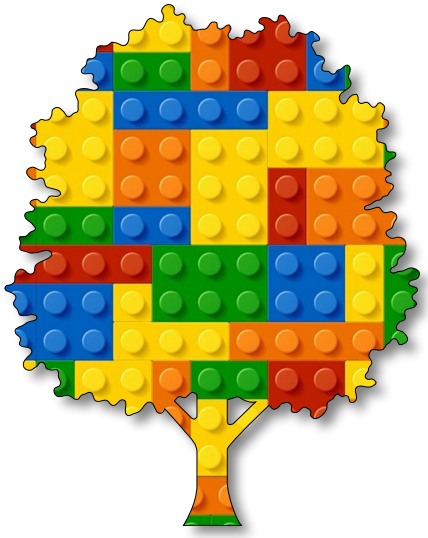
The Joy of Coding

Free Taster eBook

A very quick introduction to p5.js



Contents

Introduction to coding

What is coding?

Many people have asked me, "What is coding?" and variations on that theme. It's challenging to provide a concise and meaningful response to each person because everyone's perception is different, and the answer is quite lengthy. Therefore, I recommend trying it yourself to gain a feel for what coding entails.

This short eBook aims to give you a glimpse into the world of coding. It's particularly designed for beginners and those who have no idea what coding looks like. It's accessible to everyone, regardless of their abilities or age, as long as they have a computer, laptop, tablet, or smartphone.

What's in a name?

The beauty of this coding language is that it's all done in your web browser. There's nothing to download, pay for, or sign up for – it's completely free. So, there's really nothing stopping you. If you don't have a computer, libraries in the UK offer free or borrowed computers.

The coding language is a version of JavaScript called p5.js. Don't let that unassuming name deter you.

Enjoy the process

This coding language was developed by artists for artists who wanted a coding language they could create with. That's why it's so great: you can see what you're doing instantly, visually, and interactively. You type in your code, press the play button, and you'll see the result. You can add more lines, change things, and get the desired outcome.

As your skills progress at your own pace, you're building experience and understanding. There's no rush or pressure; learn at your own speed and, dare I say it, enjoy it. It can be rewarding and challenging, but at the end of the day, you can confidently say you can code, and that's just the beginning of your journey.

Baking a cake

I like to illustrate coding by comparing it to baking a cake. You have a list of ingredients and a recipe. It's crucial to follow the steps in the correct order without missing anything. This is true of coding as well. You're simply telling the computer a series of instructions that it will execute step by step.

It will only do what you tell it to do. Be very specific and clear in your instructions, or you'll get bad results. Mistakes are good; they're how you learn. So, be prepared to get it wrong and celebrate your successes. The core skill of coding is problem-solving.

Another way to think about it is like learning another language. There are rules to any language, a vocabulary that's relatively small compared to a living language, and a syntax that can be full of exceptions. In coding, the syntax is simple and generally repetitive, following simple rules.

Is it maths?

It's not hard maths. People assume it is, and if you look at some computer code from a software engineer, it may seem complex, full of obscure letters, words, and numbers. That alone can put people off, but it doesn't have to be like that. Usually, it isn't. Some software engineers just want to show off.

There's some basic maths involved, but the key to coding is the logic. You have to think through the logical steps very carefully. Even something as creative as painting requires logical planning, with the background first and then building up the detail as you progress. This isn't like painting by numbers.

To summarise:

- Work at your own pace.
- Take away what you want and create what you want.
- You can't fail, but you can learn.
- Take a small step each day. If you get frustrated, stop and come back to it the next day.
- Practise every day, but not for too long.
- Above all, try to enjoy it. You'll learn best when you're enjoying it, even when things don't work as you hoped or expected. Happy coding!

Getting started

This guide will help you get started with the software. The best part is that it's web-based and you can use it directly from the internet, just like a website.

Another great thing is that it's completely free. You can save your work and share it with your friends, parents, or grandchildren. This code was designed by artists to be easy to use and create stunning visual graphics for art, games, web design, or artificial intelligence.

Step by step

Step 1: Open your Chrome browser. Other browsers work, but Chrome works best.

Step 2: Type editor.p5js.org into your web browser. You should get a default page like the one below.

Step 3: Delete all the default code. Then, type in the code you see in this eBook.

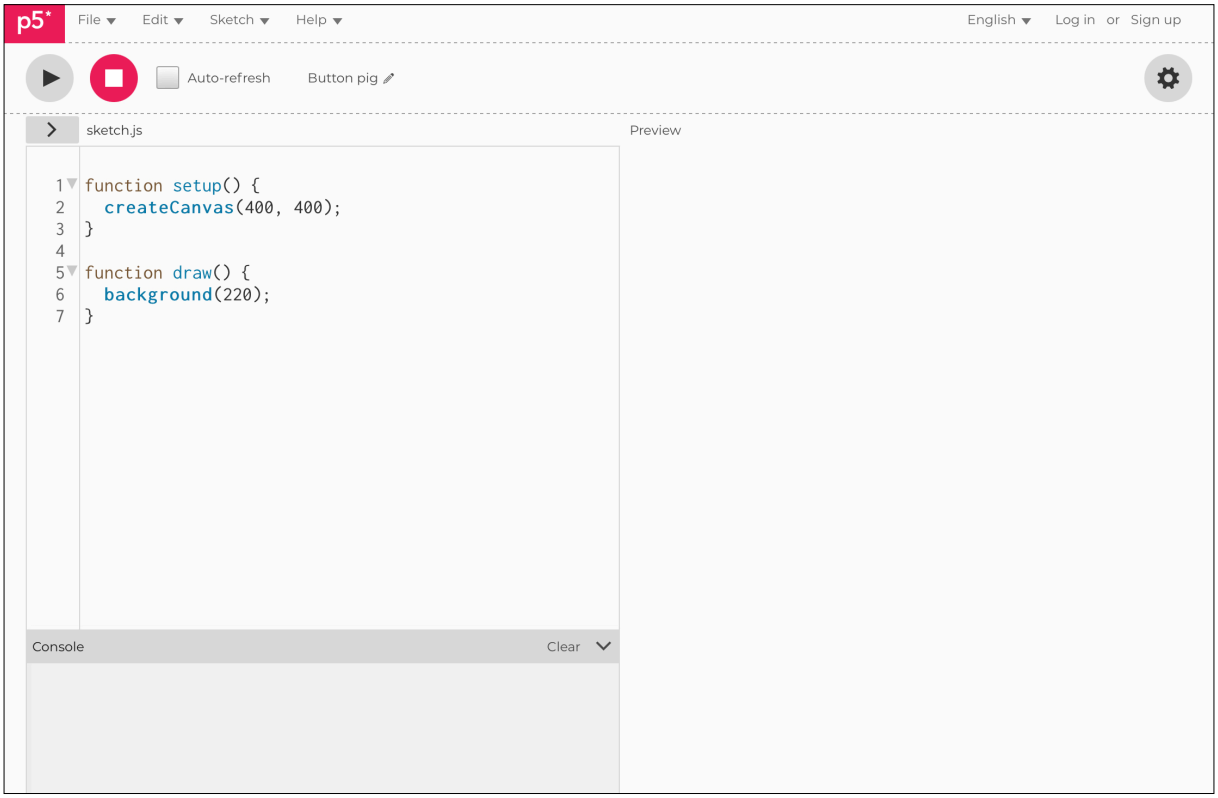
Step 4: Copy the code in the boxes in this eBook. Press the play button to see it appear on the canvas. If you get an error message in the console (at the bottom under the code), check that you haven't made a mistake.

