Algorithmic Art Workbook #7 Sound and Vision

Table of Contents

MIT Licence

Brief summary:

Permissions (what you can do)

Commercial use
Modification
Distribution
Private use

Limitations (what is not covered)

Liability
Warranty

Workbook #7 Quick Content Summary

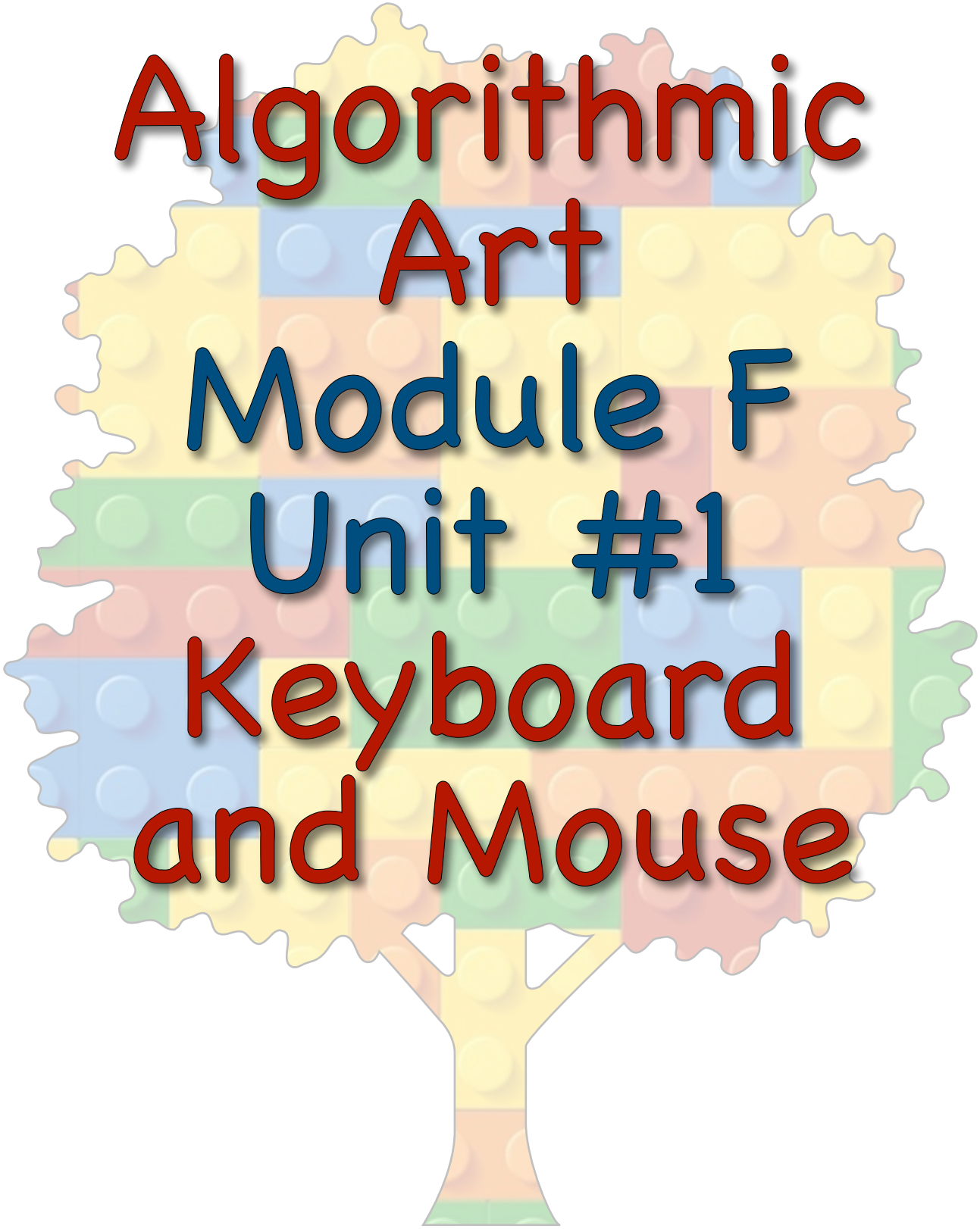
I will be honest this workbook is a bit of mishmash of things that you may well find very useful and some things that are just interesting to note. In the transition from version 1.x to version 2.x some of the functionality seems to have been lost.

You could do a lot more if you revert to and use a much older version but if you are starting now on your coding journey I wanted to make sure that as much as possible you get used to using version 2.

I will leave that avenue for you to explore at your leisure if you are interested.

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

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


Algorithmic
Art
Module F
Unit #1
Keyboard
and Mouse

Module F Unit #1: keyboard and mouse

This unit shows how you can interact with the mouse and keyboard. There are several useful functions built into p5.js. It can determine where the (x, y) position of the mouse is on the canvas, and know whether the mouse has been clicked or pressed, etc.

The Keyboard

You can use the keyboard to interact with objects and text. There are some special keys that are available in p5.js. Some of the special keys in p5.js are... (notice that they are uppercase)

The up arrow is `UP_ARROW`
The down arrow is `DOWN_ARROW`
The left arrow is `LEFT_ARROW`
The right arrow is `RIGHT_ARROW`
Return is `RETURN`

The Mouse

This can also be used to move objects around and interact with functions. This is a comprehensive list in the sketches below, but you will be surprised when you might need it.

Sketch F1.1 KeyIsPressed

! Start a new sketch

First, you will need to click on the canvas and then press any key on your keyboard.

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

function draw()
{
  if (keyIsPressed === true)
  {
    fill(0)
  }
  else
  {
    fill(255)
  }
  square(100, 100, 100)
}
```

Notes

The square will change from white to black when you press a key.

Code Explanation

`if (keyIsPressed === true)`	Check to see if any key on the keyboard has been pressed

Figure F1.1

Sketch F1.2 Key

! Start a new sketch.

This prints any key that is pressed on the canvas.

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

function draw()
{
  background(220)
  text(key, 100, 100)
}
```

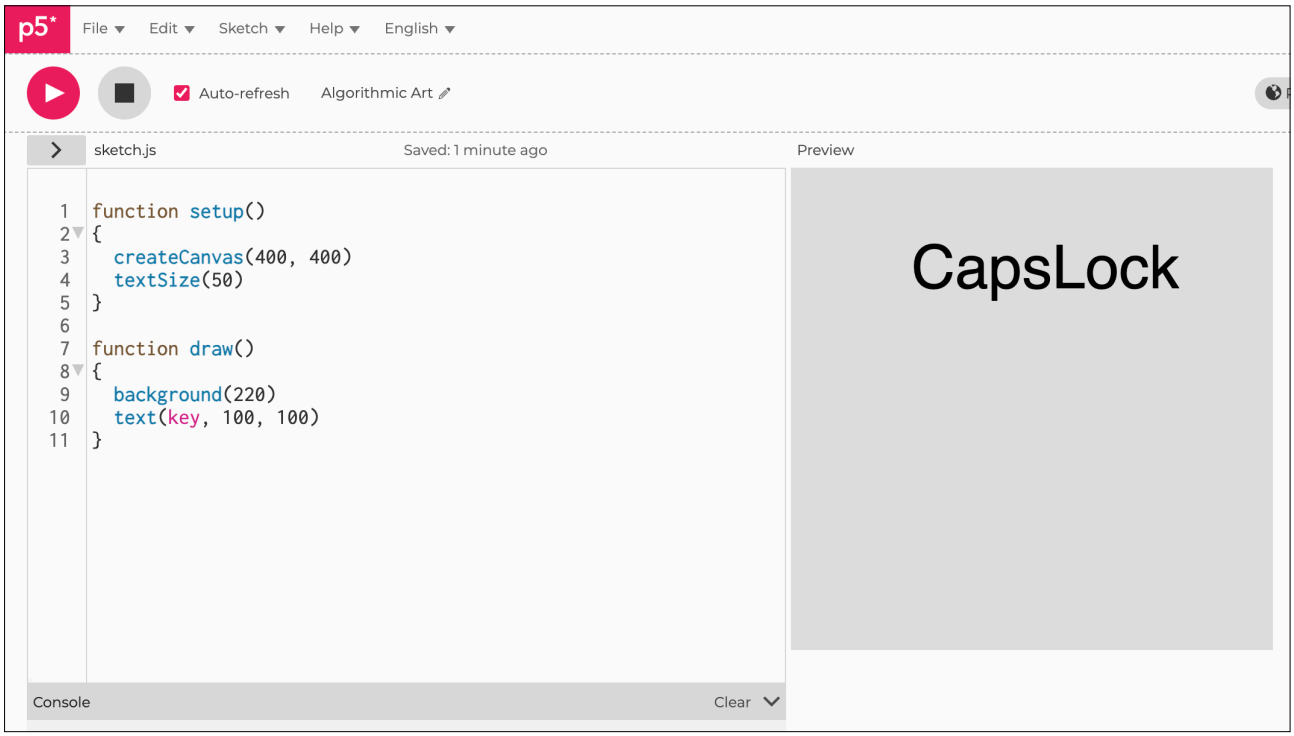
Notes

Showed the pressing of the Caps Lock key.

Code Explanation

`text(key, 100, 100)`	Returns the key pressed

Figure F1.2



Sketch F1.3 keyPressed()

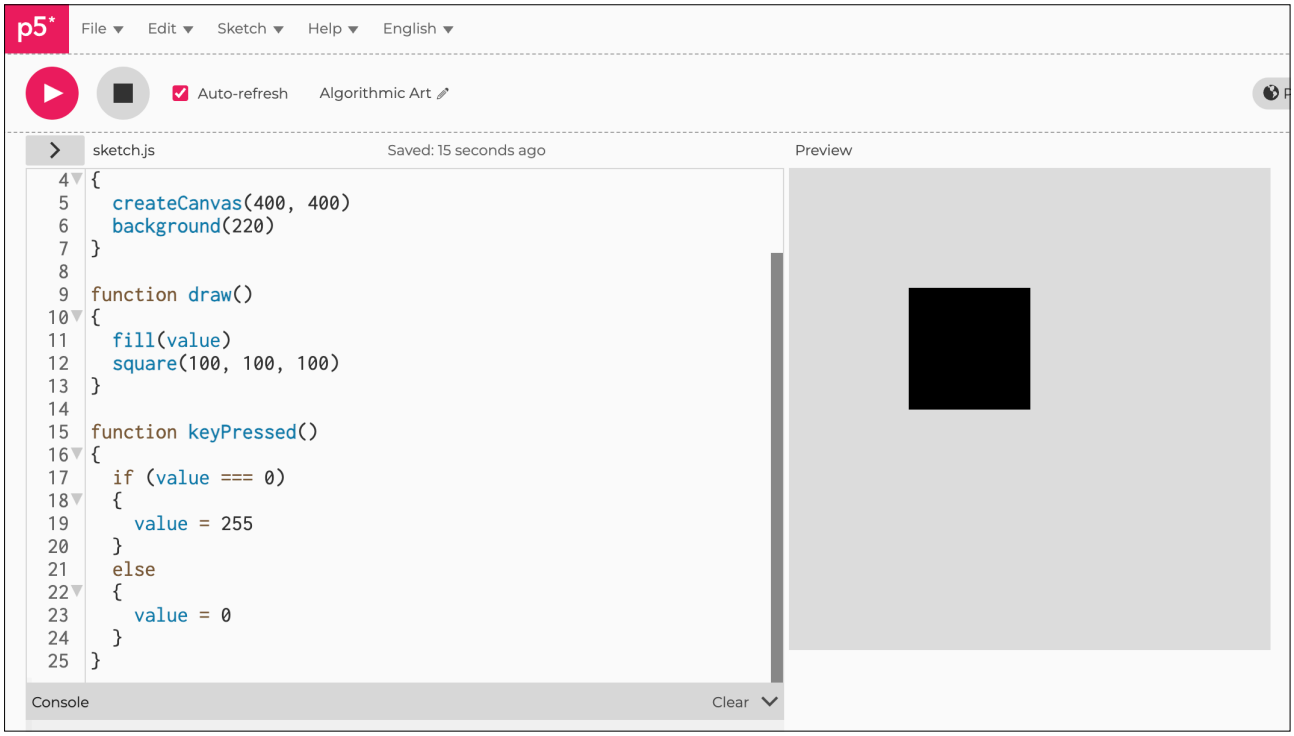
! Start a new sketch
This toggles the square fill colour every time a key is pressed.

```
let value = 0

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

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

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

Notes

Toggles between being filled white and black every time you press any key.

Figure F1.3



Sketch F1.4 keyReleased()

This simply acts when the key is released

```
let value = 0

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

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

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

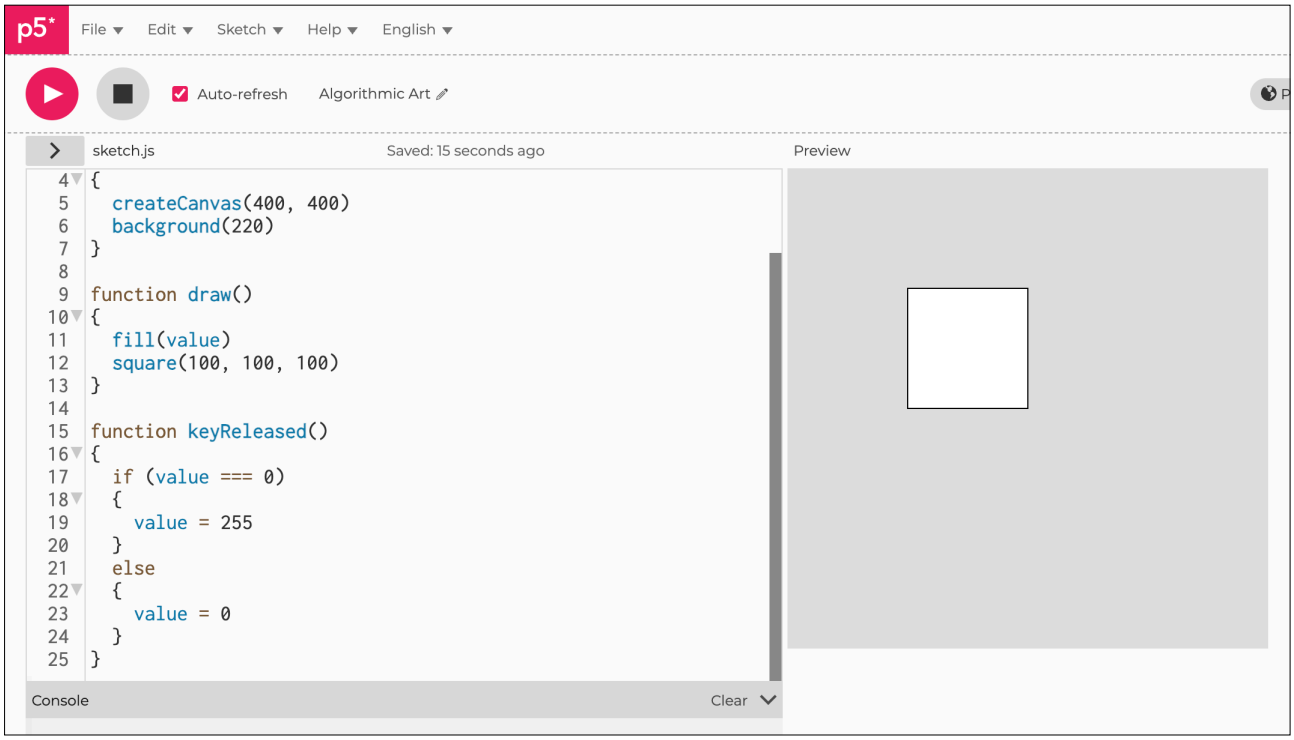
Notes

Only changes at the point of release.

Code Explanation

`function keyReleased()`	As you let go of the key

Figure F1.4



Sketch F1.5 KeyCode

Here we check for a specific key to be pressed; in this case, the up arrow (38) changes it to white, and the down arrow (40) toggles it to black.

```
let value = 126

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

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

function keyPressed()
{
  if (keyCode === 38)
  {
    value = 255
  }
  else if(keyCode === 40)
  {
    value = 0
  }
}
```

Notes

All keys have a numeric value. The effect is to change the colour of the square, black or white, depending on whether you press the up arrow or down arrow on your keyboard.

Sketch F1.6 keyTyped()

Press key A on the keyboard for a white square and B for a black square.

```
let value = 0

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

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

function keyTyped()
{
  if (key === 'a')
  {
    value = 255
  }
  else if (key === 'b')
  {
    value = 0
  }
}
```

Notes

We toggle black to white and vice versa.

Sketch F1.7 KeyIsDown()

! Suggest starting a new sketch.
This creates an event only while that specific key is held down. Use the arrow keys: up, down, left, and right to move the red circle.

```
let x = 100
let y = 100

function setup()
{
  createCanvas(400, 400)
  fill(200, 0, 0)
}

function draw()
{
  background(220)
  if (keyIsDown(LEFT_ARROW))
  {
    x -= 5
  }
  if (keyIsDown(RIGHT_ARROW))
  {
    x += 5
  }
  if (keyIsDown(UP_ARROW))
  {
    y -= 5
  }
  if (keyIsDown(DOWN_ARROW))
  {
    y += 5
  }
  circle(x, y, 50)
}
```

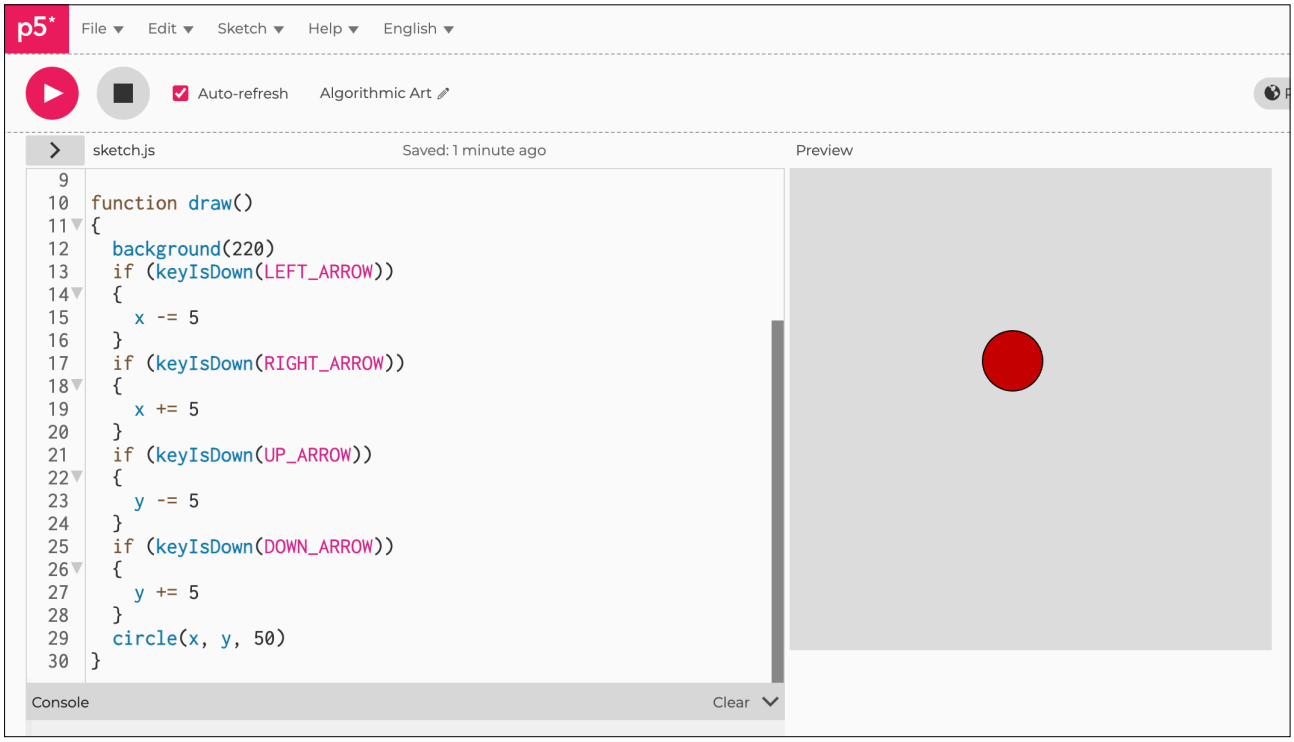
Notes

The red circle should move easily around the canvas.

Code Explanation

`if (keyIsDown(LEFT_ARROW))`	An example of calling a specific arrow key

Figure F1.7



Sketch F1.8 movedX

! Start a new sketch.
The variable movedX contains the horizontal movement of the mouse since the last frame.

```
let x = 200

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

function draw()
{
  background(220)
  if (x > width/2)
  {
    x -= 2
  }
  else if (x < width/2)
  {
    x += 2
  }
  x += movedX
  square(x, width/2, 50)
}
```

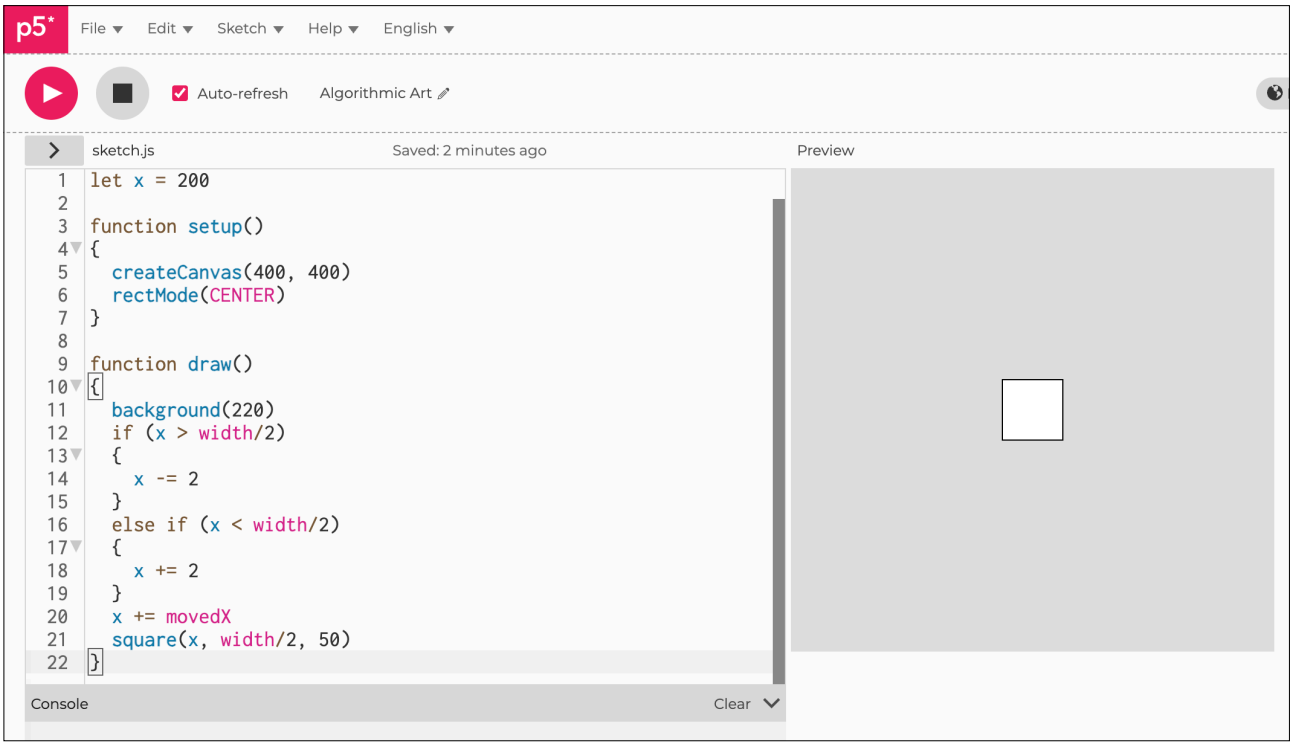
Notes

With this example, move your mouse to the left or right and then stop, you should see the square gently slide back This will also apply to movedY.

Code Explanation

x += movedX	It is a variable based on the distance each frame from its last position.

Figure F1.8



Sketch F1.9 winMouseX and pwinMouseX

! Start a new sketch.
This one is an interesting one with many things happening at once, where the pointer moves the canvas around, and the speed of the canvas determines the size of the circle.

```
let myCanvas
let speed

function setup()
{
  myCanvas = createCanvas(200, 200)
}

function draw()
{
  background(220)
  speed = winMouseX - pwinMouseX
  circle(width/2, height/2, 10 + speed * 5)
  myCanvas.position(winMouseX, winMouseY)
}
```

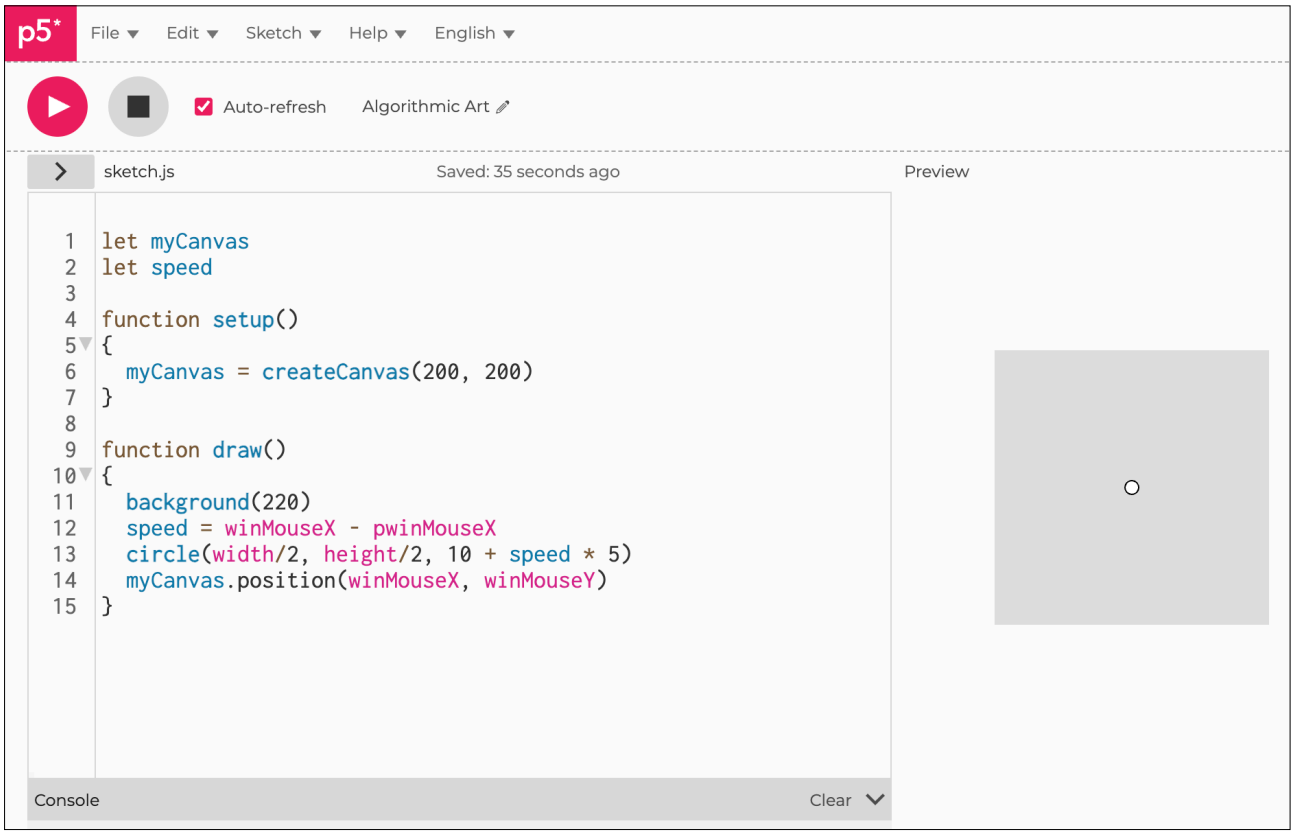
Notes

The speed variable is the distance between the current position of the mouse and the previous position (at any one time), so if you move it fast, then that distance is larger momentarily until the previous one catches up to the current.

Code Explanation

`myCanvas = createCanvas(200, 200)`	Create a new canvas object

Figure F1.9



Sketch F1.10 mouseIsPressed()

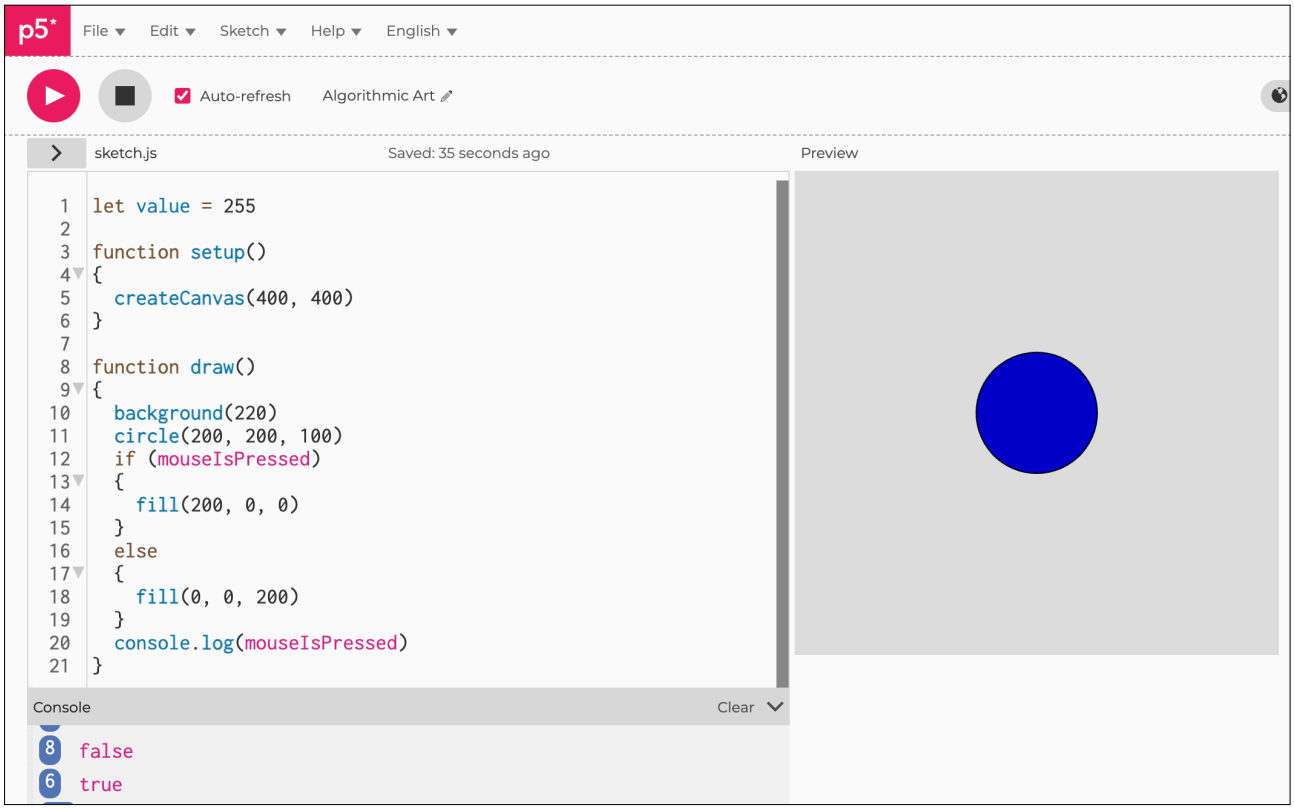
! Start a new sketch.
Simple responds when any mouse button is pressed.

```
let value = 255

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

function draw()
{
  background(220)
  circle(200, 200, 100)
  if (mouseIsPressed)
  {
    fill(200, 0, 0)
  }
  else
  {
    fill(0, 0, 200)
  }
  console.log(mouseIsPressed)
}
```

Notes

When you click on one of the mouse buttons, the circle turns red. The console records a true or false response to the mouse button.

Figure F1.10



Sketch F1.11 mouseButton()

This provides feedback depending on which button you press on your mouse (including the wheel).
! Remove console.log()

```
let value = 255

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

function draw()
{
  background(220)
  circle(200, 200, 100)
  if (mouseIsPressed)
  {
    if (mouseButton.left)
    {
      fill(200, 0, 0)
    }
    if (mouseButton.right)
    {
      fill(0, 200, 0)
    }
    if (mouseButton.center)
    {
      fill(0, 0, 200)
    }
  }
}
```

Notes

Depending on which button you click, it will change the colour of the circle accordingly. Useful if you need to identify which button is pressed.

Sketch F1.12 mouseMoved()

! A new sketch.
Move your mouse cursor across the canvas.

```
let value = 0

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

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

function mouseMoved()
{
  value += 10
  if (value > 255)
  {
    value = 0
  }
}
```

Notes

As you move the mouse, you get the effect of changing the greyscale colour of the square in degrees of 10.

Sketch F1.13 mouseDragged()

This time it only changes when the mouse button is held down as it is moved.

```
let value = 0

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

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

function mouseDragged()
{
  value += 10
  if (value > 255)
  {
    value = 0
  }
}
```

Notes

The same effect. This is a useful mouse function.

Sketch F1.14 mousePressed()

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

Just toggles between a black square and a white square.

Sketch F1.15 mouseReleased()

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

Almost the same effect.

Sketch F1.16 mouseClicked()

Very similar to mousePressed.

```
let value = 0

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

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

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

Notes

There is a very subtle difference. Useful in certain conditions.

Sketch F1.17 doubleClicked()

As it says on the tin.

```
let value = 0

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

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

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

Notes

Just another option, especially if you want one action for one click and something different for a double-click.

Sketch F1.18 mouseWheel()

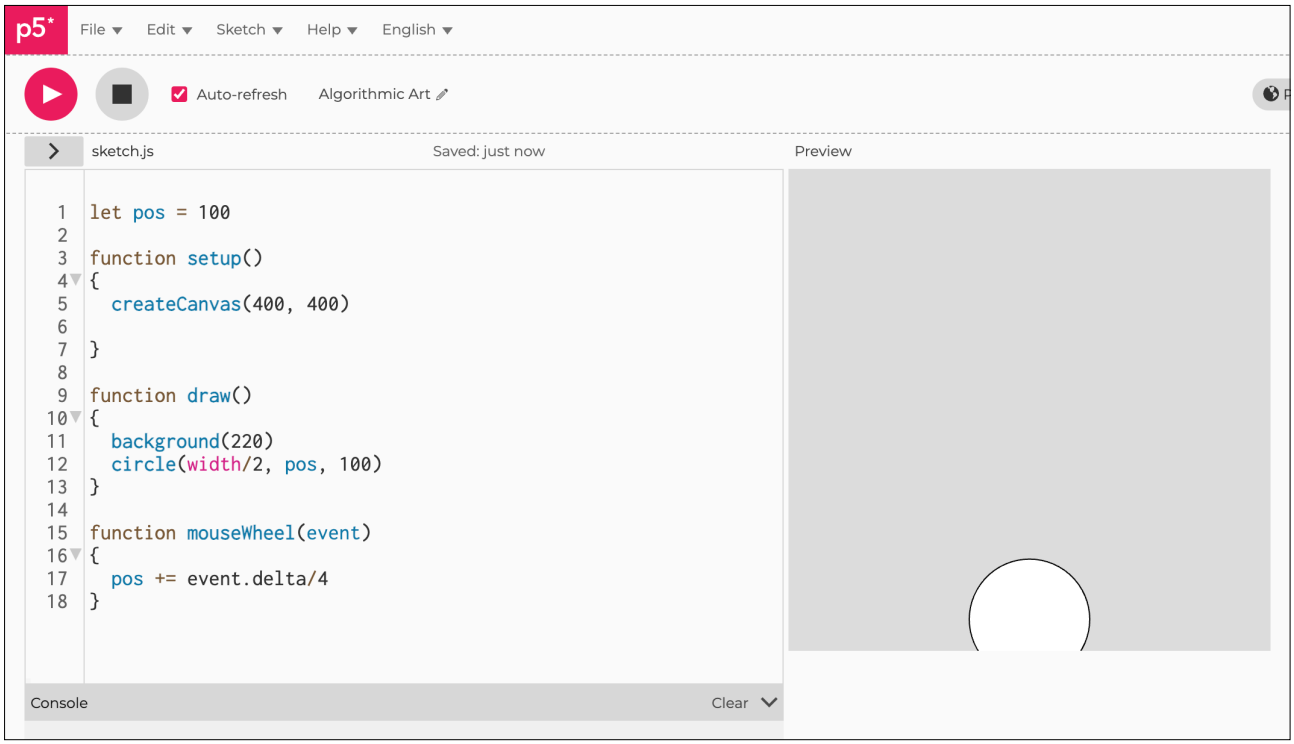
! A new sketch.
The event.delta property returns the amount the mouse wheel has scrolled.

```
let pos = 100

function setup()
{
  createCanvas(400, 400)

}

function draw()
{
  background(220)
  circle(width/2, pos, 100)
}

function mouseWheel(event)
{
  pos += event.delta/4
}
```

Notes

The circle moves up and down with the wheel of the mouse. The event.delta value can be altered to suit your mouse. I have divided by 4 just for a slightly slower response. You can change the direction using -= instead.

Figure F1.18



Sketch F1.19 easing

! A new sketch.
Smoothing out the motion.

```
let x = 0
let y = 0
let targetX
let targetY
let incrementX
let incrementY
let easing = 0.05

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

function draw()
{
  background(220)
  targetX = mouseX
  incrementX = targetX - x
  x += incrementX * easing
  targetY = mouseY
  incrementY = targetY - y
  y += incrementY * easing

  circle(x, y, 50)
}
```

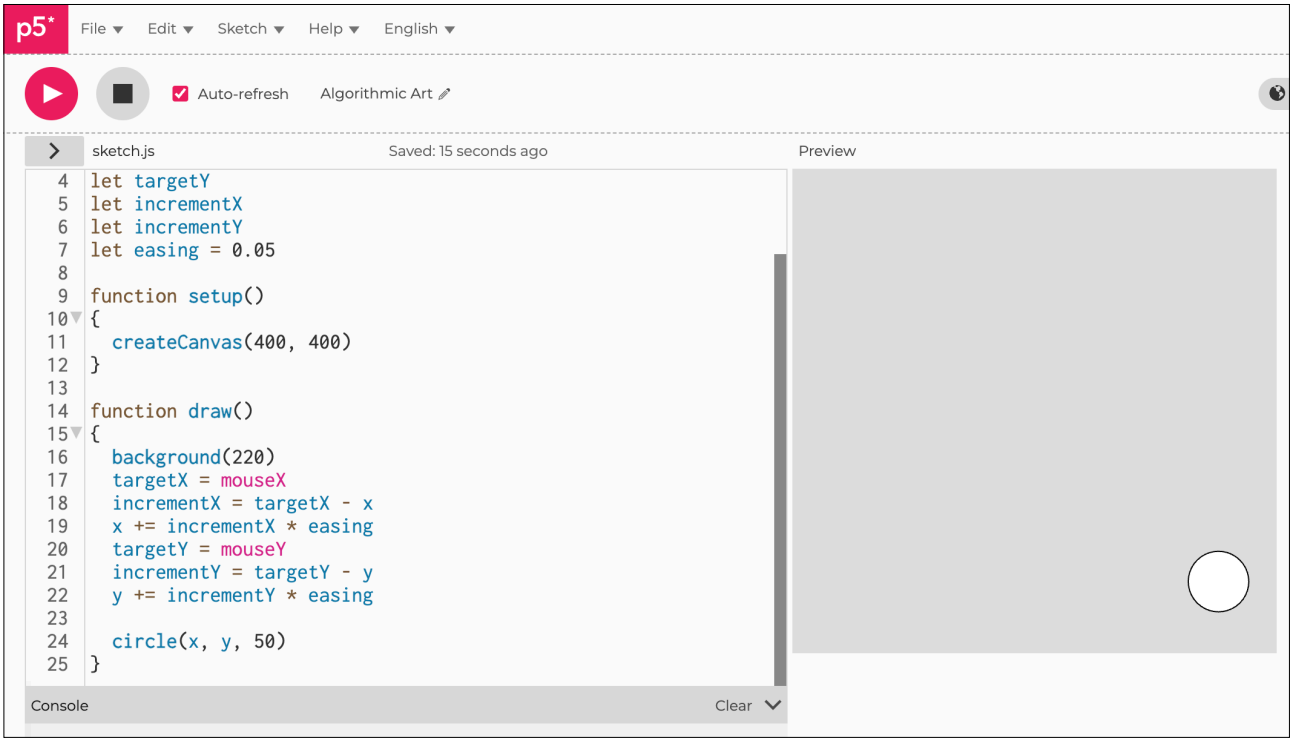
Notes

This is just a bit of fun to show how to make an object move smoothly. It could be applied to many things. The variable `easing` is not a function but just a variable name, but easing is the description of smoothing.

Challenge

Try making the value of easing = 1 and see what happens.

Figure F1.19




Algorithmic
Art
Module F
Unit #2
HTML

Module F Unit #2: HTML

In short, text boxes, sliders, and buttons.

This unit covers a topic broadly called DOM elements that you might use in a website involving CSS, HTML, and JavaScript. Here, we will be looking at the index.html file as well as the sketch.js file, so we will be giving them a separate heading to identify them like so...

sketch.js

index.html

You can access the list of files as shown below in Fig. 1. Just click on the arrow in the grey box, and the list of files will open up on the left-hand side. Then click on the tab for index.html, and you should get the following. You will be able to see the following files...

index.html
sketch.js
style.css

It defaults to sketch.js. Notice in the above image that it includes other sketches which you will learn to add and use later on. Don't worry at this stage what all the lines of code mean; they are there because you need them to run p5.js. Have a close look and see what you can glean. But for now, it is only sketch.js and index.html that we are bothered about.

Challenge

Click on the style.css tab

Sketch F2.1 starting sketch

Starting with this standard sketch of drawing a circle with a grey background.

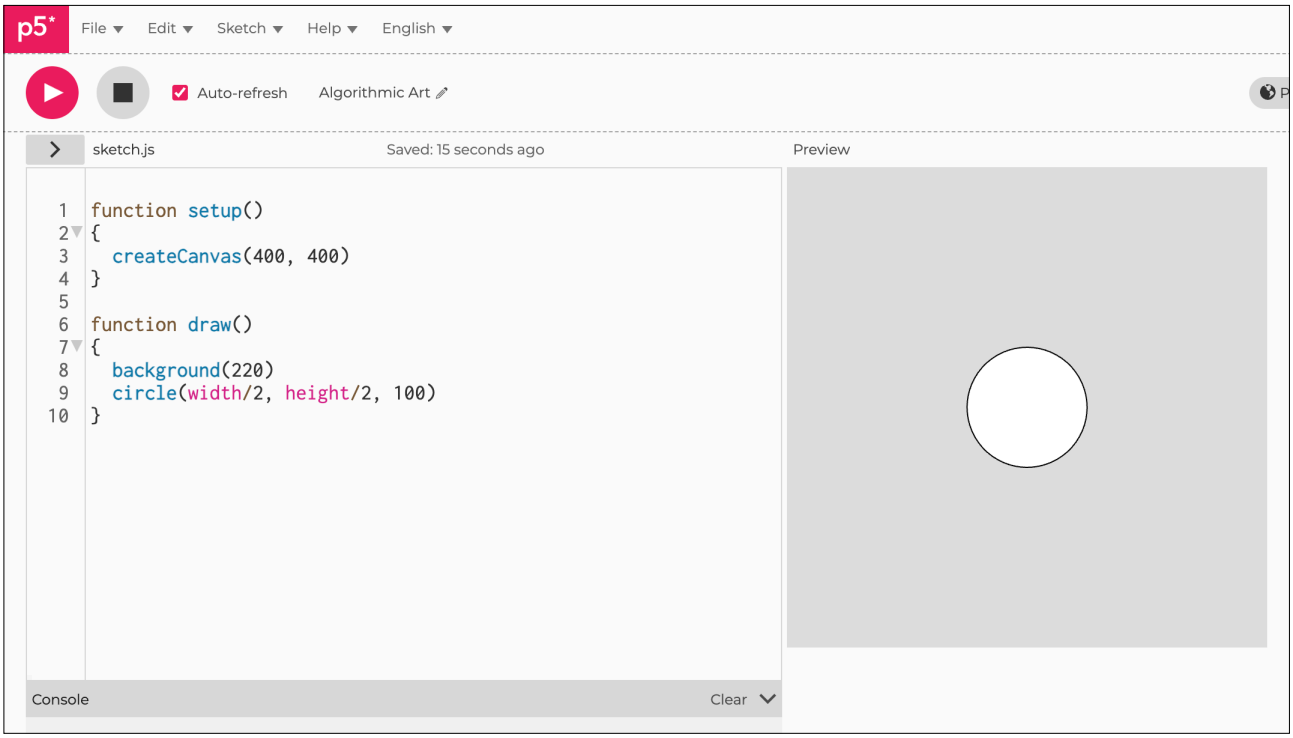
sketch.js

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

function draw()
{
  background(220)
  circle(width/2, height/2, 100)
}
```

Notes

Nothing very new here.

Figure F2.1



Sketch F2.2 creating html elements

Using the mousePressed() function to create an HTML event that is a header with the words "This is a h2 header". Also creates a paragraph element for a random number.

sketch.js

```
function setup()
{
  createCanvas(400, 400)
  createElement('h2', 'This a h2 header')
  createP('Click the mouse to generate a random number')
}


function mousePressed()
{
  createP('a random number ' + floor(random(100)))
}


function draw()
{
  background(220)
  circle(width/2, height/2, 100)
}
```

Notes

We have added `createElement()` which allows us to add header HTML in p5.js. It takes two arguments, `h1` to `h6`, and the text in speech marks. The `createP()` creates a paragraph. Notice that they are both added after the canvas, not in the canvas. You can easily interact with these DOM elements in the p5.js sketch.

Challenges

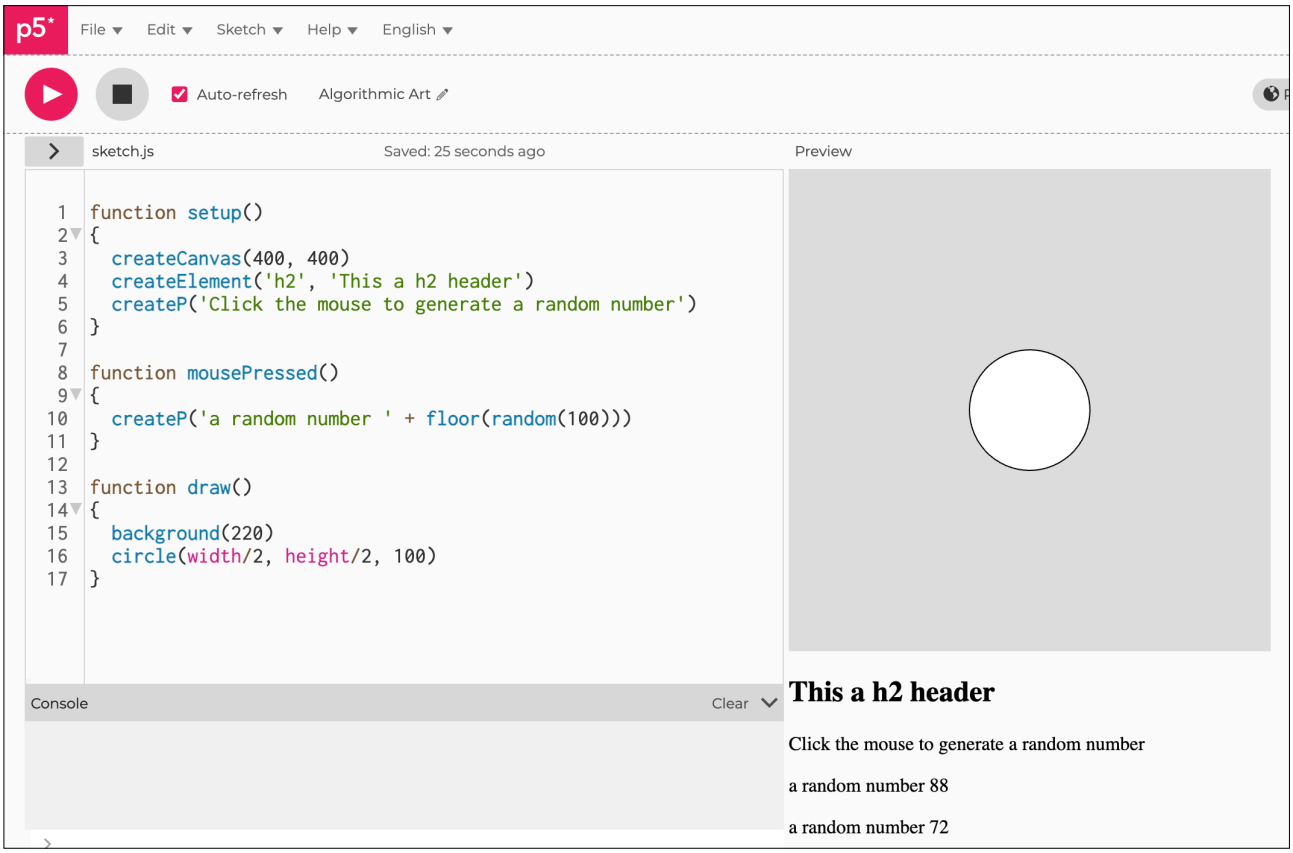
1.Create more interactive paragraph elements

2.Use more heading values, i.e. cycle through the six different sizes of headings - maybe using a loop.

Code Explanation

`createElement('h2', 'This a h2 header')`	Creates an element which is a header of size h2
`createP`	Creates a paragraph

Figure F2.2



Sketch F2.3 position

! It's quicker and easier to start a new sketch.
There are a number of fun things happening here. It is to give you a taster of what you can do with HTML code.

sketch.js

```
let canvas
let heading
let x = 100
let y = 100

function setup()
{
  canvas = createCanvas(400, 400)
  canvas.position(200, 200)
  heading = createElement('h1', 'click on the mouse')
}

function mousePressed()
{
  heading.html('I am feeling a bit wobbly')
}

function draw()
{
  background(220)
  circle(x, y, 100)
  heading.position(x, y)
  x += random(-2, 2)
  y += random(-2, 2)
}
```

Notes

In this example, you can set the position of the canvas separate from the DOM element. Although the wobbly header follows the same x, y, the circle is relative to the top left-hand corner of the canvas (0, 0), and the wobbly text is relative to the top left-hand corner of the preview window (browser).

You create a variable called heading (it can be called anything) so that it will store the createElement(h1, 'Click on the mouse'). This can then be changed later to 'I'm feeling a bit wobbly'.

Challenges

1. Try differently named variables for heading and canvas.
2. Could you create another event other than changing the heading?

Figure F2.3

Sketch F2.4 starting sketch

! A new starting sketch. In this bit, we are going to look at adding buttons.

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

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

Notes

Bog standard sketch.

Sketch F2.5 background colour

First off, we will create a variable for the background colour.

