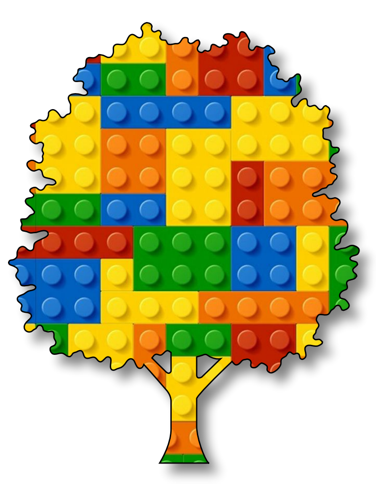
Algorithmic Art
Module F
Unit #3
video capture



Content

Module F Unit #3 using video

Sketch F3.1 scale
Sketch F3.2 scale with mouse
Sketch F3.3 scale negatively
Sketch F3.4 video capture
Sketch F3.5 creating the canvas
Sketch F3.6 video on the canvas
Sketch F3.7 size and position
Sketch F3.8 hide the streaming video
Sketch F3.9 taking a selfie
Sketch F3.10 multiple selfies
Sketch F3.11 getting pixels
Sketch F3.12 returning the video
Sketch F3.13 pixelating the image
Sketch F3.14 making it grey scale
Sketch F3.15 brightness mirror
Sketch F3.16 threshold image
Sketch F3.17 optional threshold
Sketch F3.18 starting sketch
Sketch F3.19 array of particles
Sketch F3.20 the Particle class
Sketch F3.21 show something
Sketch F3.22 getting the pixels
Sketch F3.23 the 300
Sketch F3.24 randomise the position
Creating the mirror effect
Sketch F3.25 mirror the image #1
Sketch F3.26 mirror the image #2

Introduction to using video

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.

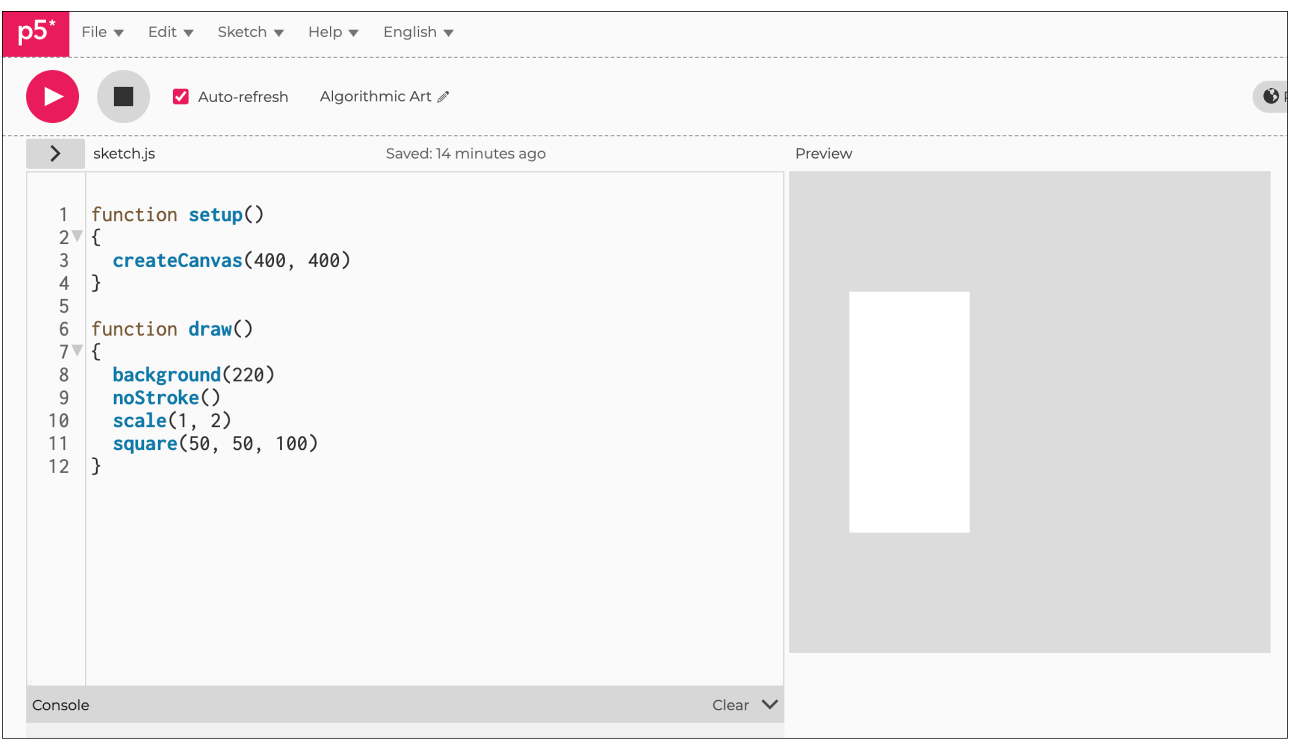
🌻 Challenges

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)
}
```

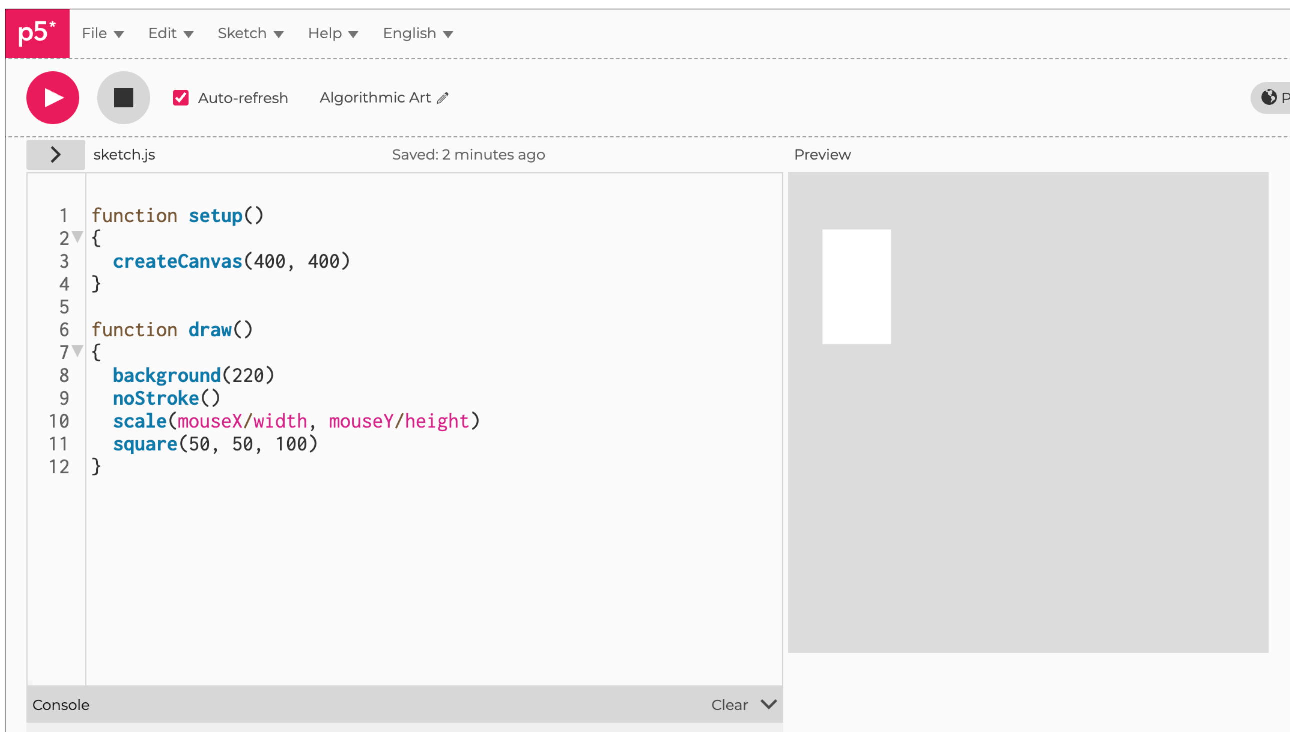
Notes

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)
}
```

