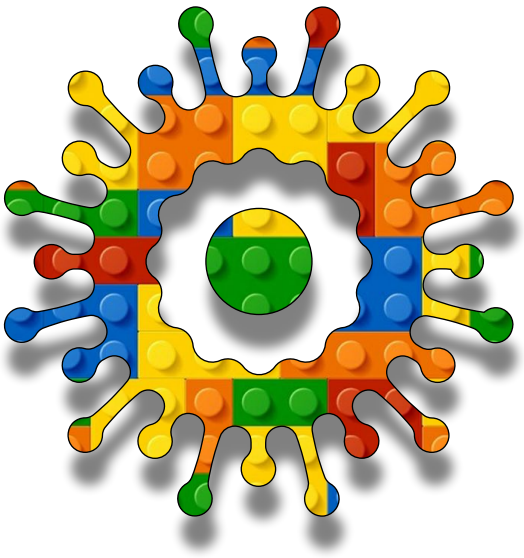
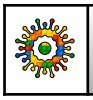
Artificial Intelligence Module A Unit #1 p5.js code snippets 1





Content

Module A Unit #1 code snippets part 1

Introducing arrays

Introduction to code snippets part 1 with p5.js

This is a gentle introduction to p5.js as it relates to the following modules on developing simple machine learning models with ml5.js on the p5.js platform. They are a brief introduction to the code you will be using. These snippets are also helpful for those who have never coded before or for those who are unfamiliar with p5.js.

My recommendation is that you work quickly through the coding snippets but at your own pace and play around with the code; it will help you to learn more about coding in p5.js. If there are challenges, then try them out and see what else you can learn; the more practice the better. Over time, it will become intuitive, like learning to ride a bike, drive a car, or learning the piano. At first, it may seem strange, but later it becomes second nature.

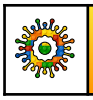
A brief explanation

- You will have the code in the yellow box to type into the editor.
- Any changes from the previous lines of code are highlighted in blue.
- A sketch is a complete code which you can run (usually).
- There will be a brief description of what you are doing and why.
- You will have some additional information in the form of notes.
- Sometimes there will be some challenges; they are optional.
- To help understand the code, there will be very brief descriptions about some lines of code which will appear in a box shown below.

Line of code	Explanation

- The snippets will be kept as brief as possible.
- Delete the default code and start with an empty editor.
- Wherever relevant, I will include an image of what it should look like.
- As you code each step, it doesn't mean that the sketch is ready to run.
- Make sure you have it set up as you want with the font size, theme, etc.

🀄 The curly brackets are important, as are any commas or semicolons; take care.



Sketch A1.1 function setup()

❗ this is our first sketch. Delete all of the default code and write the code below.
The `setup()` function is a built-in function. It simply runs through all the code between the two curly brackets `{...}` once. It is used to set up the code you are going to run.

```
function setup()
{


}
```

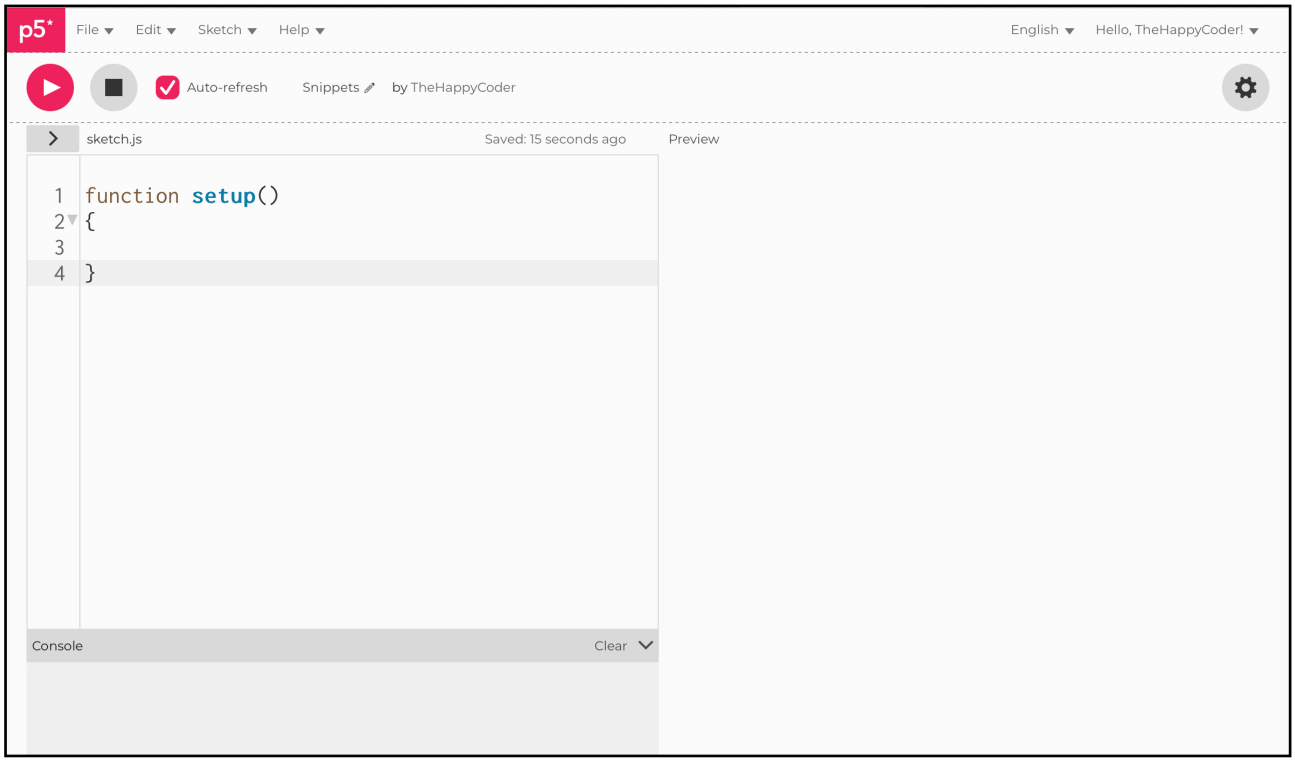
🗒️ Notes

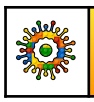
The `setup()` function is called once when the sketch begins running. Declaring the function `setup()` sets a code block to run once automatically when the sketch starts running. It's used to perform setup tasks such as creating the canvas and initialising variables.

🛠️ Code Explanation

`function setup()`	Runs everything coded in this function once

Figure A1.1


p5*
File
Edit
Sketch
Help
English
Hello, TheHappyCoder!
Auto-refresh
Snippets
by TheHappyCoder
sketch.js
Saved: 15 seconds ago
Preview
1 function setup()
2 {
3
4 }
Console
Clear



Sketch A1.2 function draw()

❗ now add the code highlighted in blue.

The `draw()` function is the second built-in function. It is a loop function because it goes through all the code between the curly `{...}` brackets continuously in a loop.

```
function setup()
{


}

function draw()
{


}
```

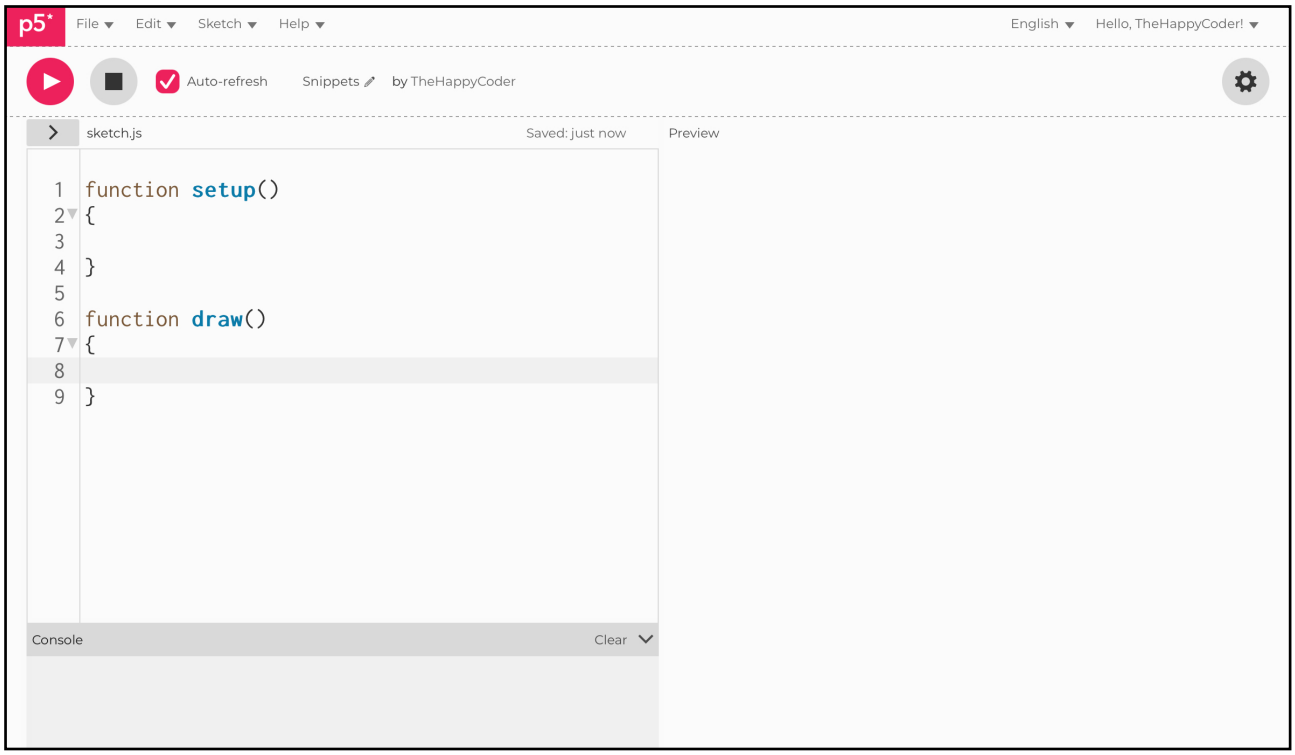
🗒️ Notes

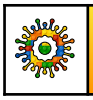
This is a pre-built function. The `draw()` function is one that is called repeatedly while the sketch runs. Declaring the function `draw()` sets a block of code to run repeatedly once the sketch starts. It's used to create animations and respond to user inputs.

🛠️ Code Explanation

`function draw()`	Loops through the code inside the brackets continuously

Figure A1.2


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



Sketch A1.3 createCanvas()

This function creates the canvas on which you are going to draw. It requires at least two arguments. The first one is the `width`, and the second is the `height`. Both are measured in `pixels`.

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

function draw()
{

}
```

🗒️ Notes

This creates a canvas element on the web page. The `createCanvas()` function creates the main drawing canvas for a sketch. It should only be called once at the beginning of `setup()`. Calling `createCanvas()` more than once causes unpredictable behaviour. The first two parameters set the dimensions of the canvas, and the values of the width and height system variables. For example, calling `createCanvas(900, 500)` creates a canvas that's `900` pixels wide by `500` pixels high. By default, width and height are both 100. We can't see anything yet because we need to give it a background to code onto.

🌻 Challenge

Play with the dimensions of the canvas.

🛠️ Code Explanation

`createCanvas(400, 400)`	This creates a canvas to code onto. The two arguments are the width and the height measured in pixels

Dimensions

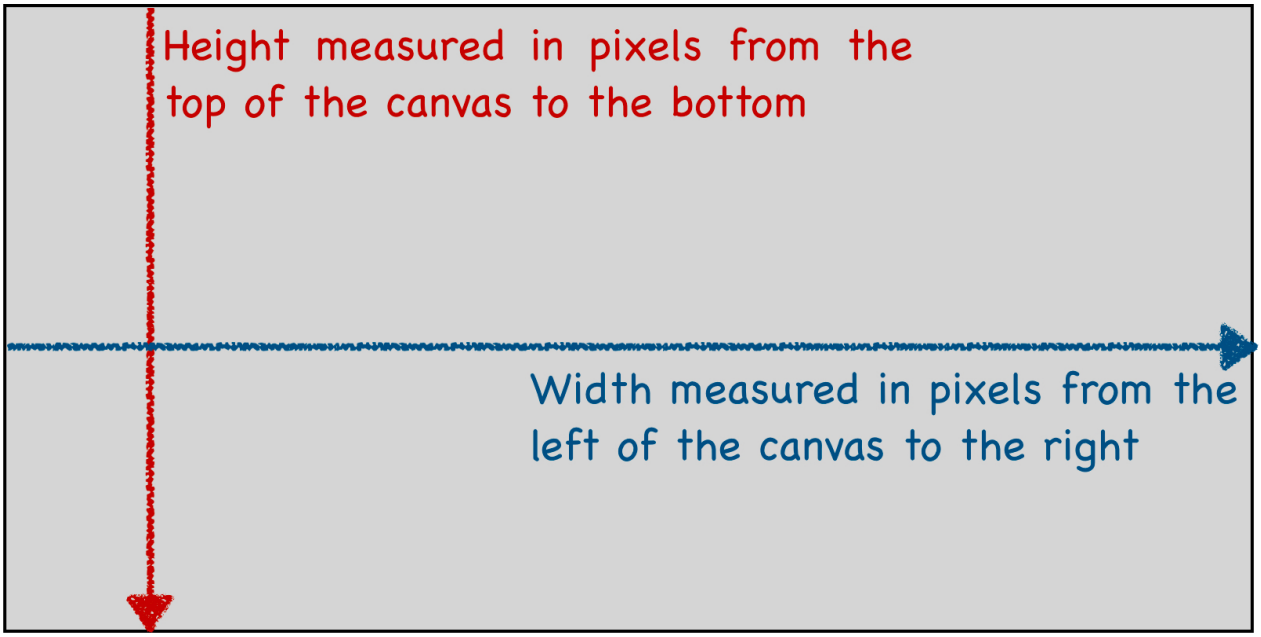
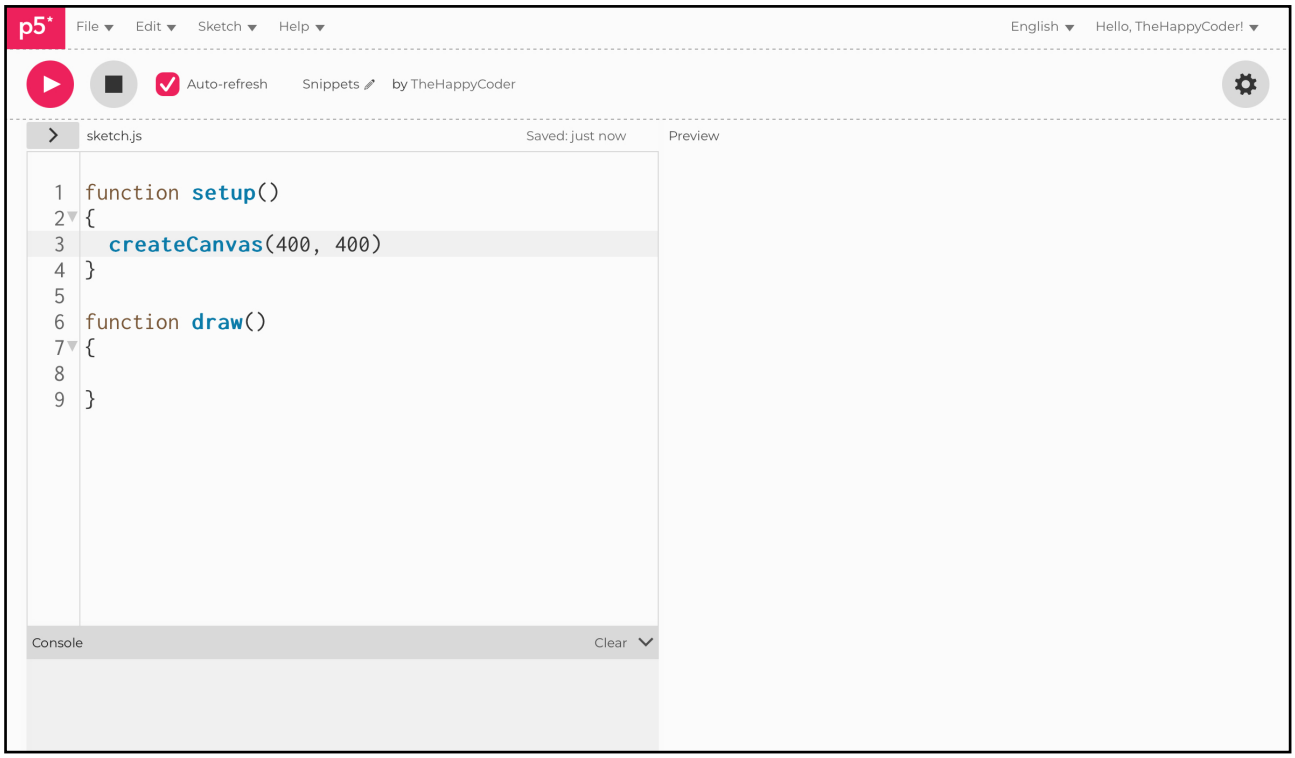
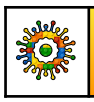

Height measured in pixels from the top of the canvas to the bottom
Width measured in pixels from the left of the canvas to the right

Figure A1.3


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



Sketch A1.4 background()

The `background()` function allows us to give the canvas a background colour. By default, it is white. Generally, we will give it a grey colour (`220`), where `0` is black and `255` is white; all the other numbers in between are shades of grey.

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

function draw()
{


}
```

Notes

Sets the colour used for the background of the canvas. By default, the background is transparent. The `background()` function is typically used within `draw()` to clear the display window at the beginning of each frame. It can also be used inside `setup()` to set the background on the first frame of animation. The value `220` gives you a very light grey, whereas a value of `50` gives you a darker grey.

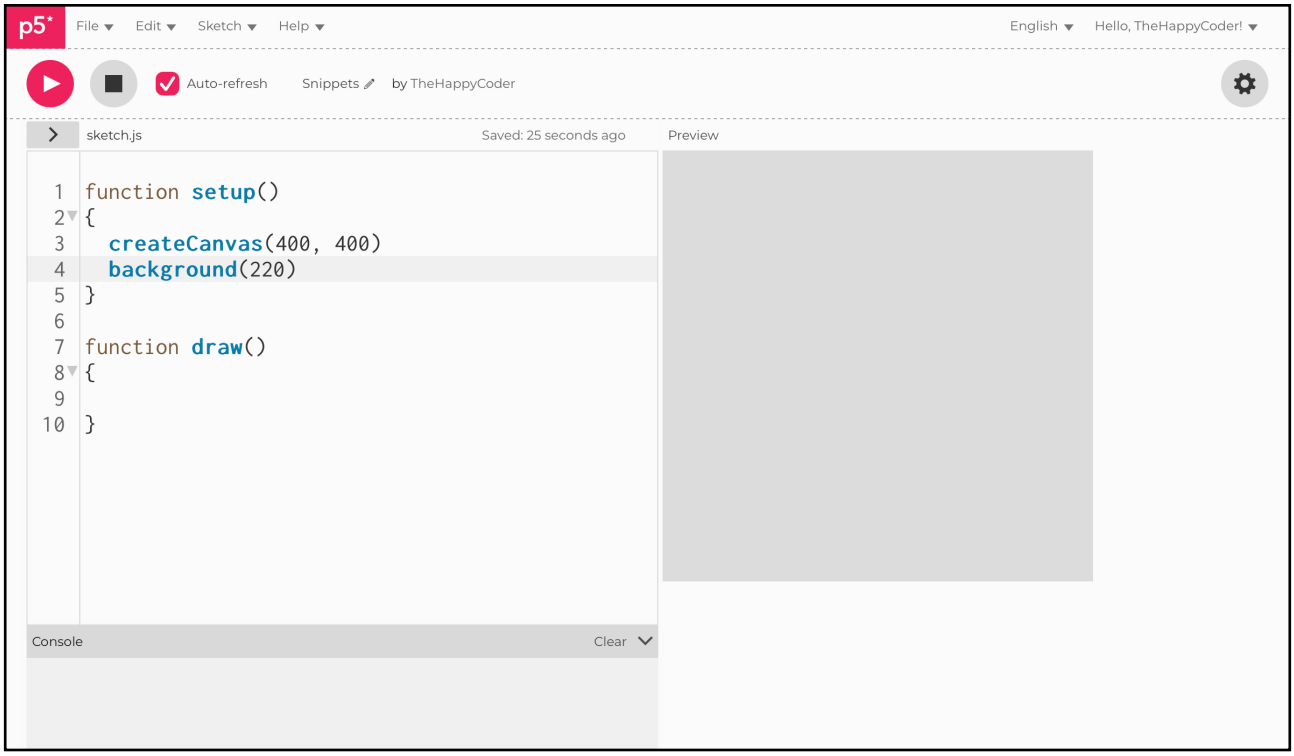
Challenges

1. Change the `220` to `0` or some other number.
2. Change the dimensions of the canvas from `(400, 400)` to `(100, 600)`.

Code Explanation

`background(220)`	Gives the background a colour

Figure A1.4


p5*
File
Edit
Sketch
Help
English
Hello, TheHappyCoder!
Auto-refresh
Snippets
by TheHappyCoder
sketch.js
Saved: 25 seconds ago
Preview
1 function setup()
2 {
3 createCanvas(400, 400)
4 background(220)
5 }
6
7 function draw()
8 {
9
10 }
Console
Clear

Sketch A1.5 RGB colours

Although most of the time a grey background is all you need, you might want to have a brighter or different colour for your finished product. Instead of one argument for the background, we can have three arguments instead. They are still between `0` and `255`, but they represent the amount of `red`, `green`, and `blue`. This means you can mix them to make lots of different colours. The default mode is `RGB`.

```
function setup()
{
  createCanvas(400, 400)
  background(200, 100, 10)
}

function draw()
{

}
```

Notes

A `background()` with three arguments will interpret them as `RGB`, `HSB`, or `HSL` colours depending on the current `colorMode()`. By default, colours are specified in `RGB` values. The example above gives us a nice orange colour. I will include much more information about colours in a much later section.

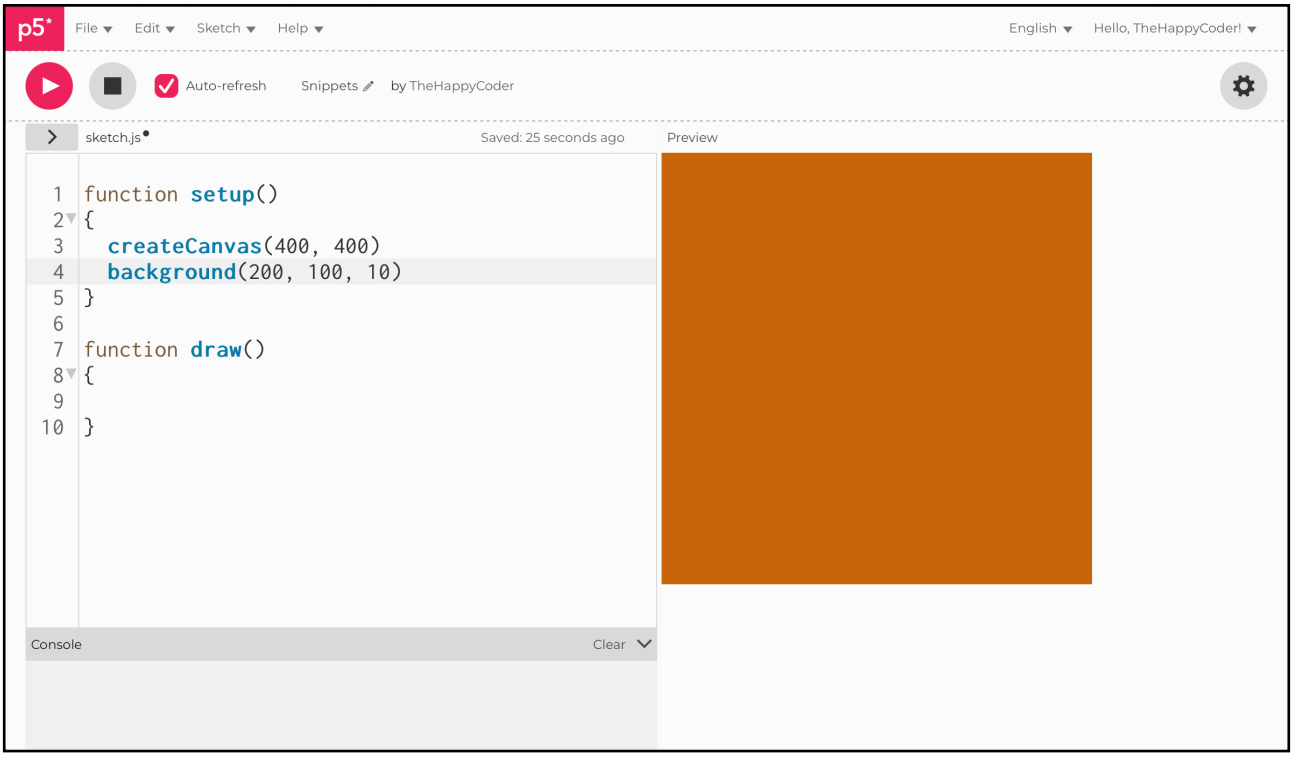
Challenge

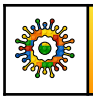
Play with the values to see what they do. If you just want red, then use `(255, 0, 0)`, if you want green, then use `(0, 255, 0)`, and so on.

