
  circle(x, y, diameter)
  angle += 0.5
  amplitudeY += 0.2
  amplitudeX += 0.2
  diameter += 0.01
}
```

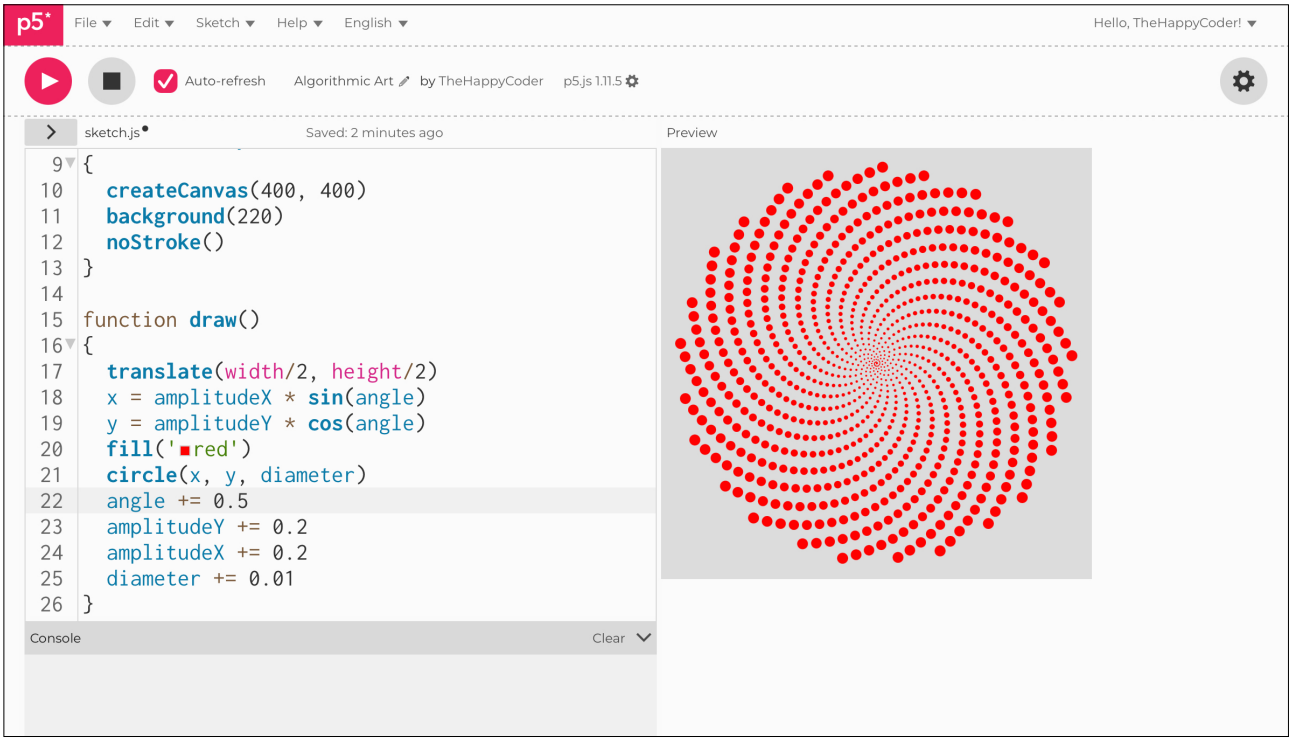
Notes

Just shows what can be achieved with a few variables.

Challenge

The idea is to be creative and make your own designs and patterns through playing. Adding colour, other shapes and levels of alpha, etc.

Figure B2.16



The Joy of Coding Algorithmic Art Module B Unit #3 text

Module B Unit #3: using text

As well as shapes, you can use text, either directly as a string or as a variable value. This gives you many opportunities to manipulate text (words, letters or numbers) or values (numbers) for various possibilities. Although you get one font with the basic sketch, you can use other fonts. Some of them you can download and others are generally available to your computer.

Key concepts:

Creating text
Text size
Text colour
Text position
Fonts
Variables

Sketch B3.1 starting sketch

We have a line of code that uses the `text()` function. The actual text is in speech marks (single or double), this is a string. The following two arguments are the distance from the left-hand side (`x`) and the distance from the top edge (`y`).

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

function draw()
{
  background(220)
  text('Hello', 200, 200)
}
```

Notes

You get to print some text on the canvas; the default size is quite small, but we can change that.

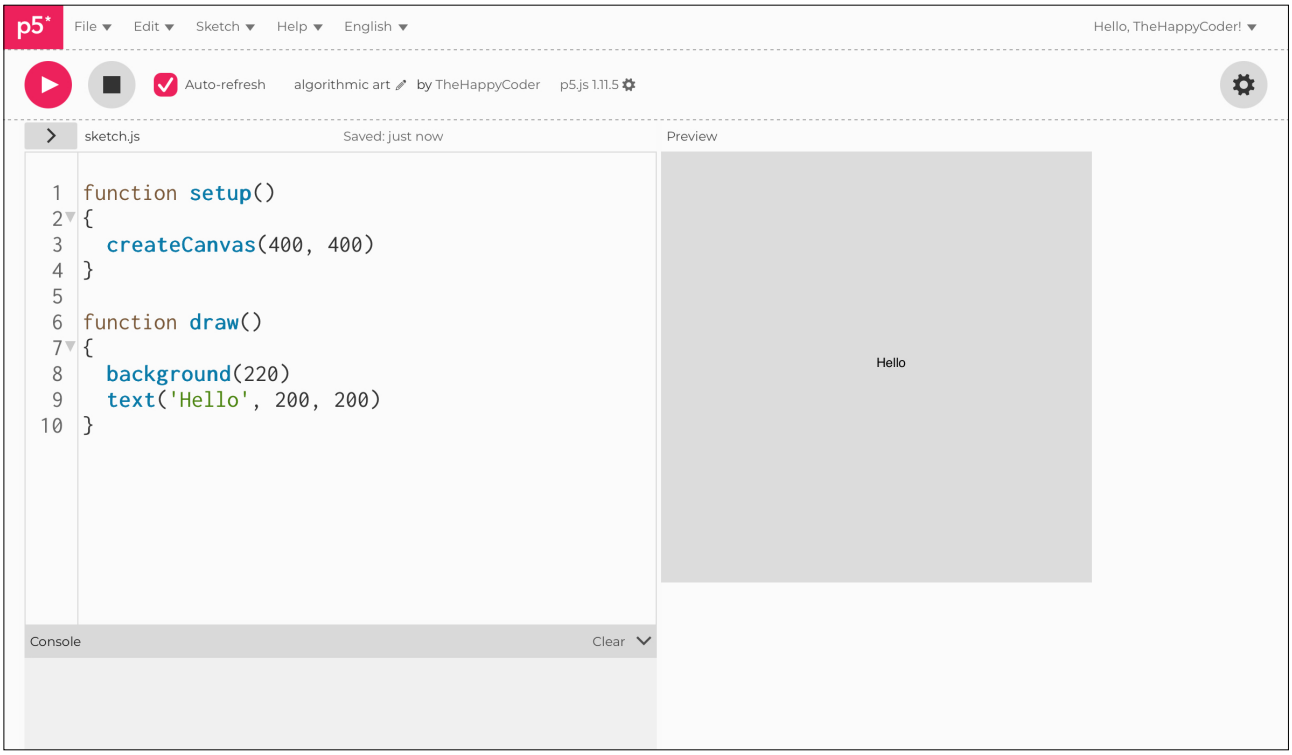
Challenge

Type some other words.

Code Explanation

`text('Hello', 200, 200)`	Write the word 'Hello' 200 from the left and 200 from the top.

Figure B3.1



Sketch B3.2 text size

You will notice that it is quite small; the default size is 12 pixels. We can make it bigger than that using the function `textSize()` and specifying the size of the font.

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

function draw()
{
  background(220)
  textSize(36)
  text('Hello', 200, 200)
}
```

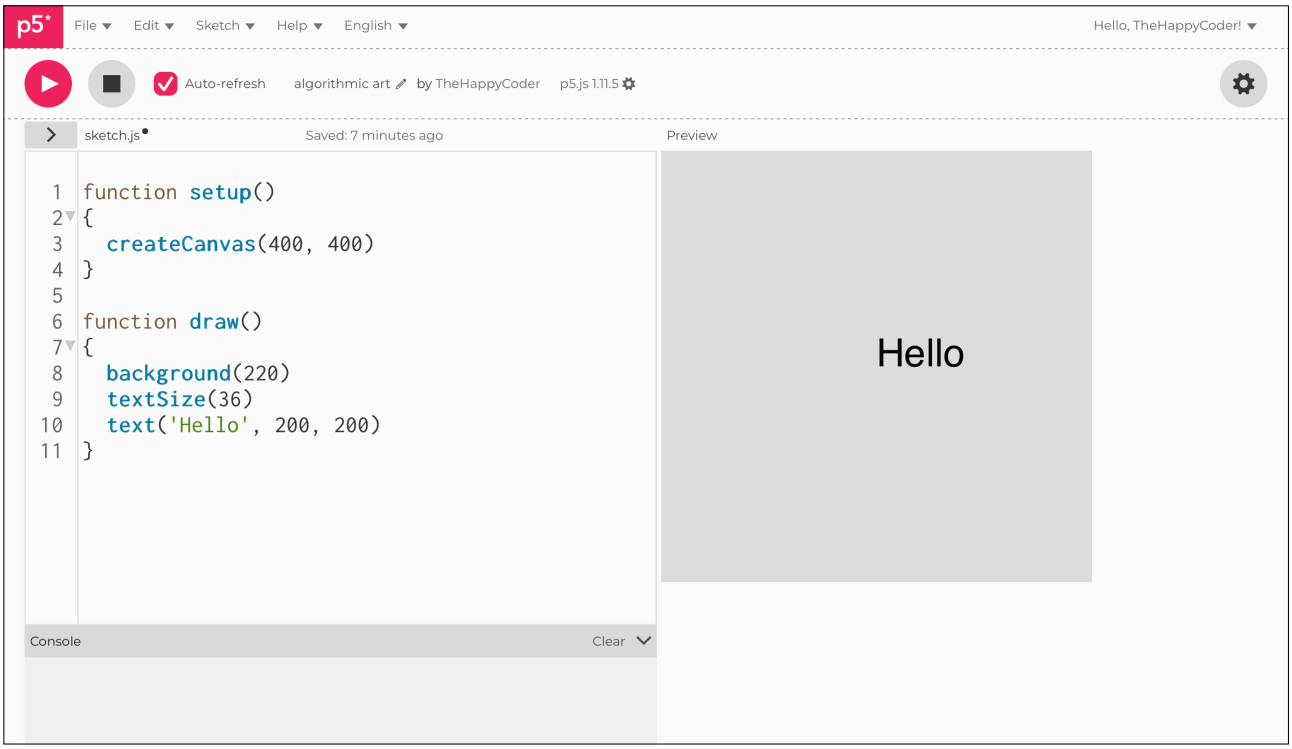
Notes

Much better.

Code Explanation

`textSize(36)`	Specifies the size of the text.

Figure B3.2



Sketch B3.3 aligning the text

Although we have set the text in the centre of the canvas, it clearly isn't. We can correct that so that the coordinates are specific to the centre of the text, not the top-left-hand corner. We use a function called `textAlign()`. We can tell it where we want the origin to be. In this case, at the very centre of the text, so we use two arguments for the horizontal and vertical alignment.

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

function draw()
{
  background(220)
  textSize(36)
  textAlign(CENTER, CENTER)
  text('Hello', 200, 200)
}
```

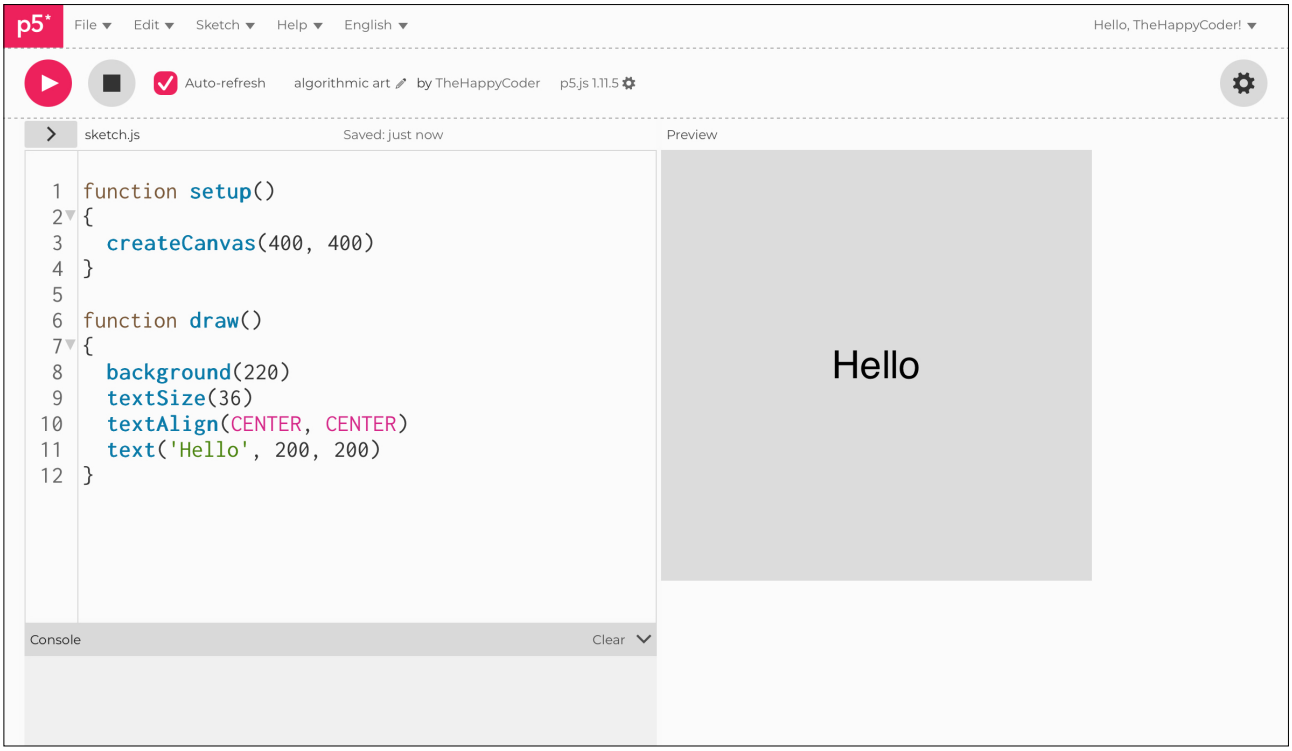
Notes

It has shifted the text ever so slightly to the centre of the canvas.

Code Explanation

`textAlign(CENTER, CENTER)`	Centres the text both vertically and horizontally.

Figure B3.3



Sketch B3.4 colour the text

We will make the text size even bigger to see the effect more clearly. We will also make the text red.

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

function draw()
{
  background(220)
  textSize(136)
  textAlign(CENTER, CENTER)
  fill('red')
  text('Hello', 200, 200)
}
```

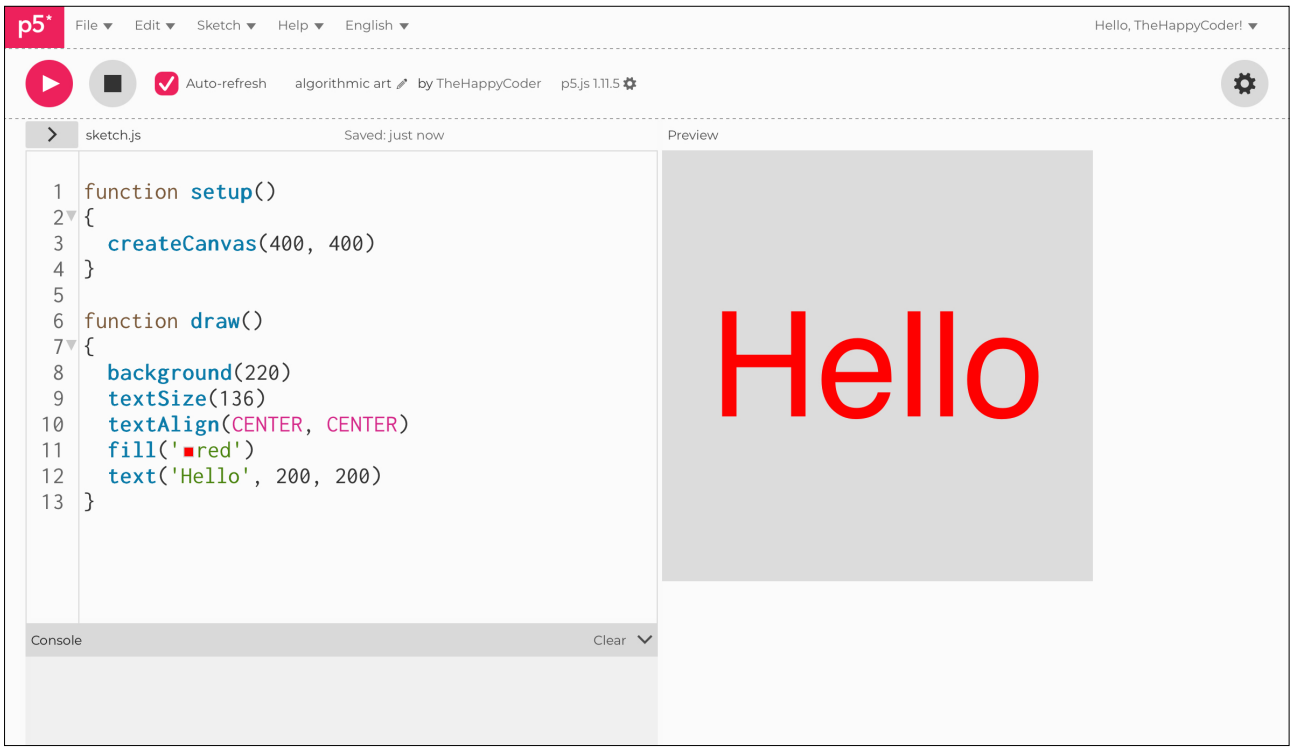
Notes

With smallish text, we get a slight distortion with the colours.

Challenge

Try other colours and alpha.

Figure B3.4



Sketch B3.5 border colour

We can add a border colour using the `stroke()` function.

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

function draw()
{
  background(220)
  textSize(136)
  textAlign(CENTER, CENTER)
  fill('red')
  stroke('blue')
  text('Hello', 200, 200)
}
```

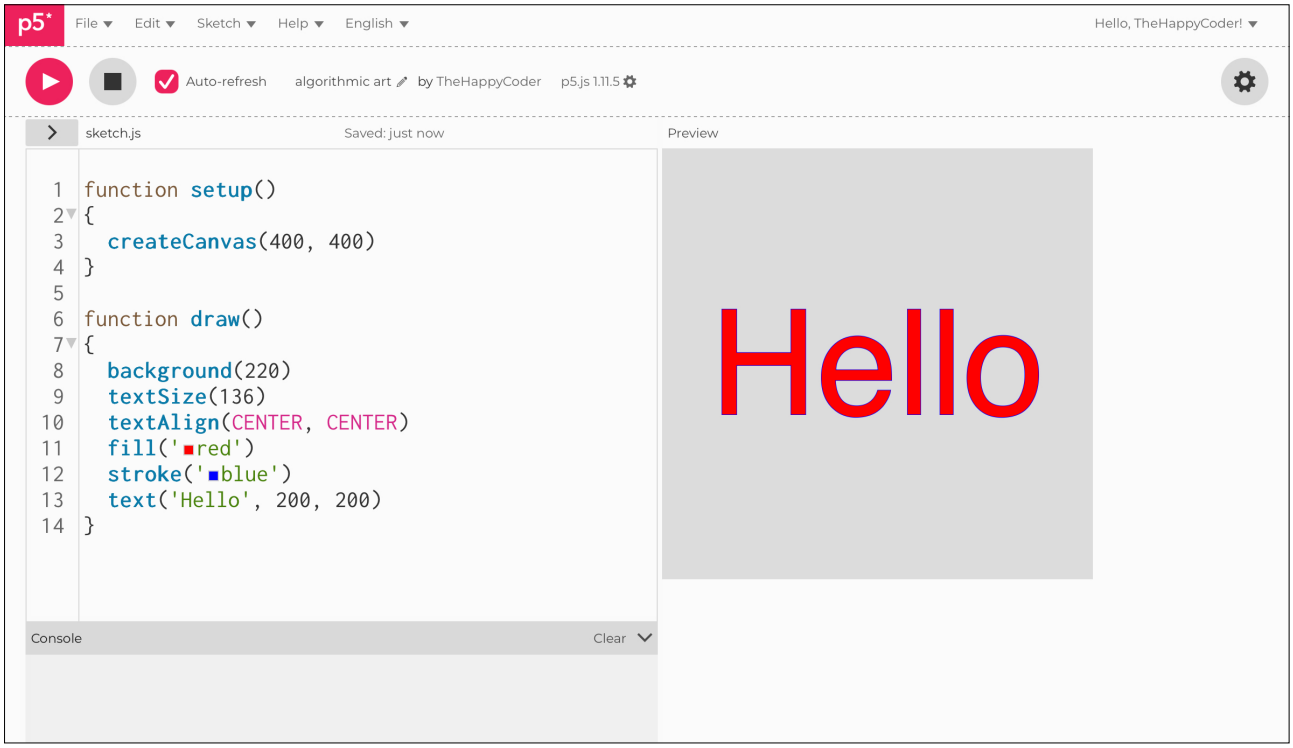
Notes

The blue border around the text is there, it is still only one pixel thick.

Challenge

Alter the `strokeWeight()`.

Figure B3.5



Sketch B3.6 stroke

! Comment out the `fill('red')`

We can do two things: we can increase the `strokeWeight()` and we could have `noFill()`, so we will do both.