Step 5: Try the challenges or just play to get a feel for it. See what happens when you change something. Learn by doing.

Step 6: Delete all the code you've typed in and move on to the next one.

Step 7: Work your way through each sketch and unit. You don't need to complete everything or fully understand everything. Most of your learning will come through regular and repeated practice. Try the challenges and read the notes; I've kept them as brief as possible.

Figure A: default page



The p5.js code

The code you type into the editor will appear in a box like the one underneath; this is called a sketch.

```
function setup()
{
  // this happens once
}
function draw()
{
  // this happens in a continuous loop
}
```

I've highlighted the only lines that have changed in blue so you don't have to retype everything. This way, you can make the necessary changes easily. After a sketch I may include some extra notes and a challenge or two. You don't have to do the challenges but I recommend playing with the code and break it to see what happens, learn from experience, not just following the instructions.

Notes

Some additional information.

Challenge

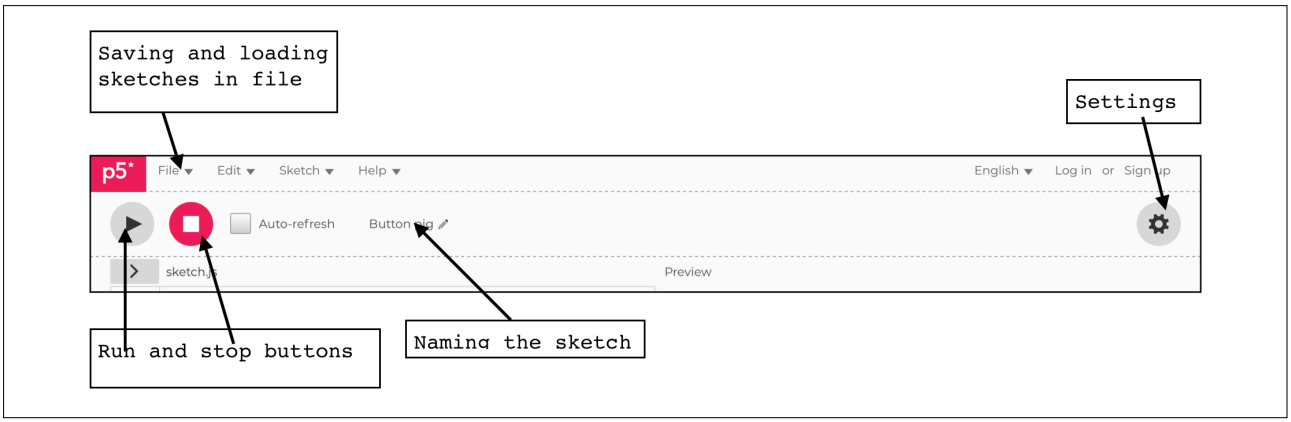
Something for you to try.

Having a look at the buttons

Everything is quite intuitive, and the best way to learn is to play with them. The `run` and `stop` buttons are the ones you'll use most. In settings, you can change the theme and font size, but if you're unsure, leave everything to the default settings.

When you work through this eBook, it's good practice to write the code repeatedly. Think of it as developing your memory muscle.

Figure B: the icons and the buttons



Sketch #1 our default sketch

You write the code on the left-hand side, and the code will draw something on the right-hand side on what is called the canvas. In the `setup()` function, we want to create the canvas, which is `400` pixels wide by `400` pixels high. In the `draw()` function, we give the canvas a light grey background colour.