Code Explanation

`background(200, 100, 10)`	This is how you can create colours with rgb values.

Figure A1.5


p5*
File
Edit
Sketch
Help
English
Hello, TheHappyCoder!
Auto-refresh
Snippets
by TheHappyCoder
sketch.js
Saved: 25 seconds ago
Preview
1 function setup()
2 {
3 createCanvas(400, 400)
4 background(200, 100, 10)
5 }
6
7 function draw()
8 {
9
10 }
Console
Clear



Sketch A1.6 a circle()

We can draw a circle using the `circle()` function. It needs three arguments. The first argument is the distance from the left-hand side. The second argument is the distance from the top. The third argument is the diameter of the circle.

```
function setup()
{
  createCanvas(400, 400)
  background(200, 100, 10)
}

function draw()
{
  circle(100, 300, 50)
}
```

🗒️ Notes

This draws a circle of a diameter of 50 pixels. It is drawn 100 pixels from the left-hand side and 300 pixels from the top.

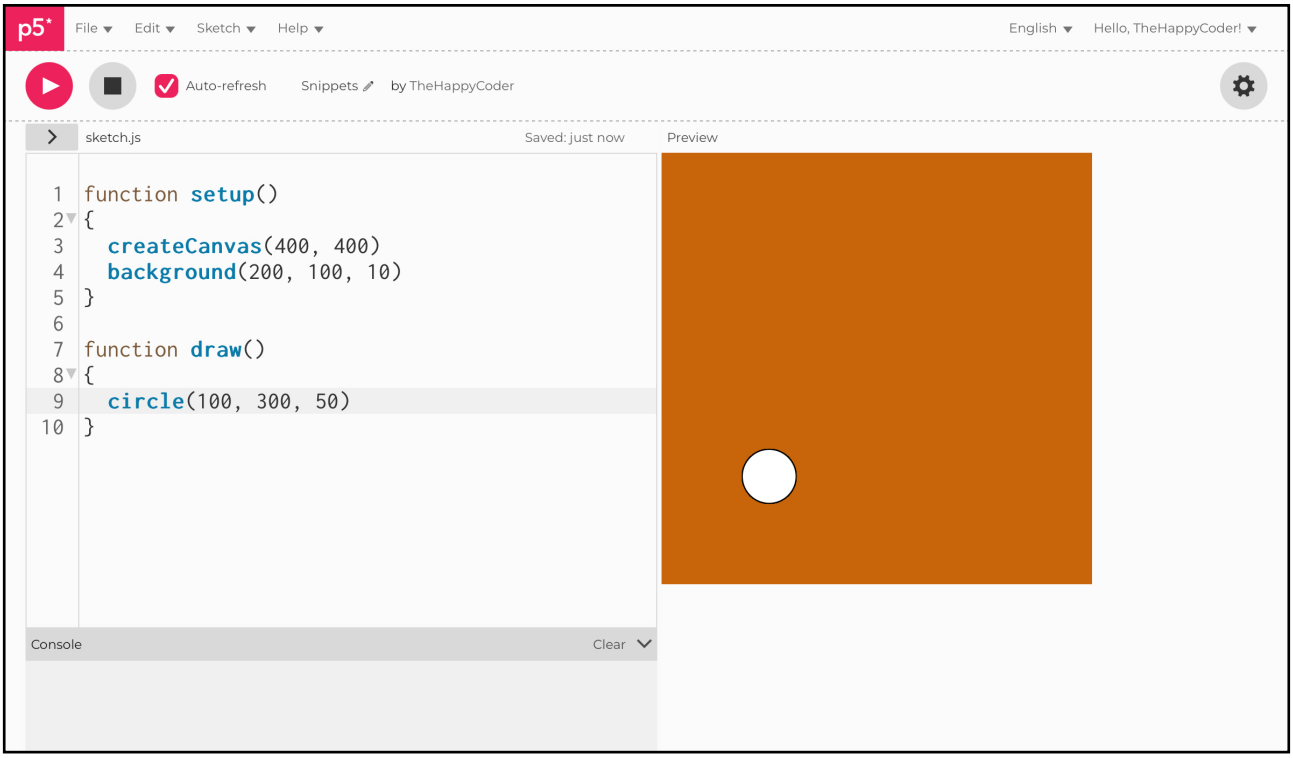
🌻 Challenge

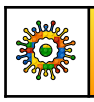
Try different diameters and co-ordinates.

🛠️ Code Explanation

`circle(100, 300, 50)`	The circle has three arguments, the first is the x co-ordinate (100), the second is the y co-ordinate (300) and the third is the diameter (50)

Figure A1.6


p5*
File
Edit
Sketch
Help
English
Hello, TheHappyCoder!
Auto-refresh
Snippets
by TheHappyCoder
sketch.js
Saved: just now
Preview
1 function setup()
2 {
3 createCanvas(400, 400)
4 background(200, 100, 10)
5 }
6
7 function draw()
8 {
9 circle(100, 300, 50)
10 }
Console
Clear



Sketch A1.7 fill()

The `fill()` function allows you to fill the shape with a colour in the same way as the background, so it can take one or three arguments.

```
function setup()
{
  createCanvas(400, 400)
  background(200, 100, 10)
}

function draw()
{
  fill(250, 250, 10)
  circle(100, 300, 50)
}
```

🗒️ Notes

We have created a nice yellow circle.

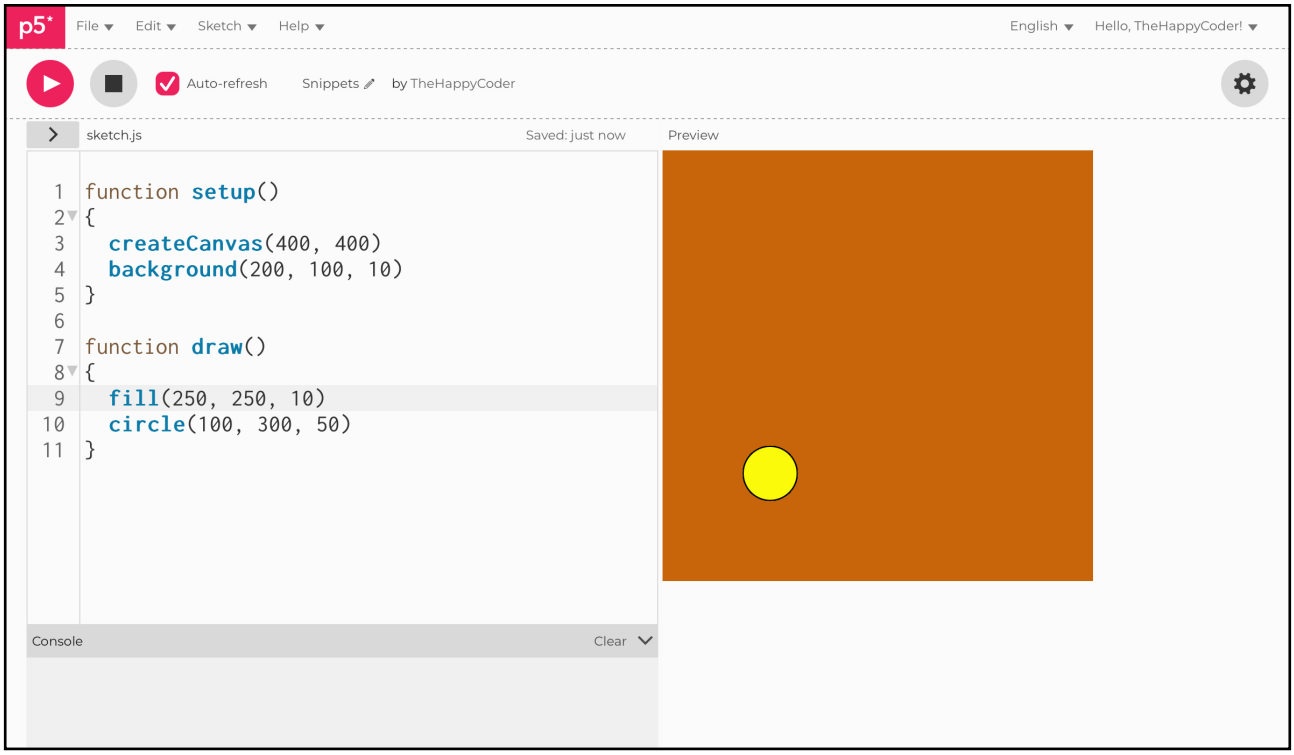
🌻 Challenge

Try some other colours or greyscale.

🛠️ Code Explanation

`fill(250, 250, 10)`	The fill() function takes one or three arguments for the colour

Figure A1.7


p5*
File
Edit
Sketch
Help
English
Hello, TheHappyCoder!
Auto-refresh
Snippets
by TheHappyCoder
sketch.js
Saved: just now
Preview
1 function setup()
2 {
3 createCanvas(400, 400)
4 background(200, 100, 10)
5 }
6
7 function draw()
8 {
9 fill(250, 250, 10)
10 circle(100, 300, 50)
11 }
Console
Clear

Sketch A1.8 strokeWeight()

We can change the thickness of the line around the circle with the `strokeWeight()` function. If you don't specify it, the default is `1`.

```
function setup()
{
  createCanvas(400, 400)
  background(200, 100, 10)
}

function draw()
{
  fill(250, 250, 10)
  strokeWeight(5)
  circle(100, 300, 50)
}
```

🗒️ Notes

Sets the width of the stroke used for points, lines, and the outlines of shapes.

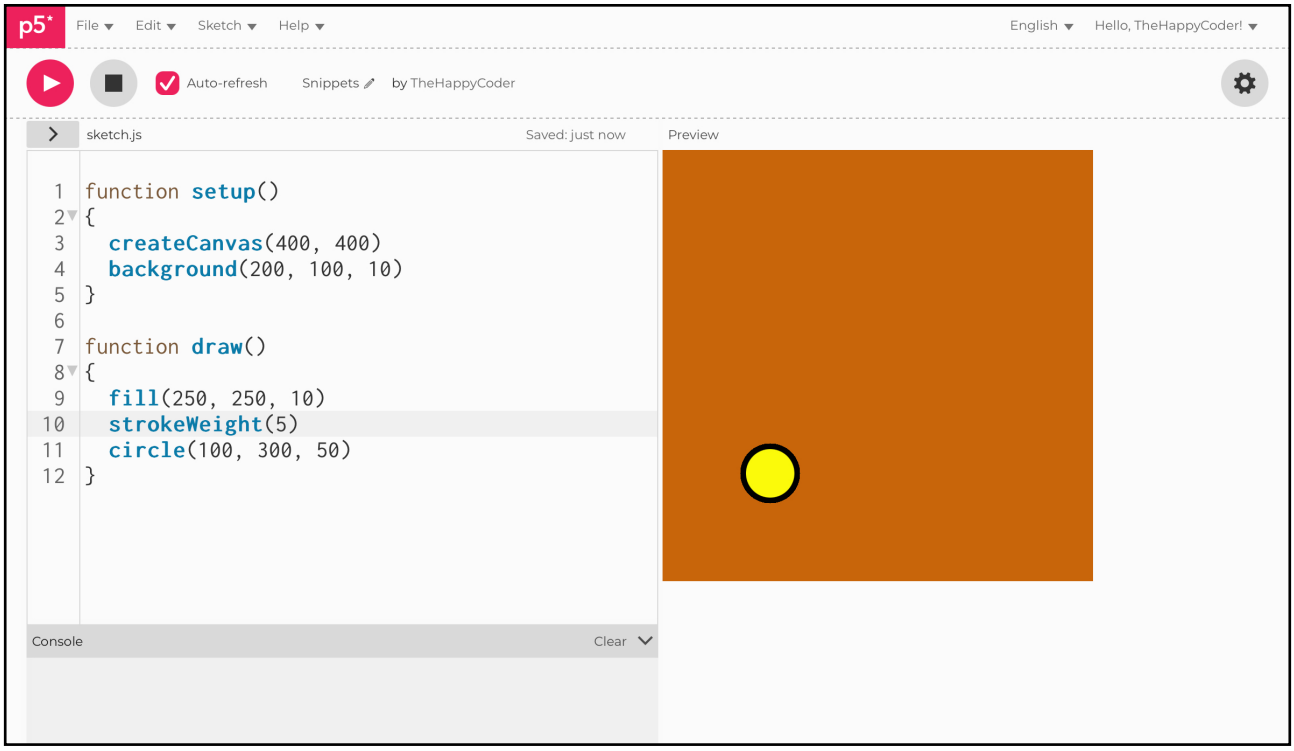
🌻 Challenge

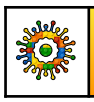
Try different stroke widths.

🛠️ Code Explanation

`strokeWeight(5)`	We can give the thickness of a line a stronger weight, the default is 1

Figure A1.8


p5*
File
Edit
Sketch
Help
English
Hello, TheHappyCoder!
Auto-refresh
Snippets
by TheHappyCoder
sketch.js
Saved: just now
Preview
function setup()
{
createCanvas(400, 400)
background(200, 100, 10)
}
function draw()
{
fill(250, 250, 10)
strokeWeight(5)
circle(100, 300, 50)
}
Console
Clear



Sketch A1.9 stroke()

We can change the colour of the line with the `stroke()` function.

```
function setup()
{
  createCanvas(400, 400)
  background(200, 100, 10)
}

function draw()
{
  fill(250, 250, 10)
  strokeWeight(5)
  stroke(0, 0, 255)
  circle(100, 300, 50)
}
```

🗒️ Notes

We get a nice blue line round the circle.

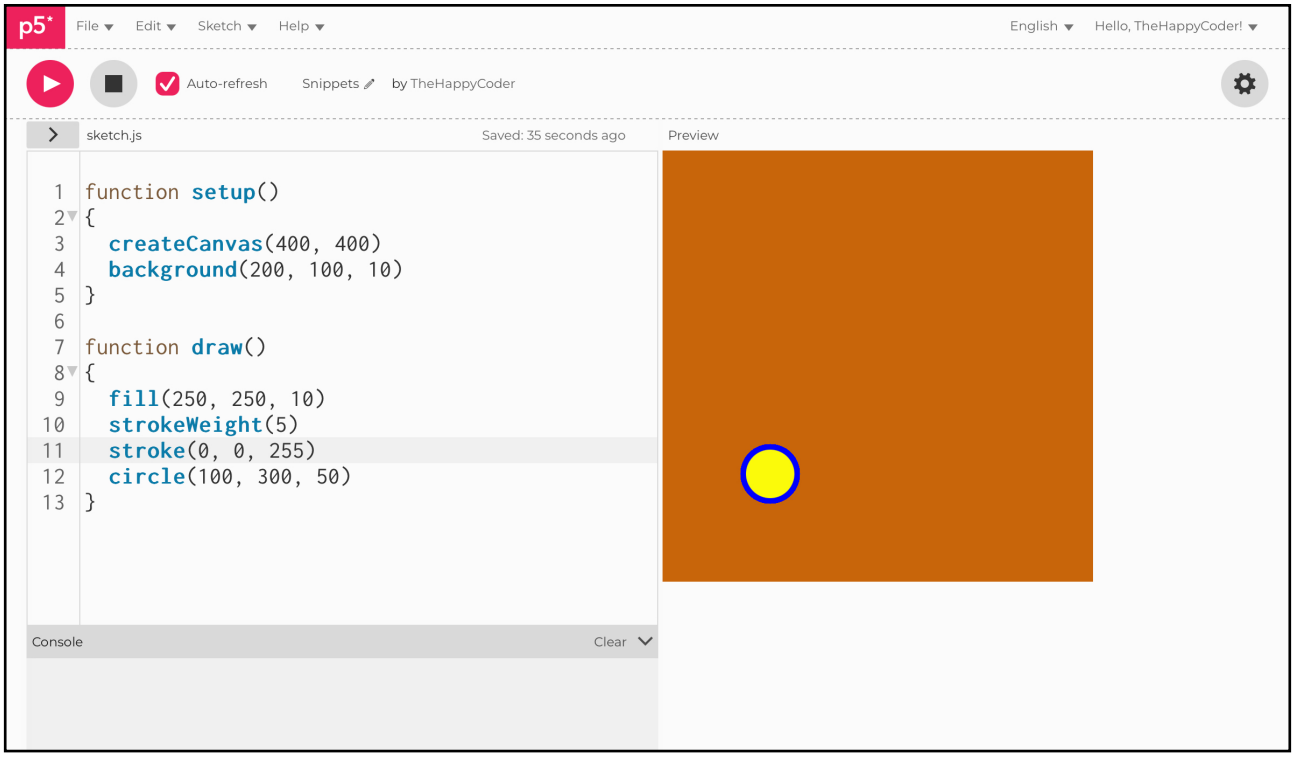
🌻 Challenge

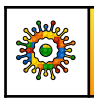
Try different `stroke()` colours, use the RGB values.

🛠️ Code Explanation

`stroke(0, 0, 255)`	Giving the line an rgb value

Figure A1.9


p5*
File
Edit
Sketch
Help
English
Hello, TheHappyCoder!
Auto-refresh
Snippets
by TheHappyCoder
sketch.js
Saved: 35 seconds ago
Preview
function setup()
{
createCanvas(400, 400)
background(200, 100, 10)
}
function draw()
{
fill(250, 250, 10)
strokeWeight(5)
stroke(0, 0, 255)
circle(100, 300, 50)
}
Console
Clear



Sketch A1.10 removing the line

Replacing the `stroke()` function with the `noStroke()` function removes the line altogether.

```
function setup()
{
  createCanvas(400, 400)
  background(200, 100, 10)
}

function draw()
{
  fill(250, 250, 10)
  strokeWeight(5)
  noStroke()
  circle(100, 300, 50)
}
```

🗒️ Notes

Disables drawing points, lines, and the outlines of shapes. Calling `noStroke()` is the same as making the stroke completely transparent, as in `stroke(0, 0)`. If both `noStroke()` and `noFill()` are called, nothing will be drawn to the screen.

🌻 Challenge

Try `stroke(0, 0)` and `noFill()`.

🛠️ Code Explanation

`noStroke()`	The noStroke() just simply removes any line round a shape

Figure A1.10


p5*
File
Edit
Sketch
Help
English
Hello, TheHappyCoder!
Auto-refresh
Snippets
by TheHappyCoder
sketch.js
Saved: 4 minutes ago
Preview
1 function setup()
2 {
3 createCanvas(400, 400)
4 background(200, 100, 10)
5 }
6
7 function draw()
8 {
9 fill(250, 250, 10)
10 strokeWeight(5)
11 noStroke()
12 circle(100, 300, 50)
13 }
Console
Clear

Introduction to variables

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

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

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

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

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

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

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

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

Sketch A1.11 let there be variables

Variables are a critical part of coding. To create a variable, you need to inform the computer programme. In p5.js (JavaScript), we use the word `let` before we name the variable. When we give the variable a name, it can be a single letter, a series of letters and numbers, or words. They cannot start with a number. To demonstrate this, we will replace the co-ordinates of the circle with variables `x` and `y`. At the same time, we will create a variable for the `diameter`.

```
let x = 100
let y = 300
let diameter = 50

function setup()
{
  createCanvas(400, 400)
  background(200, 100, 10)
}

function draw()
{
  fill(250, 250, 10)
  strokeWeight(5)
  noStroke()
  circle(x, y, diameter)
}
```

Notes

The circle is exactly the same as before. Because we called it a `let` variable, it means we have the option to change it later on in the code. We don't even need to give it an initial value, but it is a good idea to do so. When you create a variable, it is called `declaring` a variable; giving it a value is called `initialising` a variable.

A variable is a sort of container to hold a value. For example, a variable might contain the x-co-ordinate of a circle as a number or the name of someone as a string, where a string is a letter, series of letters and numbers, or words.

The scope of a variable depends on where you declare it. If you declare it in a function, it can only be used in that function; that is called a local scope. A global scope is where you declare a variable at the very start of the sketch outside any functions.

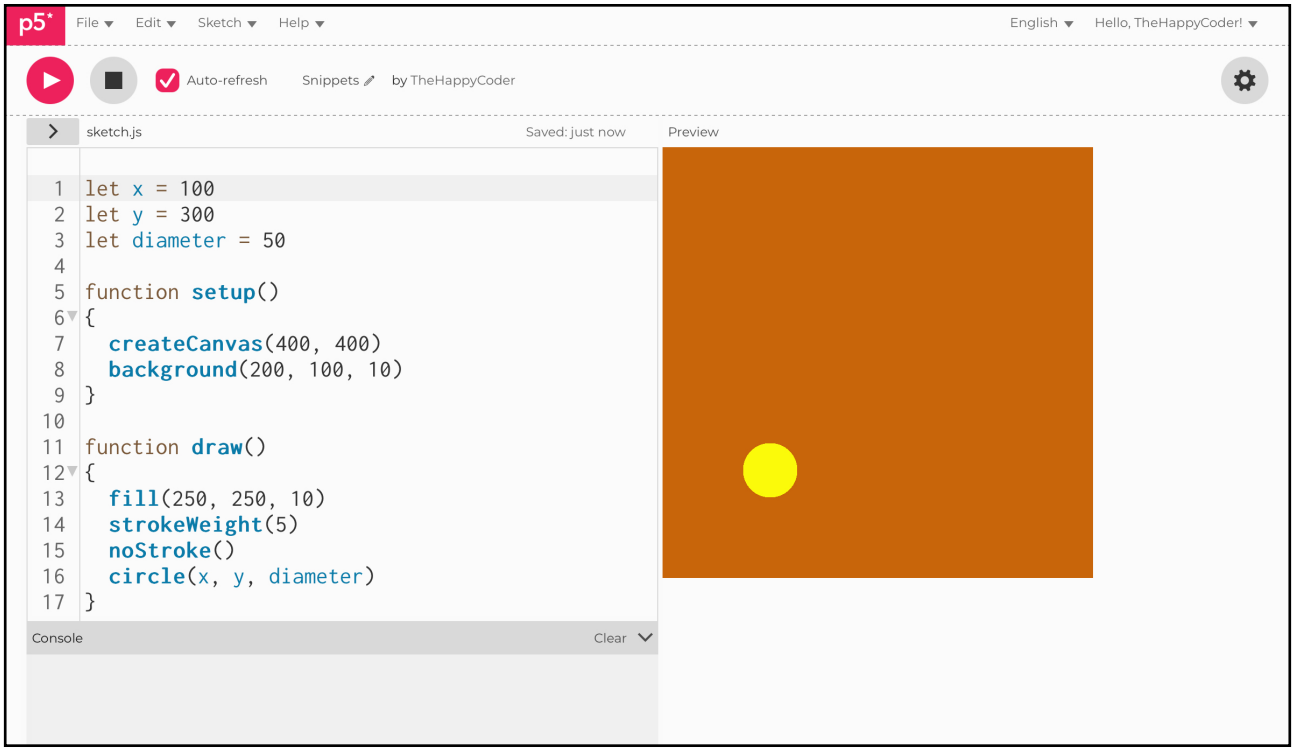
🌻 Challenge

1. Alter the x, y, and diameter variables.
2. Change the names of the variables.

🛠️ Code Explanation

`let x = 100`	We create a variable called x and initialise it with a value of 100
`let y = 300`	We create a variable called y and initialise it with a value of 300
`let diameter = 50`	We create a variable called diameter and initialise it with a value of 50

Figure A1.11


p5*
File
Edit
Sketch
Help
English
Hello, TheHappyCoder!
Auto-refresh
Snippets
by TheHappyCoder
sketch.js
Saved: just now
Preview
let x = 100
let y = 300
let diameter = 50
function setup()
{
createCanvas(400, 400)
background(200, 100, 10)
}
function draw()
{
fill(250, 250, 10)
strokeWeight(5)
noStroke()
circle(x, y, diameter)
}
Console
Clear

Sketch A1.12 width and height

We can use other built-in variables. Two of these are called `width` and `height`. They know the width of the canvas and the height of the canvas. In this demonstration, we will replace the x and y values from the `x` and `y` variables with the width and height variable names. This is going to put the circle in the middle of the canvas, which is half the width and half the height, hence `width/2` and `height/2`.

```
let x = 100
let y = 300
let diameter = 50

function setup()
{
  createCanvas(400, 400)
  background(200, 100, 10)
}

function draw()
{
  fill(250, 250, 10)
  strokeWeight(5)
  noStroke()
  circle(width/2, height/2, diameter)
}
```

Notes

This puts the circle bang in the middle.