```
let backgroundColour

function setup()
{
  createCanvas(400, 400)
  backgroundColour = color(220)
}

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

Notes

This creates exactly the same grey background as we had in the previous sketch.

Sketch F2.6 adding a button

Here we create a button. We don't give it a position, so it defaults to the bottom of the canvas.

```
let backgroundColour
let button

function setup()
{
  createCanvas(400, 400)
  backgroundColour = color(220)
  button = createButton('clcik on this button')
}

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

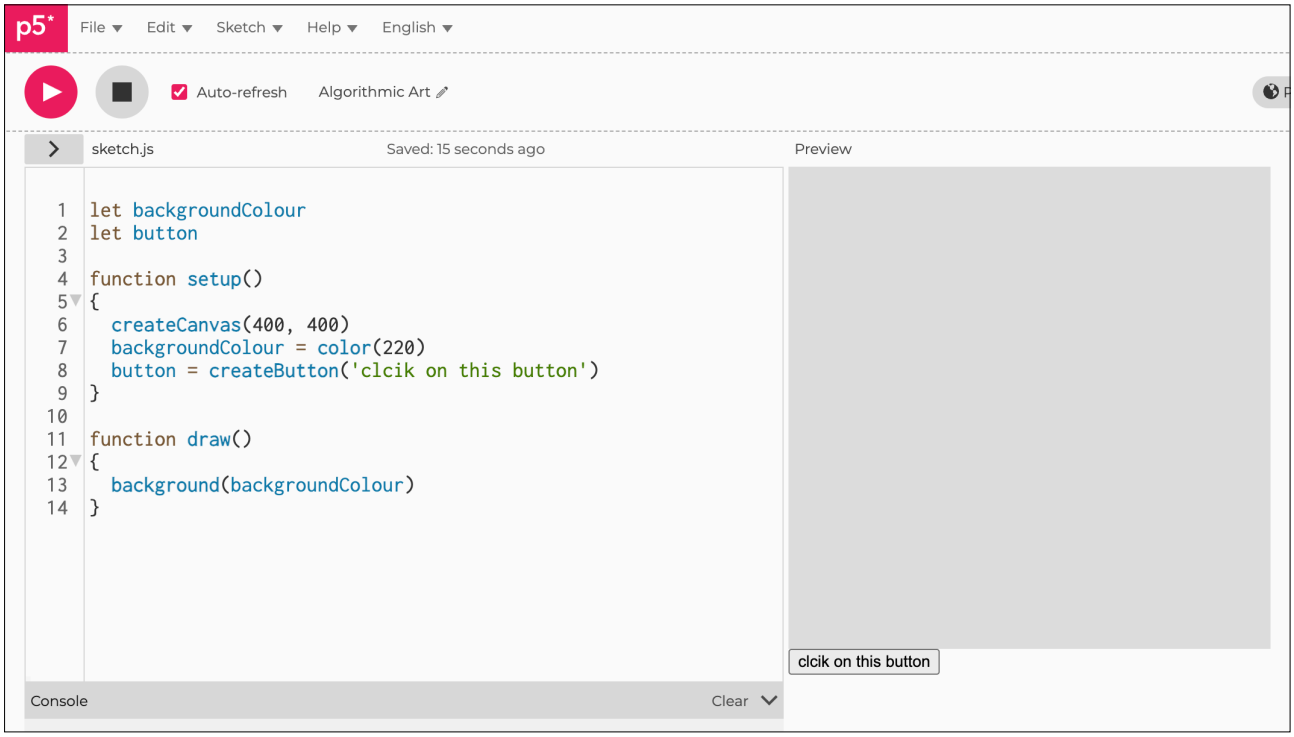
Notes

This button doesn't do anything yet because we need to give it a call back, in other words, give it something to do when it is clicked. Also, you can change the position, the font size, the colour, and even the button background colour, but that is for another day.

Challenge

Change the text.

Figure F2.6



Sketch F2.7 the callback

What we want to do in this instance is to change the background colour (random) every time we click on the button. For that, we need to create a function that is called when the button is pressed. We will call it `changeColour()`, isn't it original!

```
let backgroundColour
let button

function setup()
{
  createCanvas(400, 400)
  backgroundColour = color(220)
  button = createButton('clcik on this button')
  button.mousePressed(changeColour)
}

function changeColour()
{
  backgroundColour = color(random(255), random(255), random(255))
}

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

Notes

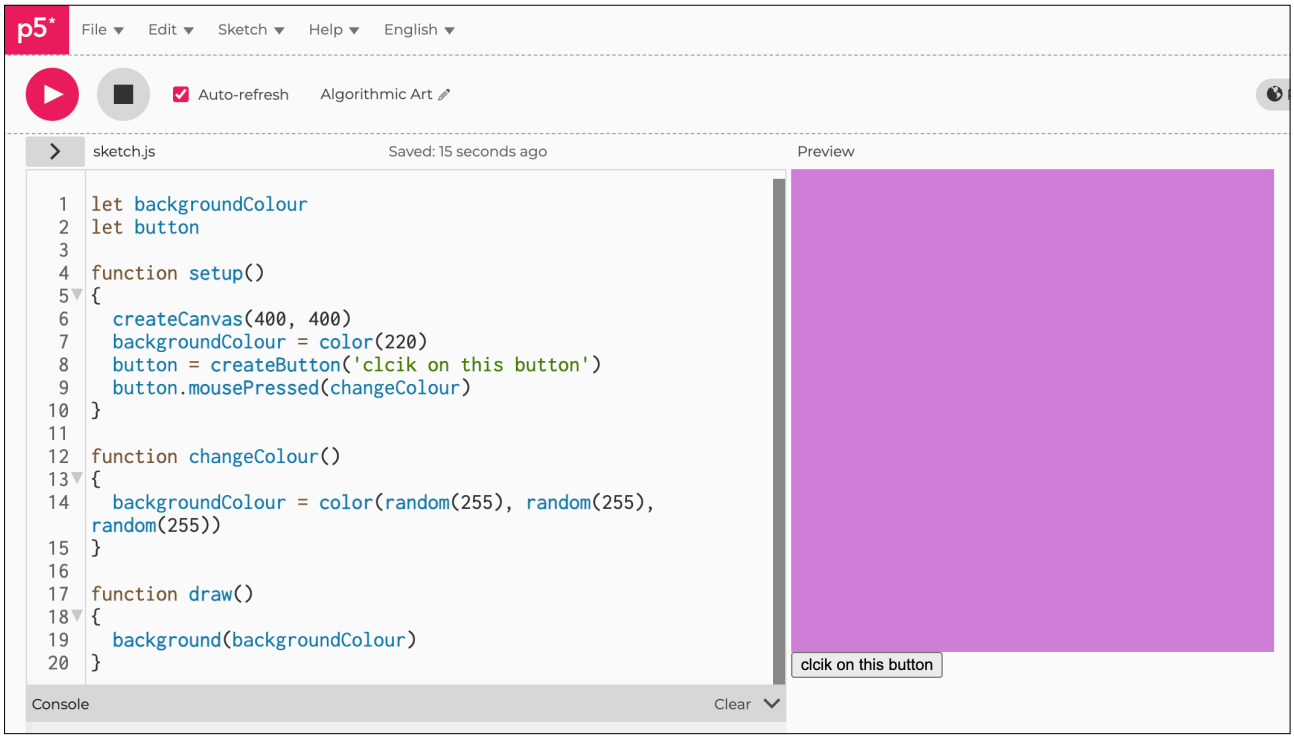
As you press the button, it calls a function called changeColour(). This creates a random background colour change, which is then executed in draw. So every time you click the button, it generates a new random background colour, which is sent to the draw function. The draw function loops continuously and updates the bicolour variable. The color() function is a special function that stores three arguments: the red, blue, and green elements.

Notice that the keyword in this color() is the American spelling; you cannot change that to the British spelling.

Challenges

1. Create another function on `button.mousePressed()` e.g. change the size or position of a circle.
2. Could you have a random number of circles or just add an extra one each time?

Figure F2.7



Sketch F2.8 slider

! Starting a new sketch.
We use the `createSlider()` function, and it returns a value depending on where the slider point is. The `createP()` function just creates the text instructions (a paragraph DOM element).

sketch.js

```
let slider

function setup()
{
  createCanvas(400, 400)
  rectMode(CENTER)
  createP('move the slider')
  slider = createSlider(1, 400, 100)
}

function draw()
{
  background(220)
  square(width/2, height/2, slider.value())
}
```

Notes

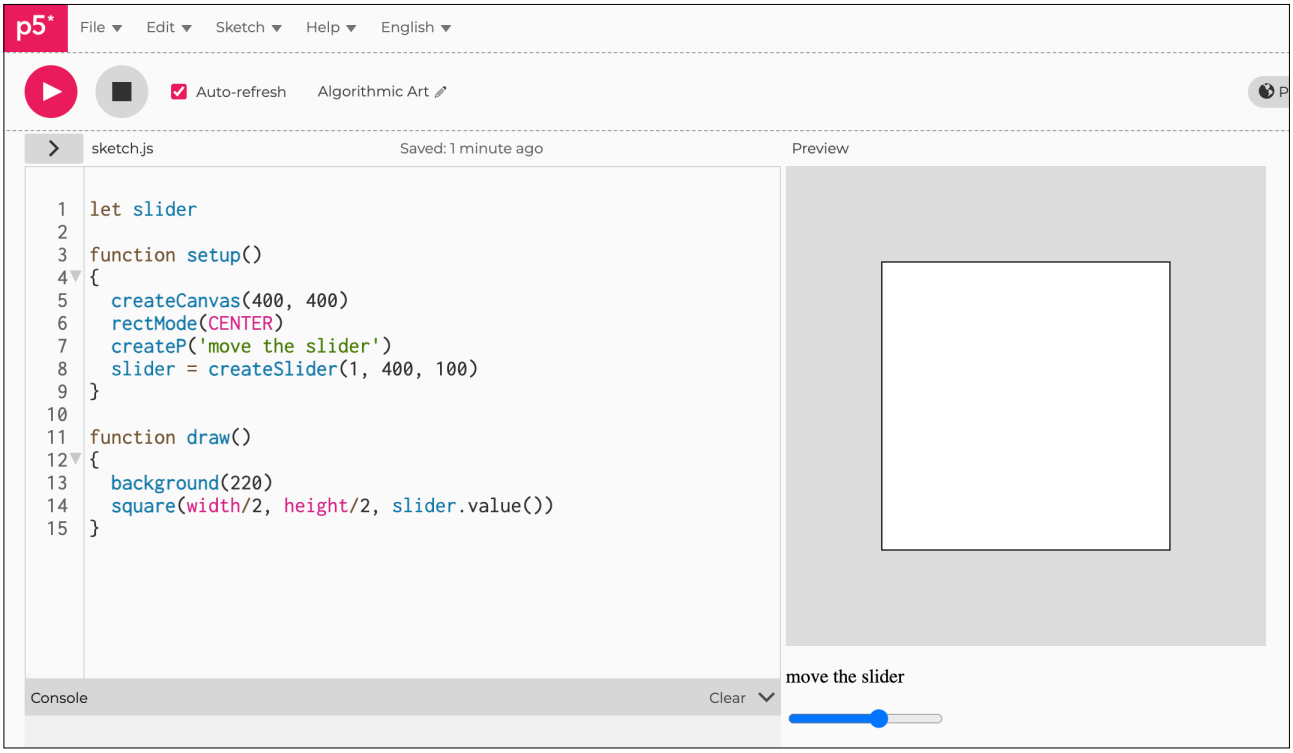
Using a slider DOM element. It has three arguments:
1. the first one is the minimum value,
2. the second is the maximum value, and
3. the third is the starting value.

It returns a single value between the minimum and maximum value as you slide the slider.

Challenges

1. Add more sliders for different effects.
2. Change the colour with the slider.

Figure F2.8



Sketch F2.9 text box

You can use text from a text box as an input.

sketch.js

```
let input

function setup()
{
  createCanvas(400, 400)
  createP('Type your name here')
  input = createInput()
  textSize(40)
}

function draw()
{
  background(220)
  text(input.value(), 10, 50)
}
```

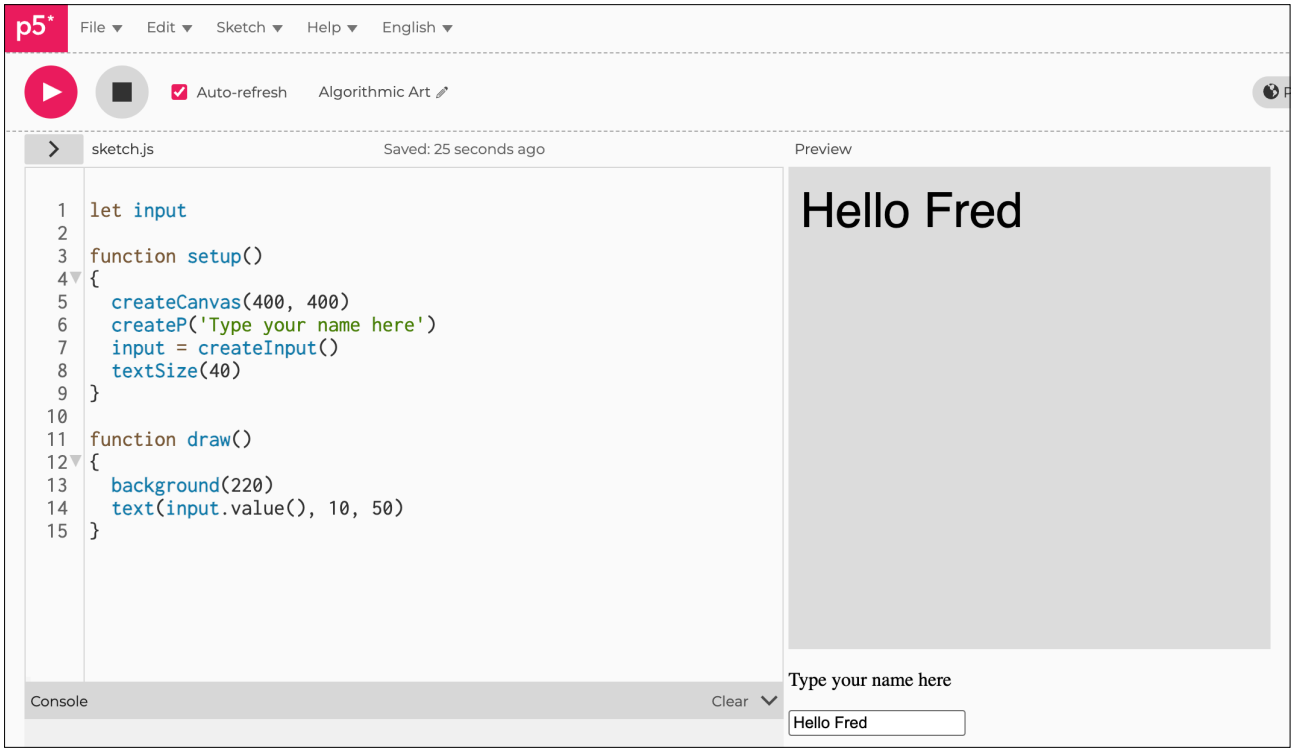
Notes

Creates a text box that you can enter text into. In this example, it prints it onto the canvas and into the window below the canvas. To handle the text, you create a variable called input and a variable called name. The input variable will hold the string from the inputted text. The name variable will adopt that and create a paragraph element using `createP()`.

Challenges

1. Make the text wobble as you type it in.
2. Add a button to make the text change colour.

Figure F2.9



Sketch F2.10 hovering over text

Using HTML to change the text

sketch.js

```
let textP

function setup()
{
  noCanvas()
  textP = createP('Hover your mouse here')
  textP.mouseOver(overpara)
  textP.mouseOut(outpara)
}

function overpara()
{
  textP.html('Hello')
}

function outpara()
{
  textP.html('Good Bye')
}
```

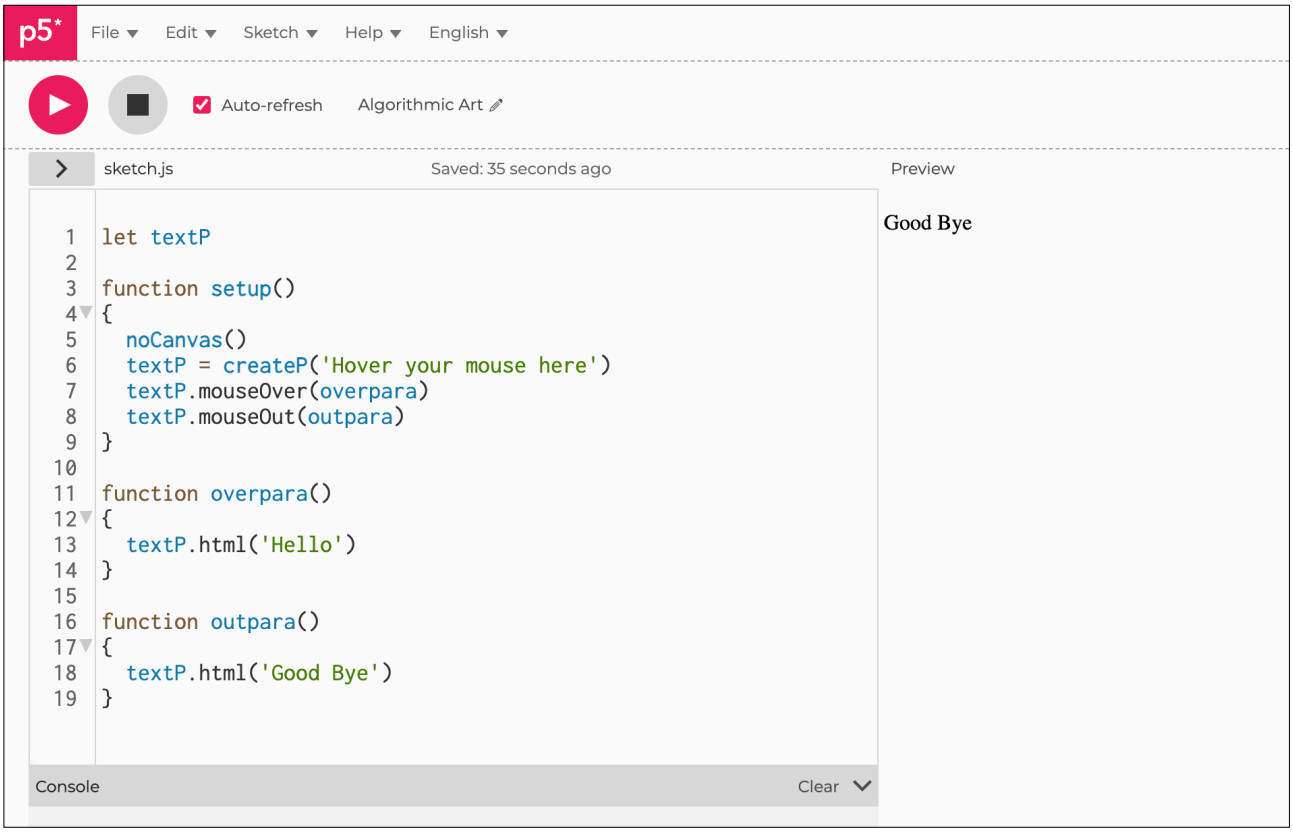
Notes

You start by creating a variable called `textP` which will store the text. The text will change when your mouse hovers over the text using the `mouseOver()` function. The reverse applies when you move your mouse away from over the text using the function `mouseOut()`. To facilitate this, you create two functions called `overpara()` and `outpara()`. The `noCanvas()` function just removes the canvas! Also, you don't need the `draw()` function.

Challenges

1. How would you create more paragraphs that change when you hover over them?
2. What else could you change when you hover over the paragraphs?

Figure F2.10



Sketch F2.11 hovering over the canvas

! Start a new sketch.
Three ways to change the colour of the background.

sketch.js

```
let backgroundColour

function setup()
{
  canvas = createCanvas(400, 400)
  canvas.mouseOver(changeColour)
  canvas.mouseOut(changeColour)
  canvas.mousePressed(changeColour)
  backgroundColour = color(220)
}

function changeColour()
{
  backgroundColour = color(random(255), random(255), random(255))
}

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

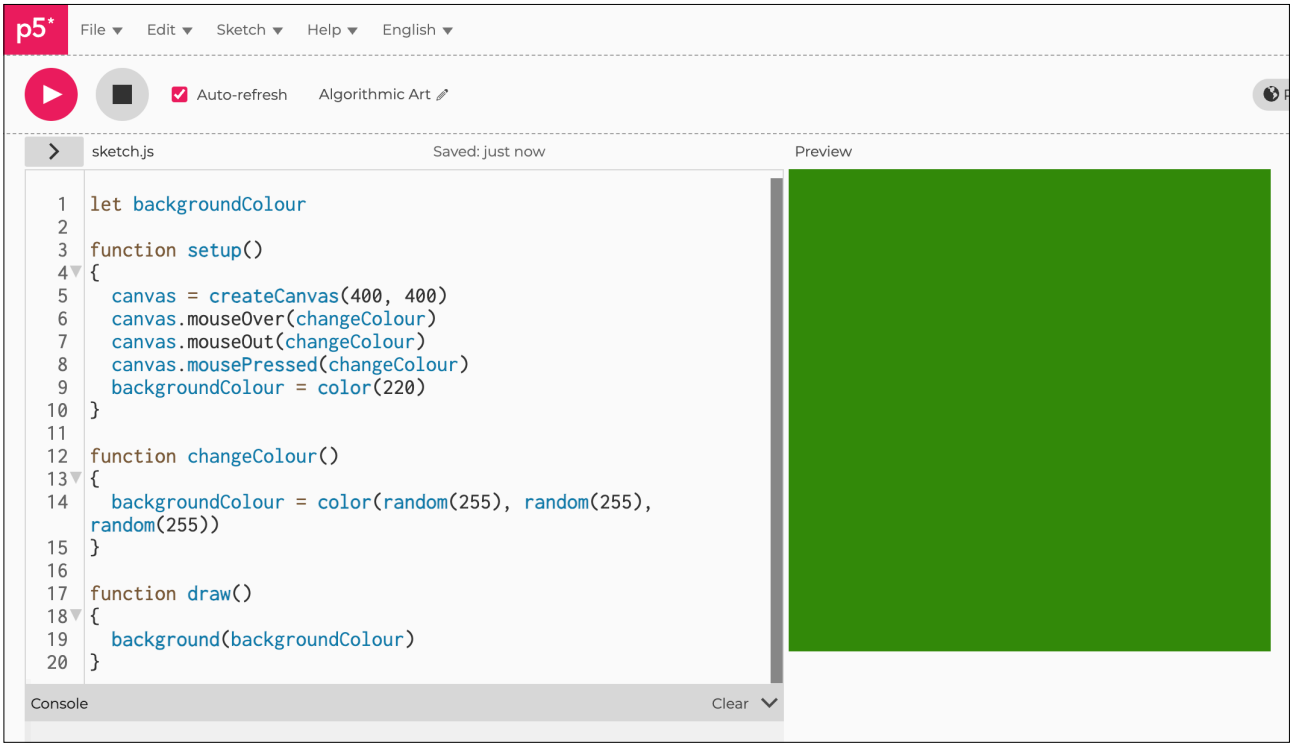
Notes

Here are three ways to change the colour of the background. Using `mouseOver()`, `mouseOut()`, and `mousePressed()`.

Challenges

1. How would you change just the colour of the circle using `mouseOver()`?
2. How would you change the colour when the mouse is released? There is a clue in the name.

Figure F2.11



Introduction to CSS

CSS stands for Cascading Style Sheets. It is the style that you see on a website, the colours, spacing, format, and general appearance.

You can have headers <h1>_____</h1>
Paragraphs _____

A great place to look at all the possibilities is w3schools.com.
This is essential if you want to build and design your own website and is an integral part of HTML working alongside JavaScript. The p5.js web editor allows you to see the results in the canvas, so I recommend that you explore this side of coding. Check out the CSS file and try something.

I would spend longer on this unit topic, but there is so much to it, and there is so much else to get through. Maybe another time I might add something about website building, only so many hours in a day, though.

Sketch F2.12 introduction to the index.html file

! Delete all the code in sketch.js and move over to the index.html file.
Using the index.html file, you can add headers and paragraphs using HTML markup. The <h1> always has a corresponding </h1> to close. Keep this index.html file as is for the next few sketches.

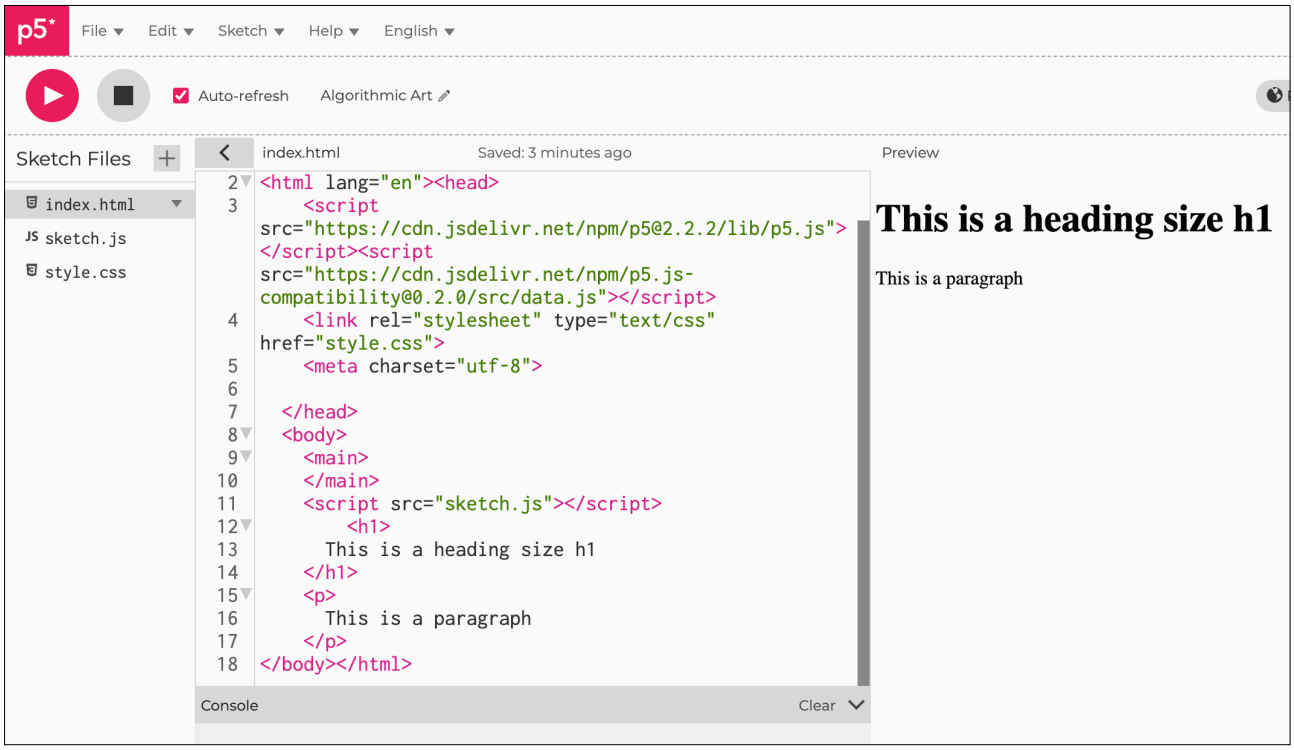
index.html

```
<!DOCTYPE html>
<html lang="en"><head>
    <script src="https://cdn.jsdelivr.net/npm/p5@2.2.2/lib/p5.js"></
script><script src="https://cdn.jsdelivr.net/npm/p5.js-compatibility@0.2.0/
src/data.js"></script>
    <link rel="stylesheet" type="text/css" href="style.css">
    <meta charset="utf-8">


  </head>
  <body>
    <main>
    </main>
    <script src="sketch.js"></script>
        <h1>
      This is a heading size h1
    </h1>
    <p>
      This is a paragraph
    </p>
</body></html>
```

Notes

This is the sort of coding you would see on a website.

Figure F2.12



Sketch F2.13 adding a sketch

! Move over to sketch.js.
The corresponding sketch.

sketch.js

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

function draw()
{
  background(220)
  circle(width/2, height/2, 100)
}
```

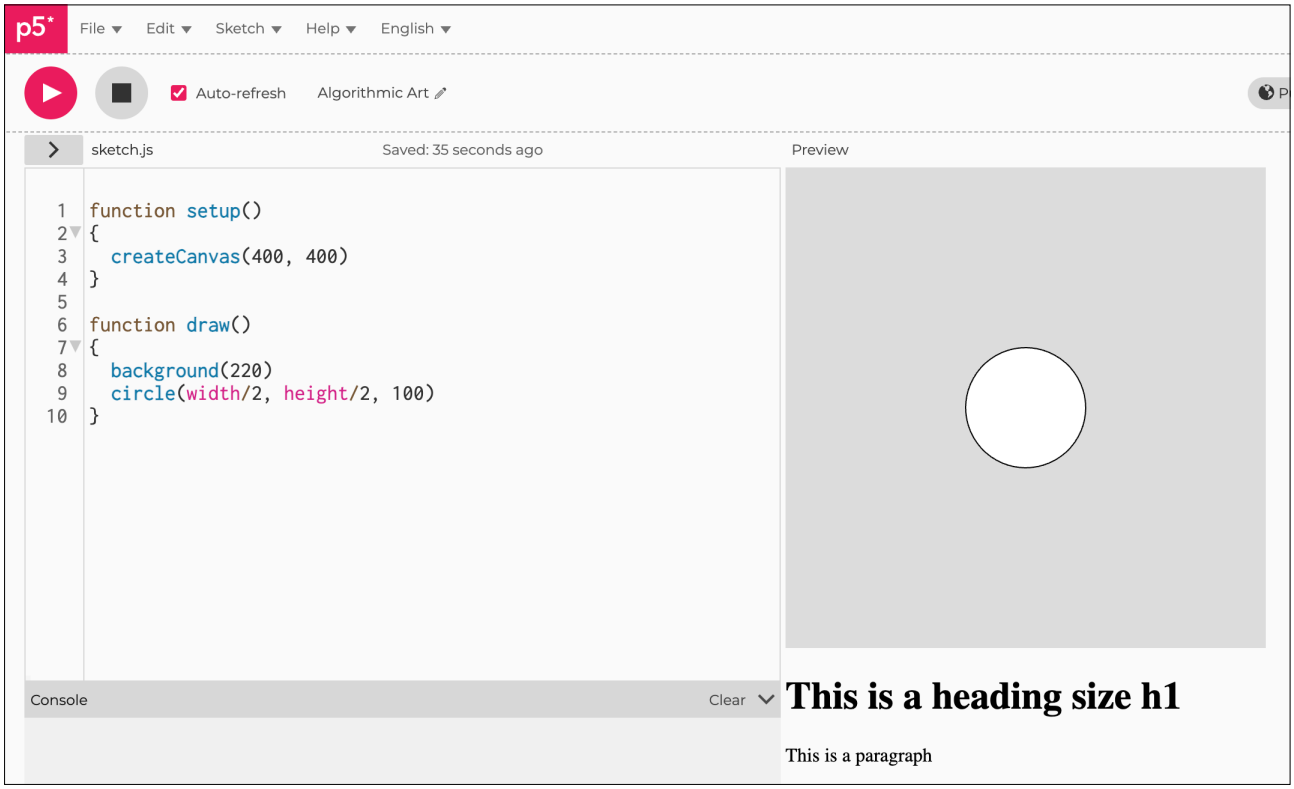
Notes

A simple example of putting a header and a paragraph in the `index.html`. The notation `<h1> </h1>` are the tags for header size 1. The `<p> </p>` tags are for a paragraph. Also note that the sketch is drawn first.

Challenges

1. Try using a different header, e.g. `<h3> </h3>` as well.
2. Add another paragraph.

Figure F2.13



Sketch F2.14 CSS changed

! Remove the draw() function and canvas (noCanvas())
To see the effect of the CSS on the text. You can describe the background colour and the padding space around the text in pixels (px).

sketch.js

```
let text

function setup()
{
  noCanvas()
  text = createP('Changing the CSS around the text')
  text.mouseOver(changeStyle)
  text.mouseOut(revertStyle)
}

function changeStyle()
{
  text.style('background-color', 'yellow')
  text.style('padding', '32px')
}

function revertStyle()
{
  text.style('background-color', 'orange')
  text.style('padding', '8px')
}
```

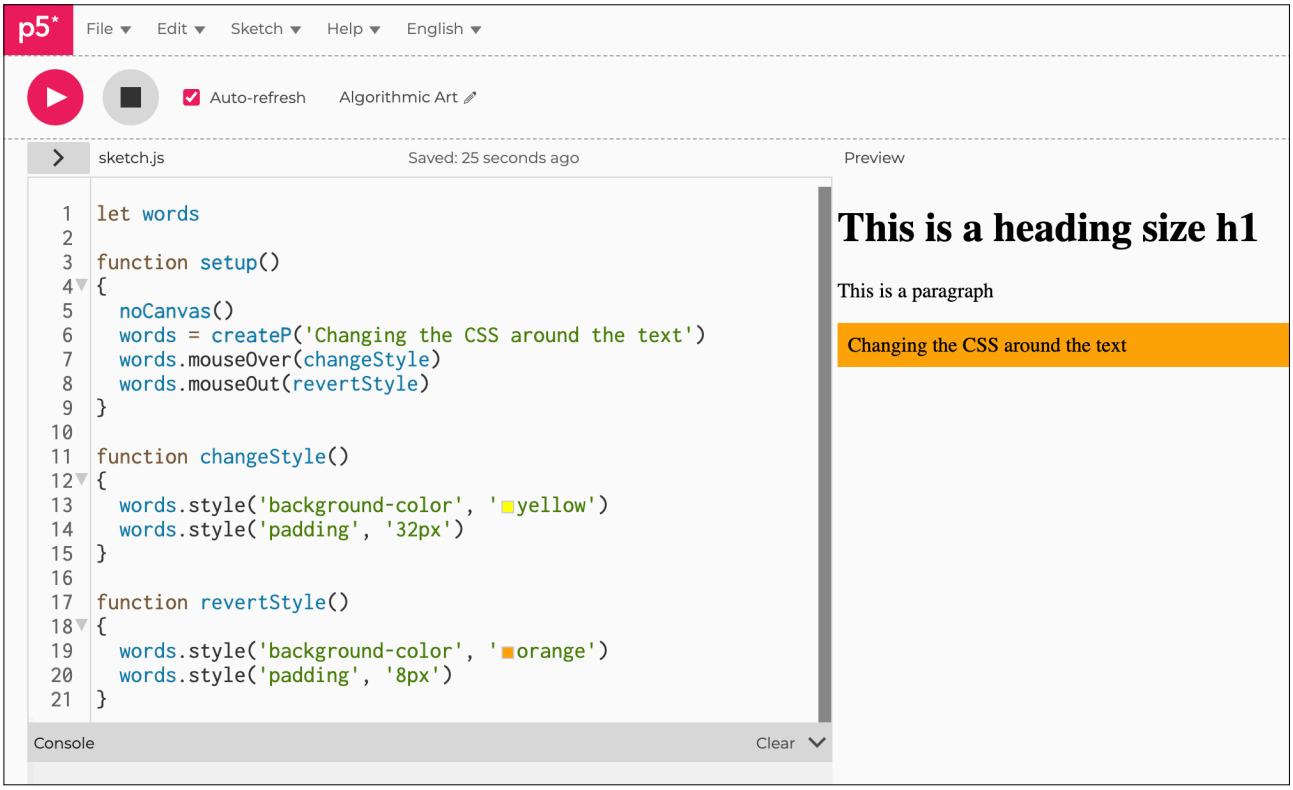
Notes

This demonstrates how you can interact with text and change the style as you hover over it. There are two elements changing the padding, which is the amount of space around the text and the background colour. The suffix px means pixels.

Challenges

1. Try different colours
2. Try different amounts of padding
3. Add more text

Figure F2.14



Sketch F2.15 submit button

In this next bit, we are going to create a submit button that will take data (your name) and push it onto the canvas.
! Start a new sketch, and also remove the HTML tags in the index.html file.
First, we create an input box to input our name.

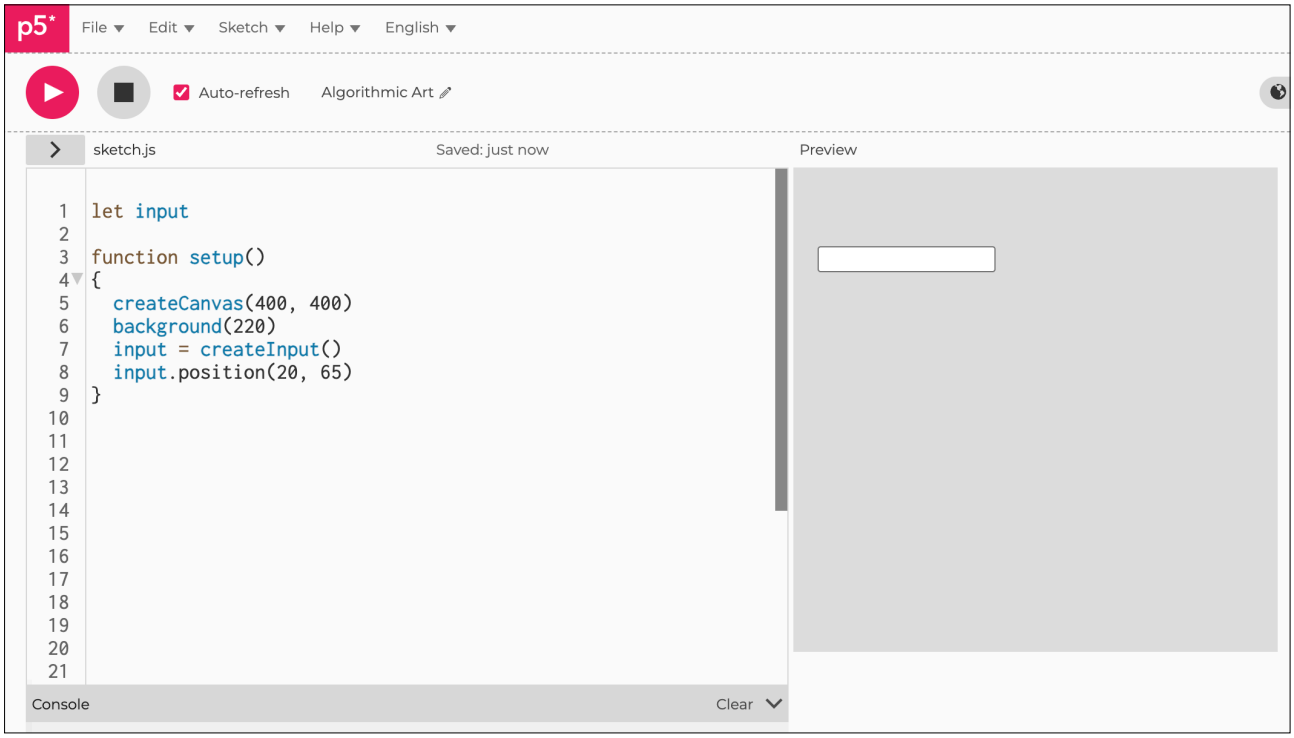
sketch.js

```
let input


function setup()
{
  createCanvas(400, 400)
  background(220)
  input = createInput()
  input.position(20, 65)
}
```

Notes

This does nothing yet, even though you can type into it.

Figure F2.15



Sketch F2.16 adding the button

We put the submit button next to the input box.

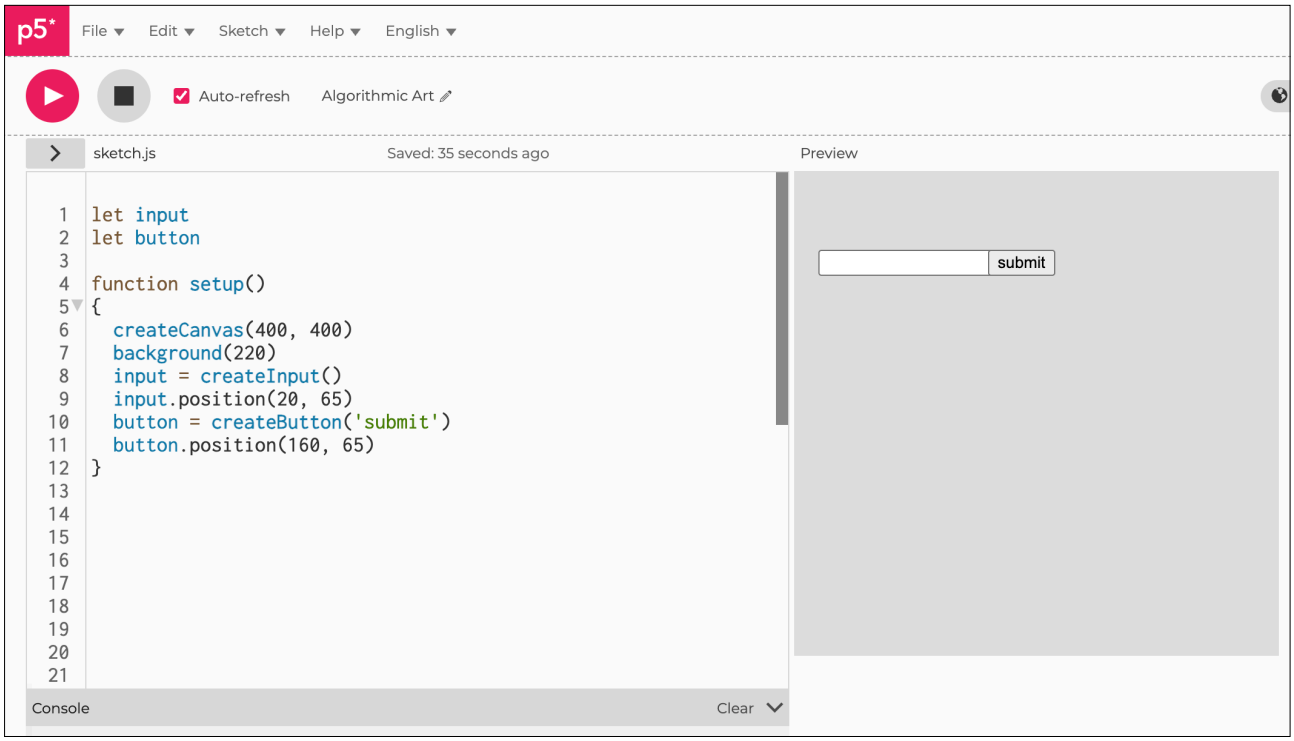
sketch.js

```
let input
let button

function setup()
{
  createCanvas(400, 400)
  background(220)
  input = createInput()
  input.position(20, 65)
  button = createButton('submit')
  button.position(160, 65)
}
```

Notes

This still does nothing.

Figure F2.16



Sketch F2.17 input function

Next, we create a function to take the input data (your name or whatever). This is a function we have called `inputName()`. The variable `name` takes the inputted name and puts it as text on the canvas.

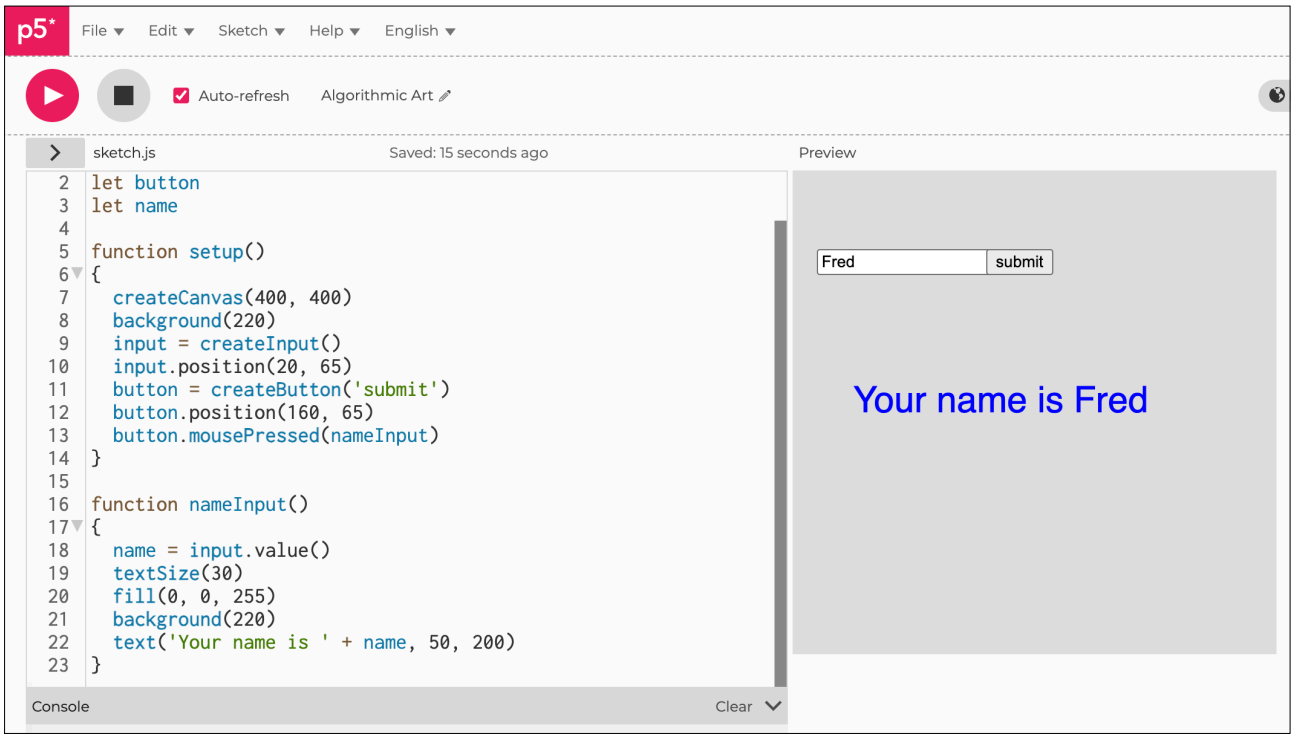
sketch.js

```
let input
let button
let name

function setup()
{
  createCanvas(400, 400)
  background(220)
  input = createInput()
  input.position(20, 65)
  button = createButton('submit')
  button.position(160, 65)
  button.mousePressed(nameInput)
}

function nameInput()
{
  name = input.value()
  textSize(30)
  fill(0, 0, 255)
  background(220)
  text('Your name is ' + name, 50, 200)
}
```

Notes

We draw the `background()` again to clear the canvas so you don't have to reload it each time.

Figure F2.17



Sketch F2.18 have the question

Finally, we can put a question on the canvas as a heading.

sketch.js

```
let input
let button
let name
let question

function setup()
{
  createCanvas(400, 400)
  background(220)
  input = createInput()
  input.position(20, 65)
  button = createButton('submit')
  button.position(160, 65)
  button.mousePressed(nameInput)
  question = createElement('h2', 'What is your name?')
  question.position(20, 5)
}

function nameInput()
{
  name = input.value()
  textSize(30)
  fill(0, 0, 255)
  background(220)
  text('Your name is ' + name, 50, 200)
}
```

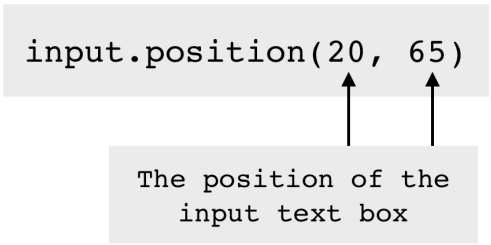
Notes

The benefit of using the DOM element as the question is that it doesn't get removed when the background is refreshed.

There is a lot happening here, but to summarise: you are creating a text box and a submit button. A text box is created when you use the `createInput()` function. You create a variable called `input` and when you type in your name, it stores it as a string. Because you are using an HTML element, you need to give it a position or else it appears at the bottom of the canvas. You create a button with the function `createButton()` and you put any text which you want to appear on the button inside the brackets with speech marks.

When the button is pressed, it calls a function we have created called function `nameInput()`. This is the function where we do something with the name we have stored. We pull that information and put it in a variable called name and use it to write text on the canvas.

The `createElement('h2', 'Type you name')` in this case `h2` is heading 2 which is the second biggest and down to `h6`, the smallest heading.