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

function draw()
{
  background(220)
  textSize(136)
  textAlign(CENTER, CENTER)
  // fill('red')
  noFill()
  strokeWeight(2)
  stroke('blue')
  text('Hello', 200, 200)
}
```

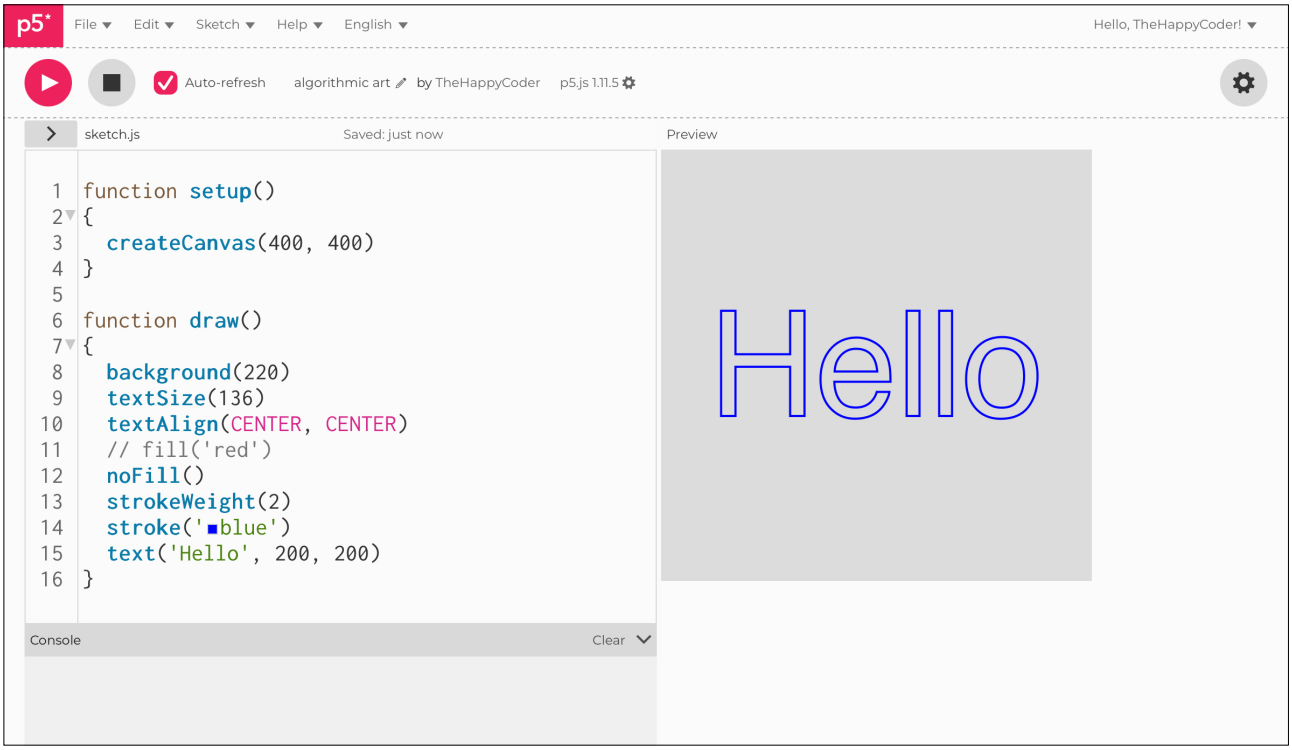
Notes

Remember that when we comment out (//) a line of code, it is ignored.

Challenge

What happens if you don't comment out `fill('red')`?

Figure B3.6



Sketch B3.7 translate

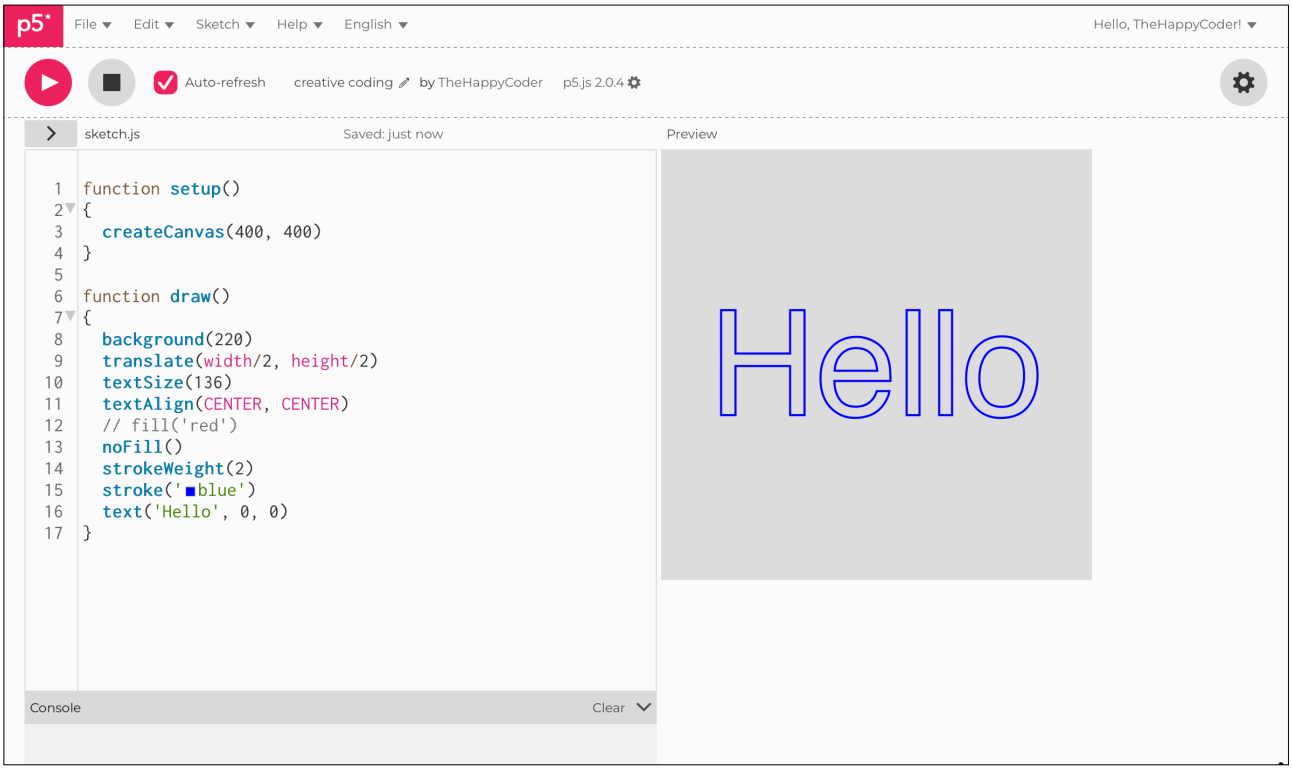
We have another trick up our sleeve: we can rotate the text. Before we do anything, we need to translate the canvas so that the origin is in the centre of the canvas. We will also need to set the coordinates of the text to (0, 0).

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

function draw()
{
  background(220)
  translate(width/2, height/2)
  textSize(136)
  textAlign(CENTER, CENTER)
  // fill('red')
  noFill()
  strokeWeight(2)
  stroke('blue')
  text('Hello', 0, 0)
}
```

Notes

Everything should be as we had before.

Figure B3.7



Sketch B3.8 angle of degree

We want an `angle` variable and we will work in degrees, so hence `angleMode()`. We will set the angle to `90°`.

```
let angle = 90

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

function draw()
{
  background(220)
  translate(width/2, height/2)
  rotate(angle)
  textSize(136)
  textAlign(CENTER, CENTER)
  // fill('red')
  noFill()
  strokeWeight(2)
  stroke('blue')
  text('Hello', 0, 0)
}
```

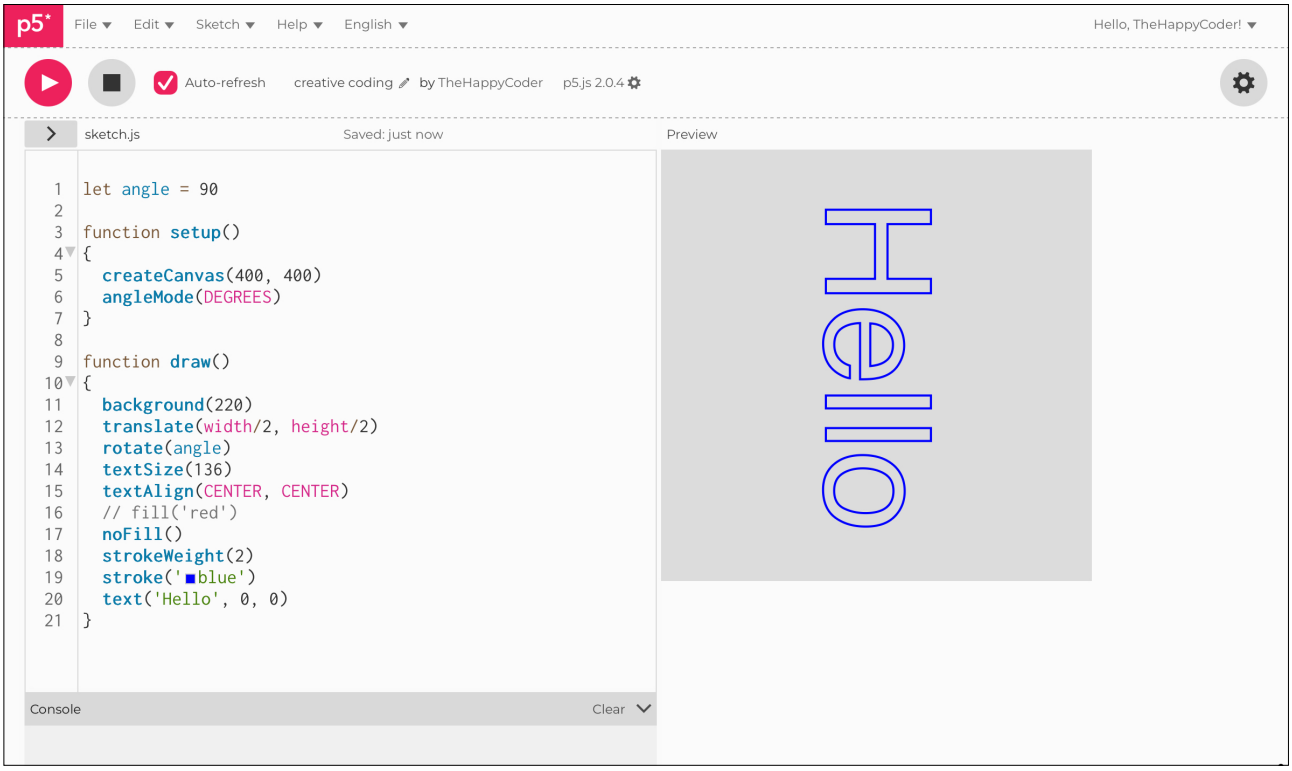
Notes

We have rotated it about the centre of the text.

Challenges

1. Try a different angle.
2. How would you rotate it?

Figure B3.8



Sketch B3.9 rotating

We can increment the angle by 1° and rotate it slowly.

```
let angle = 90

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

function draw()
{
  background(220)
  translate(width/2, height/2)
  rotate(angle)
  textSize(136)
  textAlign(CENTER, CENTER)
  // fill('red')
  noFill()
  strokeWeight(2)
  stroke('blue')
  text('Hello', 0, 0)
  angle += 1
}
```

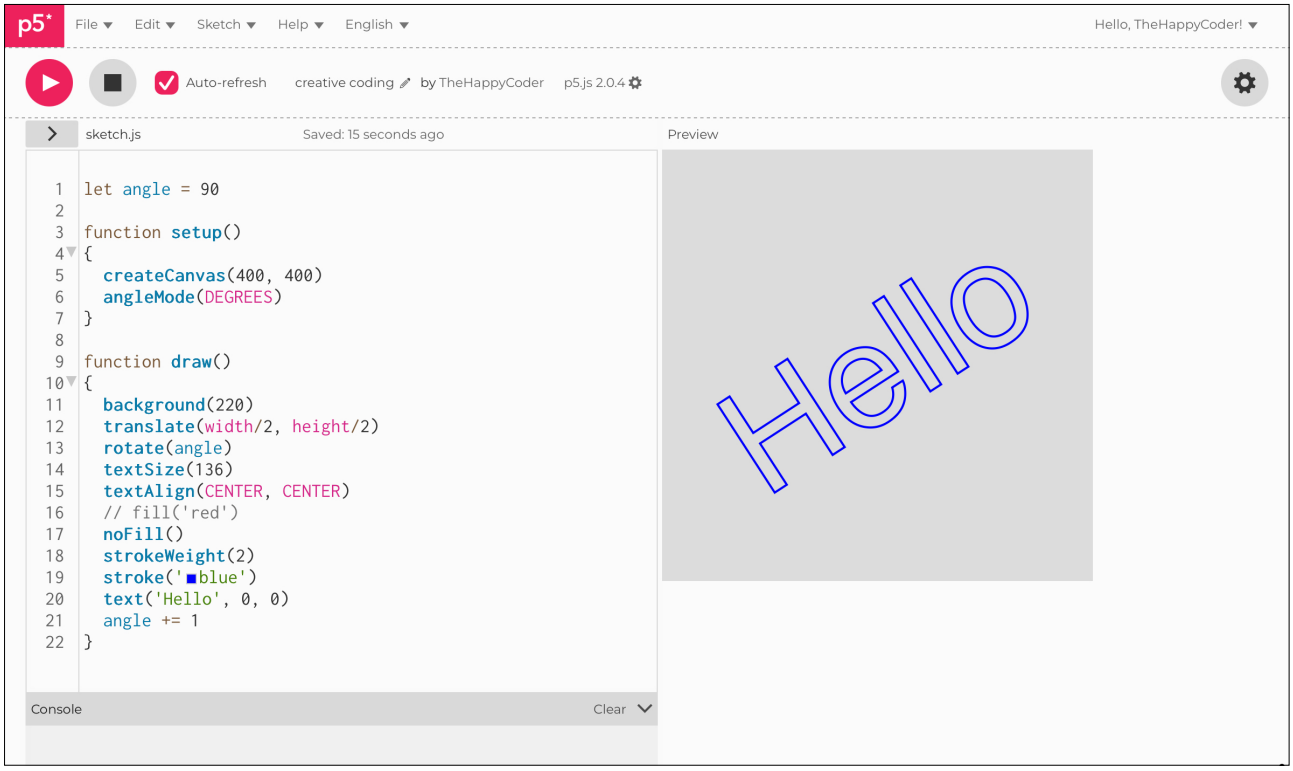
Notes

You should have the text slowly rotating.

Challenges

1. How would you make it go faster or slower?
2. Change colour as it spins.
3. Change size as it spins.

Figure B3.9



Sketch B3.10 mouse mover

! Start a new sketch

We can move text around with the mouse.

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

function draw()
{
  background(220)
  textSize(64)
  text('mouse', mouseX, mouseY)
}
```

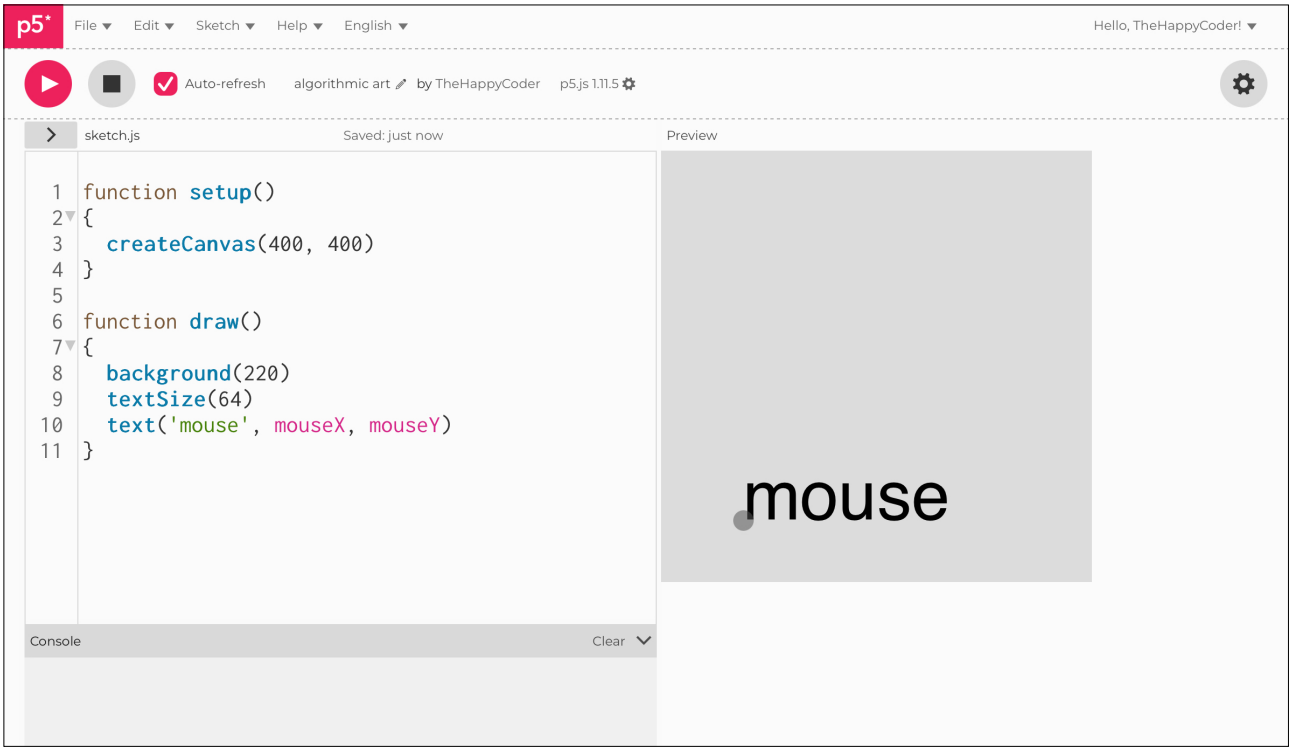
Notes

The text should follow the mouse pointer on the canvas.

Challenge

Could you make the text move in the opposite direction to the mouse?

Figure B3.10



Sketch B3.11 not a string

We can incorporate values not just strings; here, we will get the `x` coordinate of the mouse on the canvas. The `int` means integer (no decimal places).

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

function draw()
{
  background(220)
  textSize(64)
  text(int(mouseX), mouseX, mouseY)
}
```

Notes

We can see the x value on the canvas moving with the mouse.

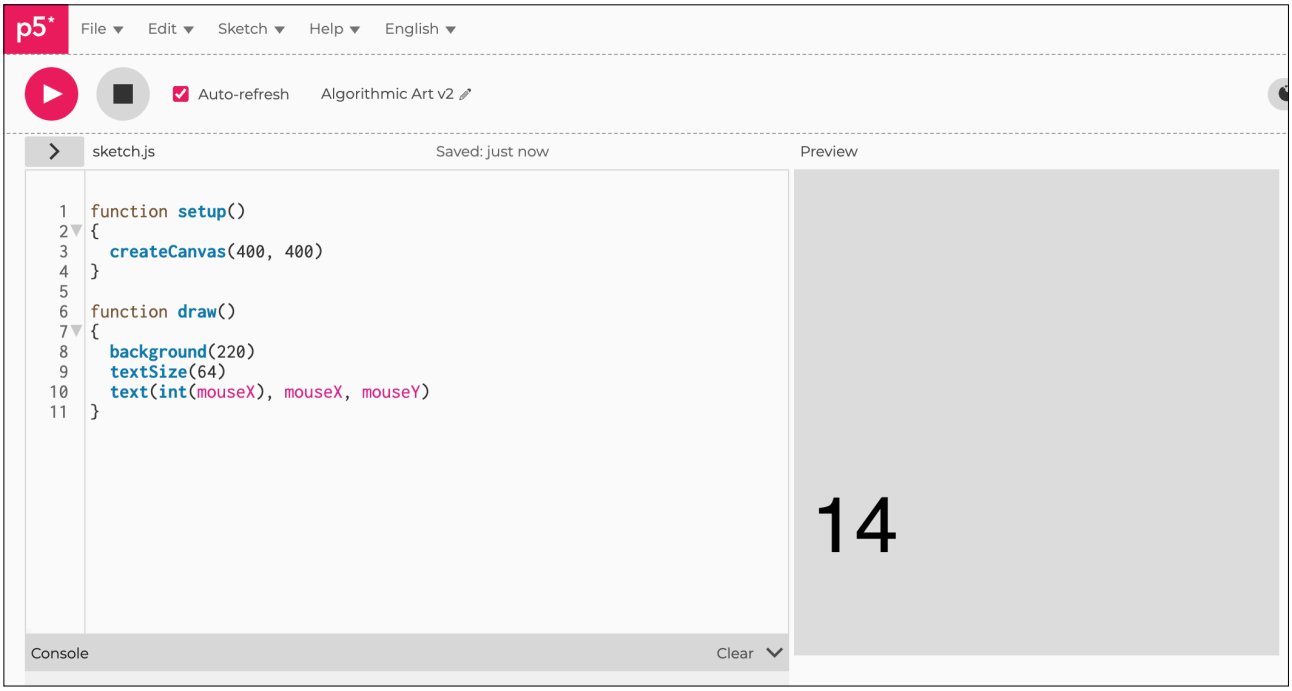
Challenges

1. Remove the `int()`
2. Add the `y` value.

Code Explanation

`text(int(mouseX), mouseX, mouseY)`	Instead of a string, it returns the value of mouseX (first parameter). The int makes it an integer; otherwise, you would get a long float.

Figure B3.11



Sketch B3.12 and mouseY

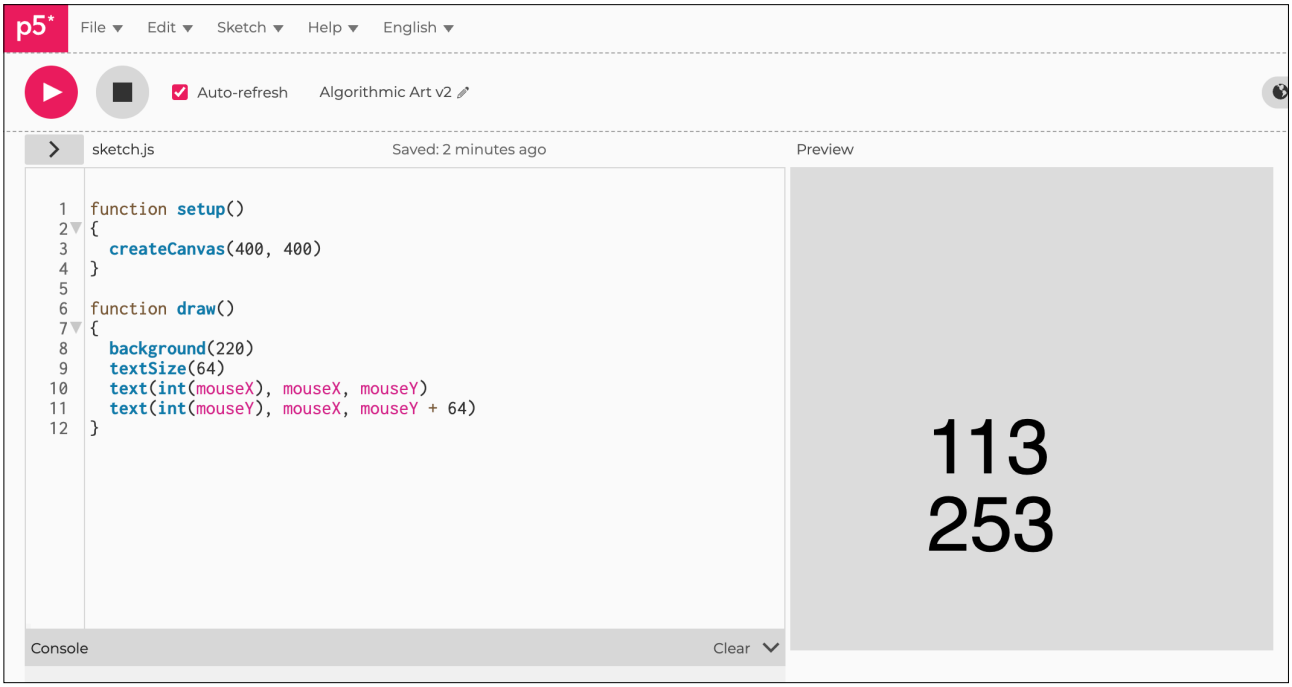
We can add the mouseY coordinate as well.

