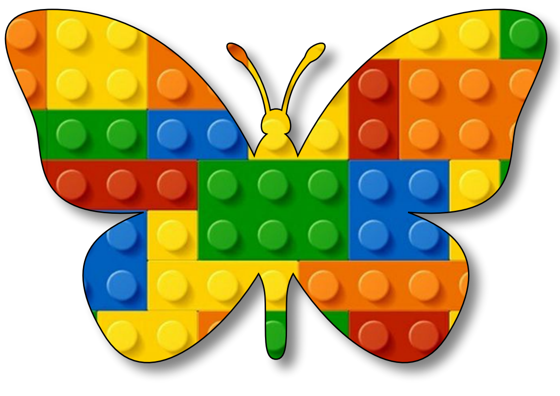
Artificial Intelligence Module A Unit #5 p5.js code snippets 2



Content

Module A Unit #5 snippets 2

Introduction to p5.js code snippets part 2

Some more useful snippets and information relevant to the next module. Just work through them, play, experiment and make sure you can follow the logic. Create something interesting yourself, it is the best way to learn.

We will be exploring how to incorporate video into our coding, in particular the webcam (if you have one) and also what we can do to all the pixels on the canvas.

Sketch A5.1 strings

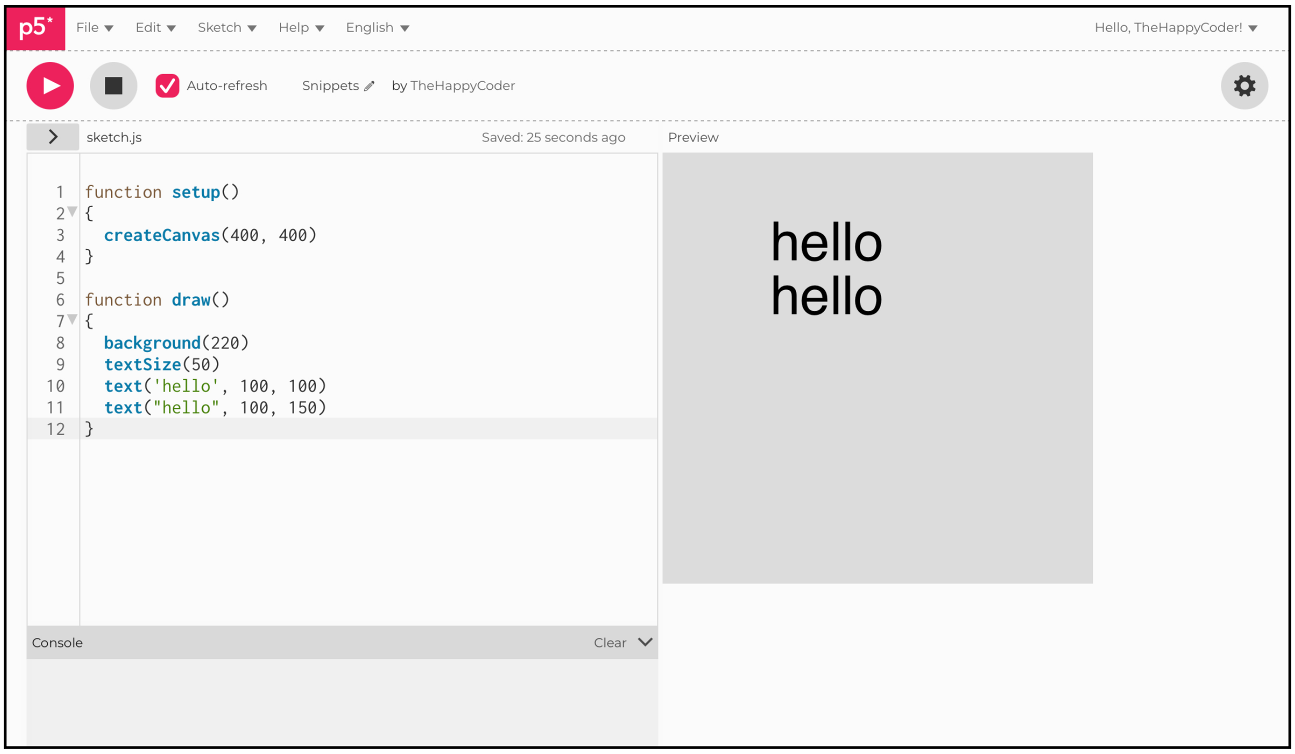
We have floats (numbers with a decimal place) and integers (numbers without a decimal place, whole numbers), another datatype is the `string`, which is either a letter, word, series of letters (and numbers) or even a number. It is then treated as text rather than an integer or a float. A `string` will have speech marks. You can usually use either single or double speech marks but never both in the same string. We will use it as a single letter or as a word or phrase.

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


function draw()
{
  background(220)
  textSize(50)
  text('hello', 100, 100)
  text("hello", 100, 150)
}
```

Notes

Either speech marks work

Figure A5.1



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

function draw()
{
  background(220)
  textSize(50)
  text('hello', 100, 100)
  text("hello", 100, 150)
}
```

```
hello
hello
```

Sketch A5.2 adding two strings

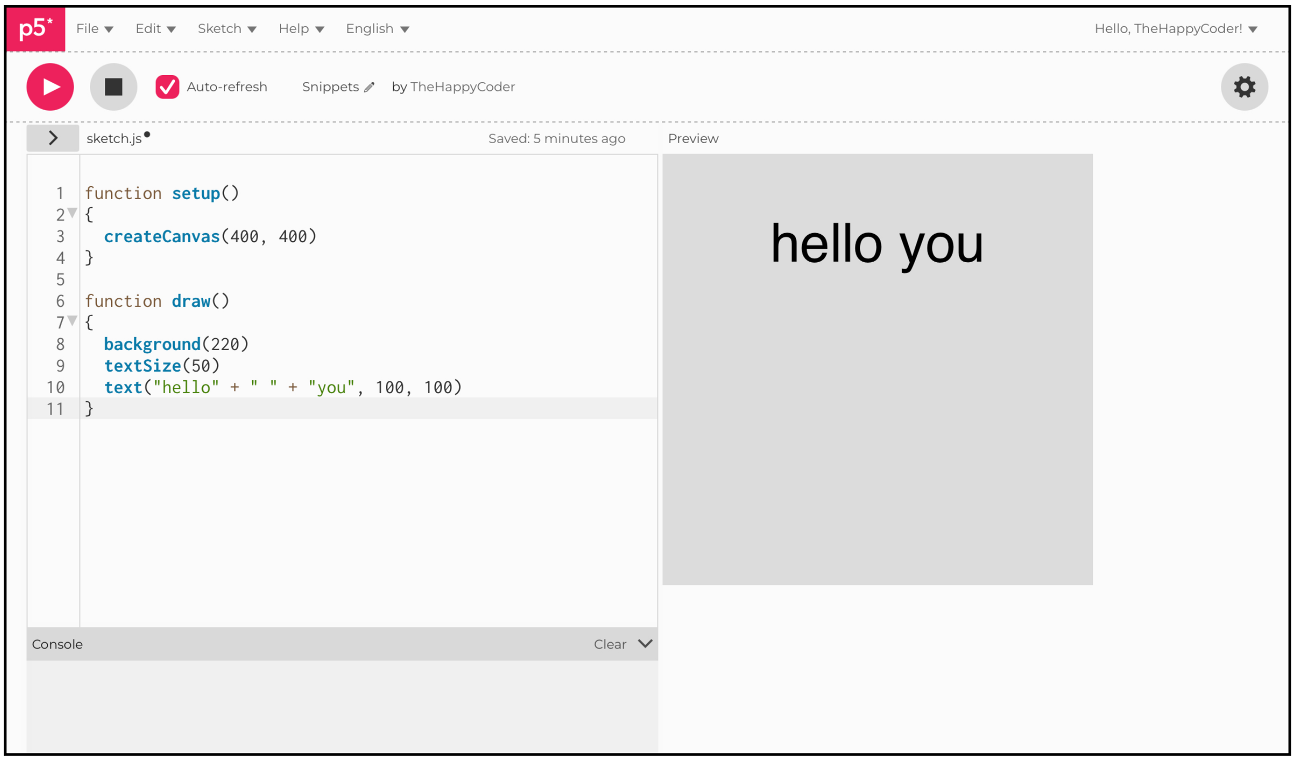
We can add strings together

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

function draw()
{
  background(220)
  textSize(50)
  text("hello" + " " + "you", 100, 100)
}
```

Notes

Just adding two strings together leaves no gap, so leave an empty string in between

Figure A5.2



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

function draw()
{
  background(220)
  textSize(50)
  text("hello" + " " + "you", 100, 100)
}
```

hello you

Sketch A5.3 strings as variables

❗ starting a new sketch

When we include a number and treat it as a string it does not behave like a number any more (except in certain situations).

```
let x = "20"
let y

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

function draw()
{
  background(220)
  textSize(50)
  y = x + 16
  text(x, 100, 100)
  text(y, 100, 150)
  text(x + y, 100, 200)
}
```