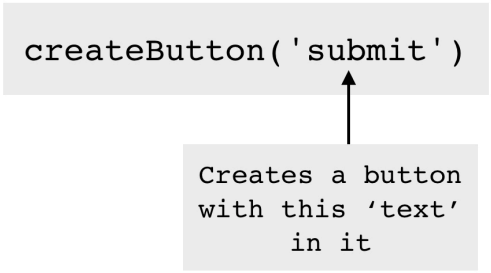
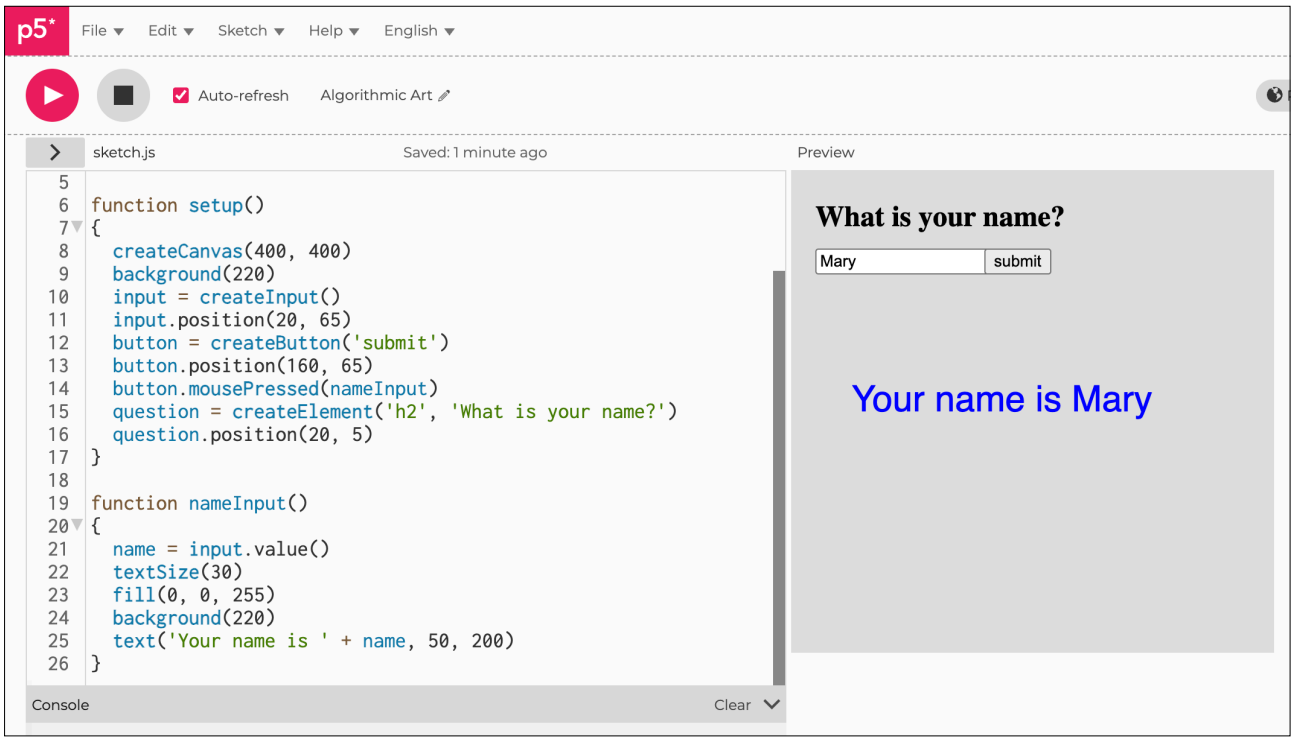


Figure F2.18


p5*
File
Edit
Sketch
Help
English
Auto-refresh
Algorithmic Art
sketch.js
Saved: 1 minute ago
Preview
function setup()
{
createCanvas(400, 400)
background(220)
input = createInput()
input.position(20, 65)
button = createButton('submit')
button.position(160, 65)
button.mousePressed(nameInput)
question = createElement('h2', 'What is your name?')
question.position(20, 5)
}
function nameInput()
{
name = input.value()
textSize(30)
fill(0, 0, 255)
background(220)
text('Your name is ' + name, 50, 200)
}
Console
Clear
What is your name?
Mary
submit
Your name is Mary

Sketch F2.19 simple calculator

! Totally new sketch.
Input two numbers that are then multiplied.

sketch.js

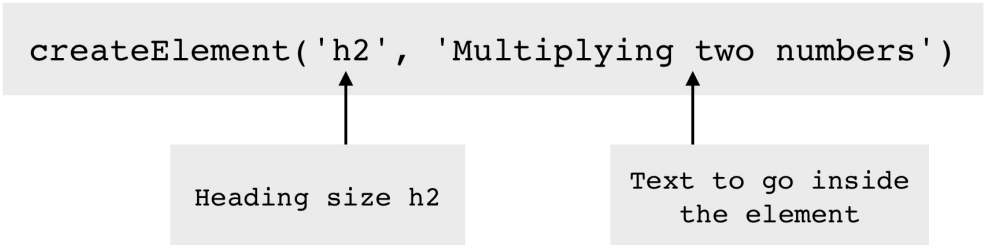
```
let input1
let input2
let button
let total
let title
let num1
let num2

function setup()
{
  createCanvas(400, 400)
  background(220)
  input1 = createInput('')
  input1.position(20, 65)
  button = createButton('submit')
  button.position(60, 130)
  button.mousePressed(number)
  input2 = createInput('')
  input2.position(20, 100)

  title = createElement('h2', 'Multiplying two numbers')
  title.position(20, 5)
  textSize(20)
}

function number()
{
  background(220)
  num1 = input1.value()
  num2 = input2.value()
  total = num1 * num2
  text('The answer is ' + total, 100, 200)
}
```

Notes

The beauty of this application is that you can start to collect data, information, and manipulate it. Here we are making a very simple calculator. The two numbers are multiplied together. However, if you were to change the `*` (multiply) to a `+` (addition), you will notice that it treats them as strings and writes the one after the other. This is because we should've made sure that the variables are read as integers, not strings (by default). The solution is to change these two lines of code...

```
num1 = int(input1.value())
num2 = int(input2.value())
```

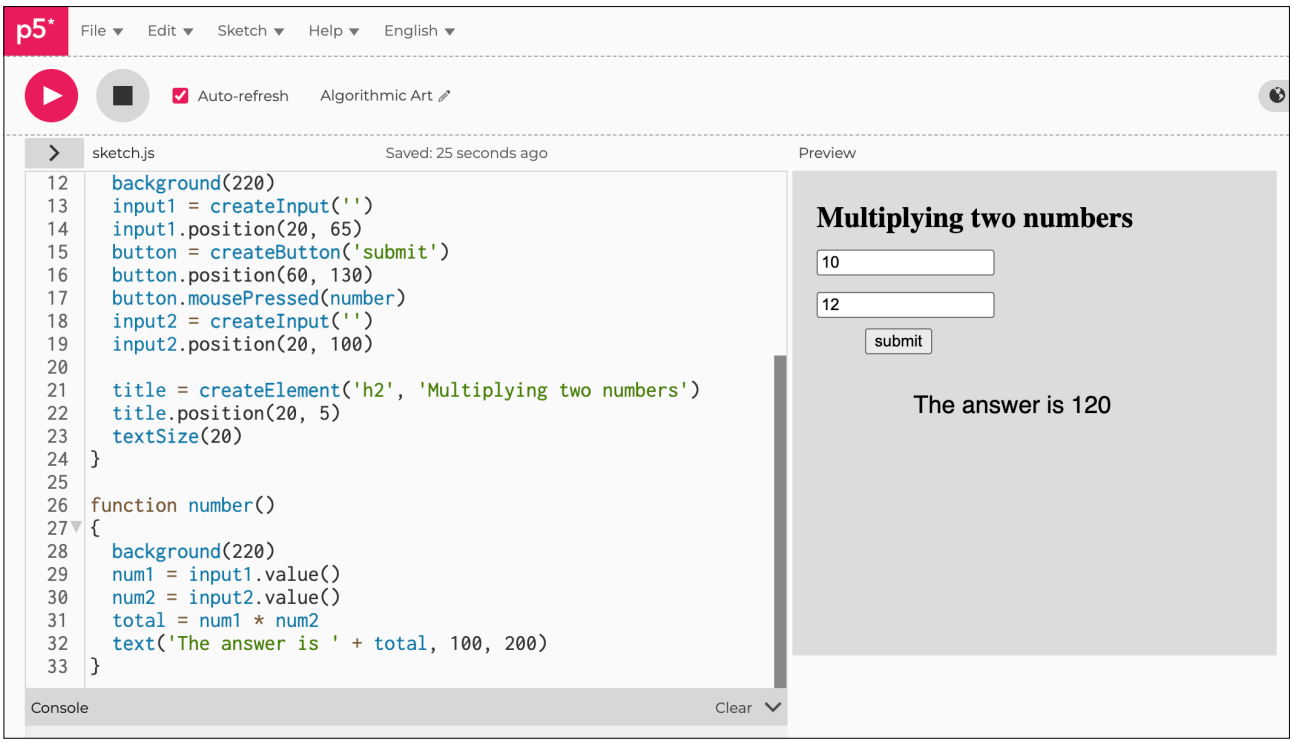
The `createElement()` has a heading `h2`. If there is a lot of text, then replace the `h2` with `p` for paragraph.



Challenge

Add more functionality with addition, subtraction, or division.

Figure F2.19


p5*
File
Edit
Sketch
Help
English
Auto-refresh
Algorithmic Art
sketch.js
Saved: 25 seconds ago
Preview
12 background(220)
13 input1 = createInput('')
14 input1.position(20, 65)
15 button = createButton('submit')
16 button.position(60, 130)
17 button.mousePressed(number)
18 input2 = createInput('')
19 input2.position(20, 100)
20
21 title = createElement('h2', 'Multiplying two numbers')
22 title.position(20, 5)
23 textSize(20)
24 }
25
26 function number()
27 {
28 background(220)
29 num1 = input1.value()
30 num2 = input2.value()
31 total = num1 * num2
32 text('The answer is ' + total, 100, 200)
33 }
Console
Clear
Multiplying two numbers
10
12
submit
The answer is 120

Sketch F2.20 slider colour circle

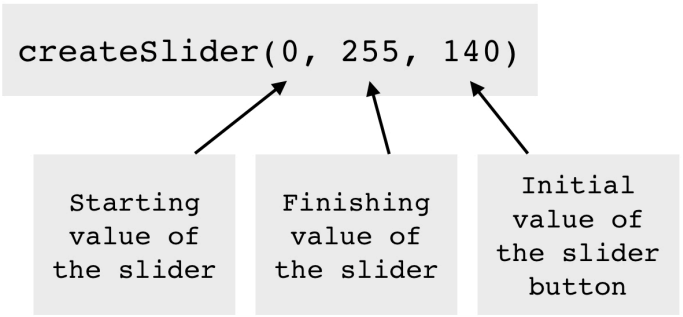
! Another new sketch.
Three RGB sliders so that you can create your own colour chart.

sketch.js

```
let sliderRed
let sliderGreen
let sliderBlue

function setup()
{
  createCanvas(400, 400)
  sliderRed = createSlider(0, 255, 140)
  sliderGreen = createSlider(0, 255, 140)
  sliderBlue = createSlider(0, 255, 140)
}

function draw()
{
  background(220)
  fill(sliderRed.value(), sliderGreen.value(), sliderBlue.value())
  textSize(50)
  circle(width/2, height/4, 100)
  textSize(30)
  fill(0)
  text('red: '+ sliderRed.value(), 200, 250)
  text('green: '+ sliderGreen.value(), 200, 300)
  text('blue: '+ sliderBlue.value(), 200, 350)
  sliderRed.position(50, 250)
  sliderGreen.position(50, 300)
  sliderBlue.position(50, 350)
}
```

Notes

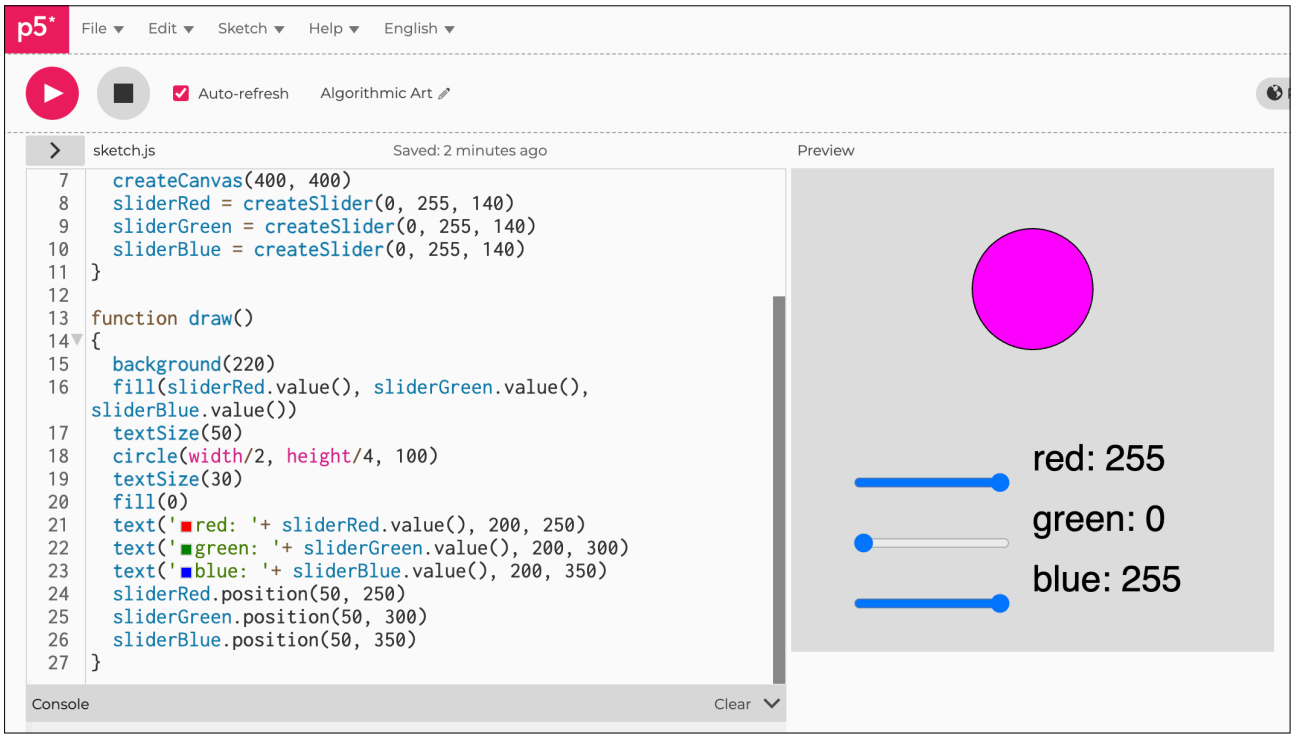
Here we can create a slider to change the colour of the circle. The line `createSlider(0, 255, 140)` gives us three arguments. The first one `(0)` is the minimum value, the second `(255)` is the maximum value, and the third `(140)` is the starting position of the slider button between the max and the min. The `sliderRed.value()` gets the value of the slider element and is then used for the amount of red colour in this instance; the value is also printed on the canvas.



Challenge

Add an extra slider for diameter.

Figure F2.20


p5*
File
Edit
Sketch
Help
English
Auto-refresh
Algorithmic Art
sketch.js
Saved: 2 minutes ago
Preview
createCanvas(400, 400)
sliderRed = createSlider(0, 255, 140)
sliderGreen = createSlider(0, 255, 140)
sliderBlue = createSlider(0, 255, 140)
}
function draw()
{
background(220)
fill(sliderRed.value(), sliderGreen.value(),
sliderBlue.value())
textSize(50)
circle(width/2, height/4, 100)
textSize(30)
fill(0)
text('red: '+ sliderRed.value(), 200, 250)
text('green: '+ sliderGreen.value(), 200, 300)
text('blue: '+ sliderBlue.value(), 200, 350)
sliderRed.position(50, 250)
sliderGreen.position(50, 300)
sliderBlue.position(50, 350)
}
Console
Clear
red: 255
green: 0
blue: 255

Sketch F2.21 slider rotate

! Yet another new sketch.
Rotating a baton with the slider, a nice demonstration.

sketch.js

```
let sliderRotate
let angle = 0

function setup()
{
  createCanvas(400, 400)
  sliderRotate = createSlider(0, 360, 0)
  strokeWeight(5)
  angleMode(DEGREES)
}

function draw()
{
  background(220)
  translate(width/2, height/2)
  rotate(angle)
  rectMode(CENTER)
  line(-100, -100, 100, 100)
  sliderRotate.position(50, 350)
  angle = sliderRotate.value()
}
```

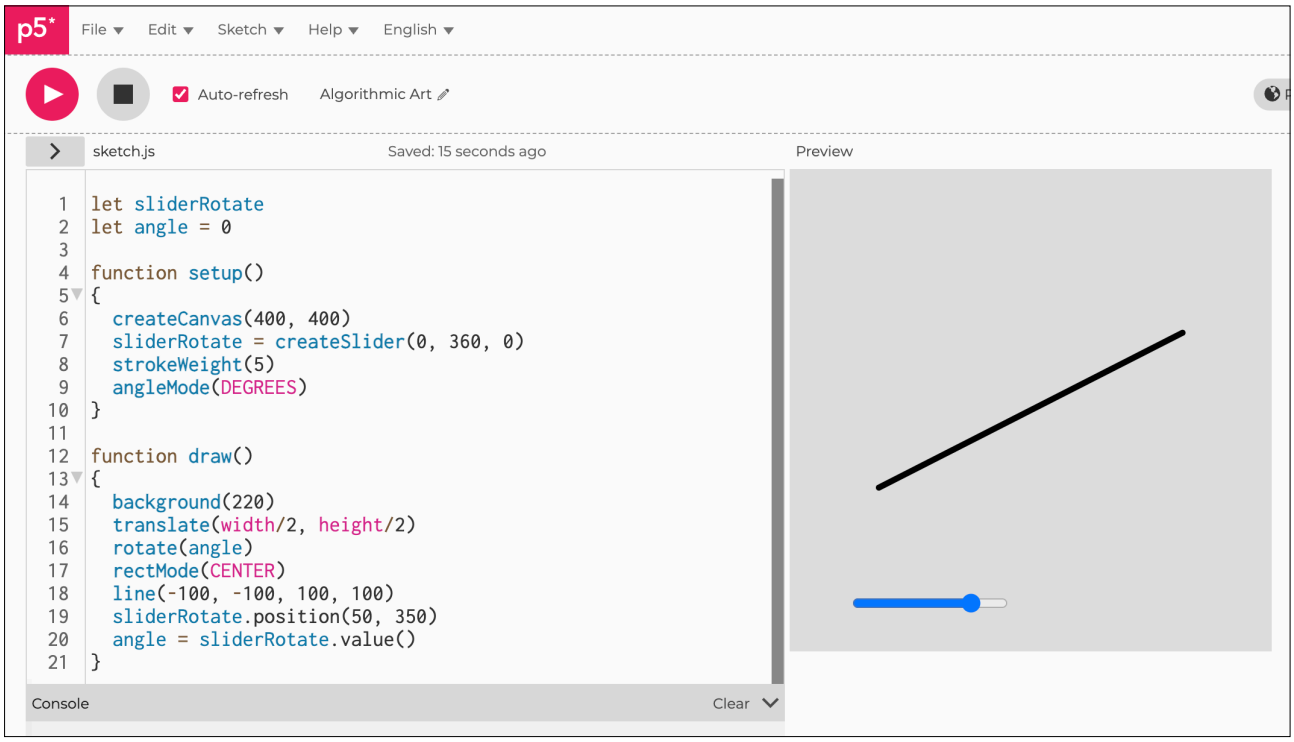
Notes

Another use of the slider value, in this instance to spin a line.

Challenge

Can you think of any other applications or fun things you could control?

Figure F2.21



Sketch F2.22 radio buttons colour

Using a predefined button to change the colour of the circle.

sketch.js

```
let radio
let val = 255

function setup()
{
  createCanvas(400, 400)
  radio = createRadio()
  radio.position(10, 10)
  radio.option('1', 'red')
  radio.option('2', 'green')
  radio.option('3', 'blue')
}

function draw()
{
  background(220)
  circle(width/2, height/2, 100)
  val = radio.value()
  if (val == 1)
  {
    fill(200, 0, 0)
  }
  if (val == 2)
  {
    fill(0, 200, 0)
  }
  if (val == 3)
  {
    fill(0, 0, 200)
  }
}
```

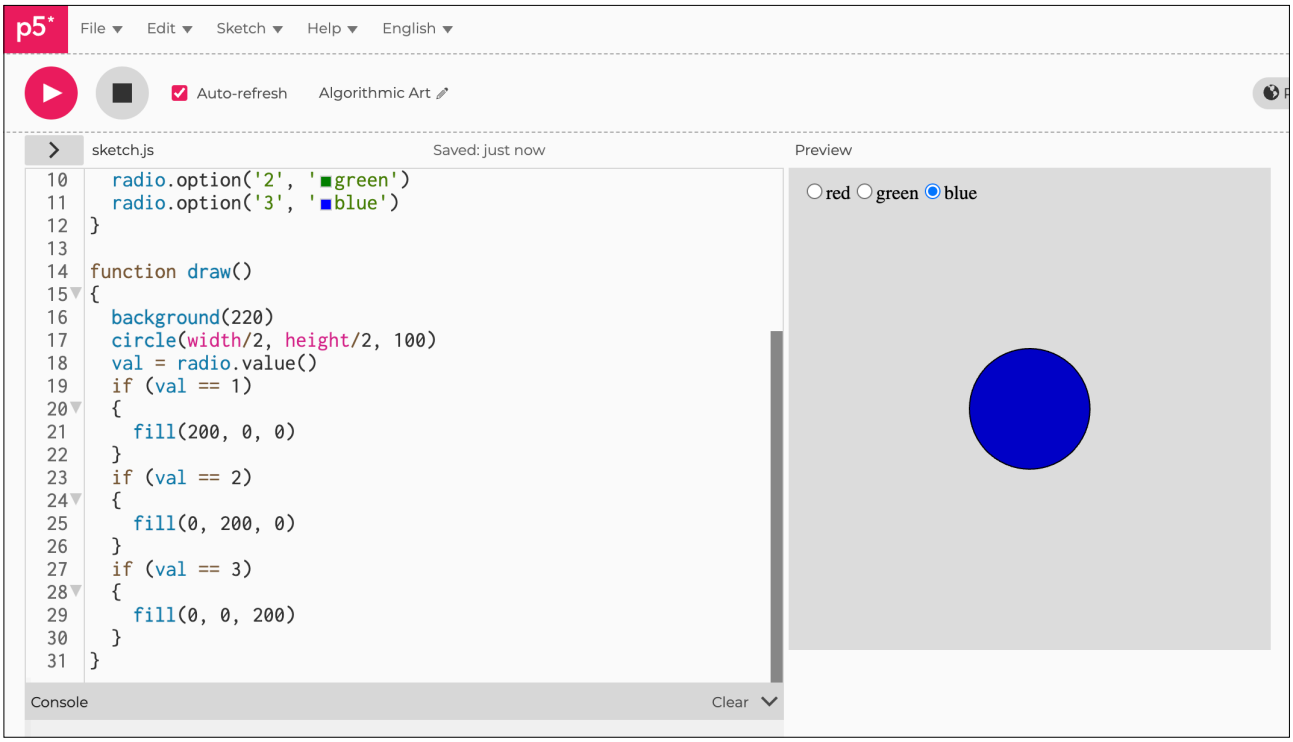
Notes

Three buttons to change the colour of the circle. Very simple but also very powerful if you want to use the data. Adds to the interactivity of your canvas, page, or website.

Challenges

1. Remove `radio.position(10, 10)`
2. Create shapes instead of colours
3. Have an event happen, e.g. `10` random bubbles
4. Use the to create a paint package

Figure F2.22




Algorithmic
Art
Module F
Unit #3
Video
Capture

Module F Unit #3: video capture

This section explores video and images. For this, you will need a webcam; most machines have them built in. Using the webcam, you can create and manipulate images and video. There is also a brief section on creating `.png` images that have transparency.

This brings in two very useful topics that have many applications. If nothing else, it is fun. For the vision, you will need a built-in webcam or attach one via a USB port. Once you have run the programme, you will need to give p5.js permission to access the webcam.

Sketch F3.1 scale

The scale function is useful in all manner of things, but in particular, it enables us to reverse the image from the webcam.

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

function draw()
{
  background(220)
  noStroke()
  scale(1, 2)
  square(50, 50, 100)
}
```

Notes

The `scale()` function can take one or two arguments. In the example above, it takes two arguments, one for the `x` and one for the `y` direction. In this case, it keeps the `x` direction as is, but the `y` direction is doubled, multiplied by `2`.

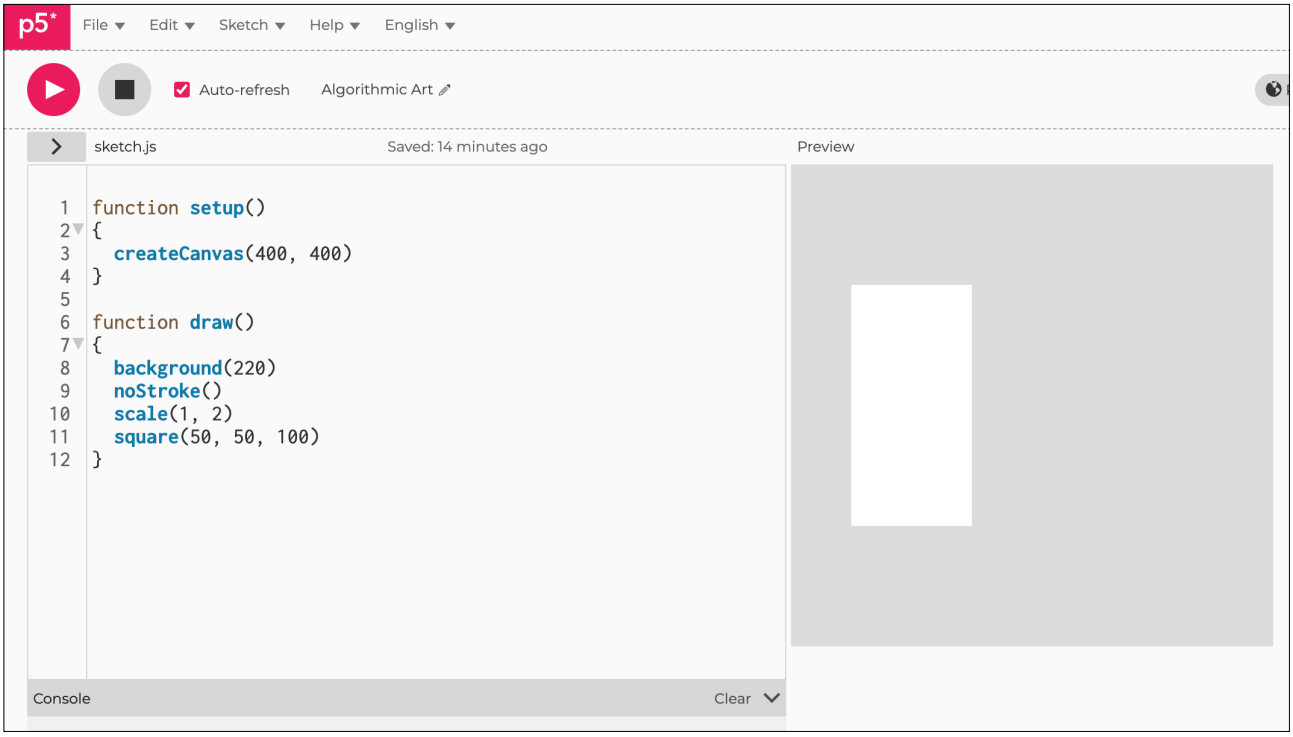
Challenges

1. Try `scale(2, 1)` and other variations.
2. What happens if you only have one argument?

Code Explanation

`scale(1, 2)`	This scales in two dimensions, first the x and then the y

Figure F3.1



Sketch F3.2 scale with mouse

Scaling when relative to the position of the mouse.

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

function draw()
{
  background(220)
  noStroke()
  scale(mouseX/width, mouseY/height)
  square(50, 50, 50)
}
```

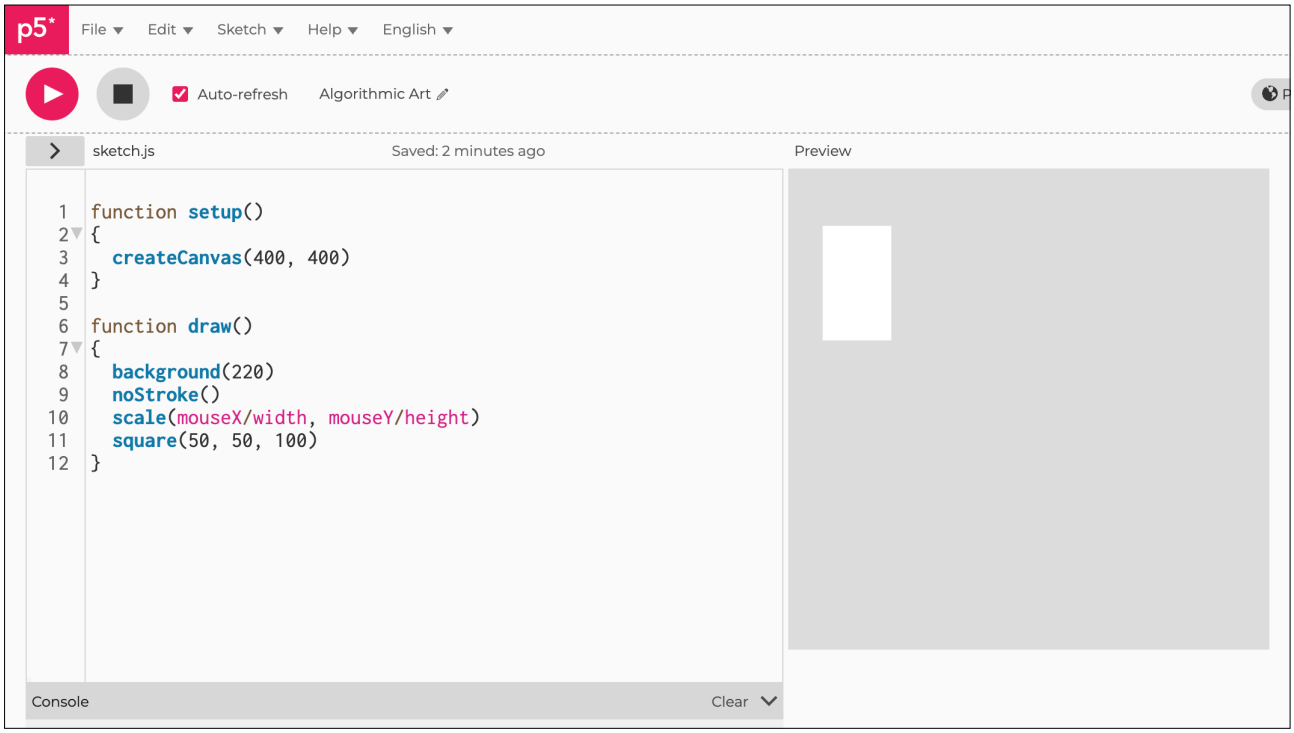
Notes

If you add in the `mouseX` and `mouseY` to the scale value, you can get a sense of how scaling works. We divide by the width and height; otherwise, the multiplication just takes it off-screen.

Challenges

1. Add the `stroke()` back in. What does scale do to that?
2. You could add a slider to change the scale.

Figure F3.2



Sketch F3.3 scale negatively

! Replace the scale() and rect() functions
Scaling in the negative direction.

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

function draw()
{
  background(220)
  noStroke()
  translate(width/2, height/2)
  fill(255)
  rect(0, 0, 100, 50)
  fill(0)
  scale(-1, -3)
  rect(0, 0, 100, 50)
}
```

Notes

Here you are drawing two identical rectangles. They are both translated to the centre of the canvas. The first one (white fill) is left as is. The second (black fill) is scaled negatively `(-1, -3)`. You will need to use `push()` and `pop()` if you want to scale them separately.

Challenges

1. Scale both of them.
2. What happens to the rectangle if you don't translate it? Why did that happen? Where is it?

Figure F3.3

Sketch F3.4 video capture

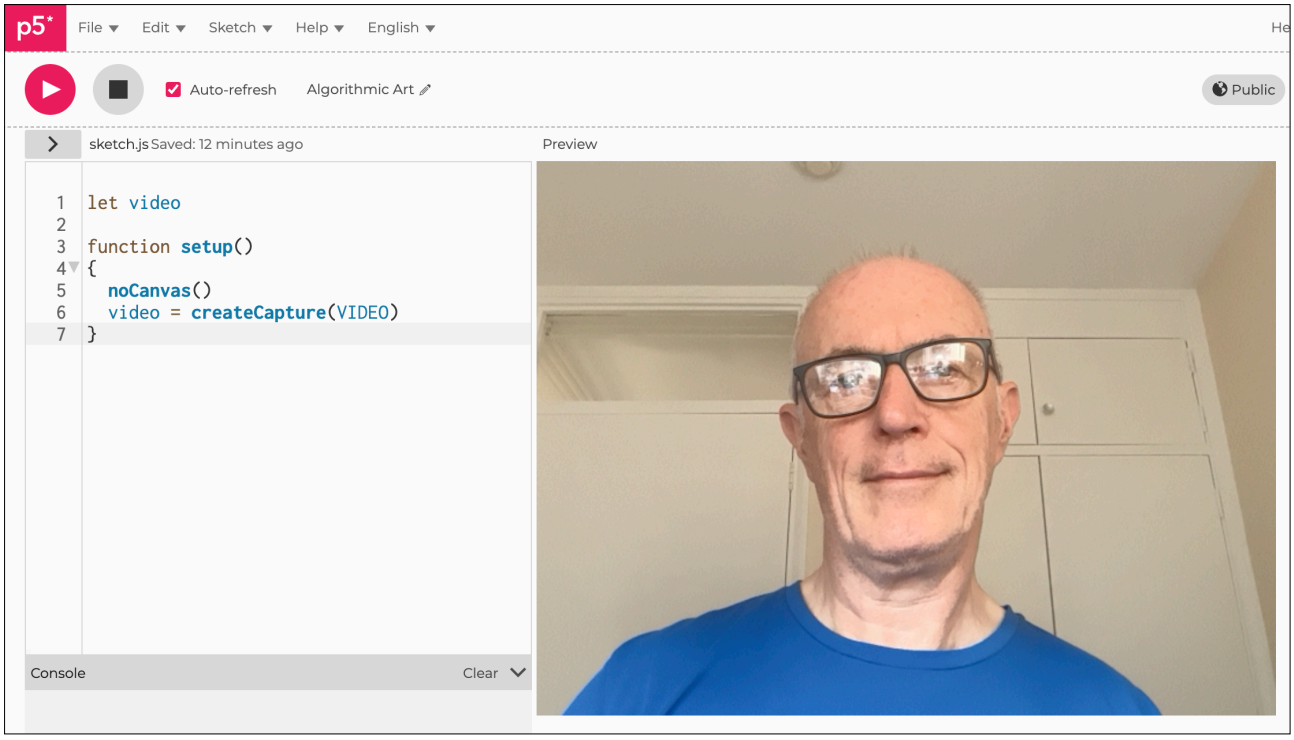
! Start a new sketch.
This calls on the computer to access your webcam; you will need to give it permission.

```
let video

function setup()
{
  noCanvas()
  video = createCapture(VIDEO)
}
```

Notes

The default size is 640 by 480 pixels, as displayed in the window.

Figure F3.4



Sketch F3.5 creating the canvas

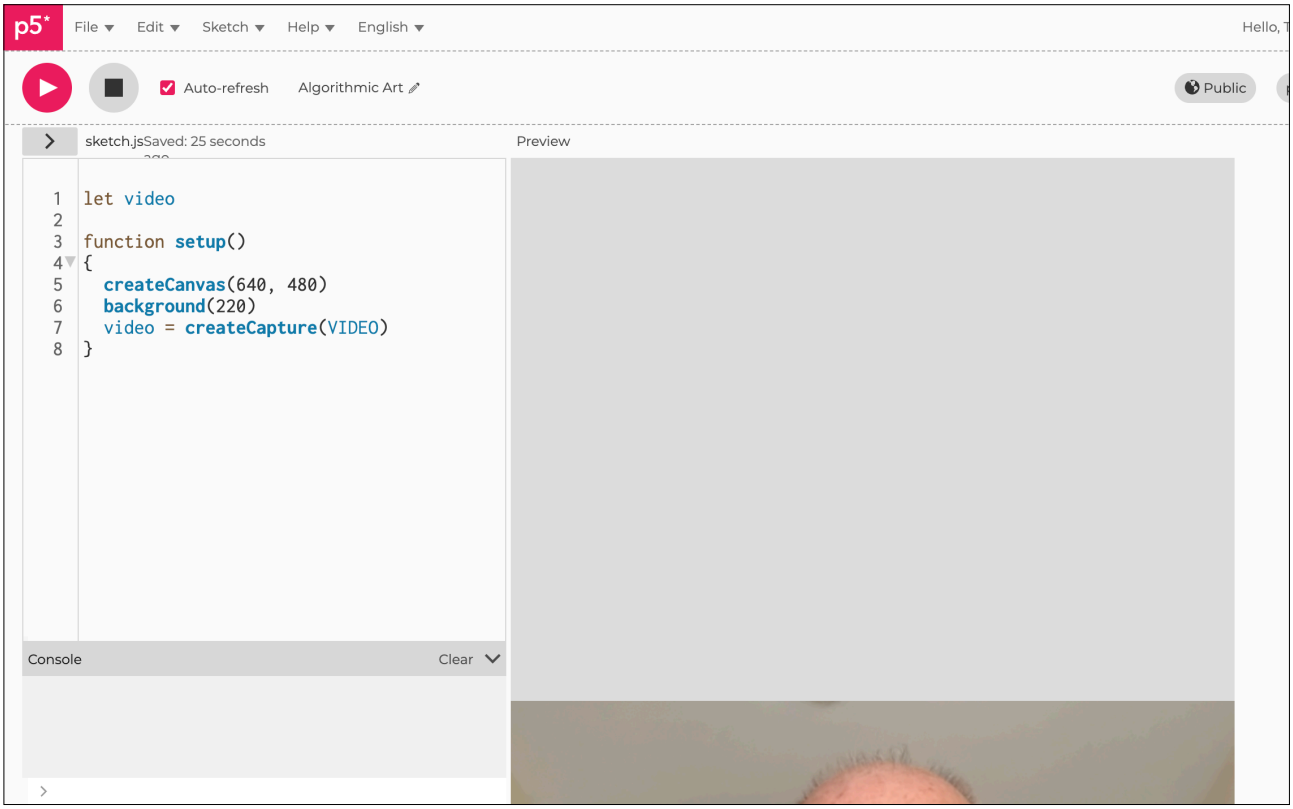
! Remove `noCanvas()` and replace with `createCanvas(640, 480)`.
Now add a `background(220)`.

```
let video

function setup()
{
  createCanvas(640, 480)
  background(220)
  video = createCapture(VIDEO)
}
```

Notes

All you get is the grey canvas and the top of your head, with the video below the canvas. We will rectify this shortly.

Figure F3.5



Sketch F3.6 video on the canvas

Drawing the video onto the canvas.

```
let video

function setup()
{
  createCanvas(640, 480)
  background(220)
  video = createCapture(VIDEO)
}

function draw()
{
  image(video, 0, 0)
}
```

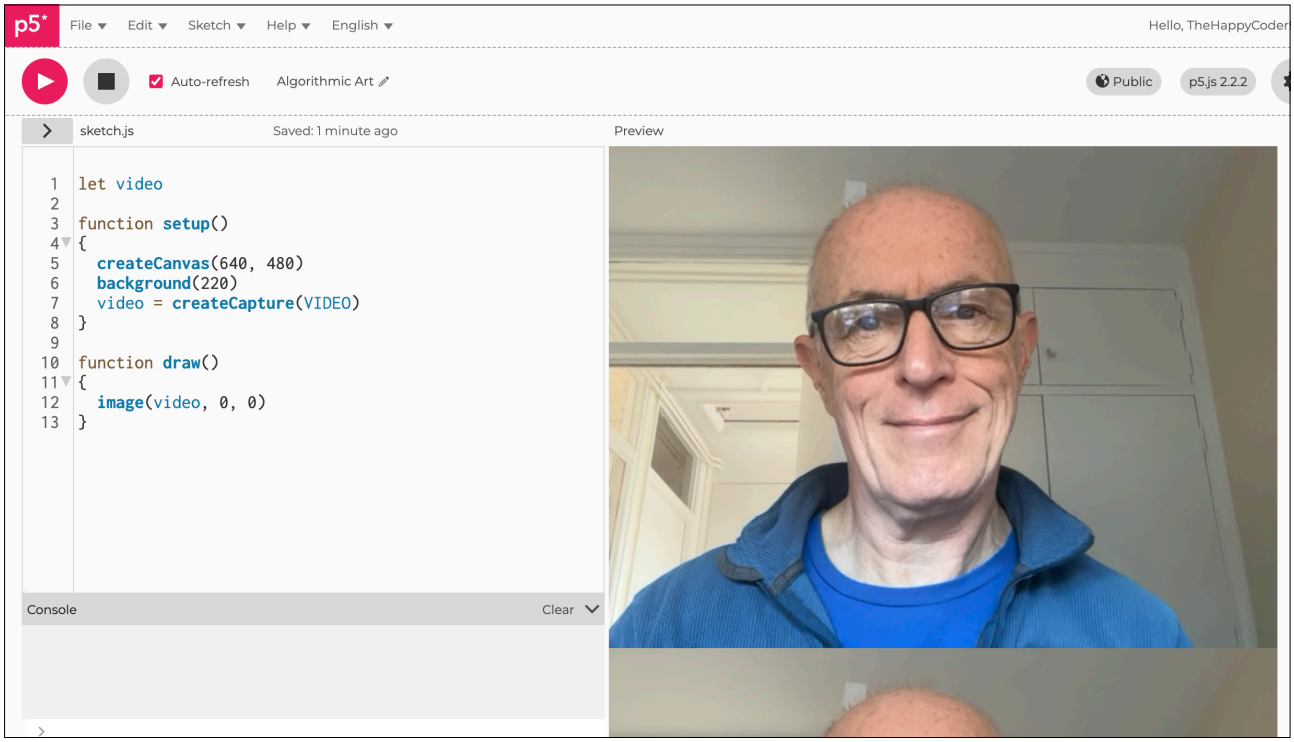
Notes

This creates two images: the top one is drawn onto the canvas, and the bottom one is the video stream.

Challenges

1. Add dimensions to the image function: `image(video, 0, 0, 320, 240)`. You should end up with an image half the size (actually a quarter of the size!).
2. Change the co-ordinates of the image function to `image(video, 200, 200, 320, 240)`. It has now moved it across the canvas.

Figure F3.6



Sketch F3.7 size and position

Resizing and repositioning.

```
let video

function setup()
{
  createCanvas(640, 480)
  background(220)
  video = createCapture(VIDEO)
}

function draw()
{
  image(video, 200, 200, 320, 240)
}
```

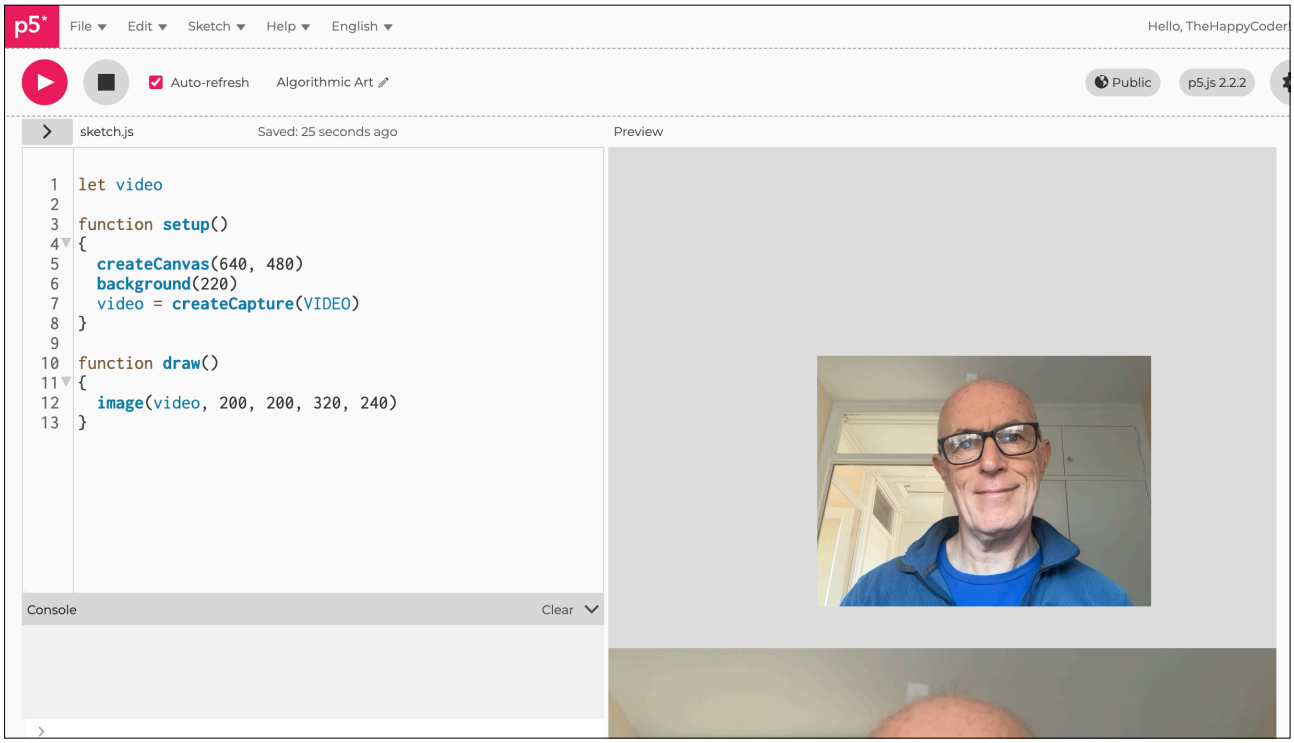
Notes

You will notice that you still have two images of yourself. But now one of them is repositioned and resized.

Code Explanation

`image(video, 200, 200, 320, 240)`	The video is 200 pixels from the left hand edge of the canvas, 200 pixels from the top, and the dimensions are half the original image size

Figure F3.7



Sketch F3.8 hide the streaming video

Just so that you get the one video drawn to the canvas.

```
let video

function setup()
{
  createCanvas(640, 480)
  background(220)
  video = createCapture(VIDEO)
  video.hide()
}

function draw()
{
  image(video, 200, 200, 320, 240)
}
```

Notes

Hide the video that is streaming (not the one on the canvas), then add the line of code `video.hide()`.

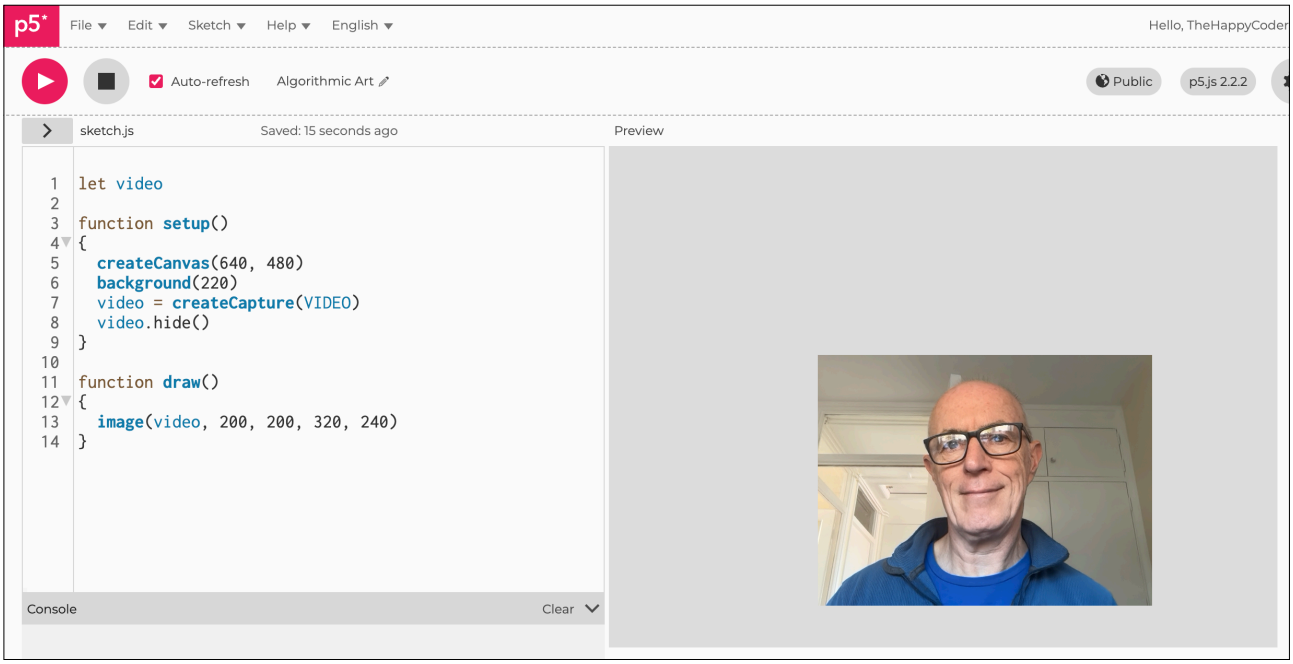
Challenges

1. Replace the `image()` function with `image(video, 0, 0)`. This now fills the canvas with the video stream.
2. If you increase the canvas to `createCanvas(800, 600)`, you will see that it doesn't alter the image's default size.
3. Now change the `image()` function to `image(video, 0, 0, 800, 600)`. This now fills the canvas.

Code Explanation

`video.hide()`	This hides the video to the canvas

Figure F3.8



Sketch F3.9 taking a selfie

Get ready to pose for the selfie!
! Replace the `draw()` function with the `takesnap()` function

```
let video
let button

function setup()
{
  createCanvas(640, 480)
  background(220)
  video = createCapture(VIDEO)
  video.hide()
  button = createButton('snap')
  button.mousePressed(takesnap)
}

function takesnap()
{
  image(video, 0, 0)
}
```

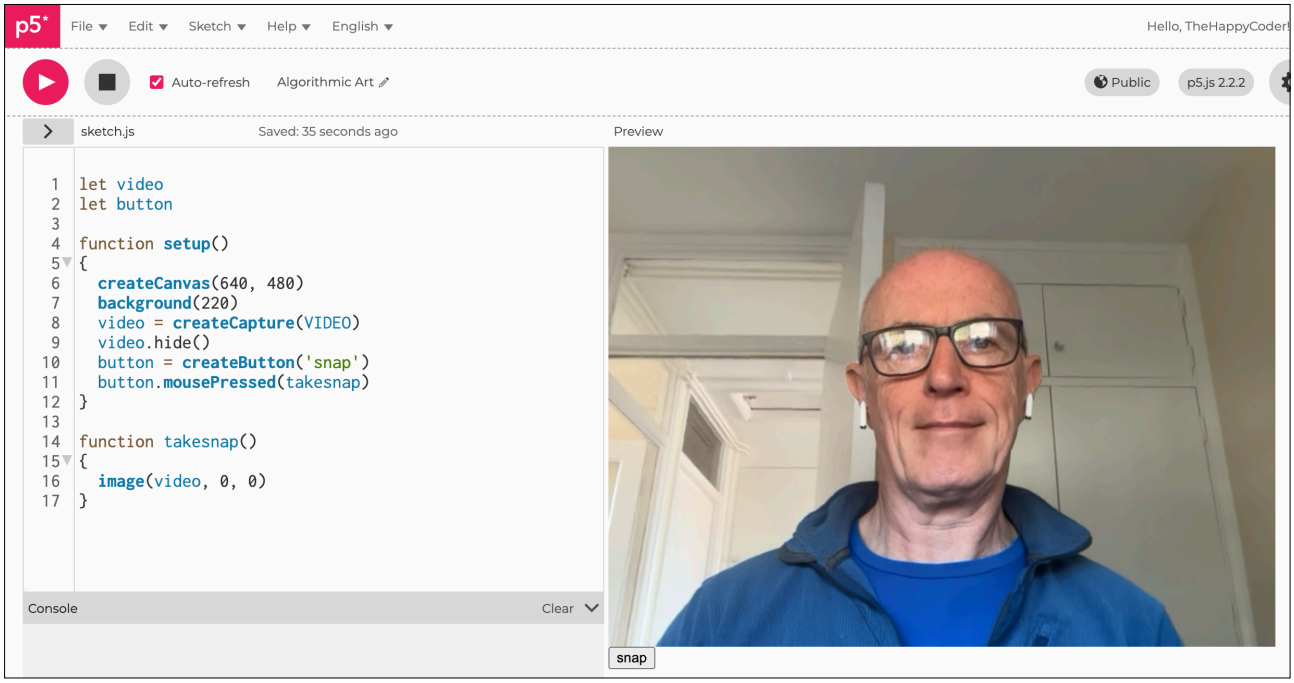
Notes

In this example, we have added a button that, when clicked, takes a picture of the video stream. The `image()` create function is now in the `takesnap()` function.

Challenge

Can you create four images or more?

Figure F3.9



Sketch F3.10 multiple selfies

! Putting the draw() function back in.
As if one wasn't enough.