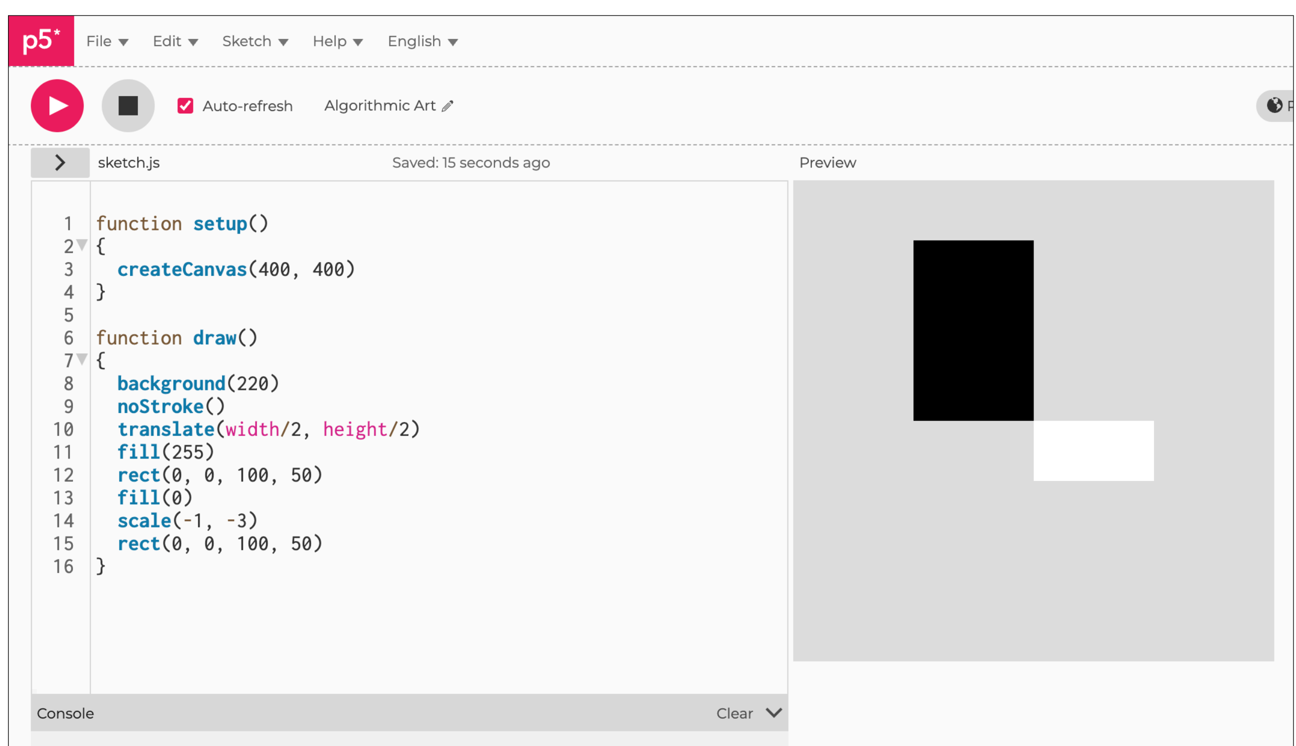
🗒️ Notes

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.

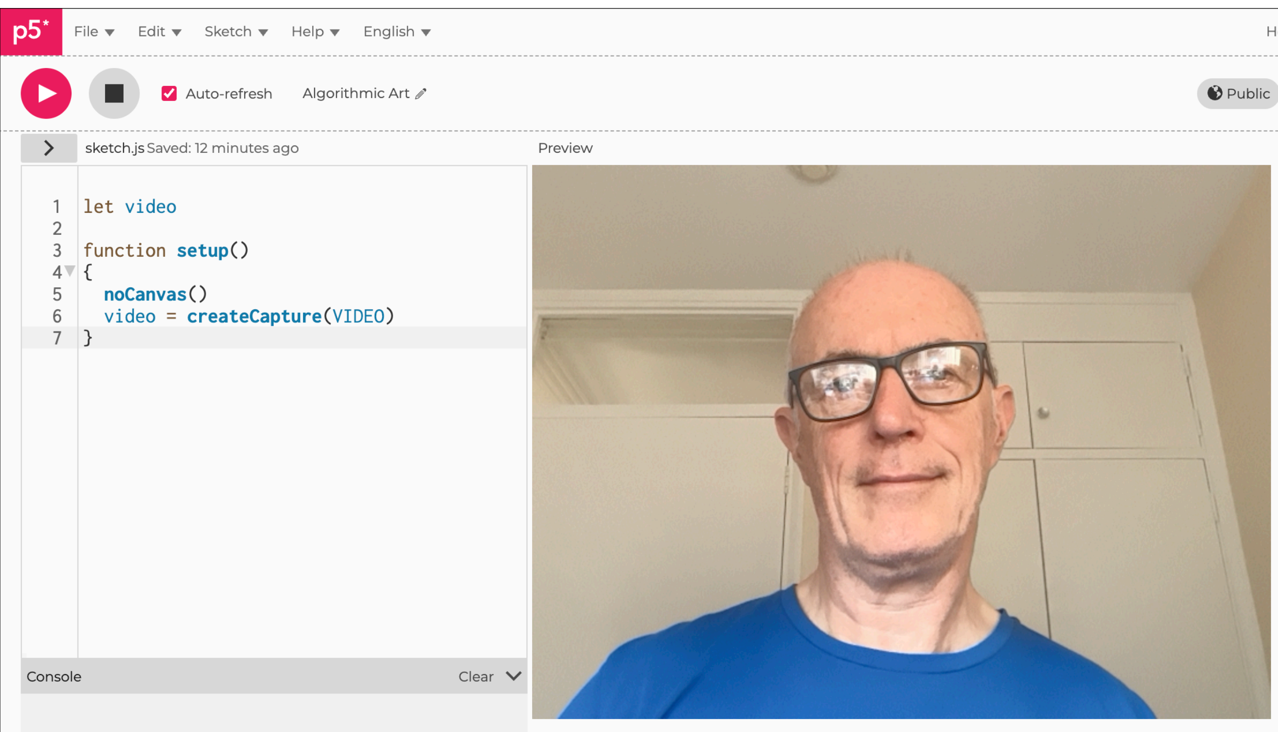
This calls on the computer to access your webcam; you will need to give it permission.

```
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

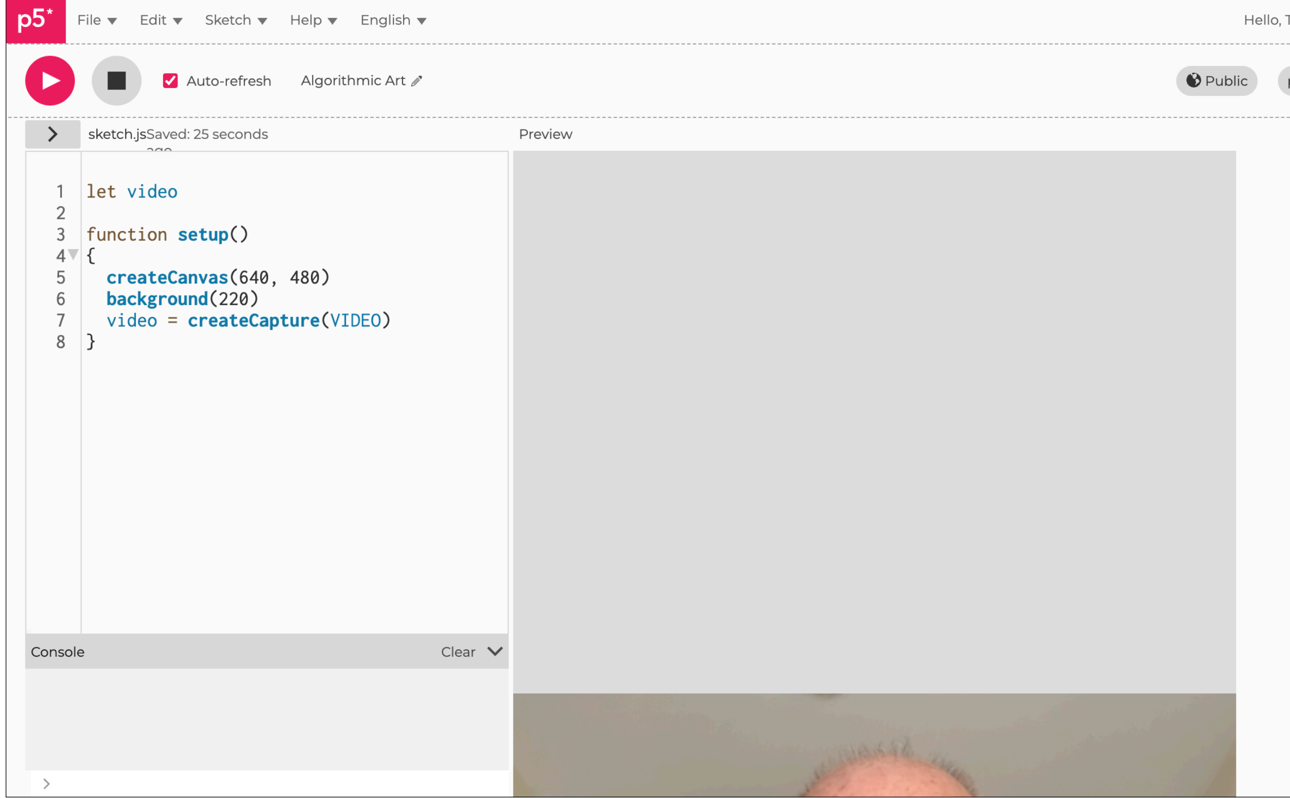
❗ Remove `noCanvas()` and replace with `createCanvas(640, 480)`. Now add a `background(220)`.