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

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

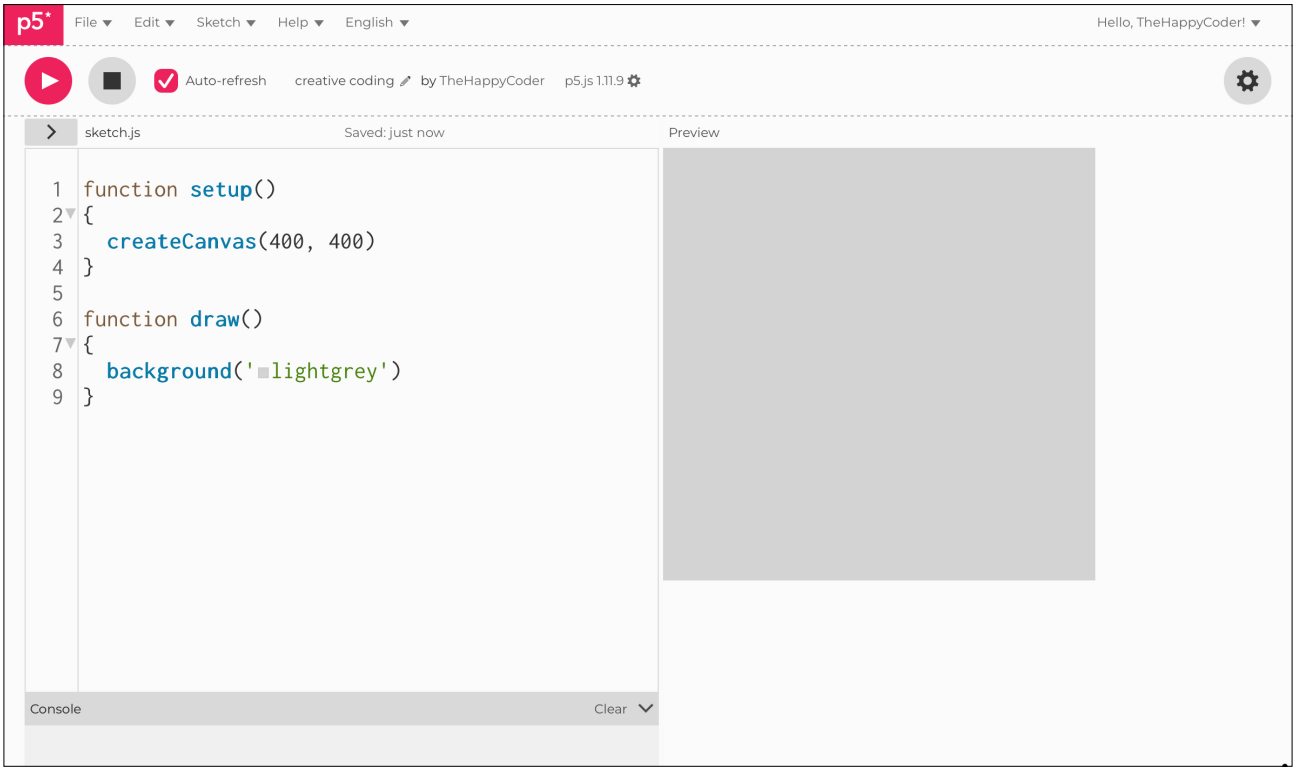
Notes

We can change the canvas size to anything you want. I have selected `400` by `400` because it should fit on any screen. The function `createCanvas()` is case-sensitive, which means what is lowercase has to be lowercase and what is in capitals has to be in capitals.

Challenge

Try `createCanvas(200, 600)` and see what happens.

Figure #1: this is what you should see when you run the code



Sketch #2 drawing your first shape

This is the first shape you will draw. The circle is drawn using three arguments: the first one is the `x` co-ordinate for the centre of the circle, the second is the `y` co-ordinate for the centre of the circle, and the third is the `diameter` of the circle.

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

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

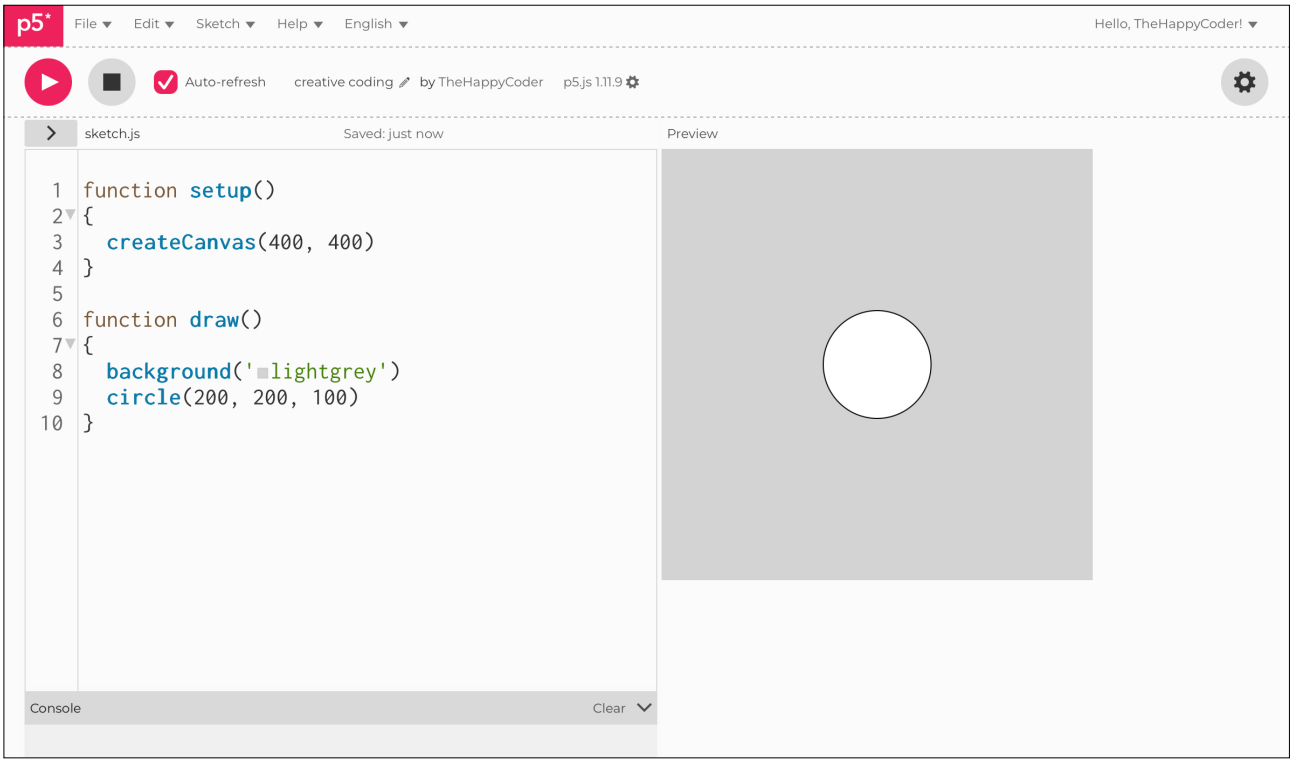
Notes

The position of the circle is measured in pixels from the top-left-hand corner. So it is `200` from the left and `200` from the top. In the `draw()` function, it first draws the background and then the circle. It then goes back to the start of the `draw()` function and draws the background again and then the circle and so on. This is repeated continuously.

Challenge

Try: `circle(100, 300, 50)`

Figure #2: drawing a simple circle, default is a white circle



Sketch #3 changing the background colour

We can use colours in four different ways, but for this eBook, I will introduce the simplest one (the course goes into much more detail). We can just write in the word for that colour in speech marks.