```
let video
let button
let snapshots = []

function setup()
{
  createCanvas(640, 480)
  background(220)
  video = createCapture(VIDEO)
  video.hide()
  button = createButton('snap')
  button.mousePressed(takesnap)
}

function takesnap()
{
  snapshots.push(video.get())
}

function draw()
{
  let x = 0
  let y = 0
  let w = width / 5
  let h = height / 5

  for (let i = 0; i < snapshots.length; i++)
  {
    image(snapshots[i], x, y, w, h)
    x = x + w
    if (x >= width)
    {
      x = 0
      y = y + h
```

```
    }
  }
}
```

Notes

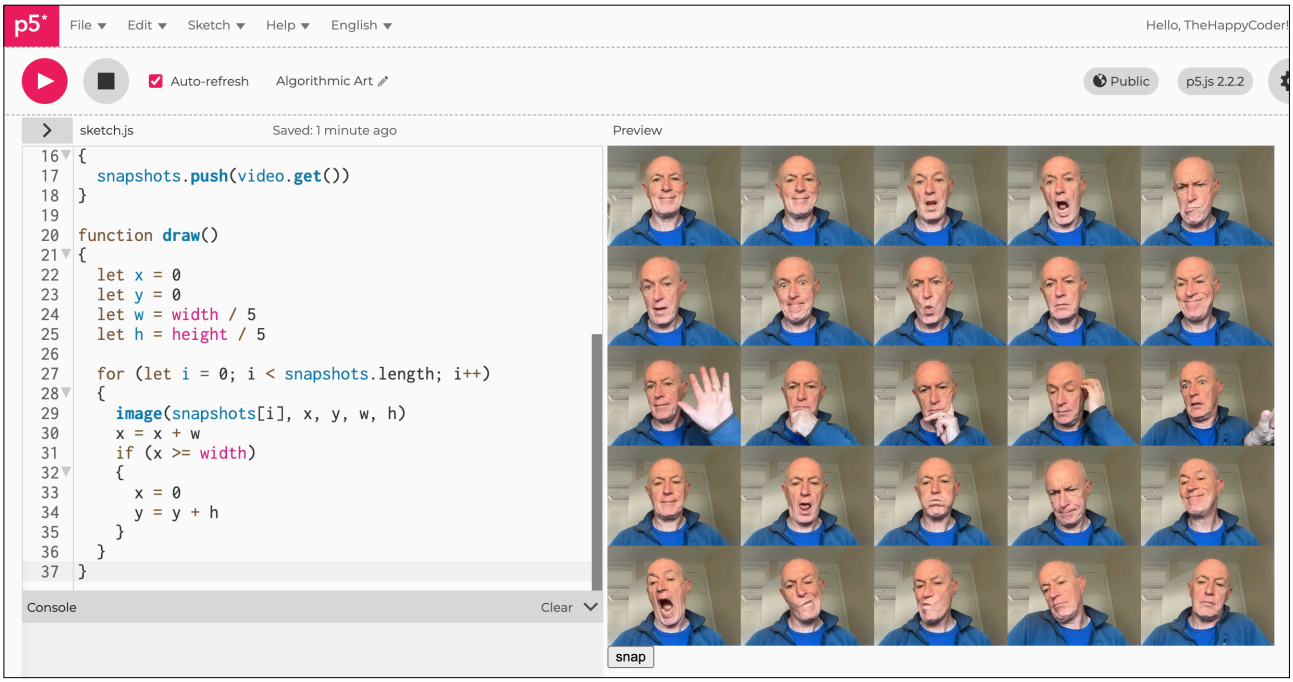
We create an empty array and call it `snapshots[]`. Every time the button is clicked, it adds an image to be stored in the array. The `get()` function gets the image, and the `push()` function pushes it into the array. By default, the `get()` function gets the whole image; you can specify the actual pixel in the image.

Then we create a loop in `draw()` where it goes through each image at index `snapshots[0]`, then `snapshots[1]`, and so on. Drawing each image of size width `w` and height `h`. The images are spaced out by adding the height and width onto the `x` and `y` co-ordinates.

Challenge

Change the size and number of the images

Figure F3.10



Sketch F3.11 getting pixels

! Start a new sketch.
We have covered pixels in previous units. Once you have an image, the next step is to get the value of each pixel so that we can do something interesting with it. This next section is starting from scratch (so delete everything and copy the code below). We will add the image in soon.

```
let x
let y
let index

function setup()
{
  createCanvas(640, 480)
  pixelDensity(1)
}

function draw()
{
  background(0)
  loadPixels()
  for (y = 0; y < height; y++)
  {
    for (x = 0; x < width; x++)
    {
      index = (x + (y * 640)) * 4
      pixels[index + 0] = 255
      pixels[index + 1] = 100
      pixels[index + 2] = 0
      pixels[index + 3] = 255
    }
  }
  updatePixels()
}
```

Notes

You should get an orange-filled rectangle.
Each pixel has four elements to it. It has a red value, a green value, and an alpha value. Each between `0` and `255`. The `pixelDensity()` function is necessary because some monitors have high-density screens, and this means there are more than four elements per pixel.

The `loadPixels()` function creates an array of pixels where each index is red, green, blue, and alpha. So for every pixel on the screen, it has four elements. So for the first pixel `(0, 0)`,

`index[0]` is the red value of the first pixel.
`index[1]` is the green value of the first pixel.
`index[2]` is the blue value of the first pixel.
`index[3]` is the alpha value of the first pixel.
`index[4]` is the red value of the second pixel.
`index[5]` is the green value of the second pixel… and so on.

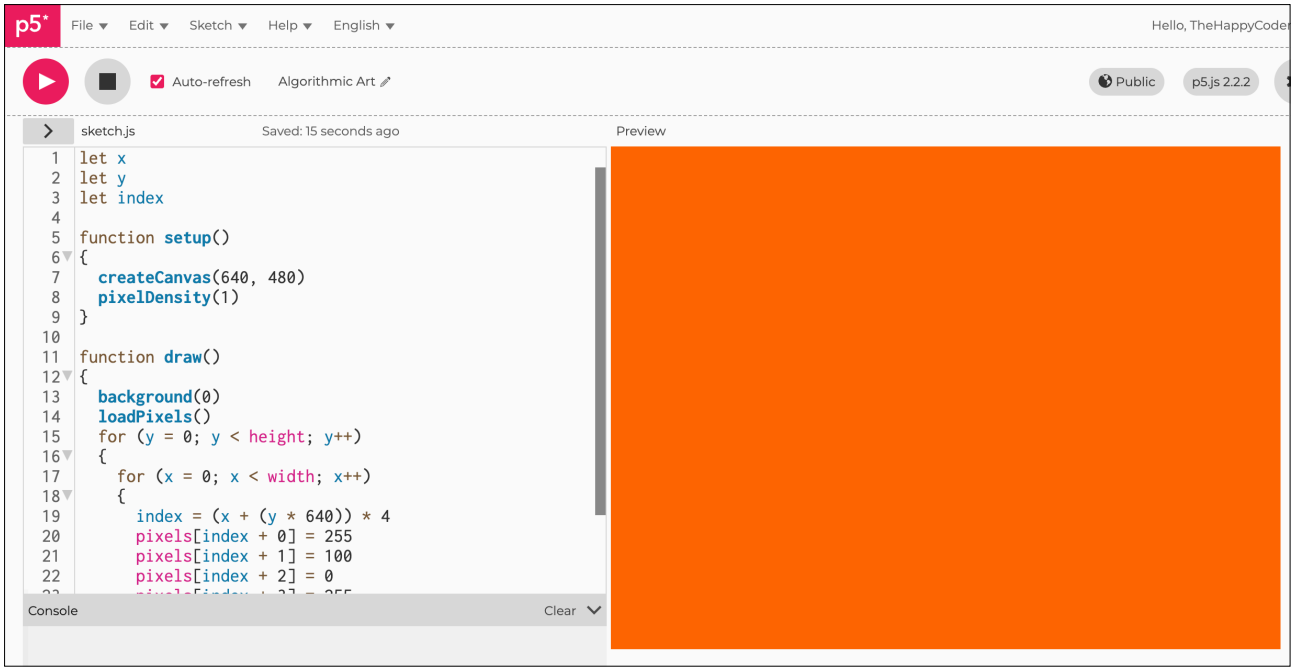
The line of code…
`index = (x + (y * 640)) * 4`
Goes through every fourth index. When it gets to the end of the row, it has done `640` of them, and so the next line of pixels starts with `640`. It sounds complicated, but this is how it works: through every pixel one at a time and then when it is at each pixel, works through the red, green, blue, and alpha element.

This means you can change not just an individual pixel but its red, green, blue, and alpha element. At the end, you simply update the pixels with the updatePixels. Notice that the original background was black, and we have changed every pixel so that it is now orange.

Challenges

1. Remove the `pixelDensity()` function.
2. Change the colour.
3. What happens if you make one of the r, g, b, or alpha elements `random()`?
4. Change the width of the canvas.

Figure F3.11


p5*
File
Edit
Sketch
Help
English
Hello, TheHappyCoder
Auto-refresh
Algorithmic Art
Public
p5.js 2.2.2
sketch.js
Saved: 15 seconds ago
Preview
let x
let y
let index
function setup()
{
createCanvas(640, 480)
pixelDensity(1)
}
function draw()
{
background(0)
loadPixels()
for (y = 0; y < height; y++)
{
for (x = 0; x < width; x++)
{
index = (x + (y * 640)) * 4
pixels[index + 0] = 255
pixels[index + 1] = 100
pixels[index + 2] = 0
Console
Clear

Sketch F3.12 returning the video

Putting the video back in and reading every pixel from the image produced.

```
let video
let x
let y
let index

function setup()
{
  createCanvas(640, 480)
  pixelDensity(1)
  video = createCapture(VIDEO)
}

function draw()
{
  background(0)
  video.loadPixels()
  loadPixels()
  for(y = 0; y < height; y++)
  {
    for(x = 0; x < width; x++)
    {
      index = (x + (y * 640)) * 4
      pixels[index + 0] = video.pixels[index + 0]
      pixels[index + 1] = video.pixels[index + 1]
      pixels[index + 2] = video.pixels[index + 2]
      pixels[index + 3] = video.pixels[index + 3]
    }
  }
  updatePixels()
}
```

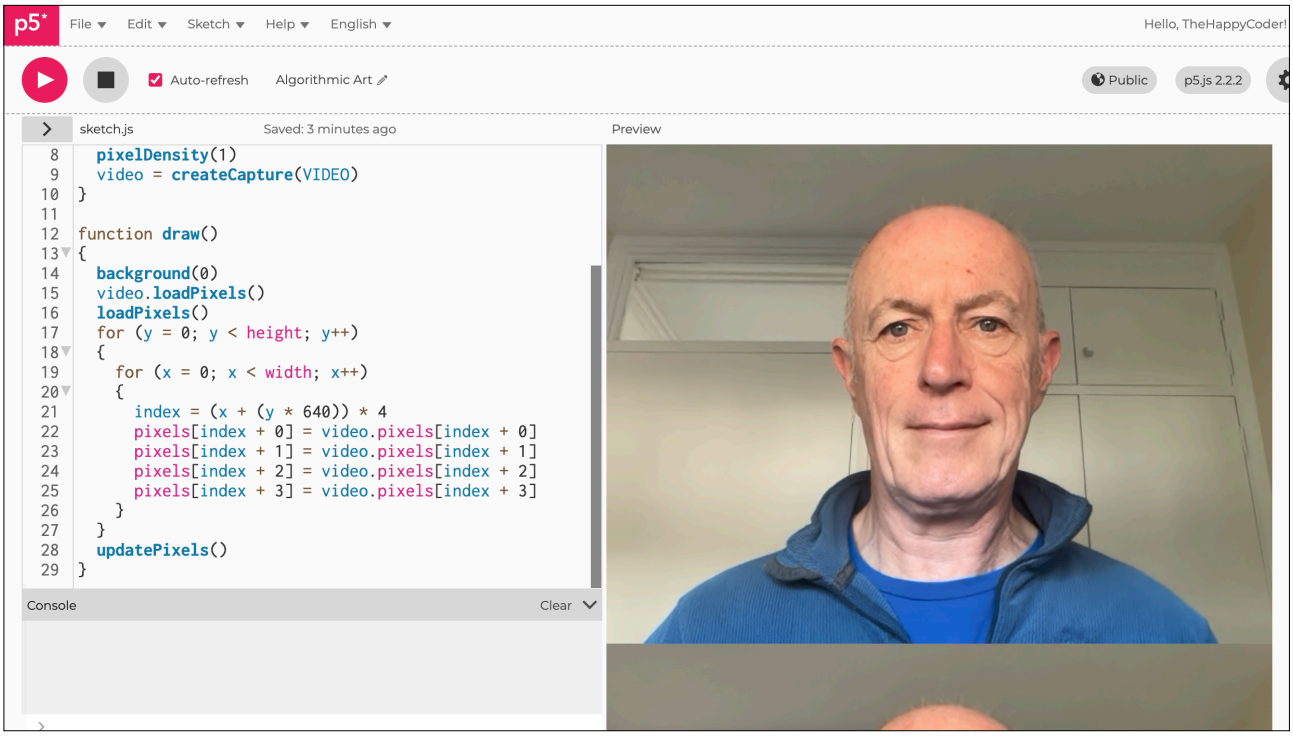
Notes

This time we have added the video back in using the video pixels. This may look exactly the same depending on the processor of your computer. To test that you are actually using the pixels, you can do the following challenge...

Challenges

1. Change: `video.pixels[index + 0]` to `video.pixels[index + 0] / 255`, which will take out the red value.
2. Try it with the green and the blue.

Figure F3.12



Sketch F3.13 pixelating the image

! Remove updatePixels().
This can be used not just for creating interesting effects, but also when using it for the purposes of AI or machine learning.

```
let video
let x
let y
let index
let r
let g
let b
let a
let vScale = 16

function setup()
{
  createCanvas(640, 480)
  pixelDensity(1)
  video = createCapture(VIDEO)
  video.size(width / vScale, height / vScale)
  video.hide()
}

function draw()
{
  background(255)
  video.loadPixels()
  loadPixels()
  for (y = 0; y < video.height; y++)
  {
    for (x = 0; x < video.width; x++)
    {
      index = (x + (y * video.width)) * 4
      r = video.pixels[index + 0]
      g = video.pixels[index + 1]
      b = video.pixels[index + 2]
      a = video.pixels[index + 3]
```

```
      fill(r, g, b, a)
      square(x * vScale, y * vScale, vScale)
    }
  }
}
```

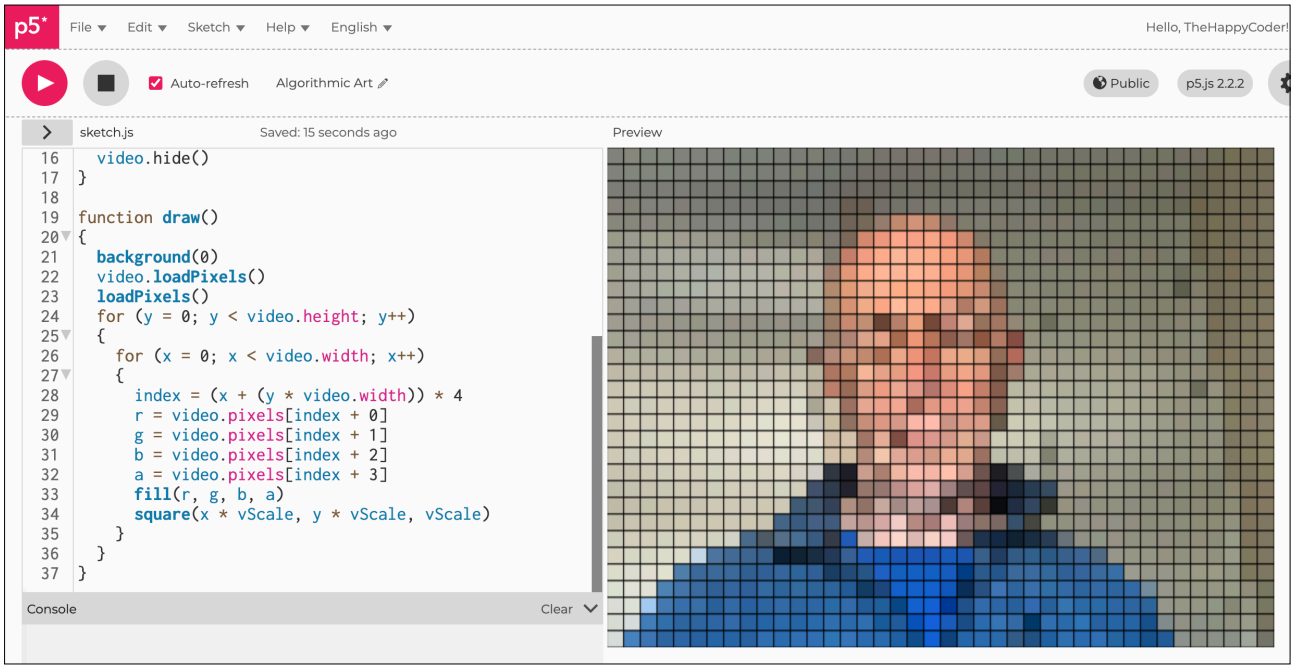
Notes

This takes the original video image, scans each pixel for the r, g, b, and a, and then scales it up to fill the canvas. A square is then filled with the colour of the original image.

Challenges

1. Change the vScale.
2. Change the shape to a circle.

Figure F3.13



Sketch F3.14 making it grey scale

You can now manipulate the colour; don't forget to remove the `a = video.pixels[index + 3]` line of code.

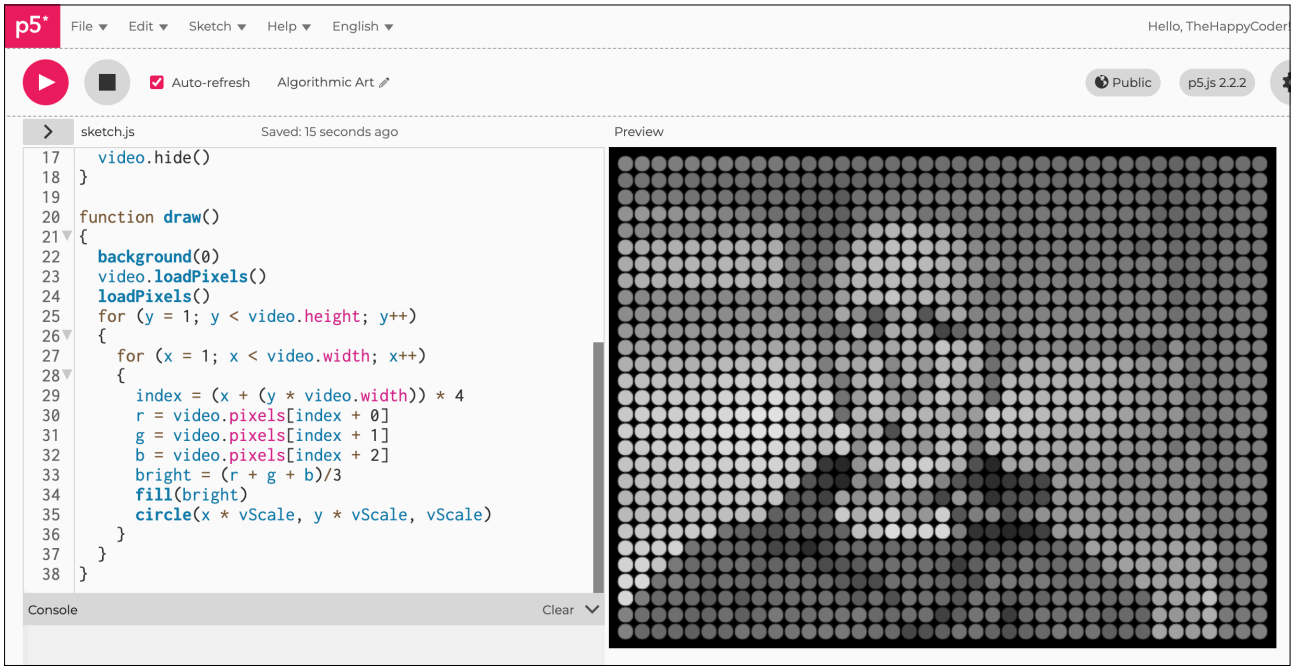
```
let x
let y
let index
let r
let g
let b
let a
let vScale = 16
let bright

function setup()
{
  createCanvas(640, 480)
  pixelDensity(1)
  video = createCapture(VIDEO)
  video.size(width / vScale, height / vScale)
  video.hide()
}

function draw()
{
  background(0)
  video.loadPixels()
  loadPixels()
  for (y = 1; y < video.height; y++)
  {
    for (x = 1; x < video.width; x++)
    {
      index = (x + (y * video.width)) * 4
      r = video.pixels[index + 0]
      g = video.pixels[index + 1]
      b = video.pixels[index + 2]
      bright = (r + g + b)/3
      fill(bright)
```

```
      circle(x * vScale, y * vScale, vScale)
    }
  }
}
```

Notes

Now we have added the video in as before, but we have added the value of the R, G, B and divided by three to get the average brightness. This is not the same as the alpha.

Figure F3.14



Sketch F3.15 brightness mirror

This takes the brightness of each pixel.

```
let x
let y
let index
let r
let g
let b
let a
let vScale = 16
let bright
let w

function setup()
{
  createCanvas(640, 480)
  pixelDensity(1)
  video = createCapture(VIDEO)
  video.size(width / vScale, height / vScale)
  video.hide()
}

function draw()
{
  background(0)
  video.loadPixels()
  loadPixels()
  for (y = 1; y < video.height; y++)
  {
    for (x = 1; x < video.width; x++)
    {
      index = (x + (y * video.width)) * 4
      r = video.pixels[index + 0]
      g = video.pixels[index + 1]
      b = video.pixels[index + 2]
      bright = (r + g + b)/3
```

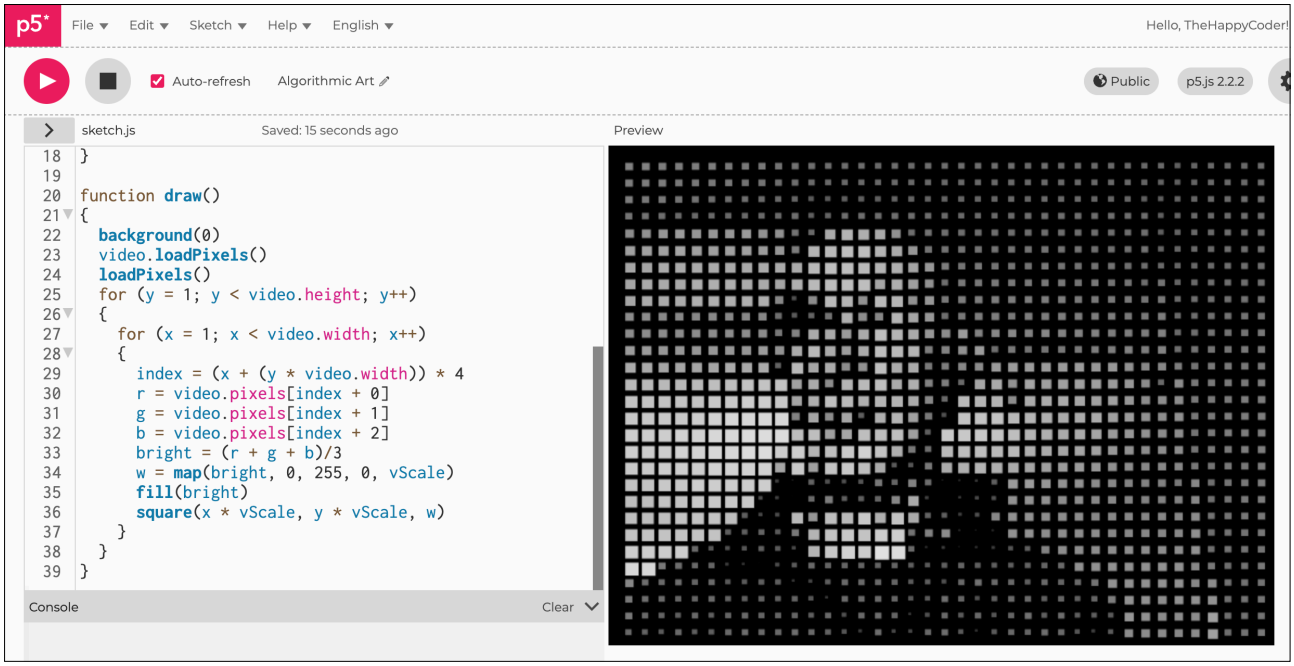
```
      w = map(bright, 0, 255, 0, vScale)
      fill(bright)
      square(x * vScale, y * vScale, w)
    }
  }
}
```

Notes

The average brightness of each pixel is calculated as before, and now it is mapped to a new variable `w`, where the width of the square is dependent on the brightness value. So `255` would correspond to full `vScale` width. Therefore, the brighter the value, the bigger the square.

Save this sketch for use in a moment.

Figure F3.15



Sketch F3.16 threshold image

! Replace the variable w with threshold.
A bit more manipulation of the brightness.

```
let x
let y
let index
let r
let g
let b
let a
let vScale = 8
let bright
let threshold

function setup()
{
  createCanvas(640, 480)
  pixelDensity(1)
  video = createCapture(VIDEO)
  video.size(width / vScale, height / vScale)
  video.hide()
  noStroke()
}

function draw()
{
  background(0)
  video.loadPixels()
  loadPixels()
  for (y = 1; y < video.height; y++)
  {
    for (x = 1; x < video.width; x++)
    {
      index = (x + (y * video.width)) * 4
      r = video.pixels[index + 0]
      g = video.pixels[index + 1]
      b = video.pixels[index + 2]
```

```
      bright = (r + g + b)/3
      threshold = 150
      if(bright > threshold)
      {
        fill(255)
      }
      else
      {
        fill(0)
      }
      circle(x * vScale, y * vScale, vScale)
    }
  }
}
```

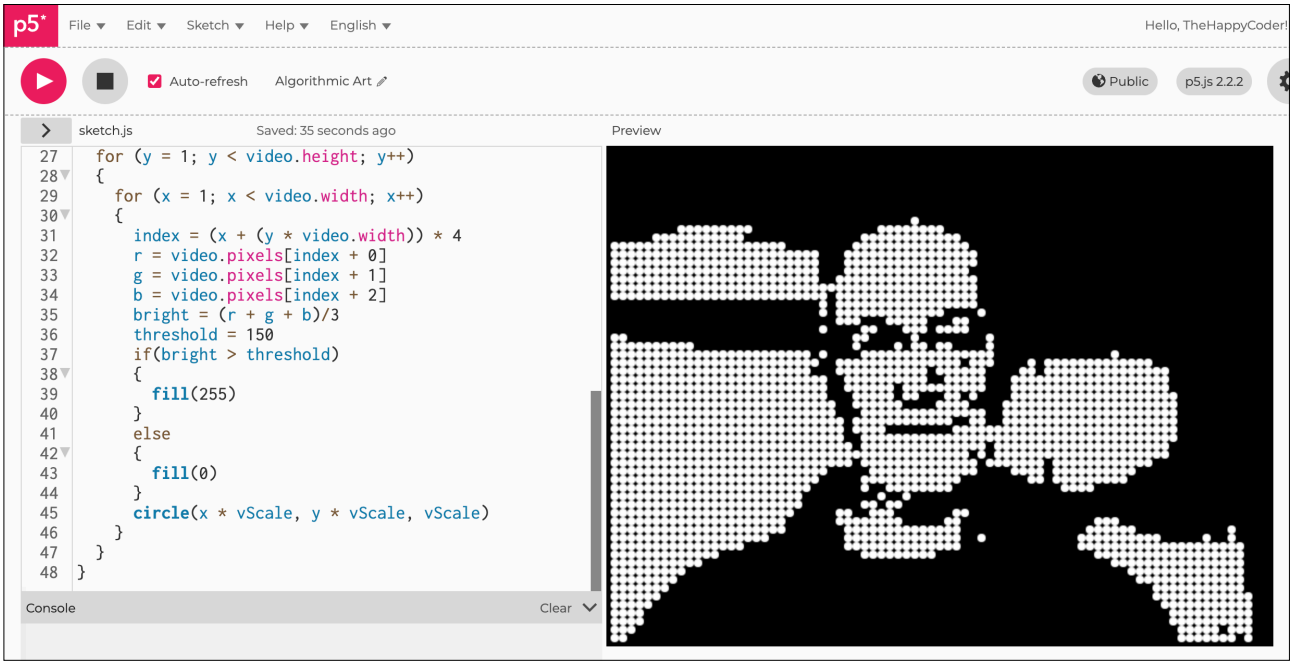
Notes

This is a really nice adaptation of the previous sketches. It does require some fiddling with the variables, notably the `threshold` value, which will in turn depend on the brightness of the room.

Challenges

1. Change the threshold.
2. Change the size and shape.
3. Add colour

Figure F3.16



Sketch F3.17 optional threshold

Developing the same theme, remove the threshold and just use the brightness.

```
let x
let y
let index
let r
let g
let b
let a
let vScale = 4
let bright

function setup()
{
  createCanvas(640, 480)
  pixelDensity(1)
  video = createCapture(VIDEO)
  video.size(width / vScale, height / vScale)
  video.hide()
  noStroke()
}

function draw()
{
  background(0)
  video.loadPixels()
  loadPixels()
  for (y = 1; y < video.height; y++)
  {
    for (x = 1; x < video.width; x++)
    {
      index = (x + (y * video.width)) * 4
      r = video.pixels[index + 0]
      g = video.pixels[index + 1]
      b = video.pixels[index + 2]
      bright = (r + g + b)/3
```

```
      if(bright > 150)
      {
        fill(255)
      }
      else if(bright > 100 && bright < 150)
      {
        fill(150)
      }
      else if(bright < 100)
      {
        fill(0)
      }
      circle(x * vScale, y * vScale, vScale)
    }
  }
}
```

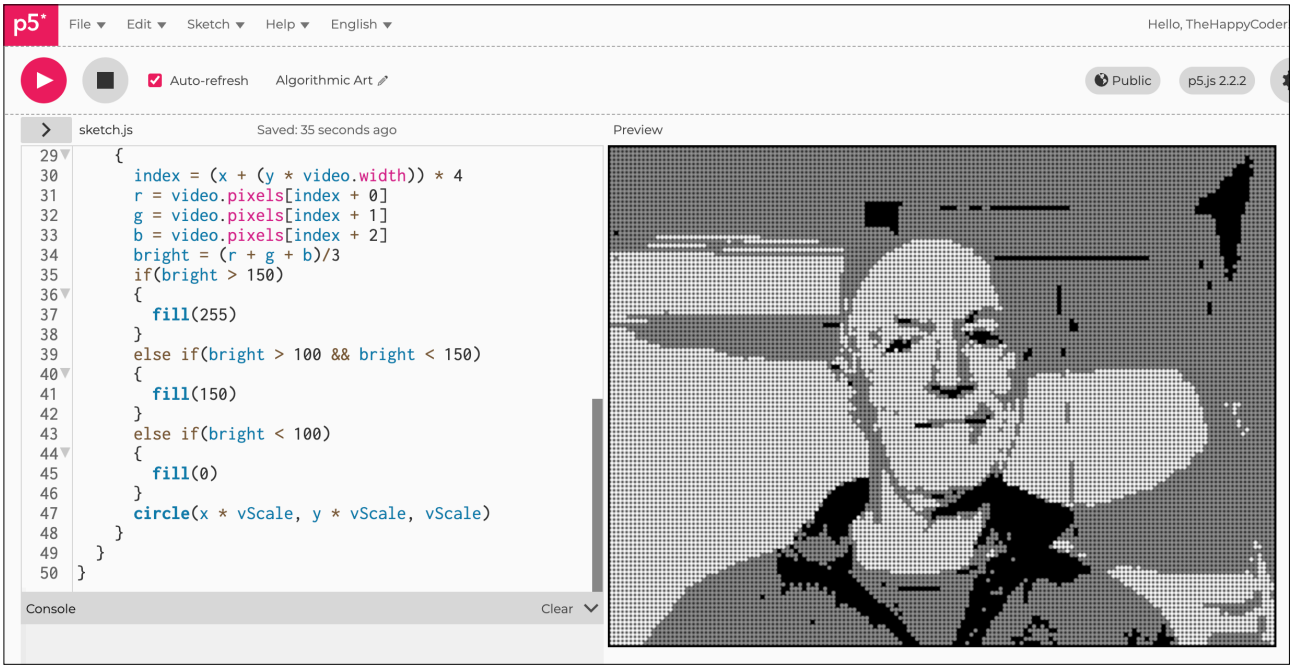
Notes

A variation on the previous sketch to have more than one threshold using `else if()` statements.

Challenges

1. Change the threshold values to suit your video image.
2. Add more thresholds.
3. Add colour.

Figure F3.17



Sketch F3.18 starting sketch

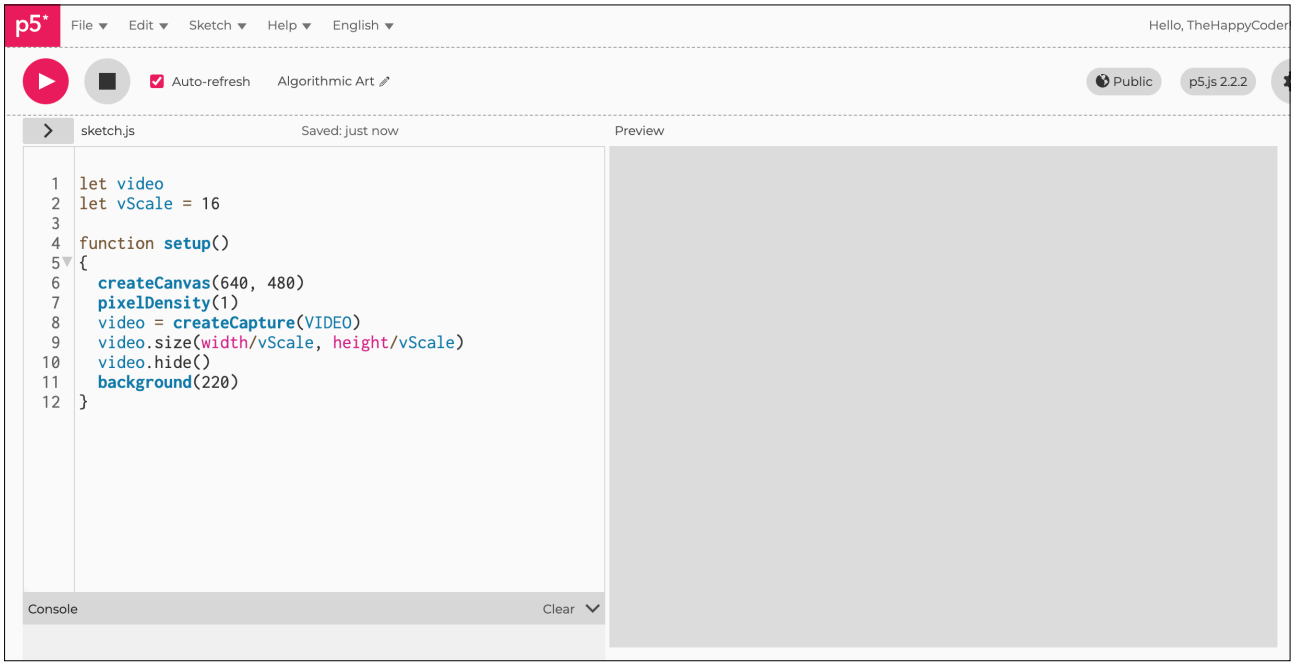
! We are going to start a new sketch.

```
let video
let vScale = 16

function setup()
{
  createCanvas(640, 480)
  pixelDensity(1)
  video = createCapture(VIDEO)
  video.size(width/vScale, height/vScale)
  video.hide()
  background(220)
}
```

Notes

Nothing to see here yet.

Figure F3.18



Sketch F3.19 array of particles

! Don't run this yet; you will get an error message.
We are going to create an array of particles from random positions on the canvas.

```
let video
let vScale = 16
let particles = []

function setup()
{
  createCanvas(640, 480)
  pixelDensity(1)
  video = createCapture(VIDEO)
  video.size(width/vScale, height/vScale)
  video.hide()
  background(220)
  for (let i = 0; i < 300; i++)
  {
    particles[i] = new Particle(random(640), random(480))
  }
}
```

Notes

We haven't created the `Particles class` yet.

Sketch F3.20 the Particle class

We have our `constructor`, which takes two arguments: the random values of `x` and `y`.

```
let video
let vScale = 16
let particles = []

function setup()
{
  createCanvas(640, 480)
  pixelDensity(1)
  video = createCapture(VIDEO)
  video.size(width/vScale, height/vScale)
  video.hide()
  background(220)
  for (let i = 0; i < 300; i++)
  {
    particles[i] = new Particle(random(640), random(480))
  }
}
```

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

Notes

There is nothing to see except a grey canvas.

Sketch F3.21 show something

Adding the `show()` function.

```
let video
let vScale = 16
let particles = []

function setup()
{
  createCanvas(640, 480)
  pixelDensity(1)
  video = createCapture(VIDEO)
  video.size(width/vScale, height/vScale)
  video.hide()
  background(220)
  for (let i = 0; i < 300; i++)
  {
    particles[i] = new Particle(random(640), random(480))
  }
}
```

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

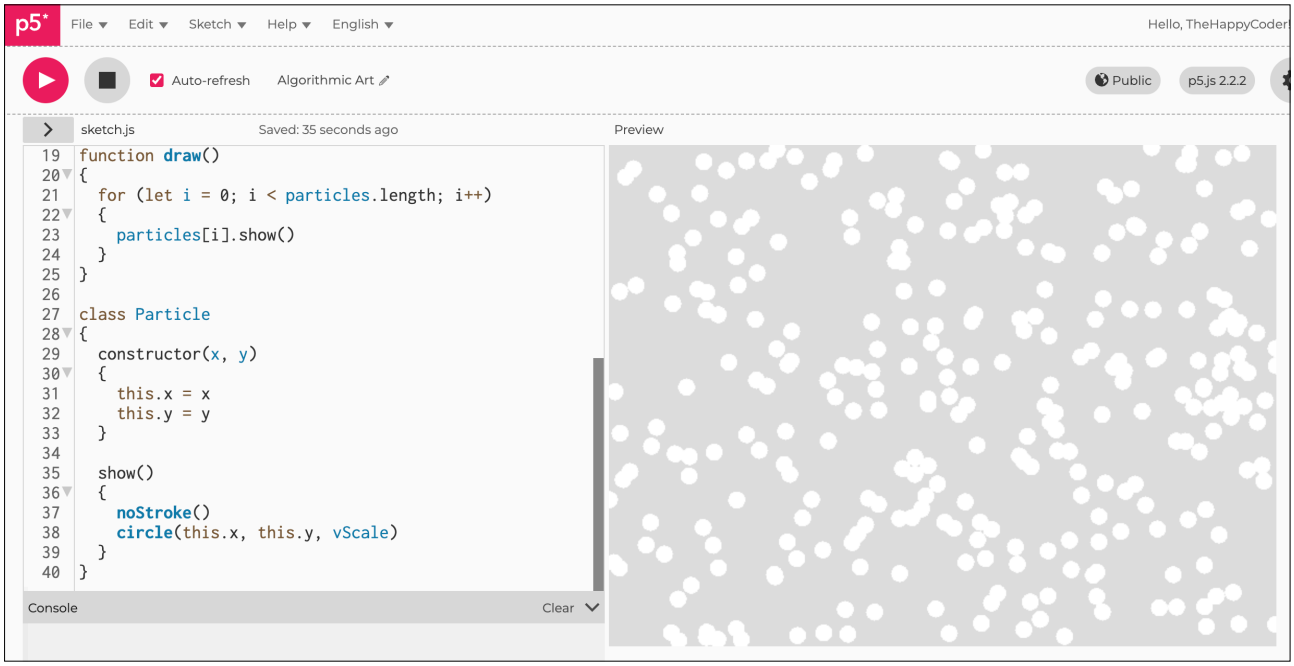
```

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

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

Notes

We just get the white circles; next, we need the pixels.

Figure F3.21



Sketch F3.22 getting the pixels

All well and good, but we want the pixels from the video to fill the circles. This is the first step.

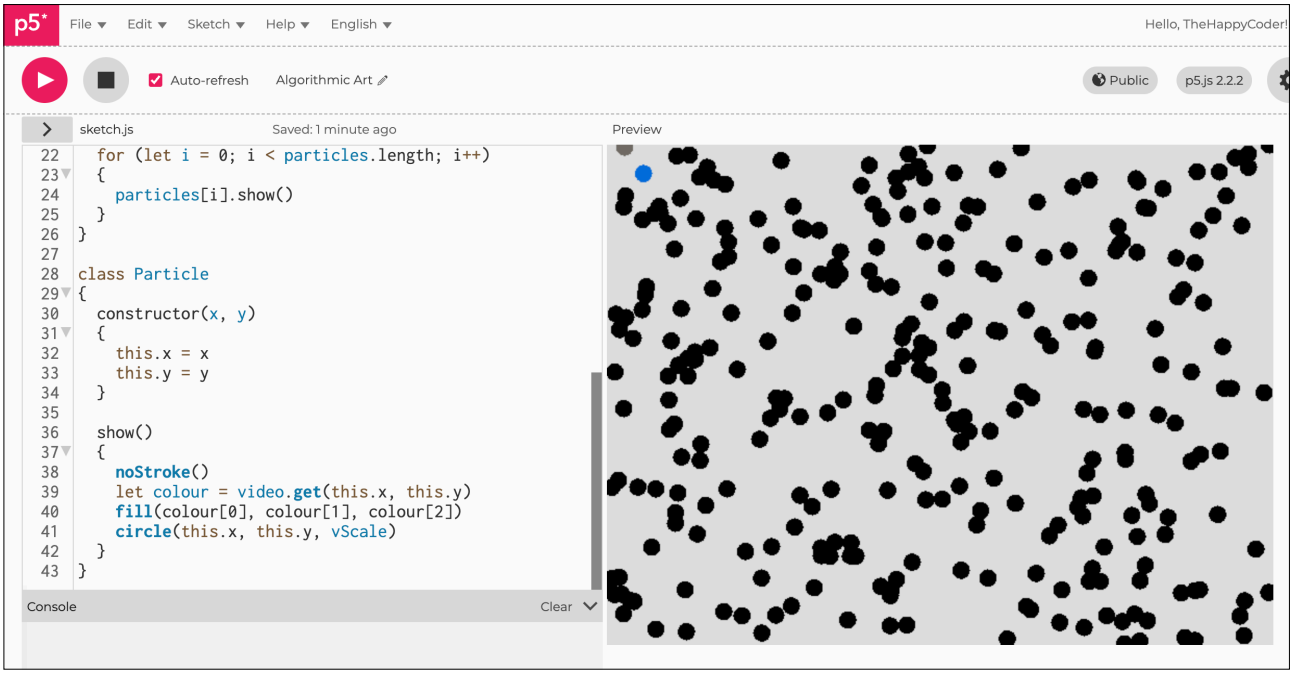
```
let video
let vScale = 16
let particles = []

function setup()
{
  createCanvas(640, 480)
  pixelDensity(1)
  video = createCapture(VIDEO)
  video.size(width/vScale, height/vScale)
  video.hide()
  background(220)
  for (let i = 0; i < 300; i++)
  {
    particles[i] = new Particle(random(640), random(480))
  }
}

function draw()
{
  video.loadPixels()
  for (let i = 0; i < particles.length; i++)
  {
    particles[i].show()
  }
}

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

```
  show()
  {
    noStroke()
    let colour = video.get(this.x, this.y)
    fill(colour[0], colour[1], colour[2])
    circle(this.x, this.y, vScale)
  }
}
```

Notes

This still doesn't really do anything. There is more to come.

Figure F3.22



Sketch F3.23 the 300

Getting the pixels that matter. These are the pixels where the circles are, whereas before we just got the first 300 pixels of the image.

```
let video
let vScale = 16
let particles = []

function setup()
{
  createCanvas(640, 480)
  pixelDensity(1)
  video = createCapture(VIDEO)
  video.size(width/vScale, height/vScale)
  video.hide()
  background(220)
  for (let i = 0; i < 300; i++)
  {
    particles[i] = new Particle(random(640), random(480))
  }
}

function draw()
{
  video.loadPixels()
  for (let i = 0; i < particles.length; i++)
  {
    particles[i].show()
  }
}

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

```
  show()
  {
    noStroke()
    let px = this.x/vScale
    let py = this.y/vScale
    let colour = video.get(px, py)
    fill(colour[0], colour[1], colour[2])
    circle(this.x, this.y, vScale)
  }
}
```

Notes

Now we have them.

Sketch F3.24 randomise the position

We initialised the particles array with 300 position vectors. We drew circles based on those positions in the array. Now we can move those positions by a random amount of −10, or 10, and redraw the circle with the relevant pixel value derived from the video.

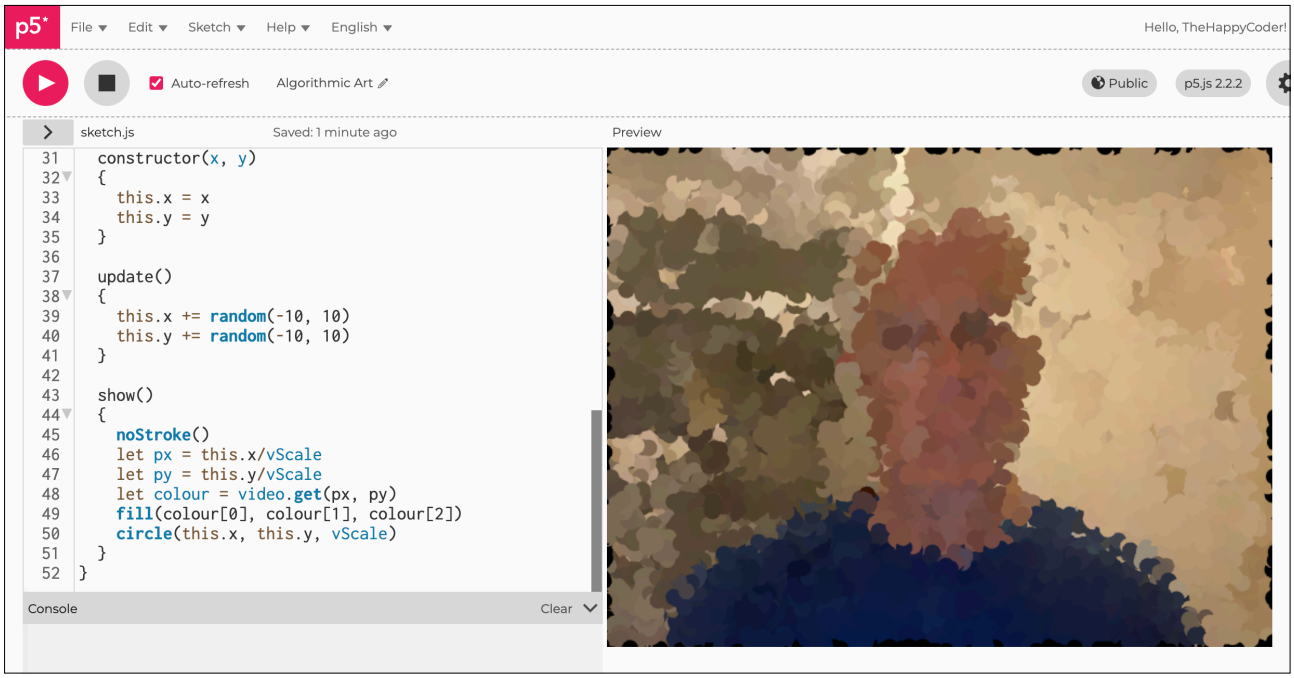
```
let video
let vScale = 16
let particles = []

function setup()
{
  createCanvas(640, 480)
  pixelDensity(1)
  video = createCapture(VIDEO)
  video.size(width/vScale, height/vScale)
  video.hide()
  background(220)
  for (let i = 0; i < 300; i++)
  {
    particles[i] = new Particle(random(640), random(480))
  }
}

function draw()
{
  video.loadPixels()
  for (let i = 0; i < particles.length; i++)
  {
    particles[i].update()
    particles[i].show()
  }
}

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

```
    this.y = y
  }

  update()
  {
    this.x += random(-10, 10)
    this.y += random(-10, 10)
  }

  show()
  {
    noStroke()
    let px = this.x/vScale
    let py = this.y/vScale
    let colour = video.get(px, py)
    fill(colour[0], colour[1], colour[2])
    circle(this.x, this.y, vScale)
  }
}
```

Notes

What we get is a video image of you that seems to be constantly moving.

Figure F3.24



Creating the mirror effect

There are two ways to create a mirror image of the video. By that, I mean it will look as if you are staring at a mirror, so that your left hand appears on your left, facing the screen.

Sketch F3.25 mirror the image #1

! Start a new sketch.
It makes things a bit more intuitive. Create a new temporary sketch for this illustration.