```
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)
}
```

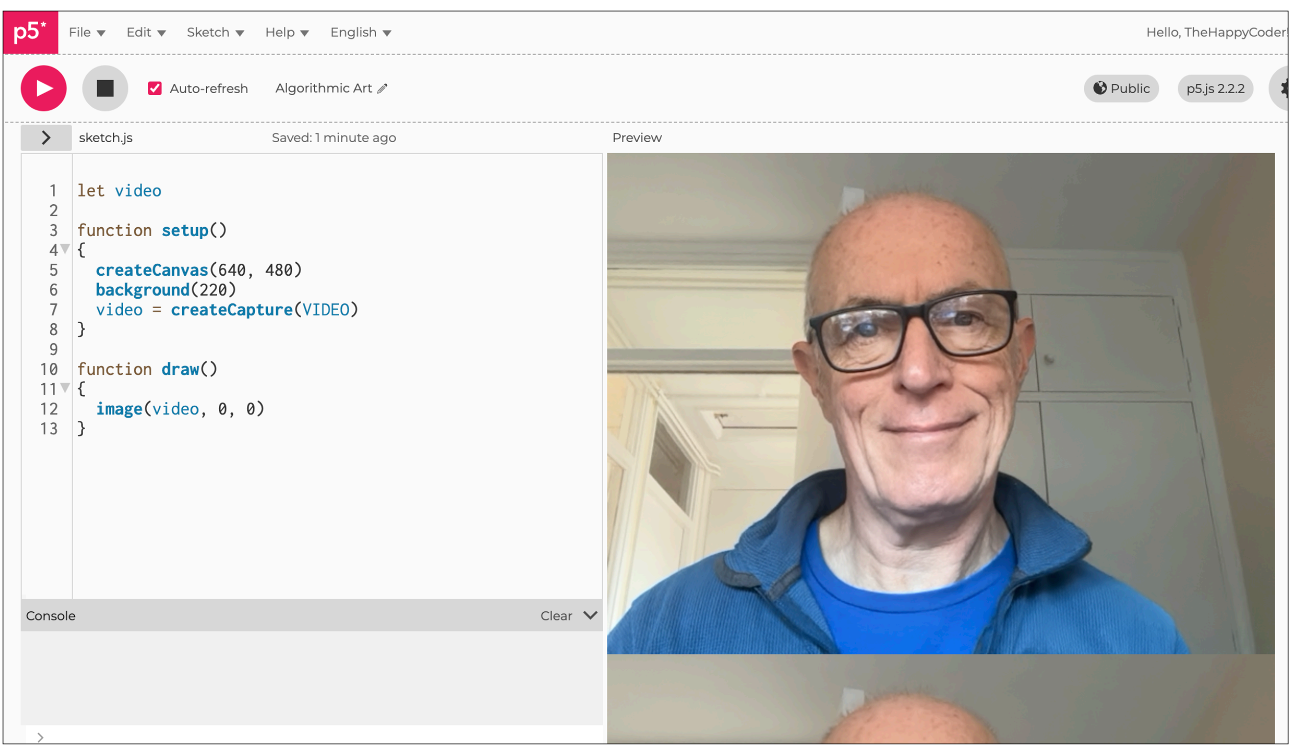
Notes

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)
}
```

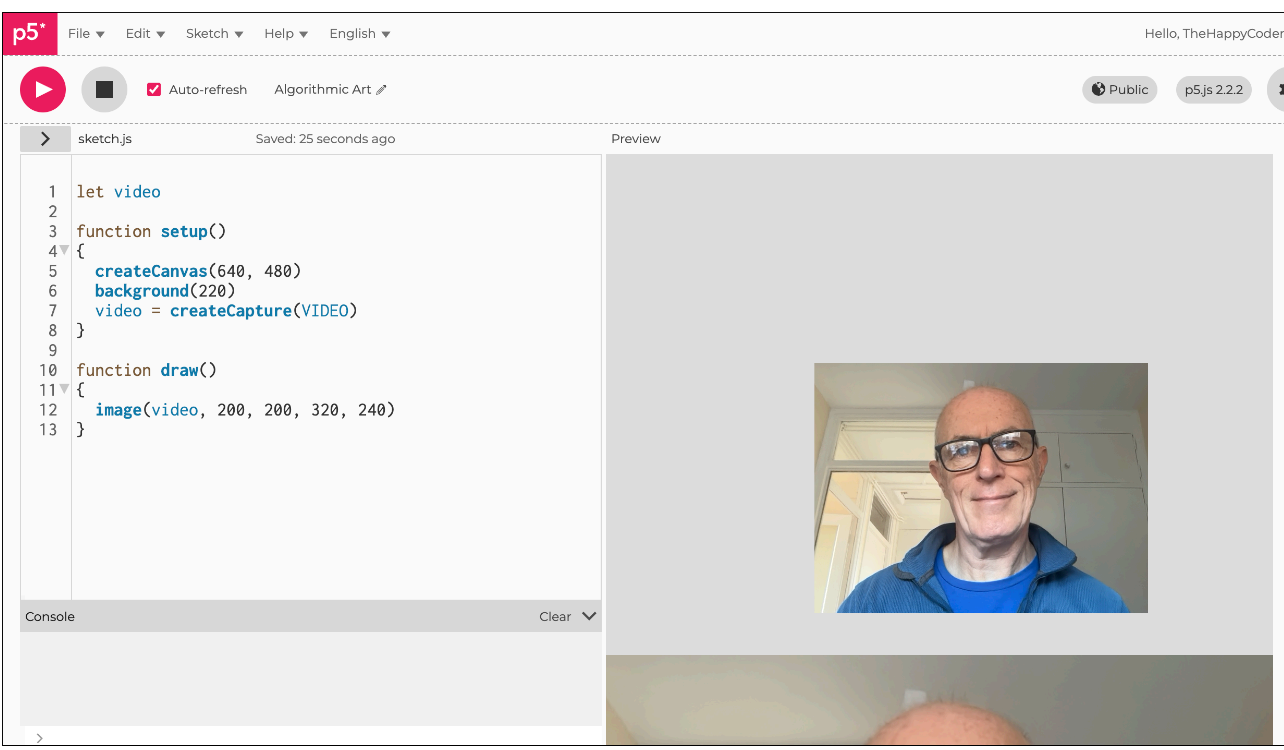
🗒️ Notes

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()`.