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

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

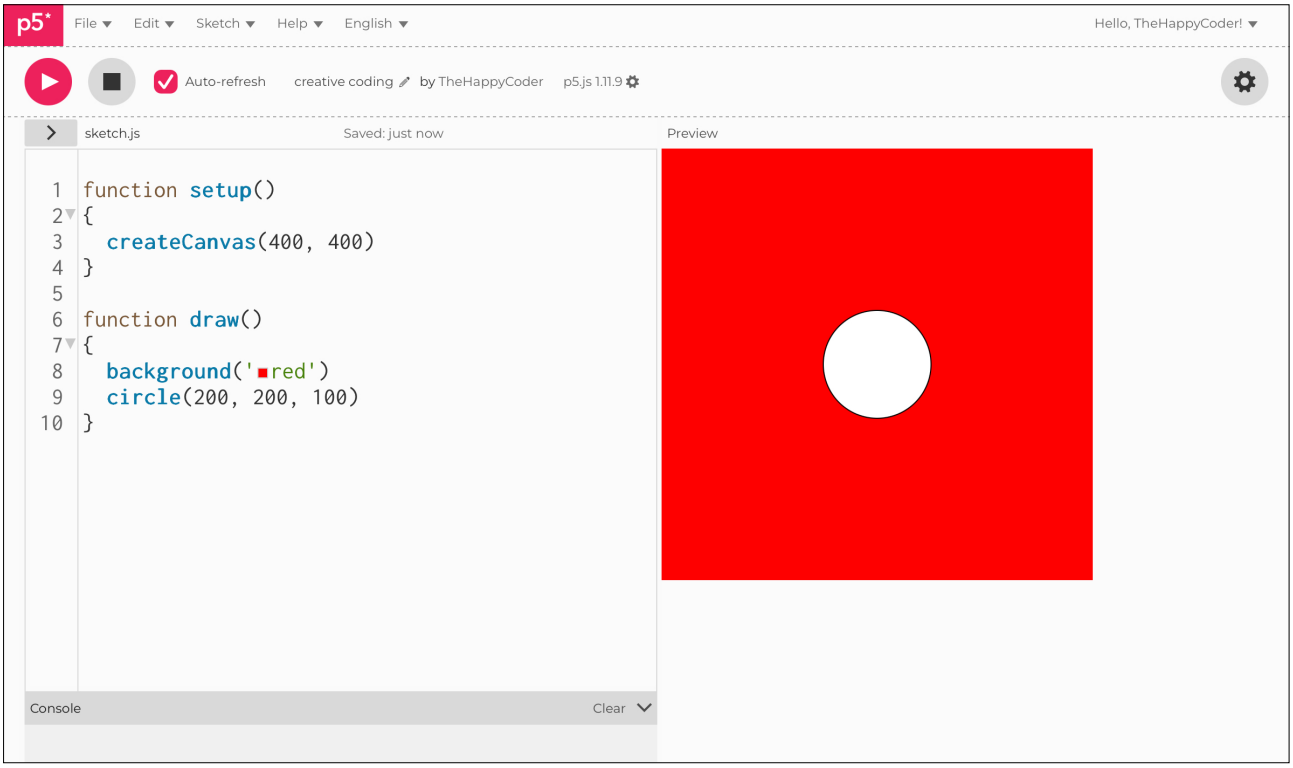
🗒️ Notes

The other ways include what is called RGB, the red, blue, and green. These are three arguments detailing how much red, green, or blue. There is also HSB, which stands for Hue, Saturation, and Brightness. The fourth one is a hex value which starts with a hashtag # followed by a string of numbers and/or letters.

🌻 Challenges

1. Try background(‘yellow’) or background(‘blue’), etc.
2. Try background(255, 0, 0) and see what happens.

Figure #3: you get a red background



Sketch #4 filling the shapes with colour

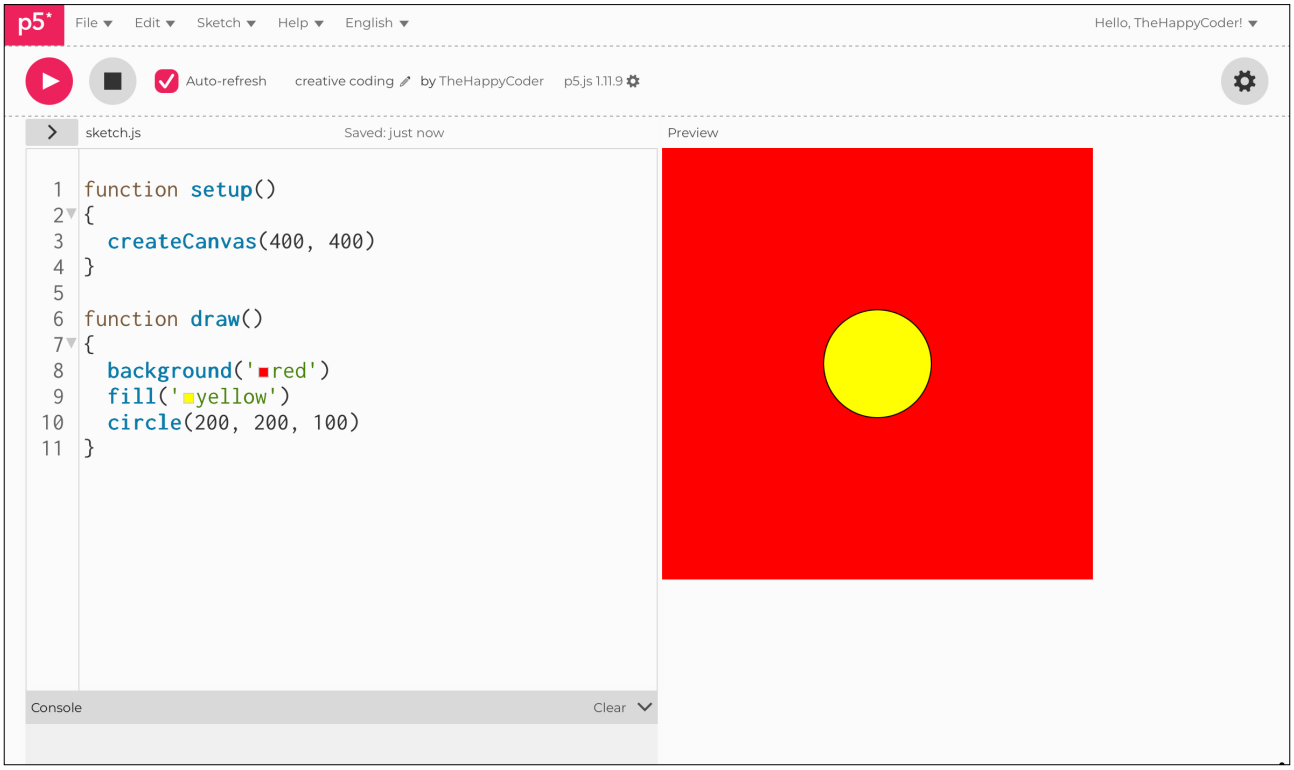
Using the `fill()` function, we can fill shapes in with colour also.