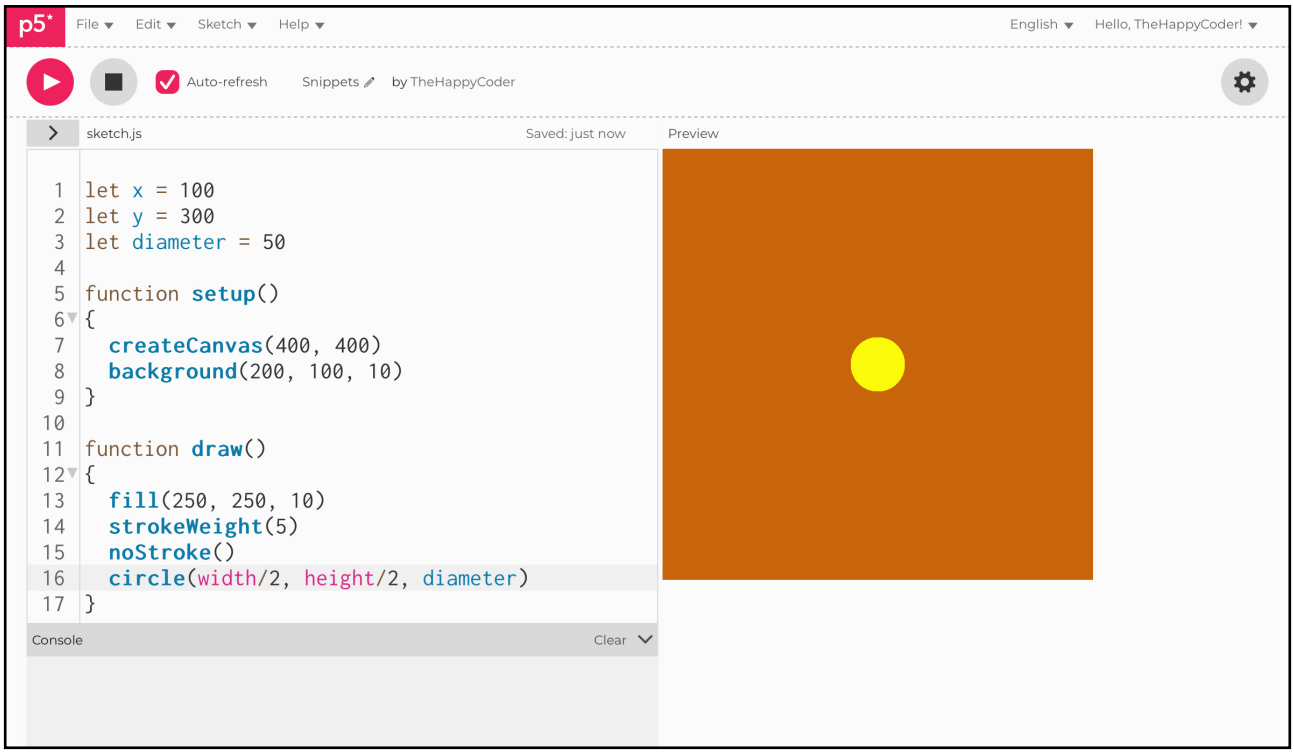
Challenges

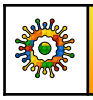
1. Change the dimensions of the canvas.
2. Change the position to `width/4` and `height/8`.
3. Change the diameter to `width/2`.

🛠️ Code Explanation

`circle(width/2, height/2, diameter)`	This gives us an x position of half the width and a y position of half the height.

Figure A1.12


p5*
File
Edit
Sketch
Help
English
Hello, TheHappyCoder!
Auto-refresh
Snippets
by TheHappyCoder
sketch.js
Saved: just now
Preview
let x = 100
let y = 300
let diameter = 50
function setup()
{
createCanvas(400, 400)
background(200, 100, 10)
}
function draw()
{
fill(250, 250, 10)
strokeWeight(5)
noStroke()
circle(width/2, height/2, diameter)
}
Console
Clear



Sketch A1.13 mouseX and mouseY

Another useful built-in variable is `mouseX` and `mouseY`. They are the co-ordinates of the mouse pointer. We can move the centre of the circle to where the mouse currently is. The co-ordinates are taken from the top-left-hand corner.

```
let x = 100
let y = 300
let diameter = 50

function setup()
{
  createCanvas(400, 400)
  background(200, 100, 10)
}

function draw()
{
  fill(250, 250, 10)
  strokeWeight(5)
  noStroke()
  circle(mouseX, mouseY, diameter)
}
```

Notes

The circle will follow the mouse around the canvas. You will notice that it draws over the canvas as if with a paintbrush. This is because we draw the background once in `setup()` and the circle is drawn repeatedly over the top of a fixed canvas. If you put the background in the `draw()` function, it draws the background on every iteration. This highlights the difference between `setup()` and `draw()`.

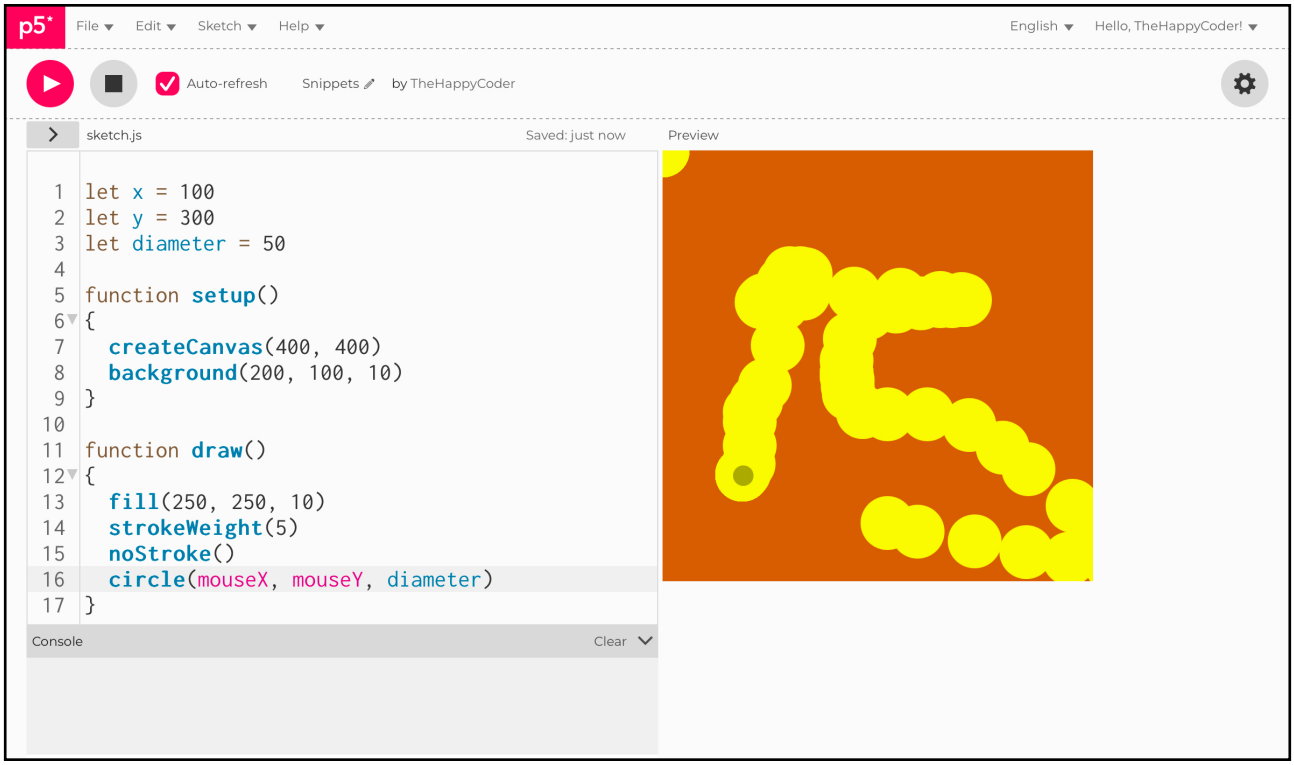
🌻 Challenge

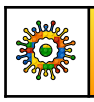
Change the diameter to mouseX or mouseY.

🛠️ Code Explanation

`circle(mouseX, mouseY, diameter)`	mouseX is the x co-ordinate of the mouse pointer and mouseY is the y co-ordinate of the mouse pointer

Figure A1.13


p5*
File
Edit
Sketch
Help
English
Hello, TheHappyCoder!
Auto-refresh
Snippets
by TheHappyCoder
sketch.js
Saved: just now
Preview
let x = 100
let y = 300
let diameter = 50
function setup()
{
createCanvas(400, 400)
background(200, 100, 10)
}
function draw()
{
fill(250, 250, 10)
strokeWeight(5)
noStroke()
circle(mouseX, mouseY, diameter)
}
Console
Clear



Sketch A1.14 moving the background

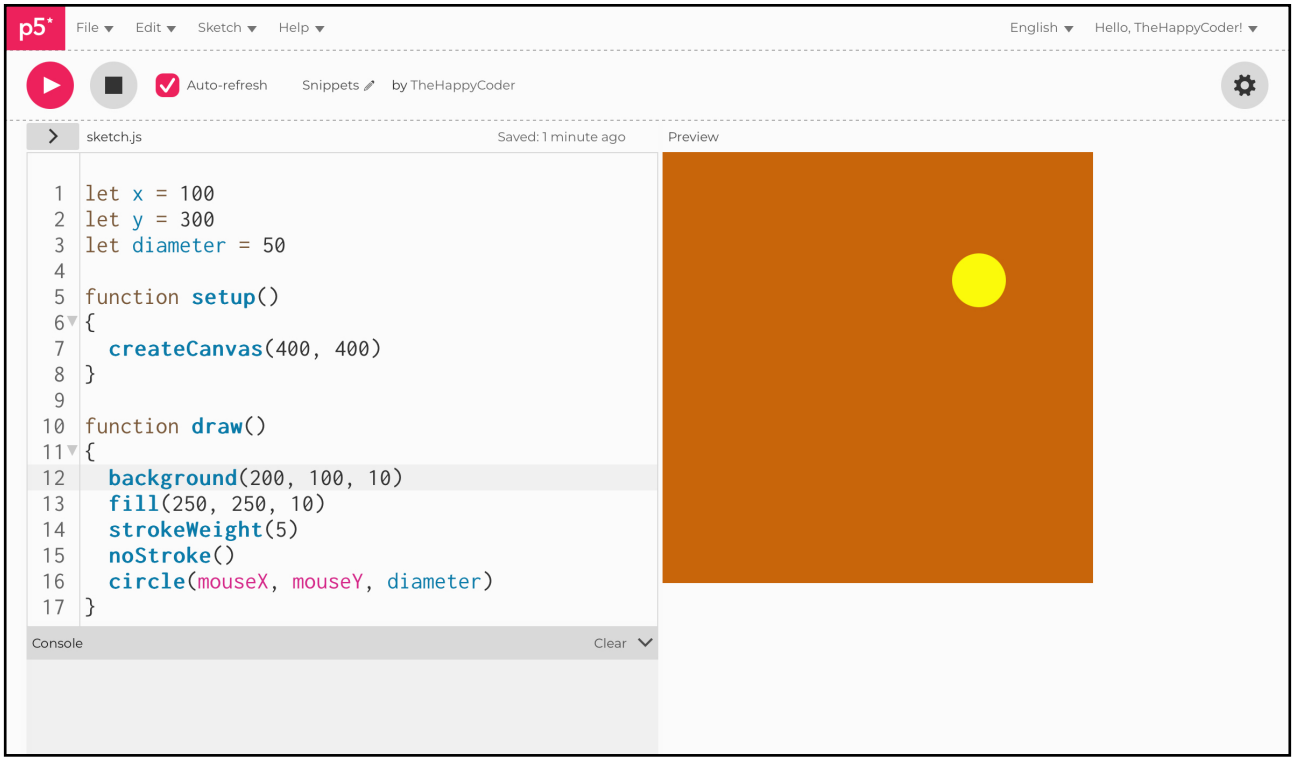
❗ remove background(200, 100, 10) from the setup() function.
Now put `background(200, 100, 10)` in the `draw()` function instead. You will see that it draws the background and the circle; it does this continuously.

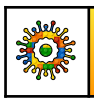
```
let x = 100
let y = 300
let diameter = 50

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

function draw()
{
  background(200, 100, 10)
  fill(250, 250, 10)
  strokeWeight(5)
  noStroke()
  circle(mouseX, mouseY, diameter)
}
```

🗒️ Notes

When you move the mouse, the circle follows it.

Figure A1.14


p5*
File
Edit
Sketch
Help
English
Hello, TheHappyCoder!
Auto-refresh
Snippets
by TheHappyCoder
sketch.js
Saved: 1 minute ago
Preview
let x = 100
let y = 300
let diameter = 50
function setup()
{
createCanvas(400, 400)
}
function draw()
{
background(200, 100, 10)
fill(250, 250, 10)
strokeWeight(5)
noStroke()
circle(mouseX, mouseY, diameter)
}
Console
Clear



Sketch A1.15 changing the values

❗ Remove the `strokeWeight()` and `noStroke()` functions.
In the `setup()` function, we are going to assign new values to the `x`, `y`, and `diameter` variables.

```
let x = 100
let y = 300
let diameter = 50

function setup()
{
  createCanvas(400, 400)
  x = 250
  y = 75
  diameter = 100
}

function draw()
{
  background(200, 100, 10)
  fill(250, 250, 10)
  circle(x, y, diameter)
}
```

🗒️ Notes

We have a larger circle in a completely new position.

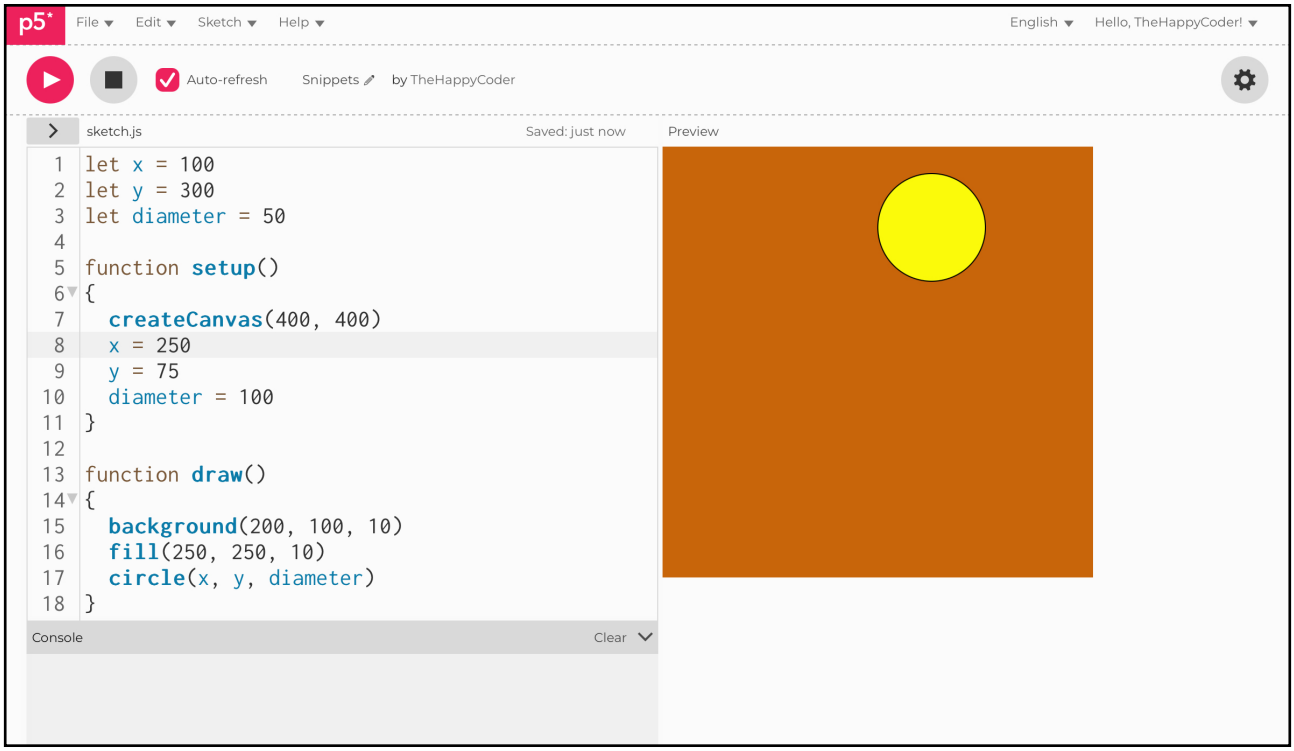
🌻 Challenge

Try other values.

🛠️ Code Explanation

`x = 250`	Re-assigned the value of x from 100 to 250
`y = 75`	Re-assigned the value of y from 300 to 75
`diameter = 100`	Re-assigned the diameter from 50 to 100

Figure A1.15


p5*
File
Edit
Sketch
Help
English
Hello, TheHappyCoder!
Auto-refresh
Snippets
by TheHappyCoder
sketch.js
Saved: just now
Preview
let x = 100
let y = 300
let diameter = 50
function setup()
{
createCanvas(400, 400)
x = 250
y = 75
diameter = 100
}
function draw()
{
background(200, 100, 10)
fill(250, 250, 10)
circle(x, y, diameter)
}
Console
Clear

Symbols we use

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

中 Arithmetic Operators

These perform basic mathematical operations; they are essential for numerical computations.

中 Comparison Operators

These compare values and return a boolean result (true or false).

中 Logical Operators

These combine boolean values to produce a single boolean result.

中 Assignment Operators

These are used to assign values to variables.

中 Maths Functions

Built-in functions that perform more complex mathematical operations.

Arithmetic Operators

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

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

Comparison Operators

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

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

Logical Operators

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

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

Assignment Operators

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

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

Maths Functions

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

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

Sketch A1.16 const

Sometimes we don't want a variable to change; if that is the case, we use constants. This is so we don't accidentally change something. For this, we use `const` rather than `let`. Watch what happens when we change it from the previous sketch.

```
const x = 100
const y = 300
const diameter = 50

function setup()
{
  createCanvas(400, 400)
  x = 250
  y = 75
  diameter = 100
}

function draw()
{
  background(200, 100, 10)
  fill(250, 250, 10)
  circle(x, y, diameter)
}
```

🗒️ Notes

You get an error message when you try to alter it later in the code.

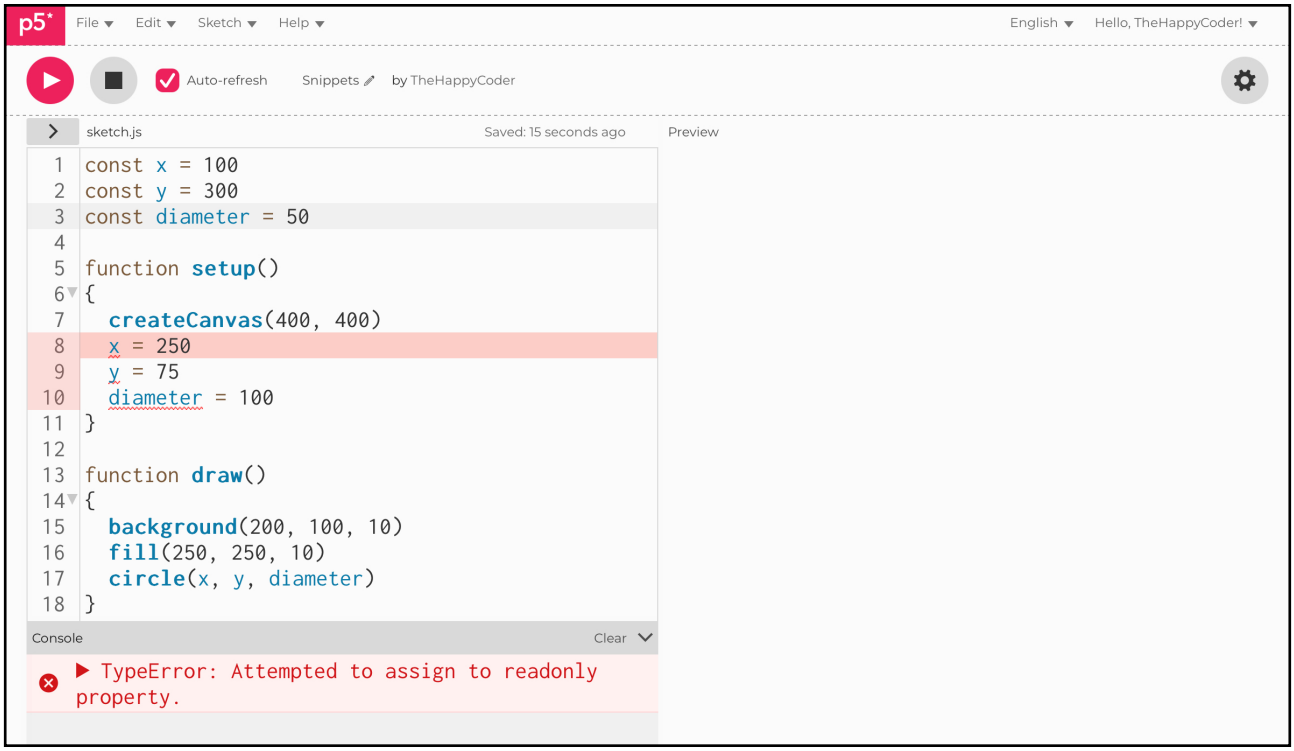
🌻 Challenge

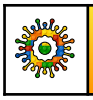
Remove the `x = 250`, `y = 75` and `diameter = 100` reference in the `setup()` function. See if that changes the outcome.

🛠️ Code Explanation

`const x = 100`	The x variable is now fixed
`const y = 300`	The y variable is now fixed
`const diameter = 50`	The diameter variable is now fixed

Figure A1.16


p5*
File
Edit
Sketch
Help
English
Hello, TheHappyCoder!
Auto-refresh
Snippets
by TheHappyCoder
sketch.js
Saved: 15 seconds ago
Preview
1 const x = 100
2 const y = 300
3 const diameter = 50
4
5 function setup()
6 {
7 createCanvas(400, 400)
8 x = 250
9 y = 75
10 diameter = 100
11 }
12
13 function draw()
14 {
15 background(200, 100, 10)
16 fill(250, 250, 10)
17 circle(x, y, diameter)
18 }
Console
Clear
▶ TypeError: Attempted to assign to readonly property.