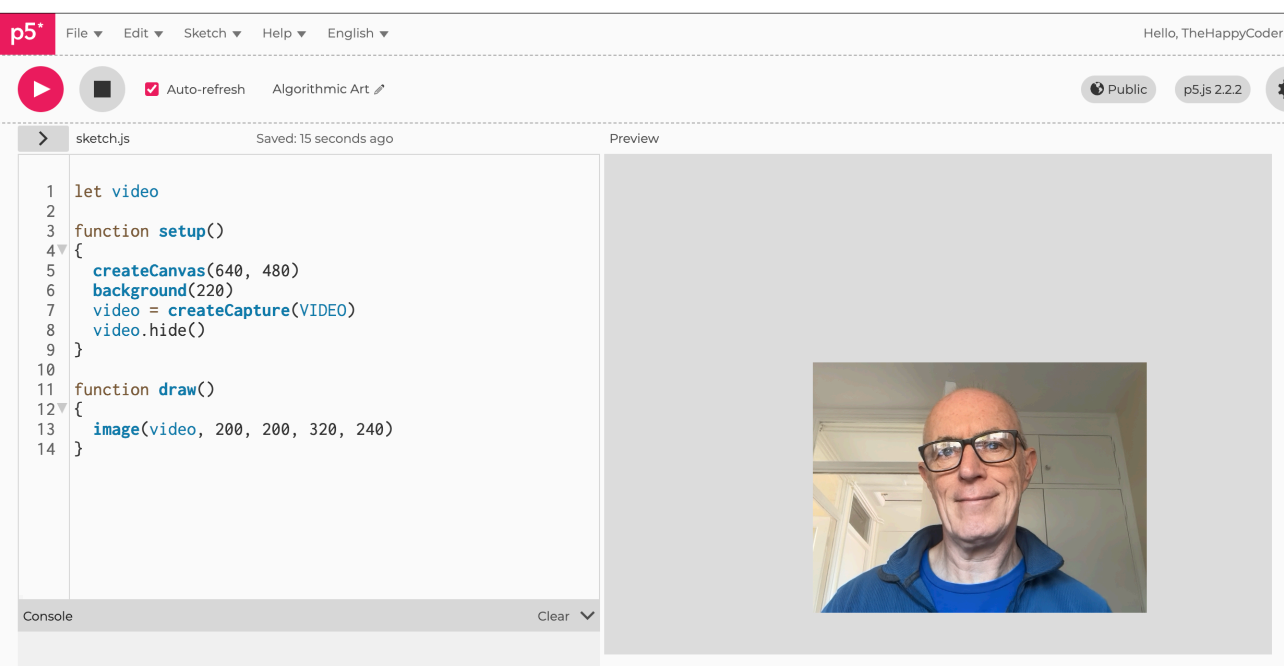
🌻 Challenges

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)
}
```

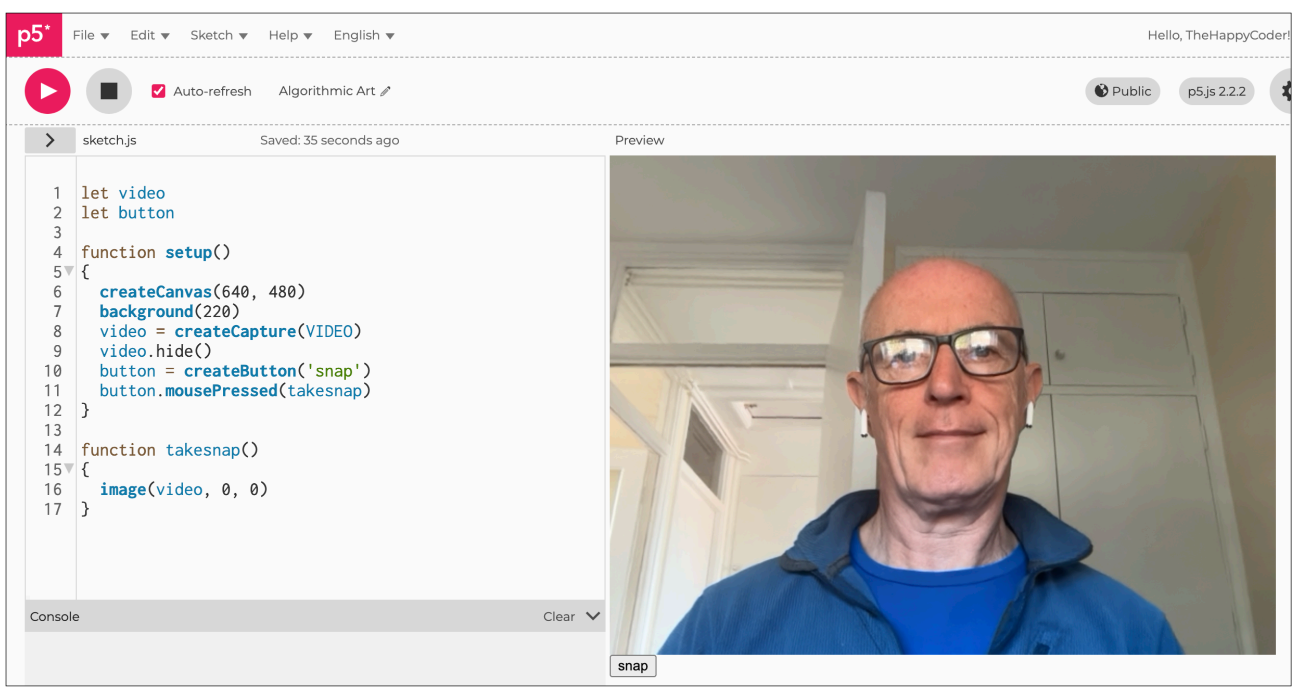
🗒️ Notes

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

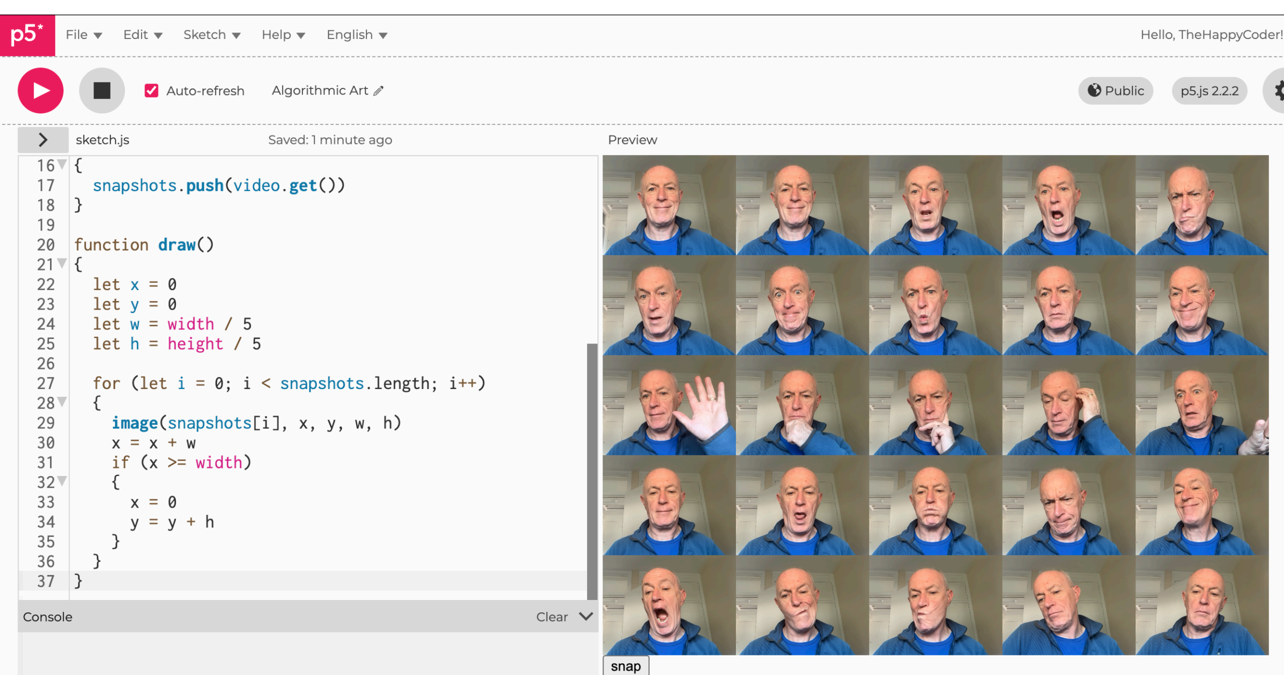
We create an empty array and call it `snapshots[]`. Every time the button is clicked, it adds an image to be stored in the array. The `get()` function gets the image, and the `push()` function pushes it into the array. By default, the `get()` function gets the whole image; you can specify the actual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