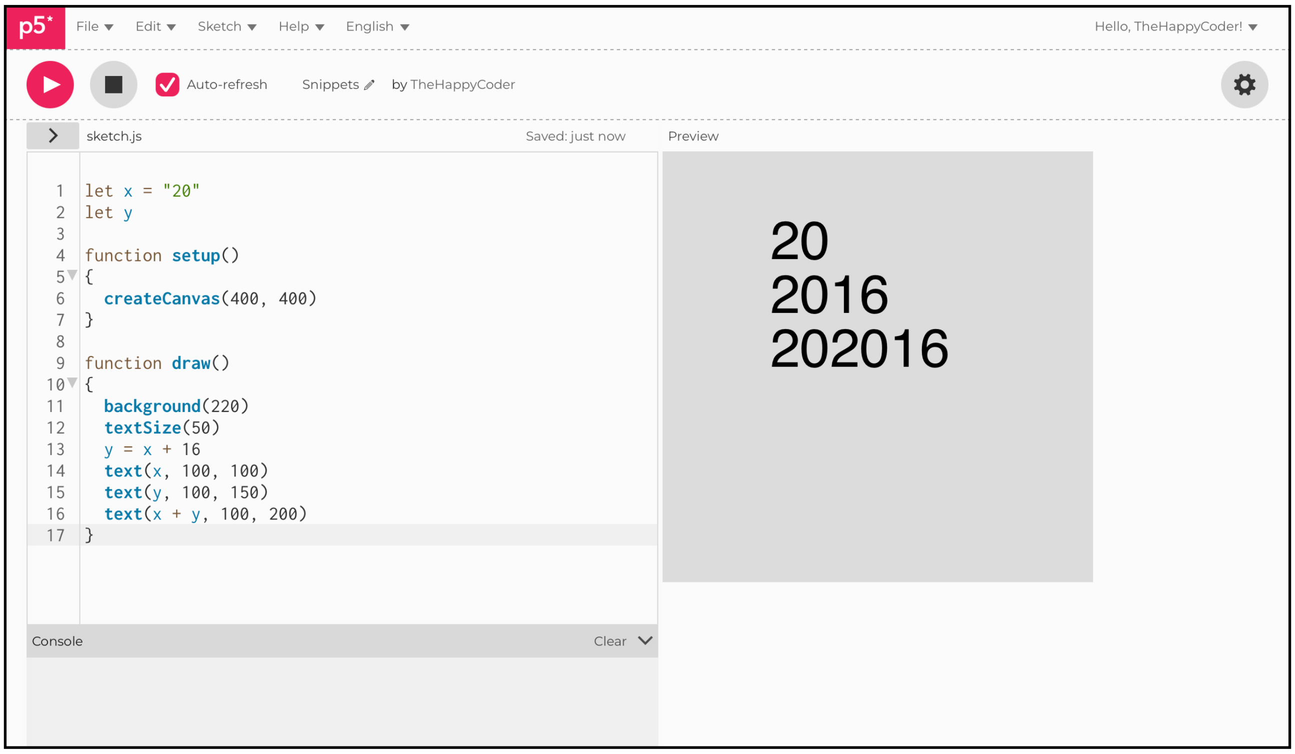
Notes

They behave quite differently

Challenge

Play around with this concept and get used to how strings and integers work, be careful as strings can suddenly start behaving as integers or floats.

Figure A5.3



```
let x = "20"
let y

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

function draw()
{
  background(220)
  textSize(50)
  y = x + 16
  text(x, 100, 100)
  text(y, 100, 150)
  text(x + y, 100, 200)
}
```

```
20
2016
202016
```

Sketch A5.4 the keyPressed() function

❗ starting a new sketch
You can use the keyboard as well as the mouse to to interact with the sketch. Here we change the colour of the circle from black to white and back again by pressing the w key for white fill, and the b key for black fill.

❗ You do need to click on the canvas after you have otherwise it will think you are still typing your code.

```
let value = 0

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

function draw()
{
  background(200)
  fill(value)
  circle(width/2, height/2, 200)
}

function keyPressed()
{
  if (key === 'w')
  {
    value = 255
  }
  else if (key === 'b')
  {
    value = 0
  }
}
```

🗒️ Notes

This only accepts lowercase, if that is a problem whereby there might be a mixture of uppercase and lower case then we can use another function to always make it uppercase.

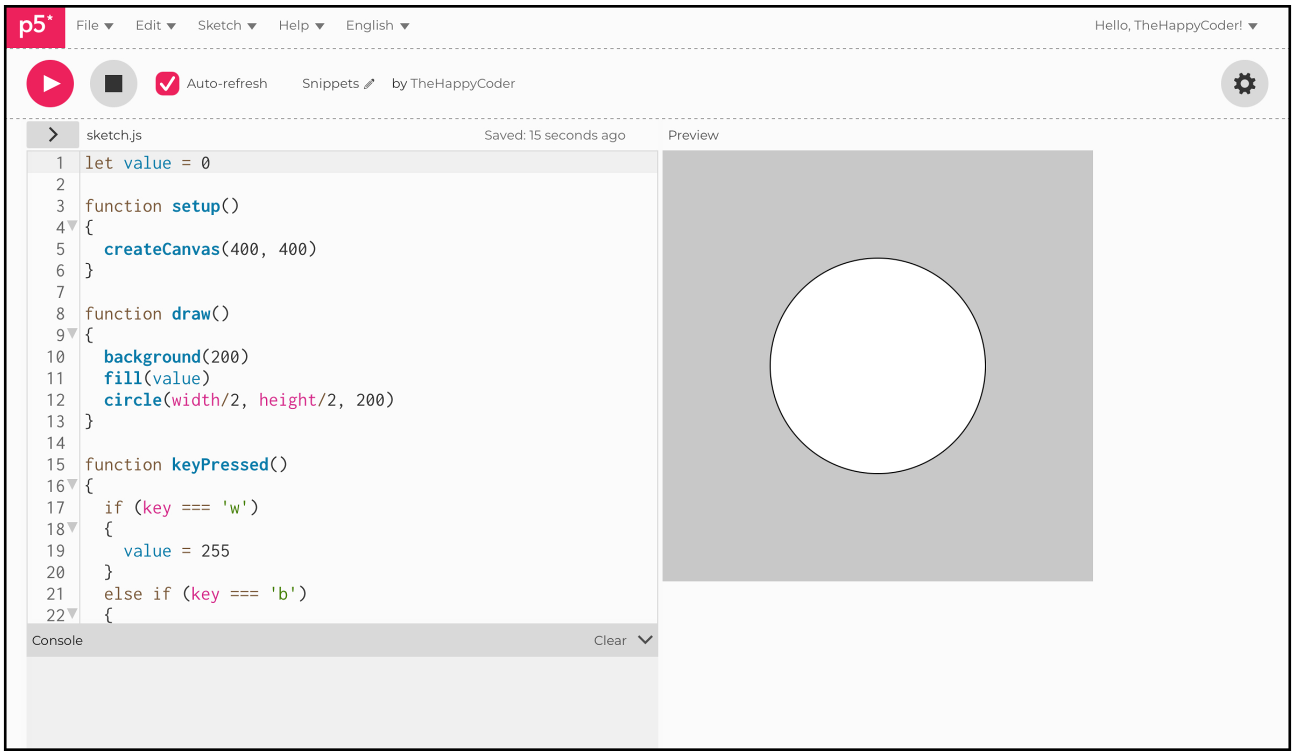
🌻 Challenge

Try other letters or use ENTER or LEFT_ARROW as the keys

🛠️ Code Explanation

`function keyPressed()`	Waits for a key to be pressed
`if (key === 'w')`	If the lowercase w is pressed
`else if (key === 'b')`	Or the lowercase b is pressed

Figure A5.4



```
let value = 0

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

function draw()
{
  background(200)
  fill(value)
  circle(width/2, height/2, 200)
}

function keyPressed()
{
  if (key === 'w')
  {
    value = 255
  }
  else if (key === 'b')
  {
```

What is the difference between == versus ===

In JavaScript, both == and === are used to compare values, but they have subtle differences, although most of the time either will work.

🀄 == (Loose Equality)
Compares values after performing type coercion (automatic conversion between data types). For example, '5' == 5 will evaluate to true because JavaScript converts the string "5" to the number 5 before comparison.

🀄 === (Strict Equality)
Compares both the values and the data types of the operands. For example, '5' === 5 will evaluate to false because the data types are different (string and number).

Use === whenever possible, as it provides stricter and more predictable comparisons. Use == only when you specifically intend to perform type coercion. By using ===, you can avoid unexpected behaviour and write more robust and maintainable JavaScript code.

This may seem academic but there is a difference even it sometimes feels a subtle one. I just wanted you to be aware that there is a difference and reason behind there being sometimes a == sign and at other times a ===. if unsure just use == unless you find there is a problem.

Sketch A5.5 the colour color() function

❗ starting a new sketch

This built-in function color() is very useful when you want to carry the RGB colour as a single variable rather than three separate variables. This is illustrated below.

```
let c

function setup()
{
  createCanvas(400, 400)
  c = color(255, 204, 0)
}

function draw()
{
  background(220)
  fill(c)
  circle(width/2, height/2, 200)
}
```

Notes

The above example simply carries the RGB in a single variable.