Sketch A1.17 for loop

❗ Start a new sketch or delete much of the previous sketch. Putting the `background()` back into the `setup()` function.

If we want to draw lots of circles, we could write a line of code for each circle, but there is a better way using loops. For this, we can use what is called a `for()` loop. The `for()` loop has three parts to it. You declare and initialise a variable, in this case `x`, with a starting value, in this case `20`. You then set the conditional limit of the value, in this case `400`, at which point the loop stops. The third and final part of the loop determines how much you are increasing (or decreasing) the variable on each iteration, in this case, we are increasing it by `40`.

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

function draw()
{
  for (let x = 20; x < 400; x += 40)
  {
    circle(x, 200, 20)
  }
}
```

Notes

You will get a line of ten circles across the canvas. You will find that `for()` loops are used a lot; after a while, they become very intuitive. Also, note that here we have to use the semicolons.

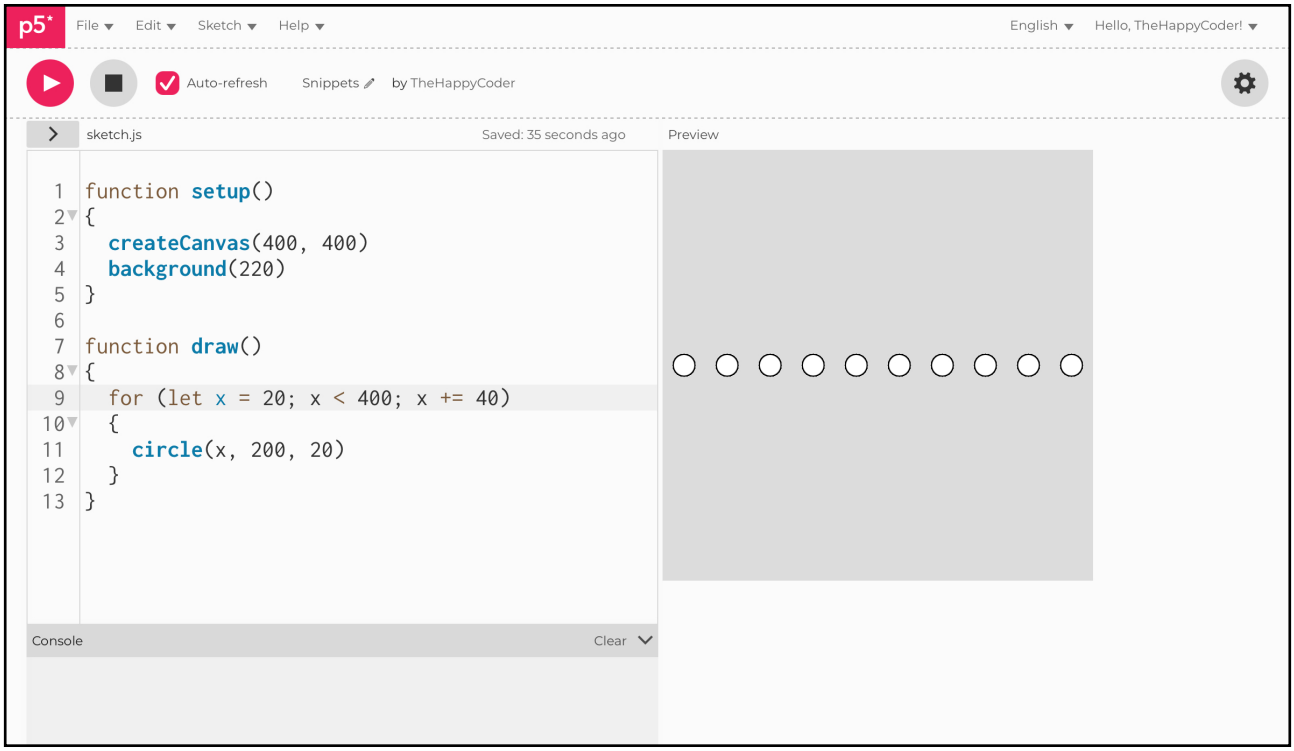
🌻 Challenge

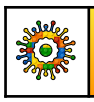
Alter the values in the `for()` loop and see what effect it has. You might inadvertently create an infinite loop, so I recommend that you pause (stop) the code running while you make changes.

🛠️ Code Explanation

`for ()`	The for() loop is continuous until a condition is met
`let x = 20;`	We define and initialise the variable being 20
`x < 400;`	We define the limit of the loop being 400
`x += 40`	We define the changes to the variable adding 40
`+=`	This is shorthand for x = x + 40

Figure A1.17


p5*
File
Edit
Sketch
Help
English
Hello, TheHappyCoder!
Auto-refresh
Snippets
by TheHappyCoder
sketch.js
Saved: 35 seconds ago
Preview
1 function setup()
2 {
3 createCanvas(400, 400)
4 background(220)
5 }
6
7 function draw()
8 {
9 for (let x = 20; x < 400; x += 40)
10 {
11 circle(x, 200, 20)
12 }
13 }
Console
Clear



Sketch A1.18 nested loop

We can have a loop within a loop. In this way, we can also loop around the `y` value. The `for()` loop for the `y` variable looks exactly the same because it is. Also, remember to put the `y` variable into the `circle()` function.

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

function draw()
{
  for (let x = 20; x < 400; x += 40)
  {
    for (let y = 20; y < 400; y += 40)
    {
      circle(x, y, 20)
    }
  }
}
```

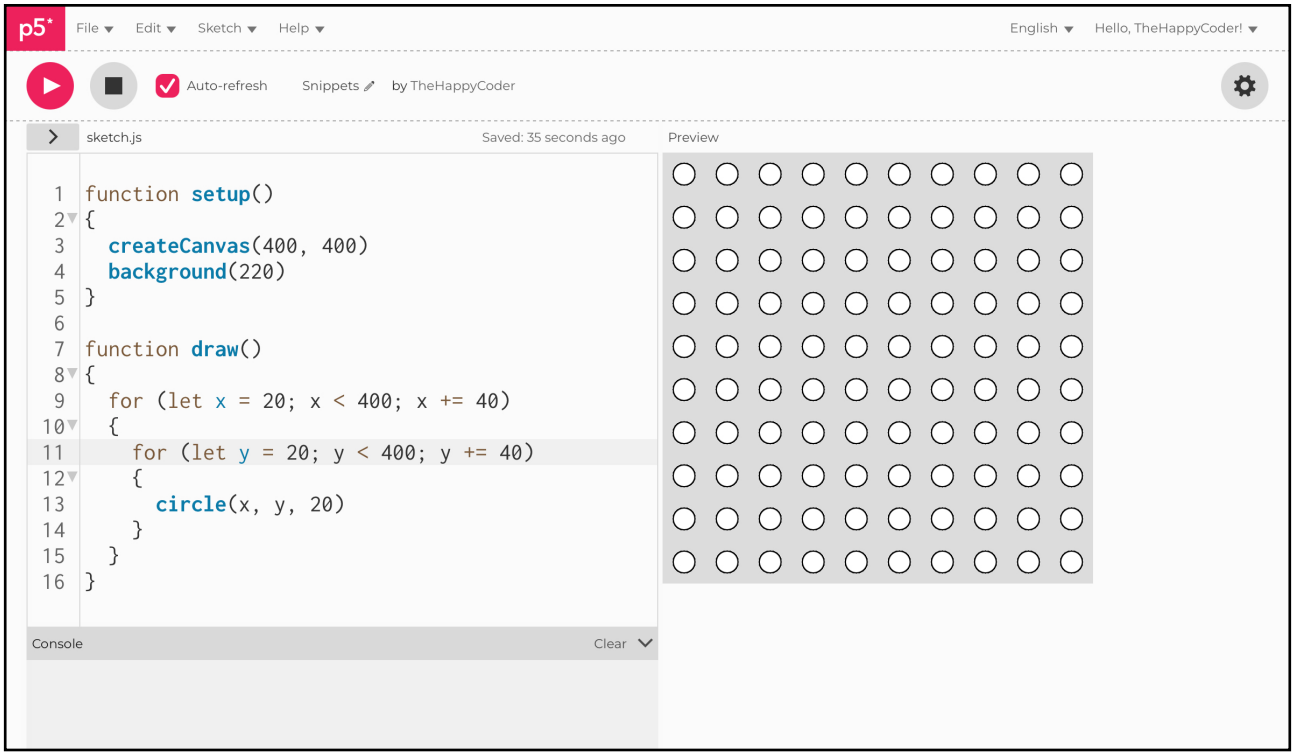
🗒️ Notes

We have now filled the canvas with circles.

🌻 Challenge

Have a third nested loop so that the diameter increases for each circle.

Figure A1.18


p5*
File
Edit
Sketch
Help
English
Hello, TheHappyCoder!
Auto-refresh
Snippets
by TheHappyCoder
sketch.js
Saved: 35 seconds ago
Preview
function setup()
{
createCanvas(400, 400)
background(220)
}
function draw()
{
for (let x = 20; x < 400; x += 40)
{
for (let y = 20; y < 400; y += 40)
{
circle(x, y, 20)
}
}
}
Console
Clear

Sketch A1.19 random()

❗ Remove the `for()` loops.

We can use a `random()` function when necessary to mix up the data. The `random()` function has two versions depending on how many arguments you have. If you have one argument, it will pick a random number between zero and that number; if you have two arguments, then it will pick a random number between those two numbers.

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

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

🗒️ Notes

You see that the `draw()` function draws each circle at a random position on the canvas.

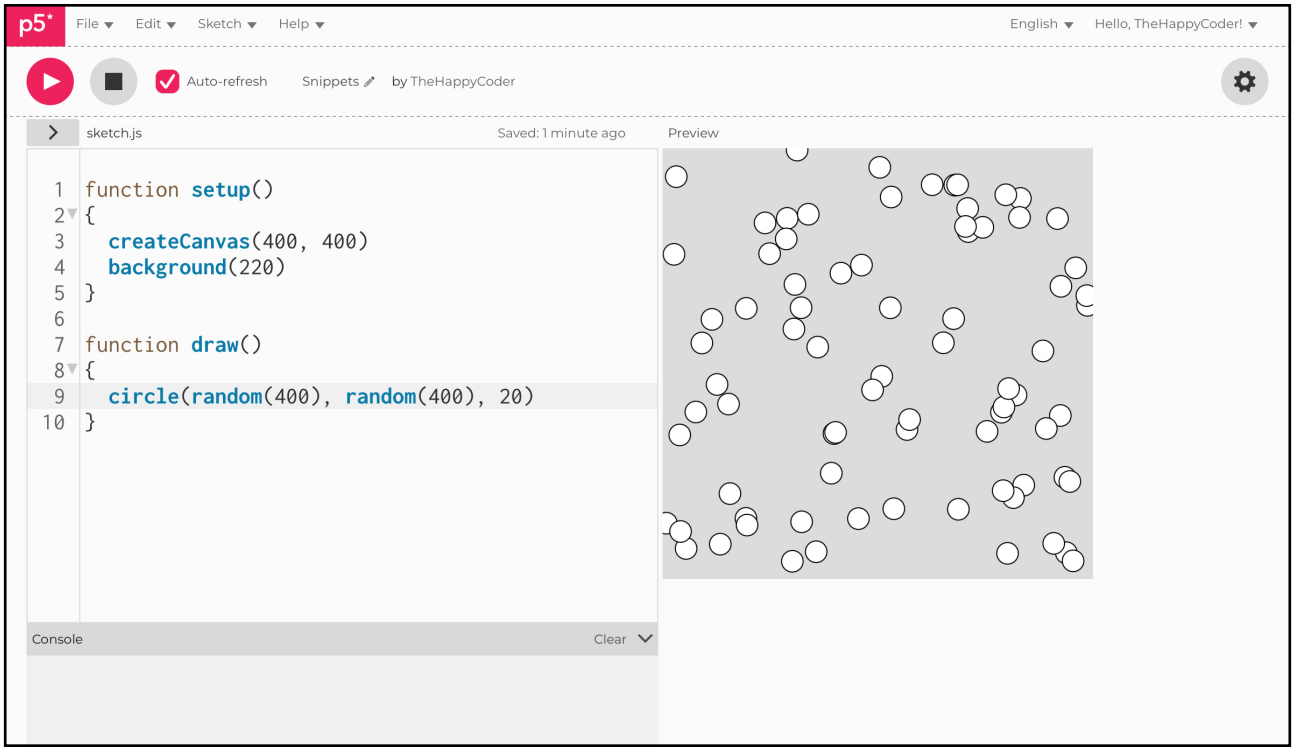
🌻 Challenge

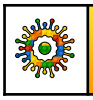
Change `random(400)` to `random(100, 300)` for both the `x` and `y` variables.

🛠️ Code Explanation

`random(400)`	Picks a random number between 0 and 400

Figure A1.19


p5*
File
Edit
Sketch
Help
English
Hello, TheHappyCoder!
Auto-refresh
Snippets
by TheHappyCoder
sketch.js
Saved: 1 minute ago
Preview
1 function setup()
2 {
3 createCanvas(400, 400)
4 background(220)
5 }
6
7 function draw()
8 {
9 circle(random(400), random(400), 20)
10 }
Console
Clear



Sketch A1.20 if() statement

Another useful function is the `if()` statement. Here, something is `true` until it is `false` (and vice versa). When a condition is met, then the process ends; in this case, we will draw `100` circles in random positions. We use a variable to keep count of the number of circles drawn, called `count` (obviously).

```
let count = 0

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

function draw()
{
  if(count <= 100)
  {
    circle(random(400), random(400), 20)
  }
  count++
}
```

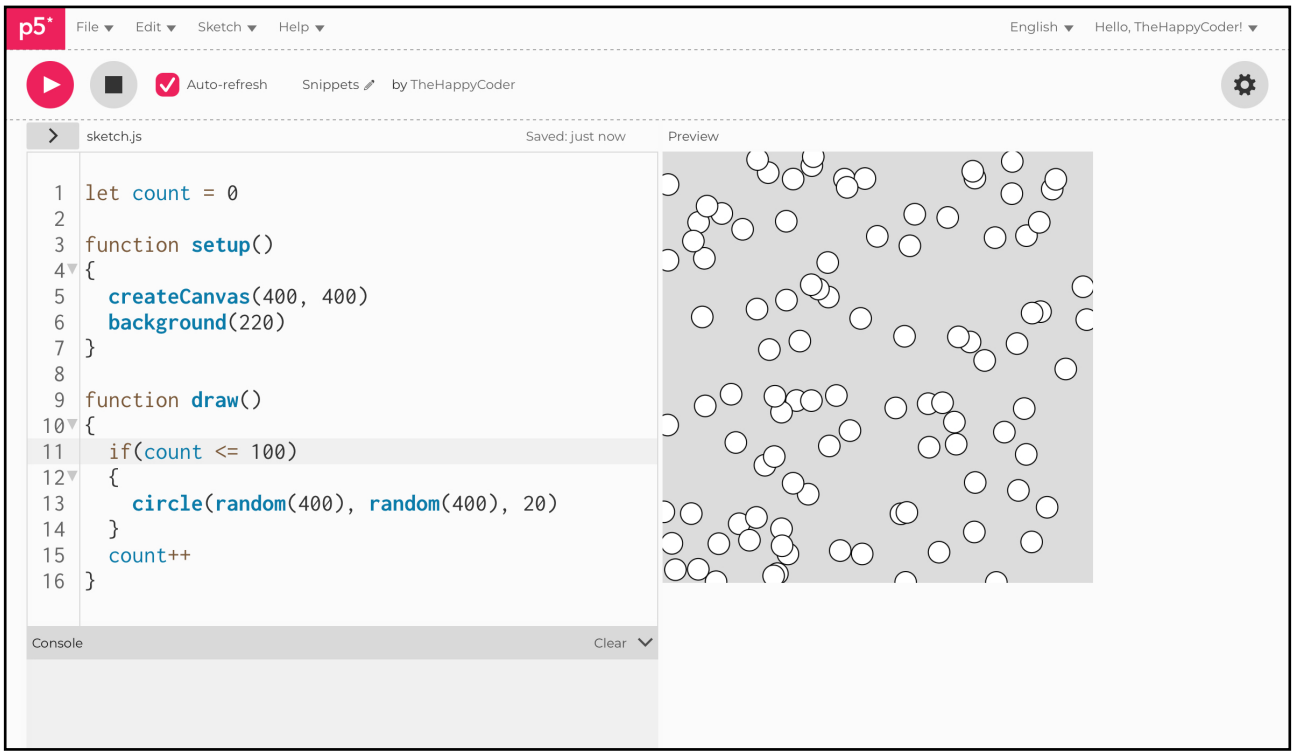
🗒️ Notes

The programme keeps drawing the circles while the condition is true (`count` is less than `100`). Once that condition is false (there are `100` circles), it stops.

🛠️ Code Explanation

`if(count <= 100)`	The if() statement where the condition is true if the count variable is less than 100, false if it is more than 100

Figure A1.20


p5*
File
Edit
Sketch
Help
English
Hello, TheHappyCoder!
Auto-refresh
Snippets
by TheHappyCoder
sketch.js
Saved: just now
Preview
let count = 0
function setup()
{
createCanvas(400, 400)
background(220)
}
function draw()
{
if(count <= 100)
{
circle(random(400), random(400), 20)
}
count++
}
Console
Clear

Sketch A1.21 drawing a line

We can draw a line as well as other shapes. The line has four arguments: the first two are the x and y co-ordinates of one end of the line (100, 100) and the other two are the x and y co-ordinates of the other end of the line (300, 300).

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

function draw()
{
  line(100, 100, 300, 300)
}
```

🗒️ Notes

It draws a simple line.

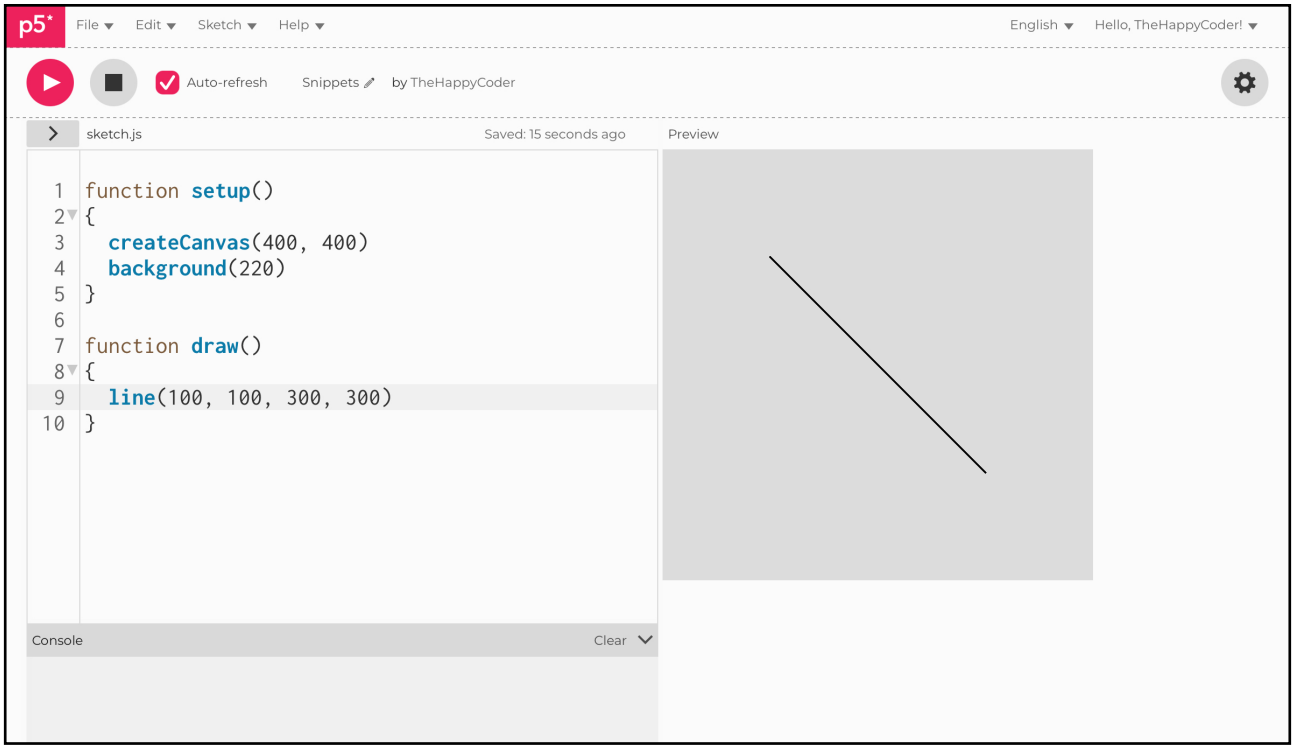
🌻 Challenge

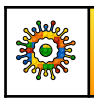
Draw several lines, all starting from the centre of the canvas.

🛠️ Code Explanation

`line(100, 100, 300, 300)`	Draws a line with co-ordinates (100, 100) and (300, 300) for each end

Figure A1.21


p5*
File
Edit
Sketch
Help
English
Hello, TheHappyCoder!
Auto-refresh
Snippets
by TheHappyCoder
sketch.js
Saved: 15 seconds ago
Preview
1 function setup()
2 {
3 createCanvas(400, 400)
4 background(220)
5 }
6
7 function draw()
8 {
9 line(100, 100, 300, 300)
10 }
Console
Clear



Sketch A1.22 stroke() and strokeWeight()

Adding a bit of colour and thickness.

```
function setup()
{
  createCanvas(400, 400)
  background(220)
  stroke(255, 0, 0)
  strokeWeight(5)
}

function draw()
{
  line(100, 100, 300, 300)
}
```

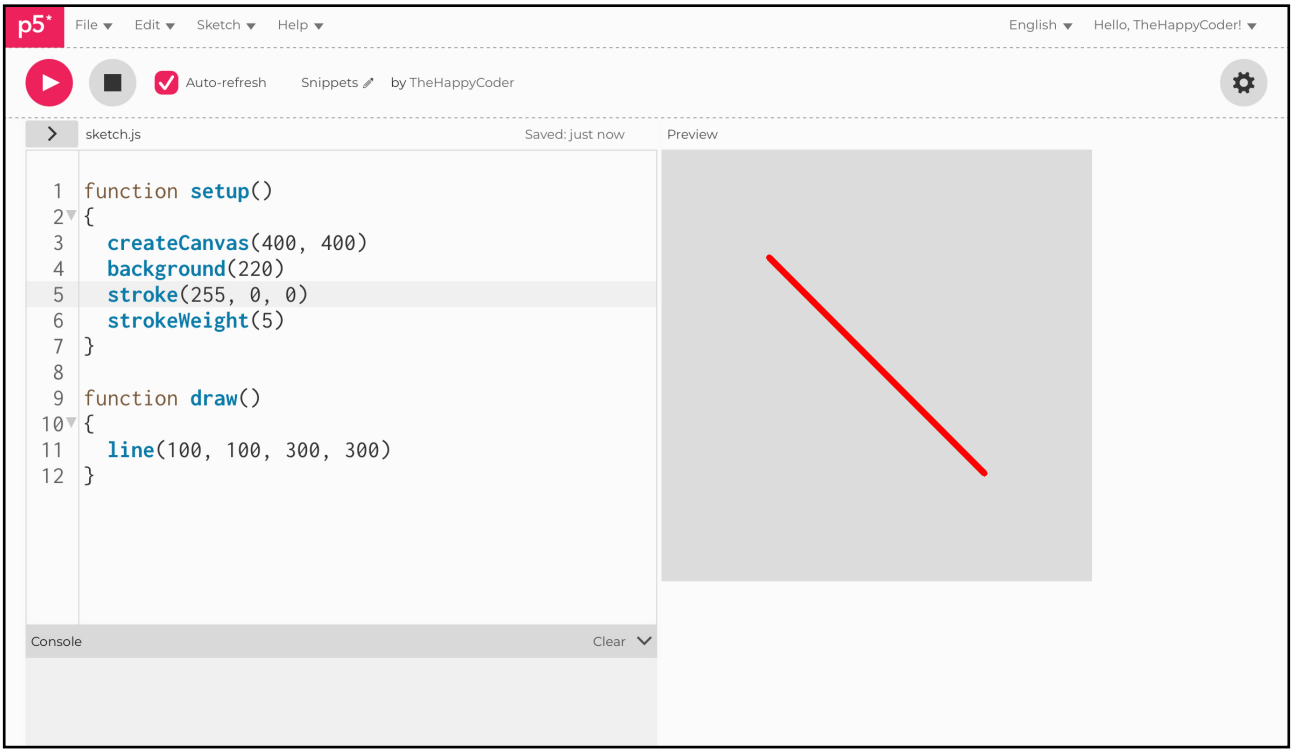
🗒️ Notes

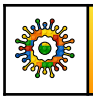
As with the circle, we can change the appearance of the line.

🌻 Challenge

Can you draw several lines in random positions, with random colours and strokes?

Figure A1.22


p5*
File
Edit
Sketch
Help
English
Hello, TheHappyCoder!
Auto-refresh
Snippets
by TheHappyCoder
sketch.js
Saved: just now
Preview
1 function setup()
2 {
3 createCanvas(400, 400)
4 background(220)
5 stroke(255, 0, 0)
6 strokeWeight(5)
7 }
8
9 function draw()
10 {
11 line(100, 100, 300, 300)
12 }
Console
Clear



Sketch A1.23 point()

The `point()` function draws a pixel at the co-ordinates `(100, 300)`. As this will be very small, we will change the `strokeWeight(10)`.