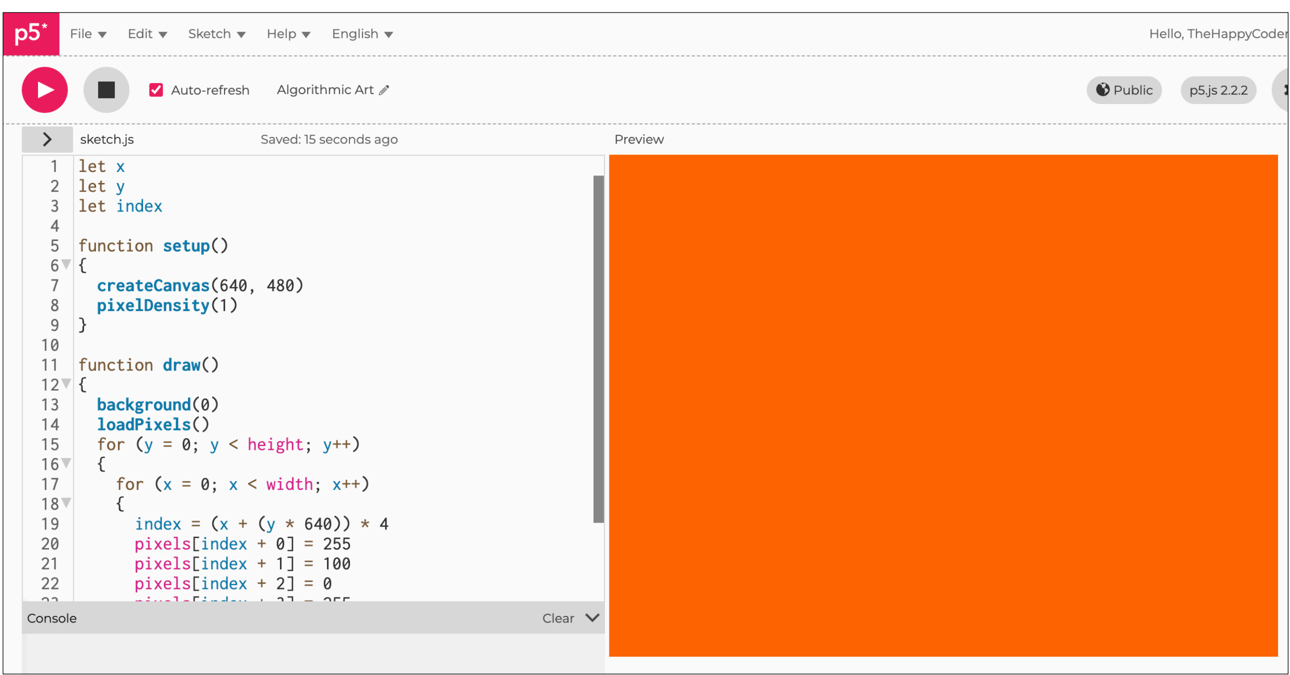
Goes through every fourth index. When it gets to the end of the row, it has done `640` of them, and so the next line of pixels starts with `640`. It sounds complicated, but this is how it works: through every pixel one at a time and then when it is at each pixel, works through the red, green, blue, and 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



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()
}
```