Challenge

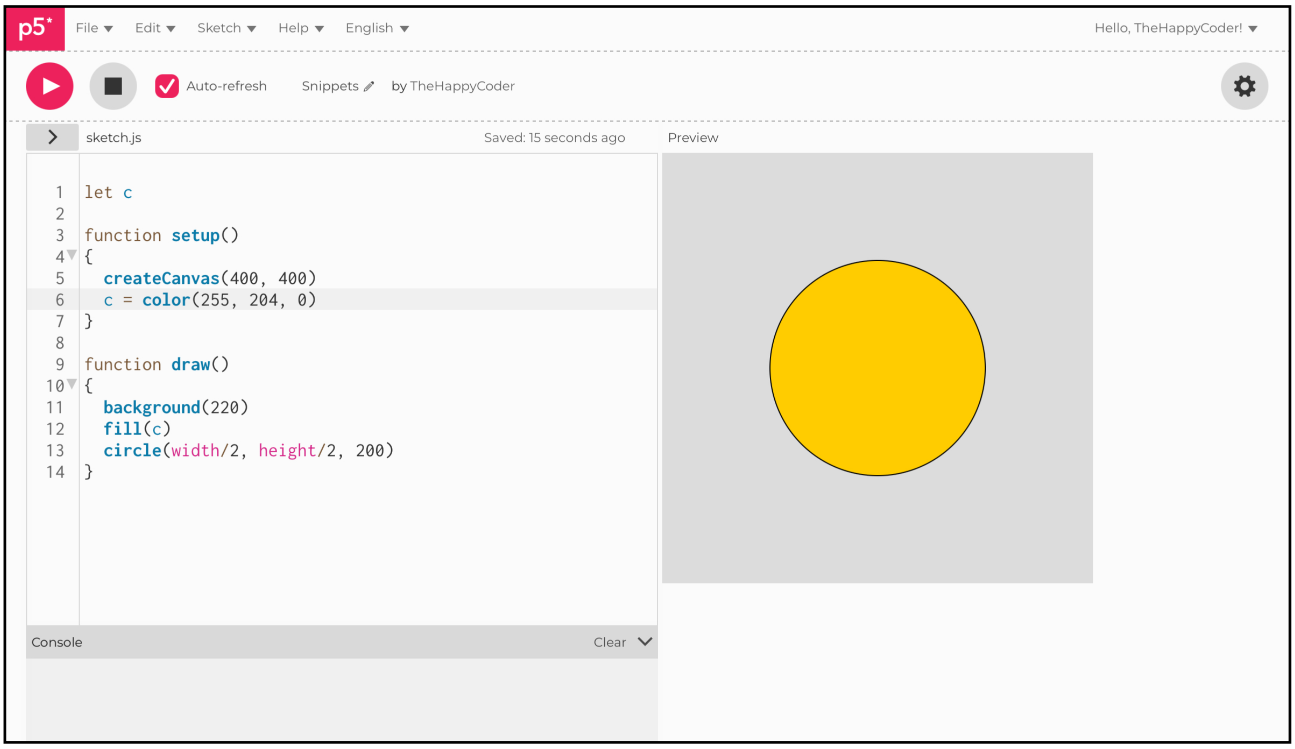
Try other colours

Try some alpha

Code Explanation

`c = color(255, 204, 0)`	The variable holds the three rgb values

Figure A5.5



```
let c

function setup()
{
  createCanvas(400, 400)
  c = color(255, 204, 0)
}

function draw()
{
  background(220)
  fill(c)
  circle(width/2, height/2, 200)
}
```

Sketch A5.6 sliders

❗ starting a new sketch

A slider is another useful interactive element we can use alongside the button already covered.

```
let slider


function setup()
{
  createCanvas(400, 400)
  slider = createSlider(0, 255, 0)
  slider.position(100, 50)
  slider.size(200)
}


function draw()
{
  background(150)
  let c = slider.value()
  fill(c)
  circle(width/2, height/2, 200)
}
```

Notes

This just gives us a greyscale value to fill the circle

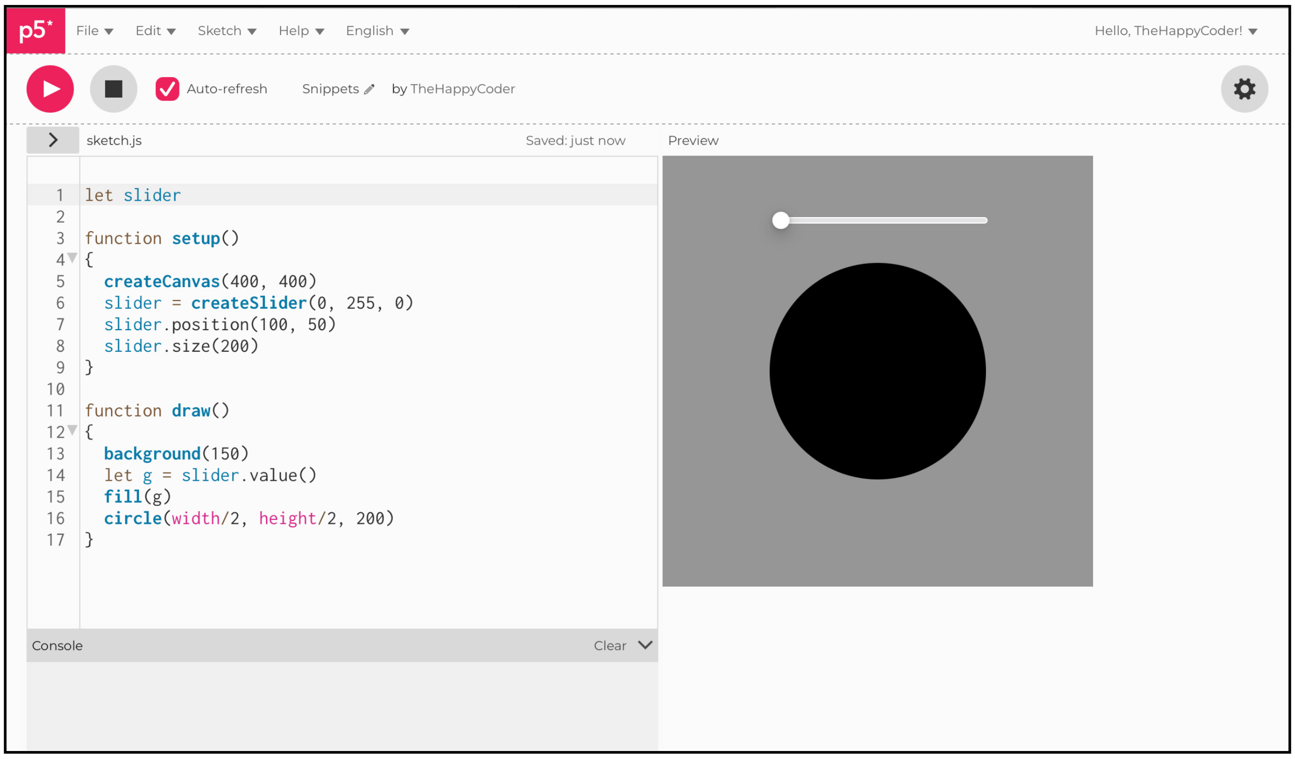
Challenges

1. Have three sliders one for each r, g, b
2. Move the slider around, different sizes and values (min, max, and initial values)

🛠️ Code Explanation

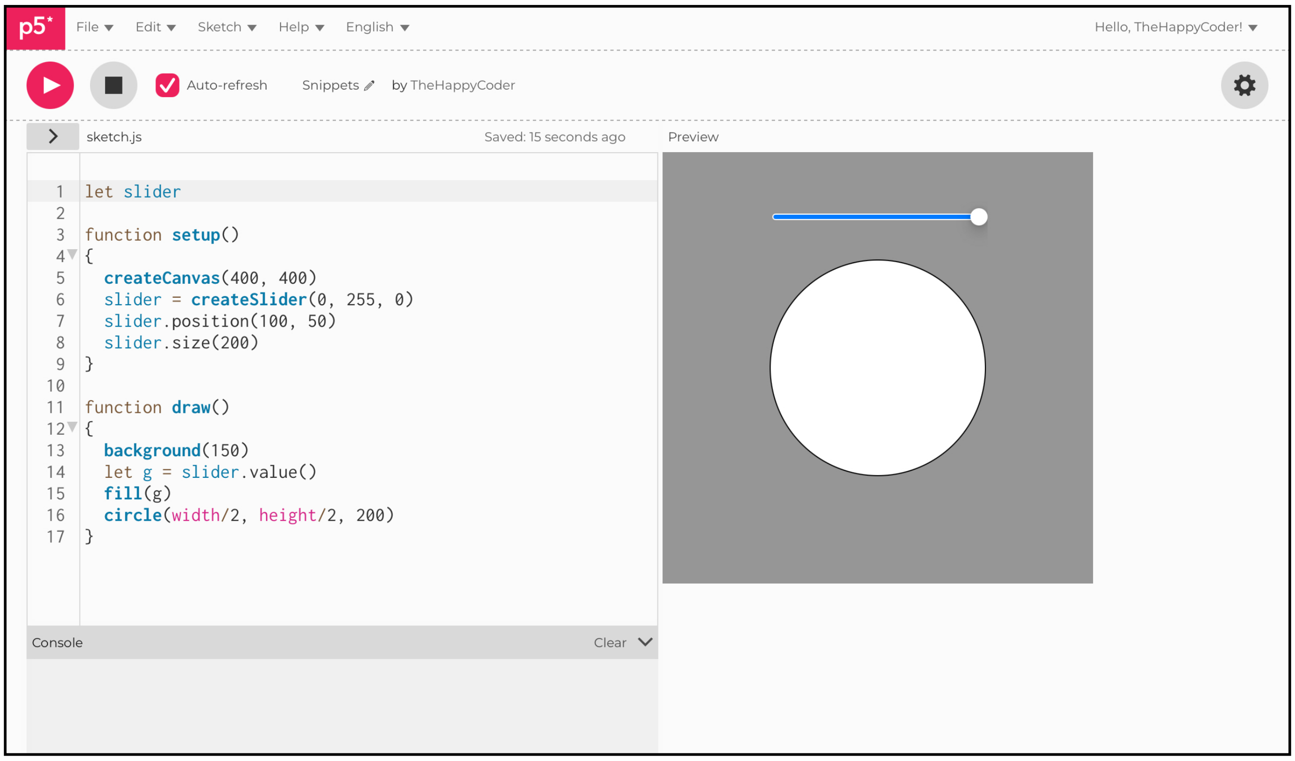
`let slider`	Slider variable name
`slider = createSlider(0, 255, 0)`	Creating the slider, with the minimum value, the maximum value and the initial starting value
`slider.position(100, 50)`	Where to position it on the canvas
`slider.size(200)`	The size of the slider
`let c = slider.value()`	Taking the value of the slider

Figure A5.6a



```
let slider

function setup()
{
  createCanvas(400, 400)
  slider = createSlider(0, 255, 0)
  slider.position(100, 50)
  slider.size(200)
}

function draw()
{
  background(150)
  let g = slider.value()
  fill(g)
  circle(width/2, height/2, 200)
}
```

Figure A5.6b



```
let slider

function setup()
{
  createCanvas(400, 400)
  slider = createSlider(0, 255, 0)
  slider.position(100, 50)
  slider.size(200)
}

function draw()
{
  background(150)
  let g = slider.value()
  fill(g)
  circle(width/2, height/2, 200)
}
```

Sketch A5.7 another shape, the square

❗ starting a new sketch

Introducing the `square`, we have drawn it in the centre of the canvas with a side length of `100` pixels.

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

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

Notes

You will notice that, although we gave it the coordinates for the centre, it drew the square with the top left hand corner in the middle of the canvas.