```
function setup()
{
  createCanvas(400, 400)
  background(220)
  stroke(255, 0, 0)
  strokeWeight(10)
}

function draw()
{
  point(100, 300)
}
```

🗒️ Notes

We have a `point` (or a pixel) at `(100, 300)`. The default is one pixel, which is too small to see easily; hence, to add a `strokeWeight()` of `10`.

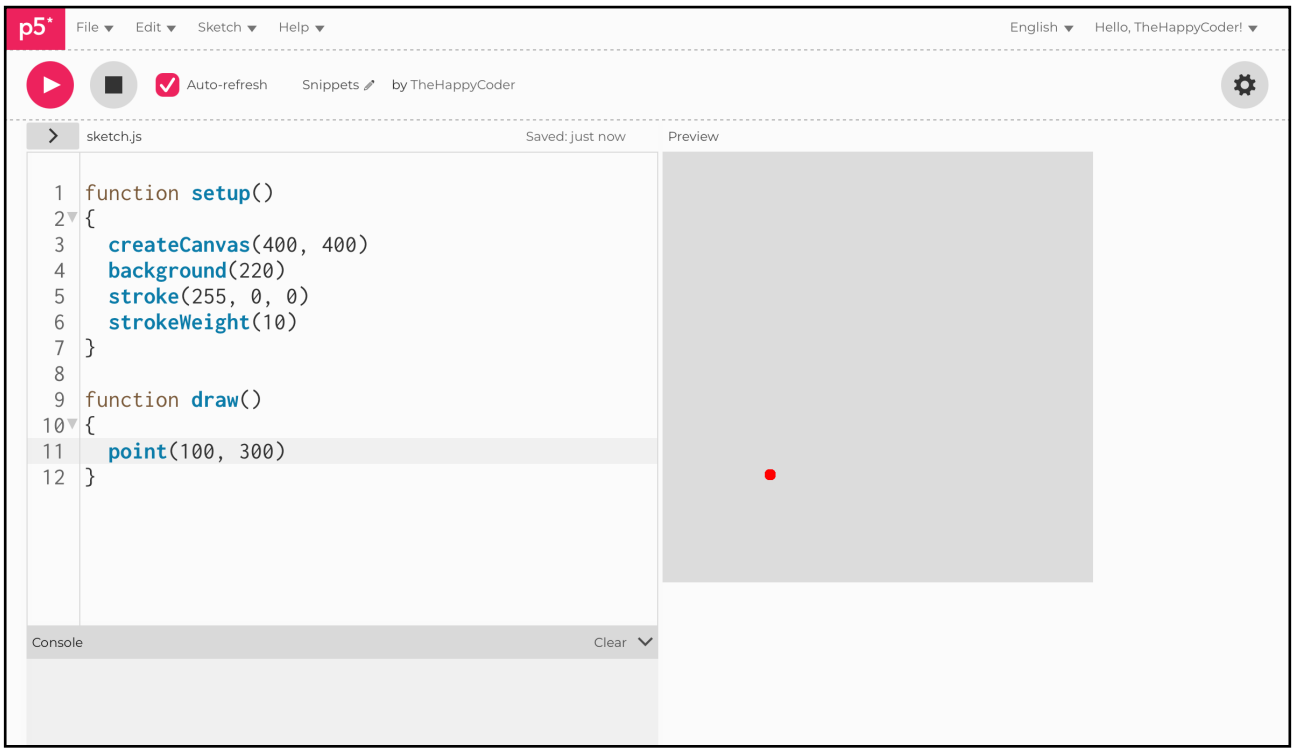
🌻 Challenge

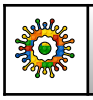
Could you draw a line of pixels with `strokeWeight(1)` using a `for()` loop or an `if()` statement?

🛠️ Code Explanation

`point(100, 300)`	Draws a pixel at those co-ordinates

Figure A1.23


p5*
File
Edit
Sketch
Help
English
Hello, TheHappyCoder!
Auto-refresh
Snippets
by TheHappyCoder
sketch.js
Saved: just now
Preview
1 function setup()
2 {
3 createCanvas(400, 400)
4 background(220)
5 stroke(255, 0, 0)
6 strokeWeight(10)
7 }
8
9 function draw()
10 {
11 point(100, 300)
12 }
Console
Clear



Functions

In p5.js, functions are at the core of everything. You can tell that there is a function because it is prefixed with that word; for instance, `function setup()` or `function draw()`, these are prebuilt functions. We can, however, create our own functions and call them whatever we want (there are some keywords we cannot use because they are already taken).

We can also call a function inside another function, as you will see later. If you create a function, it won't necessarily do anything until you call it. By that, I mean you have to reference it in another function, usually the `draw()` function, but it can be in any other active function.

In the next example, we will create a function called `thing()` and then call it from the `draw()` function. In the example following that one, we will create a function called `changeColour()`, this is only called when we click on the button we have created. The function just sits there until it is called.

Sketch A1.24 creating a function

We create a function called thing, which should draw a circle. But nothing happens.

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

function draw()
{
  background(220)
}

function thing()
{
  circle(200, 300, 100)
}
```

Notes

Although we have created a function, the computer reads it but will only action it when told to do so.

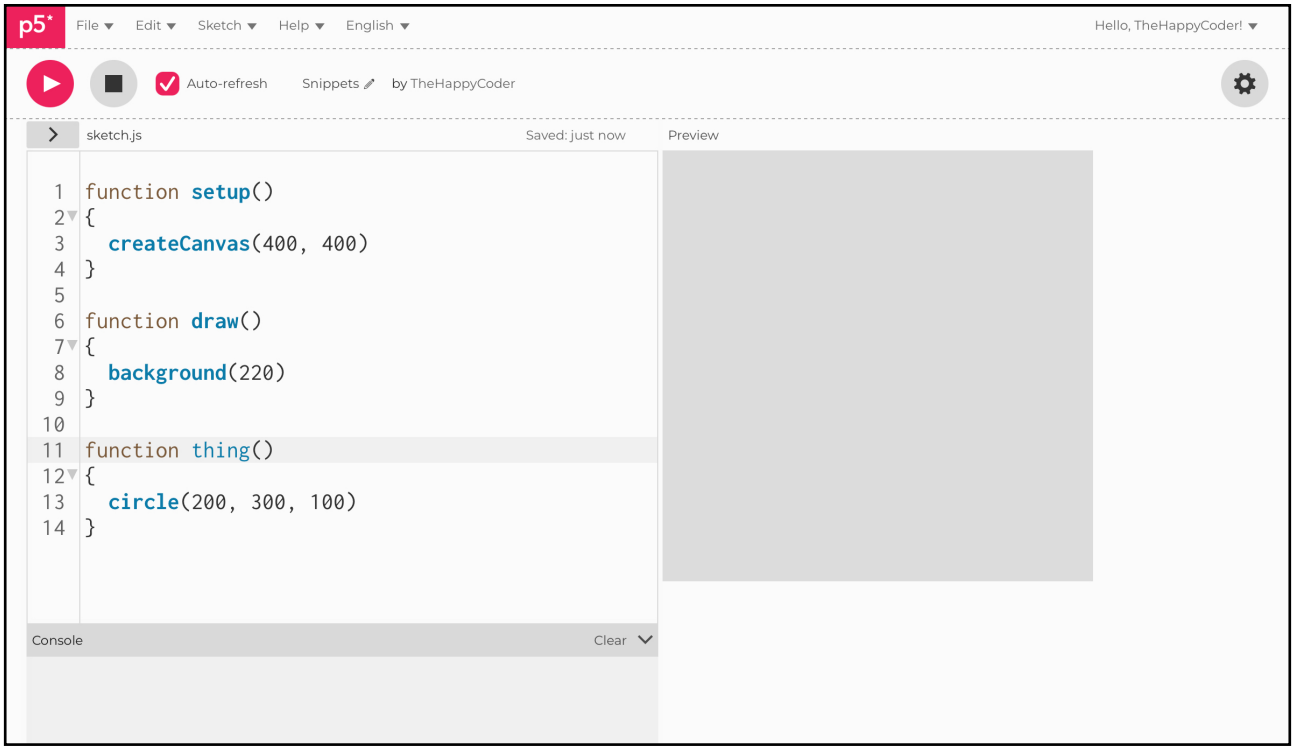
Challenges

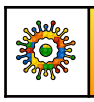
1. Change the name of the function.
2. Have the function do something else.

Code Explanation

`function thing()`	Creating and naming a function

Figure A1.24


p5*
File
Edit
Sketch
Help
English
Hello, TheHappyCoder!
Auto-refresh
Snippets
by TheHappyCoder
sketch.js
Saved: just now
Preview
1 function setup()
2 {
3 createCanvas(400, 400)
4 }
5
6 function draw()
7 {
8 background(220)
9 }
10
11 function thing()
12 {
13 circle(200, 300, 100)
14 }
Console
Clear



Sketch A1.25 calling a function

Now we call the `thing()` function in the `draw()` function.

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

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

function thing()
{
  circle(200, 300, 100)
}
```

🗒️ Notes

Now the `thing()` function is actioned through the `draw()` function. It loops continuously.

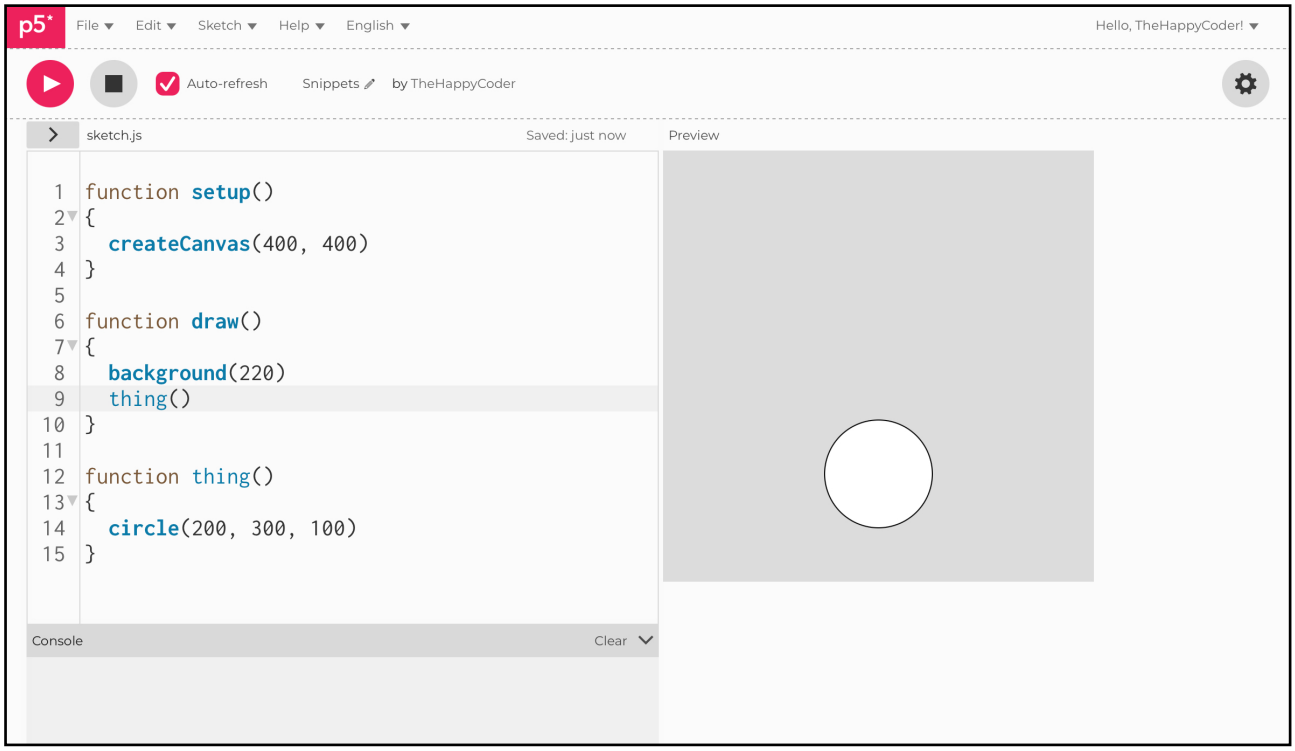
🌻 Challenge

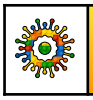
Put the `thing()` function in `setup()`.

🛠️ Code Explanation

`thing()`	Calling the function called 'thing'

Figure A1.25


p5*
File
Edit
Sketch
Help
English
Hello, TheHappyCoder!
Auto-refresh
Snippets
by TheHappyCoder
sketch.js
Saved: just now
Preview
1 function setup()
2 {
3 createCanvas(400, 400)
4 }
5
6 function draw()
7 {
8 background(220)
9 thing()
10 }
11
12 function thing()
13 {
14 circle(200, 300, 100)
15 }
Console
Clear



Sketch A1.26 creating a button (step 1)

We can create various DOM (Document Object Model) elements like buttons, radio buttons, sliders, and text boxes, which can change the appearance and make the canvas interactive. To start with, we are creating a button underneath the canvas with the words "random colour" written on it.

```
let button

function setup()
{
  createCanvas(400, 400)
  button = createButton('random colour')
  button.style('font-size', '30px')
}

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

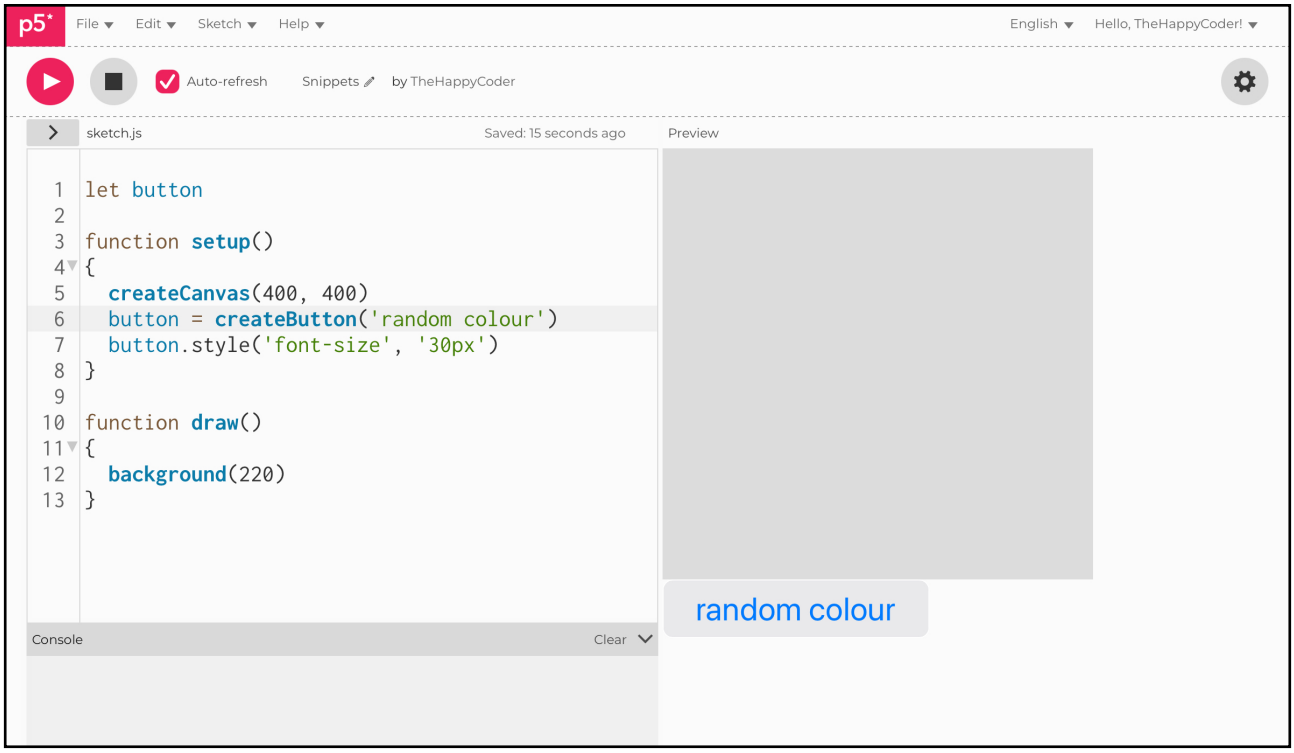
🗒️ Notes

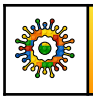
We have a button that you can click on, but nothing happens. We need another function to execute when the button is clicked.

🛠️ Code Explanation

`let button`	Create a button variable
`button = createButton('random colour')`	This creates a button with the text inside it
`button.style('font-size', '30px')`	Change the text size inside button

Figure A1.26


p5*
File
Edit
Sketch
Help
English
Hello, TheHappyCoder!
Auto-refresh
Snippets
by TheHappyCoder
sketch.js
Saved: 15 seconds ago
Preview
let button
function setup()
{
createCanvas(400, 400)
button = createButton('random colour')
button.style('font-size', '30px')
}
function draw()
{
background(220)
}
random colour
Console
Clear



Sketch A1.27 the button doing something (step 2)

Now we will need a function so that when the button is clicked, it will change the background. The new function will be called changeColour(). We add the mousePressed() function to the button, and when it is clicked, it will then call that function. We need a variable to hold the colour, and we call that newColour with an initialised value of 220.

```
let button
let newColour

function setup()
{
  createCanvas(400, 400)
  newColour = 220
  button = createButton('random colour')
  button.style('font-size', '30px')
  button.mousePressed(changeColour)
}

function draw()
{
  background(newColour)
}

function changeColour()
{
  newColour = random(255)
}
```

Notes

Now the background changes (greyscale) every time you click on the button, giving you a greyscale between 0 and 255.

🛠️ Code Explanation

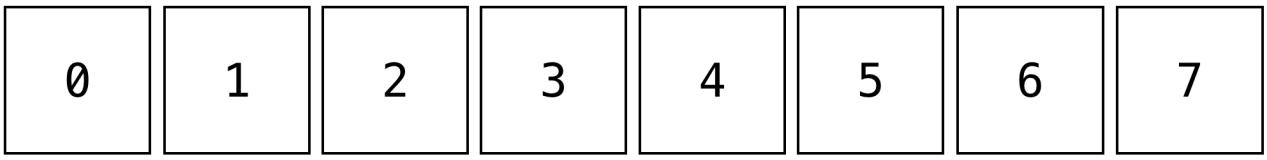
`button.mousePressed(changeColour)`	This calls the function 'changeColour' when the button is clicked with the mouse
`function changeColour()`	This is new function we have created which is activated when we click on the button
`newColour = random(255)`	We get a new random value between 0 and 255, this is the new colour
`background(newColour)`	Instead of the value we can replace with a variable name

Introducing arrays

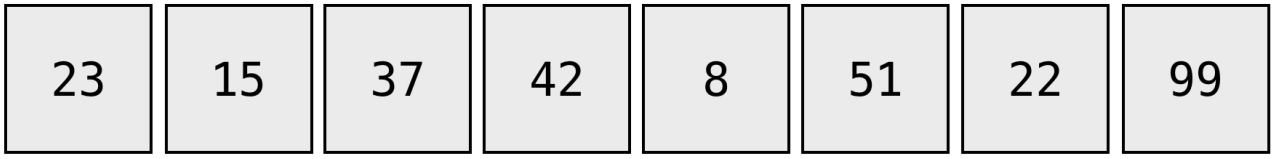
An array is a key and vital part of coding. It is a way of storing data so that it can be accessed, added to, or altered at a later date. One way to think of it is as a series of boxes that can hold bits of data (whether numbers or words). An array has a numbering system for each box, called an `index`. In coding, counting starts with `zero`, not `one`. So the first box is `index 0`, the second box is `index 1`, the third is `index 2`, and so on. You can imagine that it can be confusing that the third box is index 2.

The format to identify an array is square brackets such as `[23, 15, 37, 42, ...]` so we can describe this array as follows:
`index[0]` is `23`,
`index[1]` is `15`,
`index[2]` is `37`, and
`index[3]` is `42`, etc.

Index numbering system



The values (elements) inside each box

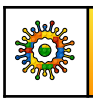


On the top row are the index[] references and on the bottom row are the actual values at those reference points. We need to give the array a name. We can use let to define the array, such as:

`let numbers = []`	This is an empty array.

We can initialise it with some initial values if we want.

`let numbers = [23, 15, 37, 42, 8, 51, 22, 99]`	The array has now got some values in it



Sketch A1.28 an empty array

! We are starting with a new sketch.

We are starting with an empty array called `data`. This means it can be any size, and we can put data into it. The data can be numbers, words, or objects.

In this example, we are going to fill an empty array with `x` and `y` values. Arrays have square brackets `[]` around them. We are calling our array `data`, and we are going to put data into it using a `push()` function. We push the values of the `x` and `y` co-ordinates into the `data` array.

As we click on the canvas, you will see something happen in the console; this is because we have used a very useful function called `console.log()` and inside the brackets, we have put the name of the array. What you see in the console is what is inside the array.

```
let data = []

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

function mousePressed()
{
  let x = mouseX
  let y = mouseY
  data.push(x, y)
  circle(x, y, 50)
  console.log(data)
}
```

🗒️ Notes

Have a look inside each array by clicking on the arrow at the start of the line. In this example, there are six elements, two for each of the circles. Just for clarification, the name of the array (`data`) is made up; we could call it anything (as long as it isn't a keyword already taken by p5.js).

🌻 Challenge

Change the name of the array.