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

function draw()
{
  background(220)
  textSize(64)
  text(int(mouseX), mouseX, mouseY)
  text(int(mouseY), mouseX, mouseY + 64)
}
```

Notes

We have to separate them, otherwise they will give the numbers on top of each other.

Figure B3.12



Sketch B3.13 static

Instead of following the mouse around, we can have the static values. Removing the integer to show the difference

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

function draw()
{
  background(220)
  textSize(64)
  text(mouseX, 50, 50)
  text(mouseY, 50, 100)
}
```

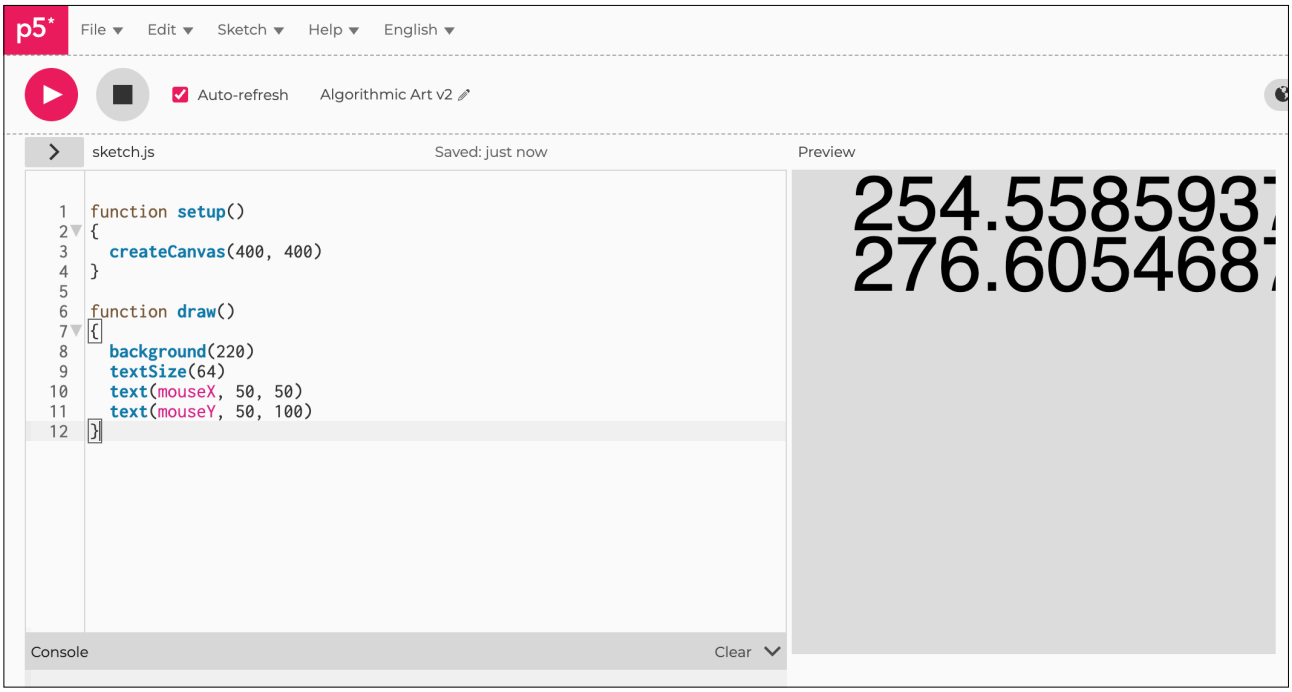
Notes

The number changes, but do not move.

Challenge

What happens if you put the `background()` in the `setup()` function?

Figure B3.13



Sketch B3.14 adding a string

The values were integers, changing values, we can combine strings (words and letters) and integers.

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

function draw()
{
  background(220)
  textSize(64)
  text('x: ' + int(mouseX), 50, 50)
  text('y: ' + int(mouseY), 50, 100)
}
```

Notes

We have both the text and numbers.

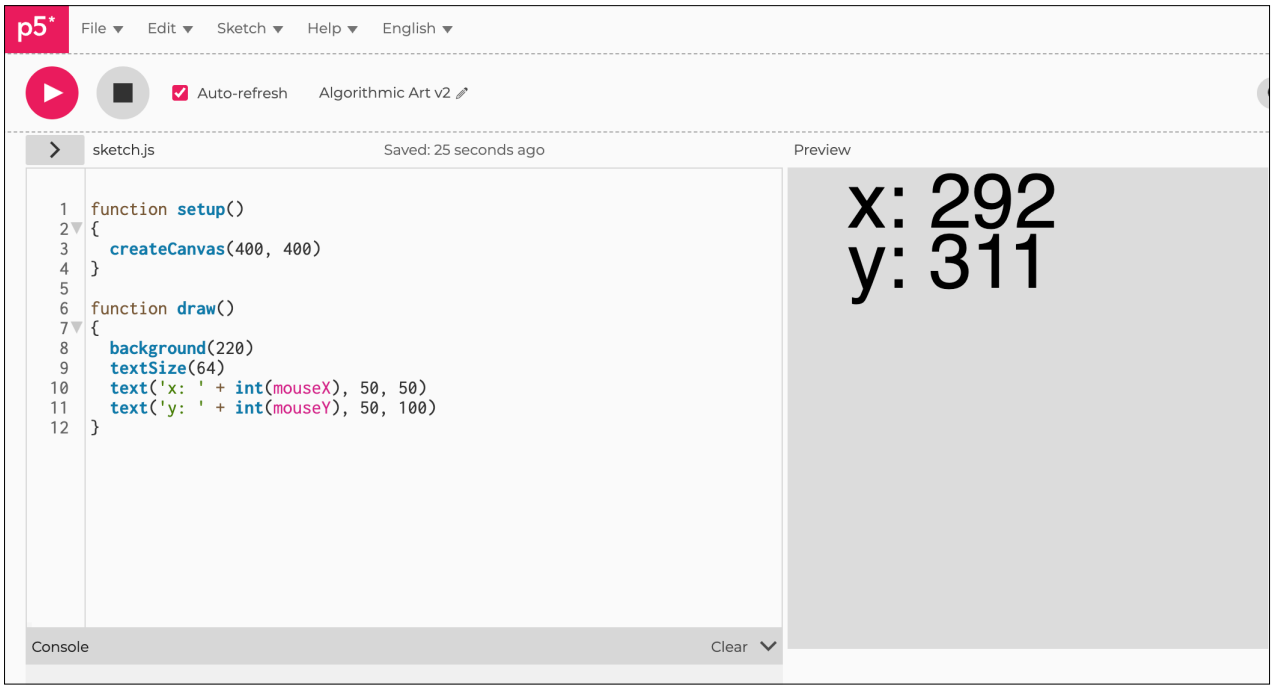
Challenge

Put speech marks around `mouseX` and `mouseY`.

Code Explanation

`text('x: ' + mouseX, 50, 50)`	We can combine text, more text, and values together on the same line.

Figure B3.14



Sketch B3.15 changing the font

We can call on many fonts that may be available to you through your computer. I suggest trial and error but I suspect that the main ones will be available.

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

function draw()
{
  textFont('papyrus')
  fill('white')
  textSize(50)
  text('Algorithmic Art', 35, 200)
}
```

Notes

I have used the font papyrus.

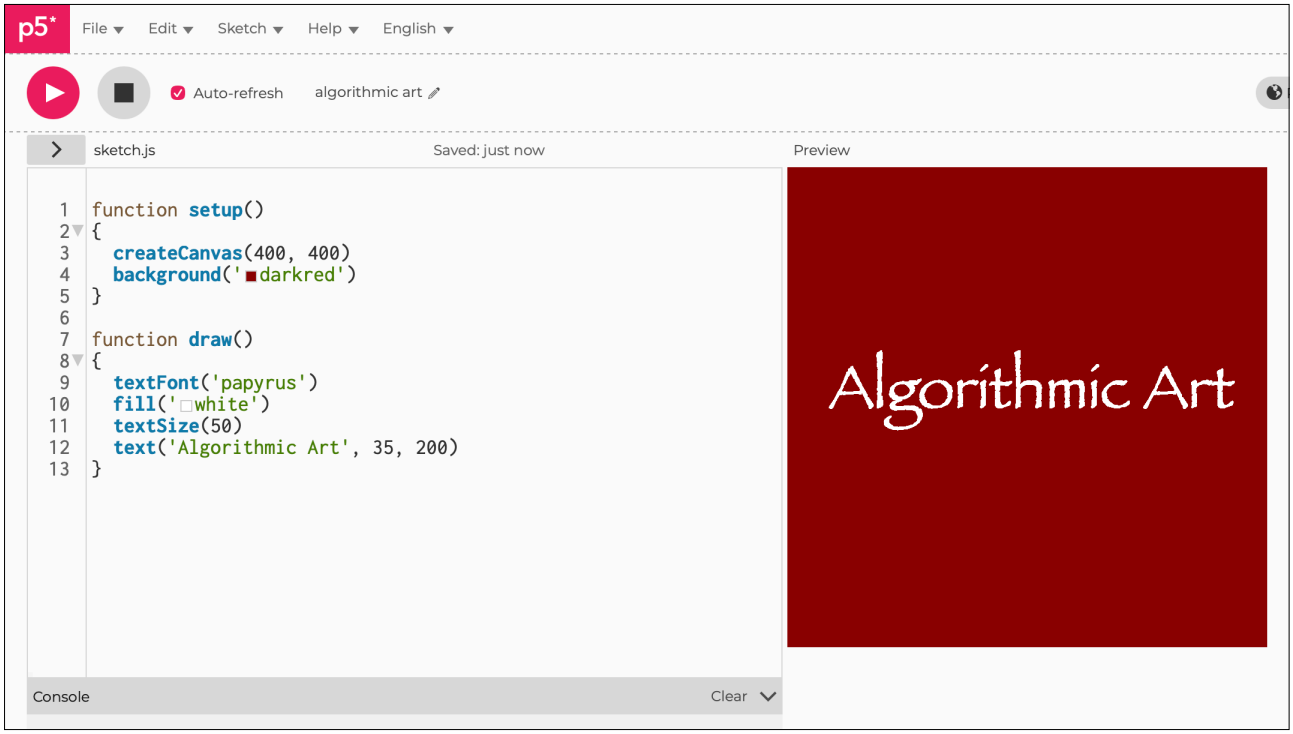
Challenge

Try other fonts.

Code Explanation

`textFont('papyrus')`	The name of the font is in speech marks.

Figure B3.15



The Joy of Coding Algorithmic Art Module B Unit #4 RGB colour slider

Module B Unit #4: RGB and the slider

So far, I have introduced colour as a name and as RGB values with red, green, and blue components. Before jumping into the other colour methods, I just want to expand on RGB.

Key concepts:

colorMode()
sliders
RGB slider colour change

Sketch B4.1 using RGB for the colour

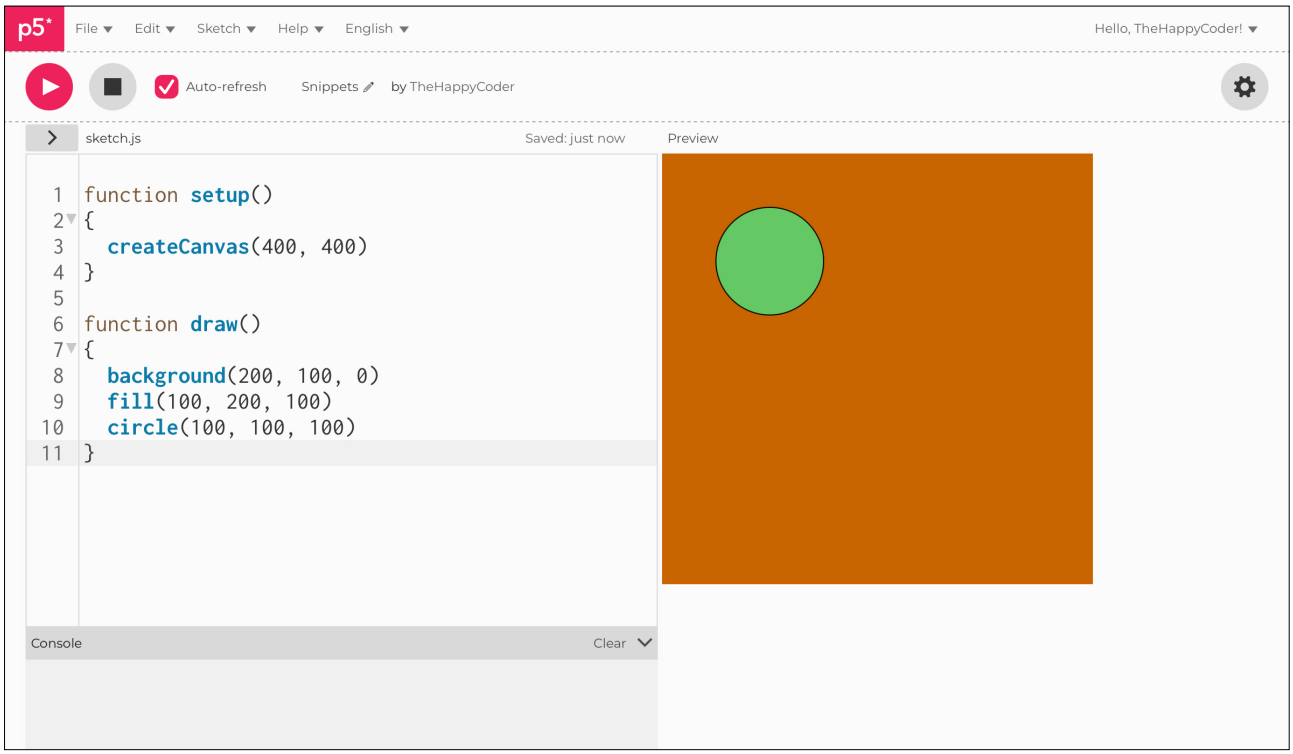
We have already introduced RGB. Here is a reminder: a green circle on an orange background.

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

function draw()
{
  background(200, 100, 0)
  fill(100, 200, 100)
  circle(100, 100, 100)
}
```

Notes

By default, it assumes values (arguments) are for RGB, three for the red, blue, and green; the fourth is the amount of transparency.

Figure B4.1



Sketch B4.2 colorMode()

We have even introduced `colorMode()` briefly. Here we can explore it more as we look at other colour modes. If we have `colorMode(RGB, 100)`, we can scale it to `100` (think of it as a percentage).

```
function setup()
{
  createCanvas(400, 400)
  colorMode(RGB, 100)
}

function draw()
{
  background(100, 100, 0)
  fill(0, 75, 75)
  circle(100, 100, 100)
}
```

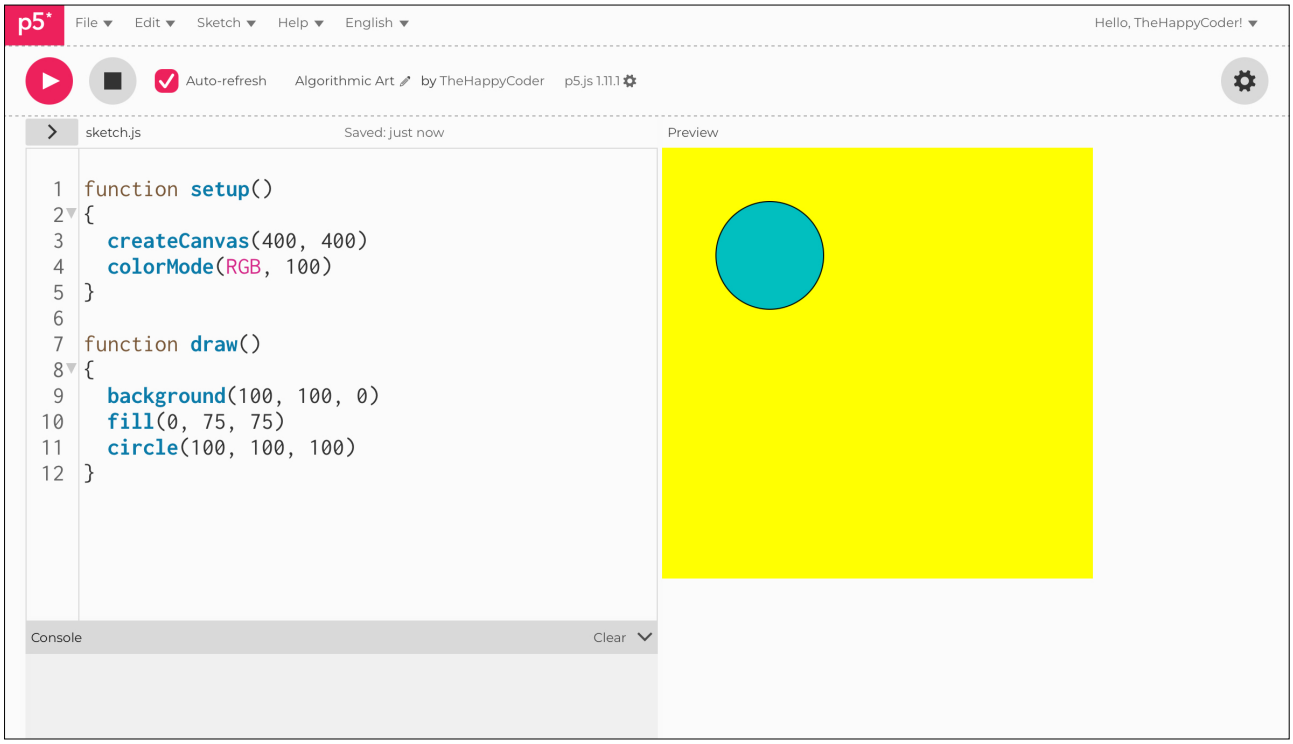
Notes

All the RGB values should be between `0` and `100`. If they exceed that value, then it will take the maximum, in this case `100`.

Challenge

You can add more arguments (any values you like), one for each element including alpha: `colorMode(RGB, 100, 100, 100, 100)`. Try it.

Figure B4.2



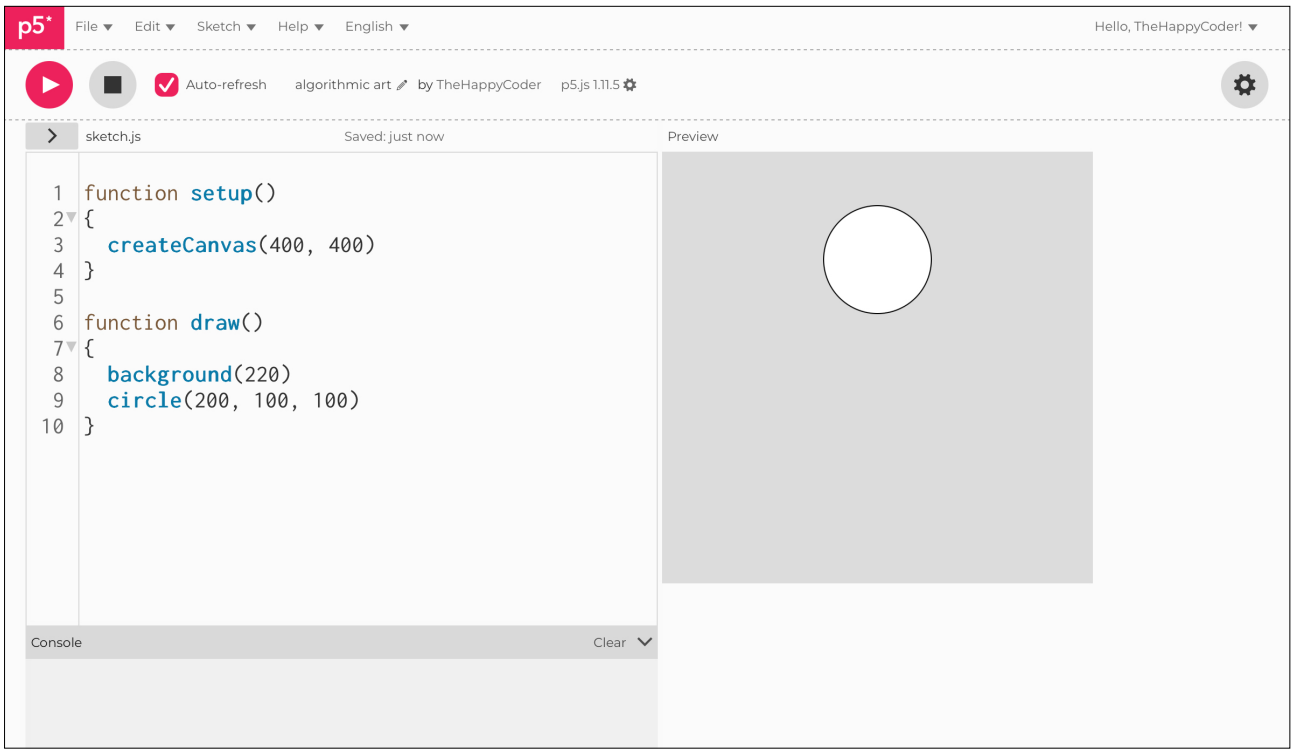
Sketch B4.3 starting sketch

! A new sketch

Circle on a blank canvas.

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

function draw()
{
  background(220)
  circle(200, 100, 100)
}
```

Figure B4.3


p5*
File
Edit
Sketch
Help
English
Hello, TheHappyCoder!
Auto-refresh
algorithmic art
by TheHappyCoder
p5.js 1.11.5
sketch.js
Saved: just now
Preview
1 function setup()
2 {
3 createCanvas(400, 400)
4 }
5
6 function draw()
7 {
8 background(220)
9 circle(200, 100, 100)
10 }
Console
Clear

Sketch B4.4 create a slider

Firstly, we give the slider a name, in this case the rather unimaginative name of `slider`. Secondly, we create the slider with three arguments:

A) Begin value (`0`)
B) End value (`255`)
C) Default value (`100`)

The range is between `0` and `255`. We will set the slider to `100` initially when you run the sketch. You can give it any value you want. Just use your mouse to slide it backwards and forwards.

```
let slider

function setup()
{
  createCanvas(400, 400)
  slider = createSlider(0, 255, 100)
}

function draw()
{
  background(220)
  circle(200, 100, 100)
}
```

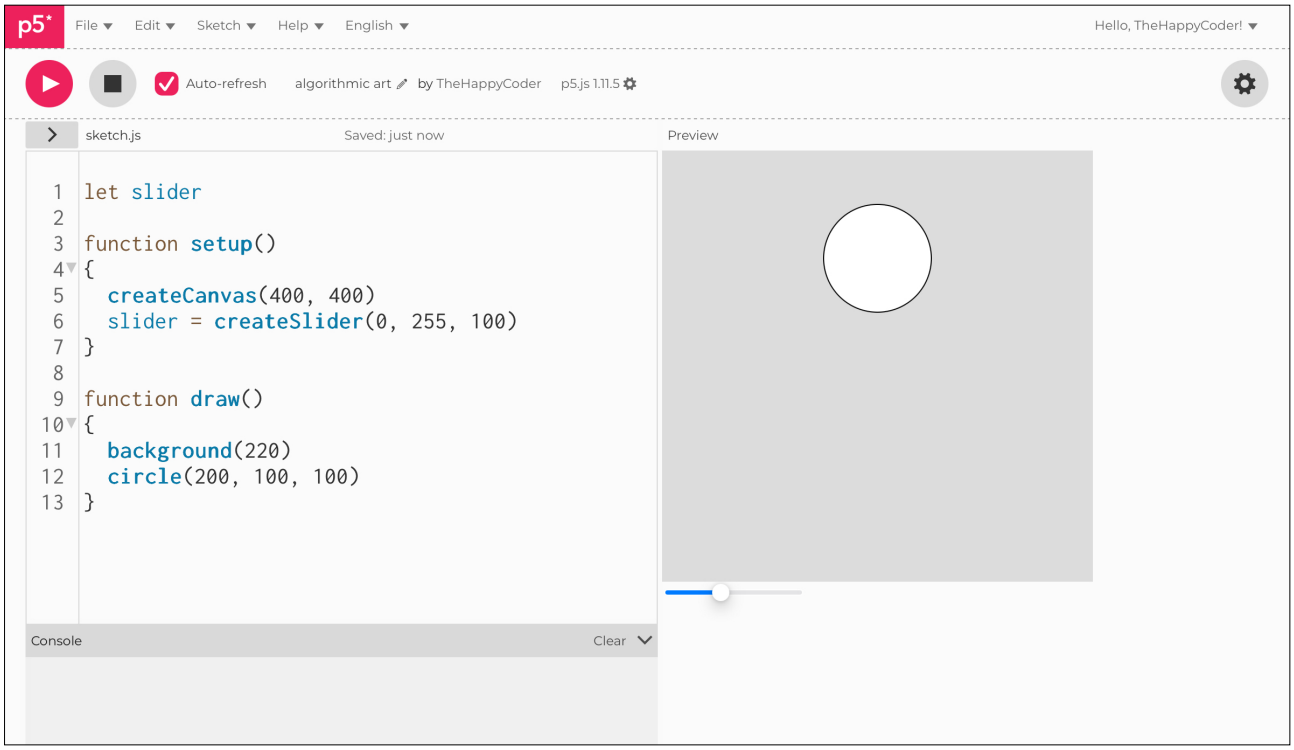
Notes

You will notice that the slider is off the canvas; this is because we have not given it any coordinates, so by default it puts it at the bottom, below the canvas. Also, the slider doesn't do anything except slide!