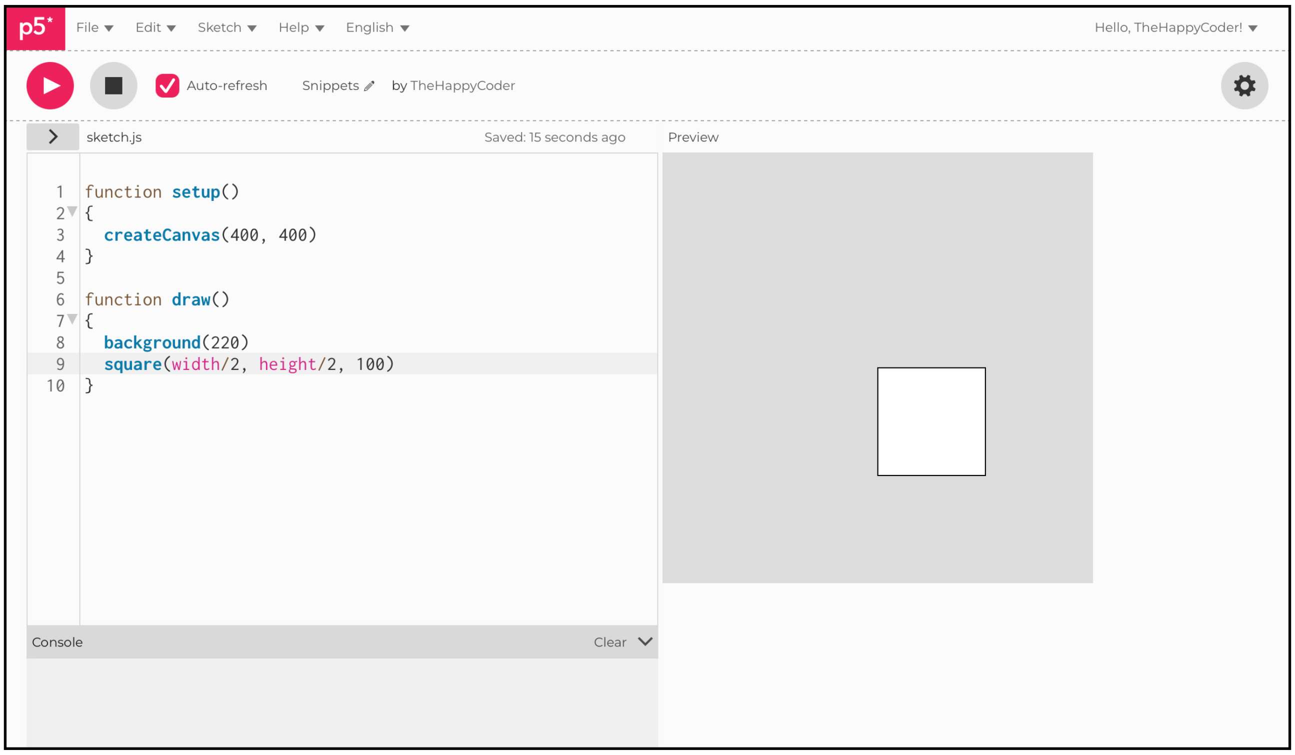
Challenge

How do you think you could change that so that the square is in the middle?

Code Explanation

`square(width/2, height/2, 100)`	Square in the middle of the canvas with a side length of 100

Figure A5.7



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

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

Sketch A5.8 the centre of the square

We can add a function called rectMode() that can move the origin coordinates of the square to the centre.

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

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

Notes

The square is now in the middle of the canvas.

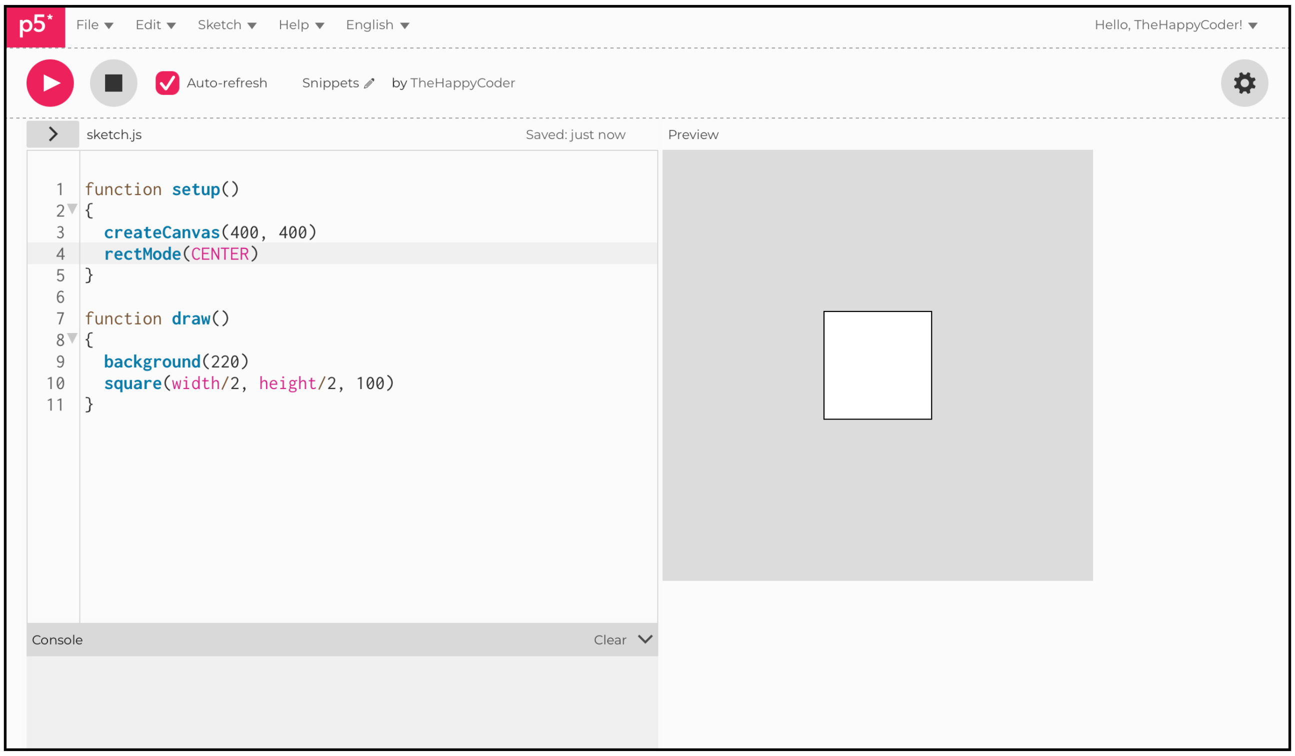
Challenge

Draw lots of squares with different positions and dimensions.

Code Explanation

rectMode(CENTER)	Function to move the co-ordinates to the centre of the square

Figure A5.8



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

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

Introduction to using the video input

We can use the built-in webcam of your computer in our code if you have one, if not then you will need a plug one in or skip this module. Most laptops, computers and tablets have a webcam these days. You could even use your smart phone at a push if necessary (why not).

The image comes in the size and ration of `640x480` pixels. We will look at ways we can integrate it into our code and in further units we will use the webcam for machine learning.

❗ a few of the following units use a webcam of sorts, especially the pretrained units.

Sketch A5.9 video capture

❗ starting a new sketch

The createCapture(VIDEO) function will ask the computer for access to your webcam, you will need to give it permission

```
let video


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

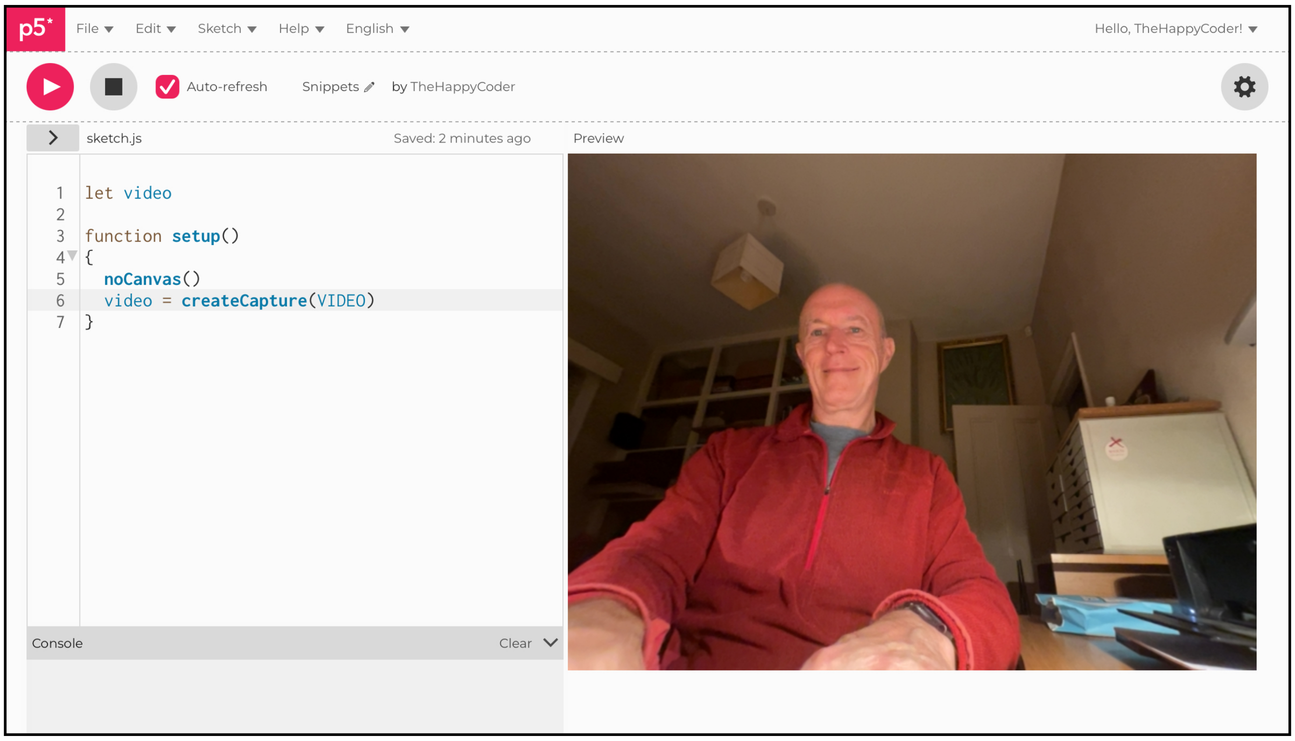
Notes

The default size is 640x480 pixels displayed in the window.

Code Explanation

`let video`	Create a variable to hold the video image
`noCanvas()`	Remove the canvas
`video = createCapture(VIDEO)`	Create the video from the webcam and attribute it to the variable

Figure A5.9