```
let video

function setup()
{
  createCanvas(640, 480)
  background(220)
  video = createCapture(VIDEO)
  video.hide()
}

function draw()
{
  scale(-1, 1)
  image(video, -width, 0)
}
```

Notes
Simple video

Challenges
1. Play around with the `scale()` function for different sizes, e.g. `scale(0.5, 0.5)`.
2. Can you get yourself to be upside down?

Sketch F3.26 mirror the image #2

This is a simpler method that has only recently become available.

```
let video

function setup()
{
  createCanvas(640, 480)
  background(220)
  video = createCapture(VIDEO, {flipped: true})
  video.hide()
}

function draw()
{
  image(video, 0, 0, width, height)
}
```

Notes

This is just a simple technique which is especially useful when creating machine learning examples (see AI tutorial).


Algorithmic
Art
Module F
Unit #4
Image Files

Module F Unit #4: image files

We can save our work as sketches and share the full-screen version online, but it would be better if we could save our creations as an image or a video (and GIF). Then we could share our hard-earned work so others can share in our brilliance.

This unit gives you the mechanisms to save your work in various formats.

Creating a png (or jpg) image

We can create a simple PNG or JPEG image of the canvas using some very simple code. Yet we can also create PNG images that are transparent, which could be very useful if you want to add interactive characters in a game or some dynamic artwork.

Sketch F4.1 bubbles

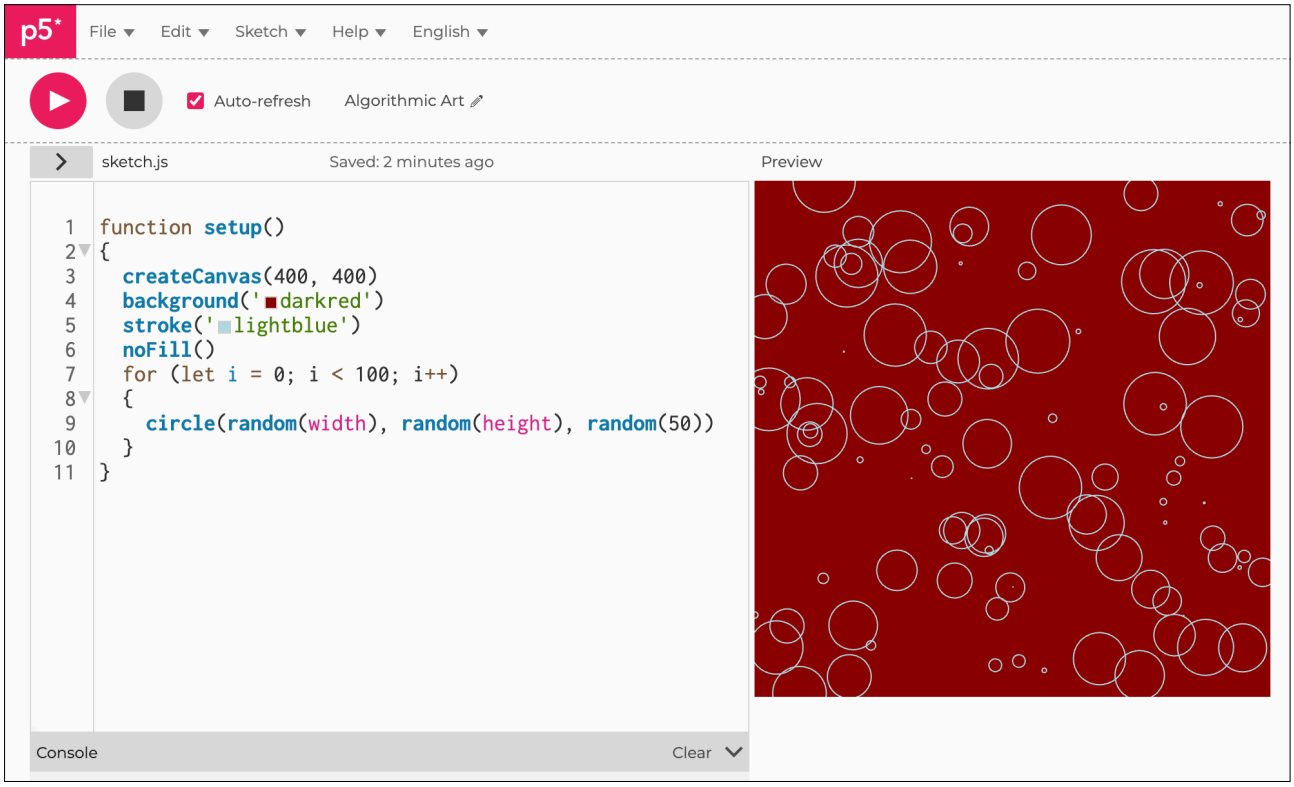
First, we will create a small canvas of bubbles.

```
function setup()
{
  createCanvas(400, 400)
  background('darkred')
  stroke('lightblue')
  noFill()
  for (let i = 0; i < 100; i++)
  {
    circle(random(width), random(height), random(50))
  }
}
```

Notes

100 randomly sized and positioned circles.

Figure F4.1



Sketch F4.2 save canvas as a .png image

When you click on the canvas, you will save an image of the canvas as a .png image.

```
function setup()
{
  createCanvas(400, 400)
  background('darkred')
  stroke('lightblue')
  noFill()
  for (let i = 0; i < 100; i++)
  {
    circle(random(width), random(height), random(50))
  }
}
```

```
function mouseClicked()
{
  save('bubbles.png')
}
```

Notes

You can also save this as a `.jpg` image by replacing `png` with `jpg`.

Challenge

Save as a `.jpg`

Sketch F4.3 png object image

We can do something interesting with a png image. We can remove the background and have what we draw on the canvas as an image. Then we can use that in another sketch. But first, we need to make some adjustments. I have changed the canvas size and the thickness of the lines.

```
let img

function setup()
{
  img = createCanvas(200, 200)
  stroke('lightblue')
  strokeWeight(2)
  noFill()
  for (let i = 0; i < 100; i++)
  {
    circle(random(width), random(height), random(50))
  }
}

function mouseClicked()
{
  save(img, 'bubbles', 'png')
}
```

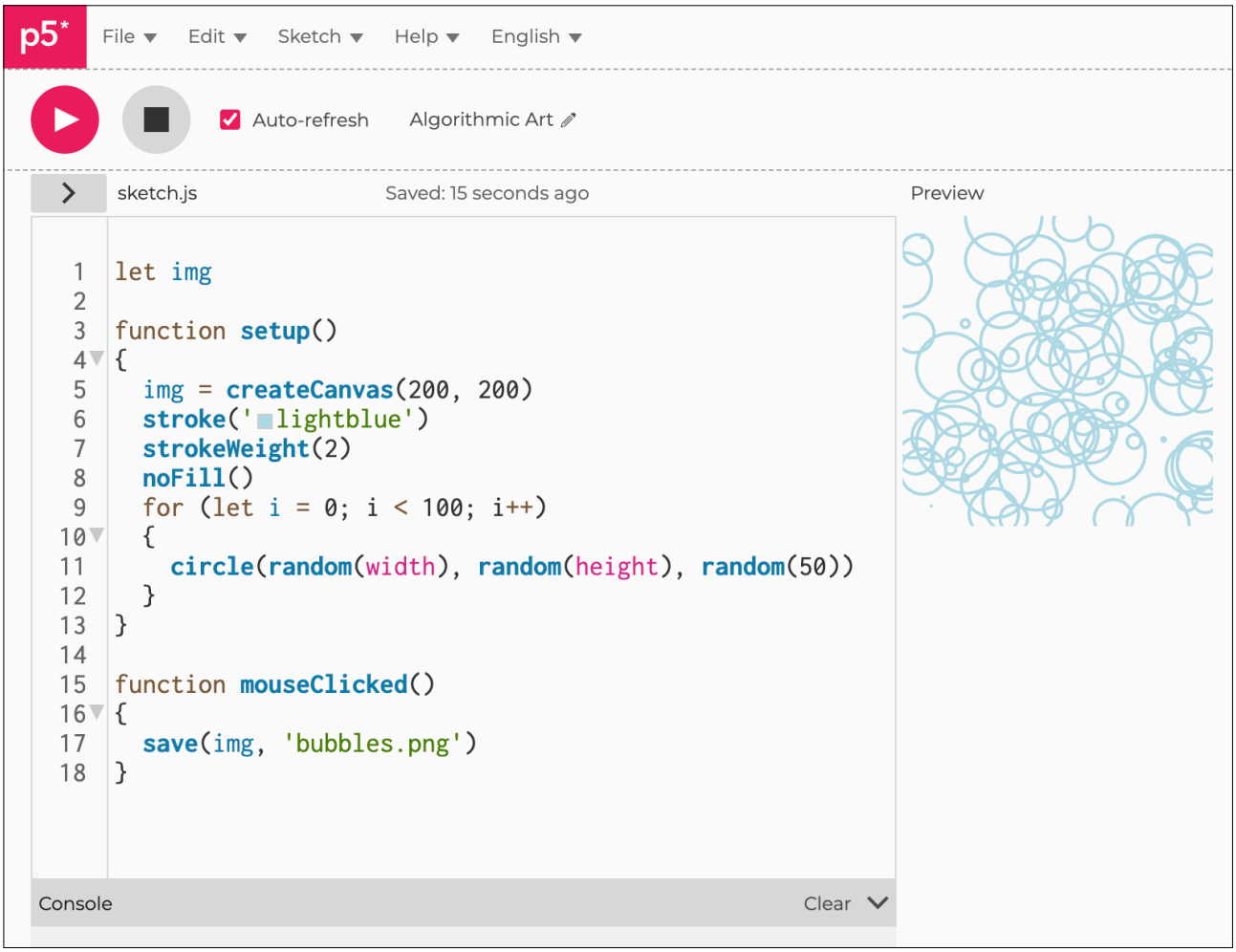
Notes

When you click on the canvas now, you will get another file called bubbles, probably with a suffix (2). Note where it is. Most likely in the downloads folder.

Challenge

Create a button to click on to save the png file.

Figure F4.3



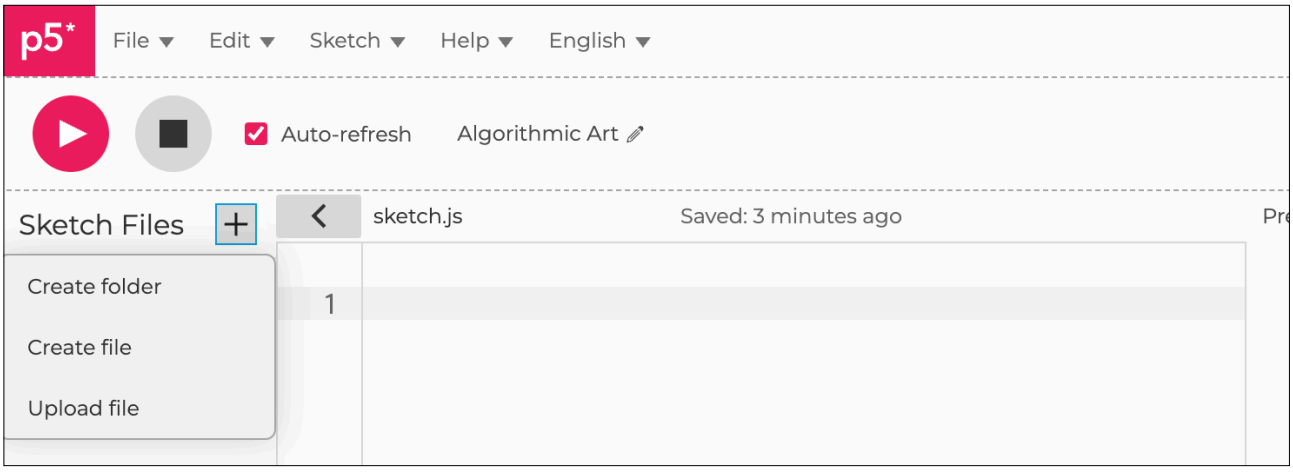
Uploading the image

To add the image, we need to upload it as a file. To do that, we click on the side arrow next to where it says sketch.js, shown in Fig.1 below.

Figure 1: rectangular arrow button

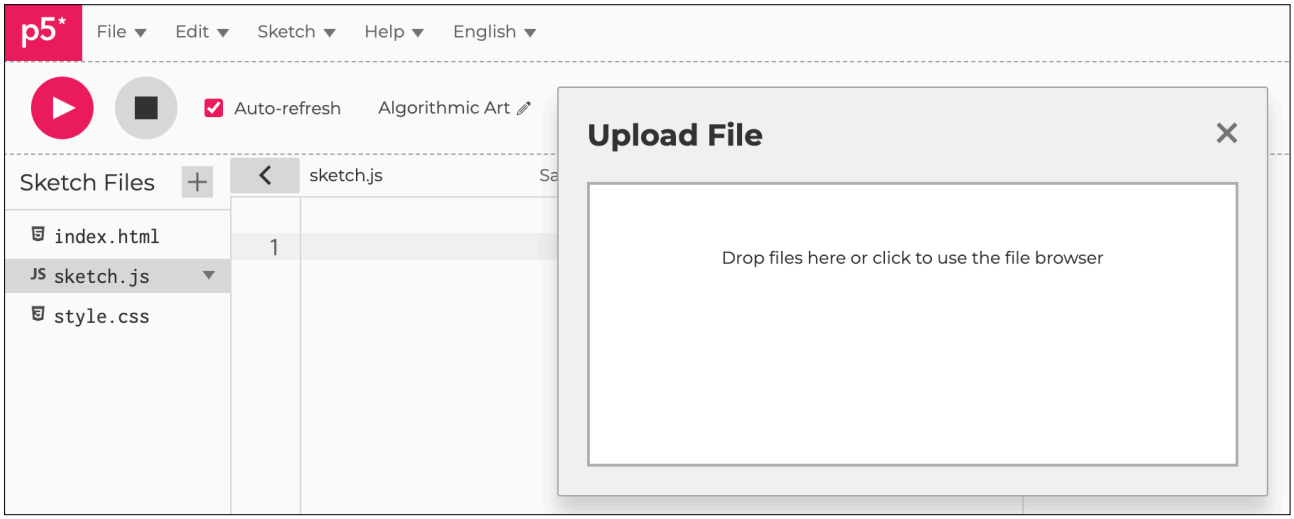
When you click on that arrow, it opens up a list of files. To add anything, you need to click on the + sign at the top, as shown in Figure 2 below.

Figure 2: the + means to add a file



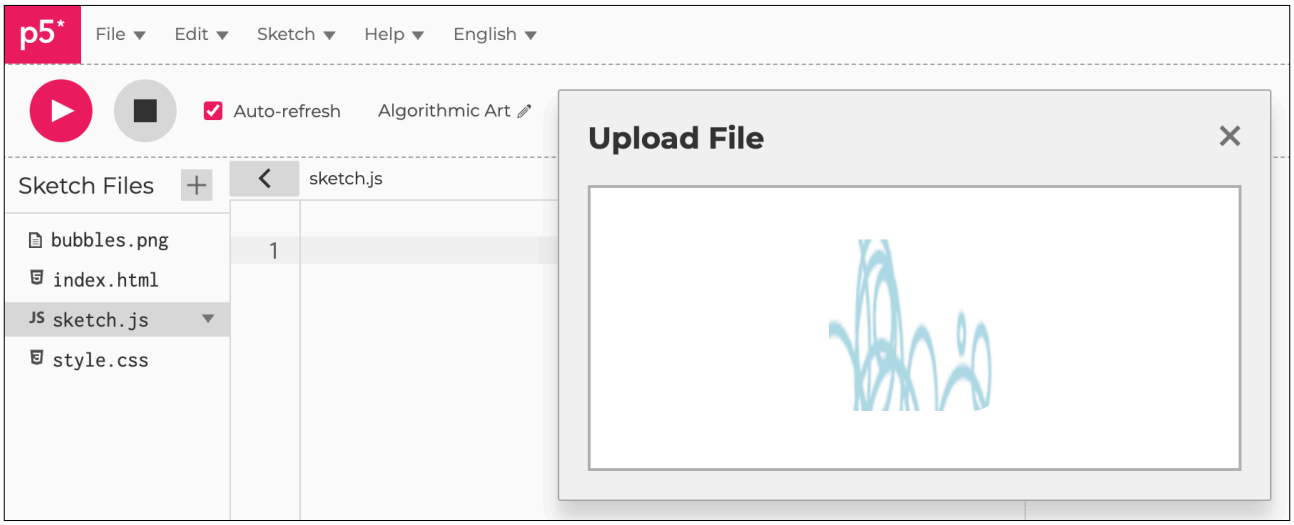
Click on the upload file, and a box appears. You can either click inside that box to browse to where your bubble file is or drag and drop it if it is easily accessible. See Fig. 3 below.

Figure 3: drop or browse window box



Once you have selected the file, then you should see it appear in the upload file box with a big tick inside it. See Fig.4 below. Notice also that the name of the file plus the extension is now listed in the list of files. If you have other versions of the png you can rename it using the arrow next to it with an additional drop-down list.

Figure 4: file (image) uploaded



Sketch F4.4 uploading the bubbles png

This will upload the image file bubbles.png. It places the image (at a size of your choosing) onto the canvas and follows the movement of the mouse.

```
let img

async function setup()
{
  createCanvas(400, 400)
  imageMode(CENTER)
  img = await loadImage('bubbles.png')
}

function draw()
{
  background('darkred')
  image(img, mouseX, mouseY, 100, 100)
}
```

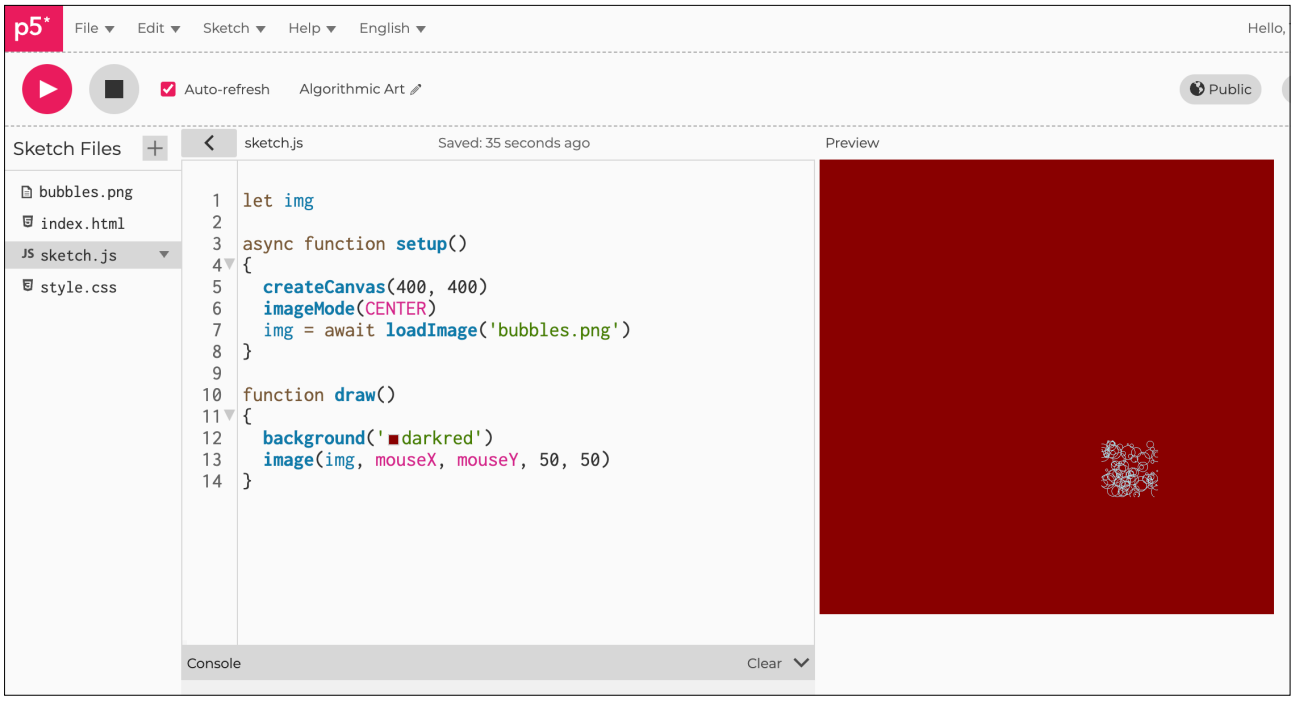
Notes

The main takeaway is that there is no background.

Challenge

Create another shape that you can manipulate by rotating or bouncing around the canvas.

Figure F4.4



Creating a gif

You may well be familiar with GIF images, which are those mini videos that repeat continuously. We can create one quite easily using the `.gif` extension as we save the file as an image. If you want to be very clever, you could count the number of frames so that it repeats seamlessly.

Sketch F4.5 rotating torus

This should be pretty familiar to you.

```
function setup()
{
  createCanvas(400, 400, WEBGL)
  rectMode(CENTER)
  frameRate(30)
}

function draw()
{
  background('darkred')
  fill('yellow')
  lights()
  rotateX(frameCount * 0.02)
  rotateY(frameCount * 0.03)
  torus(width * 0.2, width * 0.07, 32, 32)
}
```

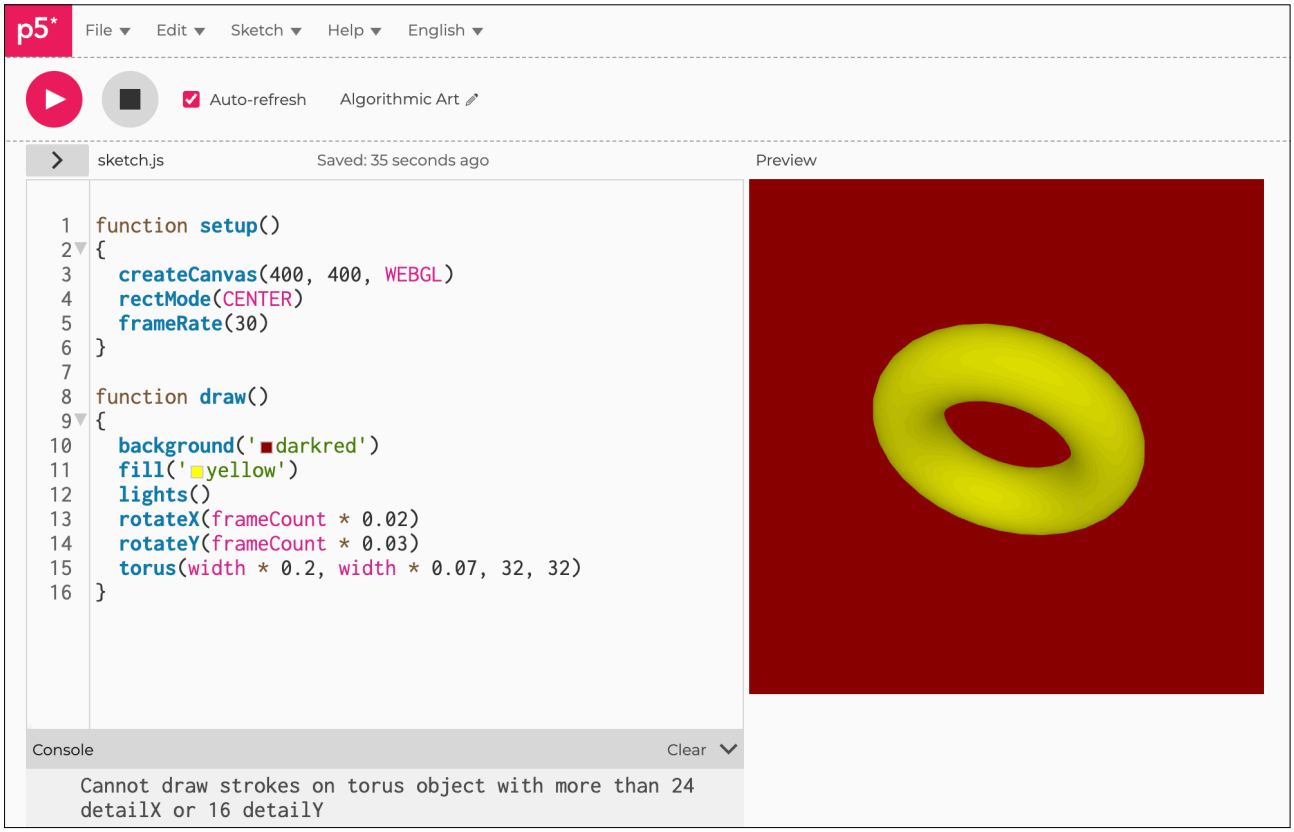
Notes

The only new addition is specifying the frame rate. This is useful if you want your GIF to end where it starts, so that it seems like a continuous motion with no breaks.

Challenge

Create your own.

Figure F4.5



Sketch F4.6 the space bar

A simple keyPressed() function that waits for the space bar to be pressed (" ").

```
function setup()
{
  createCanvas(400, 400, WEBGL)
  rectMode(CENTER)
  frameRate(30)
}

function draw()
{
  background('darkred')
  fill('yellow')
  lights()
  rotateX(frameCount * 0.02)
  rotateY(frameCount * 0.03)
  torus(width * 0.2, width * 0.07, 32, 32)
}
```

```
function keyPressed()
{
  if (key == " ")
  {
    // something happens
  }
}
```

Notes

Nothing will happen yet.

Sketch F4.7 options and frames

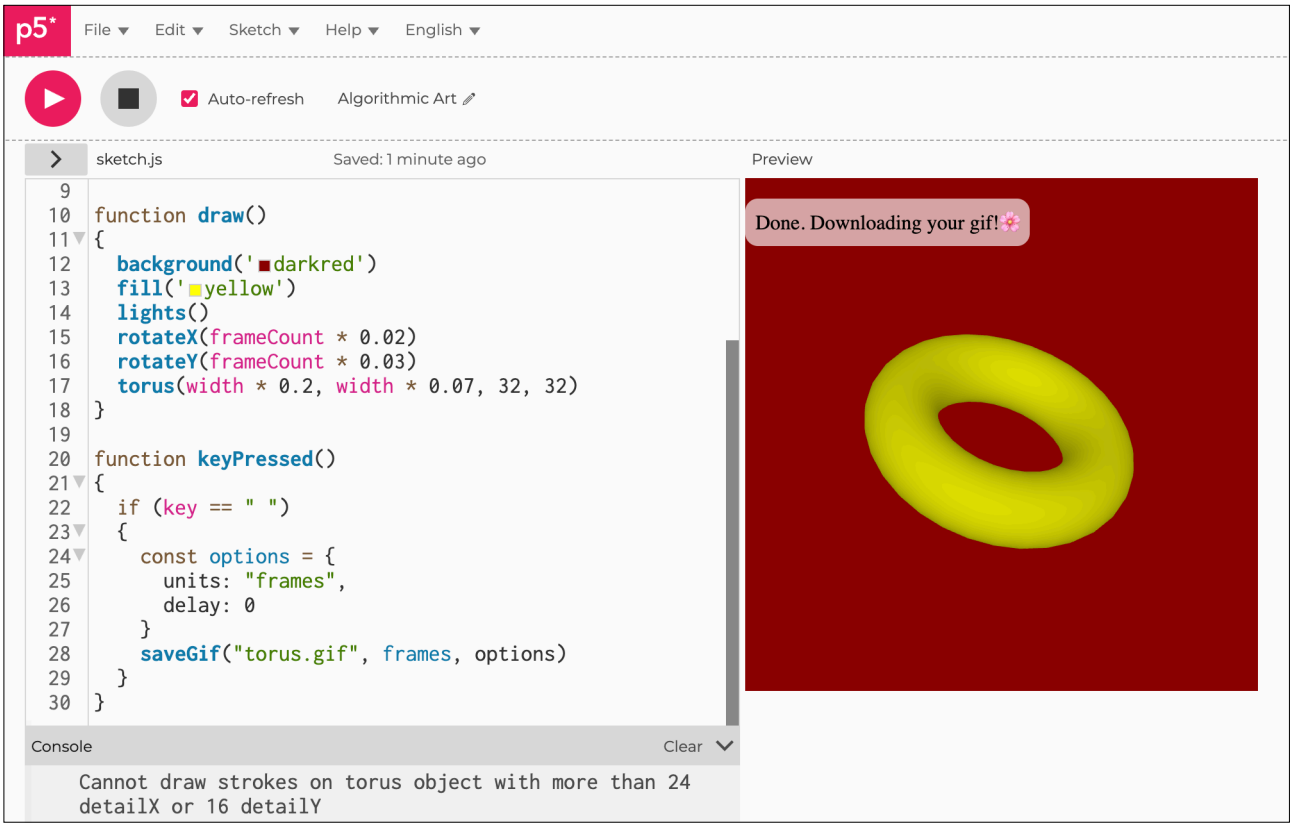
We are going to record and create a GIF. We can give the GIF a number of options: the number of frames it will last for, and if we want any delay before it starts. We have already set the `frameRate()` to `30`. We will record for `120` frames and have zero delay.

```
let frames = 120

function setup()
{
  createCanvas(400, 400, WEBGL)
  rectMode(CENTER)
  frameRate(30)
}

function draw()
{
  background('darkred')
  fill('yellow')
  lights()
  rotateX(frameCount * 0.02)
  rotateY(frameCount * 0.03)
  torus(width * 0.2, width * 0.07, 32, 32)
}

function keyPressed()
{
  if (key == " ")
  {
    const options = {
      units: "frames",
      delay: 0
    }
    saveGif("torus.gif", frames, options)
  }
}
```

Notes

The GIF will be downloaded somewhere; you might have to search for the file, but it is there somewhere.

Figure F4.7



Using mp4 capture

This is a nice way to generate an mp4 video of just the canvas, which means you don't have to crop it later. There are other formats available, but mp4 is pretty universal, and in this example, it will be the default.

Sketch F4.8 capture index.html

We need to add in the lines of code to make this library work. I have set it to mp4 because that works with my computer. You can choose other formats if you wish.

```
<!DOCTYPE html>
<html lang="en"><head>
    <script src="https://cdn.jsdelivr.net/npm/p5@2.2.0/lib/p5.js"></script>
    <script src="https://cdn.jsdelivr.net/npm/p5.capture@1.6"></script>
    <script>
      P5Capture.setDefaultOptions({
        format: "mp4",
        framerate: 30,
        quality: 1,
        verbose: true,
        disableScaling: true,
      });
    </script>
    <link rel="stylesheet" type="text/css" href="style.css">
    <meta charset="utf-8">
  </head>
  <body>
    <main>
    </main>
    <script src="sketch.js"></script>
</body></html>
```

Notes

As you can see, there are other options in the list that you can set. These are the default, and for now, that does the job.

Challenge

Visit https://github.com/tapioca24/p5.capture to see more detailed information.

Sketch F4.9 capture main sketch

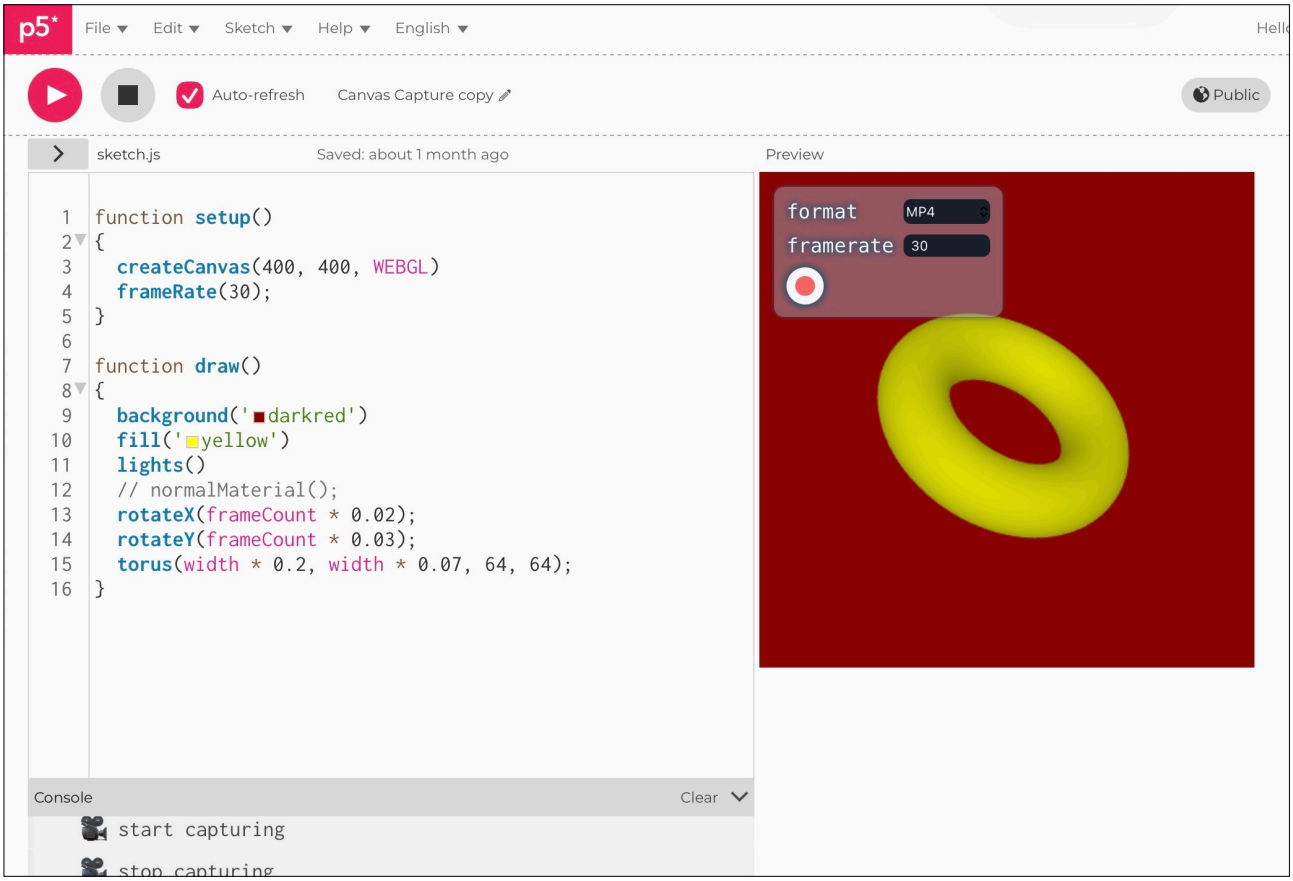
Using the same sketch as before.

```
function setup()
{
  createCanvas(400, 400, WEBGL)
  frameRate(30)
}

function draw()
{
  background('darkred')
  fill('yellow')
  lights()
  rotateX(frameCount * 0.02)
  rotateY(frameCount * 0.03)
  torus(width * 0.2, width * 0.07, 32, 32)
}
```

Notes

You will get a small box on the canvas; it won't be in the final video. You can change the format, although I recommend doing that in the options setup in the `index.html` file.

Figure F4.9



Sketch F4.10 playing the video

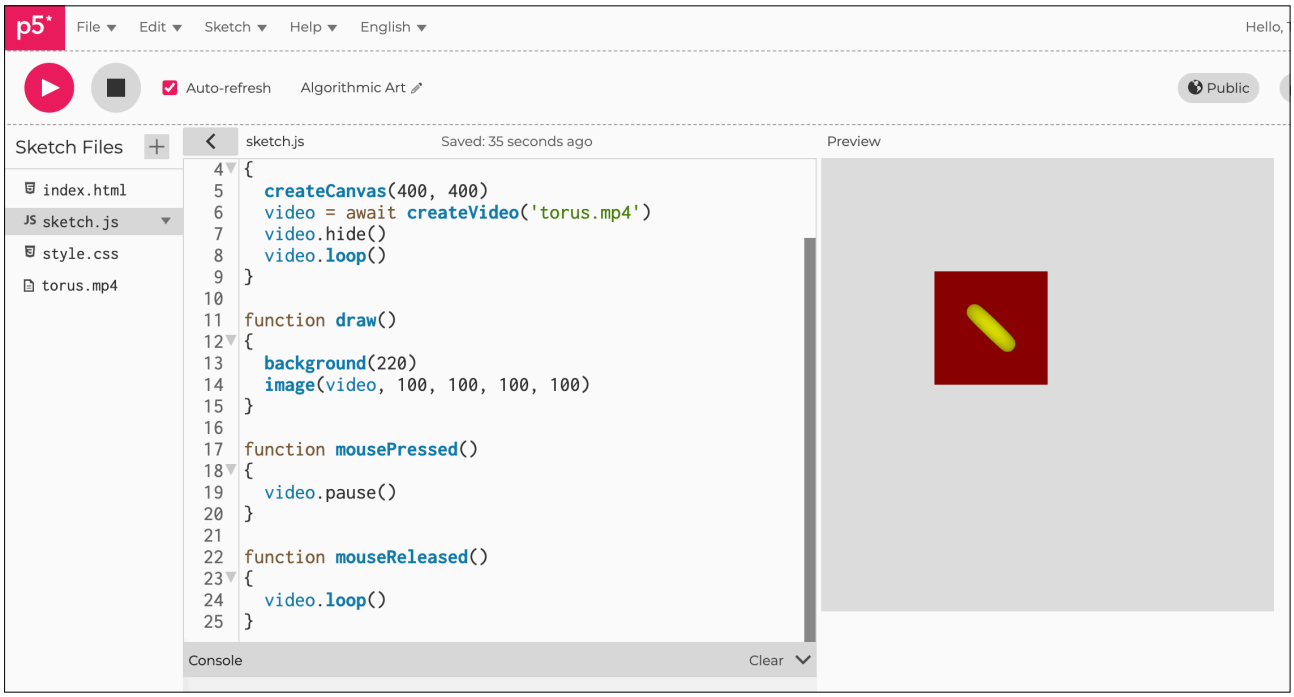
Once you have saved the video, you can include it in a sketch. You upload it just like any other file. You might want to rename it, as I have done. We use `async` and `await` so that we are sure that the video is loaded before we use it. What I have shown below is just a suggestion.

```
let video

async function setup()
{
  createCanvas(400, 400)
  video = await createVideo('torus.mp4')
  video.hide()
  video.loop()
}

function draw()
{
  background(220)
  image(video, 100, 100, 100, 100)
}

function mousePressed()
{
  video.pause()
}

function mouseReleased()
{
  video.loop()
}
```

Notes

You will most likely use the mp4 on social media or a website so that you can promote your artwork.

Figure F4.10




Algorithmic
Art
Module F
Unit #5
The Sound
of Music

Module F Unit #5: the sound of music

The music I have provided is called Tangerine Jam by Symplocarpush.

It was downloaded for free from: www.freemusicarchive.org
for non-commercial use. You can choose from a range of styles, but be aware that the size of the file matters; if it is too big, it will not upload because there is a limit on the size of the file.

Download the sound file from my website; just click on the link provided.

Upload the MP3 file as you have done with other types of files. Rename it `jam` or anything you want.

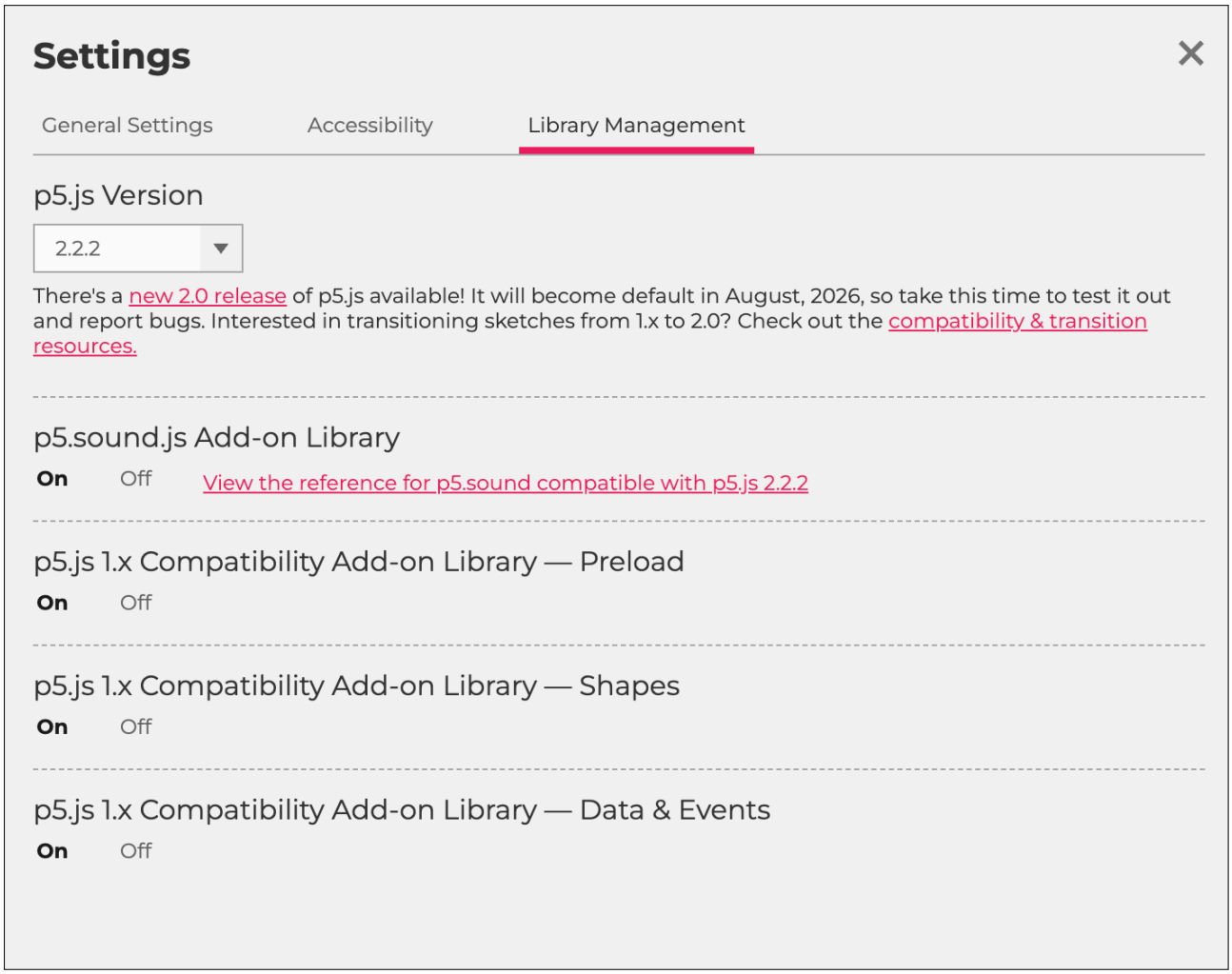
This unit covers aspects such as visualising music and the microphone sound. Although we will be working with `v2` of `p5.js`, we do need to add a library that will allow us some compatibility with p5.js version 1. This is easy to do; remember to do it, even especially if you start a new sketch.

Click on the button that indicates the version you are using; yours might be `1.11.13` or something similar. See Fig. 2.

Figure 2: the version button (top left)

Now select the p5.sound.js Add-on Library in `Settings`, as shown in Fig 3.

Figure 3: settings



Sketch F5.1 uploading

The piece of music I have provided is approximately 24 seconds long. I have called it jam for simplicity. You upload it just the same as a video and an image.

```
let music

async function setup()
{
  createCanvas(400, 400)
  music = await createAudio('jam.mp3')
}

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

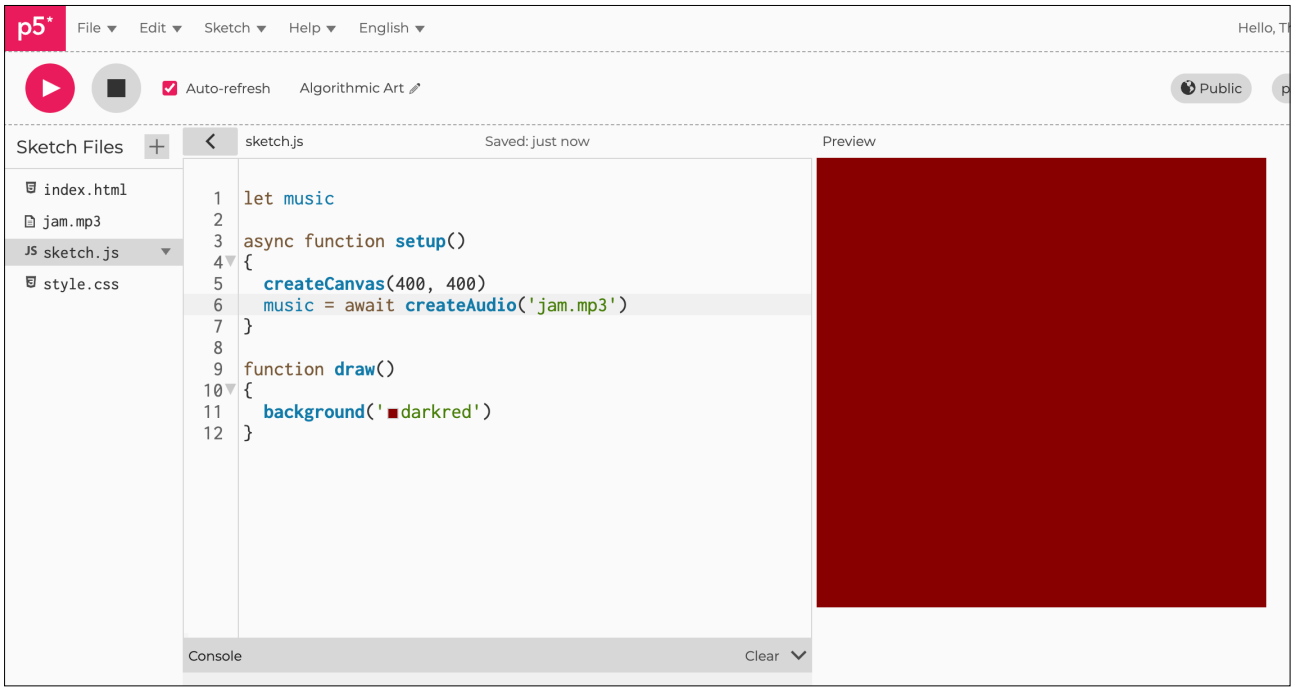
Notes

You can't hear anything yet.

Challenge

You can use your own piece of music if you wish.

Figure F5.1



Sketch F5.2 play automatically

As soon as you run the code, the music will start.

```
let music


async function setup()
{
  createCanvas(400, 400)
  music = await createAudio('jam.mp3')
  music.play()
}


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

Notes

The `play()` function needs to be in the same function as the music download, or you will get an error message.

Challenge

Try it in `draw()`.

Sketch F5.3 music loop on mouse

! Remove `play()` function.
Introducing `loop()` and `stop()`.

```
let music

async function setup()
{
  createCanvas(400, 400)
  music = await createAudio('jam.mp3')
}

function draw()
{
  background('darkred')
}

function mousePressed()
{
  music.loop()
}

function mouseReleased()
{
  music.stop()
}
```

Notes

Click the canvas (and hold the button down), release to stop.

Challenge

Replace `music.loop()` with `music.play()`.

Sketch F5.4 slider for rate

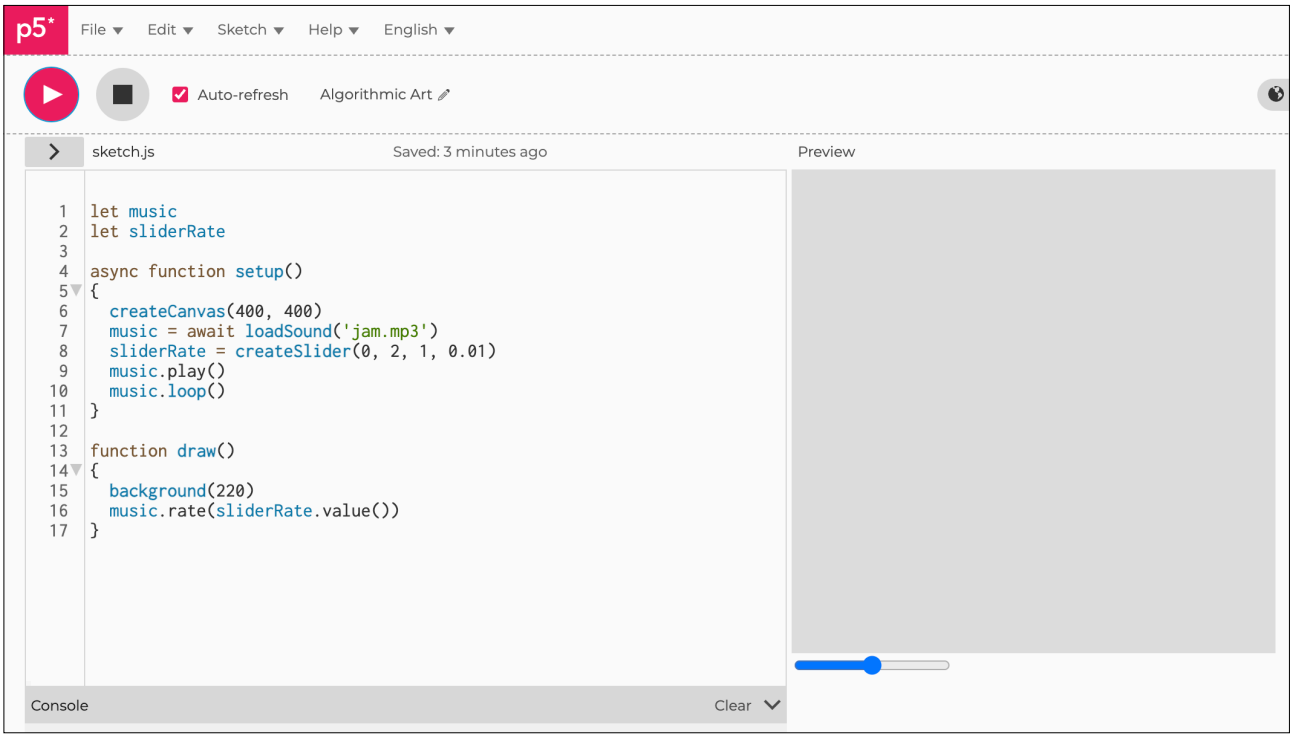
The volume slider is removed and a `rate` slider is added. The `rate` is the speed of playback. The default is `1`, half speed is `0.5`, double is `2`, etc.