Code Explanation

`let slider`	The slider object.
`slider = createSlider(0, 255, 100)`	Create the slider object with the corresponding values.

Figure B4.4



Sketch B4.5 slider value

We can take the value of the slider using `slider.value()`, the `value()` function returns the value. We can fill the circle with that value. Now when you move the slider, it changes the colour of the circle.

```
let slider

function setup()
{
  createCanvas(400, 400)
  slider = createSlider(0, 255, 100)
}

function draw()
{
  background(220)
  fill(slider.value())
  circle(200, 100, 100)
}
```

Notes

You should see it change as you move the slider. We have put it straight into the `fill()` function.

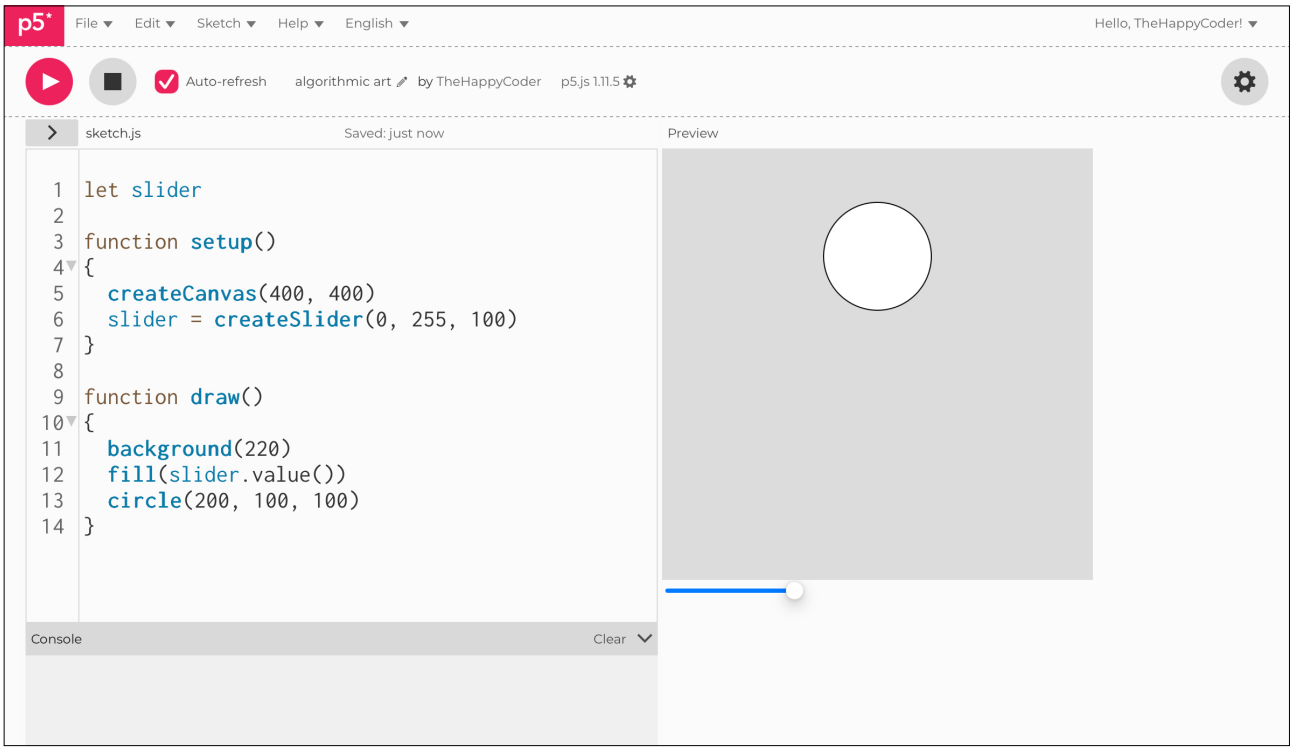
Challenges

1. Try different start, final, and initial values for the slider.
2. You could create a variable for the slider value.

Code Explanation

`fill(slider.value())`	Returns the value of the slider.

Figure B4.5



Sketch B4.6 position slider

We will now move it to the canvas; we simply give it the coordinates we want to display it at.

```
let slider

function setup()
{
  createCanvas(400, 400)
  slider = createSlider(0, 255, 100)
}

function draw()
{
  background(220)
  slider.position(50, 200)
  fill(slider.value())
  circle(200, 100, 100)
}
```

Notes

Slider now on the canvas.

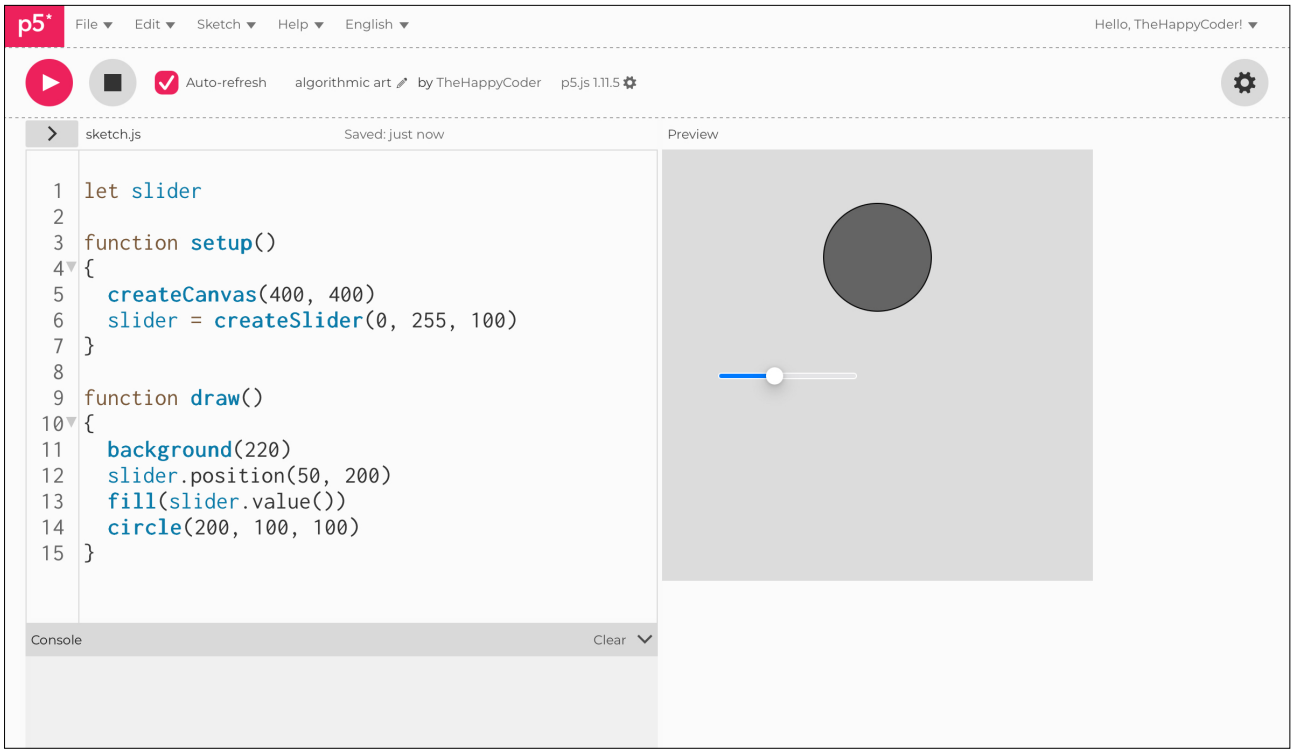
Challenge

Try different positions.

Code Explanation

`slider.position(50, 200)`	Positioned the slider at 50 from the left and 200 down from the top.

Figure B4.6



Sketch B4.7 showing the value

We can display the value of the slider.

```
let slider

function setup()
{
  createCanvas(400, 400)
  slider = createSlider(0, 255, 100)
}

function draw()
{
  background(220)
  slider.position(50, 200)
  fill(slider.value())
  circle(200, 100, 100)
  textSize(20)
  text(slider.value(), 200, 215)
}
```

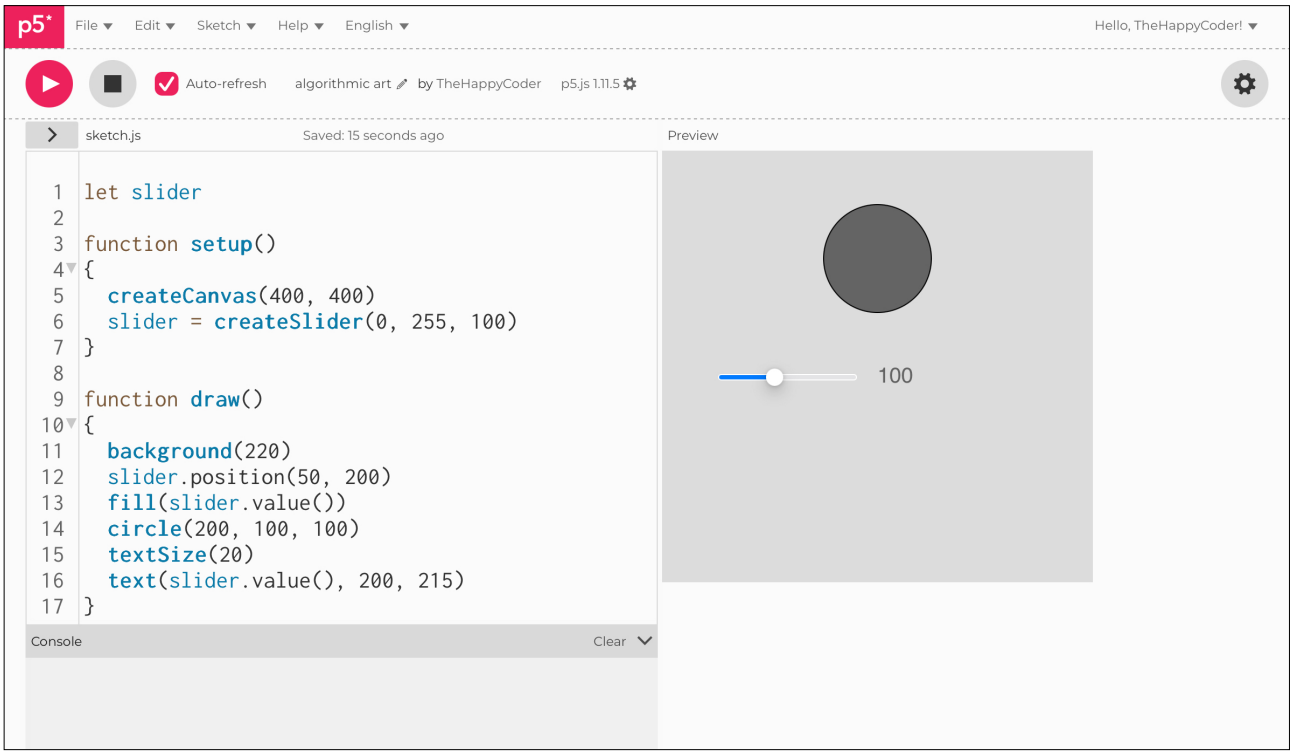
Notes

One issue, as you will see when you slide the slider, is that the text changes colour as well.

Challenge

Do you know how to stop that?

Figure B4.7



Sketch B4.8 text colour

Just to keep the text from changing colour.

```
let slider

function setup()
{
  createCanvas(400, 400)
  slider = createSlider(0, 255, 100)
}

function draw()
{
  background(220)
  slider.position(50, 200)
  fill(slider.value())
  circle(200, 100, 100)
  fill(0)
  textSize(20)
  text(slider.value(), 200, 215)
}
```

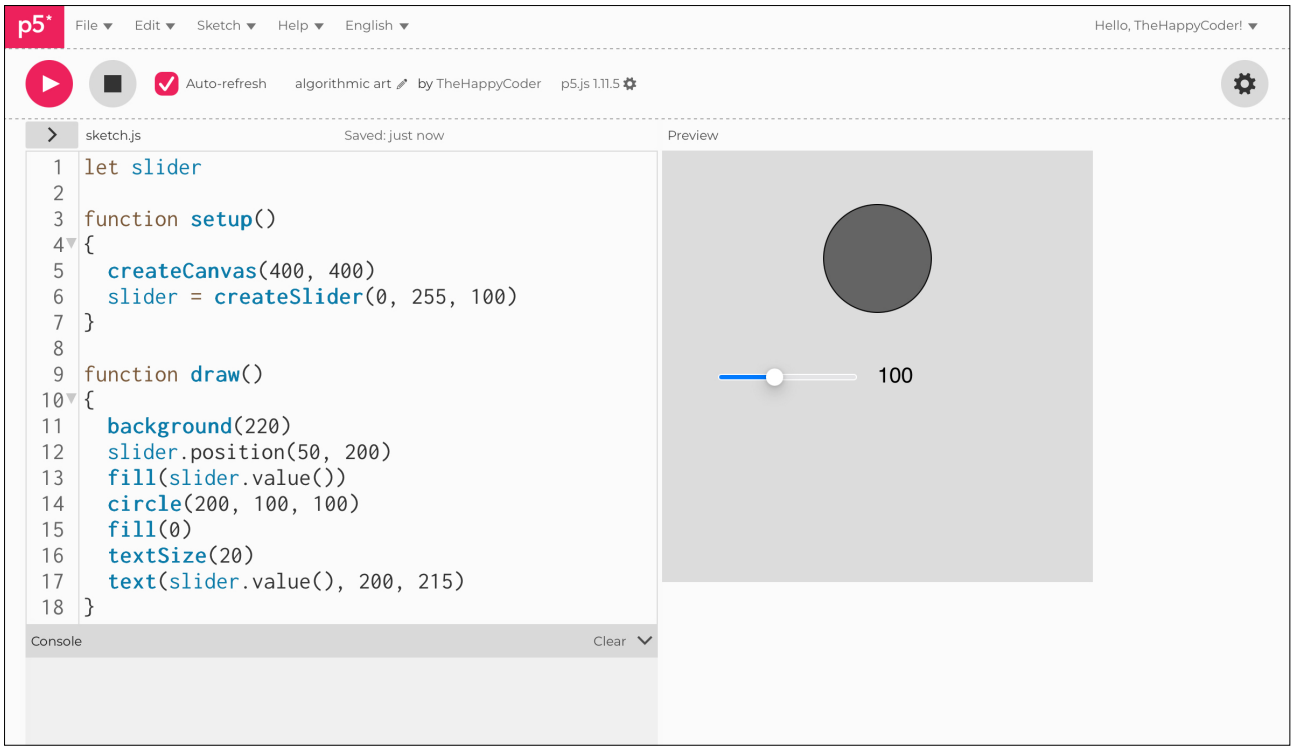
Notes

We have taken this bit nicely and slowly; next, we will look at red, green, and blue.

Challenge

Any ideas how to do red, green, and blue?

Figure B4.8



Sketch B4.9 red value

We need to change the name of the slider to `sliderRed()` everywhere, and also change the `fill()` to include the values of the green and blue (`0`).

```
let sliderRed

function setup()
{
  createCanvas(400, 400)
  sliderRed = createSlider(0, 255, 100)
}

function draw()
{
  background(220)
  sliderRed.position(50, 200)
  fill(sliderRed.value(), 0, 0)
  circle(200, 100, 100)
  fill(0)
  textSize(20)
  text(sliderRed.value(), 200, 215)
}
```

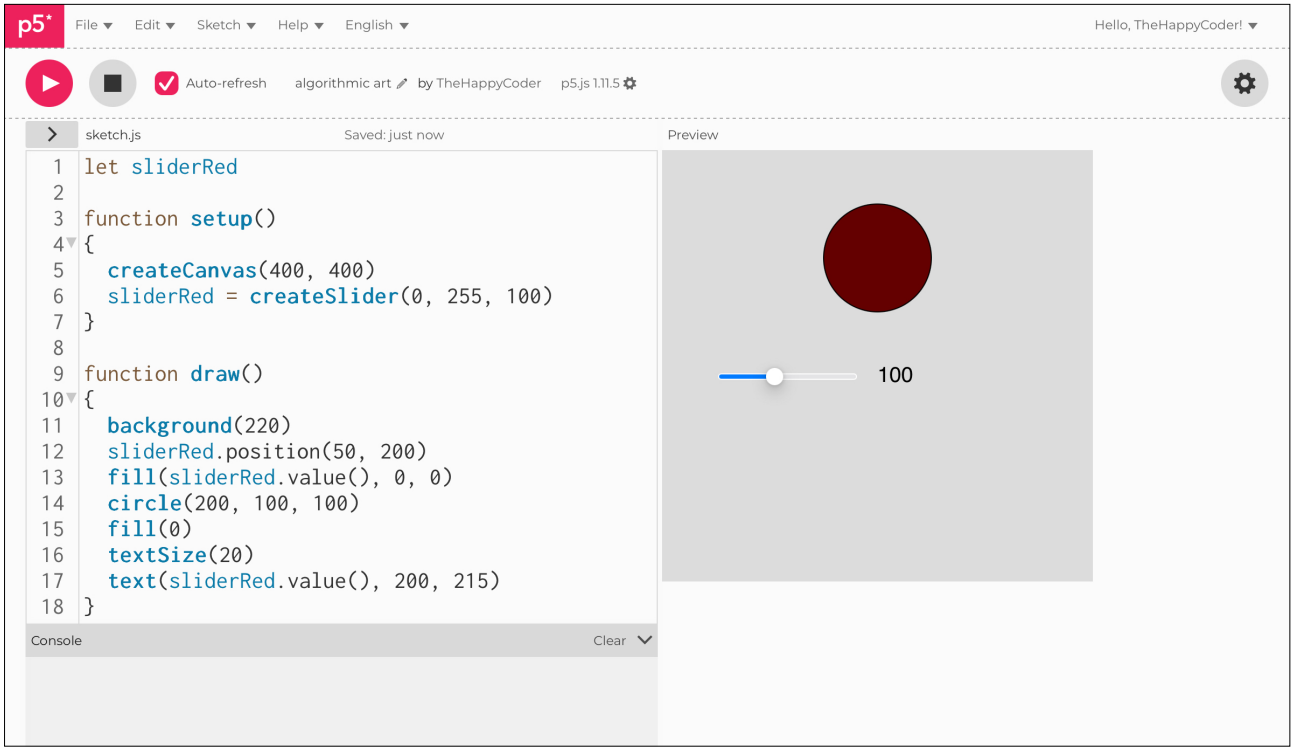
Notes

The circle will change from black to bright red.

Challenge

How about adding the green and the blue?

Figure B4.9



Sketch B4.10 full RGB

Adding the other colours.

```
let sliderRed
let sliderGreen
let sliderBlue

function setup()
{
  createCanvas(400, 400)
  sliderRed = createSlider(0, 255, 100)
  sliderGreen = createSlider(0, 255, 100)
  sliderBlue = createSlider(0, 255, 100)
}

function draw()
{
  background(220)
  sliderRed.position(50, 200)
  sliderGreen.position(50, 250)
  sliderBlue.position(50, 300)
  fill(sliderRed.value(), sliderGreen.value(), sliderBlue.value())
  circle(200, 100, 100)
  fill(0)
  textSize(20)
  text(sliderRed.value(), 200, 215)
  text(sliderGreen.value(), 200, 265)
  text(sliderBlue.value(), 200, 315)
}
```

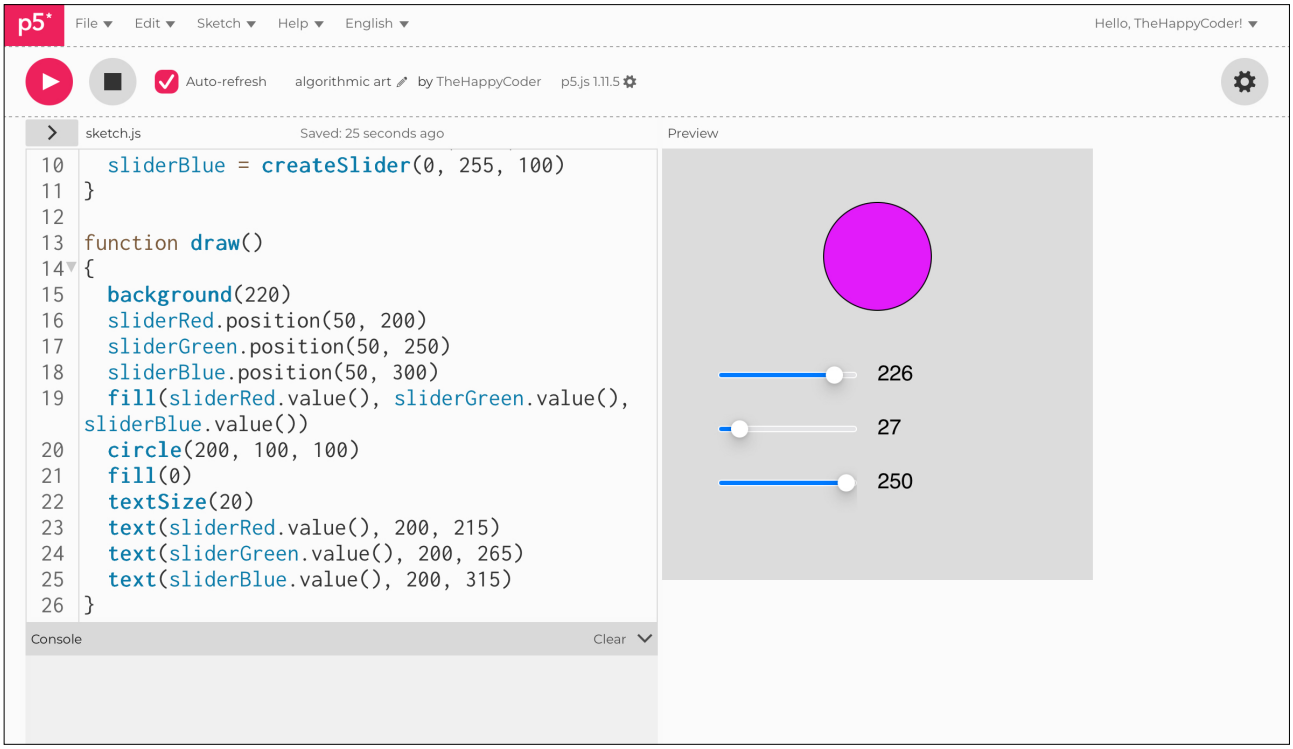
Notes

A lot of copying and pasting.

Challenge

Have it change the background instead.

Figure B4.10



The Joy of Coding Algorithmic Art

Module B Unit #5 HSB colours

Module B Unit #5: HSB colours

HSB is another colour format, where HSB stands for Hue, Saturation, and Brightness. We can only access this format using the colourMode() function. It creates pleasant colours but is less intuitive than RGB. The Hue has values from 0 to 360, Saturation has values from 0 to 100, and Brightness also has values from 0 to 100.

Key concepts:

Understanding HSB colour format
HSB values

Sketch B5.1 using HSB for the colour

HSB (or HSL) stands for Hue, Saturation, and Brightness (or Lightness).