```
let video

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

Sketch A5.10 returning the canvas

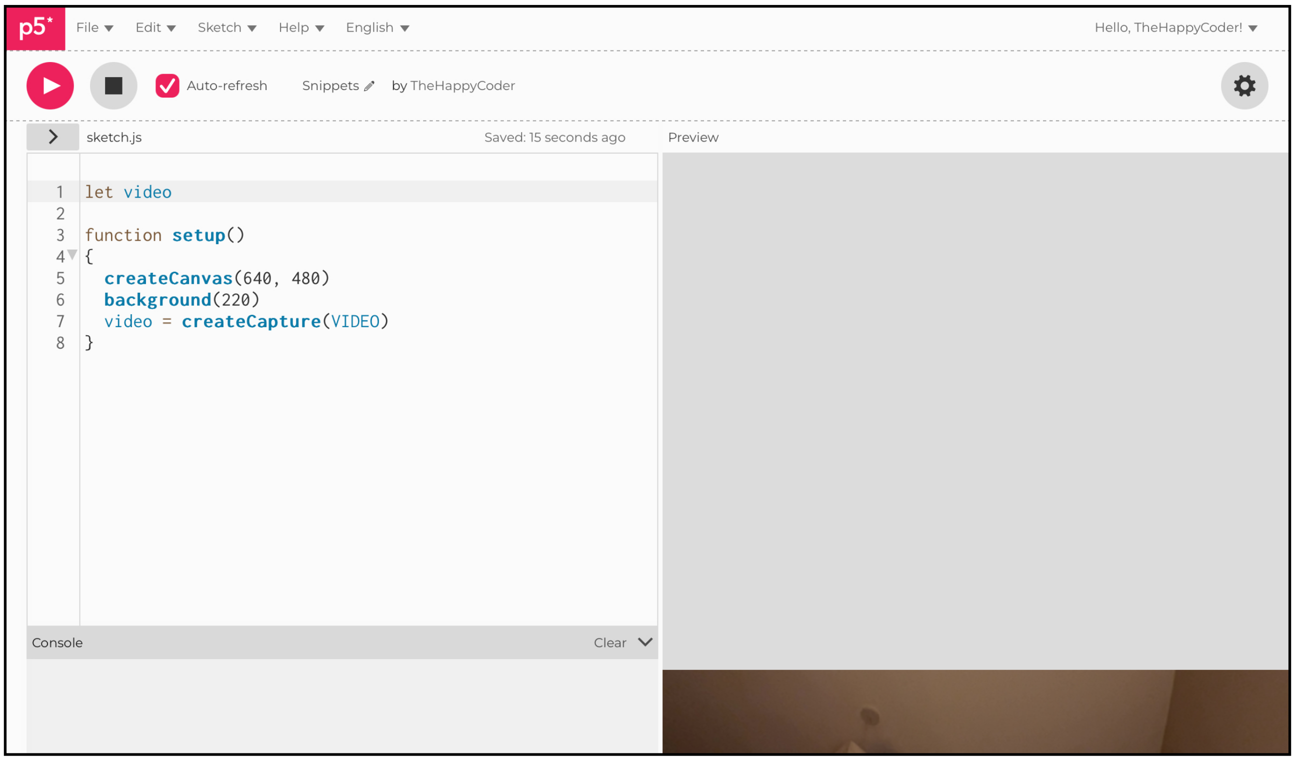
Replace the noCanvas() with createCanvas(640, 480) so we have a canvas that we can fill with the video image. Add the background.

```
let video


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

Notes

Unfortunately we get the canvas with the video image below it

Figure A5.10



```
let video

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

Sketch A5.11 video on the canvas

Now we are going to add the video to the canvas with the image() function. We set the coordinates to the top left hand corner of (0, 0)

```
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, the bottom one is the live video stream.

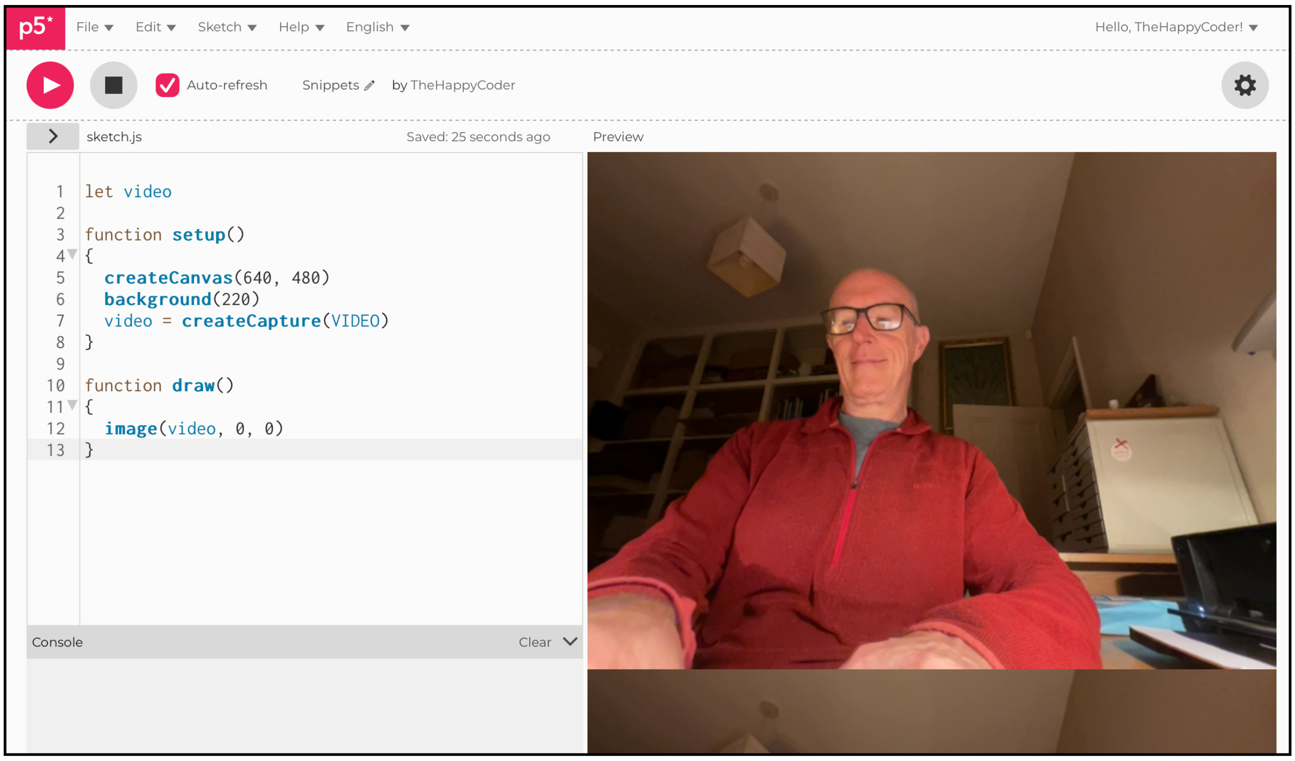
Challenges

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

🛠️ Code Explanation

`image(video, 0, 0)`	Draws an image to the canvas, in this case it is a video and its coordinates are (0, 0)

Figure A5.11



```
let video

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

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

Sketch A5.12 hide the video

We can hide the video and just have it on the canvas

```
let video

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

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

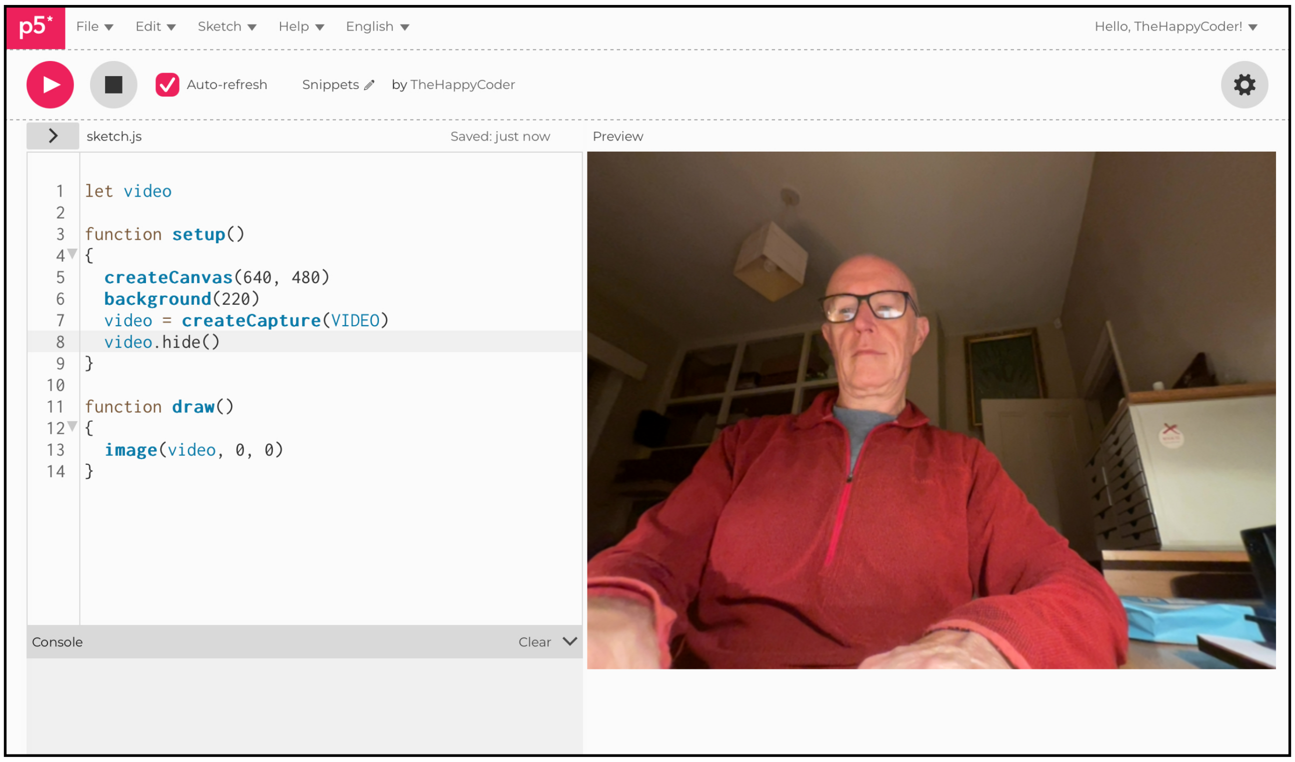
Notes

We now now the one video on the canvas

Code Explanation

`video.hide()`	This hides the streaming video

Figure A5.12