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

function draw()
{
  background('red')
  fill('yellow')
  circle(200, 200, 100)
}
```

Notes

We simply define what colour we want and use the `fill()` function. This works for all shapes.

Figure #4: now we have a yellow circle



Sketch #5 adding another circle

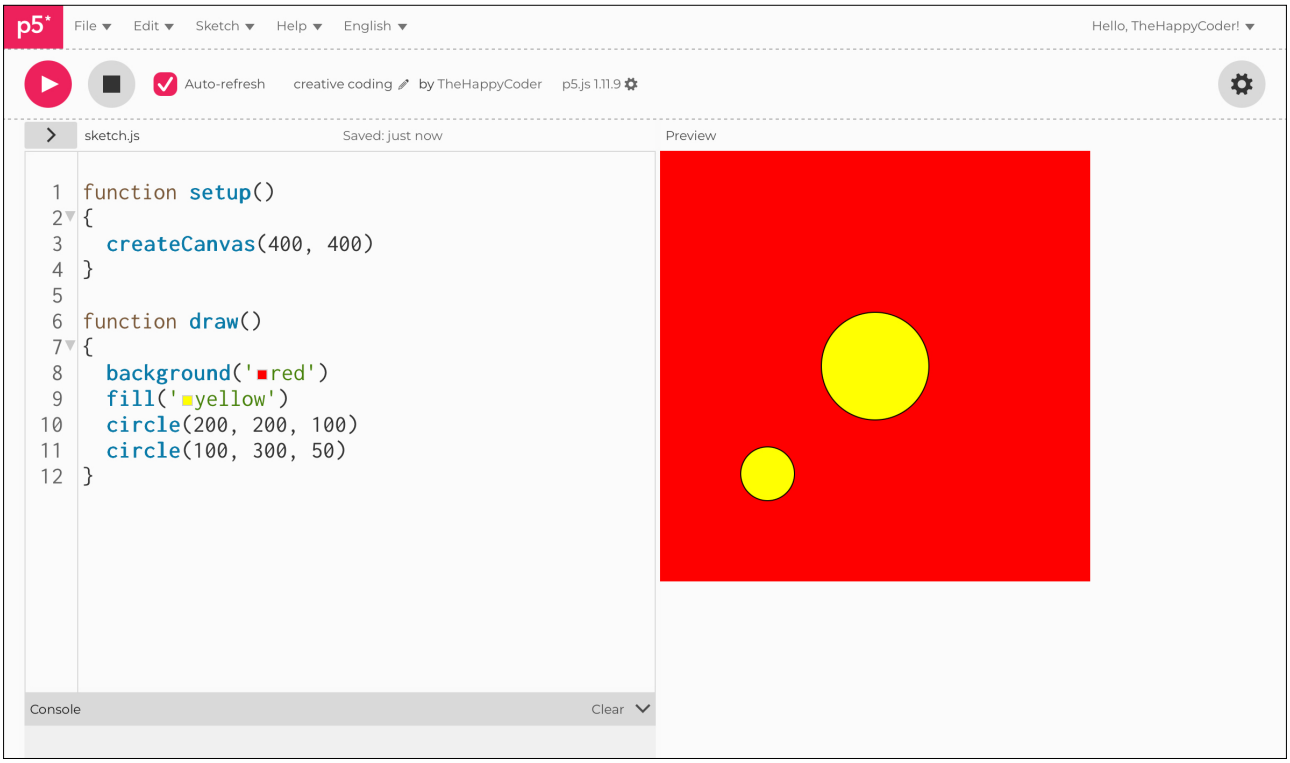
We will add another circle.

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

function draw()
{
  background('red')
  fill('yellow')
  circle(200, 200, 100)
  circle(100, 300, 50)
}
```

Notes

It also fills that one in yellow. It will fill all other shapes in yellow too. Remember, in the `draw()` function, it is an endless loop. It starts at the top, does the first line, then onto the next line, and so on until it reaches the last line of code before going back to the start of the loop.

Figure #5: two circles



Sketch #6 change the colour

We can give the second circle its own colour.