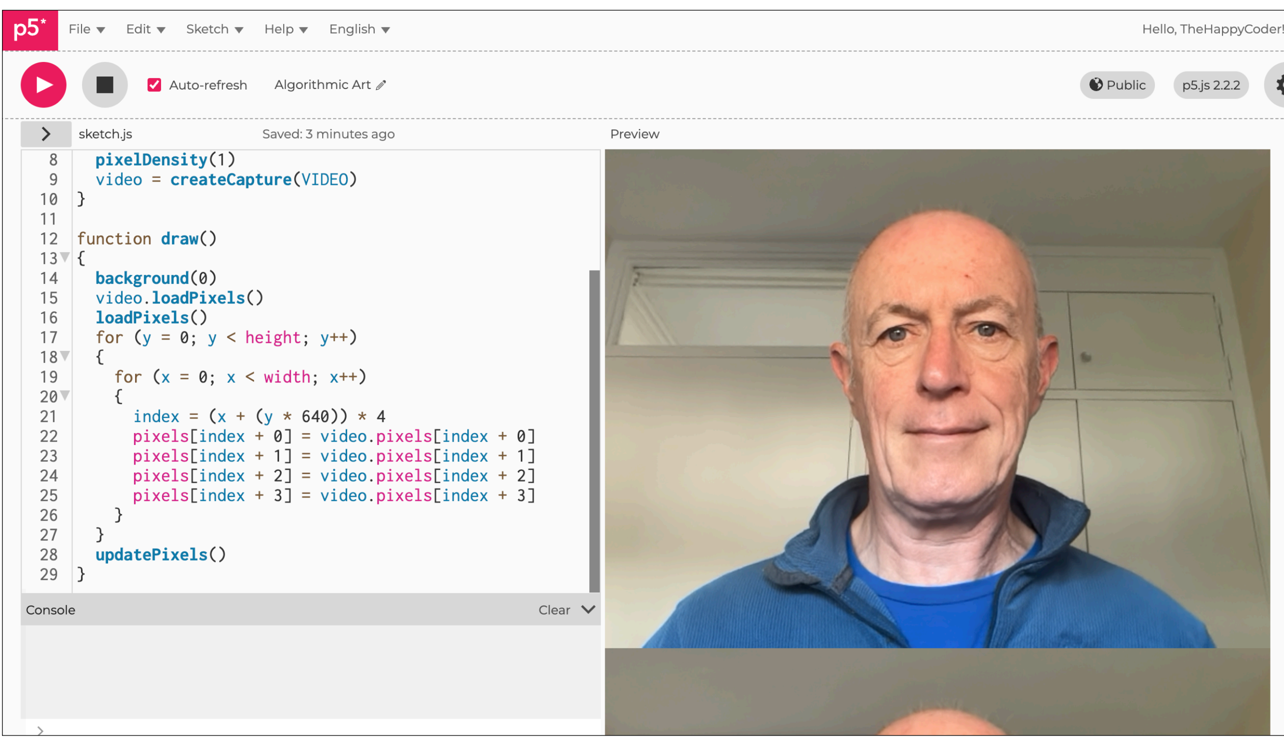
🗒️ Notes

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)
    }
  }
}
```

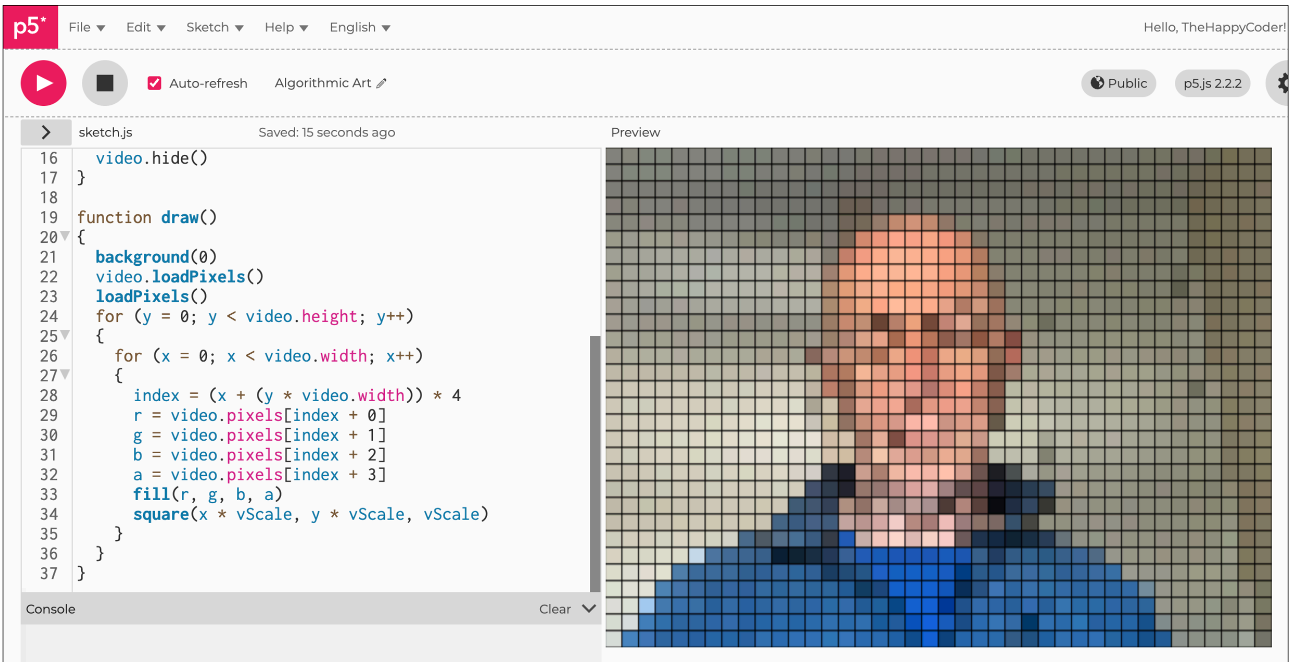
🗒️ Notes

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.

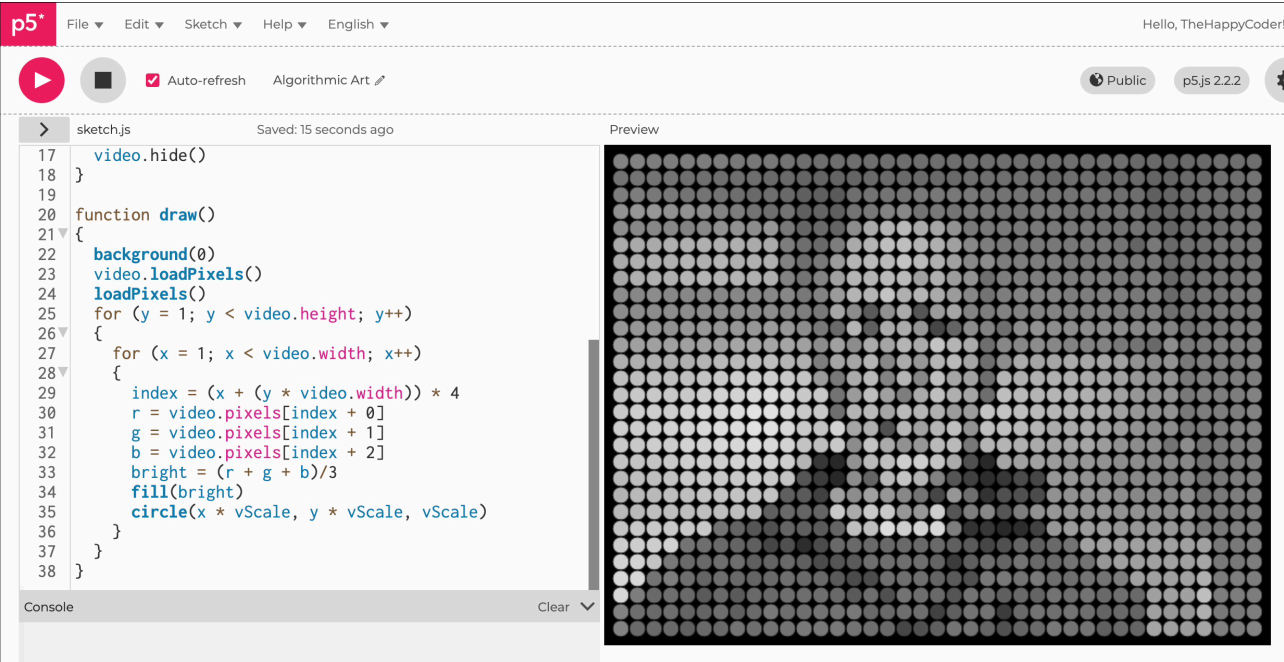
```
let x
let y
let index
let r
let g
let b
let a
let vScale = 16
let bright

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
```

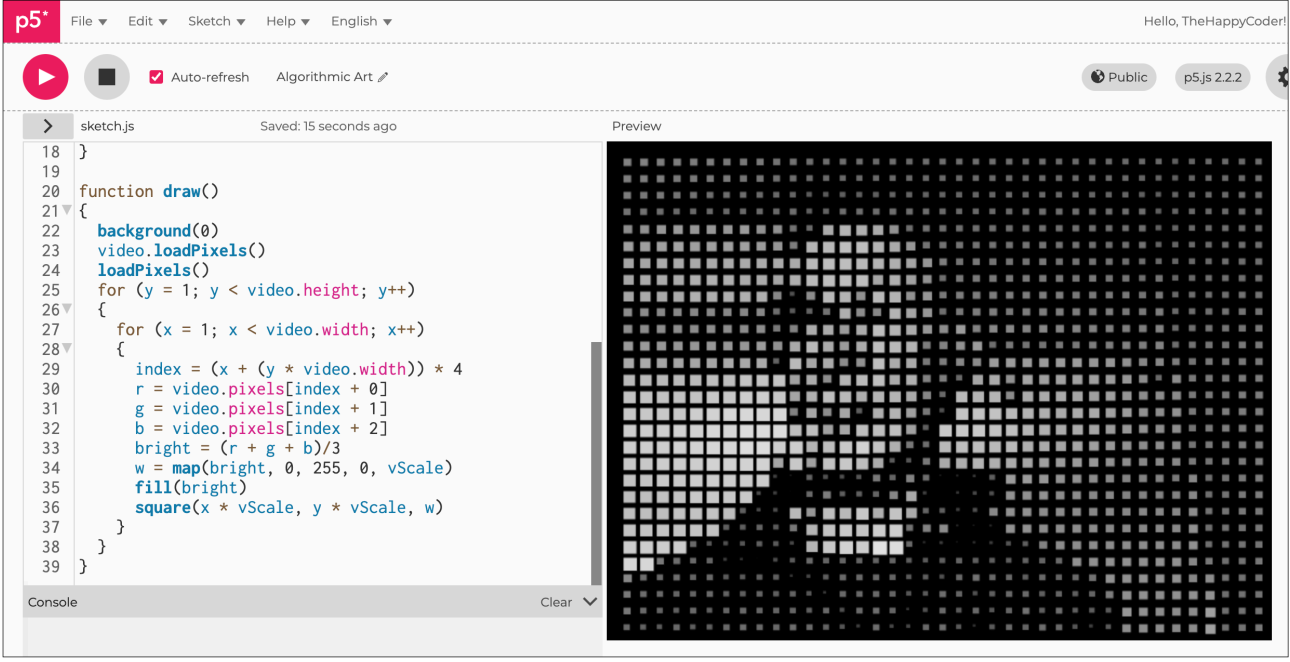
```
      w = map(bright, 0, 255, 0, vScale)
      fill(bright)
      square(x * vScale, y * vScale, w)
    }
  }
}
```

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)
    }
  }
}
```