```
let video

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

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

Sketch A5.13 flipping the video

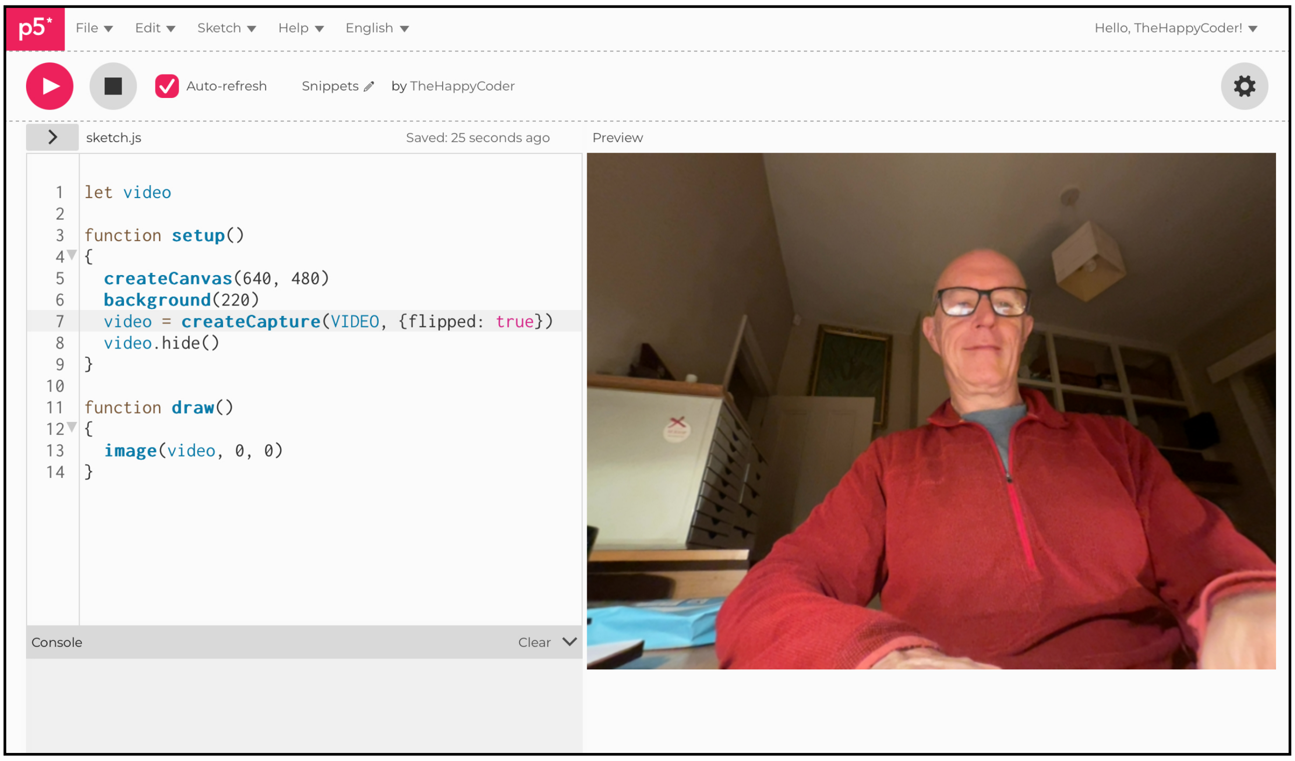
We can change the video so that it mirrors your movements, as if you were looking in a mirror. This feels a little more intuitive.

```
let video

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

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

Code Explanation

`video = createCapture(VIDEO, {flipped: true})`	This reverses the image to make it more like a mirror

Figure A5.13



```
let video

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

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

Introduction to the pixels

A canvas is made up of lots of pixels. All are very tiny and hard to see with naked eye. In a 400x400 canvas there are 160,000 pixels. Each pixel has four channels, a `red`, a `green`, a `blue` and an `alpha`. The alpha is the transparency and all have values in the range 0 to 255.

Yet we can manipulate the pixels and even the channels by picking out the individual elements that make up a pixel. We could take all the pixels and make them green and so on. There are two main functions we will use, the first is called `loadPixels()` which will look at all the pixels in the canvas or the image and store them in a pixel array (all the red, green, blue and alpha for every pixel) in one long array (640,000 elements). So to go from one pixel to another we have to jump over four in the array.

Once we have done something to the pixels we then use the other main function called `updatePixels()` to display the new, changed pixels on the canvas.

❗ we have to address pixel density because of there are actually more channels than four with HD, more on that later.

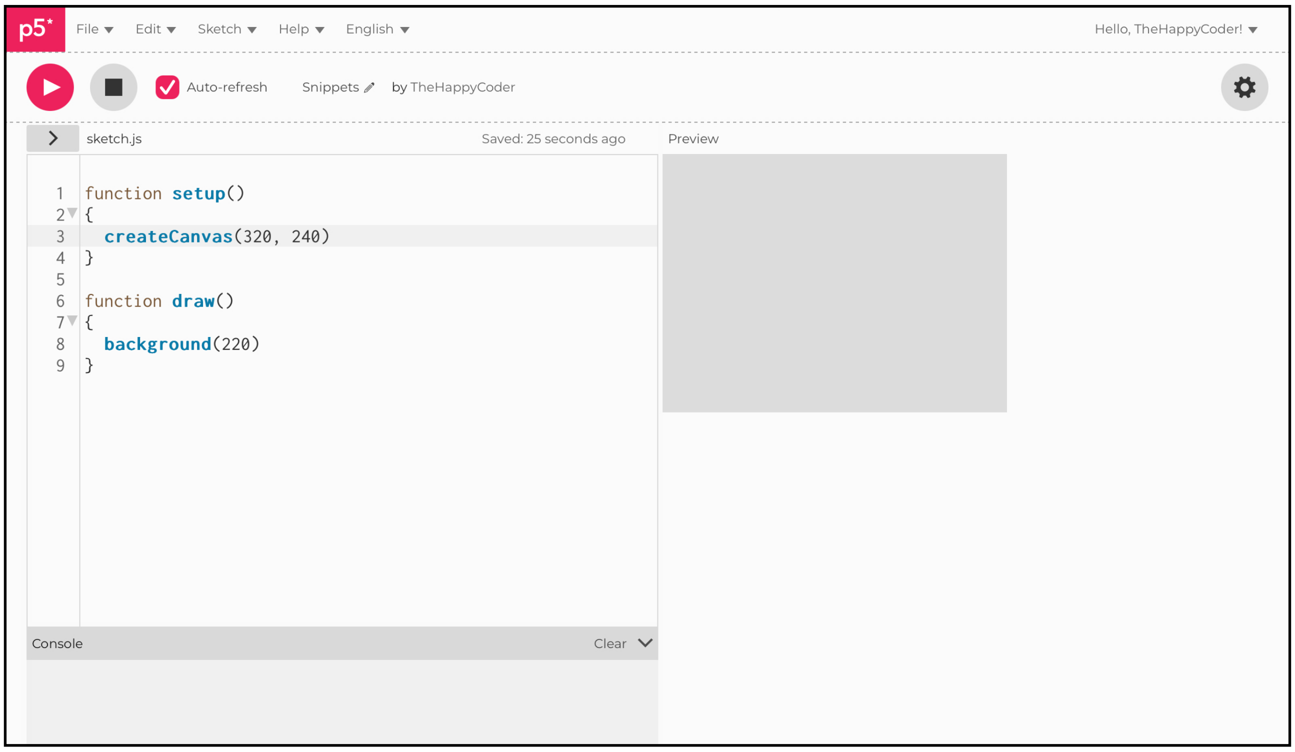
Sketch A5.14 new starting sketch

❗ starting a new sketch

We will have a canvas size of 320x240 which is half the dimensions of the video image.

```
function setup()
{
  createCanvas(320, 240)
}

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

Figure A5.14



```
function setup()
{
  createCanvas(320, 240)
}

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

Sketch A5.15 pixel array formula

The function loadPixels() gets all the datapoints from all the pixels and puts them into a pixel array, every pixel has four values (as mentioned) but we want every fourth index where the index is just for referencing the next pixel not every element (of which there are four per pixel). Hence we jump every four elements in the pixel array.

We have a simple algorithm for getting every pixel which is every fourth element in the pixel array. We can then pick out the four elements from each pixel and change them (if we want to) followed by updating the new pixel array with updatePixels() function.