```
let music
let sliderRate

async function setup()
{
  createCanvas(400, 400)
  music = await loadSound('jam.mp3')
  sliderRate = createSlider(0, 2, 1, 0.01)
  music.play()
  music.loop()
}

function draw()
{
  background(220)
  music.rate(sliderRate.value())
}
```

Notes

As you move the slider, you should hear the music speed up or slow down.

Figure F5.4



Sketch F5.5 toggle play/pause

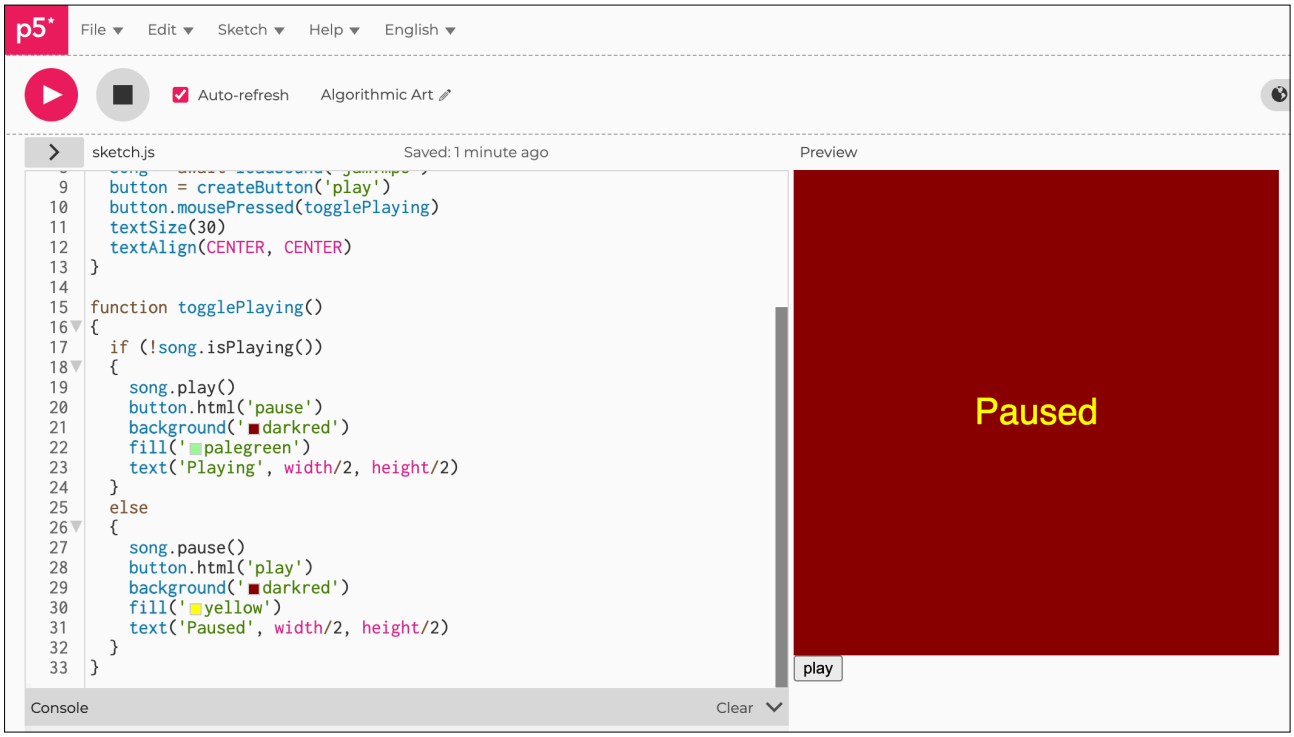
Just a simple toggle button.

```
let song
let button

async function setup()
{
  createCanvas(400, 400)
  background('darkred')
  song = await loadSound('jam.mp3')
  button = createButton('play')
  button.mousePressed(togglePlaying)
  textSize(30)
  textAlign(CENTER, CENTER)
}

function togglePlaying()
{
  if (!song.isPlaying())
  {
    song.play()
    button.html('pause')
    background('darkred')
    fill('palegreen')
    text('Playing', width/2, height/2)
  }
  else
  {
    song.pause()
    button.html('play')
    background('darkred')
    fill('yellow')
    text('Paused', width/2, height/2)
  }
}
```

Notes

Click to play or pause.

Figure F5.5



Sketch F5.6 jump

Jumps to a specific time in seconds. In this case, it jumps to the 20th second of the 23-second-long track.

```
let music
let button
let jumpButton

async function setup()
{
  createCanvas(400, 400)
  background(220)
  music = await loadSound('jam.mp3')
  button = createButton('play')
  button.mousePressed(togglePlaying)
  jumpButton= createButton('jump')
  jumpButton.mousePressed(jumpMusic)
}

function jumpMusic()
{
  music.jump(20)
}

function togglePlaying()
{
  if (!music.isPlaying())
  {
    music.play()
  }
  else
  {
    music.stop()
  }
}
```

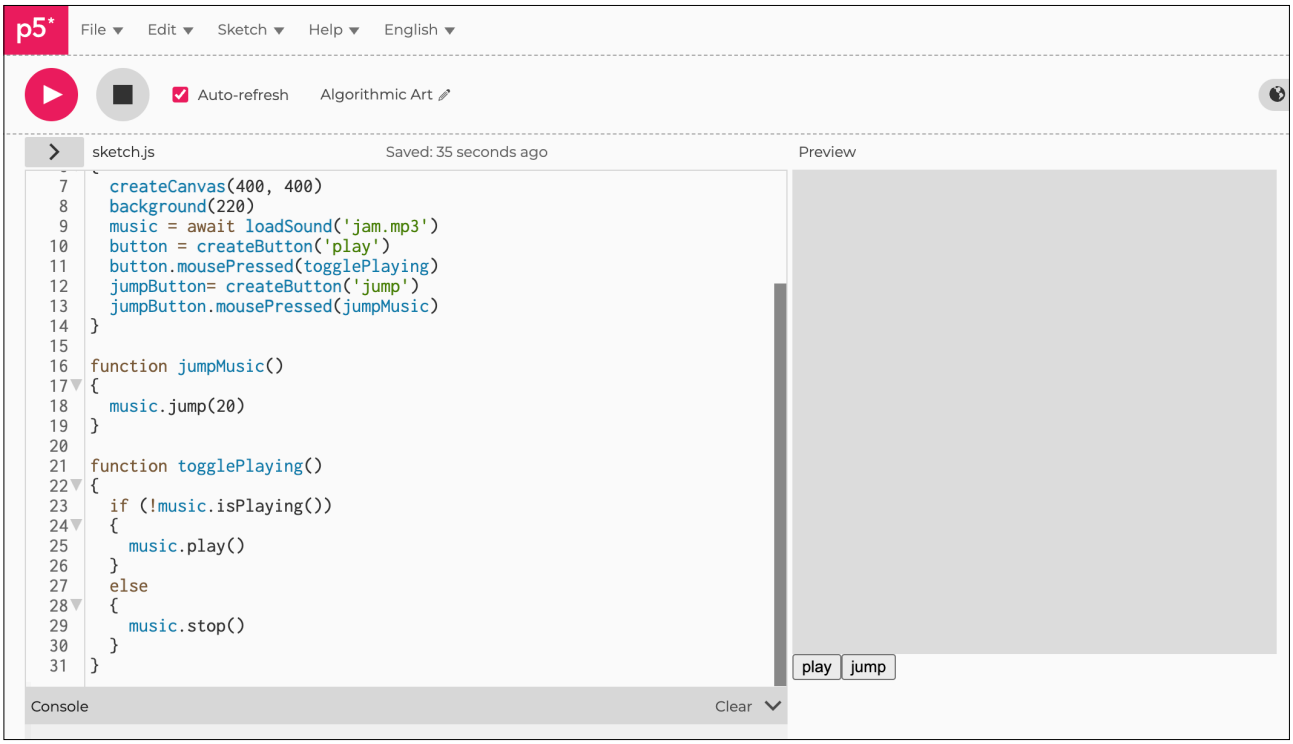
Notes

It might be hard to hear the jump.

Challenge

Try several jump buttons.

Figure F5.6



Sketch F5.7 music length

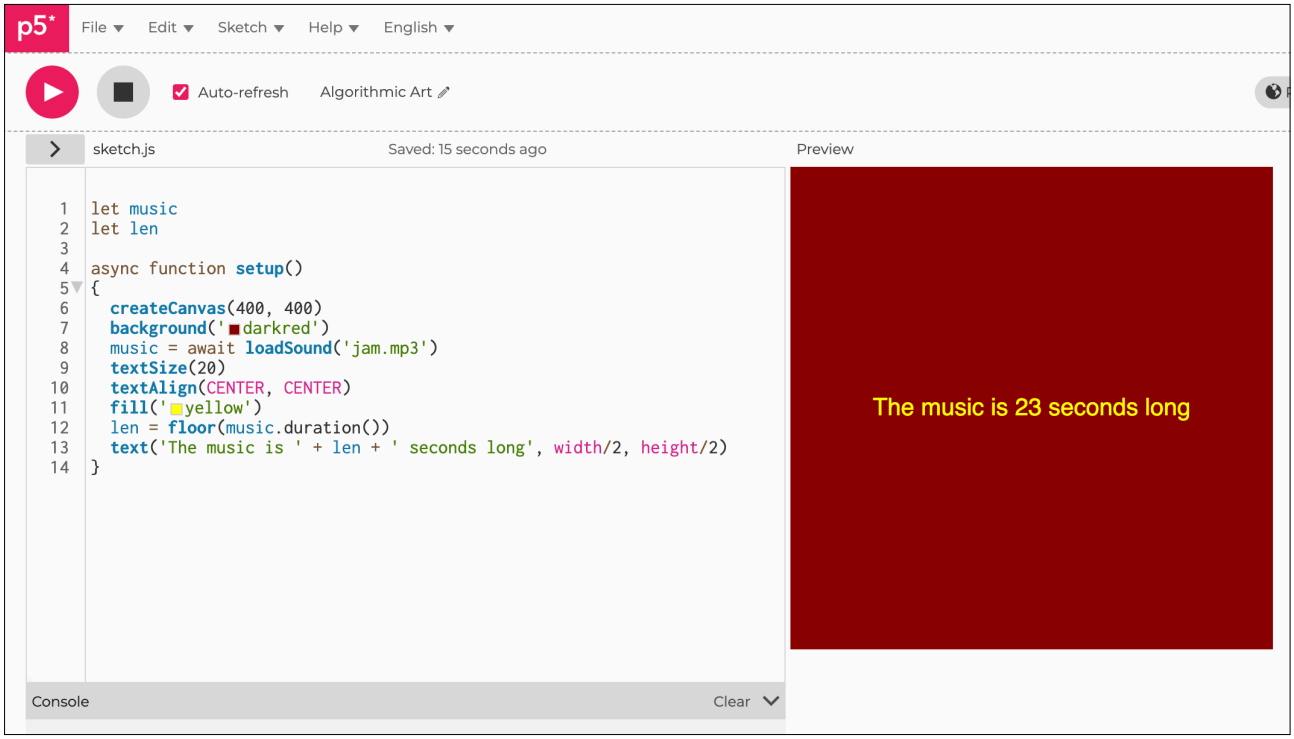
Using this function, we can measure the length of the piece of music or sound file.

```
let music
let len

async function setup()
{
  createCanvas(400, 400)
  background('darkred')
  music = await loadSound('jam.mp3')
  textSize(20)
  textAlign(CENTER, CENTER)
  fill('yellow')
  len = floor(music.duration())
  text('The music is ' + len + ' seconds long', width/2, height/2)
}
```

Notes

You should get the correct value of 23 seconds.

Figure F5.7



Get the volume

To get the volume, we need to use the p5 `Amplitude()` function. We need to connect the music to the `Amplitude()` function.

Sketch F5.8 getting the volume

We create two variables, `amp` and `vol`. We connect the amplitude to the music (sound file). From the amplitude, we get the volume (level). This is then fed into the diameter of the circle. The volume (`vol`) values can be quite small (between `0` and `1`), so we multiply by `500` just to make it a bit more dramatic.

```
let music
let amp
let vol

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

  music = await loadSound('jam.mp3')
  amp = new p5.Amplitude()
  music.connect(amp)
  music.play()
}

function draw()
{
  background('darkred')
  fill('yellow')
  vol = amp.getLevel()
  circle(200, 200, vol * 500)
}
```

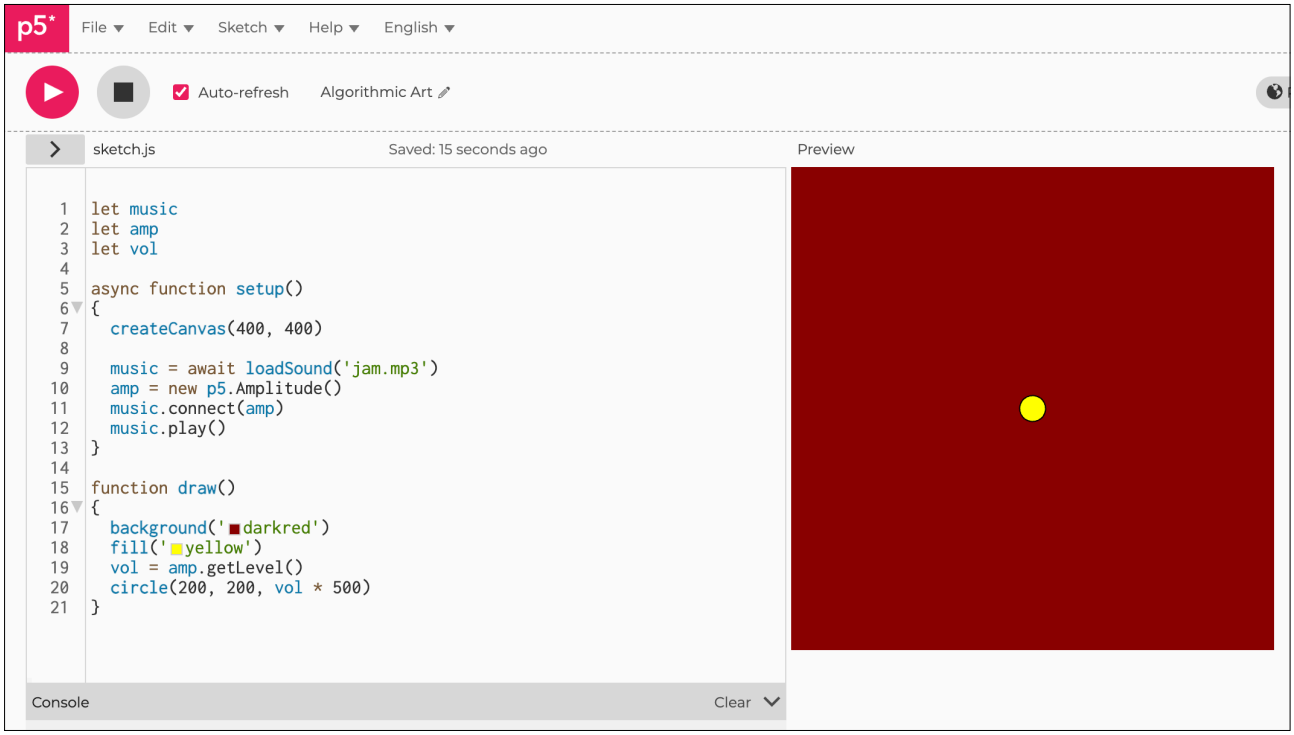
Notes

The circle should rapidly enlarge and constrict as the music is played.

Challenge

Times it by the width instead.

Figure F5.8



Sketch F5.9 toggle button

We add a button to toggle play or not play. We have a conditional if statement to check if it is not playing already. We move the `play()` function into the new function `togglePlaying()`.

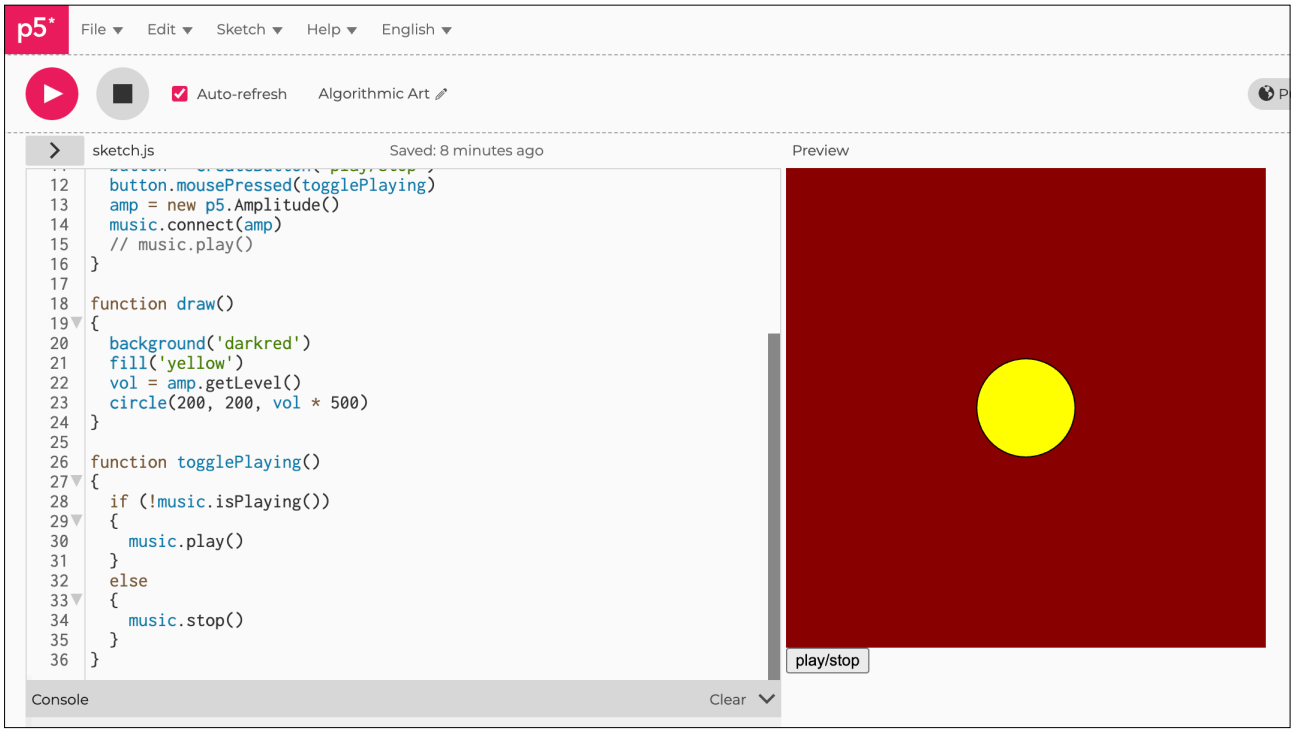
```
let music
let amp
let vol
let button

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

  music = await loadSound('jam.mp3')
  button = createButton('play/stop')
  button.mousePressed(togglePlaying)
  amp = new p5.Amplitude()
  music.connect(amp)
  // music.play()
}

function draw()
{
  background('darkred')
  fill('yellow')
  vol = amp.getLevel()
  circle(200, 200, vol * 500)
}

function togglePlaying()
{
  if (!music.isPlaying())
  {
    music.play()
  }
  else
  {
```

```
    music.stop()
  }
}
```

Notes

This is nothing new.

Figure F5.9



Sketch F5.10 a bit of style

Adding a bit of style.

```
let music
let amp
let vol
let button
let rDot
let x = 200

async function setup()
{
  createCanvas(400, 400)
  background('darkred')
  music = await loadSound('jam.mp3')
  button = createButton('play/stop')
  button.mousePressed(togglePlaying)
  amp = new p5.Amplitude()
  music.connect(amp)
  noFill()
}

function draw()
{
  noStroke()
  fill(255, 50)
  vol = amp.getLevel()
  rDot = random(-5, 5)
  x += rDot
  circle(x, height + 20 - vol*2*height, 20)
}

function togglePlaying()
{
  if (!music.isPlaying())
  {
    music.play()
  }
```

```
  else
  {
    music.stop()
    x = 200
  }
}
```

Notes

Should have a nice effect.

Challenge

Create something with lots of colour, maybe linked to the volume.

Figure F5.10

Sketch F5.11 newish sketch

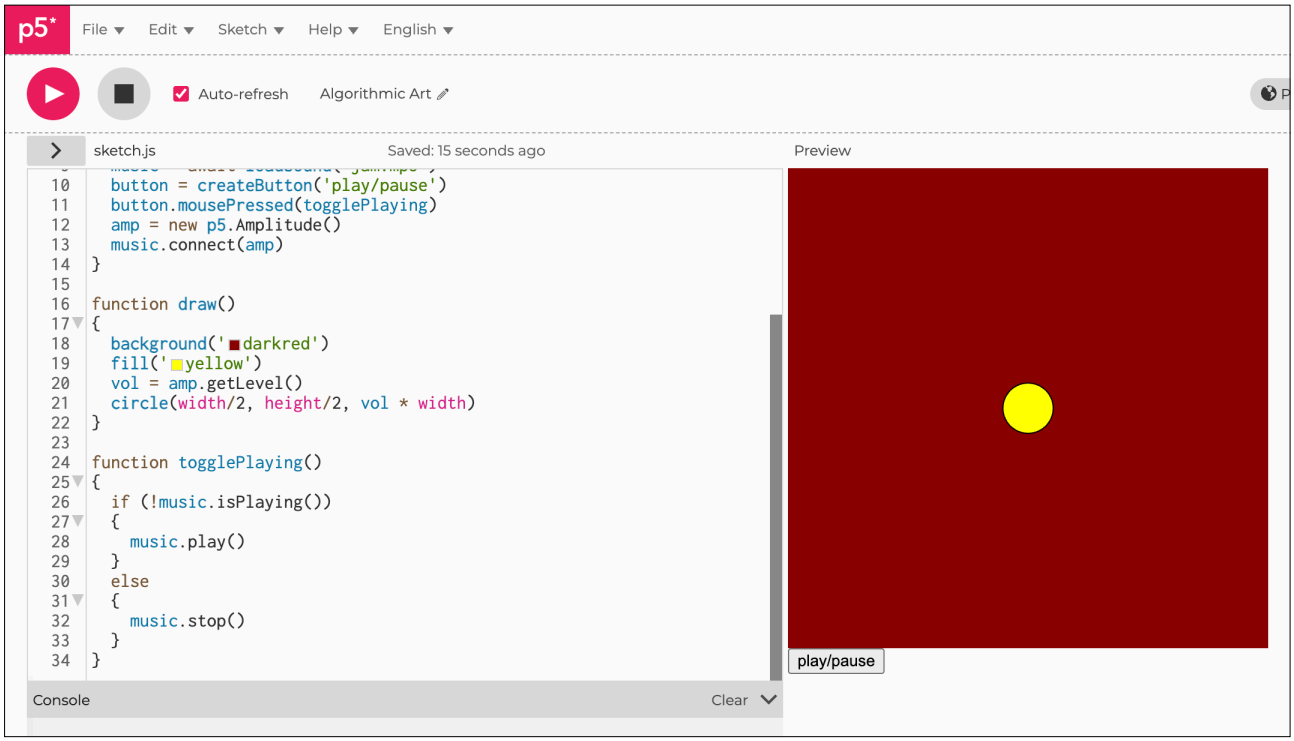
! Remove unnecessary code and put background into the draw() function.

```
let music
let amp
let vol
let button

async function setup()
{
  createCanvas(400, 400)
  music = await loadSound('jam.mp3')
  button = createButton('play/pause')
  button.mousePressed(togglePlaying)
  amp = new p5.Amplitude()
  music.connect(amp)
}

function draw()
{
  background('darkred')
  fill('yellow')
  vol = amp.getLevel()
  circle(width/2, height/2, vol * width)
}

function togglePlaying()
{
  if (!music.isPlaying())
  {
    music.play()
  }
  else
  {
    music.stop()
  }
}
```

Notes

The circle will now respond to the music.

Figure F5.11



Sketch F5.12 an array of data

We want to create an analysis graph of the sound file. To do this, we need to create an array to store all this data. We `console.log()` the length to make sure it is actually collecting the data.

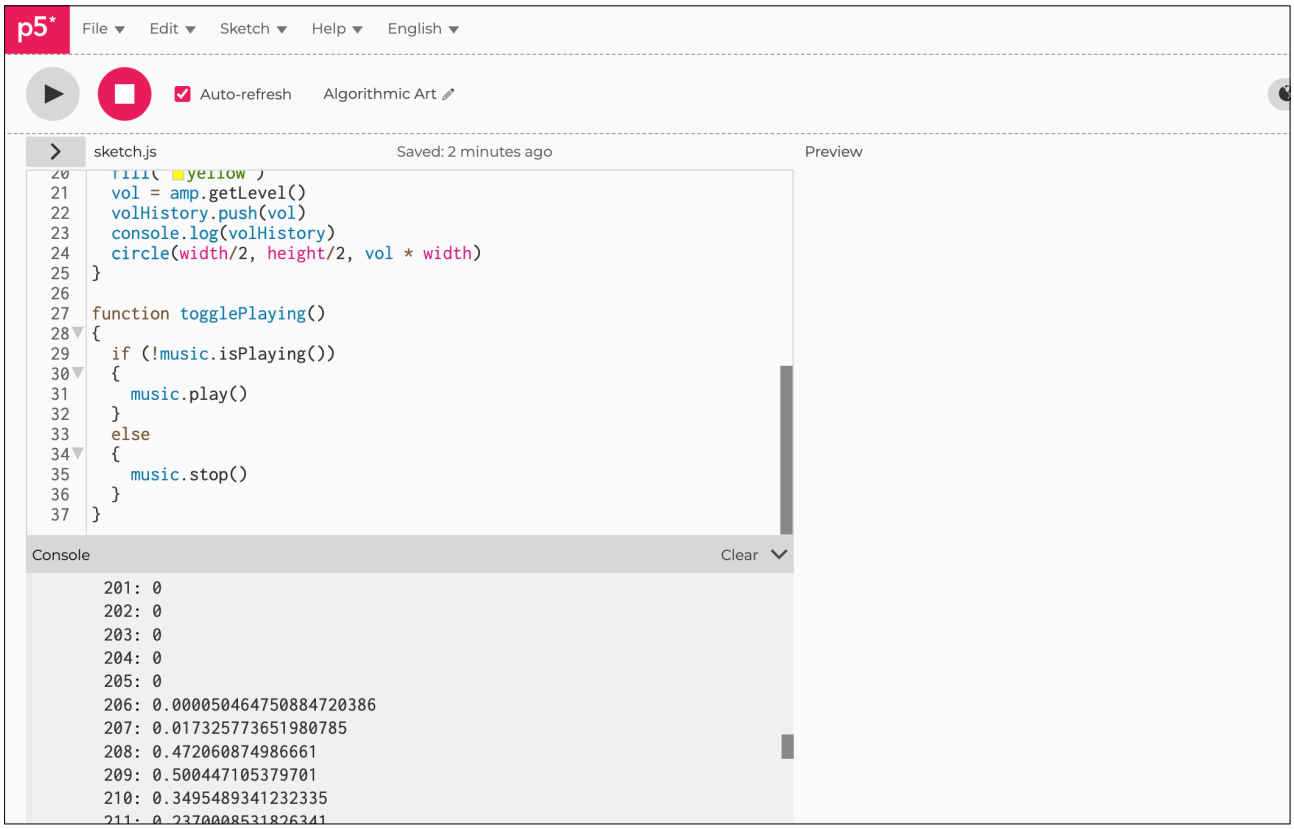
```
let music
let amp
let vol
let button
let volHistory = []

async function setup()
{
  createCanvas(400, 400)
  music = await loadSound('jam.mp3')
  button = createButton('play/pause')
  button.mousePressed(togglePlaying)
  amp = new p5.Amplitude()
  music.connect(amp)
}

function draw()
{
  background('darkred')
  fill('yellow')
  vol = amp.getLevel()
  volHistory.push(vol)
  console.log(volHistory)
  circle(width/2, height/2, vol * width)
}

function togglePlaying()
{
  if (!music.isPlaying())
  {
    music.play()
  }
  else
```

```
  {
    music.stop()
  }
}
```

Notes

We get a result in the console showing that we are increasing the number of data points; this continues even though the music stops because it will still receive 0.

Figure F5.12



Sketch F5.13 replacing the circle

! Remove the line of code for the circle and the `console.log()`.
Instead, we are going to draw points for the values of the volume.

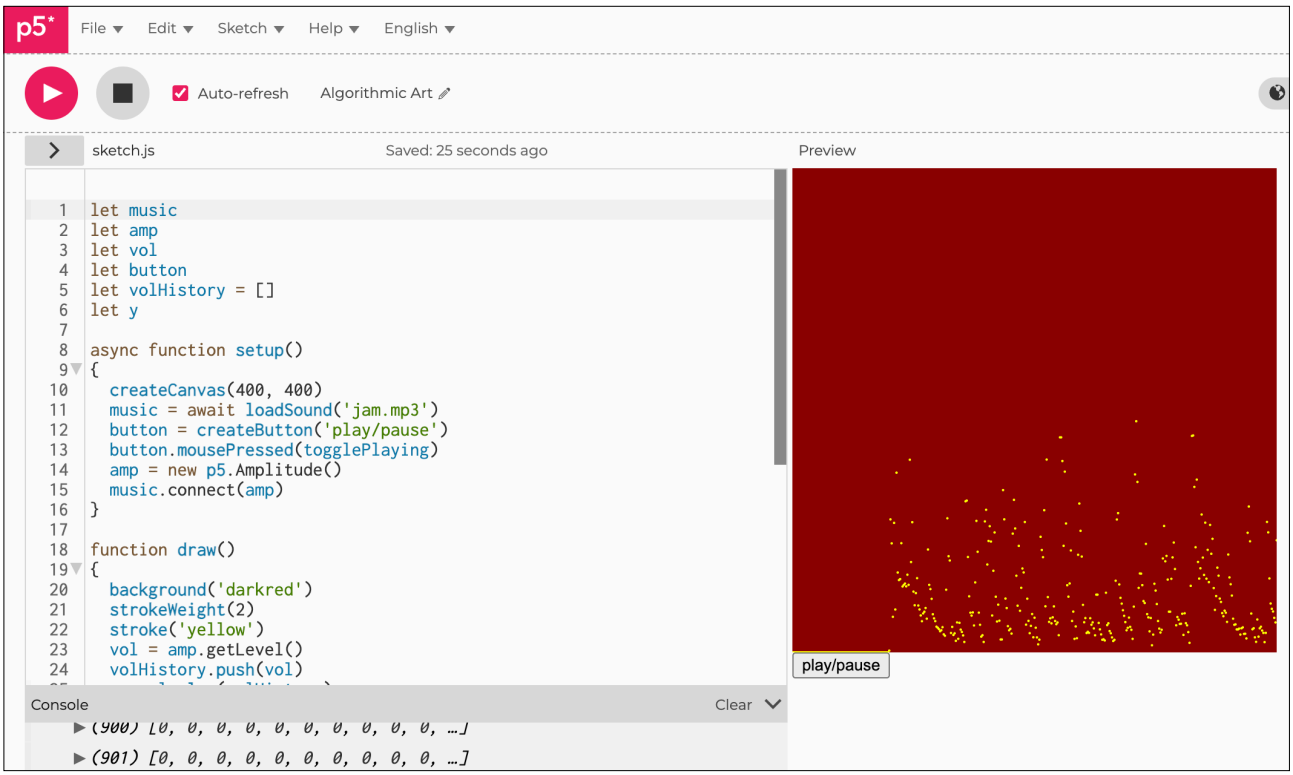
```
let music
let amp
let vol
let button
let volHistory = []
let y

async function setup()
{
  createCanvas(400, 400)
  music = await loadSound('jam.mp3')
  button = createButton('play/pause')
  button.mousePressed(togglePlaying)
  amp = new p5.Amplitude()
  music.connect(amp)
}

function draw()
{
  background('darkred')
  strokeWeight(2)
  stroke('yellow')
  vol = amp.getLevel()
  volHistory.push(vol)
  console.log(volHistory)
  for (let i = 0; i < volHistory.length; i++)
  {
    y = map(volHistory[i], 0, 1, height, 0)
    point(i, y)
  }
}

function togglePlaying()
{
```

```
  if (!music.isPlaying())
  {
    music.play()
  }
  else
  {
    music.stop()
  }
}
```

Notes

We now get a series of dots for the relevant values of the volume at any one moment in time. The dots run off the edge as the music continues to play; we will fix that in due course.

Figure F5.13



Sketch F5.14 a line of dots

! Remove `console.log()` as we don't need it anymore.
Replacing the `point()` with `vertex()`, we can draw a line joining the dots.

```
let music
let amp
let vol
let button
let volHistory = []
let y

async function setup()
{
  createCanvas(400, 400)
  music = await loadSound('jam.mp3')
  button = createButton('play/pause')
  button.mousePressed(togglePlaying)
  amp = new p5.Amplitude()
  music.connect(amp)
}

function draw()
{
  background('darkred')
  strokeWeight(2)
  stroke('yellow')
  noFill()
  vol = amp.getLevel()
  volHistory.push(vol)
  beginShape()
  for (let i = 0; i < volHistory.length; i++)
  {
    y = map(volHistory[i], 0, 1, height, 0)
    vertex(i, y)
  }
  endShape()
}
```

```
function togglePlaying()
{
  if (!music.isPlaying())
  {
    music.play()
  }
  else
  {
    music.stop()
  }
}
```

Notes

We get a line this time but it's still static and disappears off the edge!

Figure F5.14

Sketch F5.15 a continuous line

Now the graph scrolls.

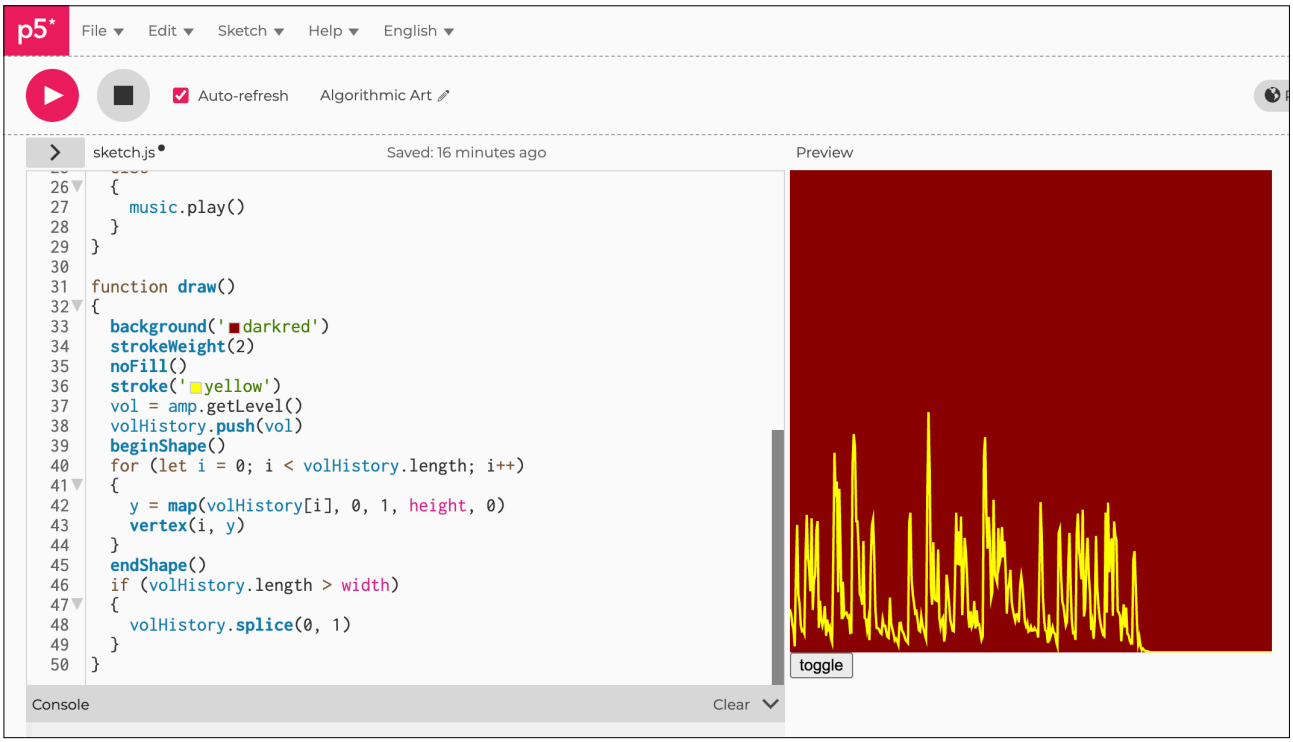
```
let music
let amp
let vol
let button
let volHistory = []
let y

async function setup()
{
  createCanvas(400, 400)
  music = await loadSound('jam.mp3')
  button = createButton('play/pause')
  button.mousePressed(togglePlaying)
  amp = new p5.Amplitude()
  music.connect(amp)
}

function draw()
{
  background('darkred')
  strokeWeight(2)
  stroke('yellow')
  noFill()
  vol = amp.getLevel()
  volHistory.push(vol)
  beginShape()
  for (let i = 0; i < volHistory.length; i++)
  {
    y = map(volHistory[i], 0, 1, height, 0)
    vertex(i, y)
  }
  endShape()
  if (volHistory.length > width)
  {
```

```
    volHistory.splice(0, 1)
  }
}

function togglePlaying()
{
  if (!music.isPlaying())
  {
    music.play()
  }
  else
  {
    music.stop()
  }
}
```

Notes

It continues to move even when the music stops.

Figure F5.15



Sketch F5.16 a rotary version

The line is now rotating.

```
let music
let amp
let vol
let button
let volHistory = []
let y
let r
let x

async function setup()
{
  createCanvas(400, 400)
  music = await loadSound('jam.mp3')
  button = createButton('play/pause')
  button.mousePressed(togglePlaying)
  amp = new p5.Amplitude()
  music.connect(amp)
  angleMode(DEGREES)
}

function draw()
{
  background('darkred')
  strokeWeight(2)
  stroke('yellow')
  noFill()
  vol = amp.getLevel()
  volHistory.push(vol)
  translate(width/2, height/2)
  beginShape()
  for (let i = 0; i < volHistory.length; i++)
  {
    r = map(volHistory[i], 0, 1, 10, width/2)
    x = r * sin(i)
```

```
    y = r * cos(i)
    vertex(x, y)
  }
  endShape()
  if (volHistory.length > 360)
  {
    volHistory.splice(0, 1)
  }
  line(0, 0, 0, width/2)
  fill('darkred')
  circle(0, 0, 20)
}

function togglePlaying()
{
  if (!music.isPlaying())
  {
    music.play()
  }
  else
  {
    music.stop()
  }
}
```

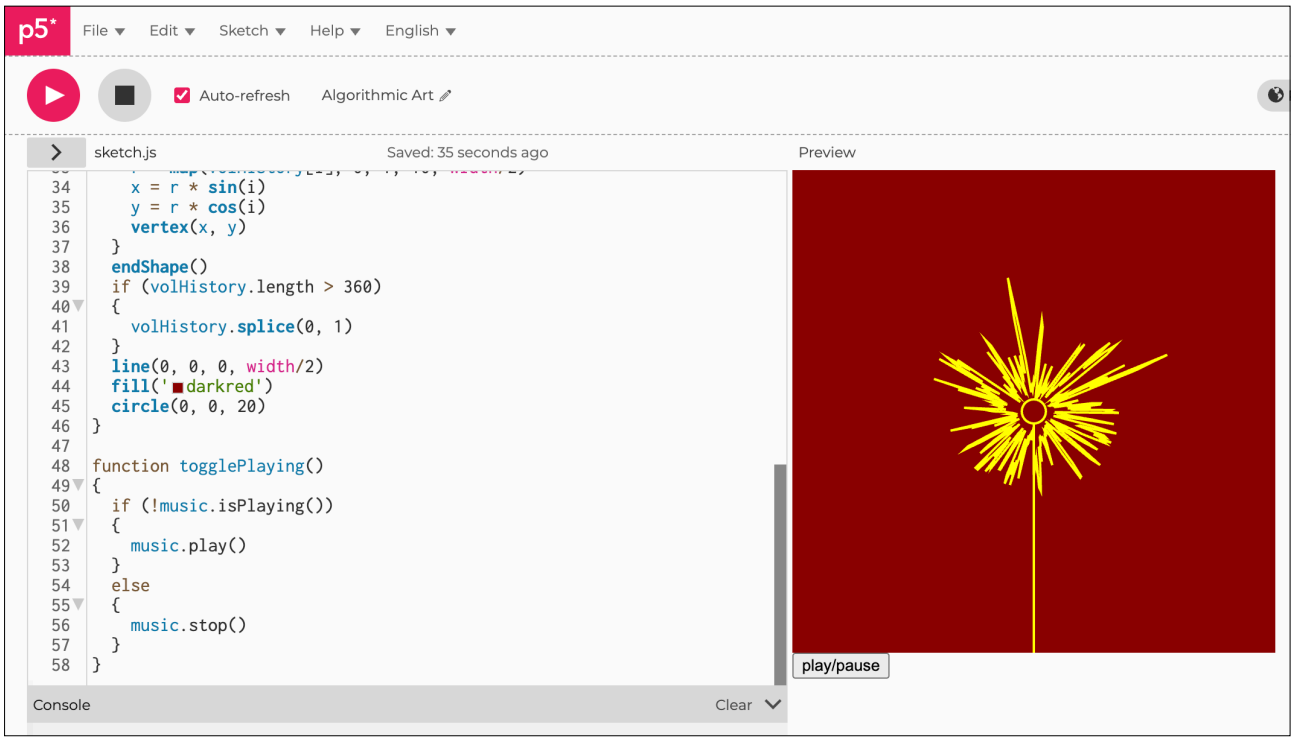
Notes

This has a spinning graph which has to start and finish somewhere. It looks a bit odd, so I have drawn a line to hide the jump.

Challenge

Can you find a better way of representing the sound file?

Figure F5.16




Algorithmic
Art
Module F
Unit #6
Microphone

Module F Unit #6: the microphone

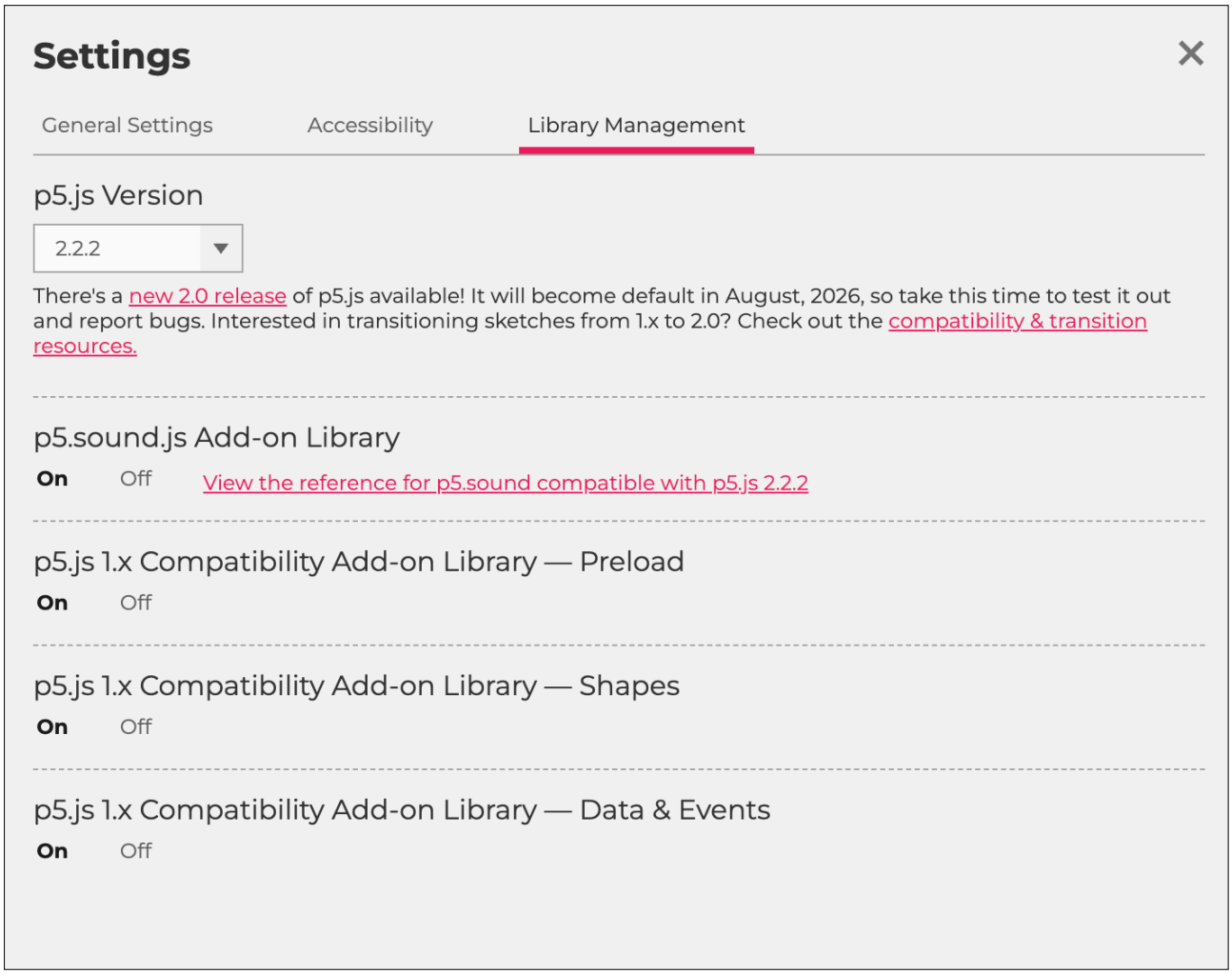
This unit covers aspects such as visualising music and the microphone sound. Although we will be working with v2 of p5.js, we do need to add a library that will allow us some compatibility with p5.js version 1. This is easy to do; remember to do it, even especially if you start a new sketch.

Click on the button that indicates the version you are using; yours might be 1.11.13 or something similar. See Fig. 1.

Figure 1: the version button (top left)

Now select the p5.sound.js Add-on Library in `Settings`, as shown in Fig 2.

Figure 2: settings



Sketch F6.1 the microphone

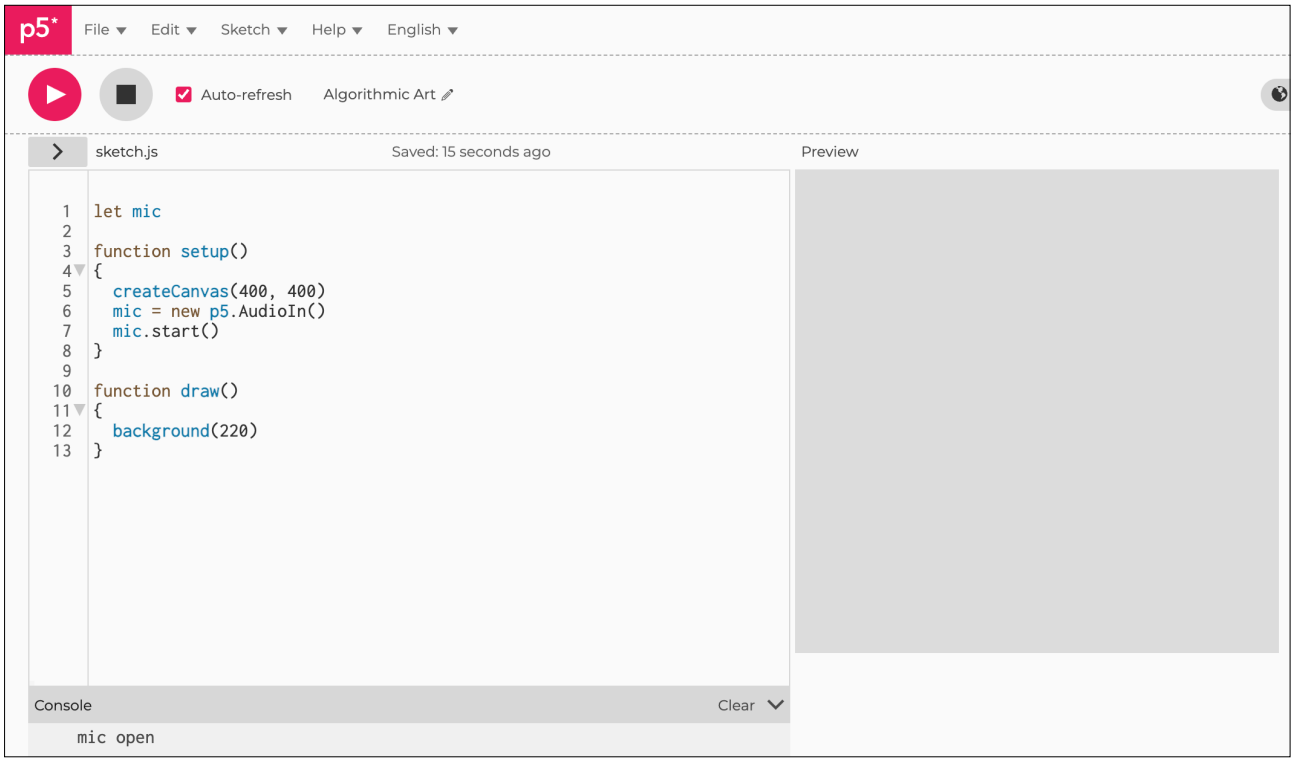
! Our starting sketch.
You may need to give your computer permission to use the microphone for p5.js.