🛠️ Code Explanation

`let data = []`	We define an empty array
`let x = mouseX`	Our x value is the mouseX position when we click
`let y = mouseY`	Our y value is the mouseY position when we click
`data.push(x, y)`	The x and y values are pushed into the data array
`console.log(data)`	We can see inside the data array

Figure A1.28a

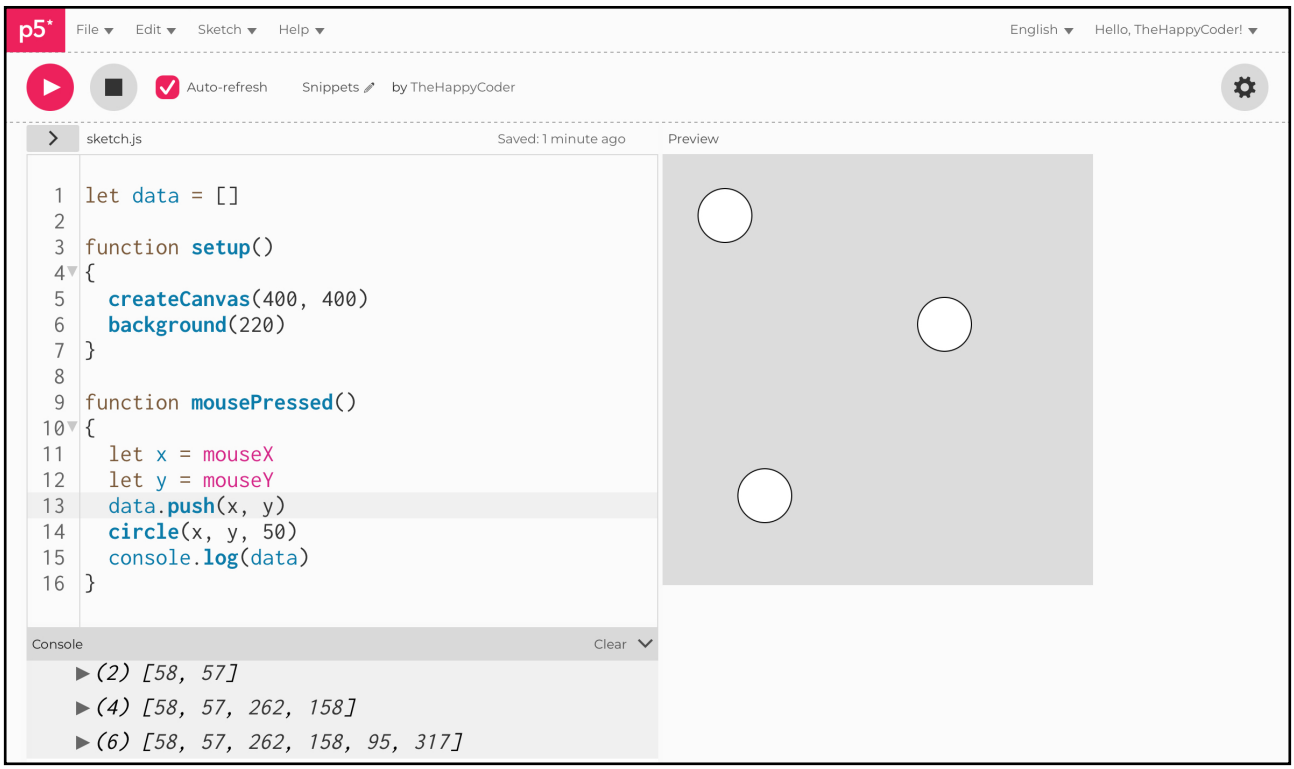
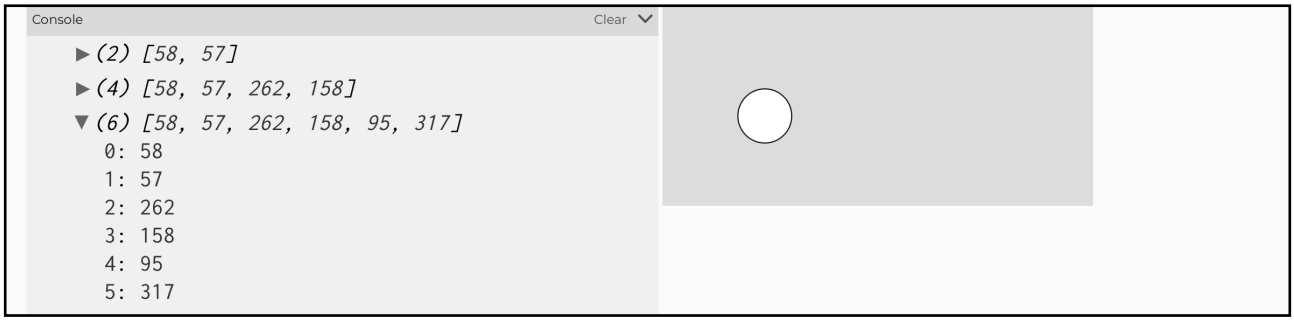


Figure A1.28b



Sketch A1.29 an array of objects

Another way to use an array is to have objects. In this example, an object is one circle. Each object has an `x` and `y` value. We will use `console.log(data)` as before to see the difference. Each object is called `inputs` to collect the `x` and `y` co-ordinates when you click on the canvas.

```
let data = []

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

function mousePressed()
{
  let inputs = {
    x: mouseX,
    y: mouseY
  }
  data.push(inputs)
  circle(inputs.x, inputs.y, 50)
  console.log(data)
}
```

Notes

We use `console.log(data)` to see inside the array. It gives you a list of objects, each one representing a circle. Inside each object are the `x` and `y` inputs. Click on the arrow to reveal the contents of the array.

🌻 Challenge

Change the variable names of `x` and `y` to, say, `flower` and `animal`. See where else you have to change them.

🛠️ Code Explanation

`let inputs = {...}`	We give the collective name for the x and y values as inputs
`x: mouseX`	For the x value we use a colon then the input value or variable
`y: mouseY`	For the y value we use a colon then the input value or variable
`circle(inputs.x, inputs.y, 50)`	The x co-ordinate of the object (inputs) is inputs.x, repeated for the y component

Figure A1.29a

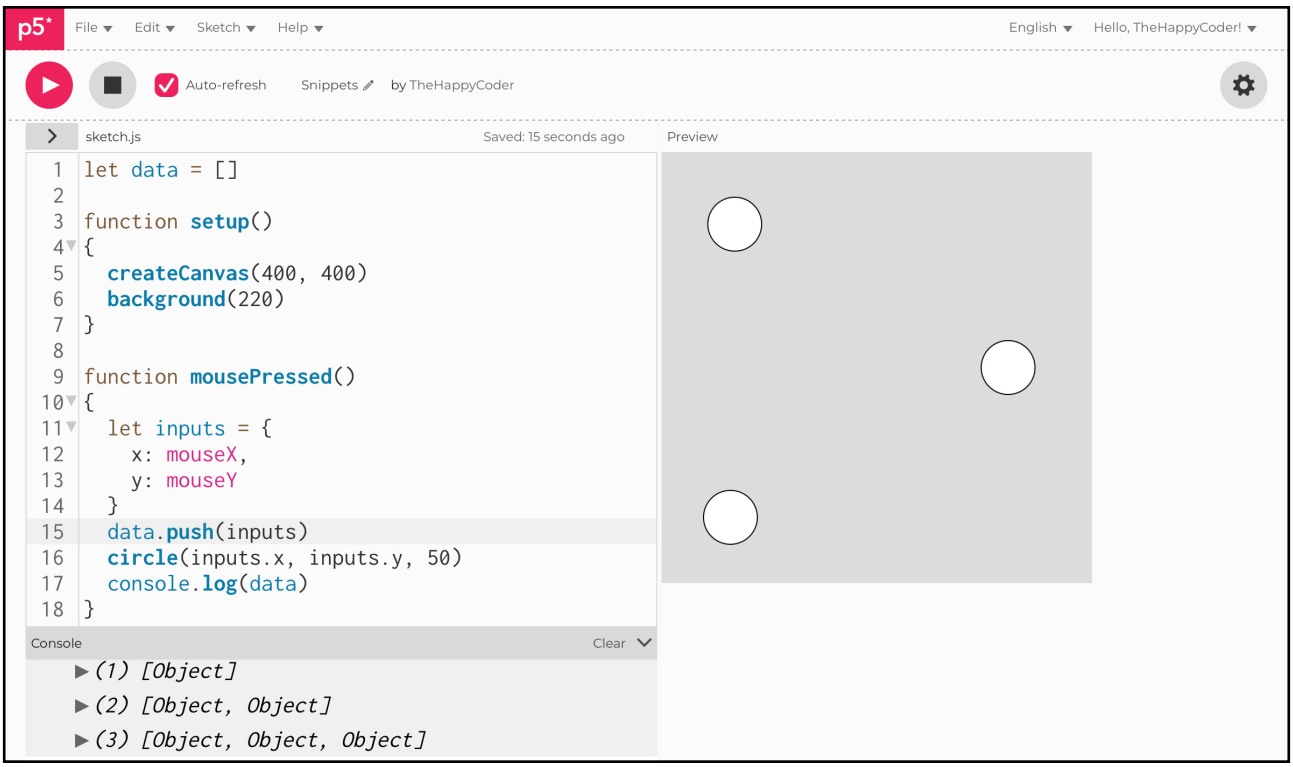
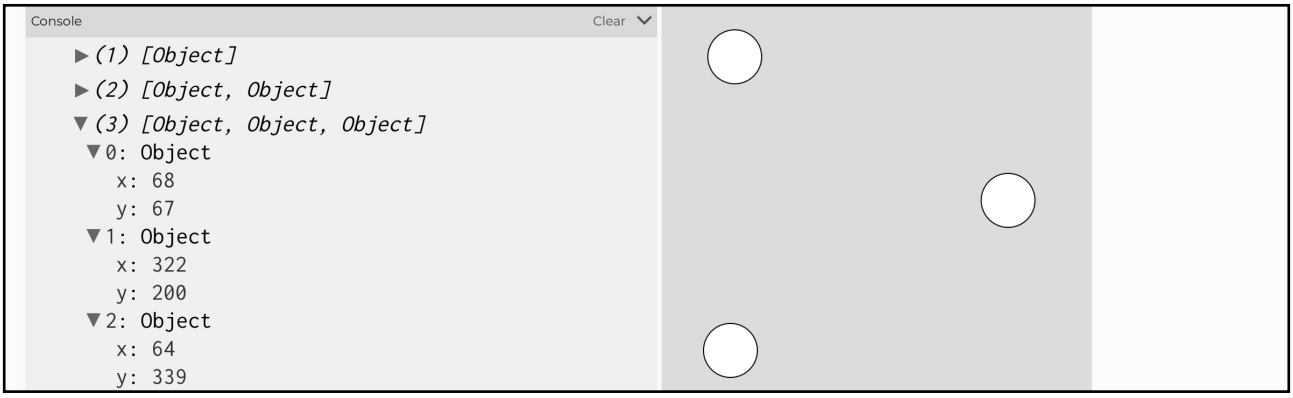
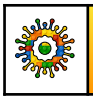


Figure A1.29b





Sketch A1.30 createVector()

❗ We are starting a completely new sketch.
This creates a new `p5.Vector` object. A vector can be thought of in different ways. One perspective is: a vector is like an arrow pointing in space. Vectors have both magnitude (length) and direction. This view is helpful for programming motion. A vector's components determine its magnitude and direction. For example, calling `createVector(3, 4)` creates a new p5.Vector object with an `x` component of `3` and a `y` component of `4`. The length is measured from the origin; this vector's tip is `3` units to the right and `4` units down.

Another really useful bit of coding is the `comments` symbol `//`. This is useful for a number of reasons:
Reason 1️⃣ to leave information in the code for yourself or someone else,
Reason 2️⃣ you can remove a line of code (without deleting it completely) to see what happens if that code is removed (for debugging a problem), and
Reason 3️⃣ keep a bunch of code that you want to use later but not need it at that precise moment.

The computer bypasses anything when a line starts with the `//` before any text; it just ignores it.

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

  // Create p5.Vector objects
  let p1 = createVector(100, 100)
  let p2 = createVector(200, 200)
  let p3 = createVector(300, 300)

  // draw circles
  // fill(255, 0, 0)
  circle(p1.x, p1.y, 50)
```

```
  circle(p2.x, p2.y, 50)
  circle(p3.x, p3.y, 50)

  // draw points
  strokeWeight(6)
  point(p1)
  point(p2)
  point(p3)
}
```

🗒️ Notes

We have drawn three circles, then three points.

🌻 Challenge

If you `console.log(p1)`, you can see inside the vector object for that vector. See image below; it is rather confusing, but you can see the difference in the array between numbers and objects (object-orientated programming).

🛠️ Code Explanation

`// Create p5.Vector objects`	Using the comments symbol, the programme ignores it, useful for information
`// fill(255, 0, 0)`	This line of code is also ignored, if you remove the // the programme will execute it.
`let p1 = createVector(100, 100)`	Creates a vector object with an x and a y component and attributes it to a variable (p1 in this case)
`circle(p1.x, p1.y, 50)`	Uses the x and y components of the object that is within the p1 variable
`point(p1)`	Drawing the point extracting the x and y components from the variable object

Sketch A1.30a

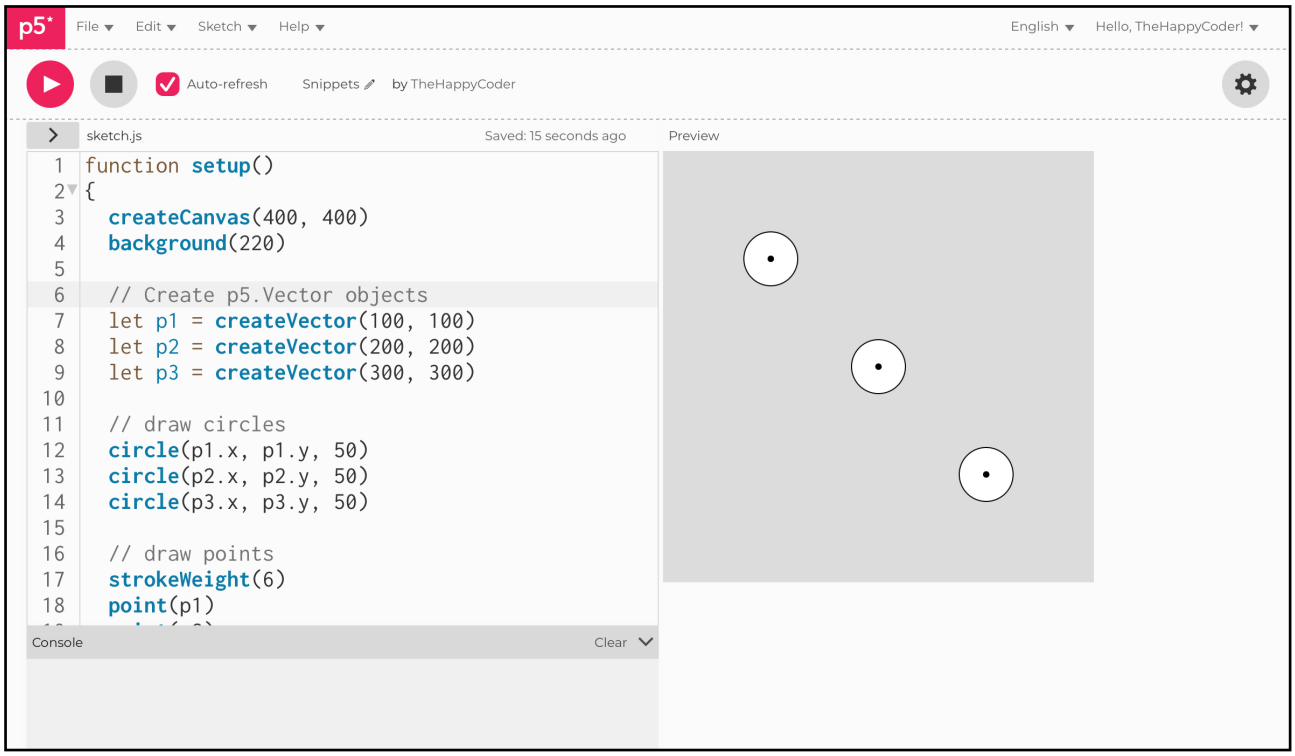
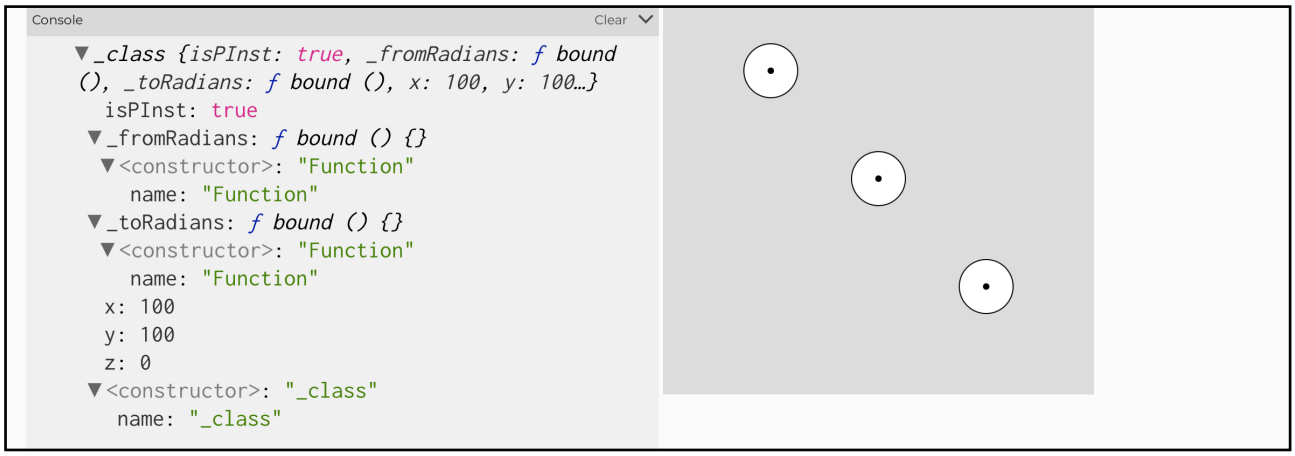


Figure A1.30b



Sketch A1.31 map() function

❗ we are starting a new sketch.

We can remap values, scaling them through the `map()` function. In this example, we have a line that follows the movement of the mouse (`mouseX`) from one side to the other. The second line is mapped from `[0, 400]` to `[0, 100]`. It still follows the movement of the mouse but is now scaled accordingly.

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

function draw()
{
  background(220)
  strokeWeight(10)
  line(0, 150, mouseX, 150)
  let x = map(mouseX, 0, 400, 0, 100)
  line(0, 250, x, 250)
}
```

🗒️ Notes

This is just an illustration of what mapping is. One way to imagine it is: if you have a scale between 27 and 187 and you want to express it as a percentage, then mapping can scale it such that 0% is 27 and 100% is 187.

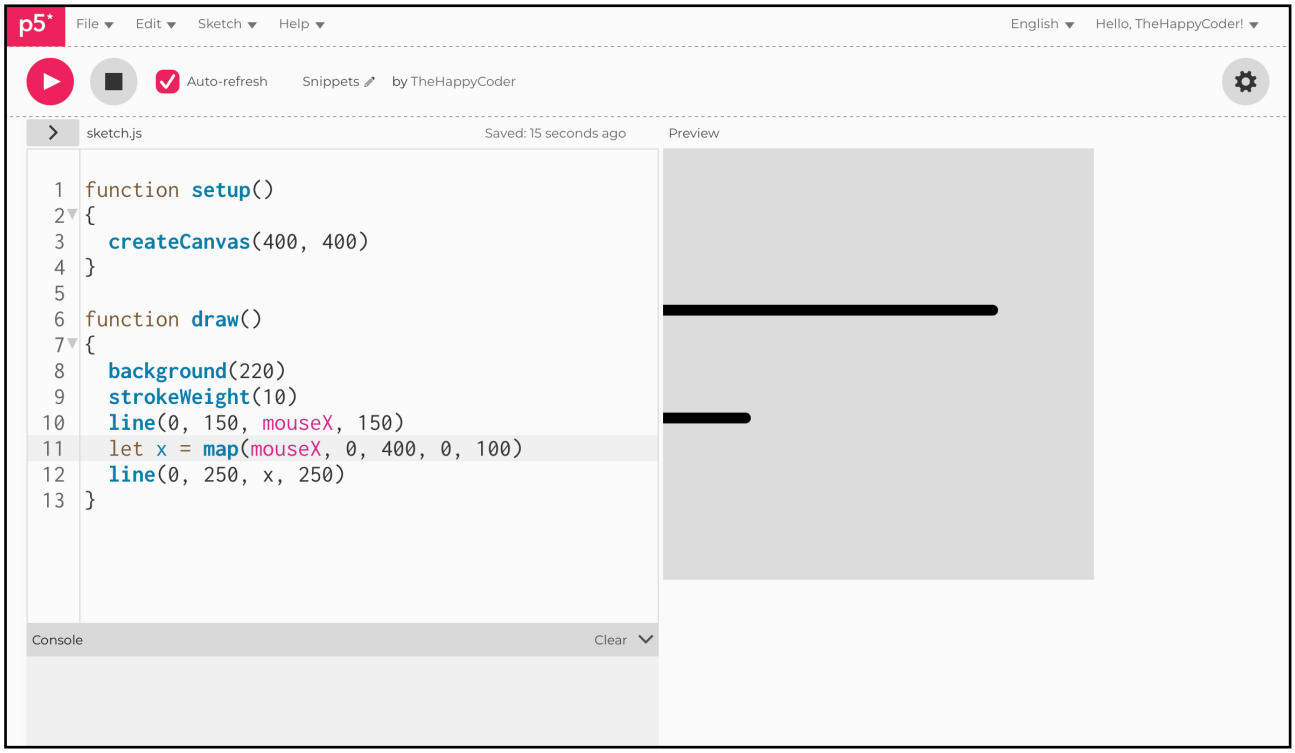
🌻 Challenge

Apply the `map()` function to the diameter of a circle.

🛠️ Code Explanation

`let x = map(mouseX, 0, 400, 0, 100)`	This maps the value of the mouseX value from 0-400 to 0-100 and returns the value to the x variable

Figure A1.31


p5*
File
Edit
Sketch
Help
English
Hello, TheHappyCoder!
Auto-refresh
Snippets
by TheHappyCoder
sketch.js
Saved: 15 seconds ago
Preview
1 function setup()
2 {
3 createCanvas(400, 400)
4 }
5
6 function draw()
7 {
8 background(220)
9 strokeWeight(10)
10 line(0, 150, mouseX, 150)
11 let x = map(mouseX, 0, 400, 0, 100)
12 line(0, 250, x, 250)
13 }
Console
Clear

Sketch A1.32 sin() function

❗ We start with a new sketch.

We have, at our disposal, the mathematical `sin()` function. The output is always between `-1` and `+1`. The input is measured in radians by default, but to make it more intuitive, we change the units to degrees with the `angleMode()` function. We make the canvas width `360` (as in 360° cycle) and we map the output from `(-1, 1)` to `(0, height)` to be more visible.

```
function setup()
{
  createCanvas(360, 400)
  angleMode(DEGREES)
  strokeWeight(5)
}

function draw()
{
  background(220)
  for (let x = 0; x < width; x++)
  {
    let y = sin(x)
    y = map(y, -1, 1, 0, height)
    point(x, y)
  }
}
```

🗒️ Notes

We get a nice sine wave.