To go through all the pixels in the array we have a nested loop where we start with the first line of pixels and work our way down. So that is why we start the first nested loop with the y coordinate and then the x coordinates. For each y value we loop through all the x values before going onto the next y value (one line at a time).

```
let x
let y
let index


function setup()
{
  createCanvas(320, 240)
}


function draw()
{
  background(220)
  loadPixels()
  for (y = 0; y < height; y++)
  {
    for (x = 0; x < width; x++)
    {
      index = (x + (y * width)) * 4
    }
  }
  updatePixels()
}
```

🗒️ Notes

Nothing changes with the canvas, you should get the grey background. The formula makes more sense when you plug in some values for `y` and for `x`. All you are doing is giving the index value to every fourth element in the array. Next we will break down the four elements in a pixel further to components of `red`, `green`, `blue` and `alpha`.

🌻 Challenge

Put in the values for `y = 0`, `x = 0`, then `x = 1`, `x = 2` and so on and you will, hopefully, see the logic of this algorithm.

🛠️ Code Explanation

`loadPixels()`	Loads all the pixels into an array
`index = (x + (y * width)) * 4`	A simple formula for working through the pixel array four at a time
`updatePixels()`	Updates the array and returns them to the canvas

Sketch A5.16 changing the pixels

Here we can give each pixel element, the red, the green, the blue, and the alpha new values. We have a grey canvas where the red, green and blue all have the same value of 220. We usually just have the one value for greyscale but in reality all three are the same, the alpha is 255 by default.

This may seem a bit confusing but in the pixel array which is has all the reds, greens, blues and alphas for all the pixels. The index we have created is for the actual pixel, so index + 0 is the first element which is red, index + 1 is the second which is green, index + 2 is the third element which is blue and index + 3 is the fourth element which is the alpha. Here we are going to give you the values of the grey scale and then next change them to an orange colour.

❗ I have added an extra line of code pixelDensity(1) because there are extra elements in the pixel array if the display is of a higher density and you get funny results. You can remove it if it causes any problems.

```
let x
let y
let index

function setup()
{
  createCanvas(320, 240)
  pixelDensity(1)
}

function draw()
{
  background(220)
  loadPixels()
  for (y = 0; y < height; y++)
  {
    for (x = 0; x < width; x++)
    {
      index = (x + (y * width)) * 4
      pixels[index + 0] = 220
      pixels[index + 1] = 220
      pixels[index + 2] = 220
      pixels[index + 3] = 255
    }
  }
  updatePixels()
}
```

🗒️ Notes

At present you simply get the same grey canvas

🛠️ Code Explanation

`pixels[index + 0] = 220`	The red element of the pixel value
`pixels[index + 1] = 220`	The green element of the pixel value
`pixels[index + 2] = 220`	The blue element of the pixel value
`pixels[index + 3] = 255`	The alpha element of the pixel value

Sketch A5.17 orange pixels

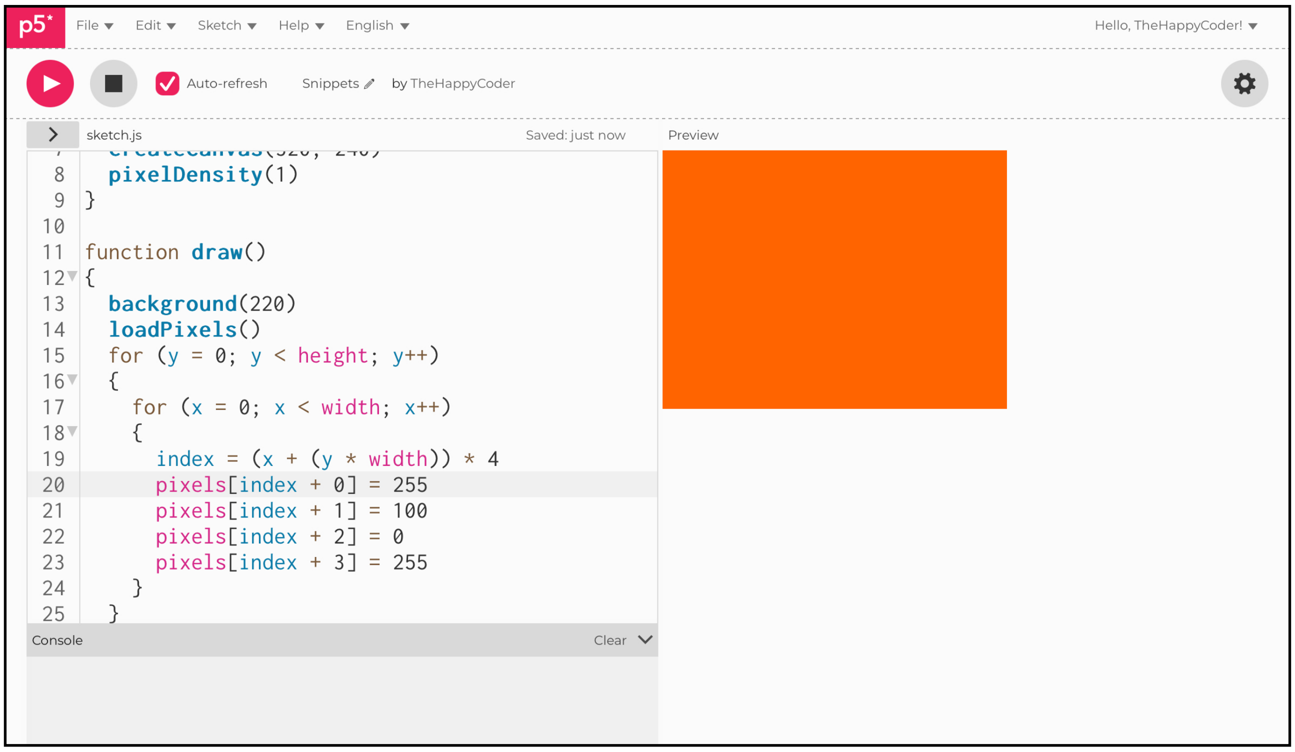
Now we can play around with the pixel element values.

```
let x
let y
let index

function setup()
{
  createCanvas(320, 240)
  pixelDensity(1)
}

function draw()
{
  background(220)
  loadPixels()
  for (y = 0; y < height; y++)
  {
    for (x = 0; x < width; x++)
    {
      index = (x + (y * width)) * 4
      pixels[index + 0] = 255
      pixels[index + 1] = 100
      pixels[index + 2] = 0
      pixels[index + 3] = 255
    }
  }
  updatePixels()
}
```

🗒️ Notes
We now have a nice bright orange canvas

🌻 Challenge
Try other combinations of colours and alpha

Figure A5.17



```
  pixelDensity(1)
}

function draw()
{
  background(220)
  loadPixels()
  for (y = 0; y < height; y++)
  {
    for (x = 0; x < width; x++)
    {
      index = (x + (y * width)) * 4
      pixels[index + 0] = 255
      pixels[index + 1] = 100
      pixels[index + 2] = 0
      pixels[index + 3] = 255
    }
  }
```

Sketch A5.18 return of the video

Adding in a lot of familiar code to return the video to the canvas, we introduce the `video.size()` function in `setup()` and a new pixel array for the video. The `loadPixels()` function takes the pixels from the canvas (not the video). The pixel array is updated with the video pixels array through the nested loop and then updates the pixels in the canvas.

```
let video
let x
let y
let index

function setup()
{
  createCanvas(320, 240)
  pixelDensity(1)
  video = createCapture(VIDEO, {flipped: true})
  video.size(320, 240)
  video.hide()
}

function draw()
{
  background(220)
  video.loadPixels()
  loadPixels()
  for(y = 0; y < height; y++)
  {
    for(x = 0; x < width; x++)
    {
      index = (x + (y * width)) * 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()
}
```

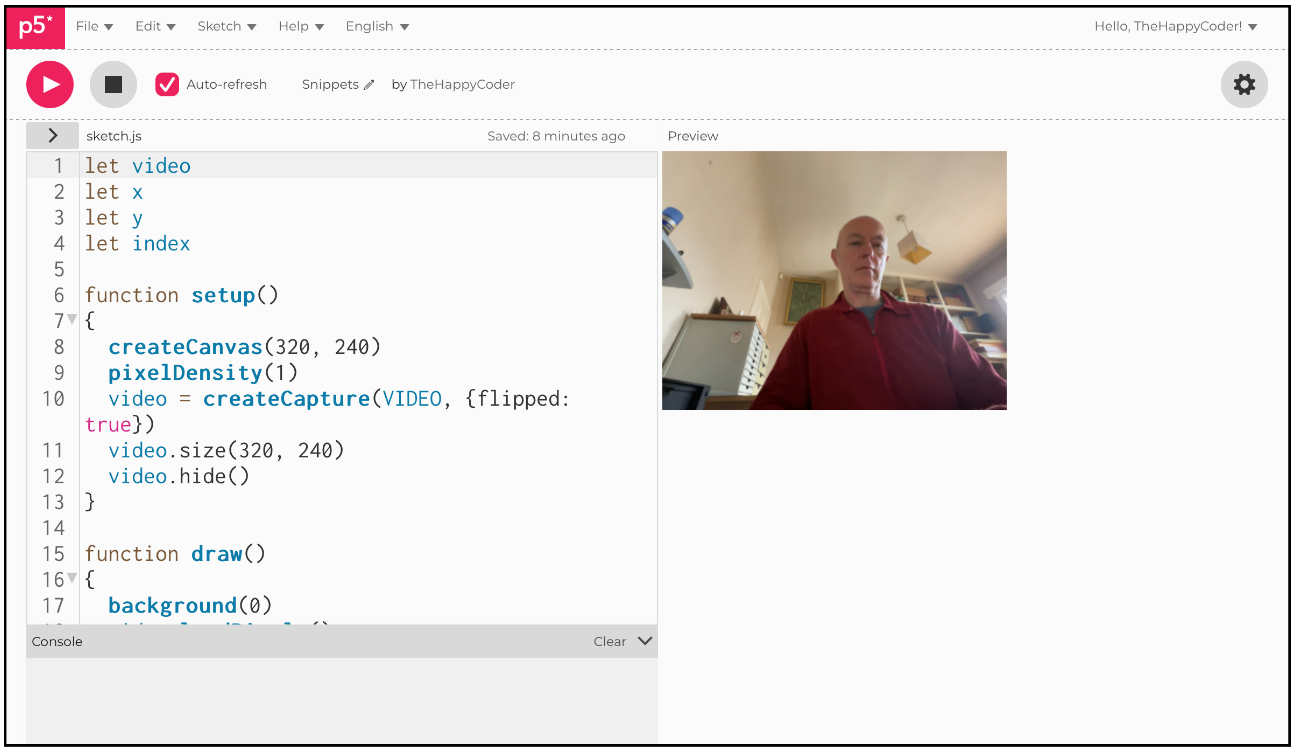
🗒️ Notes

We now have an image drawn onto the canvas. This means we can manipulate the image.