Hue

The first argument is the hue, which is on a colour circle from 0 to 360, where 0 is red, 120 is green, and 240 is blue.

Saturation

The second argument is the saturation, which is a percentage from 0 to 100. It is the amount of colour, so 0 is white to 100 being full colour.

Brightness

The third argument is the level of brightness, which is from 0 to 100 also, where 0 is completely dark, and 100 fully bright.

! Start a new sketch and add in the highlighted lines of code. For this, we have to change the colorMode() to HSB. Here we get a green background and a blue circle.

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

function draw()
{
  background(0, 50, 100)
  fill(200, 50, 100)
  circle(100, 100, 100)
}
```

Notes

The default is RGB, so that is why we call the `colorMode()` for HSB. You can switch back by calling the `colorMode(RGB)`. HSB produces some very pleasing colours.

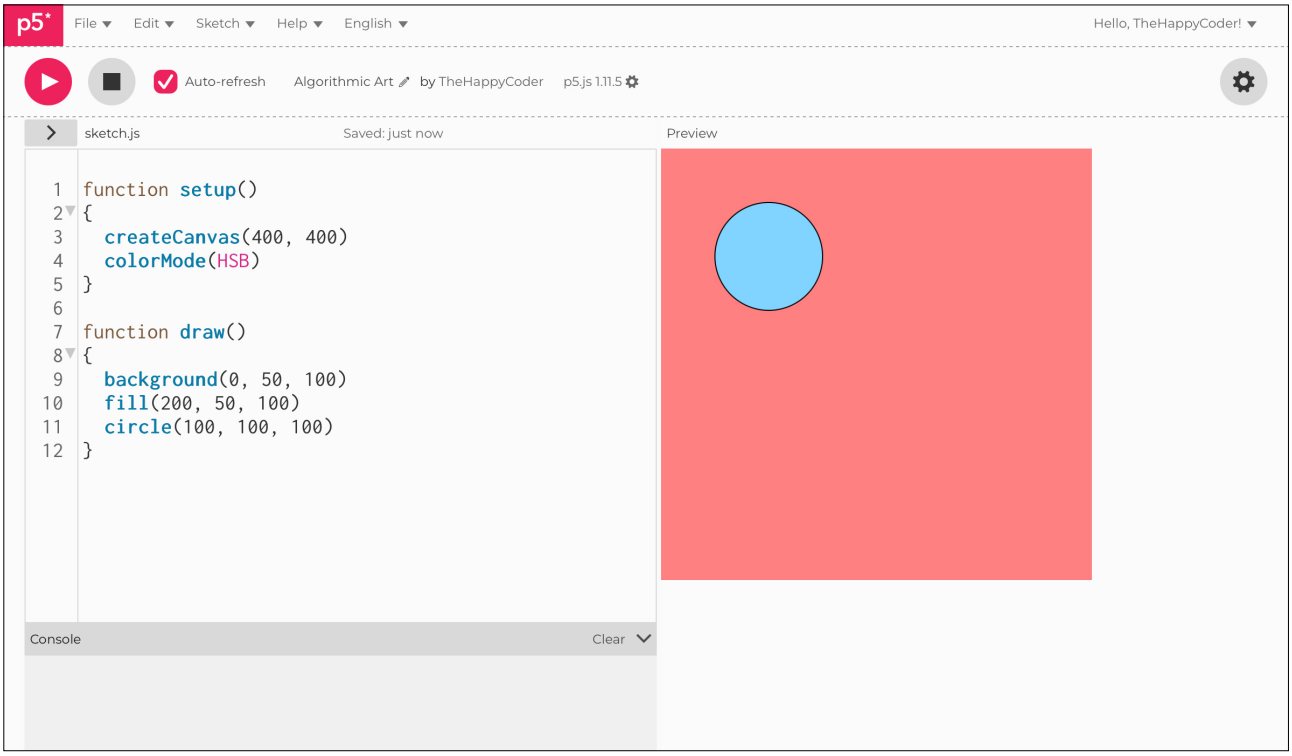
Challenge

Play with the numbers.

Code Explanation

`colorMode(HSB)`	Sets the mode for HSB values.
`background(0, 50, 100)`	A pale red background.
`fill(200, 50, 100)`	A pale blue circle.

Figure B5.1



Sketch B5.2 HSB colour chart

! A newish sketch

We iterate through the colours 10 steps at a time from 0 to 360. This is the first argument. Notice that we have a canvas of 360x400 for a change.

```
function setup()
{
  createCanvas(360, 400)
  colorMode(HSB)
  noStroke()
}

function draw()
{
  for (let i = 0; i < 360; i++)
  {
    fill(i, 100, 100)
    rect(i, 1, 1, 400)
  }
}
```

Notes

We start with red at 0 and end with red at 360. We have drawn 360 rectangles (thickness of 1) and coloured (filled) each one with a different colour. We needed noStroke() otherwise it would be just black! It gives us the

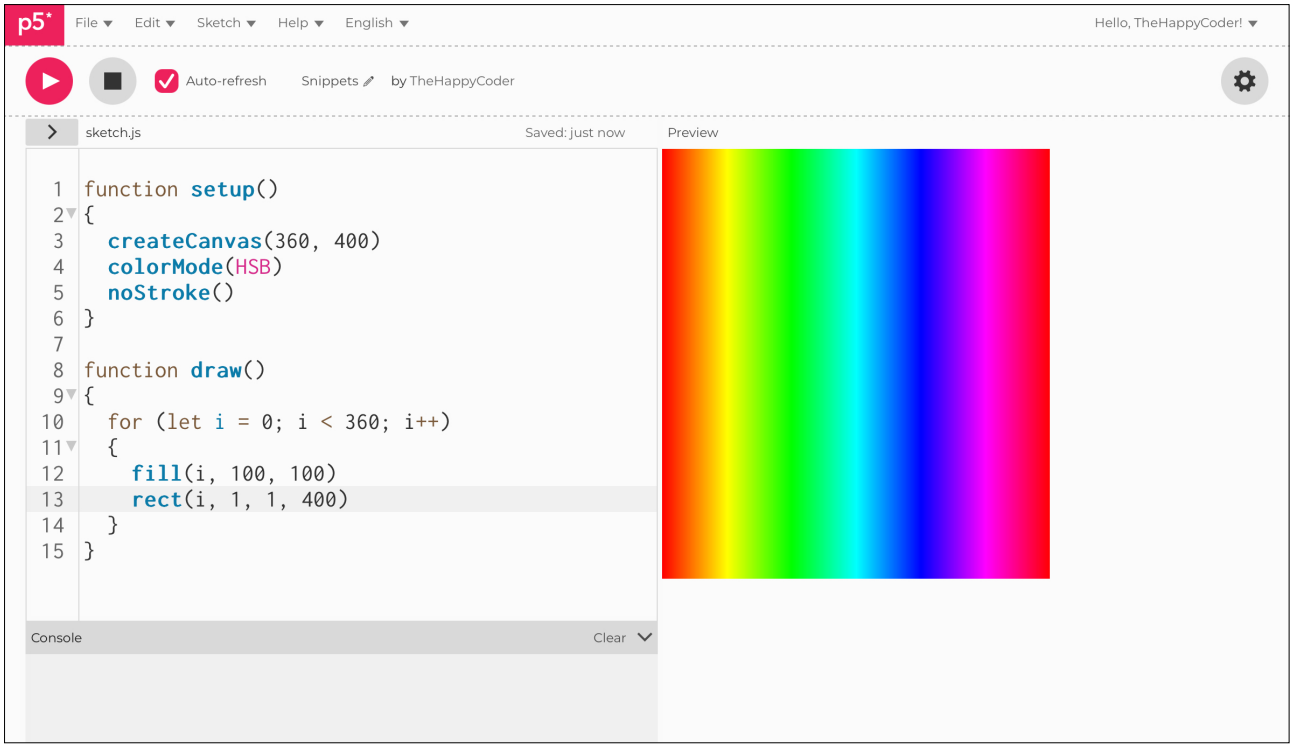
Challenge

You could try to create a circular colour palette.

Code Explanation

`for (let i = 0; i < 360; i++)`	Iterates from 0 to 360 one at a time.
`fill(i, 100, 100)`	Increases the hue value one at a time.
`rect(i, 1, 1, 400)`	Draws a rectangle, incrementing each rectangle one pixel at a time.

Figure B5.2



Sketch B5.3 circle of colour

! Another new sketch

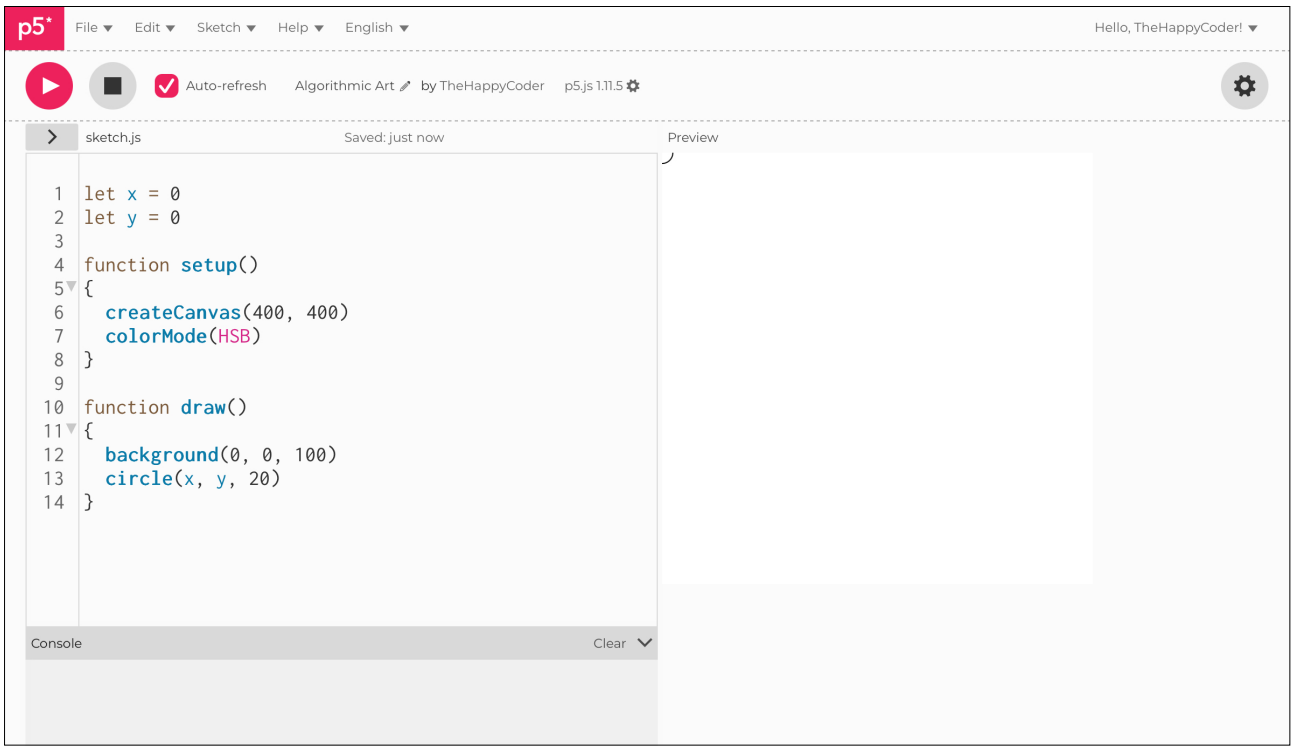
To get white, you put the saturation to zero.

```
let x = 0
let y = 0

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

function draw()
{
  background(0, 0, 100)
  circle(x, y, 20)
}
```

Notes

Notice how we have given the background a white colour and a circle hiding in the top left hand corner.

Figure B5.3

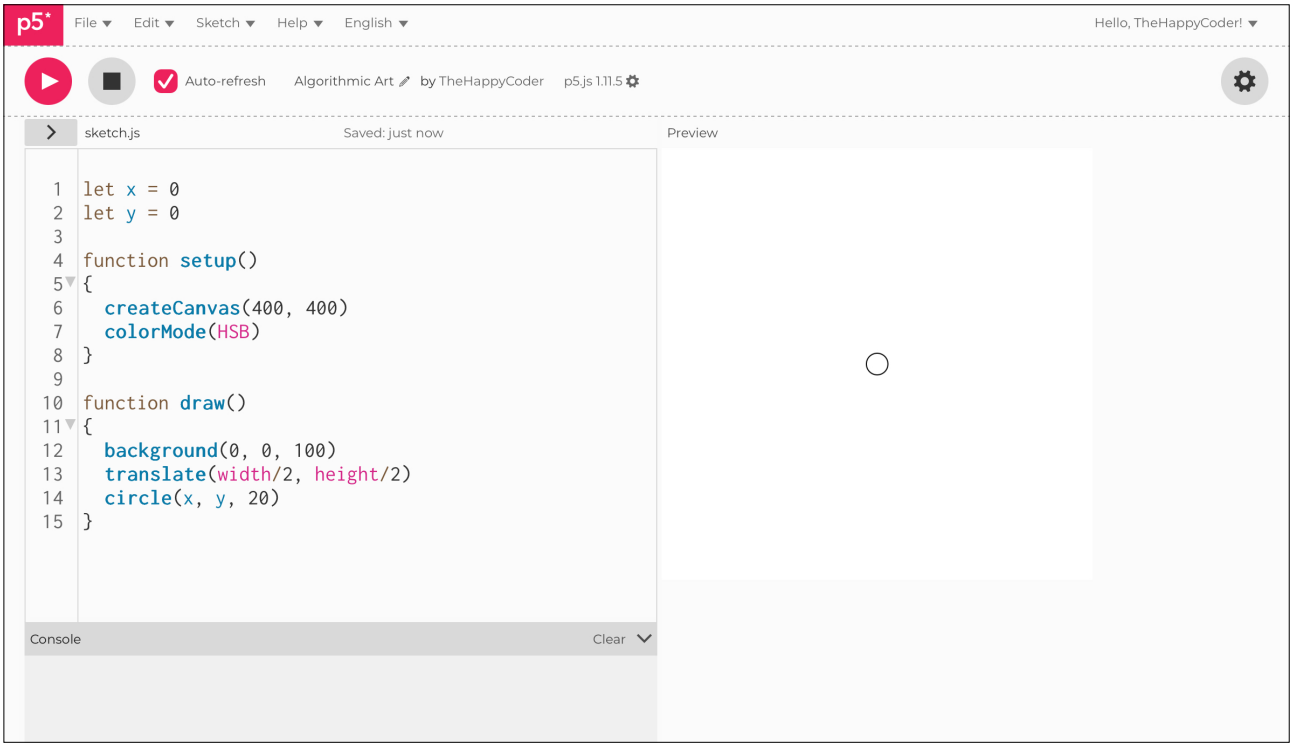


Sketch B5.4 centring the circle

It is hard to see, so we will translate the circle to the centre.

```
let x = 0
let y = 0

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

function draw()
{
  background(0, 0, 100)
  translate(width/2, height/2)
  circle(x, y, 20)
}
```

Figure B5.4


p5*
File
Edit
Sketch
Help
English
Hello, TheHappyCoder!
Auto-refresh
Algorithmic Art
by TheHappyCoder
p5.js 1.11.5
sketch.js
Saved: just now
Preview
let x = 0
let y = 0
function setup()
{
createCanvas(400, 400)
colorMode(HSB)
}
function draw()
{
background(0, 0, 100)
translate(width/2, height/2)
circle(x, y, 20)
}
Console
Clear

Sketch B5.5 angle of attack

We want to draw a series of small circles around a larger circle, spaced out evenly. We will use a `for()` loop to increment `8°` from `0°` to `360°`. We will do this in `degrees` using `angleMode(DEGREES)`.

```
let x = 0
let y = 0

function setup()
{
  createCanvas(400, 400)
  colorMode(HSB)
  angleMode(DEGREES)
}

function draw()
{
  background(0, 0, 100)
  translate(width/2, height/2)
  for (let angle = 0; angle < 360; angle += 8)
  {
    fill(angle, 100, 100)
    circle(x, y, 20)
  }
}
```

Notes

Notice all this seems to do is fill the circle red; this is because the `360°` (and `0°`) value of Hue is red. What happens is that it spins through all the colours very fast in the `for()` loop.

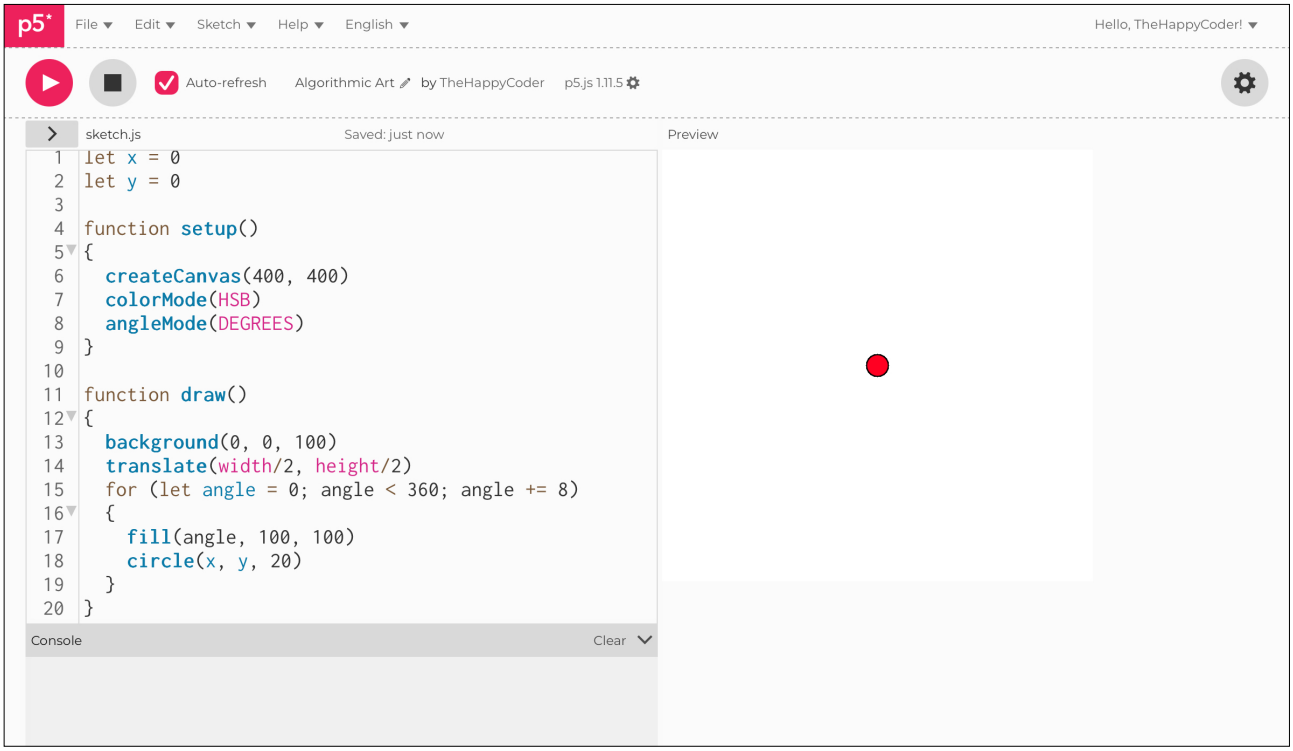
Challenge

Change the `360` to `180`.

Code Explanation

`for (let angle = 0; angle < 360; angle += 8)`	Spin through all the angles from 0° to 360° by 8° at a time.
`fill(angle, 100, 100)`	Fill the hue value from the angle.

Figure B5.5



Sketch B5.6 sine and cosine

We use the principles discussed a few units back, where the coordinates on a circle can be described using sine and cosine for x and y about the centre of the circle.

```
let x = 0
let y = 0
let radius = 180

function setup()
{
  createCanvas(400, 400)
  colorMode(HSB)
  angleMode(DEGREES)
}

function draw()
{
  background(0, 0, 100)
  translate(width/2, height/2)
  for (let angle = 0; angle < 360; angle += 8)
  {
    fill(angle, 100, 100)
    x = radius * sin(angle)
    y = radius * cos(angle)
    circle(x, y, 20)
  }
}
```

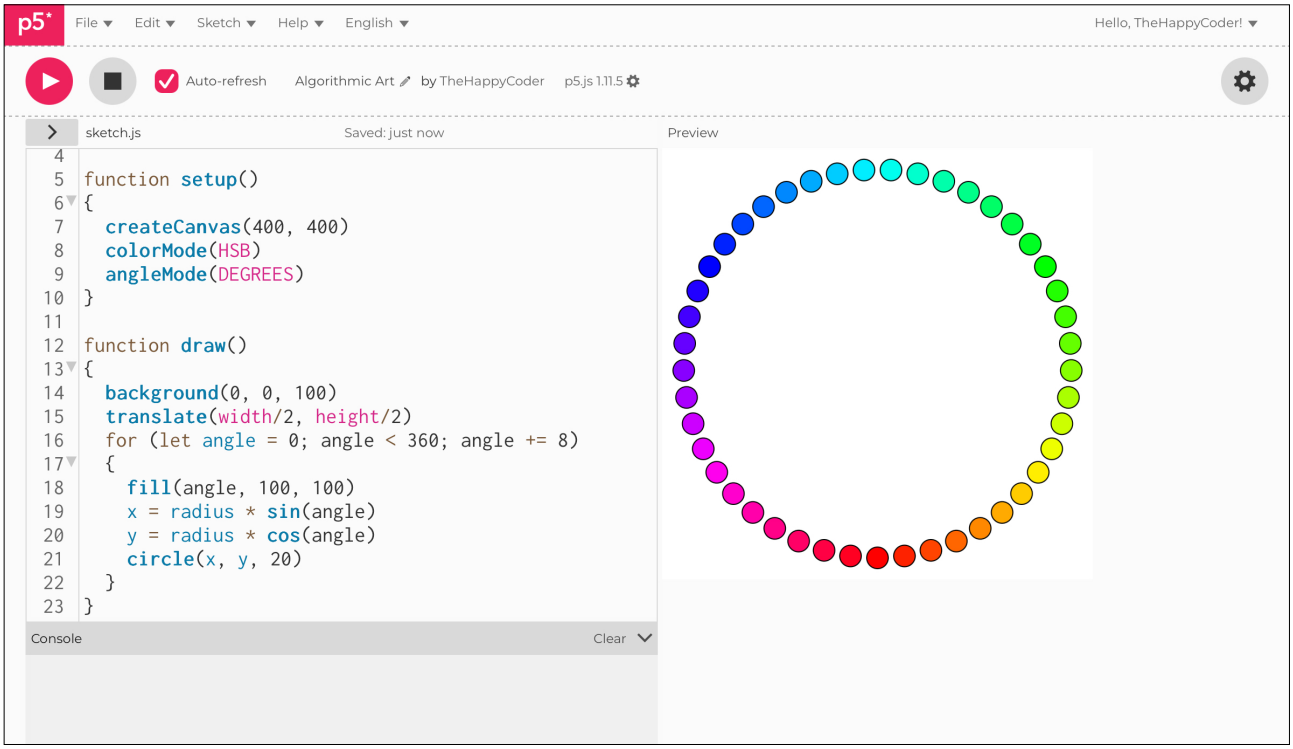
Notes

You now have the 360-degree series of circles.

Challenge

Try different radii and spacings of the circles.

Figure B5.6



Sketch B5.7 one final thing

Let's draw a nice box round the canvas so we can see where the canvas starts. Even for something as simple as this, you need to think about translation and the impact on coordinates.