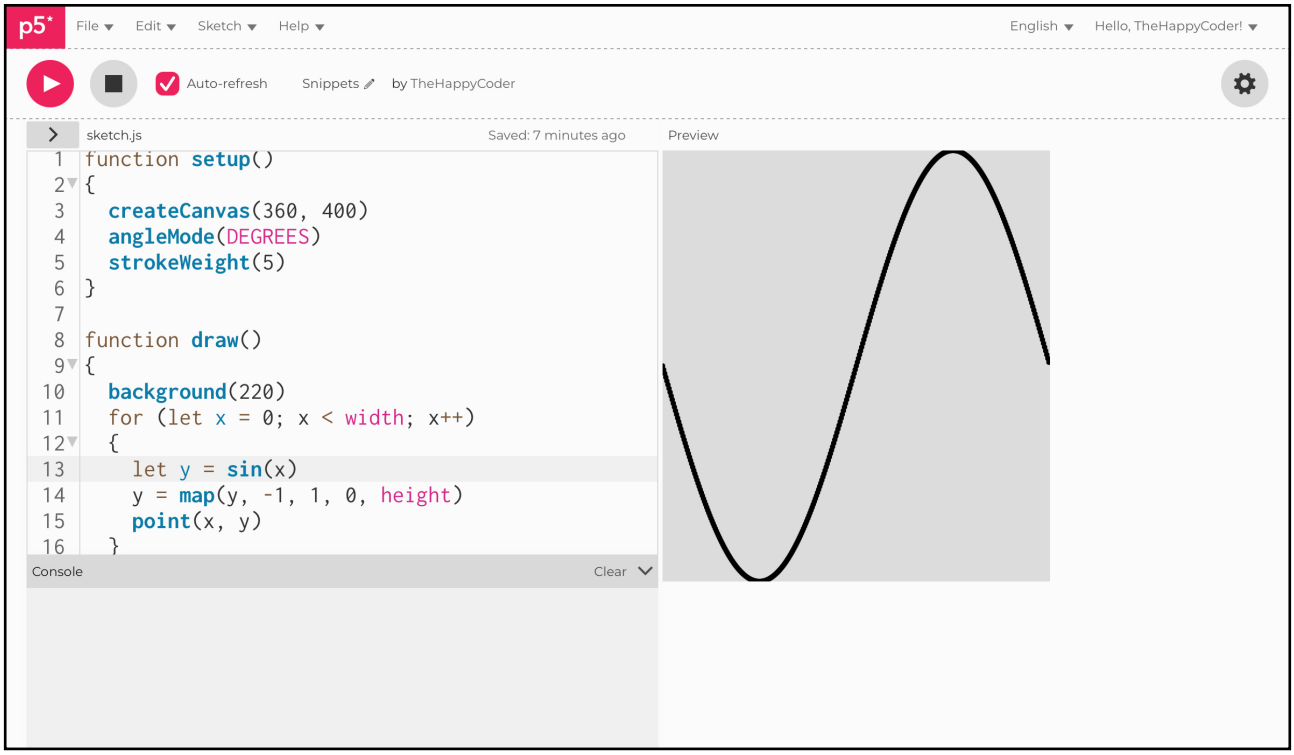
🌻 Challenge

Try `cos()`.

🛠️ Code Explanation

`angleMode(DEGREES)`	Converts the angle units from radians (default) to degrees.
`let y = sin(x)`	The y variable is the result of the sin() function on the x value

Figure A1.32


p5*
File
Edit
Sketch
Help
English
Hello, TheHappyCoder!
Auto-refresh
Snippets
by TheHappyCoder
sketch.js
Saved: 7 minutes ago
Preview
1 function setup()
2 {
3 createCanvas(360, 400)
4 angleMode(DEGREES)
5 strokeWeight(5)
6 }
7
8 function draw()
9 {
10 background(220)
11 for (let x = 0; x < width; x++)
12 {
13 let y = sin(x)
14 y = map(y, -1, 1, 0, height)
15 point(x, y)
16 }
Console
Clear

Sketch A1.33 alpha

❗ New sketch again

With colour, we can add a fourth argument, which is the alpha value, which determines how much transparency there is. With a value of 0, it is completely transparent; with a value of 255, it is completely opaque.

```
function setup()
{
  createCanvas(400, 400)
  background(255)
}

function draw()
{
  fill(255, 0, 0, 100)
  circle(random(width), random(height), 50)
}
```

🗒️ Notes

Creating lots of random red circles with an alpha of 100. This alpha also works with greyscale (if there are just two arguments) and with `stroke()`.

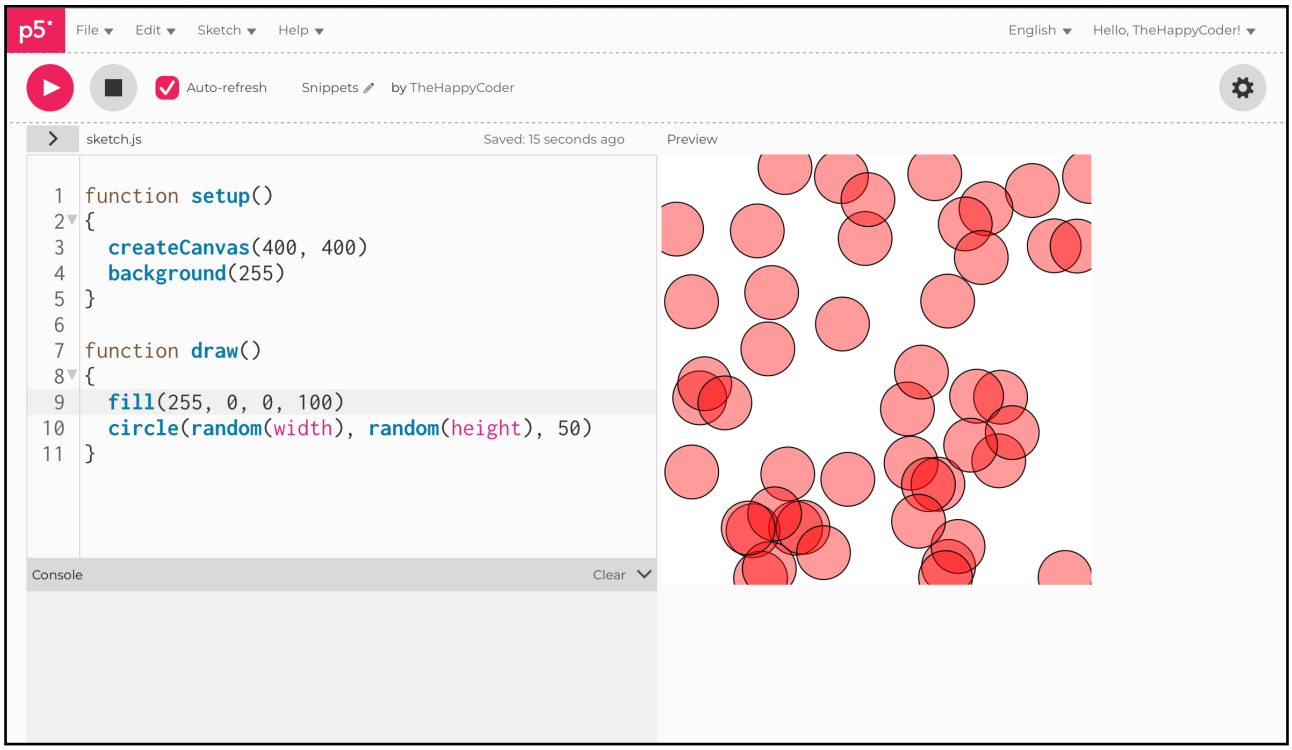
🌻 Challenge

Try different values of alpha.

🛠️ Code Explanation

`fill(255, 0, 0, 100)`	The fill() function now has four arguments, red, green, blue and alpha.

Figure A1.33


p5*
File
Edit
Sketch
Help
English
Hello, TheHappyCoder!
Auto-refresh
Snippets
by TheHappyCoder
sketch.js
Saved: 15 seconds ago
Preview
1 function setup()
2 {
3 createCanvas(400, 400)
4 background(255)
5 }
6
7 function draw()
8 {
9 fill(255, 0, 0, 100)
10 circle(random(width), random(height), 50)
11 }
Console
Clear

Sketch A1.34 text

❗ starting a new sketch.

We can write text on the canvas using the `text()` function. The `text()` function has three arguments: the first is the text (or named variable), the second and third are the `x` and `y` co-ordinates. The origin of the text is taken at the top right-hand corner. To put the origin in the centre of the text, we use the `textAlign()` function. The text is what is called a string, and it has to have speech (single or double) marks around the text.

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

function draw()
{
  background(220)
  textAlign(CENTER, CENTER)
  textSize(64)
  text('Hello', 200, 200)
}
```

🗒️ Notes

We can add colour with `stroke()` and `fill()`. You can also add new fonts, but you have to upload them.

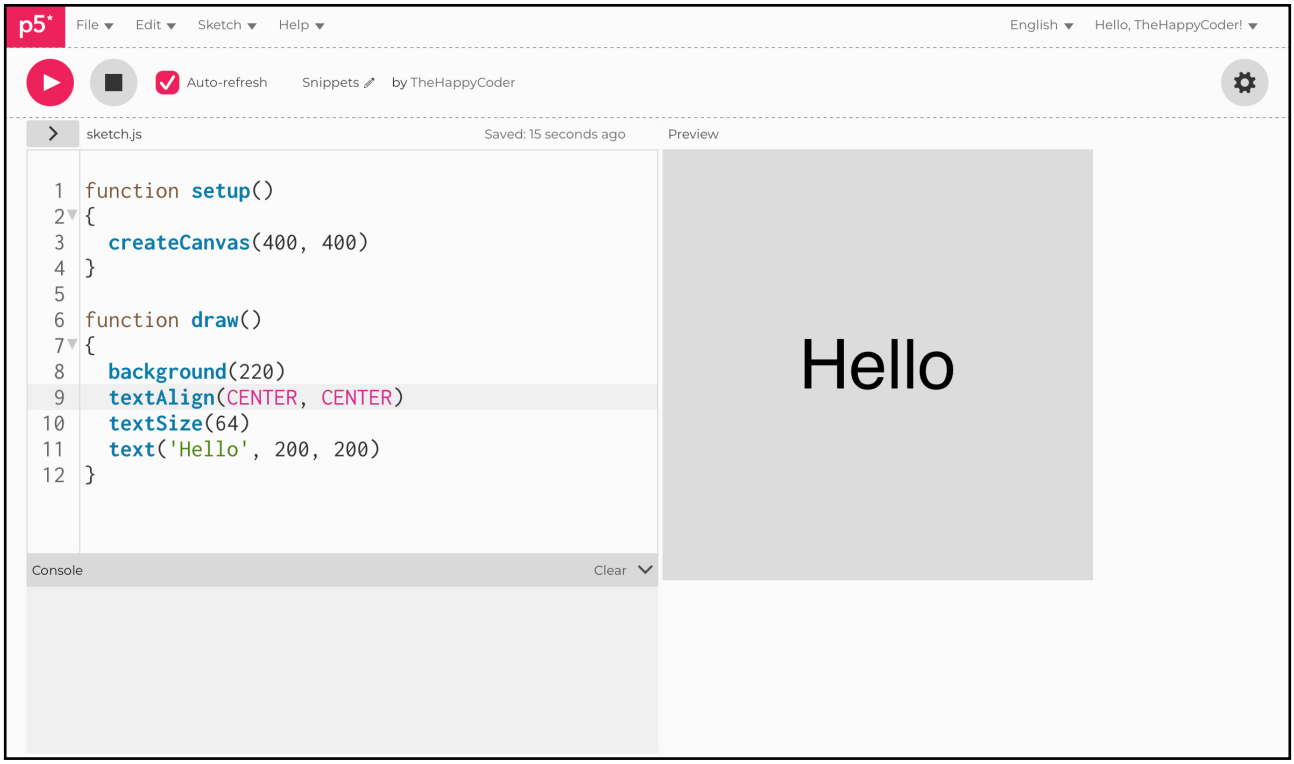
🌻 Challenge

Try commenting out the `textAlign()` line of code.

🛠️ Code Explanation

`textAlign(CENTER, CENTER)`	This moves the origin from the top left hand corner of the text into the centre of the text
`textSize(64)`	You can increase the text size
`text('Hello', 200, 200)`	The text function has three arguments, the first is the text, the second and third are the co-ordinates

Figure A1.34


p5*
File
Edit
Sketch
Help
English
Hello, TheHappyCoder!
Auto-refresh
Snippets
by TheHappyCoder
sketch.js
Saved: 15 seconds ago
Preview
1 function setup()
2 {
3 createCanvas(400, 400)
4 }
5
6 function draw()
7 {
8 background(220)
9 textAlign(CENTER, CENTER)
10 textSize(64)
11 text('Hello', 200, 200)
12 }
Hello
Console
Clear

Sketch A1.35 variable text

Instead of text as a string, we can use the value of a variable and put that on the canvas as text. In this example, we will give the values of the x and y co-ordinates to the mouse pointer.

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

function draw()
{
  background(220)
  textAlign(CENTER, CENTER)
  textSize(64)
  fill(200, 0, 0)
  text(mouseX, mouseX, mouseY - 32)
  fill(0, 0, 200)
  text(mouseY, mouseX, mouseY + 32)
}
```

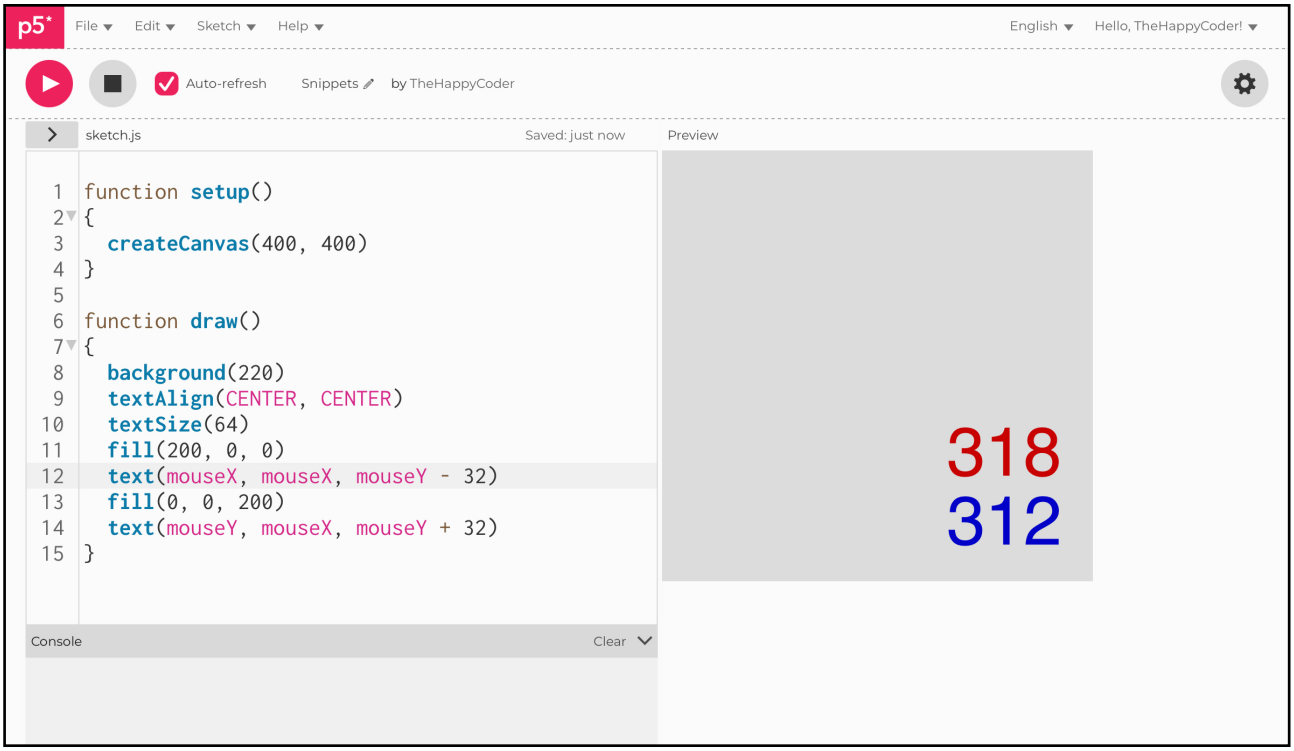
Notes

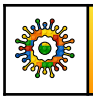
Added a splash of colour and spaced them out. It gives you the value of the variable in this case, the x position and the y position of the mouse point.

Code Explanation

`text(mouseX, mouseX, mouseY - 32)`	The mouse x position is displayed using mouseX
`text(mouseY, mouseX, mouseY + 32)`	The mouse y position is displayed using mouseY

Figure A1.35


p5*
File
Edit
Sketch
Help
English
Hello, TheHappyCoder!
Auto-refresh
Snippets
by TheHappyCoder
sketch.js
Saved: just now
Preview
1 function setup()
2 {
3 createCanvas(400, 400)
4 }
5
6 function draw()
7 {
8 background(220)
9 textAlign(CENTER, CENTER)
10 textSize(64)
11 fill(200, 0, 0)
12 text(mouseX, mouseX, mouseY - 32)
13 fill(0, 0, 200)
14 text(mouseY, mouseX, mouseY + 32)
15 }
318
312
Console
Clear



Sketch A1.36 the AND gate

❗ Start a new sketch.

We are dividing the canvas into four quadrants. If the mouse is in the top left, the circle is filled red; if top right, then filled blue; bottom right, it is filled green; and the bottom left, the circle will be just white. This makes use of the AND logic, which uses the symbols &&.

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

function draw()
{
  background(220)
  if(mouseX <= 200 && mouseY <= 200)
  {
    fill(200, 0, 0)
  }
  else if(mouseX >= 200 && mouseY <= 200)
  {
    fill(0, 0, 200)
  }
  else if(mouseX >= 200 && mouseY >= 200)
  {
    fill(0, 200, 0)
  }
  else
  {
    fill(255)
  }
```

```
  circle(200, 200, 100)
}
```

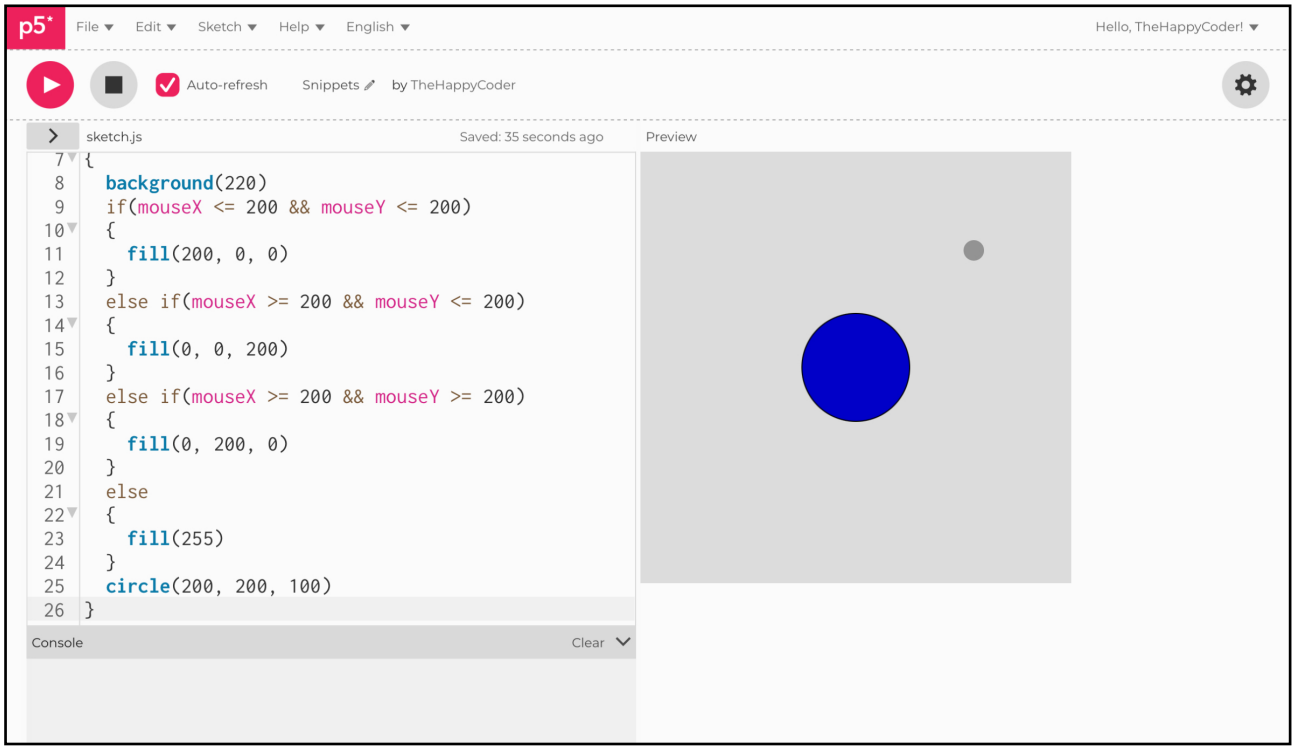
🗒️ Notes

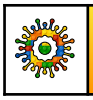
As you move the mouse around the circle in each quadrant, it changes the colour of the circle. This makes the simple AND, OR, and NOT logic very powerful, even though you only have two basic variables. In other examples, you may well have many more variables.

🛠️ Code Explanation

`if(mouseX <= 200 && mouseY <= 200)`	In summary it is saying, if the x is less than (or equal to) 200 AND y is less than 200
`else if(mouseX >= 200 && mouseY <= 200)`	Using the else if() function. If x is greater 200 AND y is less than 200
`else if(mouseX >= 200 && mouseY >= 200)`	If x is greater than 200 AND y is greater than 200
`else`	If none of the others are true then do this…

Figure A1.36


p5*
File
Edit
Sketch
Help
English
Hello, TheHappyCoder!
Auto-refresh
Snippets
by TheHappyCoder
sketch.js
Saved: 35 seconds ago
Preview
{
background(220)
if(mouseX <= 200 && mouseY <= 200)
{
fill(200, 0, 0)
}
else if(mouseX >= 200 && mouseY <= 200)
{
fill(0, 0, 200)
}
else if(mouseX >= 200 && mouseY >= 200)
{
fill(0, 200, 0)
}
else
{
fill(255)
}
circle(200, 200, 100)
}
Console
Clear



Sketch A1.37 the OR gate

The OR gate is the opposite of the AND logic. In this one, we simply change one line of code from AND to OR, and the difference is obvious. Work through the logic of why that is the case. The OR symbol is two vertical lines (pipes) ||.

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

function draw()
{
  background(220)
  if(mouseX <= 200 || mouseY <= 200)
  {
    fill(200, 0, 0)
  }
  else if(mouseX >= 200 && mouseY <= 200)
  {
    fill(0, 0, 200)
  }
  else if(mouseX >= 200 && mouseY >= 200)
  {
    fill(0, 200, 0)
  }
  else
  {
    fill(255)
  }
  circle(200, 200, 100)
}
```

🗒️ Notes

We get three quarters of the canvas red (top left, top right and bottom left); the remainder is still green.

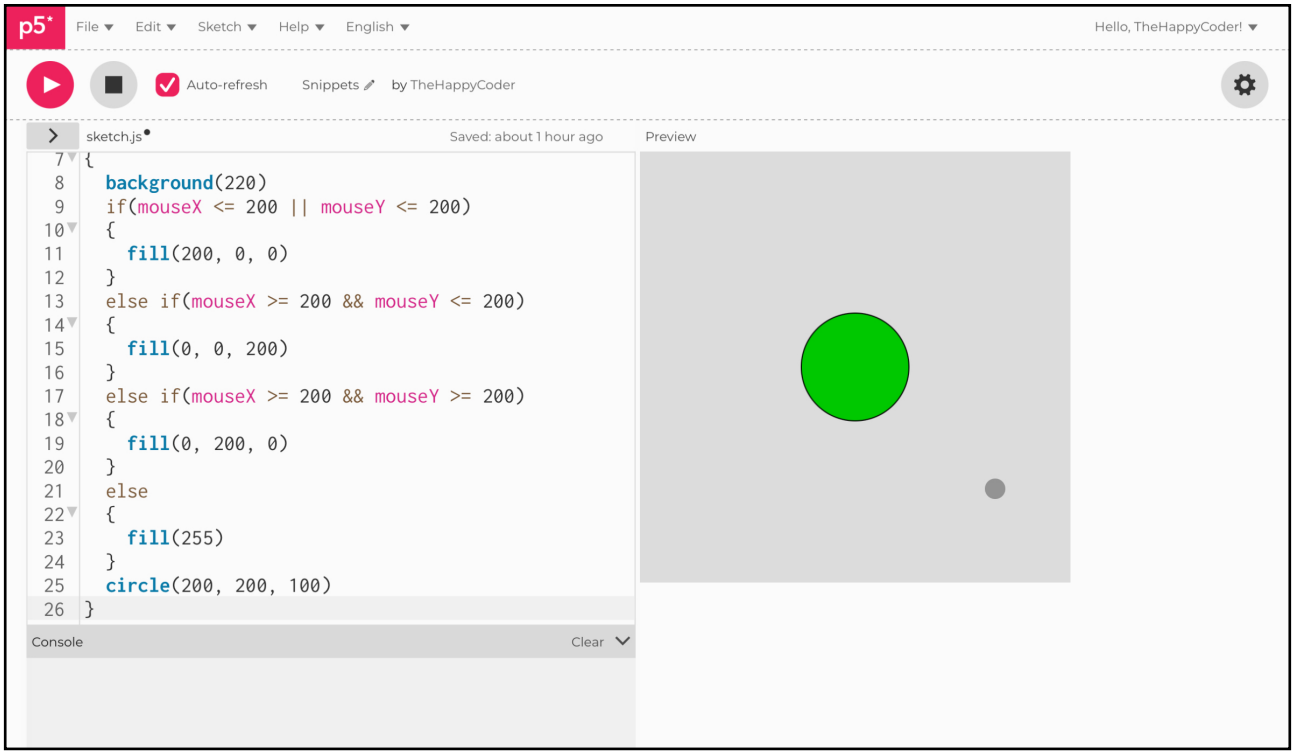
🌻 Challenges

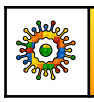
1. Try other variations of AND or OR logic.
2. Recode this sketch so that you colour different segments.
3. Try using the NOT logic !.