🛠️ Code Explanation

`video.size(320, 240)`	Alter the size of the video image
`video.loadPixels()`	Loading all the video pixels into a new array
`pixels[index + 0] = video.pixels[index + 0]`	Changing the pixel array with the new pixel array values from the video

Figure A5.18



```
let video
let x
let y
let index

function setup()
{
  createCanvas(320, 240)
  pixelDensity(1)
  video = createCapture(VIDEO, {flipped:
true})
  video.size(320, 240)
  video.hide()
}

function draw()
{
  background(0)
```

Sketch A5.19 pixelating the image

Another useful technique is to pixelate the image, we will use this in one of the units later. All the changes are highlighted in blue, it looks a lot but they are mostly just new variables. We change the size of the video by 1/16th. Then we scale it back up to fill the canvas. The pixels are drawn from the new video size which is a lot smaller and if you could see it, it would be quite blurry.

❗ Don't forget to remove the `updatePixels()` near the end of the code

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

function setup()
{
  createCanvas(320, 240)
  pixelDensity(1)
  video = createCapture(VIDEO, {flipped: true})
  video.size(width / vScale, height / vScale)
  video.hide()
}

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

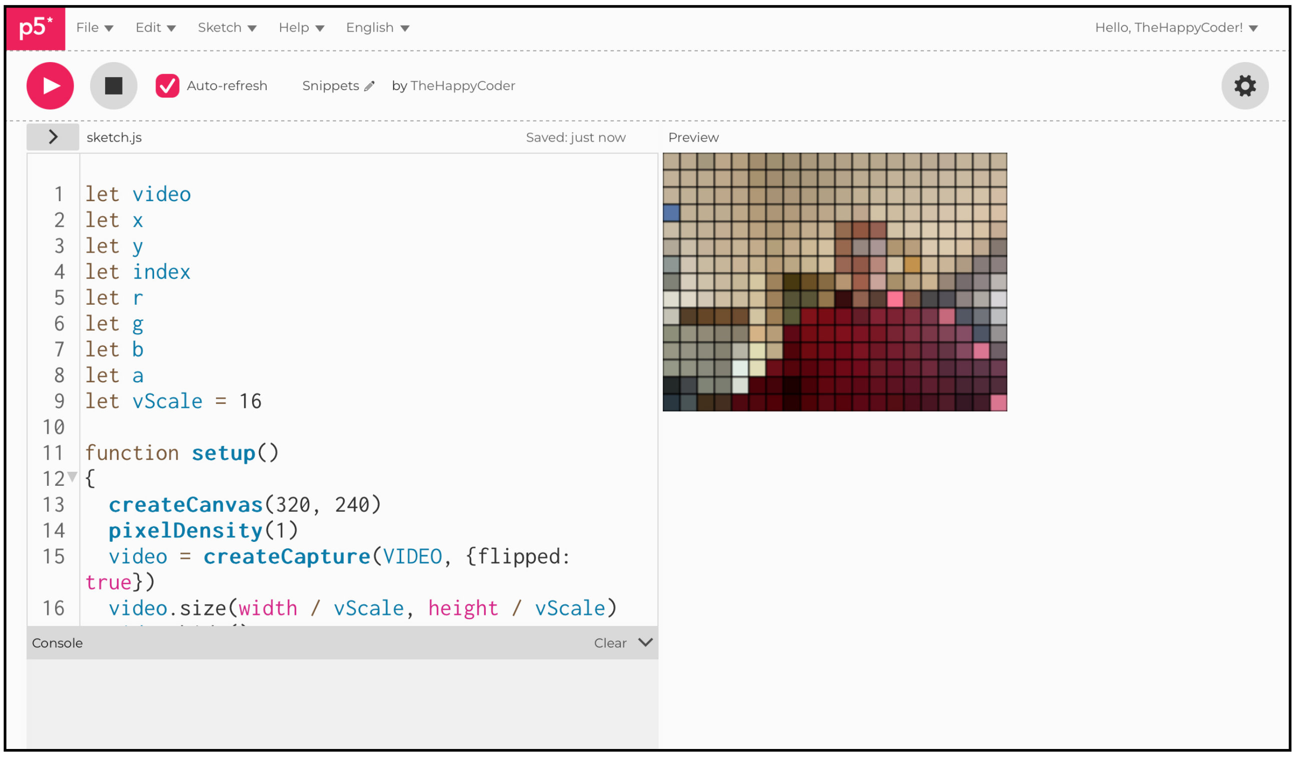
🗒️ Notes

This takes the original video image, scans each pixel for the `red`, `green`, `blue`, `alpha` and then scales it up to fill the canvas. A square is then filled with the colour of the original image. We reduce the image by vScale, analyse the image and then scale it back up to full canvas size.

🌻 Challenges

1. Change the vScale (you may get an error message)
2. Change the shape to a circle

Figure A5.19



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

function setup()
{
  createCanvas(320, 240)
  pixelDensity(1)
  video = createCapture(VIDEO, {flipped:
true})
  video.size(width / vScale, height / vScale)
```

Sketch A5.20 brightness grey scale

We can now take the average of the three colours in each pixel and call that value bright. We halve the size of the squares. I have added floor() when calculating the video size as it kept throwing up errors at certain vScale values. We also removed the lines round the boxes.

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

function setup()
{
  createCanvas(320, 240)
  pixelDensity(1)
  video = createCapture(VIDEO, {flipped: true})
  video.size(floor(width/vScale), floor(height/vScale))
  video.hide()
  noStroke()
}

function draw()
{
  background(0)
  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]
      bright = (r + g + b)/3
      fill(bright)
      square(x * vScale, y * vScale, vScale)
    }
  }
}
```

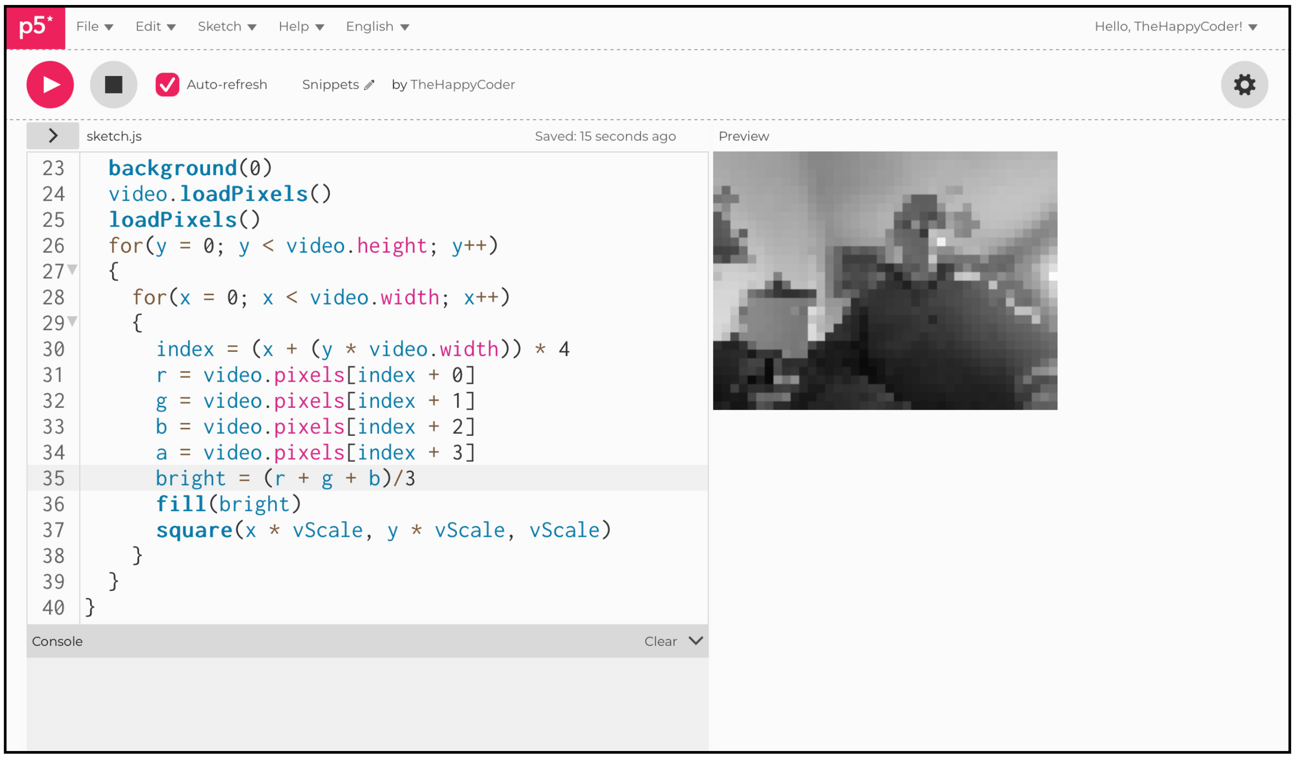
🗒️ Notes

Now we have added the video in as before but we have added the value of the red, green and blue and divided by three to get the average brightness. This is not the same as the alpha (which we have ignored).

🛠️ Code Explanation

`video.size(floor(width/vScale), floor(height/vScale))`	Take the floor value for the dimensions of the video size
`bright = (r + g + b)/3`	Average (bright) the colours in each pixel
`fill(bright)`	Fill that square with that average, bright

Figure A5.20



```
background(0)
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]
    bright = (r + g + b)/3
    fill(bright)
    square(x * vScale, y * vScale, vScale)
  }
}
}
```