```
let x = 0
let y = 0
let radius = 180

function setup()
{
  createCanvas(400, 400)
  colorMode(HSB)
  angleMode(DEGREES)
}

function draw()
{
  background(0, 0, 100)
  translate(width/2, height/2)
  for (let angle = 0; angle < 360; angle += 8)
  {
    fill(angle, 100, 100)
    x = radius * sin(angle)
    y = radius * cos(angle)
    circle(x, y, 20)
  }
  noFill()
  rect(-width/2, -height/2, width, height)
}
```

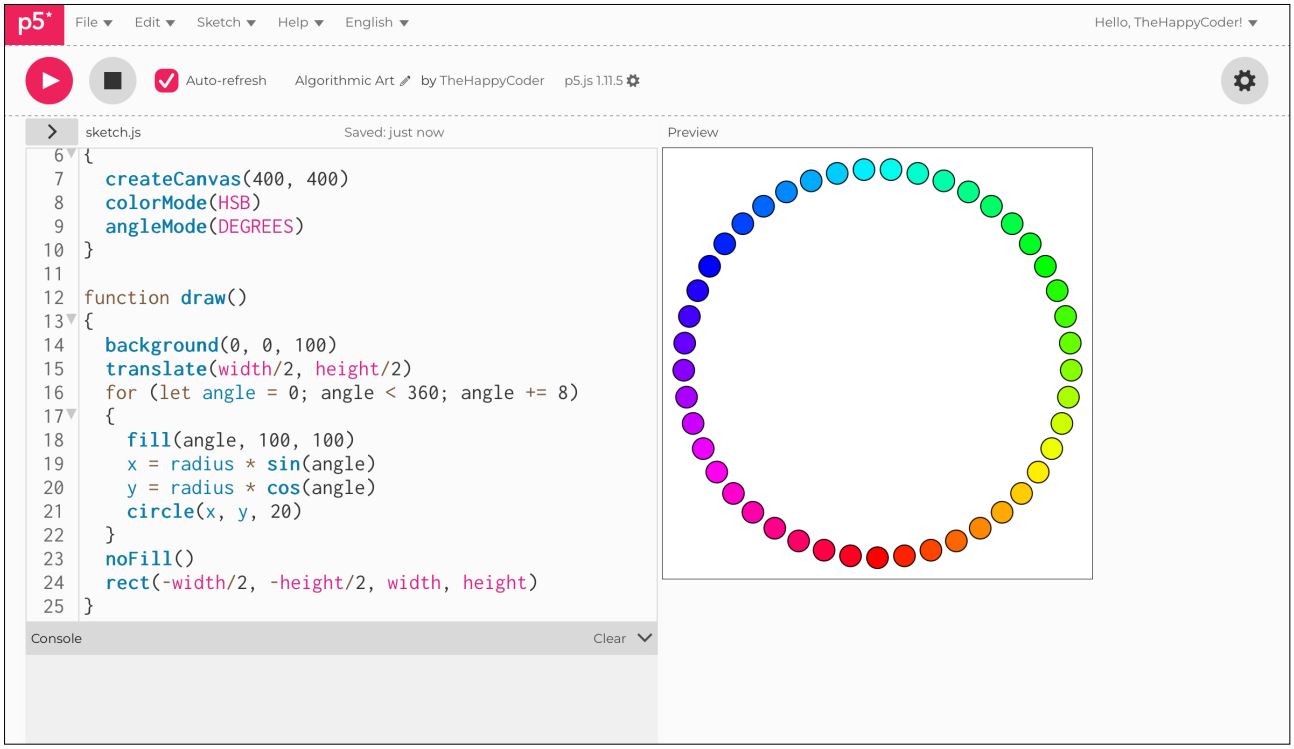
Notes

Looks a lot nicer.

Challenge

If you put it inside the loop, what would you need to change?

Figure B5.7



Sketch B5.8 100 rectangles

! Starting a new sketch

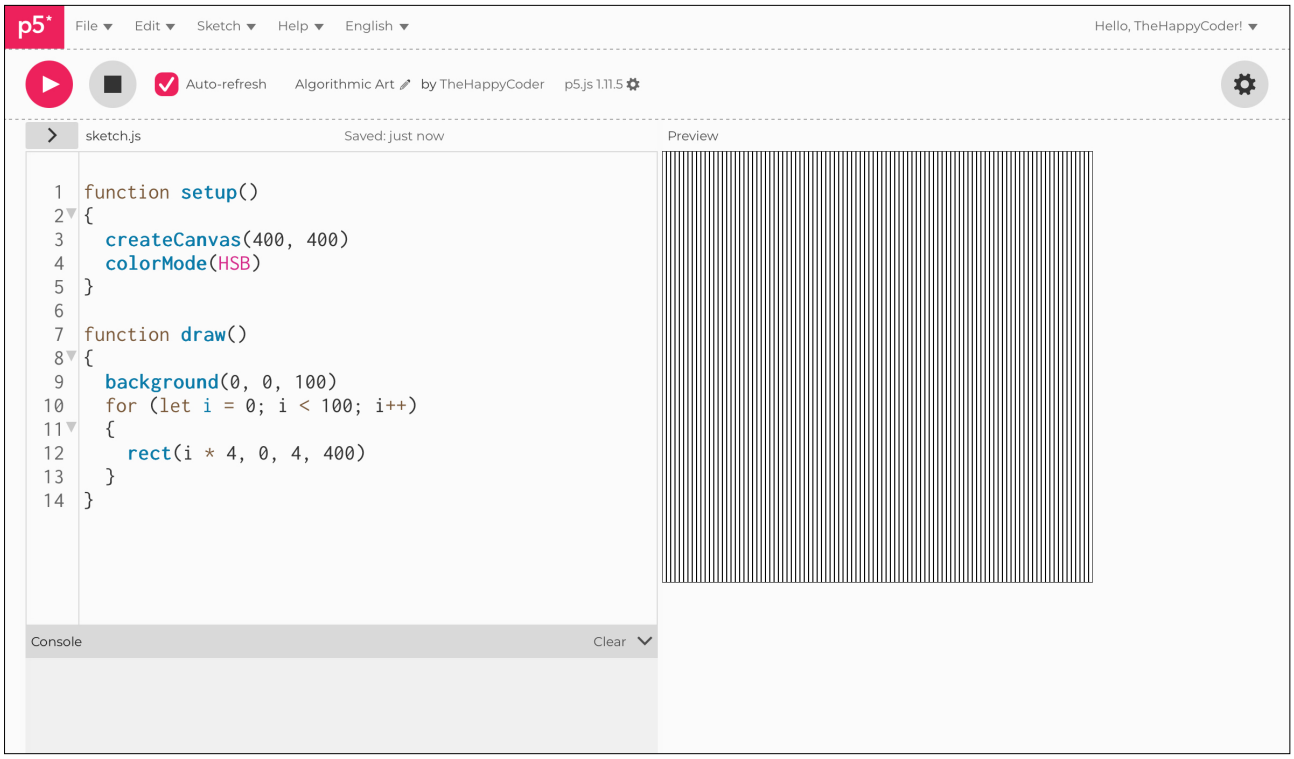
Here we loop through 100 rectangles, with each rectangle being 4 pixels wide and 400 pixels long; hence, we multiply the i variable by 4.

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

function draw()
{
  background(0, 0, 100)
  for (let i = 0; i < 100; i++)
  {
    rect(i * 4, 0, 4, 400)
  }
}
```

Notes

We just get a bunch of rectangles from left to right.

Figure B5.8



Sketch B5.9 HSB saturation

Then we fill each rectangle with a hue of 0, which is red; the saturation (second argument) we increase by 1 on each iteration of the loop, keeping the brightness (third argument) at maximum (100).

```
function setup()
{
  createCanvas(400, 400)
  colorMode(HSB)
  noStroke()
}

function draw()
{
  background(0, 0, 100)
  for (let i = 0; i < 100; i++)
  {
    fill(0, i, 100)
    rect(i * 4, 0, 4, 400)
  }
}
```

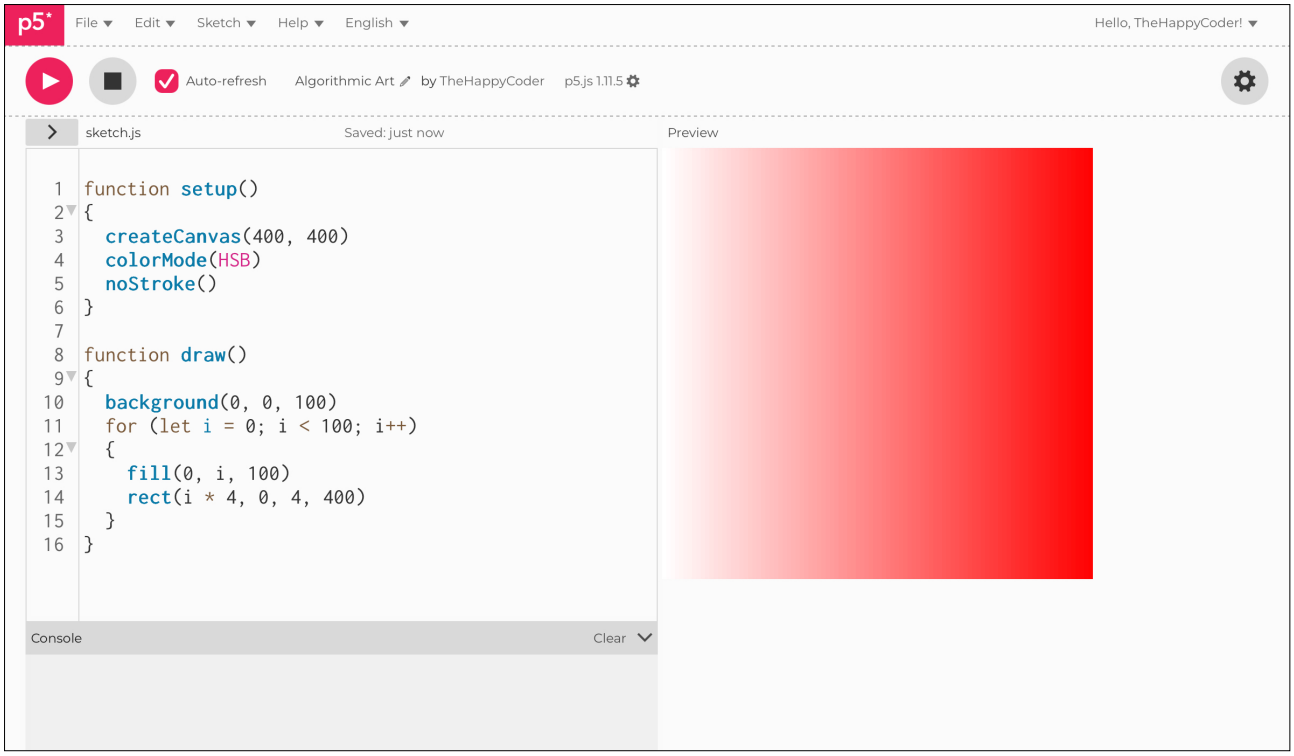
Notes

A saturation of 0 is no saturation (white), and 100 is the maximum; then everything in between.

Challenge

Do this with other hues.

Figure B5.9



Sketch B5.10 HSB brightness

Now for the brightness, this is the third argument: we change the `fill()` function, giving it `0` hue, `100` saturation, and an incrementing `i` variable for the brightness. This means that it starts off with `zero` brightness and increments to `full` brightness (100%).

```
function setup()
{
  createCanvas(400, 400)
  colorMode(HSB)
  noStroke()
}

function draw()
{
  background(0, 0, 100)
  for (let i = 0; i < 100; i++)
  {
    fill(0, 100, i)
    rect(i * 4, 0, 4, 400)
  }
}
```

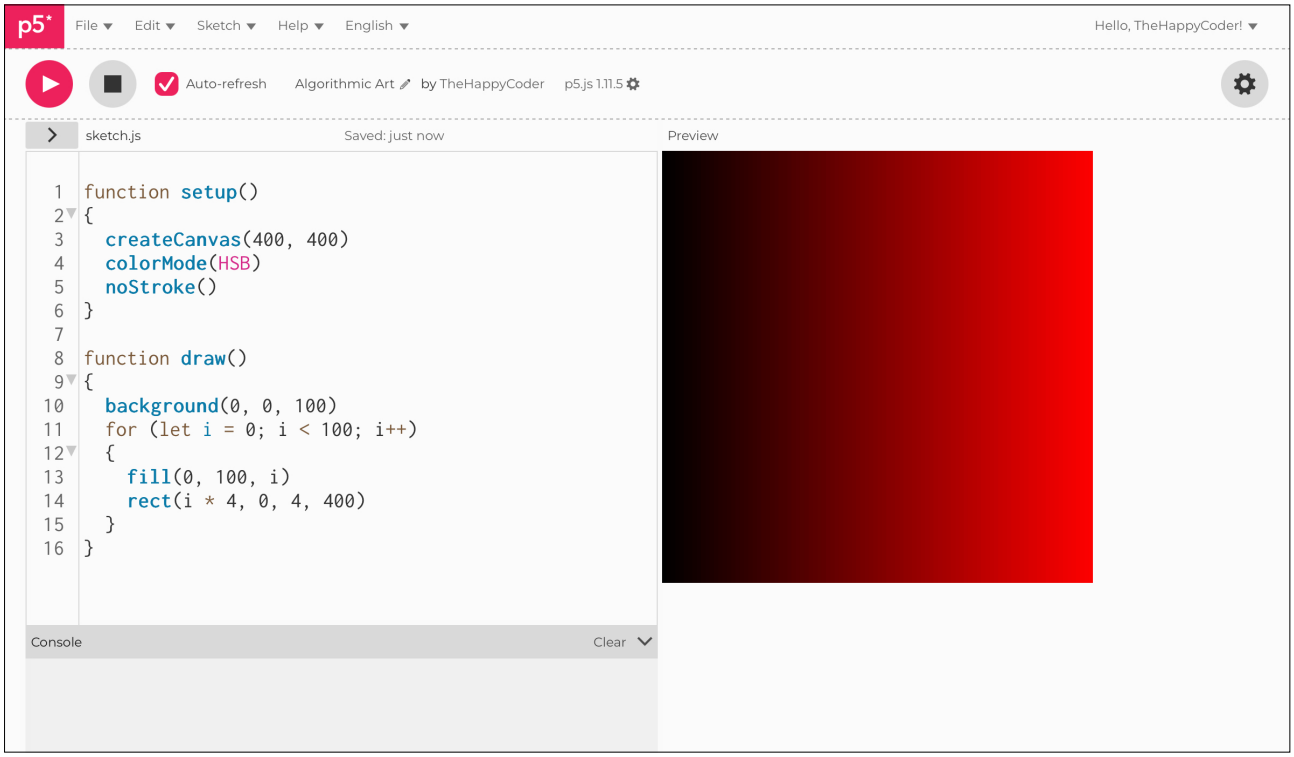
Notes

Fairly obvious results.

Challenges

1. Create a palette of colours that you might want to use regularly.
2. Use sliders for Hue, saturation and Brightness.

Figure B5.10



The Joy of Coding Algorithmic Art Module B Unit #6 colour charts and pickers

Module B Unit #6: colour charts and pickers

In this brief unit, we will look at two other colouring systems that use names and hexadecimal values. The hexadecimal system is quite common in coding circles. They are available across all browsers, and that is why they are often used for websites, etc.

Also, we will look at something you might find useful or interesting called lerpColour(). This interpolates between two colours, however identified, and calculates all the values in between in increments. So, if you want to see the colour change from, say, red to orange in increments.

Finally, I have included a comprehensive list of all the colour names, their hex and RGB corresponding values. A good resource if nothing else.

Key concepts:

Colour charts
Interpolating colours
Hex colour values
Colour picker

Sketch B6.1 using a name for the colour

We can, if we wish, just name the colours. There is a wide range of colours, too numerous to list them here (see end of unit for full list). The name must be in speech marks.

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

function draw()
{
  background('orange')
  fill('green')
  circle(100, 100, 100)
}
```

Notes

You will notice that it isn't the same exact colour. Because you put the name of the colour in speech marks, it gives you a square indicator to the colour. There are quite a few colour names. If you use `lightgreen` with no gap, you get light green. Some colours will take the word dark, e.g. `darkred`. Other names are ones like `teal`, `magenta`, and so on. You can use single or double quotes. It is useful if you just want a simple colour rather than trying to remember the values, especially if you want more than three colours.

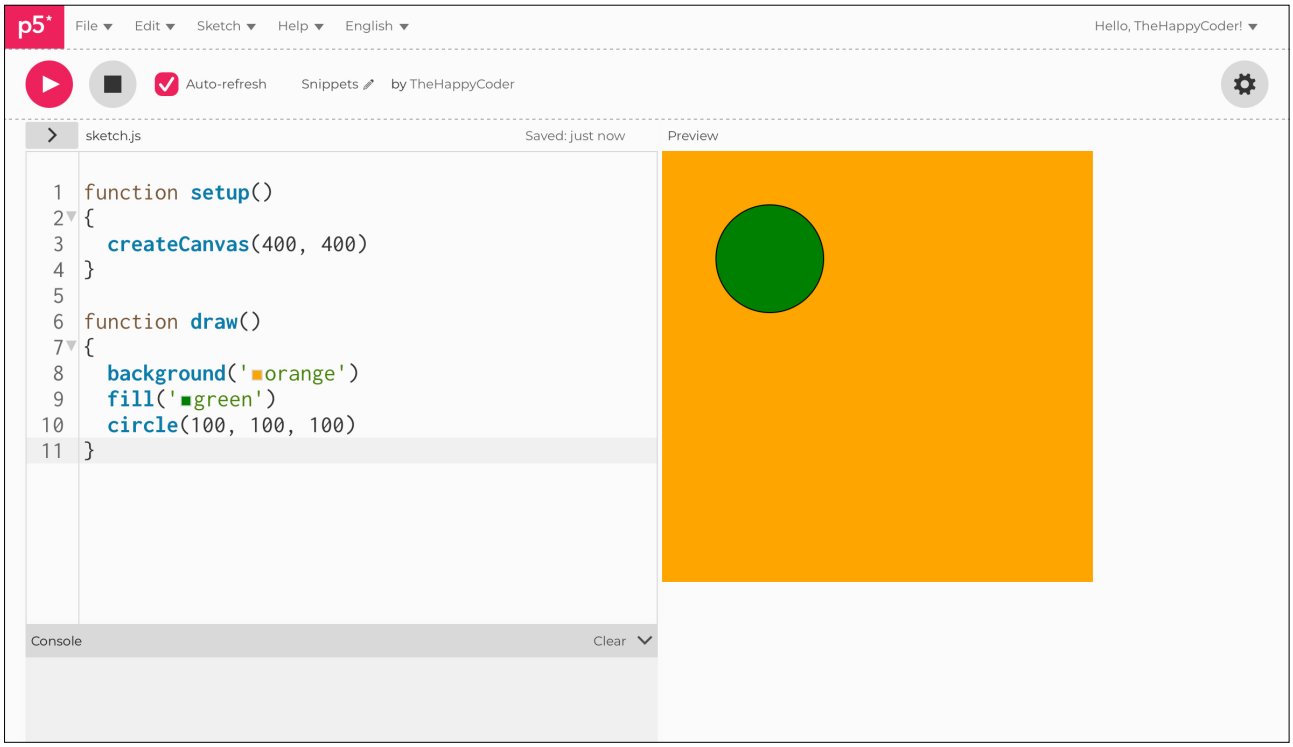
Challenges

1. Just experiment.
2. Search online for JavaScript colour names (or see the charts at the end of this unit).

Code Explanation

`background('orange')`	Gives you an orange background.
`fill('green')`	Fills the green circle.

Figure B6.1



Sketch B6.2 using a hex value for the colour

Hex, this is one colour format often used in website design, and it is a hexadecimal number (base 16) that starts with a # symbol. The hex number indicates a particular colour; you will notice that it uses letters as well as numbers. To explore the hex values, I will include a colour picker so that you can copy and paste the colour of your choice using the hex values.

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

function draw()
{
  background('#0000FF')
  fill('#FF0000')
  circle(100, 100, 100)
}
```

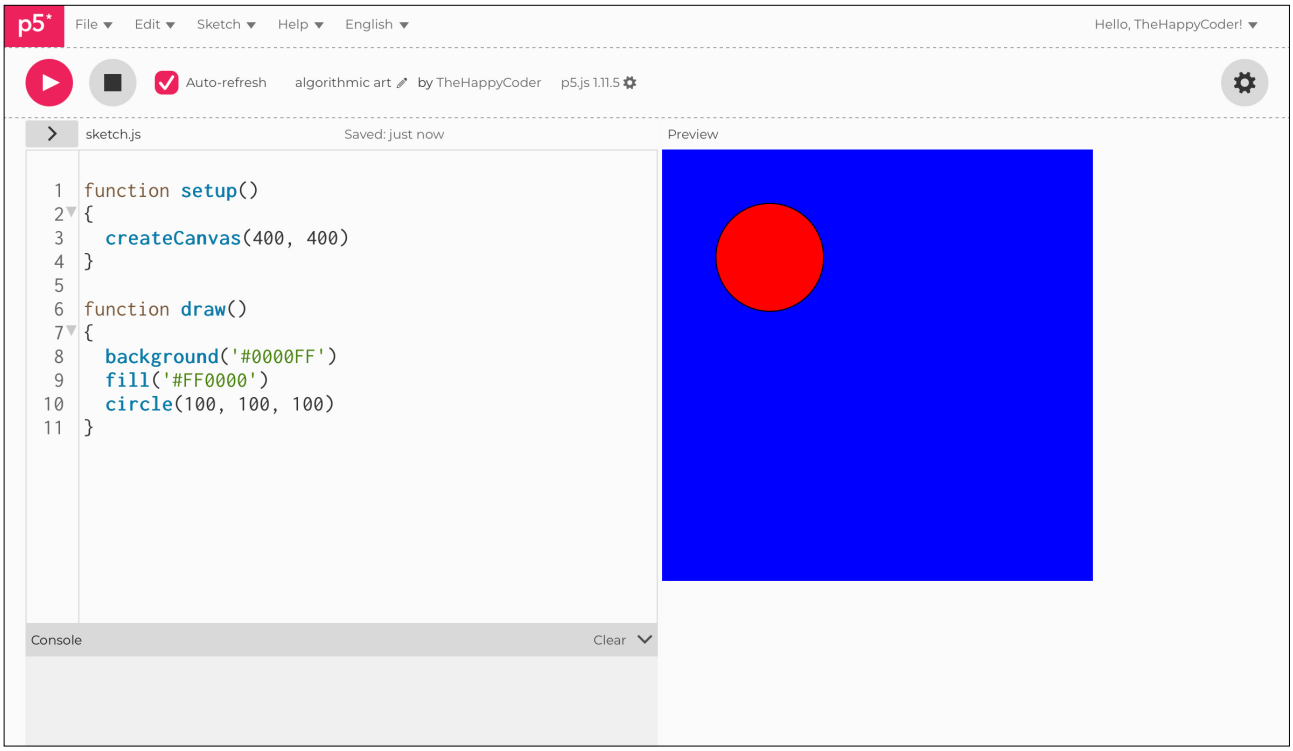
Notes

Here are some basic colours; notice that when you code it, you get the colour icon as well. See the colour charts at the end of this unit for more values.

Code Explanation

#FF0000	Red.
#00FF00	Green.
#0000FF	Blue.
#000000	Black.
#FFFFFF	White.

Figure B6.2



Sketch B6.3 bench of ten

! Jumping in with a new sketch

We have drawn ten circles using a `for()` loop, evenly spaced out.