🛠️ Code Explanation

`if(mouseX <= 200 || mouseY <= 200)`	This contains two possible truths. Either x is less than 200 OR y is less than 200

Figure A1.37


p5*
File
Edit
Sketch
Help
English
Hello, TheHappyCoder!
Auto-refresh
Snippets
by TheHappyCoder
sketch.js
Saved: about 1 hour ago
Preview
{
background(220)
if(mouseX <= 200 || mouseY <= 200)
{
fill(200, 0, 0)
}
else if(mouseX >= 200 && mouseY <= 200)
{
fill(0, 0, 200)
}
else if(mouseX >= 200 && mouseY >= 200)
{
fill(0, 200, 0)
}
else
{
fill(255)
}
circle(200, 200, 100)
}
Console
Clear



Sketch A1.38 p5.Vector.sub() function

Here we are going to create two vector objects, one vector is the centre of the canvas, the other vector is the position of the mouse point. We subtract the mouse point vector (mouseX and mouseY) from the centre of the canvas vector.

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

function draw()
{
  background(220)
  let v1 = createVector(mouseX, mouseY)
  let v2 = createVector(200, 200)
  let v3 = p5.Vector.sub(v1, v2)
  strokeWeight(5)
  stroke('red')
  line(v3.x, v3.y, v2.x, v2.y)
  stroke('blue')
  line(v1.x, v1.y, v2.x, v2.y)
}
```

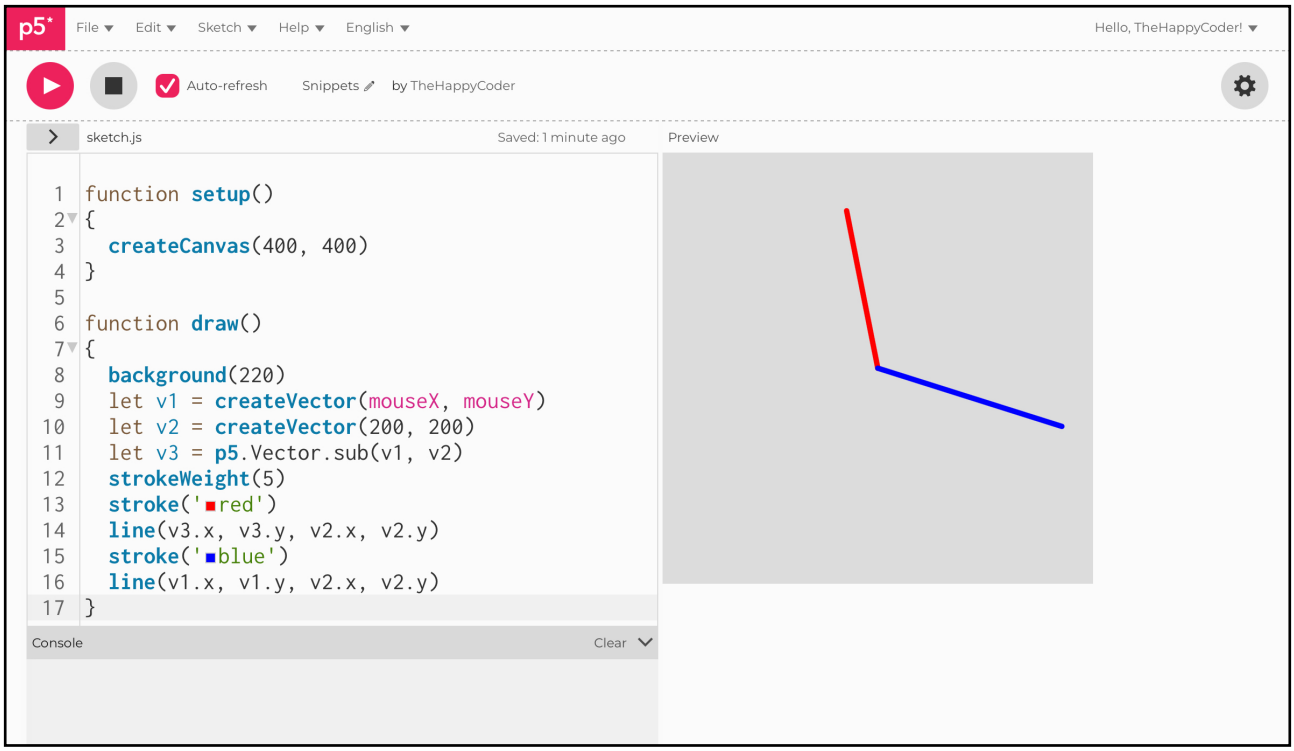
Notes

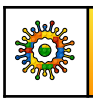
The red line is the subtraction of the other two vectors.

🛠️ Code Explanation

`let v3 = p5.Vector.sub(v1, v2)`	This function allows us the subtract two vectors, return a third vector object
`stroke('red')`	As well as rgb colour values we can use names as a shortcut. There lots of colours to chose from. Needs to have speech marks

Figure A1.38


p5*
File
Edit
Sketch
Help
English
Hello, TheHappyCoder!
Auto-refresh
Snippets
by TheHappyCoder
sketch.js
Saved: 1 minute ago
Preview
1 function setup()
2 {
3 createCanvas(400, 400)
4 }
5
6 function draw()
7 {
8 background(220)
9 let v1 = createVector(mouseX, mouseY)
10 let v2 = createVector(200, 200)
11 let v3 = p5.Vector.sub(v1, v2)
12 strokeWeight(5)
13 stroke('red')
14 line(v3.x, v3.y, v2.x, v2.y)
15 stroke('blue')
16 line(v1.x, v1.y, v2.x, v2.y)
17 }
Console
Clear



Sketch A1.39 p5.Vector.add() function

The result may look the same, but now the p1 vector is being added to the p2 vector, making the p3 vector (red) much bigger.

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

function draw()
{
  background(220)
  let v1 = createVector(mouseX, mouseY)
  let v2 = createVector(200, 200)
  let v3 = p5.Vector.add(v1, v2)
  strokeWeight(5)
  stroke('red')
  line(v3.x, v3.y, v2.x, v2.y)
  stroke('blue')
  line(v1.x, v1.y, v2.x, v2.y)
}
```

🗒️ Notes

There are a whole raft of vector functions; they are too many to mention here.

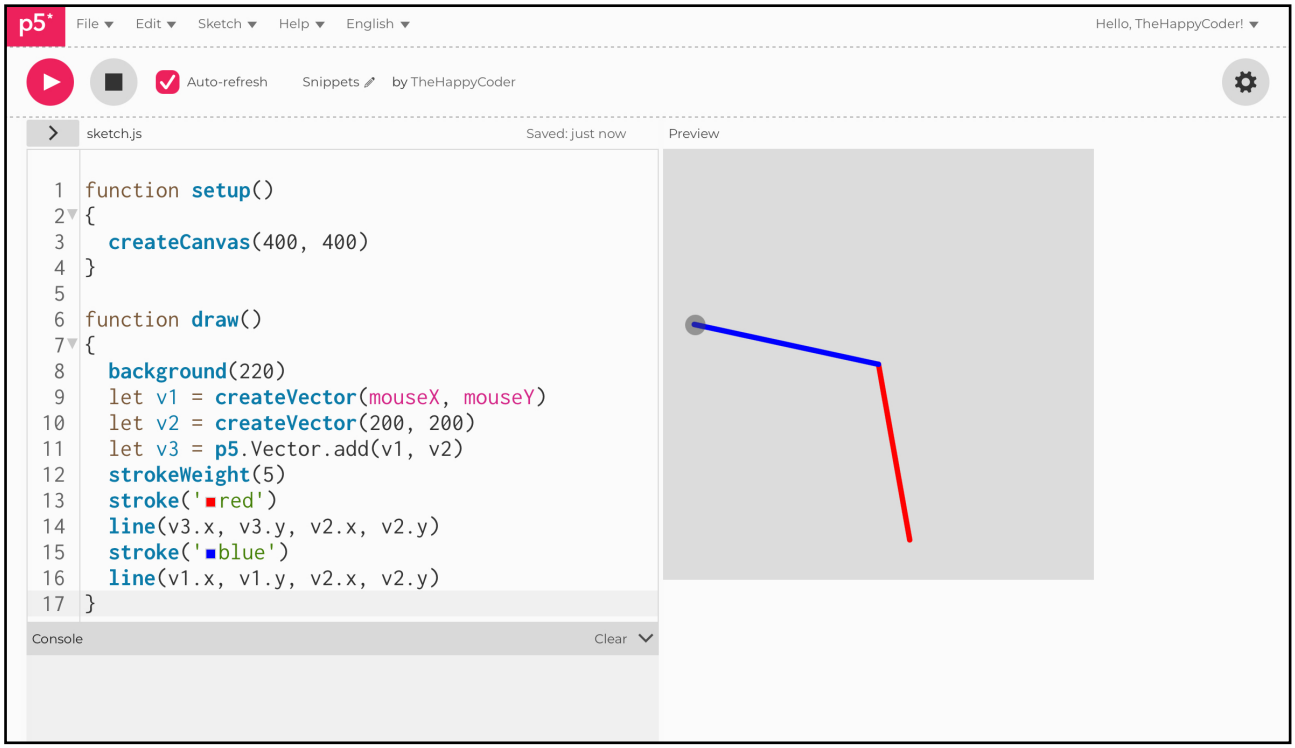
🌻 Challenge

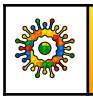
Try .mult() instead of .add().

🛠️ Code Explanation

`let v3 = p5.Vector.add(v1, v2)`	Adds two vectors returning a third vector

Figure A1.39


p5*
File
Edit
Sketch
Help
English
Hello, TheHappyCoder!
Auto-refresh
Snippets
by TheHappyCoder
sketch.js
Saved: just now
Preview
1 function setup()
2 {
3 createCanvas(400, 400)
4 }
5
6 function draw()
7 {
8 background(220)
9 let v1 = createVector(mouseX, mouseY)
10 let v2 = createVector(200, 200)
11 let v3 = p5.Vector.add(v1, v2)
12 strokeWeight(5)
13 stroke('red')
14 line(v3.x, v3.y, v2.x, v2.y)
15 stroke('blue')
16 line(v1.x, v1.y, v2.x, v2.y)
17 }
Console
Clear



Sketch A1.40 a bit more mouse

❗ New sketch

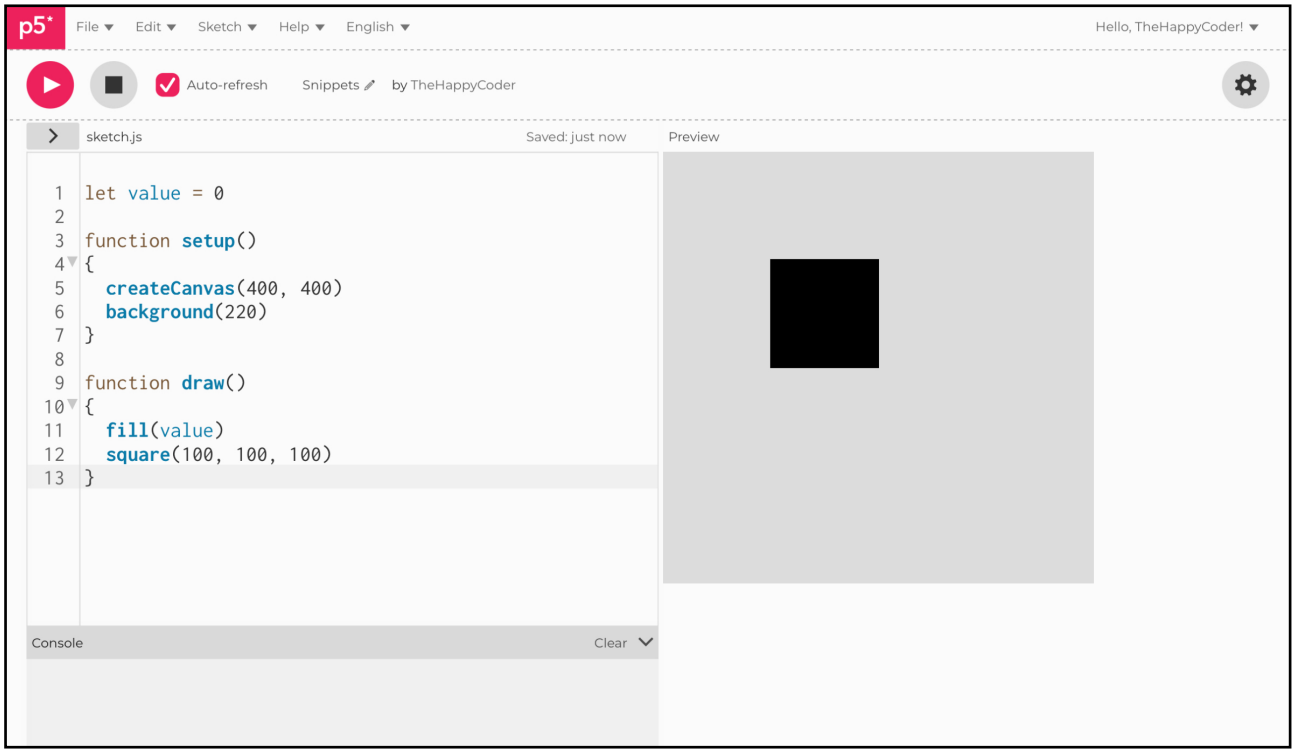
There are a lot of functions we can use with the mouse: its movement, the buttons, and the mouse wheel. Here are just a few to demonstrate functions relating to the mouse buttons. This is our starting sketch.

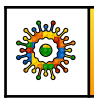
```
let value = 0

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

function draw()
{
  fill(value)
  square(100, 100, 100)
}
```

Notes

We get a black square, nothing more, nothing less.

Figure A1.40


p5*
File
Edit
Sketch
Help
English
Hello, TheHappyCoder!
Auto-refresh
Snippets
by TheHappyCoder
sketch.js
Saved: just now
Preview
let value = 0
function setup()
{
createCanvas(400, 400)
background(220)
}
function draw()
{
fill(value)
square(100, 100, 100)
}
Console
Clear



Sketch A1.41 mouseDragged() function

Click on the canvas and keep the button down as you move the mouse across the canvas. We can change the colour of the square as we click and drag the mouse across the canvas.

```
let value = 0

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

function draw()
{
  fill(value)
  square(100, 100, 100)
}

function mouseDragged()
{
  value += 2
  if (value > 255)
  {
    value = 0
  }
}
```

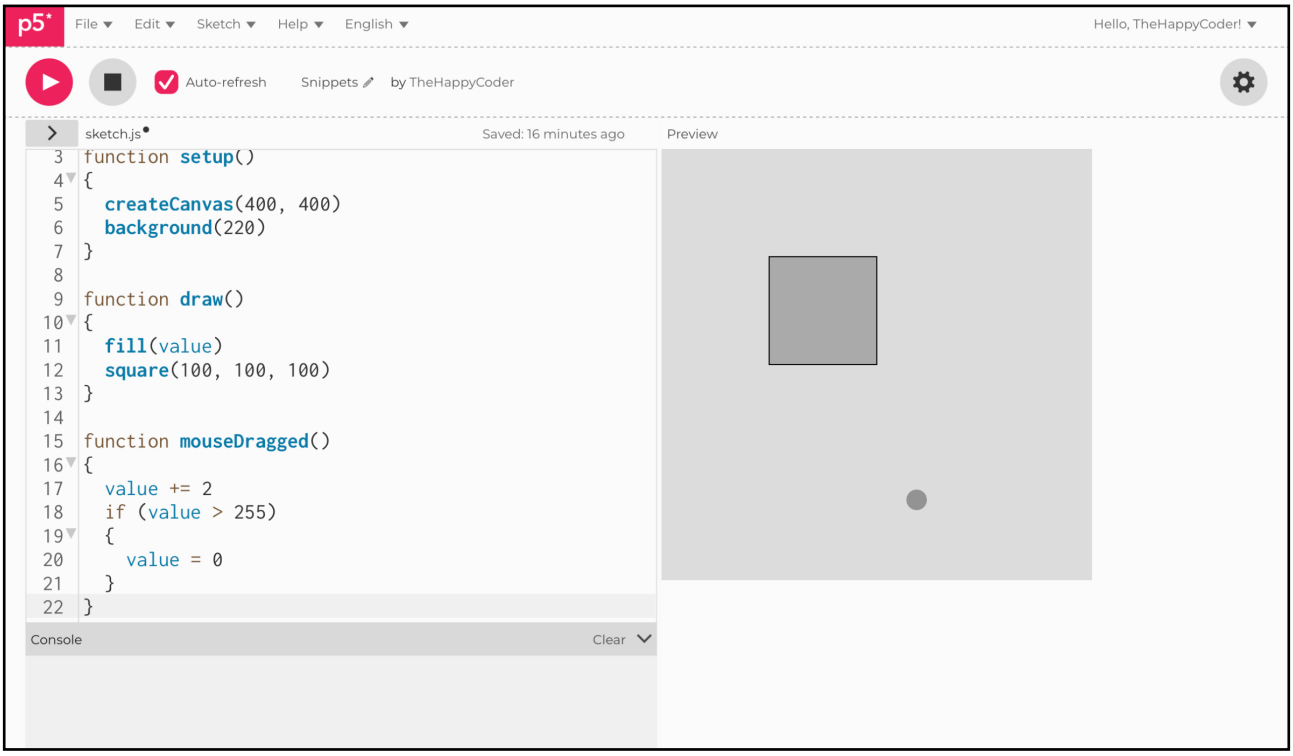
Notes

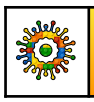
The greyness of the square changes as we drag the mouse.

🛠️ Code Explanation

`function mouseDragged()`	Senses when the mouse has been click and moved at the same time.
`value += 2`	Adds 2 every time the mouse is moved

Figure A1.41


p5*
File
Edit
Sketch
Help
English
Hello, TheHappyCoder!
Auto-refresh
Snippets
by TheHappyCoder
sketch.js
Saved: 16 minutes ago
Preview
3 function setup()
4 {
5 createCanvas(400, 400)
6 background(220)
7 }
8
9 function draw()
10 {
11 fill(value)
12 square(100, 100, 100)
13 }
14
15 function mouseDragged()
16 {
17 value += 2
18 if (value > 255)
19 {
20 value = 0
21 }
22 }
Console
Clear



Sketch A1.42 mousePressed() function

This is a simple way to toggle using a mouse button.

```
let value = 0

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

function draw()
{
  fill(value)
  square(100, 100, 100)
}

function mousePressed()
{
  if (value === 0)
  {
    value = 255
  }
  else
  {
    value = 0
  }
}
```

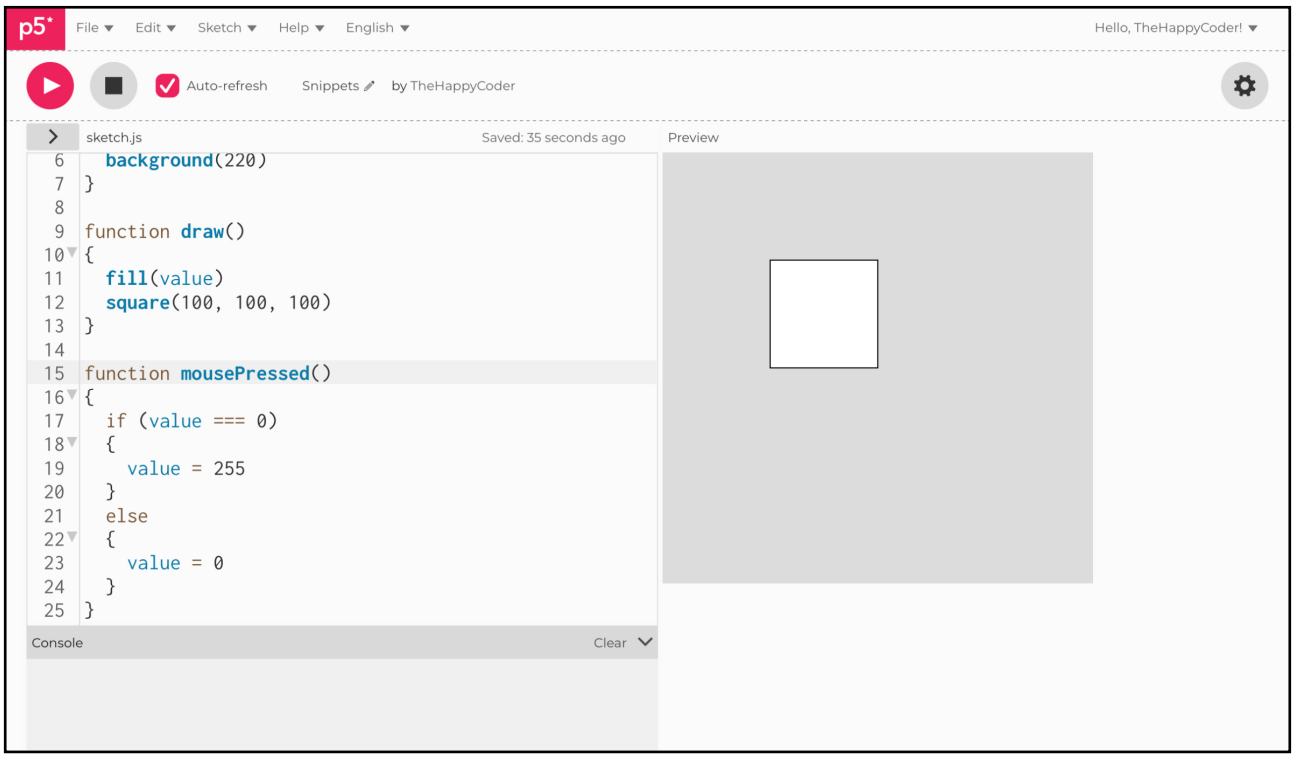
Notes

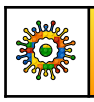
Every time you click on the canvas, the square toggles between white and then black.

🛠️ Code Explanation

`function mousePressed()`	Senses when the mouse button has been pressed

Figure A1.42


p5*
File
Edit
Sketch
Help
English
Hello, TheHappyCoder!
Auto-refresh
Snippets
by TheHappyCoder
sketch.js
Saved: 35 seconds ago
Preview
6 background(220)
7 }
8
9 function draw()
10 {
11 fill(value)
12 square(100, 100, 100)
13 }
14
15 function mousePressed()
16 {
17 if (value === 0)
18 {
19 value = 255
20 }
21 else
22 {
23 value = 0
24 }
25 }
Console
Clear



Sketch A1.43 mouseReleased() function

Does the same but only acts when the button is released.

```
let value = 0

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

function draw()
{
  fill(value)
  square(100, 100, 100)
}

function mouseReleased()
{
  if (value === 0)
  {
    value = 255
  }
  else
  {
    value = 0
  }
}
```

Notes

A slight difference, but it can be useful when drawing a line that finishes when you release the mouse button.

🛠️ Code Explanation

`function mouseReleased()`	Senses when the mouse button has been released

Finally

If you worked through all this, well done. Now you have a basic but sound understanding of the workings of p5.js. Next, we can start the fun bit of machine learning. Enjoy.