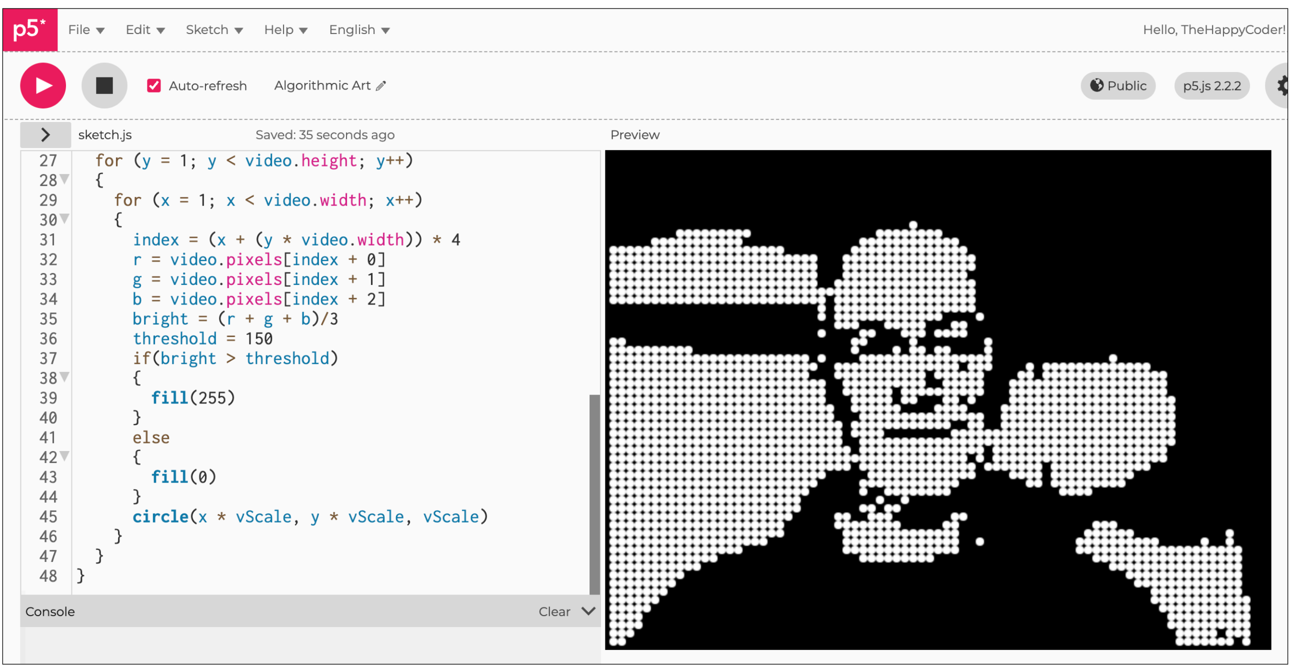
🗒️ Notes

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)
    }
  }
}
```

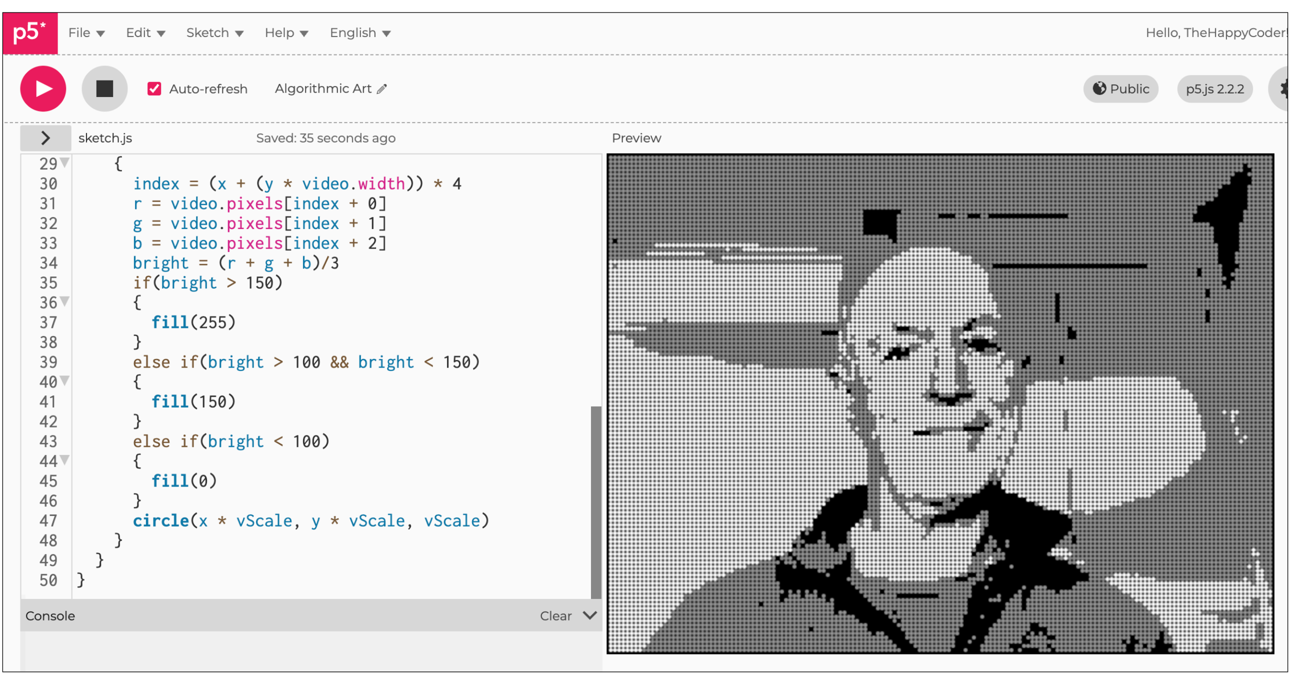
🗒️ Notes

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.

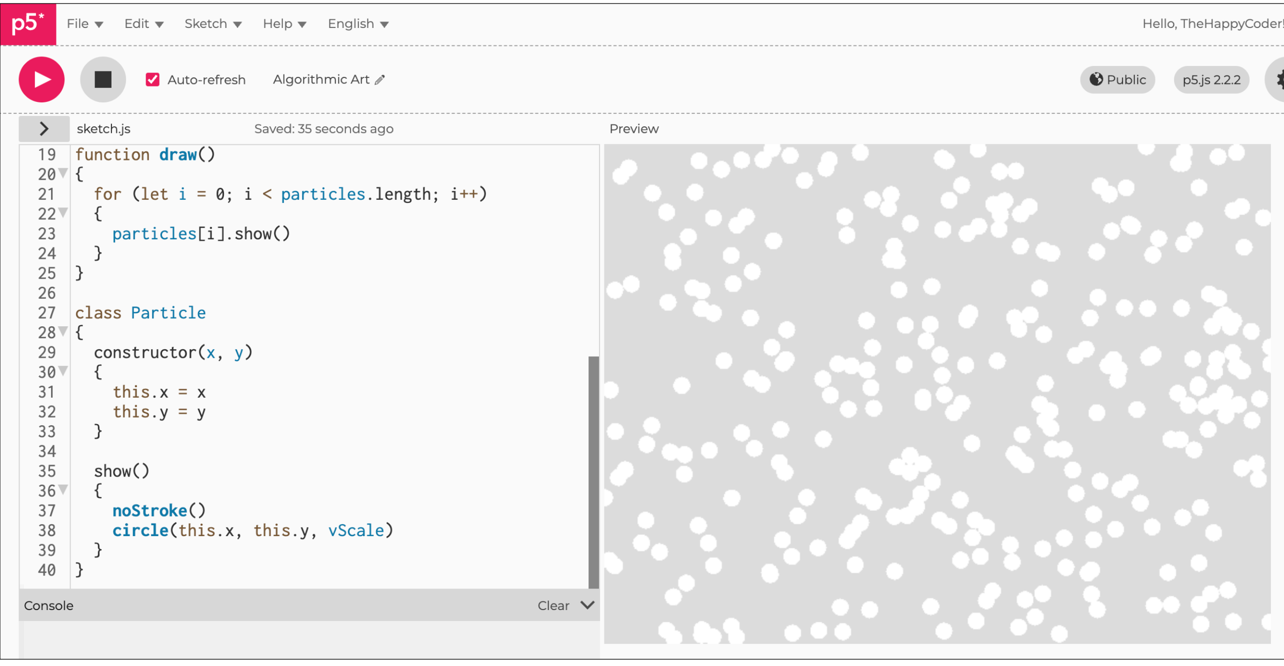
```
let video
let vScale = 16
let particles = []

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.