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

function draw()
{
  background('red')
  fill('yellow')
  circle(200, 200, 100)
  fill('green')
  circle(100, 300, 50)
}
```

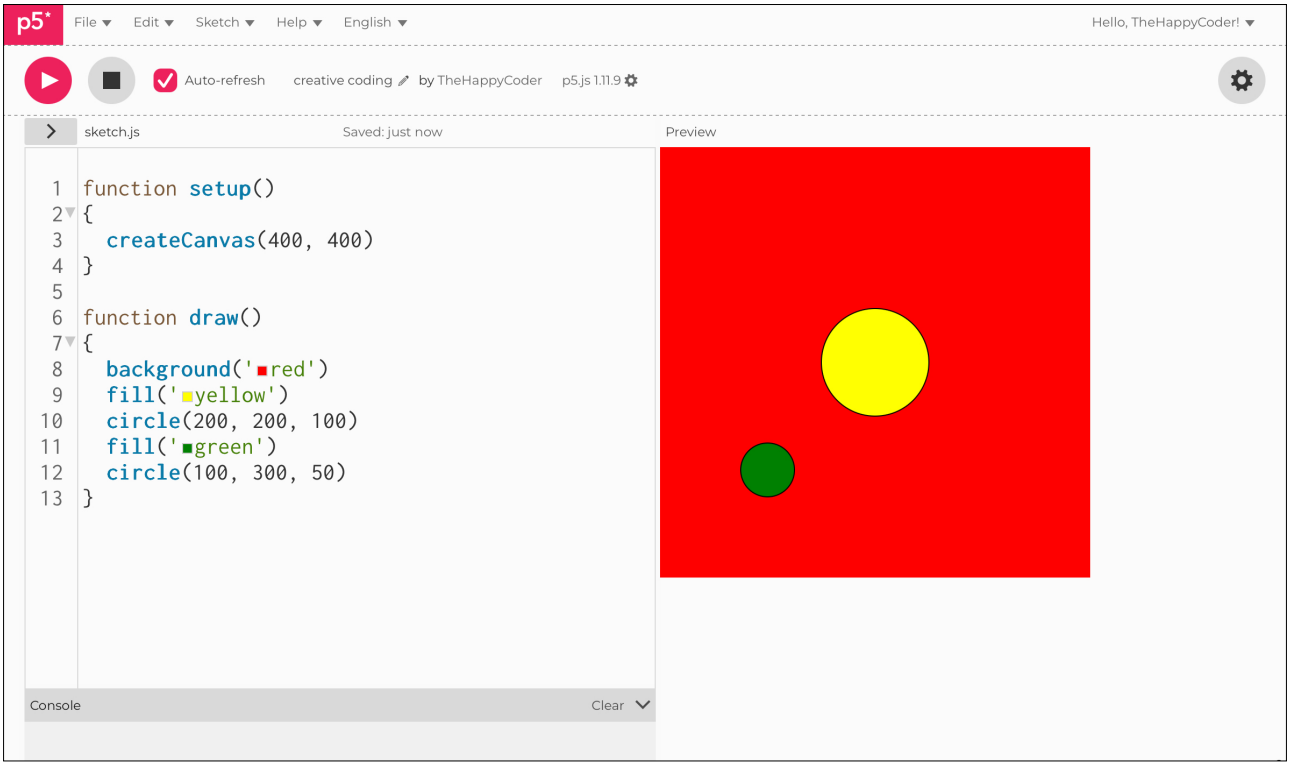
Notes

This new colour has to be put in the right place between the two circles.

Challenges

Draw more circles.
Give each one a different colour.

Figure #6: it is now green



Sketch #7 creating a variable for (x, y)

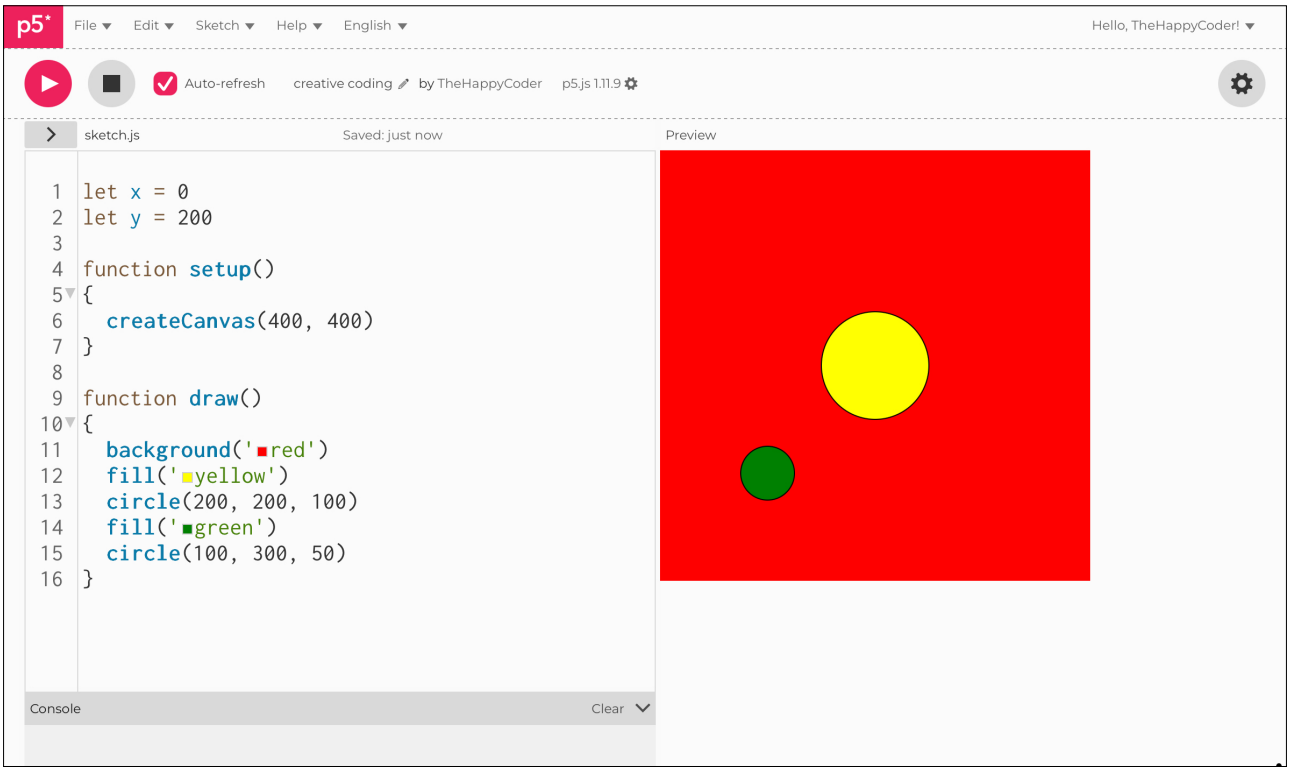
We are going to give the co-ordinates of our circle an `x` value and a `y` value. It is done using the variable type `let`, which tells the computer that what comes next is a variable name.

```
let x = 0
let y = 200

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

function draw()
{
  background('red')
  fill('yellow')
  circle(200, 200, 100)
  fill('green')
  circle(100, 300, 50)
}
```

Notes

We now have a variable called `x` with a value of `0`, and we have a variable called `y` with a value of `200`. Nothing appeared to change.

Figure #7: nothing new has happened



Sketch #8 using the variables

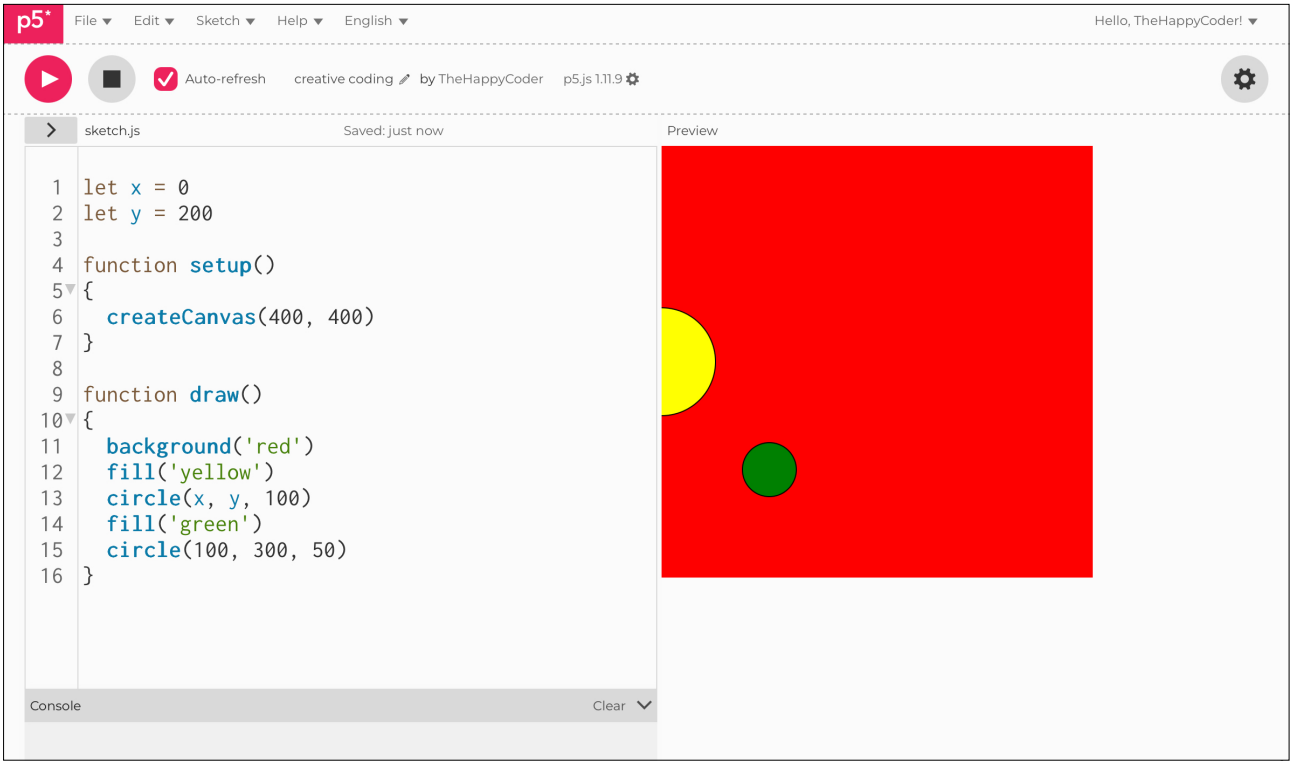
We have the variables named and given them a value, so now we need to use them. We are going to give them to the circle.