```
let mic

function setup()
{
  createCanvas(400, 400)
  mic = new p5.AudioIn()
  mic.start()
}

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

Notes

We now have the microphone working, hopefully.

Figure F6.1



Sketch F6.2 the amplitude

We need to use the p5.Amplitude() function to get the volume, and connect it to the microphone.

```
let mic
let amp

function setup()
{
  createCanvas(400, 400)
  mic = new p5.AudioIn()
  amp = new p5.Amplitude()
  mic.connect(amp)
  mic.start()
}

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

Notes

This connects the microphone to the built-in amplitude function.

Sketch F6.3 the volume

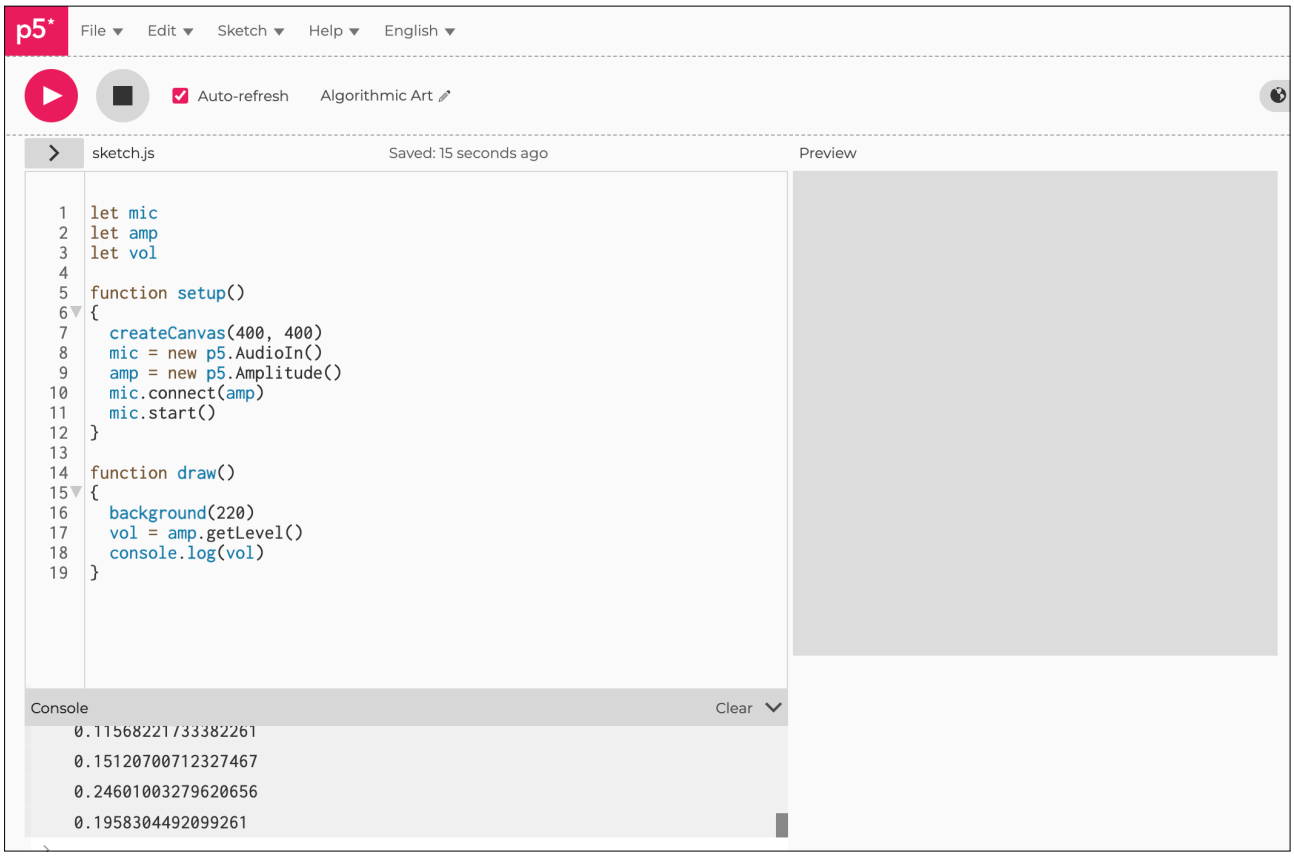
Using the amplitude value, amp, we can get the volume levels.

```
let mic
let amp
let vol

function setup()
{
  createCanvas(400, 400)
  mic = new p5.AudioIn()
  amp = new p5.Amplitude()
  mic.connect(amp)
  mic.start()
}

function draw()
{
  background(220)
  vol = amp.getLevel()
  console.log(vol)
}
```

Notes

The `console.log()` will give us the volume between `0` and `1`.

Figure F6.3



Sketch F6.4 a circle

! Remove the `console.log()`.
We can draw a circle that responds to the noise level the microphone picks up.

```
let mic
let amp
let vol

function setup()
{
  createCanvas(400, 400)
  mic = new p5.AudioIn()
  amp = new p5.Amplitude()
  mic.connect(amp)
  mic.start()
}

function draw()
{
  background(220)
  vol = amp.getLevel()
  vol = map(vol, 0, 1, 0, width)
  circle(200, 200, vol * 2)
}
```

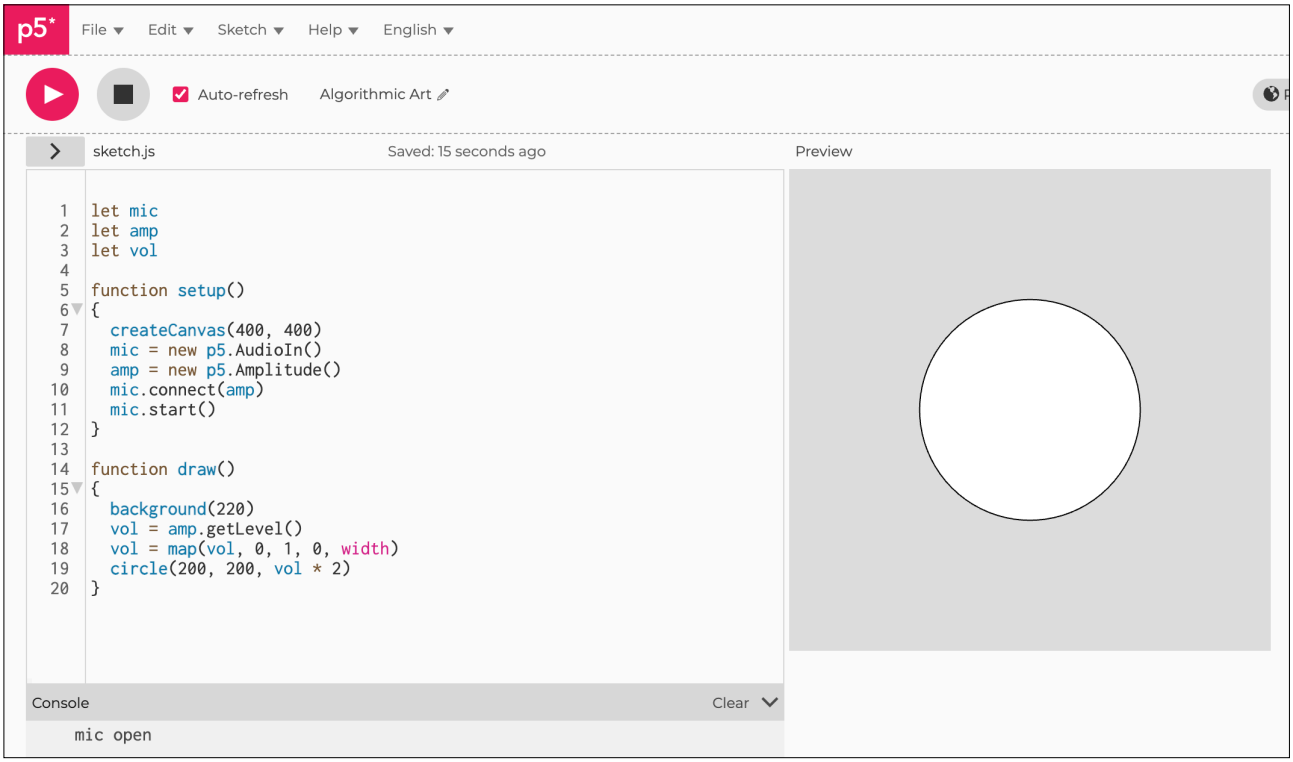
Notes

You can play around with the value of the diameter (`vol`) of the circle if there appears nothing or too much happening.

Challenges

1. Could you connect the `vol` to the colour of a circle, e.g. white to black, or other colours?
2. Constrain the circle between two limits. For instance: `vol = constrain(vol, 50, 200)`.

Figure F6.4



Sketch F6.5 bouncing ball?

! Remove mapping the volume.
We can move the ball with our voice, sort of.

```
let mic
let amp
let vol
let y

function setup()
{
  createCanvas(400, 400)
  mic = new p5.AudioIn()
  amp = new p5.Amplitude()
  mic.connect(amp)
  mic.start()
}

function draw()
{
  background('darkred')
  fill('yellow')
  noStroke()
  vol = amp.getLevel()
  y = height - (vol * height * 5)
  circle(200, y - 50, 100)
}
```

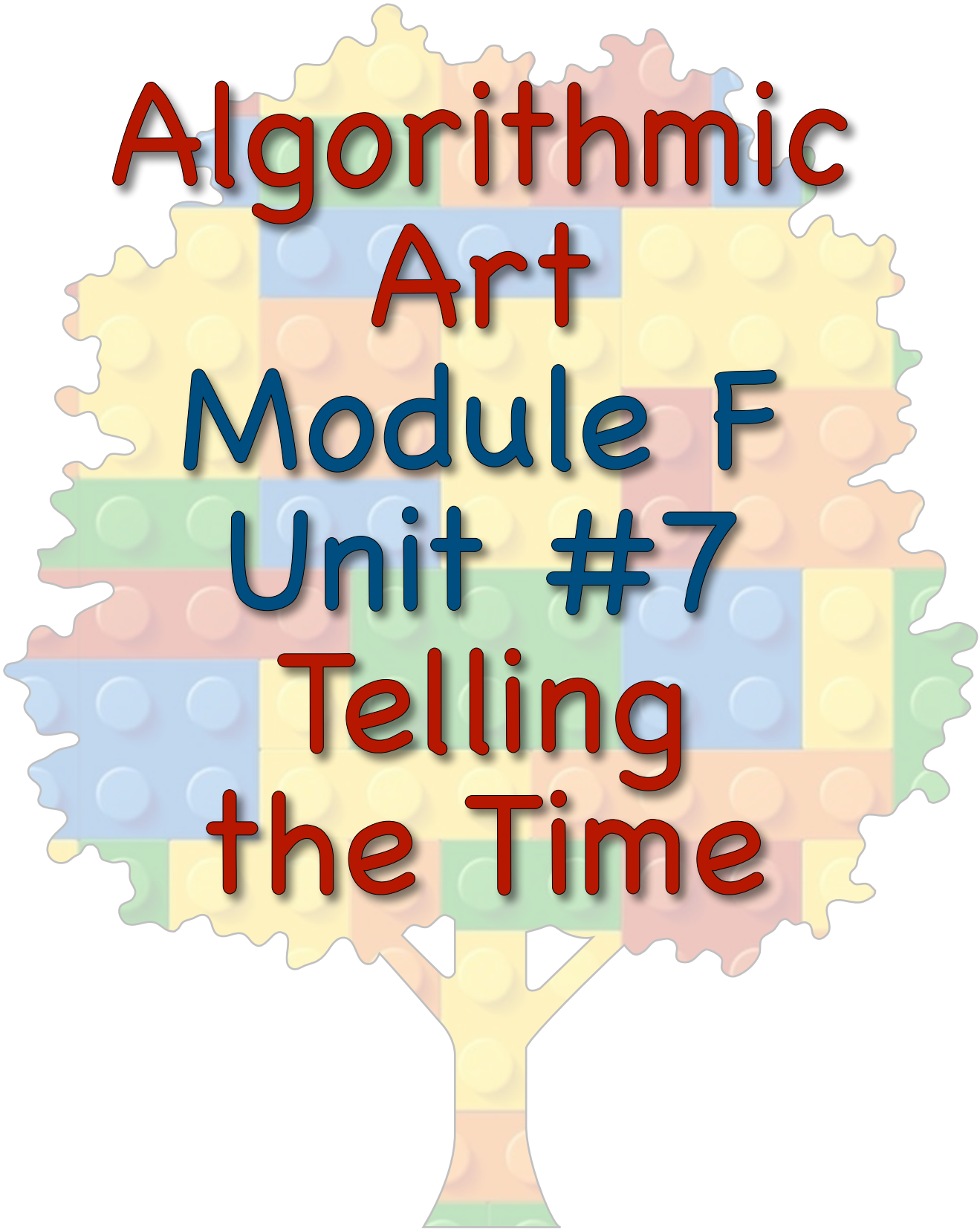
Notes

Every time you make a sound, the ball bounces, or at least moves.

Challenge

Can you create other designs that respond to your voice?

Figure F6.5


Algorithmic
Art
Module F
Unit #7
Telling
the Time

Module F Unit #7: telling the time

A brief look at how you can incorporate the `time` and `date`, and using the `millis()` function to measure the passage of time.

This unit is about getting the time and date from your computer. What it also serves is to look at some other features like text alignment. One challenge that you could try is to create a seven-segment display using the time and a lot of code.

Sketch F7.1 every second counts

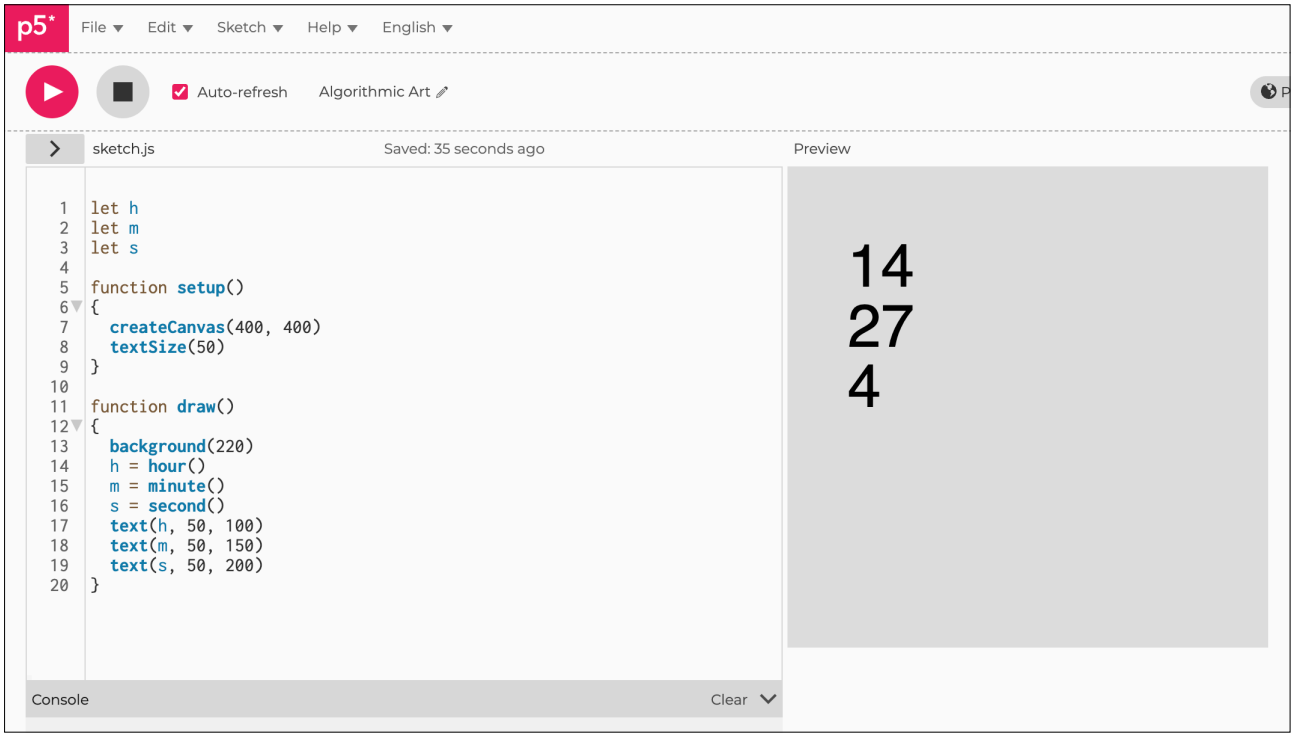
Getting the hour, minute, and seconds of the current time.

```
let h
let m
let s

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

function draw()
{
  background(220)
  h = hour()
  m = minute()
  s = second()
  text(h, 50, 100)
  text(m, 50, 150)
  text(s, 50, 200)
}
```

Notes

Gives the hour (24-hour clock), the minutes, and the seconds. Notice that it gives the minutes and seconds in single figures for below 10.

Figure F7.1



Time functions

`hour()` gives you the current hour.
`minute()` gives you the current minutes.
`second()` gives you the current seconds.

Notes

Collects data from the internet.

Challenge

How would you put text in front (hint: `text('hours: ' + h, 50, 100)`)

Sketch F7.2 a second look

! Replacing `text(m)` and `text(s)`.
A better-looking clock.

```
let h
let m
let s

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

function draw()
{
  background(220)
  h = hour()
  m = minute()
  s = second()
  text(h, 50, 100)
  if (m < 10)
  {
    text('0' + m, 50, 150)
  }
  else
  {
    text(m, 50, 150)
  }
  if (s < 10)
  {
    text('0' + s, 50, 200)
  }
  else
  {
    text(s, 50, 200)
  }
}
```

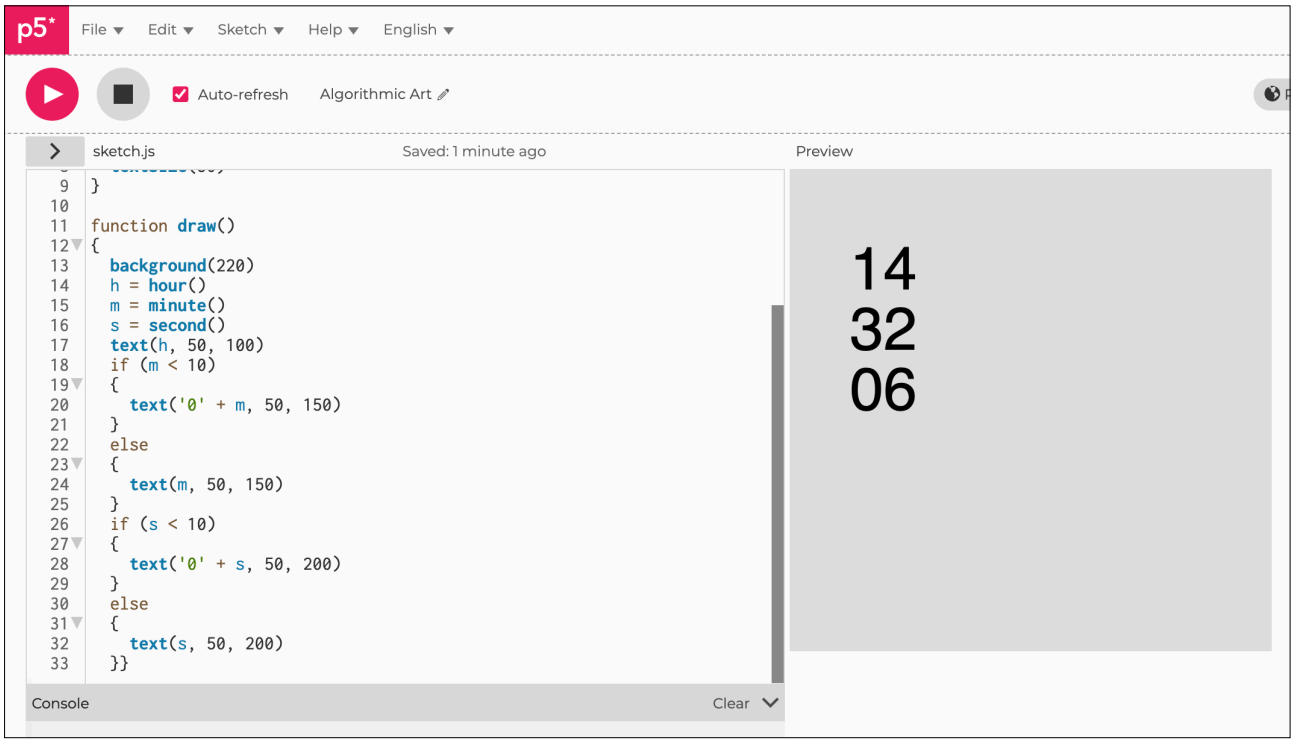
Notes

Improves the look by putting the 0 in front of the digit if less than 10 for minutes and seconds.

Challenge

1. Now put 0 before the hour (if earlier than 10 o'clock).
2. What if you don't want the 24-hour clock?

Figure F7.2



Sketch F7.3 what a year

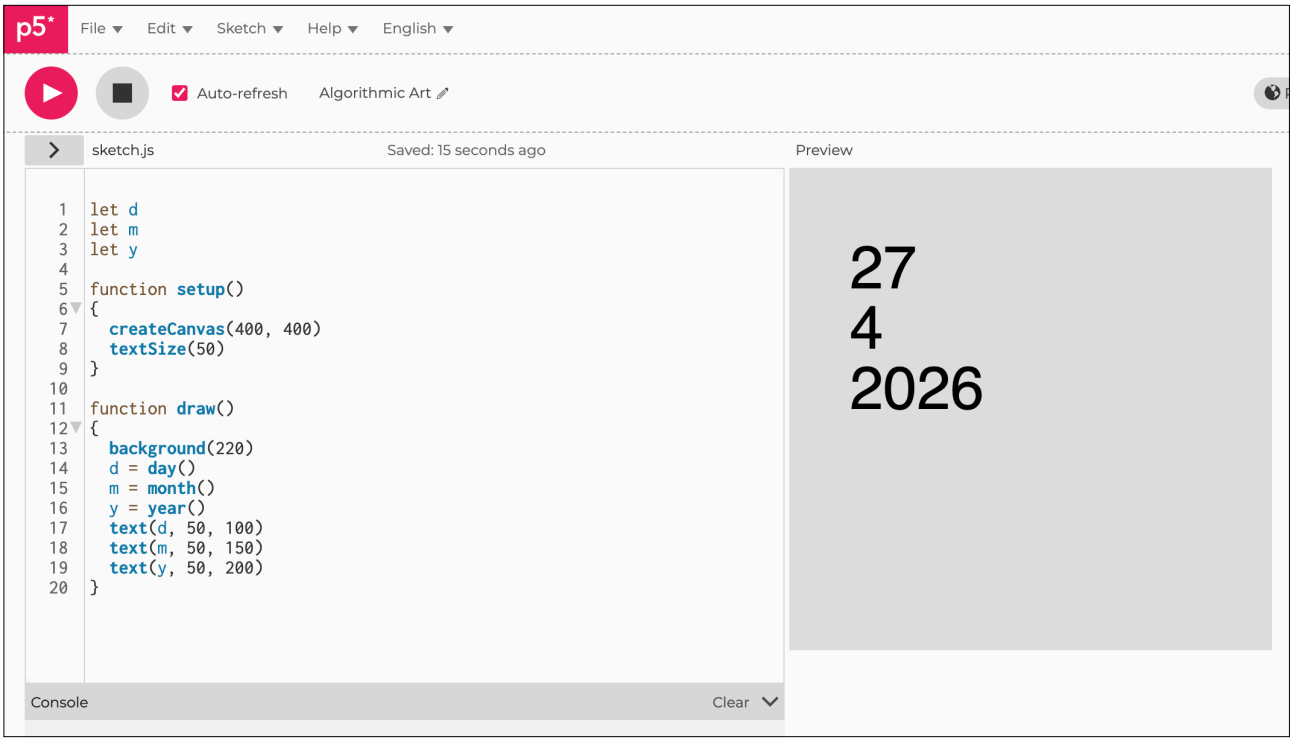
! Start a newish sketch.
Adding the date.

```
let d
let m
let y

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

function draw()
{
  background(220)
  d = day()
  m = month()
  y = year()
  text(d, 50, 100)
  text(m, 50, 150)
  text(y, 50, 200)
}
```

Notes

Gives the day in numbers. The month in numbers, and also the year.

Figure F7.3



Date functions

`day()`	gives you the current day (as a number).
`month()`	gives you the current month (as a number).
`year()`	gives you the current year.

Notes

These pull the data from the internet.

Challenge

Using the date and time, how creative could you be in showing all this information graphically?

Sketch F7.4 milliseconds of time

! Starting a new sketch.
Milliseconds of time represented differently, as a float, an integer, and as seconds.

```
function setup()
{
  createCanvas(400, 400)
  textSize(30)
}

function draw()
{
  background(220)
  text(millis(), 50, 50)
  text(int(millis()), 50, 100)
  text(int(millis()/1000), 50, 150)
}
```

Notes

The digits will appear as a blur. The millis() function returns the number of milliseconds that have elapsed since the programme started running as a `float`. It uses `int` to stop the numbers from having too many decimal points. The last one counts the number of seconds.

Challenge

Create a stopwatch effect where the draw() loop (using `noLoop()`) stops when you click the mouse.

Figure F7.4

Sketch F7.5 in the blink of an eye

! Start a new sketch.
This is a rather long-winded way of a blinking (every second) sketch, but it demonstrates how it can be done. You could do it for any length of time.

```
let a = 0
let b = 0
let c = 0
let value = 255

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

function draw()
{
  b = millis()
  c = b - a
  fill(value)
  circle(width/2, height/2, 100)
  if (c <= 1000)
  {
    value = 255
  }
  if (c >= 1000)
  {
    value = 0
  }
  if (c >= 2000)
  {
    a = b
  }
}
```

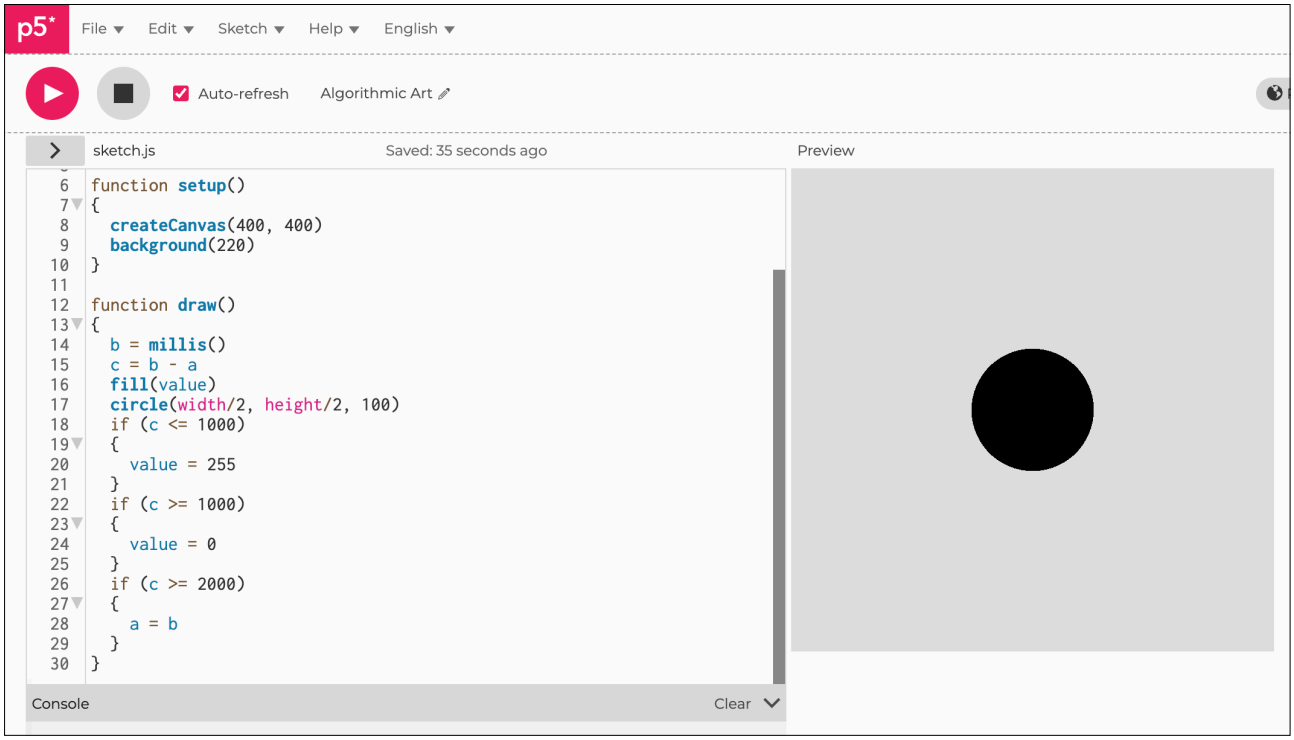
Notes

A simple blink programme. Uses a bit of simple maths to count 1,000 milliseconds (1 second).

Challenges

1. Make it blink faster or slower.
2. Create a slider to change the rate of blinking.

Figure F7.5



Sketch F7.6 alternative modulo

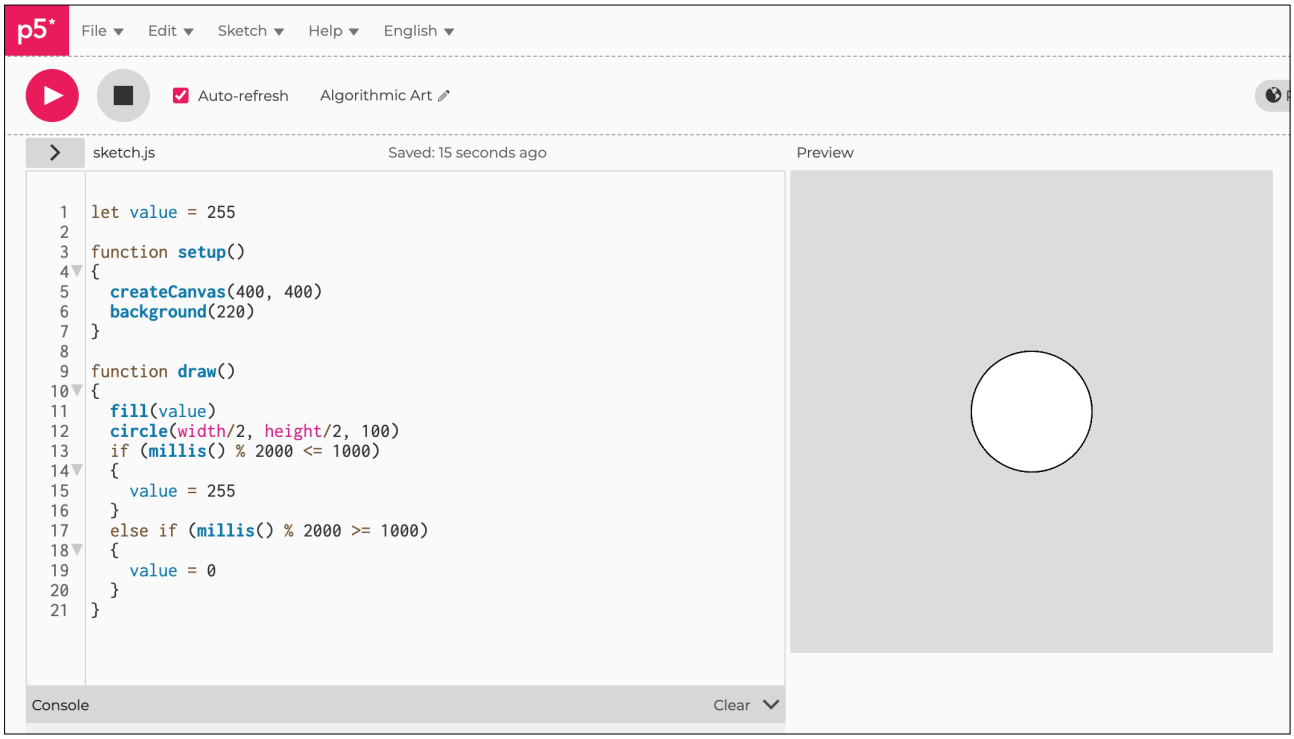
! Start a new sketch.
This does the same thing where modulo returns the remainder.

```
let value = 255

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

function draw()
{
  fill(value)
  circle(width/2, height/2, 100)
  if (millis() % 2000 <= 1000)
  {
    value = 255
  }
  else if (millis() % 2000 >= 1000)
  {
    value = 0
  }
}
```

Notes

The same result: the circle blinks black and white every second but a much simpler code.

Figure F7.6



Sketch F7.7 making a clock

We will do this in stages so you get an idea of how the bits go together. First off, let's draw the clock.

```
let radius = 200
let clockDiameter

function setup()
{
  createCanvas(400, 400)
  clockDiameter = radius * 1.7
  stroke('teal')
  strokeWeight(2)
}

function draw()
{
  background('orange')
  translate(width/2, height/2)
  fill('lightyellow')
  circle(0, 0, clockDiameter)
}
```

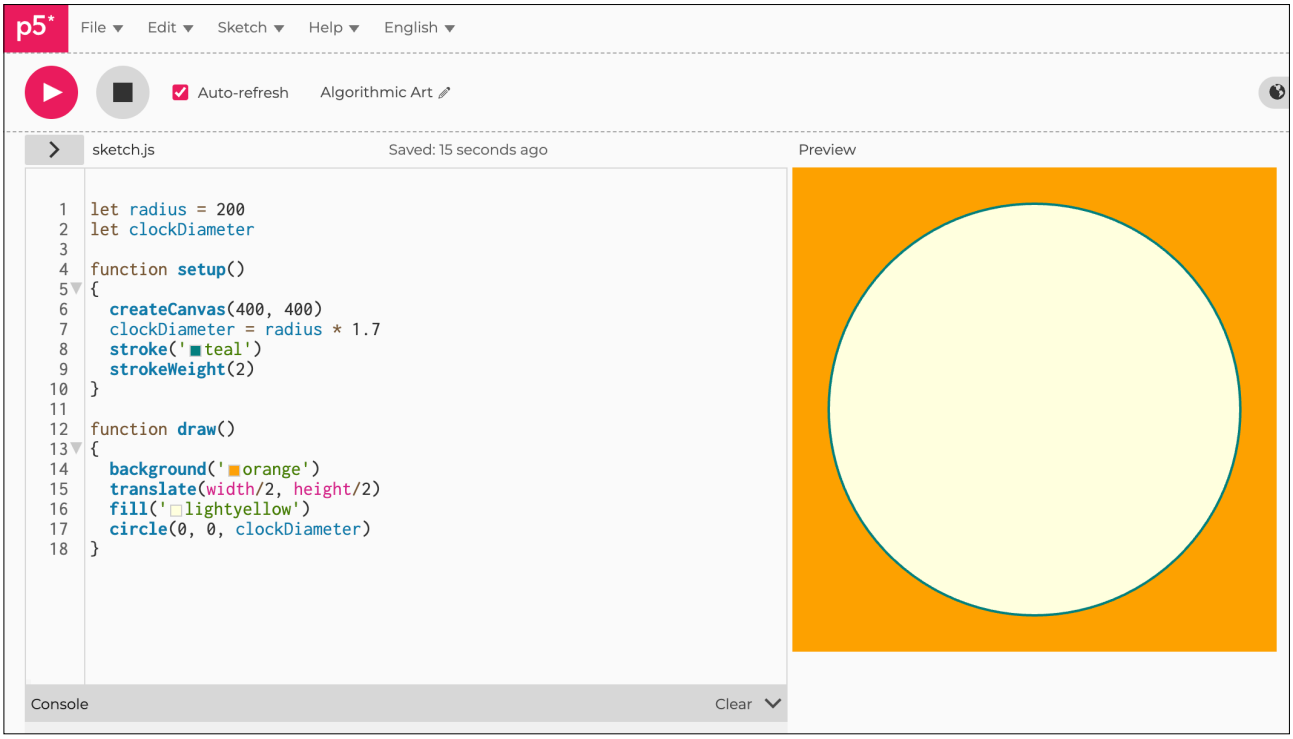
Notes

Simple circle on a simple background. The radius is our reference point for all other diameters/radii.

Challenge

Feel free to use other colours.

Figure F7.7



Sketch F7.8 other radii

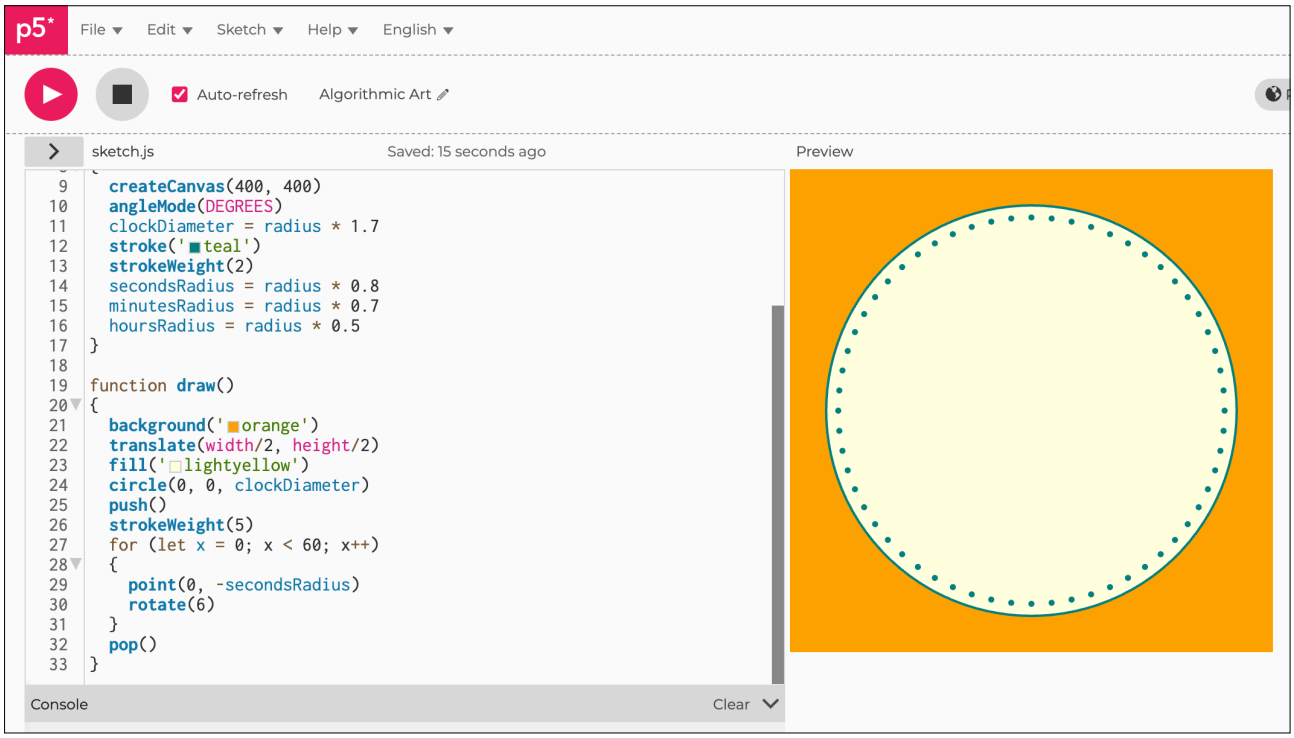
Set the radii for the seconds, minutes, and hours. We draw the ticks for the seconds; we are going to work in degrees.

```
let radius = 200
let clockDiameter
let secondsRadius
let minutesRadius
let hoursRadius

function setup()
{
  createCanvas(400, 400)
  angleMode(DEGREES)
  clockDiameter = radius * 1.7
  stroke('teal')
  strokeWeight(2)
  secondsRadius = radius * 0.8
  minutesRadius = radius * 0.7
  hoursRadius = radius * 0.5
}

function draw()
{
  background('orange')
  translate(width/2, height/2)
  fill('lightyellow')
  circle(0, 0, clockDiameter)
  push()
  strokeWeight(5)
  for (let x = 0; x < 60; x++)
  {
    point(0, -secondsRadius)
    rotate(6)
  }
  pop()
}
```

Notes

We have our foundation; next, we need to draw the hands. We use `push()` and `pop()` so that we do not compound the rotation each time.

Figure F7.8



Sketch F7.9 the angles

Calculating the angle from the time.

```
let radius = 200
let clockDiameter
let secondsRadius
let minutesRadius
let hoursRadius
let secondsAngle
let minutesAngle
let hoursAngle

function setup()
{
  createCanvas(400, 400)
  angleMode(DEGREES)
  clockDiameter = radius * 1.7
  stroke('teal')
  strokeWeight(2)
  secondsRadius = radius * 0.8
  minutesRadius = radius * 0.7
  hoursRadius = radius * 0.5
}

function draw()
{
  background('orange')
  translate(width/2, height/2)
  fill('lightyellow')
  circle(0, 0, clockDiameter)
  push()
  strokeWeight(5)
  for (let x = 0; x < 60; x++)
  {
    point(0, -secondsRadius)
    rotate(6)
  }
```

```
  pop()
  secondsAngle = map(second(), 0, 60, 0, 360)
  minutesAngle = map(minute(), 0, 60, 0, 360)
  hoursAngle = map(hour(), 0, 12, 0, 360)
}
```

Notes

Nothing new to see here; we are just calculating the angles by mapping the number of seconds, minutes, and hours to 360.

Sketch F7.10 drawing the hands

Now we will draw the seconds hand, the minutes hand, and the hours hand.

```
let radius = 200
let clockDiameter
let secondsRadius
let minutesRadius
let hoursRadius
let secondsAngle
let minutesAngle
let hoursAngle

function setup()
{
  createCanvas(400, 400)
  angleMode(DEGREES)
  clockDiameter = radius * 1.7
  stroke('teal')
  strokeWeight(2)
  secondsRadius = radius * 0.8
  minutesRadius = radius * 0.7
  hoursRadius = radius * 0.5
}

function draw()
{
  background('orange')
  translate(width/2, height/2)
  fill('lightyellow')
  circle(0, 0, clockDiameter)
  push()
  strokeWeight(5)
  for (let x = 0; x < 60; x++)
  {
    point(0, -secondsRadius)
    rotate(6)
  }
```

```
  pop()
  secondsAngle = map(second(), 0, 60, 0, 360)
  minutesAngle = map(minute(), 0, 60, 0, 360)
  hoursAngle = map(hour(), 0, 12, 0, 360)

  push()
  rotate(secondsAngle)
  stroke('darkred')
  line(0, 20, 0, -secondsRadius)
  pop()

  push()
  strokeWeight(3)
  rotate(minutesAngle)
  stroke('blue')
  line(0, 0, 0, -minutesRadius)
  pop()

  push()
  strokeWeight(5)
  rotate(hoursAngle)
  stroke('grey')
  line(0, 0, 0, -hoursRadius)
  pop()

  stroke('darkred')
  circle(0, 0, 5)
}
```

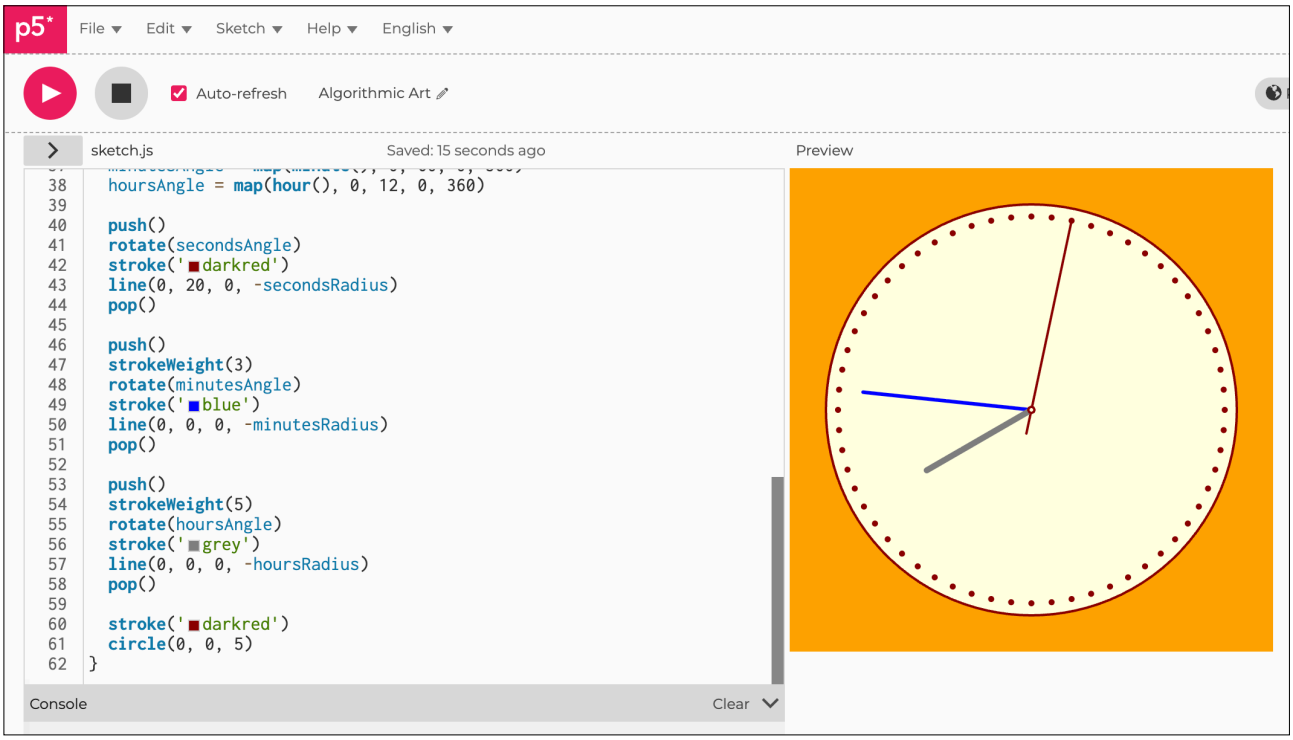
Notes

We have our finished clock.

Challenge

Can you think of some creative ways to illustrate the time, for instance, using lines or 3D shapes?

Figure F7.10




Algorithmic
Art
Module F
Unit #8
Local
Storage

Module F Unit #8: local storage

You can store information and data on your machine and in your browser, calling it up at a later date. This may or may not be that useful for algorithmic art, but it does have potential other uses and applications. Either way, it is worth being aware of if you suddenly have a creative or interactive idea where you might need this feature.

Sketch F8.1 sliders

By sliding the sliders, we can change the background.

```
let rSlider
let gSlider
let bSlider

function setup()
{
  createCanvas(400, 400)
  rSlider = createSlider(0, 255, 0)
  gSlider = createSlider(0, 255, 0)
  bSlider = createSlider(0, 255, 0)
}

function draw()
{
  let r = rSlider.value()
  let g = gSlider.value()
  let b = bSlider.value()
  background(r, g, b)
}
```

Notes

This is jumping straight in, but we have covered sliders already. Three sliders at the bottom for red, green, and blue. When you create your colour and refresh the canvas, you go straight back to the default settings.

Figure F8.1

Sketch F8.2 storing data

We cannot save these settings when we close down the programme and then reopen it later on the same machine. Yet we can if we use two `p5.js` functions. They do exactly as they say on the tin.

We create our own function called `storedData()`. Inside that function, we store the value of the red slider. We will call the variable `redValue` for the red slider colour setting for now; we will do all the slider colours later.

```
let rSlider
let gSlider
let bSlider

function setup()
{
  createCanvas(400, 400)
  rSlider = createSlider(0, 255, 0)
  gSlider = createSlider(0, 255, 0)
  bSlider = createSlider(0, 255, 0)
  rSlider.changed(storeData)
}

function storeData()
{
  storeItem('redValue', rSlider.value())
}

function draw()
{
  let r = rSlider.value()
  let g = gSlider.value()
  let b = bSlider.value()
  background(r, g, b)
}
```

Notes

This stores the data for the red slider, but when you refresh the page, it simply reverts back to the default settings. We need to get the stored value. The key functions are:

`storeItem()`	which, as it suggests, stores the data we want stored
`getItem()`	this retrieves the said data

Sketch F8.3 getting the data

The `redValue` has been saved. Now we add the `getItem()` function. We only want to get the value if a value exists, hence the conditional `!==` if statement.