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

function draw()
{
  background(220)
  for (let i = 0; i < 10; i++)
  {
    circle(20 + (i * 40), height/2, 25)
  }
}
```

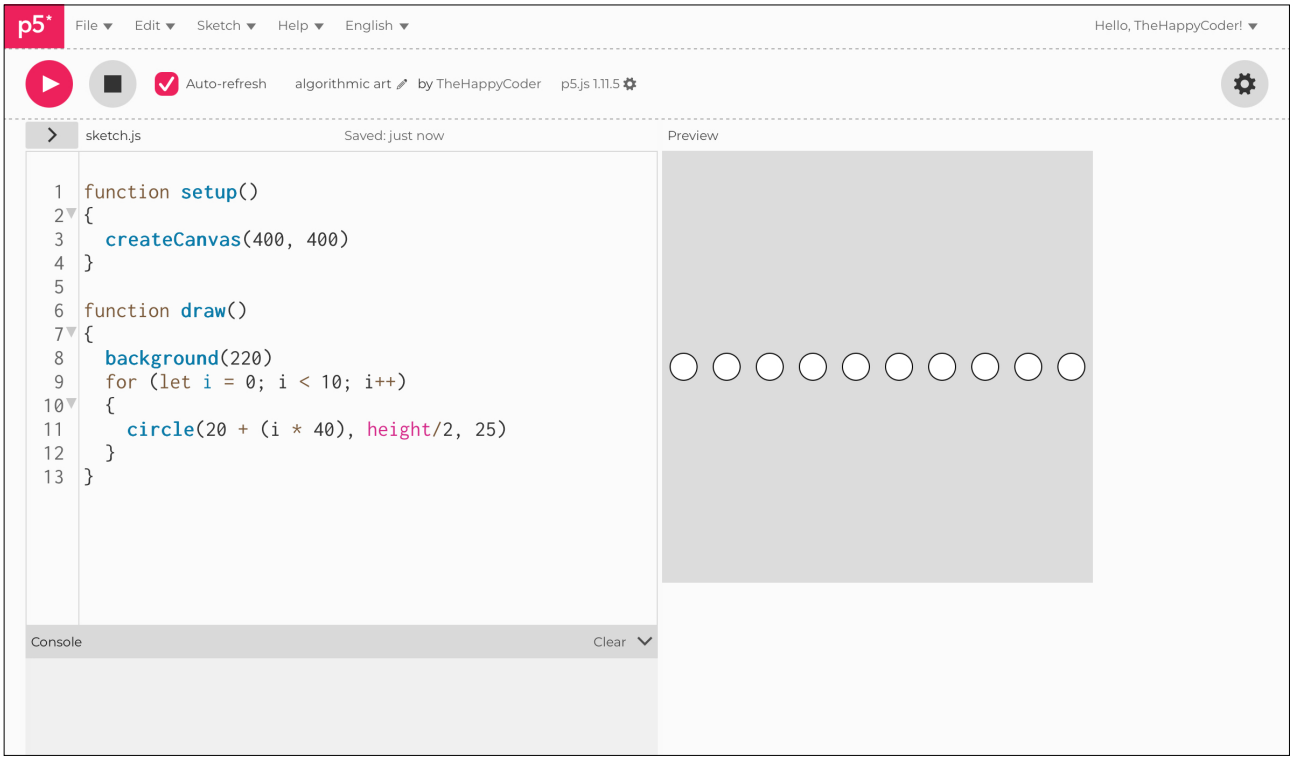
Notes

We multiply by 40 to get the spacing.

Challenge

You can have more circles, smaller and closer together if you wish for this exercise.

Figure B6.3



Sketch B6.4 new colour variable

We add a new variable called newColour and give it the colour orange.

```
let newColour = 'orange'

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

function draw()
{
  background(220)
  for (let i = 0; i < 10; i++)
  {
    fill(newColour)
    circle(20 + (i * 40), height/2, 25)
  }
}
```

Notes

A nice row of orange circles. We are using a variable for a good reason, you will see.

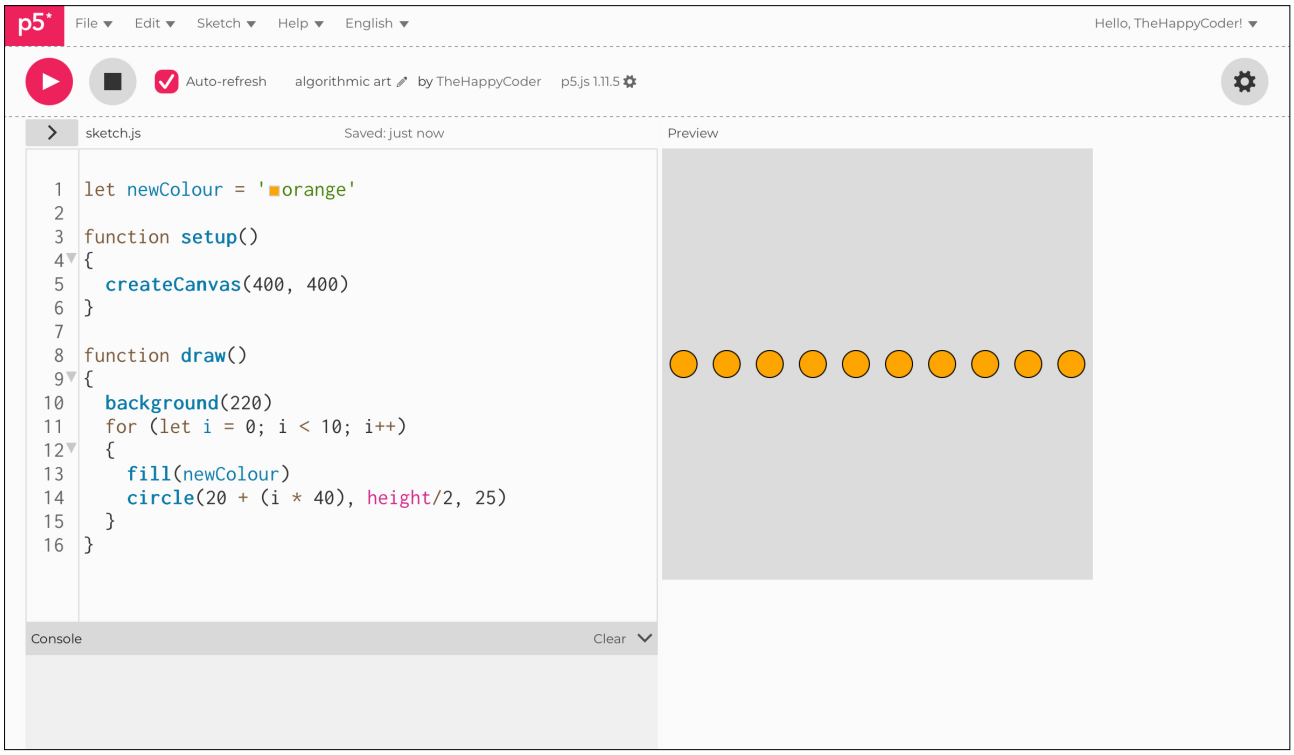
Challenge

You can choose a different colour if you wish.

Code Explanation

`let newColour = 'orange'`	The variable is initiated and defined by being given the colour string 'orange'.
`fill(newColour)`	Fill the circles with 'orange'.
`circle(20 + (i * 40), height/2, 25)`	Circle is spaced out every 40 pixels and starts at 20 from the left, evenly spaced.

Figure B6.4



Sketch B6.5 the lerp effect

We can interpolate between two colours with the `lerpColor()` function, which takes three arguments:
A) The `first` colour (lightyellow)
B) The `last` colour (darkred)
C) The `incremental` change (increment)

Lerp is short for interpolation.

! We add the `noLoop()` otherwise we just get all the circles filled with the final colour.

```
let newColour = 'orange'
let increment = 0

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

function draw()
{
  background(220)
  for (let i = 0; i < 10; i++)
  {
    newColour = lerpColor(color('lightyellow'), color('darkred'), increment)
    fill(newColour)
    circle(20 + (i * 40), height/2, 25)
    increment += 0.1
  }
}
```

Notes

This is all very hard-coded. We can work out the increments, that they increase by `0.1` because it has to be between `0` and `1`, and `0.1` is a `tenth` of `1`, we have `10` circles. The next sketch will show a slightly better way with more circles.

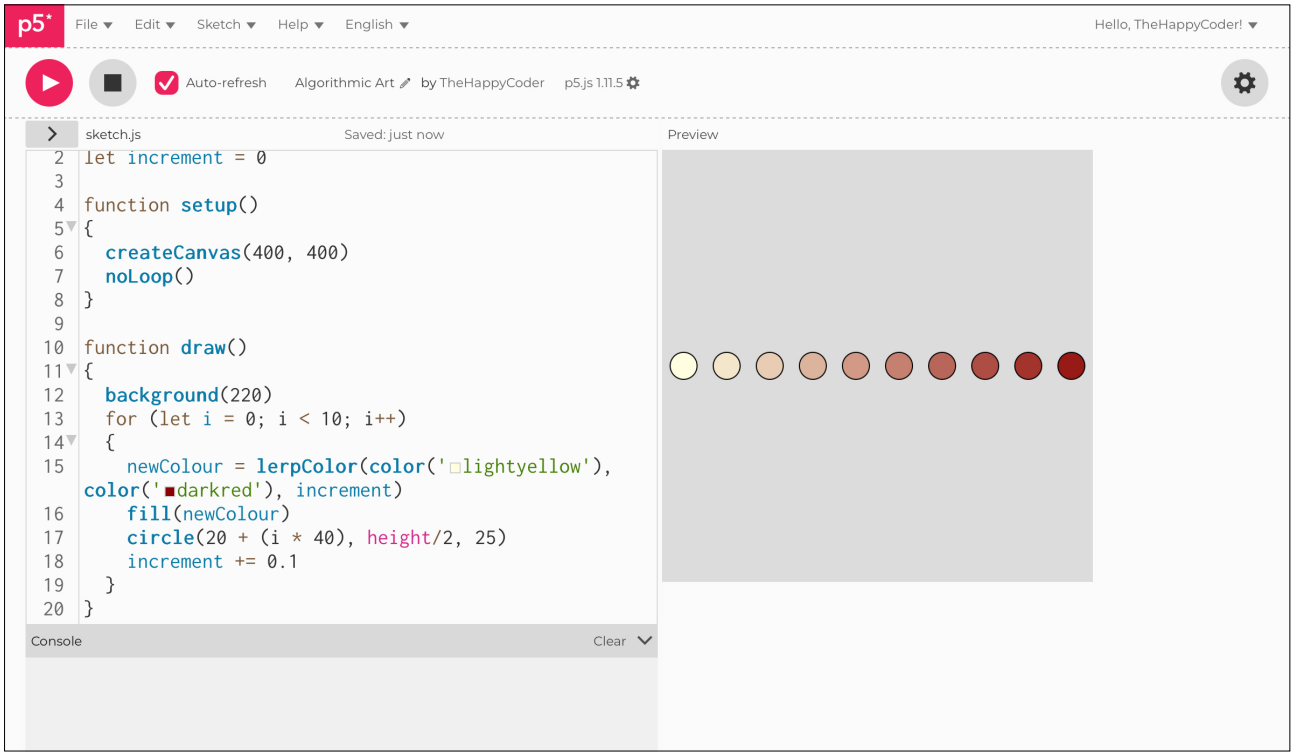
Challenge

Try combinations of different colours.

Code Explanation

`lerpColor(color('lightyellow'), color('darkred'), increment)`	The new colour is something between the 'lightyellow' and the 'darkred' depending on the increment.

Figure B6.5



Sketch B6.6 refining lerp

Using a variable (num) to calculate the incremental steps based on the number of circles. Changing from lightyellow to yellow and from darkred to purple.

```
let newColour = 'orange'
let increment = 0
let num = 10

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

function draw()
{
  background(220)
  for (let i = 0; i < num; i++)
  {
    newColour = lerpColor(color('yellow'), color('purple'), increment)
    fill(newColour)
    circle(20 + (i * 40), height/2, 25)
    increment += 1/num
  }
}
```

Notes

Changing the colours as well.

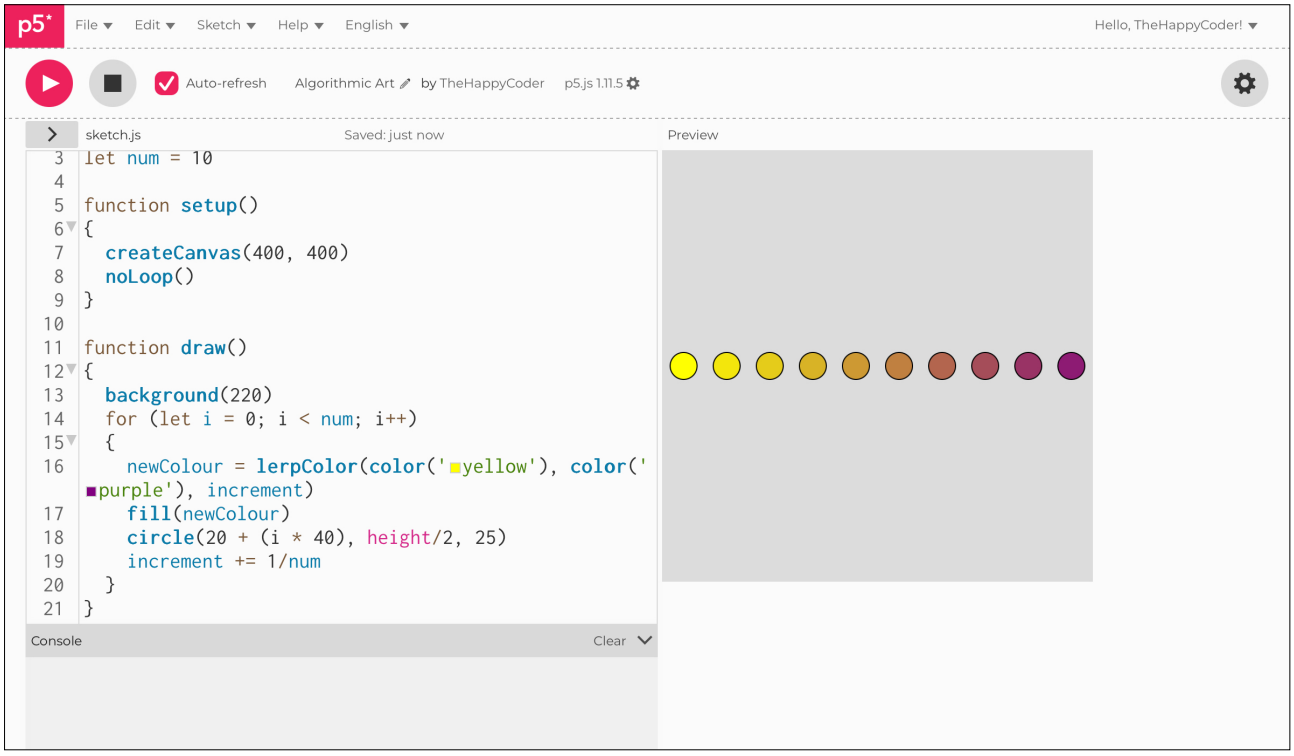
Challenge

Have 20 or more circles (or other shapes) to see the more gradual change in the migration or interpolation.

Code Explanation

`for (let i = 0; i < num; i++)`	Uses this variable to determine how many circles to draw.
`increment += 1/num`	Gives you the fraction for interpolation increment.

Figure B6.6



Sketch B6.7 the colour ‘c’

! Starting new sketch

Creating a variable `c` to hold any new colour. We will be using all three colour modes in this example.

```
let c = 220

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

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

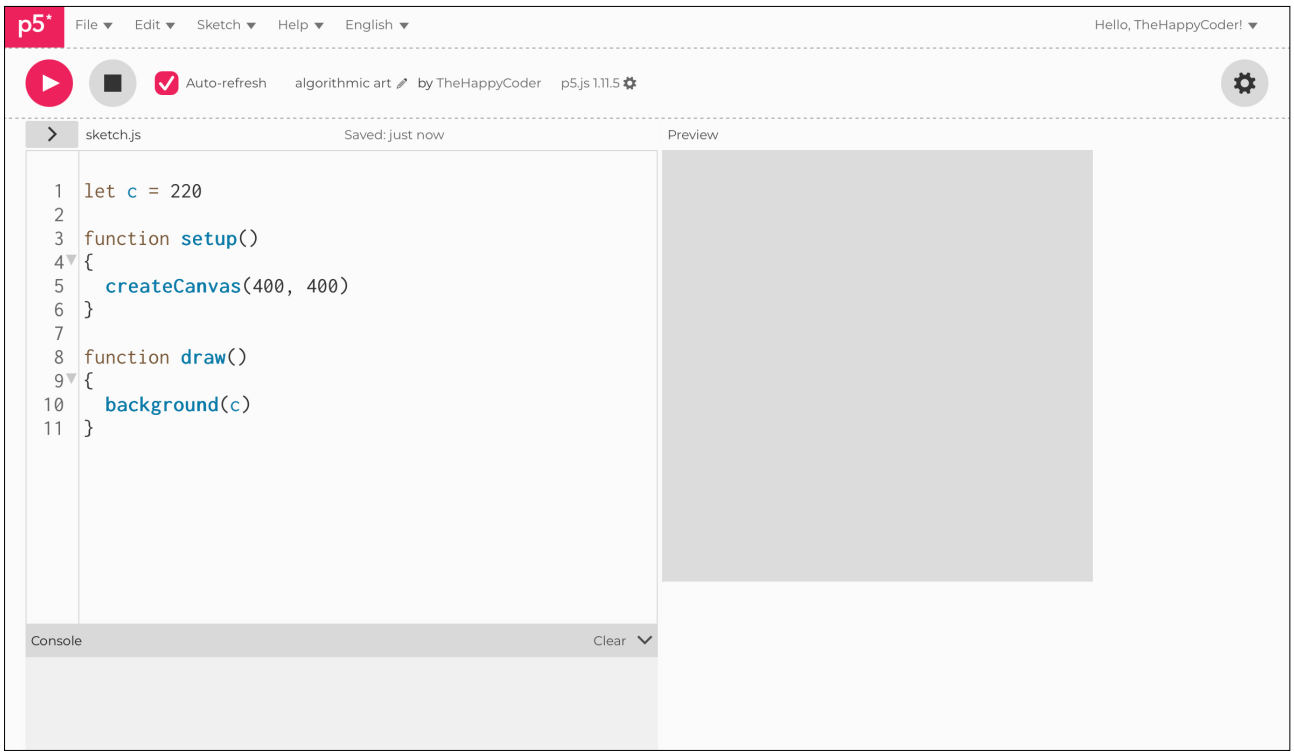
Notes

We are just creating a background variable and giving it a grey colour.

Code Explanation

`background(c)`	Giving the background the value of the variable c.

Figure B6.7



Sketch B6.8 blue circle

Let us make a nice blue circle using a hex colour value.

```
let c = 220

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

function draw()
{
  background(c)
  fill('#CBF0FF')
  circle(200, 200, 200)
}
```

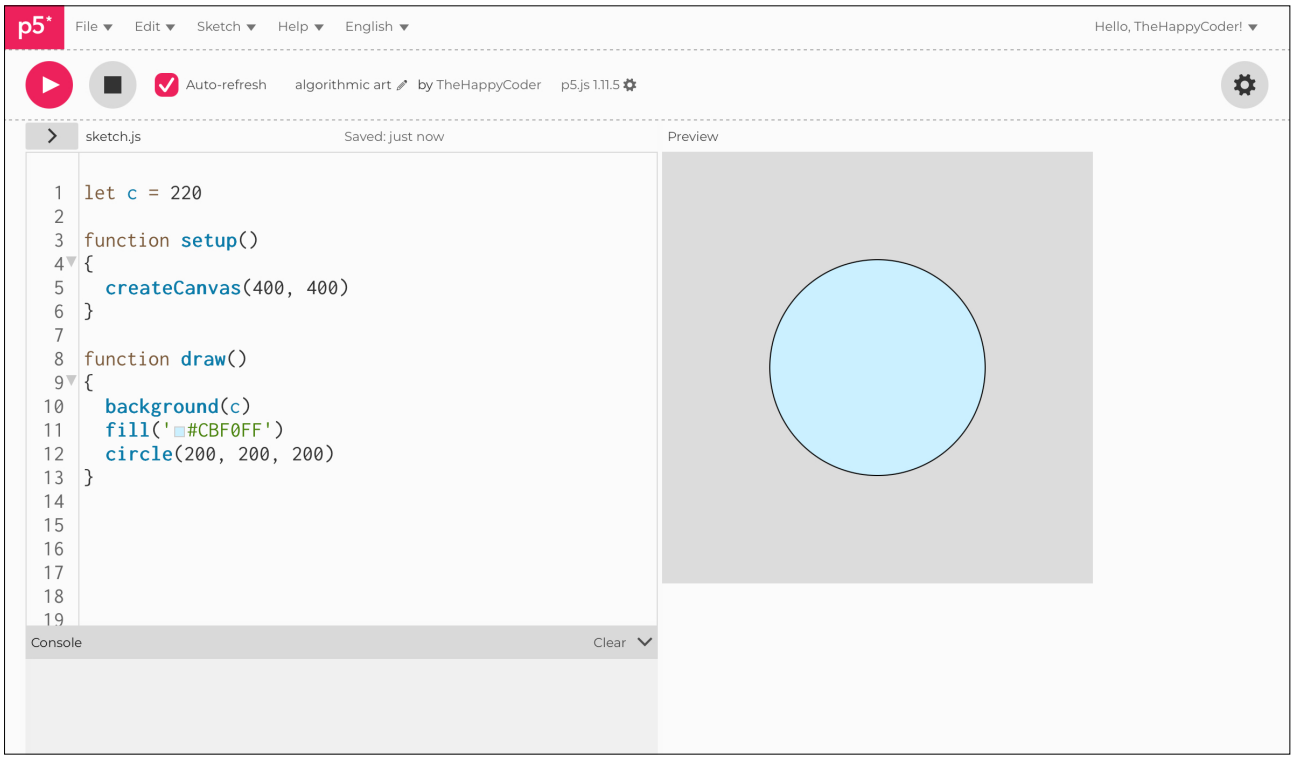
Notes

This gives us a colour to compare the background to.

Code Explanation

`fill('#CBF0FF')`	This is the hex value for light blue.