```
let x = 0
let y = 200

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

function draw()
{
  background('red')
  fill('yellow')
  circle(x, y, 100)
  fill('green')
  circle(100, 300, 50)
}
```

Notes

You may be wondering why bother, but doing that, the reason is it then gives you many possibilities to use variables to create many pleasing and powerful effects.

Figure #8: the yellow circle has moved to the left



Sketch #9 it disappears

We are going to start moving the yellow circle by adding `1` to `x` in each iteration (cycle) of the `draw()` loop.

```
let x = 0
let y = 200

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

function draw()
{
  background('red')
  fill('yellow')
  circle(x, y, 100)
  fill('green')
  circle(100, 300, 50)
  x = x + 1
}
```

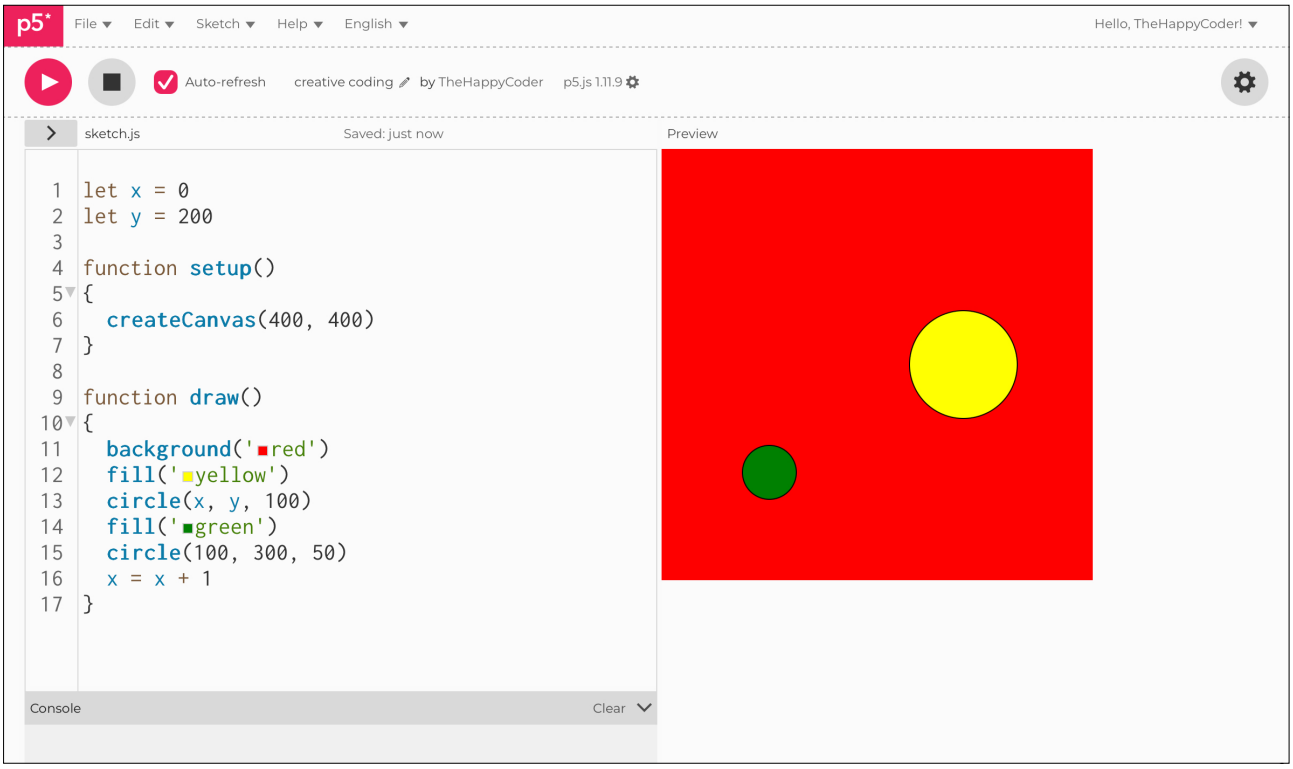
Notes

The line of code `x = x + 1` simply adds `1` to itself.

Challenge

Try: `x = x + 5`.

Figure #9: the circle moves serenely across the canvas and then disappears



Sketch #10 it reappears

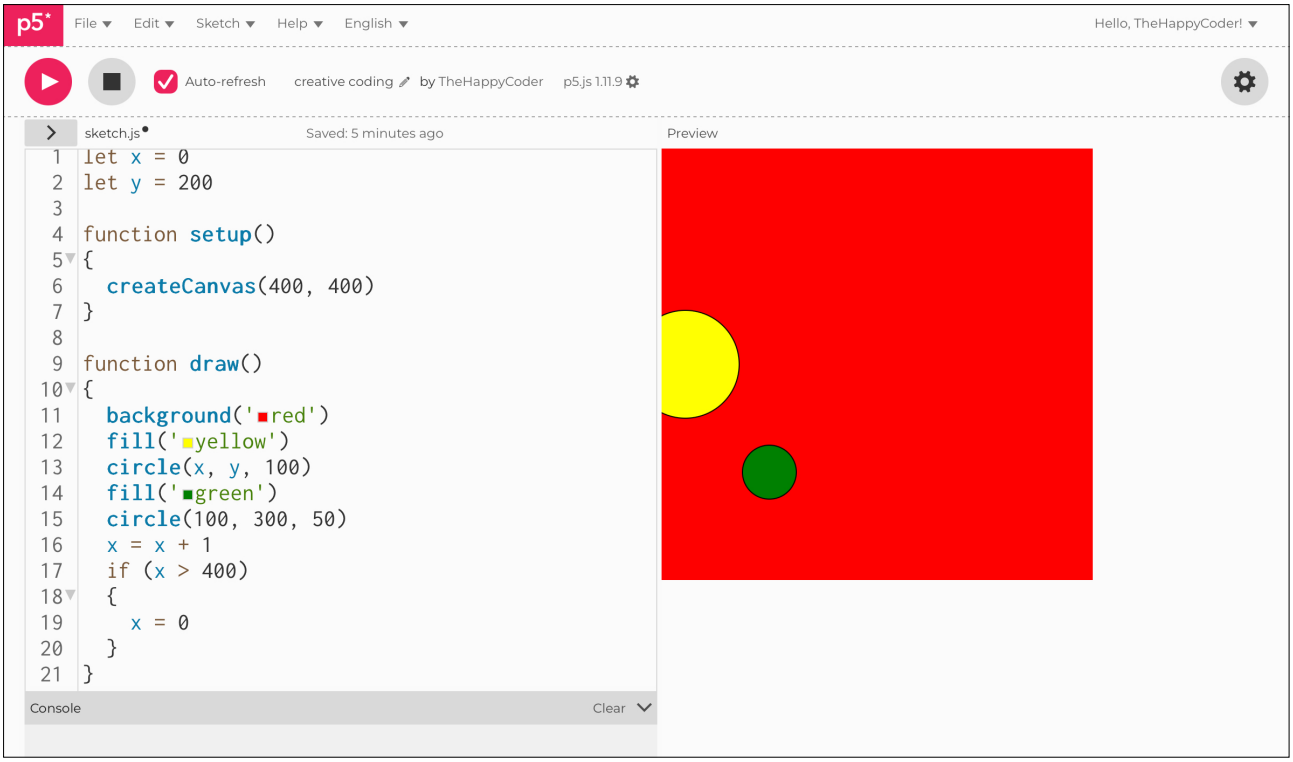
That was nice, but is there any way of bringing it back? Yes, there is. What we need is a conditional statement. Something like if `x` is more than the width of the canvas, move it back to the start. We can write this in code shorthand: `if (x > 400)`.