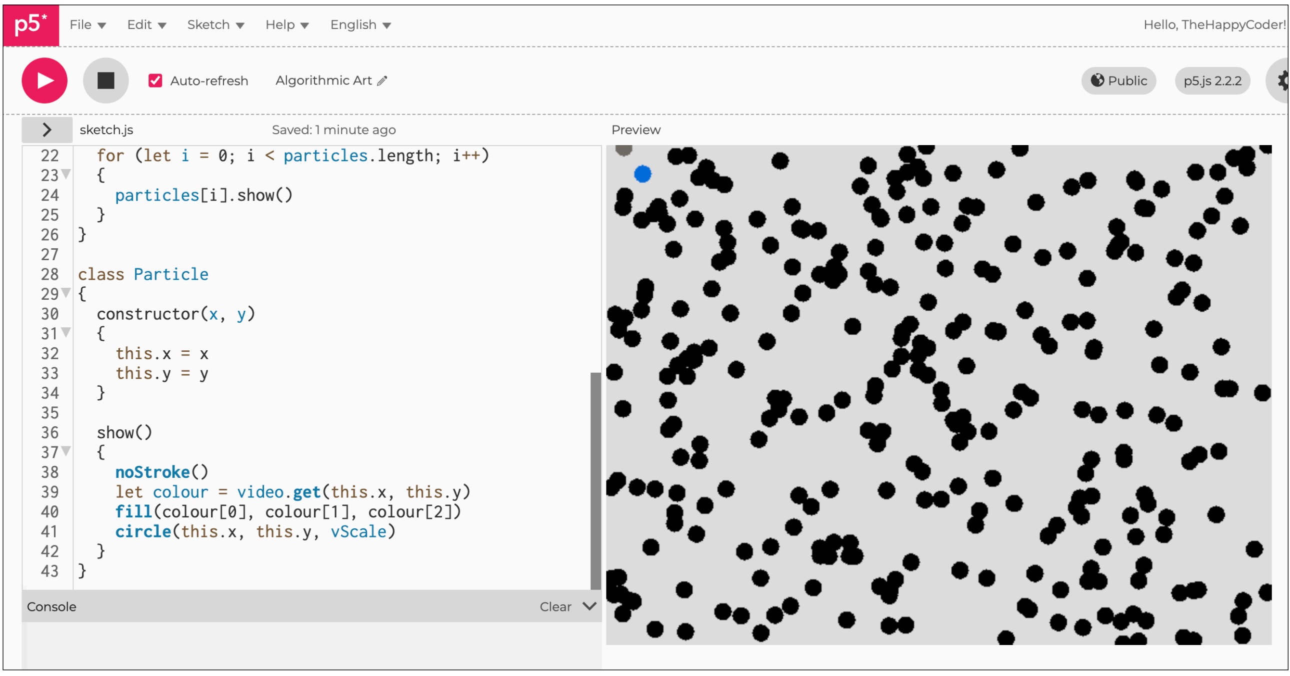
```
let video
let vScale = 16
let particles = []

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.

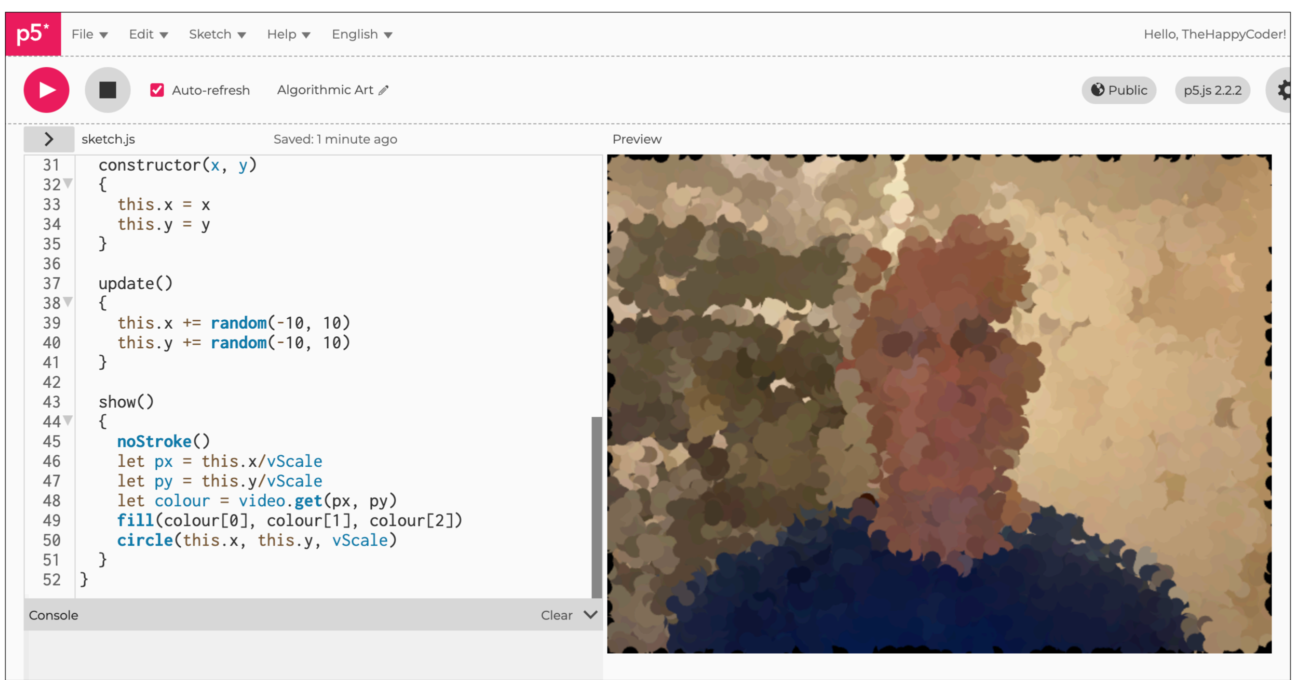
```
let video
let vScale = 16
let particles = []

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

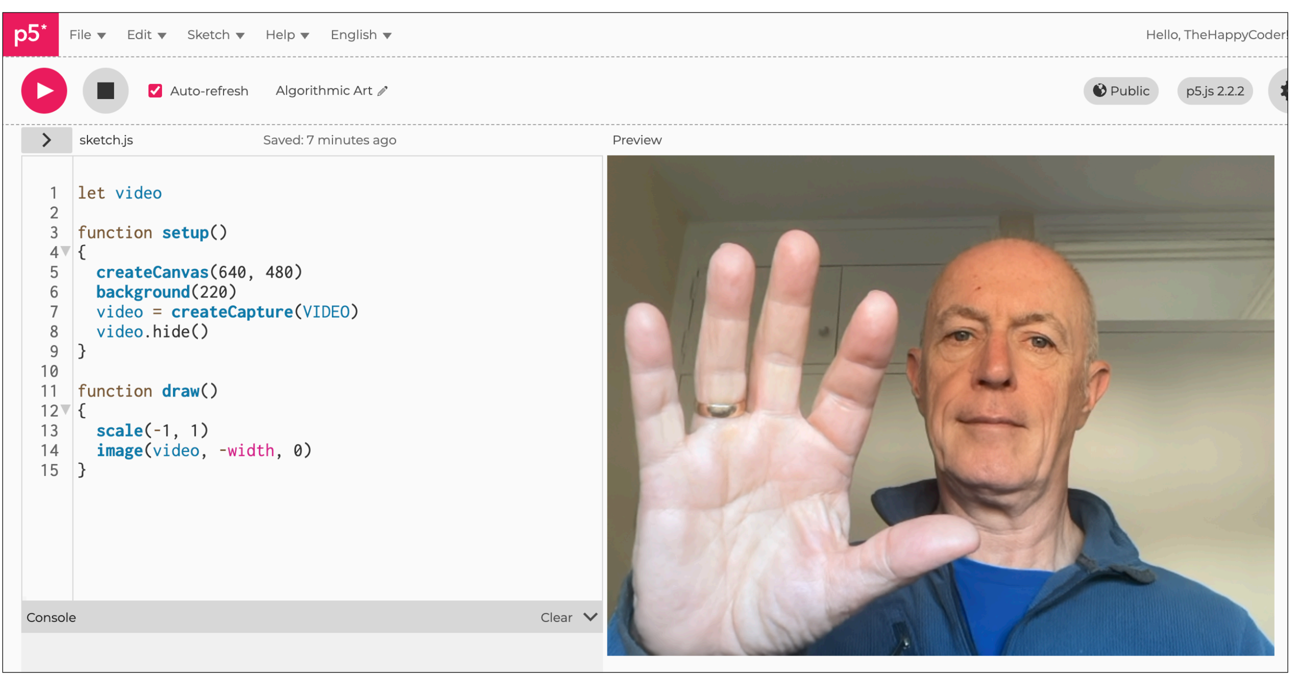
❗ Start a new sketch.
It makes things a bit more intuitive. Create a new temporary sketch for this illustration.

```
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?

Figure F3.25



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).

Figure F3.26