Figure B6.8



Sketch B6.9 some text

We will put the value of the `c` variable (background colour) on the canvas; you will see why shortly.

```
let c = 220

function setup()
{
  createCanvas(400, 400)
  textSize(50)
}

function draw()
{
  background(c)
  fill('#CBF0FF')
  circle(200, 200, 200)
  fill(0)
  stroke(255)
  text(c, 150, 75)
}
```

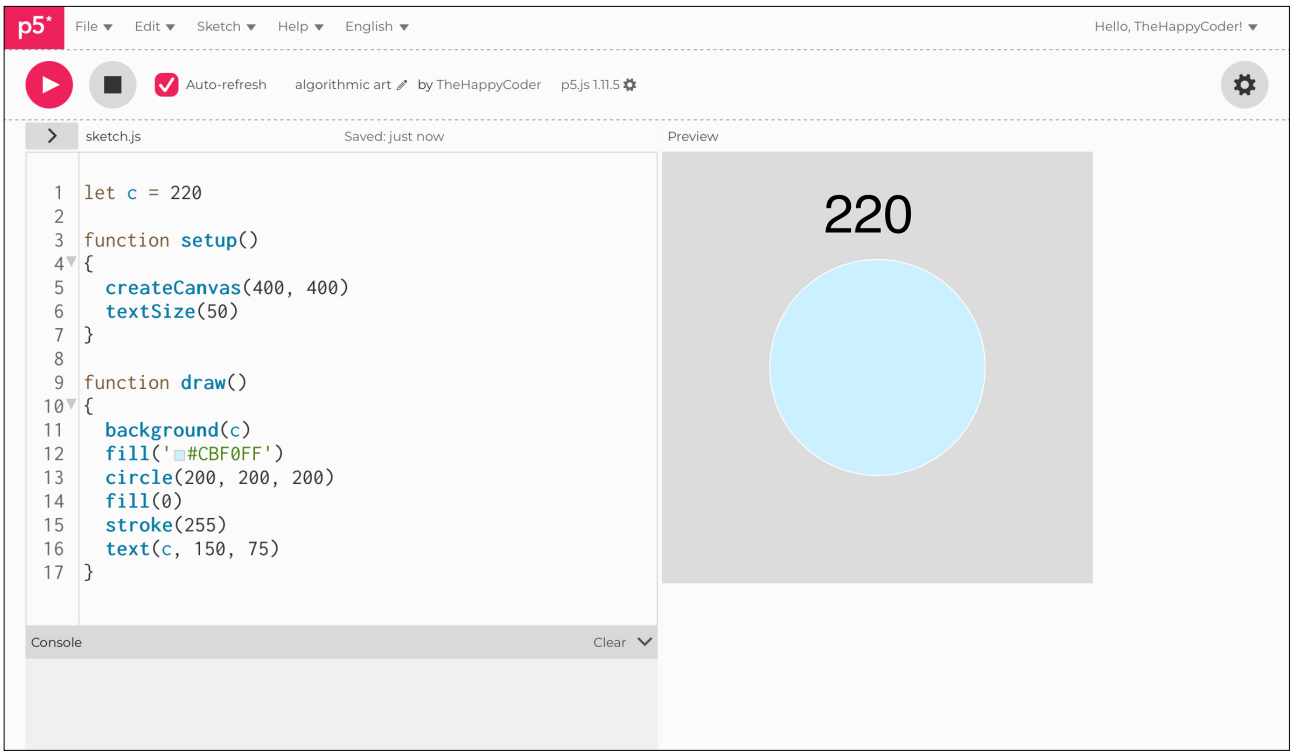
Notes

We want to see what the hex value will be; for now, it shows us the value of the background (the `c` value).

Code Explanation

`text(c, 150, 75)`	Puts the value of the variable as text on the canvas.

Figure B6.9



Sketch B6.10 colour picker

There is a function we can use that is built in, which allows us to pick colours. The `colorPicker()` function is now included. This produces a convenient table of useful colour information. You can even copy and paste the hex value you have selected.

```
let c = 220
let myPicker

function setup()
{
  createCanvas(400, 400)
  textSize(50)
  myPicker = createColorPicker('yellow')
}

function draw()
{
  c = myPicker.value()
  background(c)
  fill('#CBF0FF')
  circle(200, 200, 200)
  fill(0)
  stroke(255)
  text(c, 150, 75)
}
```

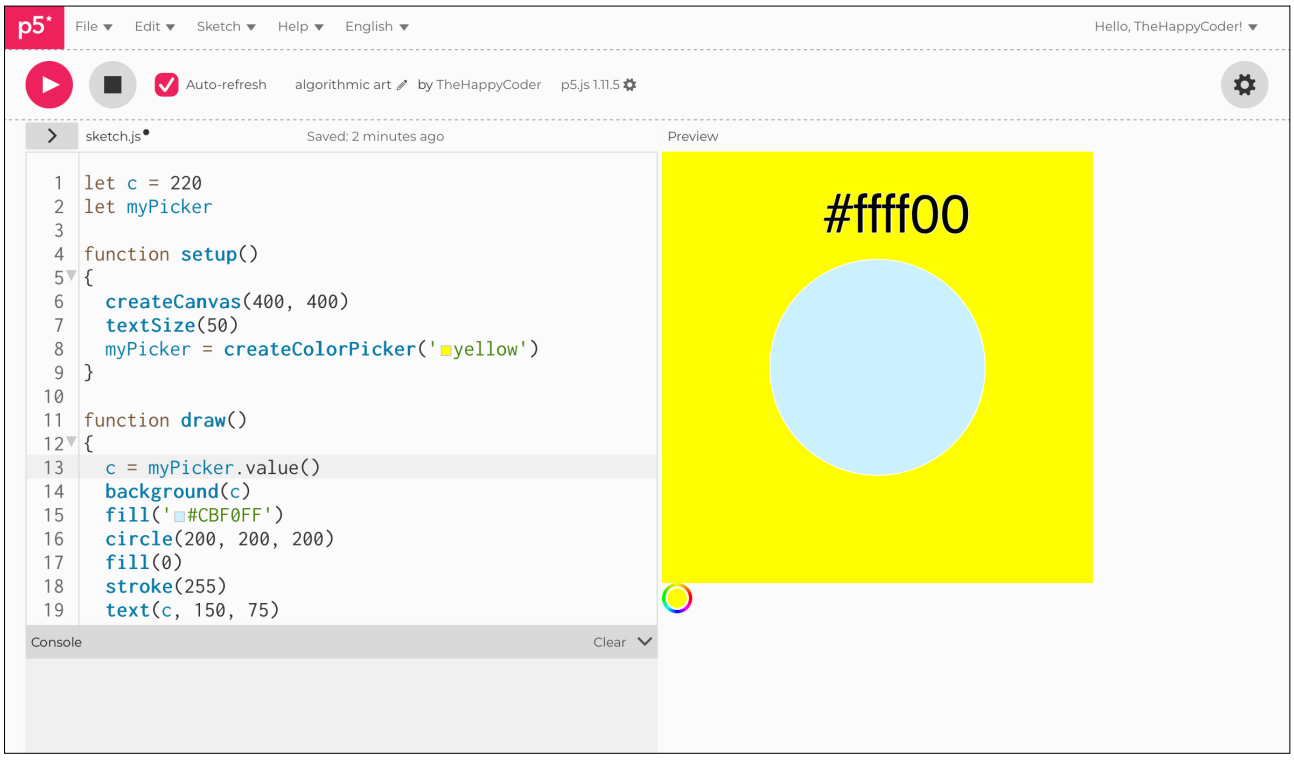
Notes

If you click on the icon at the bottom, you get presented with the option of three windows; each one is a different way to select a colour. The third one gives you the hex value as text, and you can copy that and paste it into your code to change the colour of the circle fill. If in the future you are using version `2.x` of `p5.js` the colour picker will look a bit different.

Code Explanation

`createColorPicker('yellow')`	We give the colorPicker() function an initial colour (yellow).
`c = myPicker.value()`	The variable c now pulls the hex value from the colour.
`text(c, 150, 75)`	This value is now displayed on the canvas.

Figure B6.10



Sketch B6.11 a bit of a tweak

We will move the colour picker. If you slide the canvas to the left, then you aren't obscuring the canvas when you click on the icon.

```
let c = 220
let myPicker

function setup()
{
  createCanvas(400, 400)
  textSize(50)
  myPicker = createColorPicker('yellow')
  myPicker.position(width, height)
}

function draw()
{
  c = myPicker.value()
  background(c)
  fill('#CBF0FF')
  circle(200, 200, 200)
  fill(0)
  stroke(255)
  text(c, 150, 75)
}
```

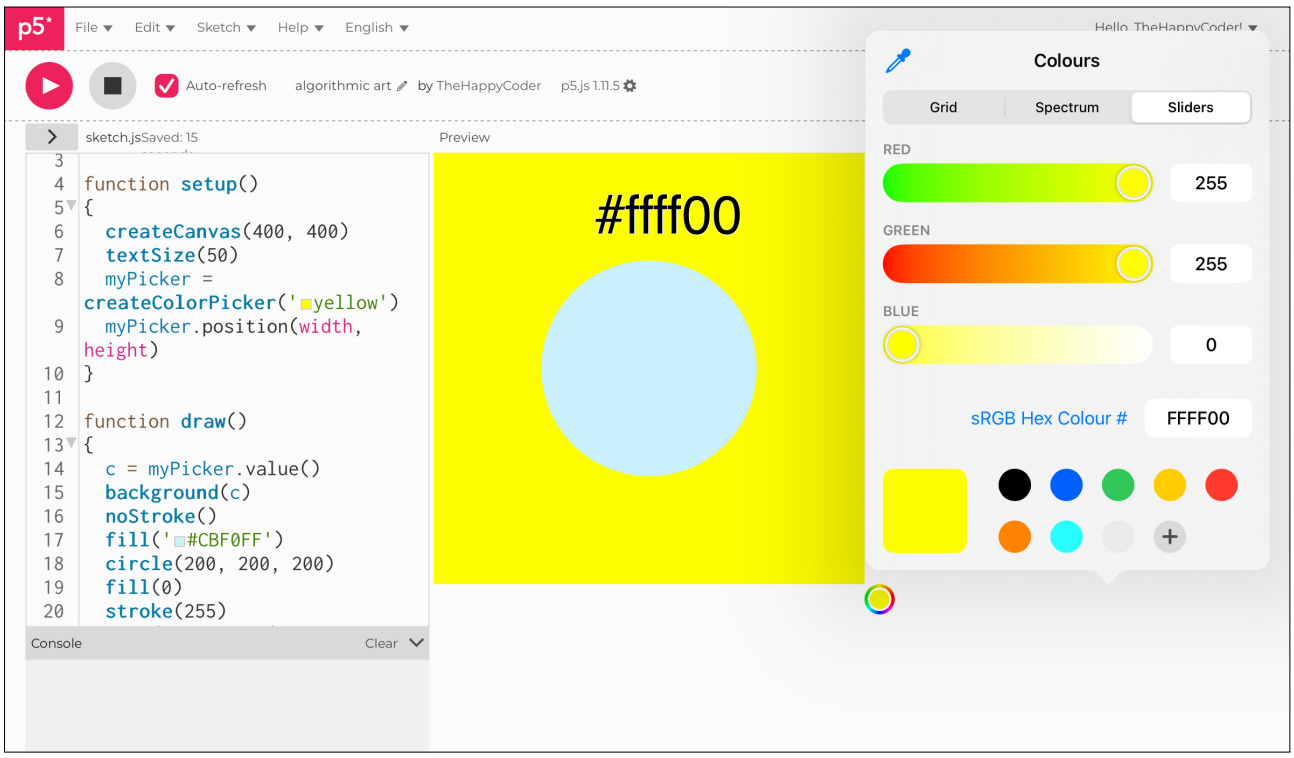
Notes

Works for me.

Code Explanation

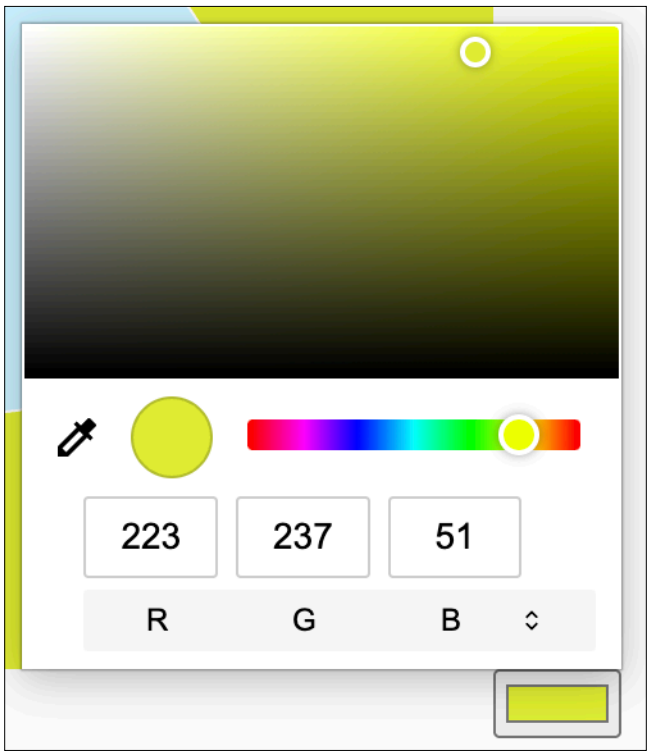
`myPicker.position(width, height)`	How to position the colour picker icon.

Figure B6.11

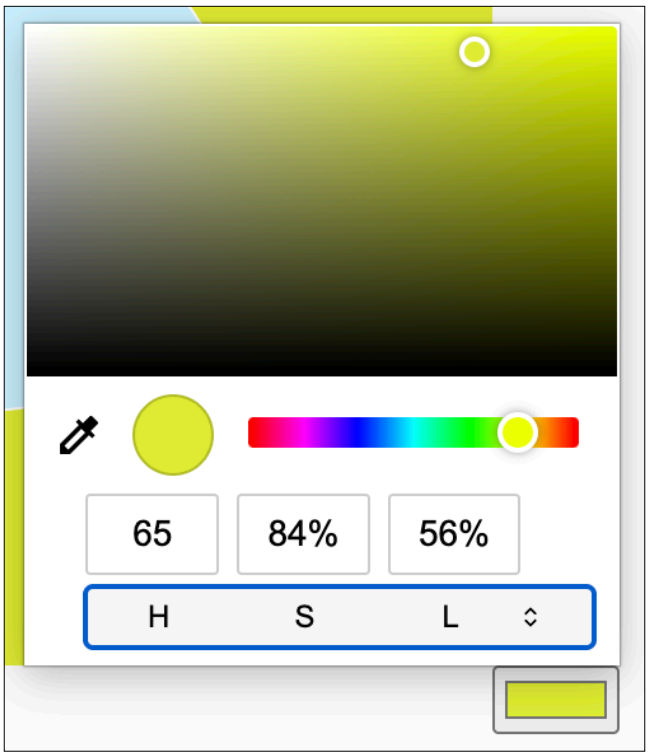


Version 2.x of p5.js (in Chrome) gives this choice (see below), as well as a colour picker.

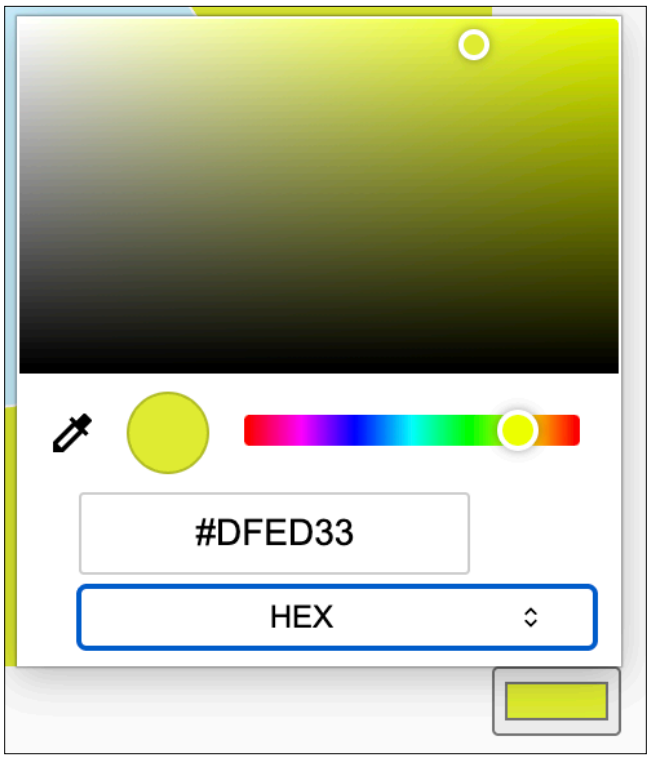
RGB values



HSL/HSB values



Hex values



Introduction to the colour charts

All the colour names, their HEX and RGB values. The groupings of colours are: Pink, Purple, Yellow, Red, Orange, Green, Blue, Brown, Cyan, Grey, and White.

! Remember that # precedes the hex value.

Pink Colours

Colour name	Hex value	RGB values
Pink	FFC0CB	255,192,203
LightPink	FFB6C1	255,182,193
HotPink	FF69B4	255,105,180
DeepPink	FF1493	255,20,147
PaleVioletRed	DB7093	219,112,147
MediumVioletRed	C71585	199,21,133

Pink Colours

Purple Colours

Colour name	Hex value	RGB values
Lavender	E6E6FA	230,230,250
Thistle	D8BFD8	216,191,216
Plum	DDA0DD	221,160,221
Orchid	DA70D6	218,112,214
Violet	EE82EE	238,130,238
Fuchsia	FF00FF	255,0,255
Magenta	FF00FF	255,0,255
MediumOrchid	BA55D3	186,85,211
DarkOrchid	9932CC	153,50,204
DarkViolet	9400D3	148,0,211
BlueViolet	8A2BE2	138,43,226
DarkMagenta	8B008B	139,0,139
Purple	800080	128,0,128
MediumPurple	9370DB	147,112,219
MediumSlateBlue	7B68EE	123,104,238
SlateBlue	6A5ACD	106,90,205
DarkSlateBlue	483D8B	72,61,139
RebeccaPurple	663399	102,51,153
Indigo	4B0082	75,0,130

Purple Colours

Yellow Colours

Colour name	Hex value	RGB values
Gold	FFD700	255,215,0
Yellow	FFFF00	255,255,0
LightYellow	FFFFE0	255,255,224
LemonChiffon	FFFACD	255,250,205
LightGoldenRodYellow	FAFAD2	250,250,210
PapayaWhip	FFEFD5	255,239,213
Moccasin	FFE4B5	255,228,181
PeachPuff	FFDAB9	255,218,185
PaleGoldenRod	EEE8AA	238,232,170
Khaki	F0E68C	240,230,140
DarkKhaki	BDB76B	189,183,107

Yellow Colours

Red Colours

Colour name	Hex value	RGB values
LightSalmon	FFA07A	255,160,122
Salmon	FA8072	250,128,114
DarkSalmon	E9967A	233,150,122
LightCoral	F08080	240,128,128
IndianRed	CD5C5C	205,92,92
Crimson	DC143C	220,20,60
Red	FF0000	255,0,0
FireBrick	B22222	178,34,34
DarkRed	8B0000	139,0,0

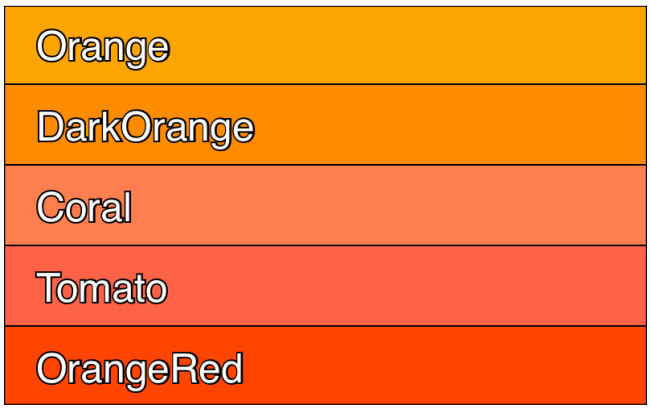
Red Colours


LightSalmon
Salmon
DarkSalmon
LightCoral
IndianRed
Crimson
Red
FireBrick
DarkRed

Orange Colours

Colour name	Hex value	RGB values
Orange	FFA500	255,165,0
DarkOrange	FF8C00	255,140,0
Coral	FF7F50	255,127,80
Tomato	FF6347	255,99,71
OrangeRed	FF4500	255,69,0

Orange Colours



Green Colours

Colour name	Hex value	RGB values
GreenYellow	ADFF2F	173,255,47
Chartreuse	7FFF00	127,255,0
LawnGreen	7CFC00	124,252,0
Lime	00FF00	0,255,0
LimeGreen	32CD32	50,205,50
PaleGreen	98FB98	152,251,152
LightGreen	90EE90	144,238,144
MediumSpringGreen	00FA9A	0,250,154
SpringGreen	00FF7F	0,255,127
MediumSeaGreen	3CB371	60,179,113
SeaGreen	2E8B57	46,139,87
ForestGreen	228B22	34,139,34
Green	008000	0,128,0
DarkGreen	006400	0,100,0
YellowGreen	9ACD32	154,205,50
OliveDrab	6B8E23	107,142,35
DarkOliveGreen	556B2F	85,107,47
MediumAquaMarine	66CDAA	102,205,170
DarkSeaGreen	8FBC8F	143,188,143
LightSeaGreen	20B2AA	32,178,170
DarkCyan	008B8B	0,139,139
Teal	008080	0,128,128

Green Colours

Blue Colours

Colour name	Hex value	RGB values
CadetBlue	5F9EA0	95,158,160
SteelBlue	4682B4	70,130,180
LightSteelBlue	B0C4DE	176,196,222
LightBlue	ADD8E6	173,216,230
PowderBlue	B0E0E6	176,224,230
LightSkyBlue	87CEFA1	35,206,250
SkyBlue	87CEEB	135,206,235
CornflowerBlue	6495ED	100,149,237
DeepSkyBlue	00BFFF	0,191,255
DodgerBlue	1E90FF	30,144,255
RoyalBlue	4169E1	65,105,225
Blue	0000FF	0,0,255
MediumBlue	0000CD	0,0,205
DarkBlue	00008B	0,0,139
Navy	000080	0,0,128
MidnightBlue	191970	25,25,112

Blue Colours

Brown Colours

Colour name	Hex value	RGB values
Cornsilk	FFF8DC	255,248,220
BlanchedAlmond	FFEBCD	255,235,205
Bisque	FFE4C4	255,228,196
NavajoWhite	FFDEAD	255,222,173
Wheat	F5DEB3	245,222,179
BurlyWood	DEB887	222,184,135
Tan	D2B48C	210,180,140
RosyBrown	BC8F8F	188,143,143
SandyBrown	F4A460	244,164,96
GoldenRod	DAA520	218,165,32
DarkGoldenRod	B8860B	184,134,11
Peru	CD853F	205,133,63
Chocolate	D2691E	210,105,30
Olive	808000	128,128,0
SaddleBrown	8B4513	139,69,19
Sienna	A0522D	160,82,45
Brown	A52A2A	165,42,42
Maroon	800000	128,0,0

Brown Colours

Cyan Colours

Colour name	Hex value	RGB values
Aqua	00FFFF	0,255,255
Cyan	00FFFF	0,255,255
LightCyan	E0FFFF2	24,255,255
PaleTurquoise	AFEEEE	175,238,238
Aquamarine	7FFFD4	127,255,212
Turquoise	40E0D0	64,224,208
MediumTurquoise	48D1CC	72,209,204
DarkTurquoise	00CED1	0,206,209

Cyan Colours

Grey Colours

Colour name	Hex value	RGB values
Gainsboro	DCDCDC	220,220,220
LightGrey	D3D3D3	211,211,211
Silver	C0C0C0	192,192,192
DarkGrey	A9A9A9	169,169,169
DimGrey	696969	105,105,105
Grey	808080	128,128,128
LightSlateGrey	778899	119,136,153
SlateGrey	708090	112,128,144
DarkSlateGrey	2F4F4F	47,79,79
Black	000000	0,0,0

Grey Colours

White Colours

Colour name	Hex value	RGB values
White	FFFFFF	255,255,255
Snow	FFFAFA	255,250,250
HoneyDew	F0FFF0	240,255,240
MintCream	F5FFFA	245,255,250
Azure	F0FFFF	240,255,255
AliceBlue	F0F8FF	240,248,255
GhostWhite	F8F8FF	248,248,255
WhiteSmoke	F5F5F5	245,245,245
SeaShell	FFF5EE	255,245,238
Beige	F5F5DC	245,245,220
OldLace	FDF5E6	253,245,230
FloralWhite	FFFAF0	255,250,240
Ivory	FFFFF0	255,255,240
AntiqueWhite	FAEBD7	250,235,215
Linen	FAF0E6	250,240,230
LavenderBlush	FFF0F5	255,240,245
MistyRose	FFE4E1	255,228,225

White Colours

White
Snow
HoneyDew
MintCream
Azure
AliceBlue
GhostWhite
WhiteSmoke
SeaShell
Beige
OldLace
FloralWhite
Ivory
AntiqueWhite
Linen
LavenderBlush
MistyRose