```
let x = 0
let y = 200

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

function draw()
{
  background('red')
  fill('yellow')
  circle(x, y, 100)
  fill('green')
  circle(100, 300, 50)
  x = x + 1
  if (x > 400)
  {
    x = 0
  }
}
```

Notes

The yellow circle disappears and then magically reappears.

Figure #10: and here it is



Sketch #11 a bit nicer

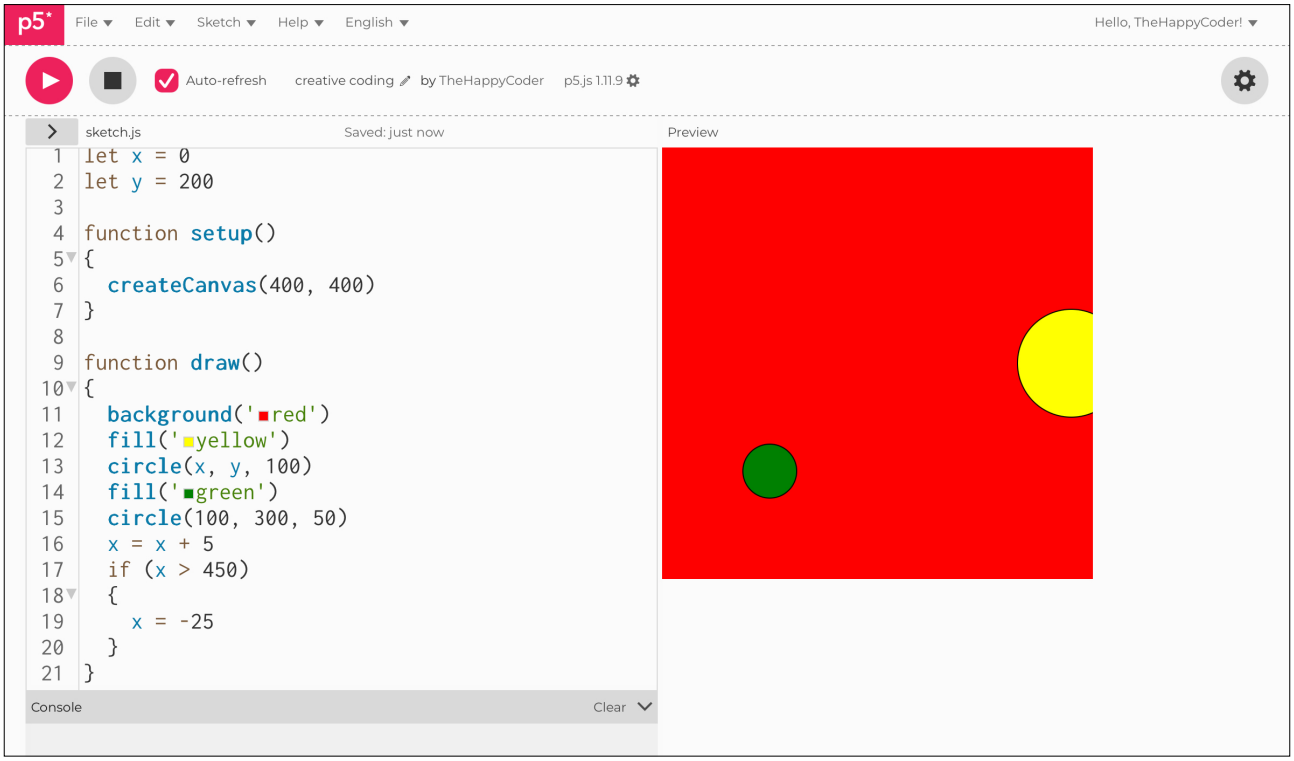
Let's just tweak it a little. First, we will make it go much faster, and secondly, we will make it reappear when it is more than 450, which includes the radius of the circle.

```
let x = 0
let y = 200

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

function draw()
{
  background('red')
  fill('yellow')
  circle(x, y, 100)
  fill('green')
  circle(100, 300, 50)
  x = x + 5
  if (x > 450)
  {
    x = -25
  }
}
```

Notes

One of the purposes of creative coding is to make things seem natural.

Figure #11: it returns nicely



If you are still interested...

I have a series of workbooks available on Amazon (eBooks). They cover everything in more depth and breadth. Go to my Website (if not already there at: www.ElegantAi.org for more information and additional units.