```
let rSlider
let gSlider
let bSlider

function setup()
{
  createCanvas(400, 400)
  rSlider = createSlider(0, 255, 0)
  let r = getItem('redValue')
  if (r !== null)
  {
    rSlider.value(r)
  }
  gSlider = createSlider(0, 255, 0)
  bSlider = createSlider(0, 255, 0)
  rSlider.changed(storeData)
}

function storeData()
{
  storeItem('redValue', rSlider.value())
}

function draw()
{
  let r = rSlider.value()
  let g = gSlider.value()
  let b = bSlider.value()
  background(r, g, b)
}
```

Notes

Save the sketch, now change the red slider (make sure that auto-refresh is off), and close the sketch; you can even close the browser. Open it up again and you should have the same red value for the slider and background.

Challenge

You can check by changing the other sliders and you will see the difference!

Sketch F8.4 replacing all the values

Putting the other values for the background colour into the local storage.

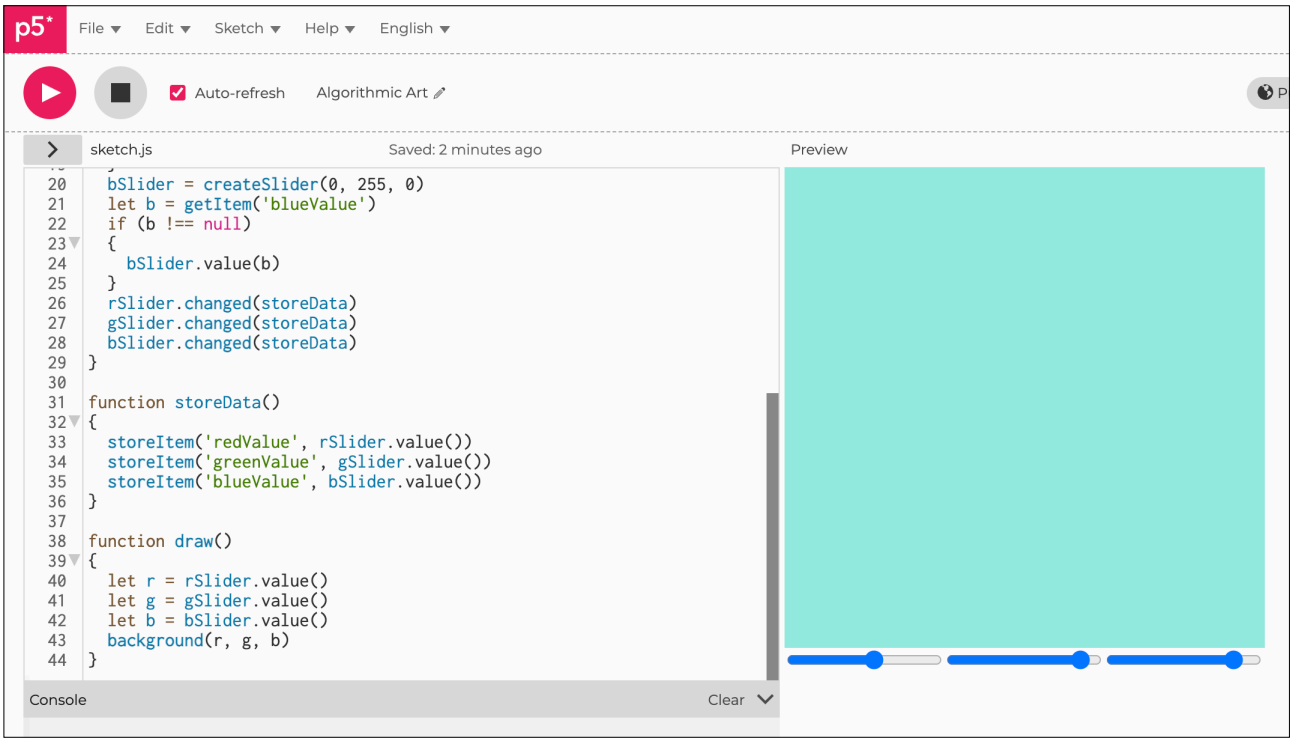
```
let rSlider
let gSlider
let bSlider

function setup()
{
  createCanvas(400, 400)
  // get the red value
  rSlider = createSlider(0, 255, 0)
  let r = getItem('redValue')
  if (r !== null)
  {
    rSlider.value(r)
  }
  // get the green value
  gSlider = createSlider(0, 255, 0)
  let g = getItem('greenValue')
  if (g !== null)
  {
    gSlider.value(g)
  }
  // get the blue value
  bSlider = createSlider(0, 255, 0)
  let b = getItem('blueValue')
  if (b !== null)
  {
    bSlider.value(b)
  }
  // store the values of the red, green and blue
  rSlider.changed(storeData)
  gSlider.changed(storeData)
  bSlider.changed(storeData)
}
```

```
function storeData()
{
  storeItem('redValue', rSlider.value())
  storeItem('greenValue', gSlider.value())
  storeItem('blueValue', bSlider.value())
}

function draw()
{
  let r = rSlider.value()
  let g = gSlider.value()
  let b = bSlider.value()
  background(r, g, b)
}
```

Notes

I've put in the // to help break up the code into the three separate colour sliders. Now when you refresh it will save all three slider positions.

Figure F8.4




Algorithmic
Art
Module F
Unit #9
Using
json Files

Module F Unit #9: using json files

This is primarily about data files in the .json format. This covers how they are created and can be used. JavaScript makes good use of this data file type, as do other coding languages.

Data is formatted in many different forms; two very common ones are CSV or JSON, which are easily accessed by computers. From those file types, you can access the data in any form or order you like.

Sketch F9.1 a bubble object

This is just a simple sketch with one bubble described as an object and drawing it filled with yellow.

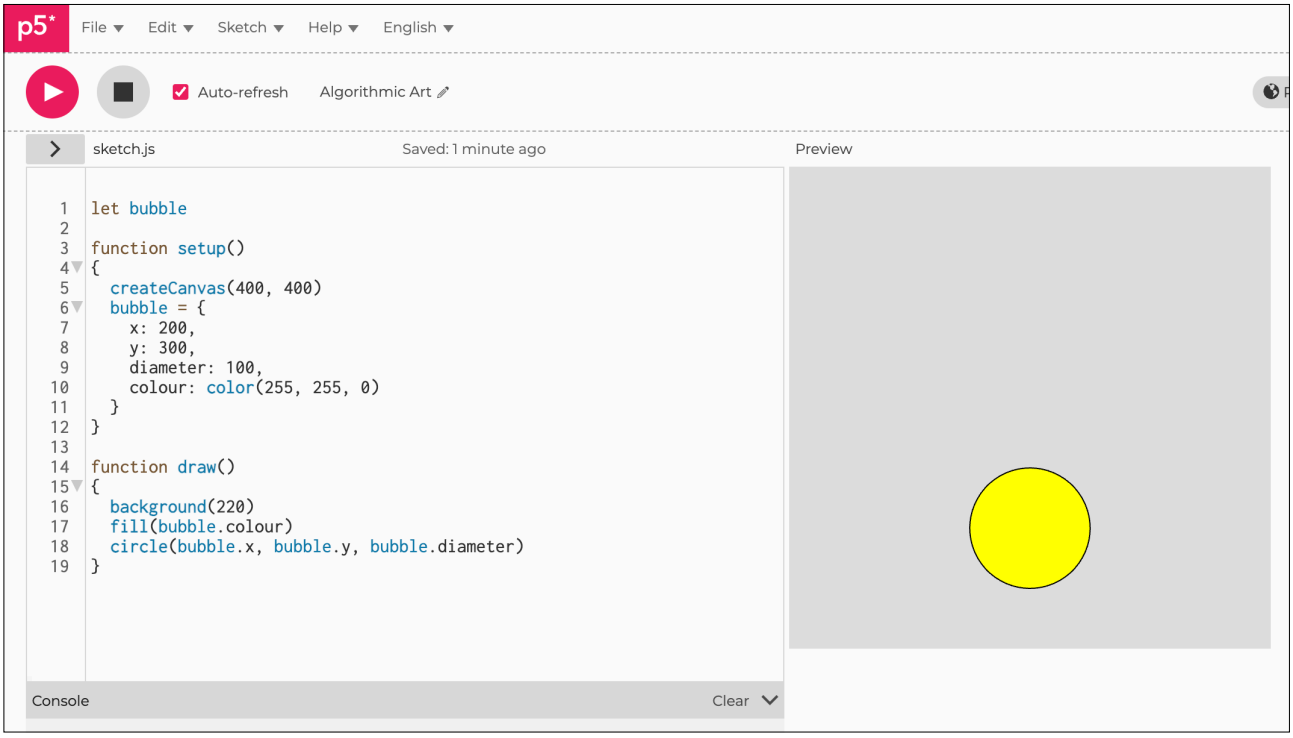
sketch.js

```
let bubble

function setup()
{
  createCanvas(400, 400)
  bubble = {
    x: 200,
    y: 300,
    diameter: 100,
    colour: color(255, 255, 0)
  }
}

function draw()
{
  background(220)
  fill(bubble.colour)
  circle(bubble.x, bubble.y, bubble.diameter)
}
```

Notes

Creating a simple dataset of one yellow bubble with `x`, `y` co-ordinates and a `diameter`. The bubble is an object with those elements. This works fine for one bubble, but what if you want `100` different circles? It can be improved by putting the data in a separate .json file, which is a JavaScript data file.

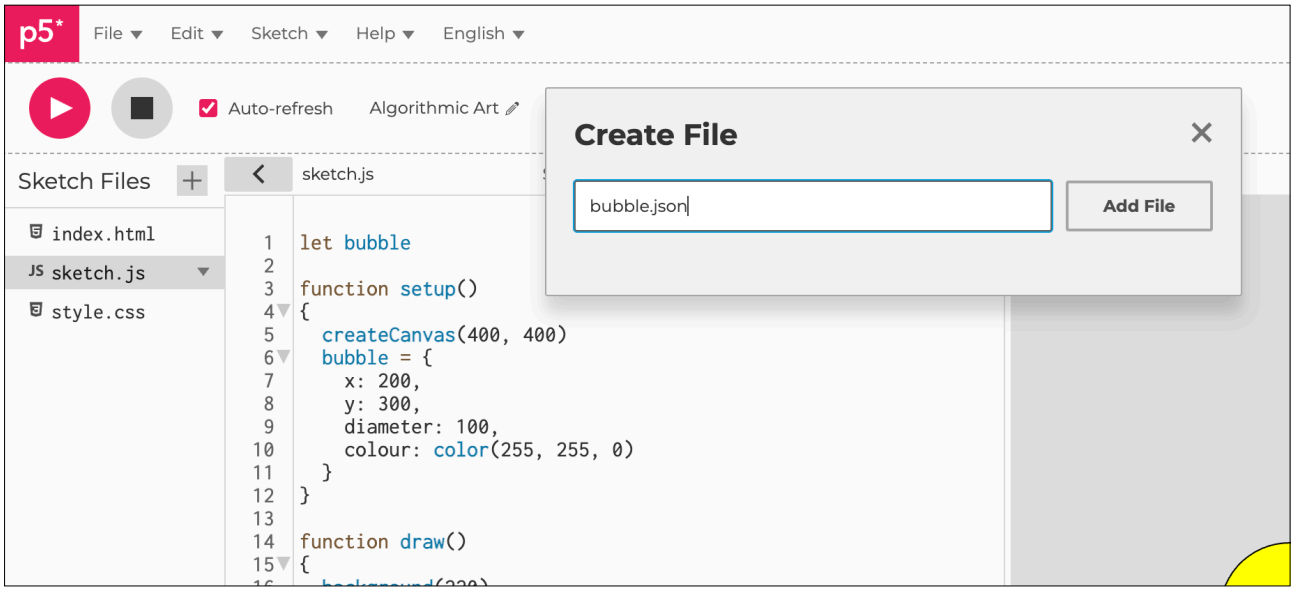
Figure F9.1



Creating a json file

Create a new file and call it `bubble.json`, add it to the list of files and add it there as we have done before.

Figure 1: creating the bubble.json file



Sketch F9.2 the json data file

! The .json file.

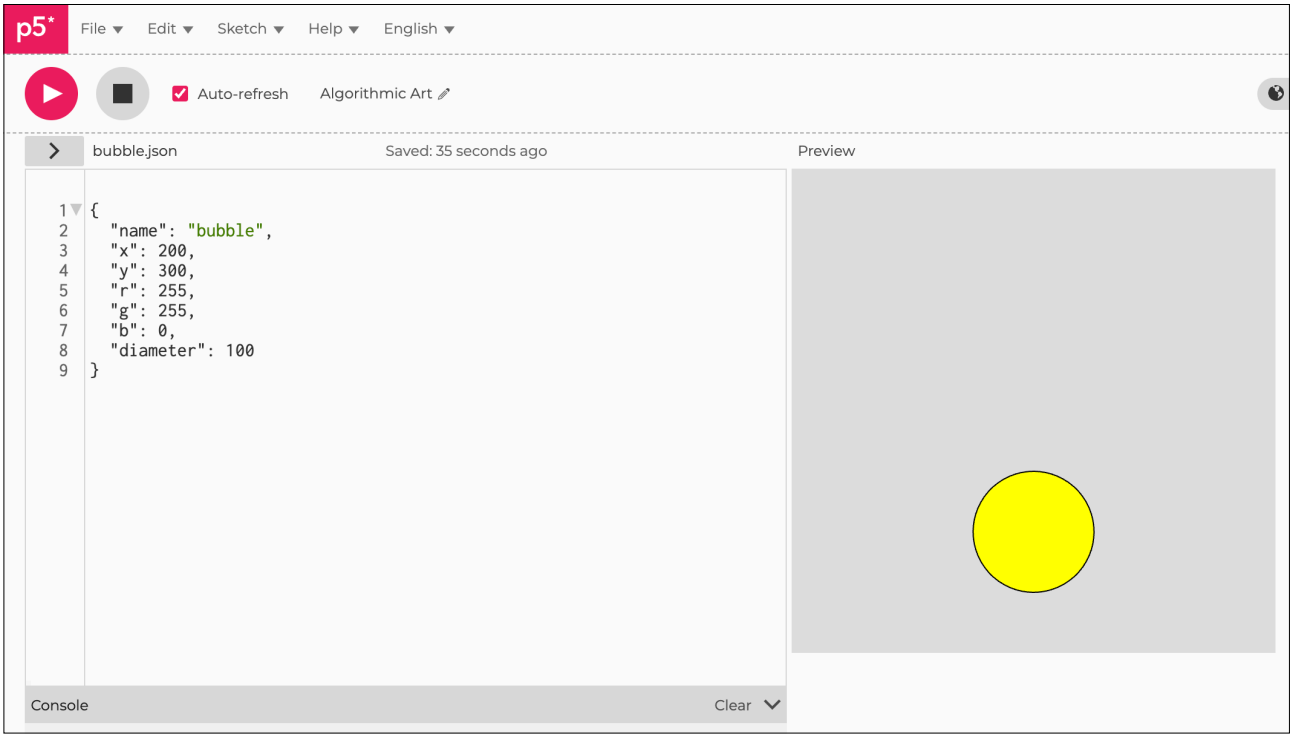
bubble.json

```
{
  "name": "bubble",
  "x": 200,
  "y": 300,
  "r": 255,
  "g": 255,
  "b": 0,
  "diameter": 100
}
```

Notes

You can check if the format is correct because it does need to be. The following website will do that by you copying and pasting it into the website, and it will return the result:
https://jsonformatter.curiousconcept.com/#
is a way of checking if the JSON is OK.

Figure F9.2



Sketch F9.3 adapting the sketch

! Back to sketch.js.
Now we adapt the sketch accordingly. Notice we don't need the bubble data in the sketch; we just need to upload it.

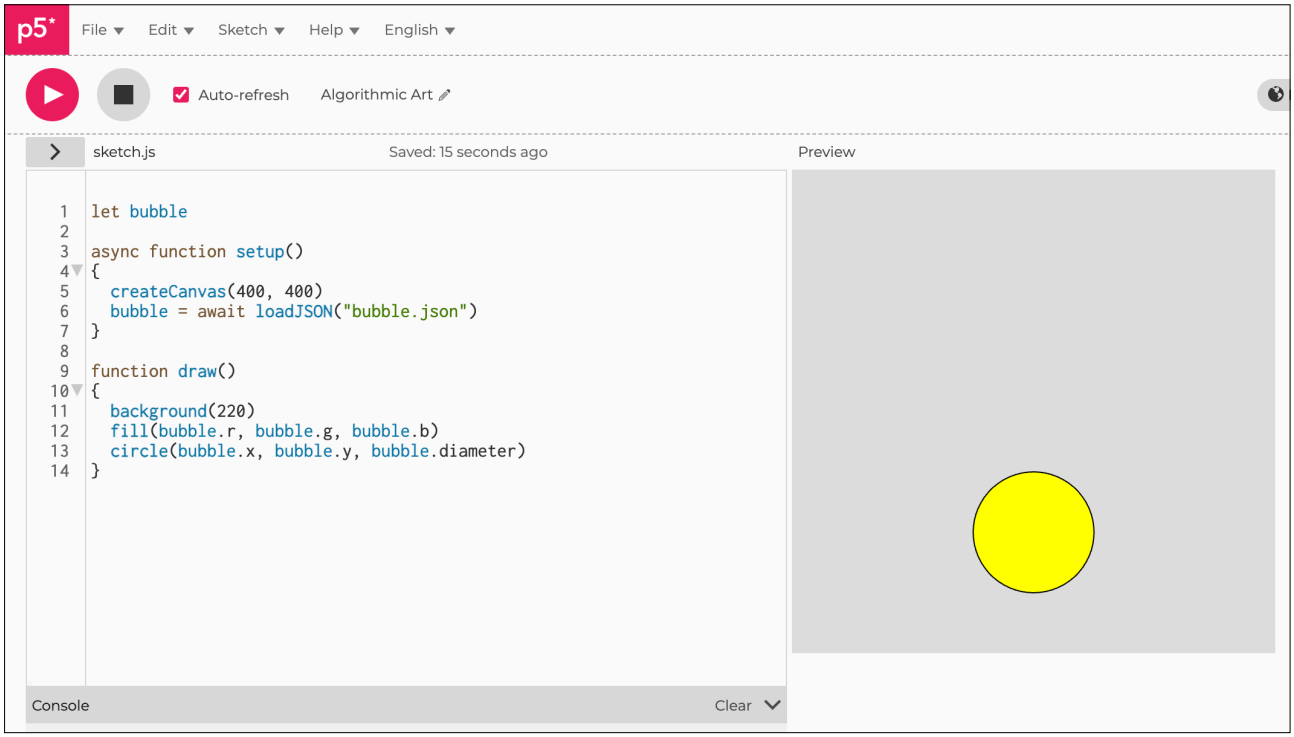
sketch.js

```
let bubble


async function setup()
{
  createCanvas(400, 400)
  bubble = await loadJSON("bubble.json")
}


function draw()
{
  background(220)
  fill(bubble.r, bubble.g, bubble.b)
  circle(bubble.x, bubble.y, bubble.diameter)
}
```

Notes

We load the data and extract it accordingly.

Figure F9.3



Sketch F9.4 more than one bubble

! Revisiting the bubble.json file.
This is the bubble.json file with two bubbles, a red bubble and a green bubble. The bubble is now an array called bubble[]. The bubble[0] is the red bubble and bubble[1] is the green bubble.

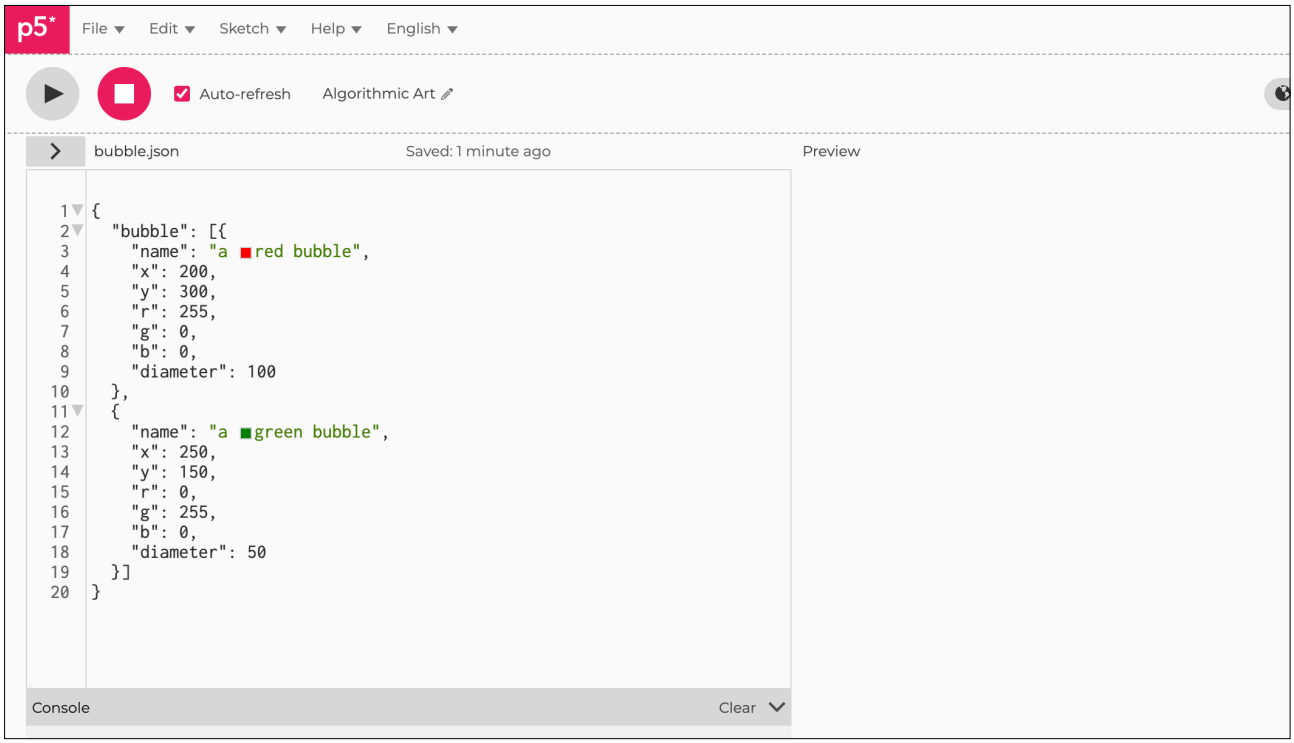
bubble.json

```
{
  "bubble": [{
    "name": "a red bubble",
    "x": 200,
    "y": 300,
    "r": 255,
    "g": 0,
    "b": 0,
    "diameter": 100
  },
  {
    "name": "a green bubble",
    "x": 250,
    "y": 150,
    "r": 0,
    "g": 255,
    "b": 0,
    "diameter": 50
  }]
}
```

Notes

This array has two bubble elements.

Figure F9.4



Sketch F9.5 a long winded way

! In the sketch.js.
We can now draw these two bubbles. We create a variable called data, and the JSON file is loaded into that data variable.

sketch.js

```
let data

async function setup()
{
  createCanvas(400, 400)
  data = await loadJSON("bubble.json")
}

function draw()
{
  background(220)
  fill(data.bubble[0].r, data.bubble[0].g, data.bubble[0].b)
  circle(data.bubble[0].x, data.bubble[0].y, data.bubble[0].diameter)
  fill(data.bubble[1].r, data.bubble[1].g, data.bubble[1].b)
  circle(data.bubble[1].x, data.bubble[1].y, data.bubble[1].diameter)
}
```

Notes

I am sure you can think of a way to do this more efficiently with a for loop.

Challenges

1. Add more bubbles
2. Add different shapes
3. There is some useful JSON data on GitHub which you can copy and create your own JSON file and try to pull data from it. The link is: https://github.com/dariusk/corpora/tree/master/data.
4. Have a go at using this dataset and see what you can do.

Sketch F9.6 a more concise way

That is so much better, using a simple `for()` loop.

sketch.js

```
let data

async function setup()
{
  createCanvas(400, 400)
  data = await loadJSON("bubble.json")
}

function draw()
{
  background(220)
  for (let i = 0; i < data.bubble.length; i++)
  {
    fill(data.bubble[i].r, data.bubble[i].g, data.bubble[i].b)
    circle(data.bubble[i].x, data.bubble[i].y, data.bubble[i].diameter)
  }
}
```

Notes

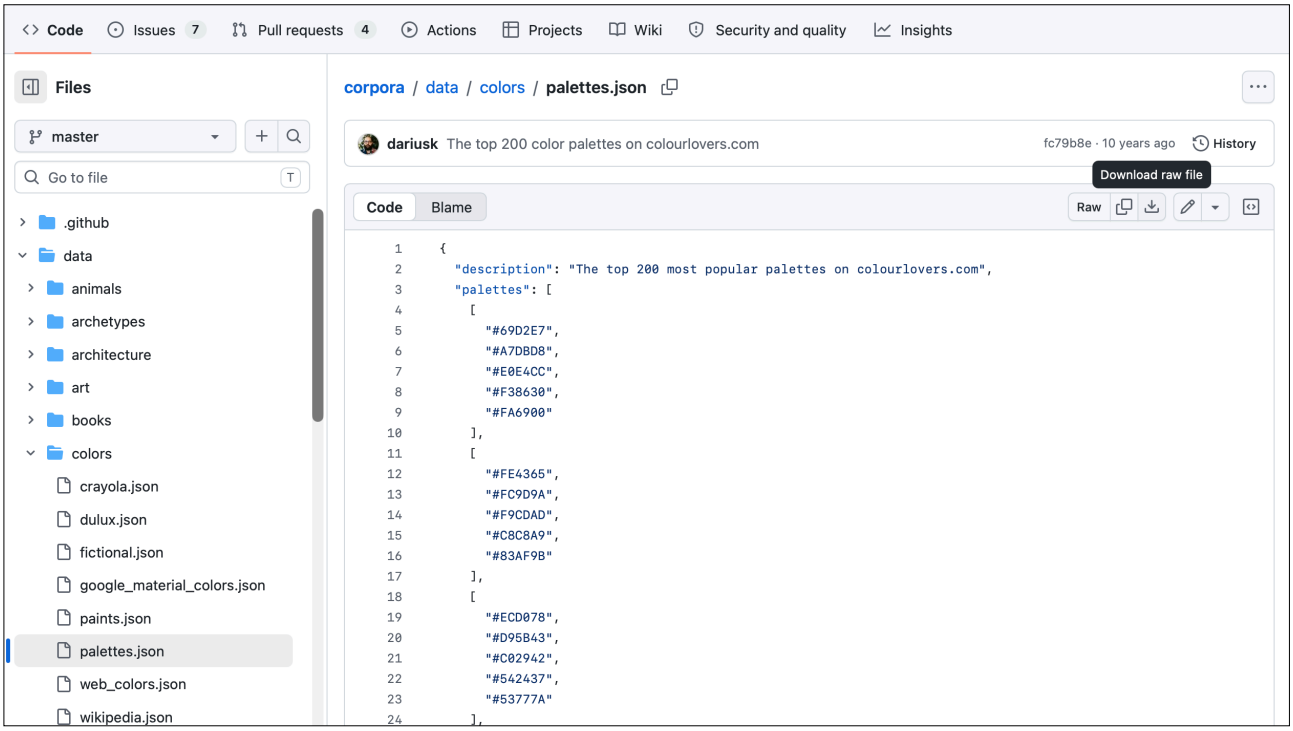
Always try to find a way of using the fewest lines of code.

Using a pre-made .json file

We are going to use a pre-made JSON file that has a collection of some of the favourite colour palettes in hex values. We can access this file and use the five colours in each set of palettes; there are 200 palettes altogether.

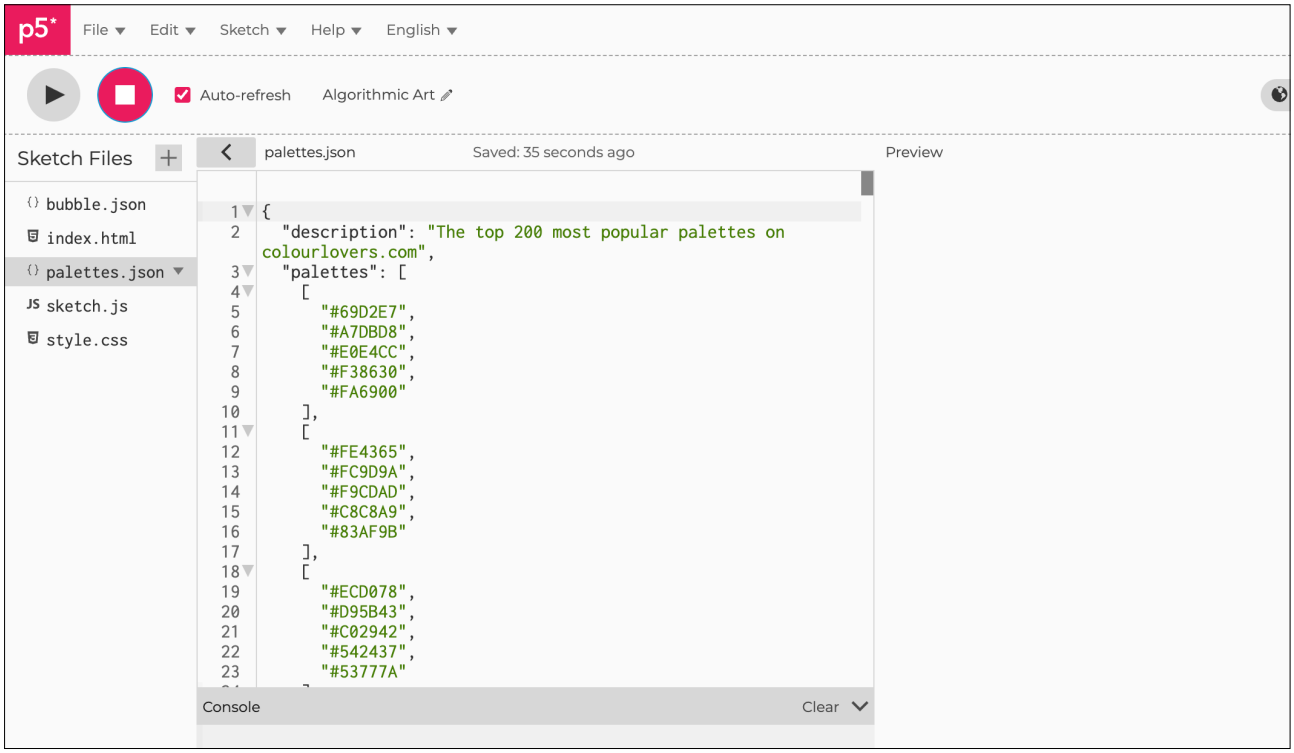
To get the JSON file, follow the link here or use the link on the website to download the file:
https://github.com/dariusk/corpora/blob/master/data/colors/palettes.json
You may notice there are other collections that might be of interest.

Figure 2: download the file



Do the usual upload:

Figure 3: the uploaded file



Sketch F9.7 console log

! Start a new sketch.
The great thing about the console log is that it can help you understand the structure of a file such as this one. We are going to upload the file and check what is inside.

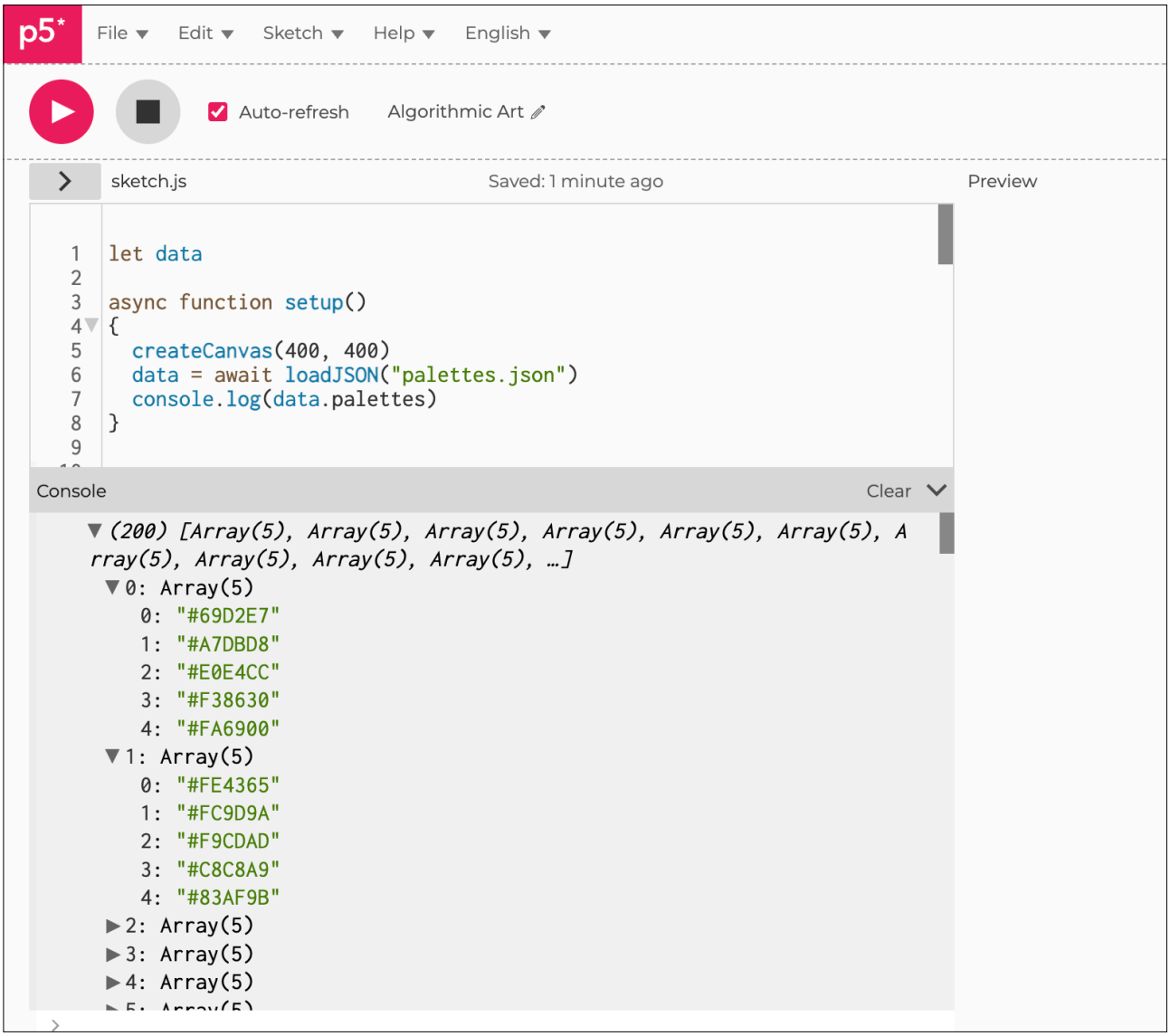
sketch.js

```
let data


async function setup()
{
  createCanvas(400, 400)
  data = await loadJSON("palettes.json")
  console.log(data.palettes)
}
```

Notes

This gives us a comprehensive list of all the palettes and their colours, as you could see in the GitHub repository.

Figure F9.7



Sketch F9.8 palette number 10

If we want to look inside one palette, we need to specify which one. This is palette number 10 (counting starts at 0).

sketch.js

```
let data


async function setup()
{
  createCanvas(400, 400)
  data = await loadJSON("palettes.json")
  console.log(data.palettes[10])
}
```

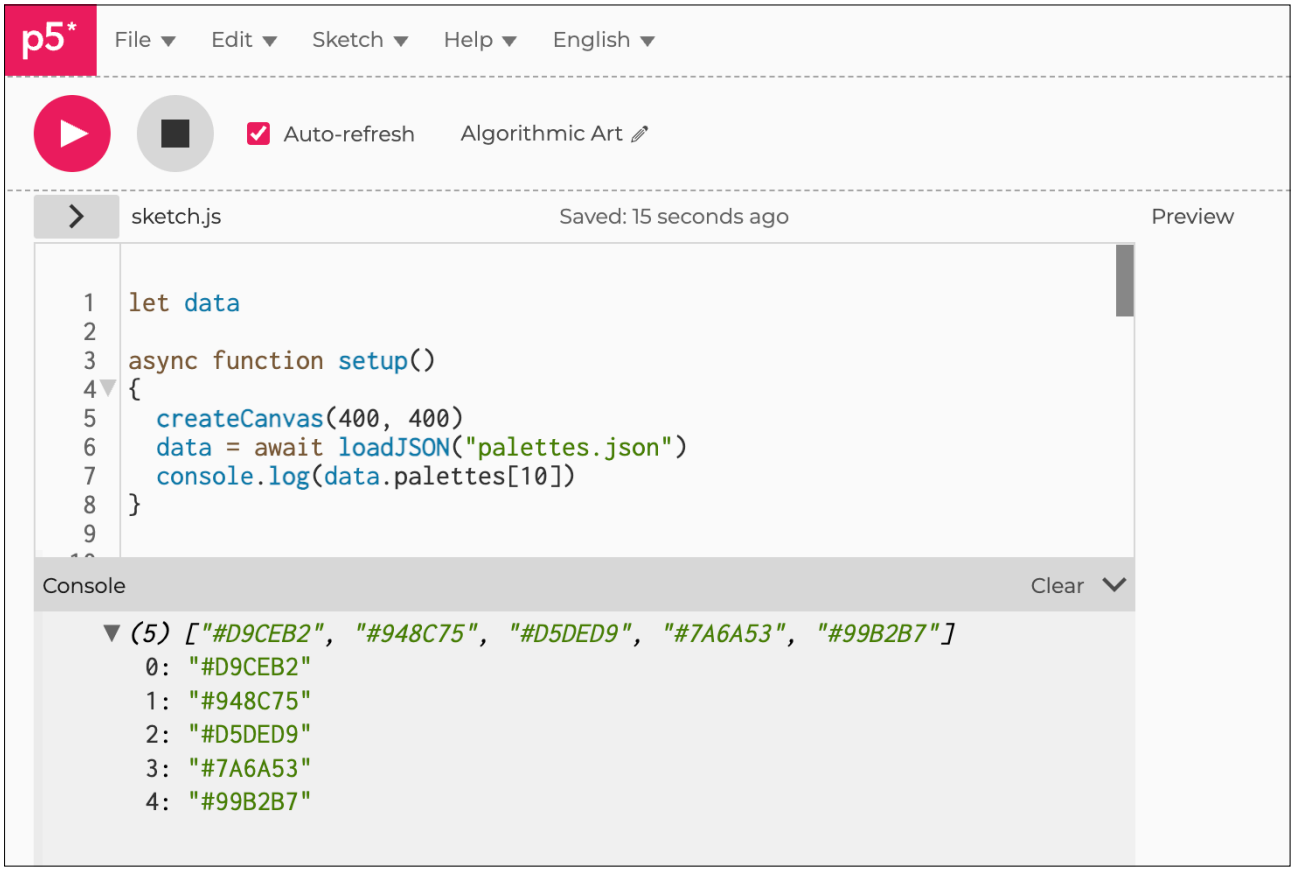
Notes

Randomly picked a palette.

Challenge

Select your own palette.

Figure F9.8



Sketch F9.9 specific colour

To select just one colour from that palette of five colours, we specify it thus. In this case, it is colour number 2 (the 3rd colour) in palette number 10 (the 11th palette).

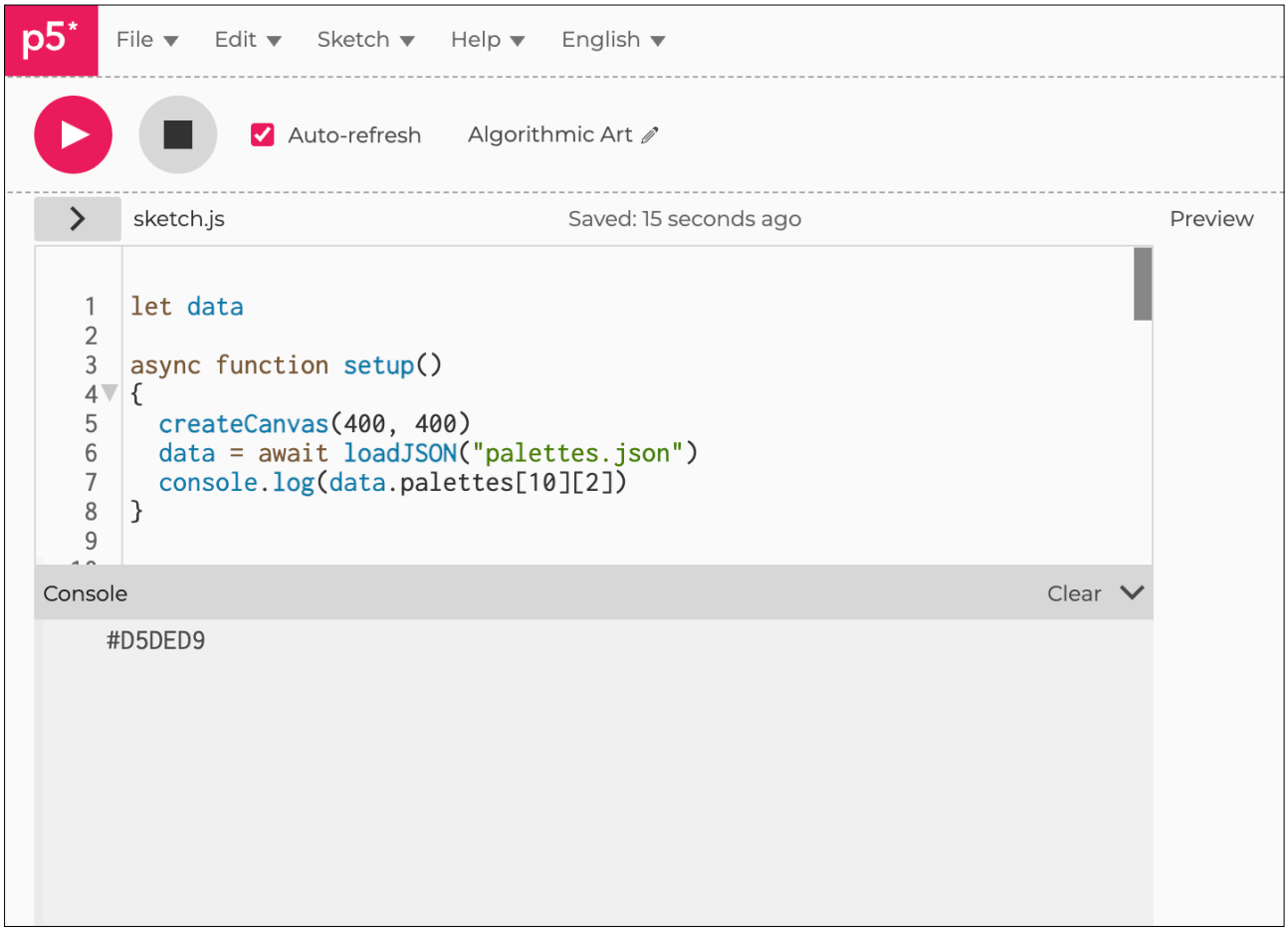
sketch.js

```
let data


async function setup()
{
  createCanvas(400, 400)
  data = await loadJSON("palettes.json")
  console.log(data.palettes[10][2])
}
```

Notes

Unless you speak hex, it will be a bit meaningless.

Figure F9.9



Sketch F9.10 fill the circle

Let's have a look at it.
! Remove console log.

sketch.js

```
let data


async function setup()
{
  createCanvas(400, 400)
  data = await loadJSON("palettes.json")
}


function draw()
{
  background('white')
  fill(data.palettes[10][2])
  circle(100, 100, 100)
}
```

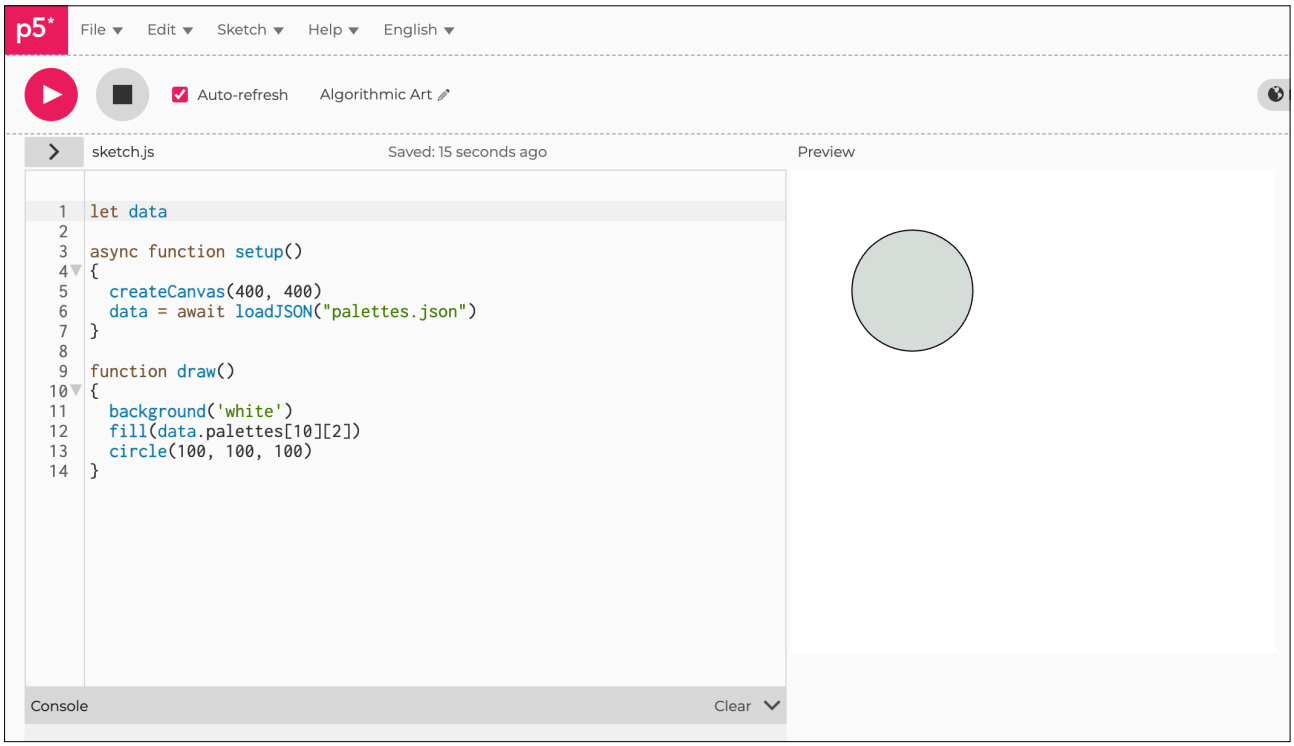
Notes

We get a rather dull colour, but it is a palette colour, and I did choose it randomly.

Challenges

1. Select another one and see what you get.
2. What would be nice is to scroll through all the palettes one by one. Can you think of a way of doing that?

Figure F9.10



Sketch F9.11 cycle through the palette

I won't highlight all the changes, as there are many. The main function is to iterate through each palette one at a time.

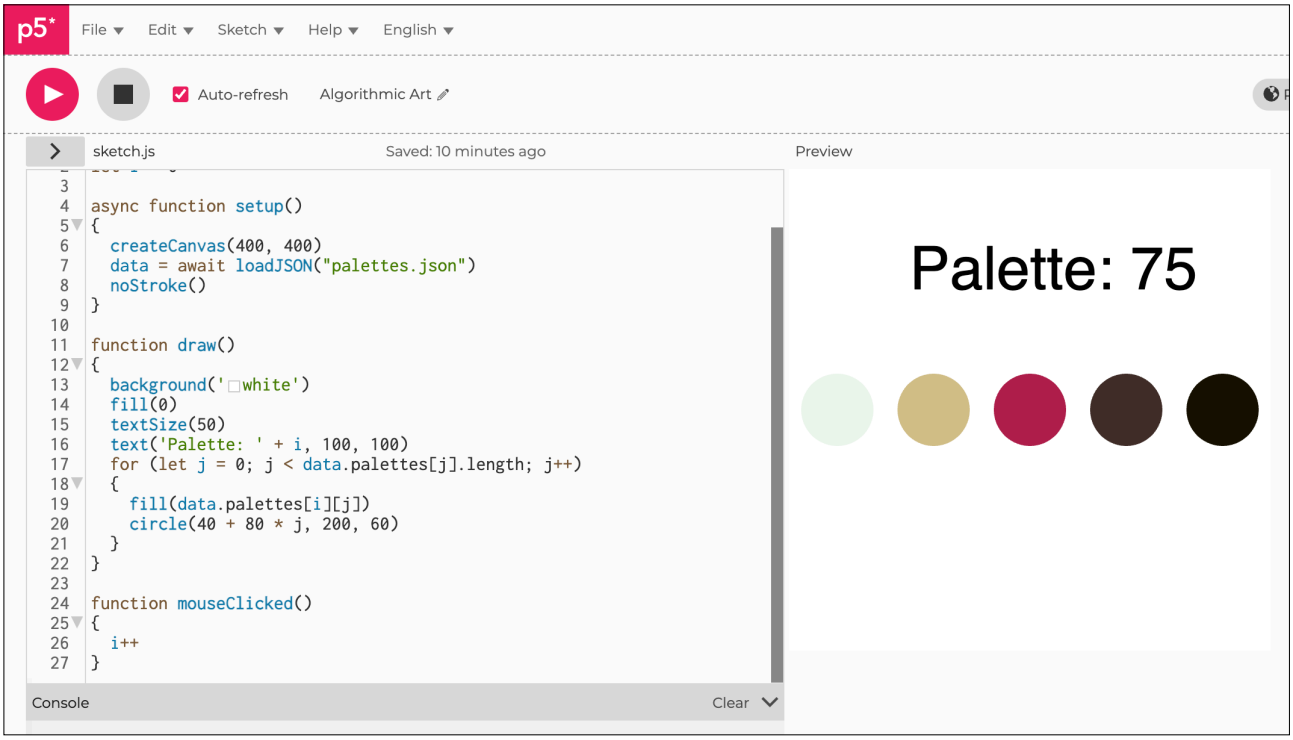
sketch.js

```
let data
let i = 0

async function setup()
{
  createCanvas(400, 400)
  data = await loadJSON("palettes.json")
  noStroke()
}

function draw()
{
  background('white')
  fill(0)
  textSize(50)
  text('Palette: ' + i, 100, 100)
  for (let j = 0; j < data.palettes[j].length; j++)
  {
    fill(data.palettes[i][j])
    circle(40 + 80 * j, 200, 60)
  }
}

function mouseClicked()
{
  i++
}
```

Notes

Every time you click the mouse, you get the next palette.

Figure F9.11



Sketch F9.12 saveJSON()

! Starting a new sketch.
Inserting the two bubble JSON data into the main sketch this time. We can now save this as our own JSON file. Click on the canvas and then find the saved file and open it with whatever software it offers to open it if you want to look inside.

sketch.js

```
let data

function setup()
{
  createCanvas(400, 400)
  background(220)
  textSize(20)
  text("click here", 50, 50)
}

function mousePressed()
{
  data = {
    bubble: [
      {
        name: "a red bubble",
        x: 200,
        y: 300,
        r: 255,
        g: 0,
        b: 0,
        diameter: 100,
      },
      {
        name: "a green bubble",
        x: 250,
        y: 150,
        r: 0,
        g: 255,
        b: 0,
        diameter: 50,
```

```
      },
    ],
  }
  saveJSON(data, "bubbles")
}
```

Notes

If you can look inside it, this is what you see.

Challenge

1. Try loading this or some other JSON file you have created.
2. Create your own palette, perhaps using RGB values instead.

Figure F9.12

bubbles.json

```
{
  "bubble": [
    {
      "name": "a red bubble",
      "x": 200,
      "y": 300,
      "r": 255,
      "g": 0,
      "b": 0,
      "diameter": 100
    },
    {
      "name": "a green bubble",
      "x": 250,
      "y": 150,
      "r": 0,
      "g": 255,
      "b": 0,
      "diameter": 50
    }
  ]
}
```