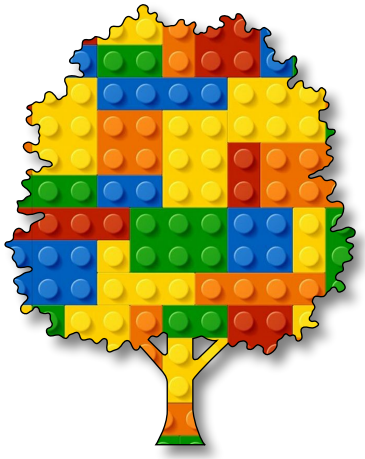
The Joy of Coding
Module D
Unit #6
Pixel arrays



Content

Module D Unit #6 pixel arrays

Introduction to pixel arrays

A canvas is a grid of pixels, a `pixel` array. Each with their own colour. Each pixel is made up of four channels, a `red`, a `green`, a `blue` and an `alpha`. We use this when we are colouring in our shapes in `RGB` mode.

We can get all the pixel information and also alter it in our code. For that we use two functions `loadPixels()` and `updatePixels()`, there is still quite a bit happening between these two functions.

We have a grid where we can allocate an (`x`, `y`) co-ordinate for each pixel. We use nested `for()` loops to scan all the pixels and update them.

If you are using a mac or a Retina display you will need to incorporate a function called `pixelDensity()` otherwise weird things happen.

We need to get our head round how the pixel array is arranged otherwise it will see a bit confusing. The array holds the four values for each pixel. One pixel may have something like this:

```
let pixel = [132, 231, 5, 255]
```

Where `132` is the red values, `231` is the green value, `5` is the blue value and `255` is the alpha value. If we have two pixels then the array looks like this:

```
let pixel = [132, 231, 5, 255, 65, 21, 200, 157]
```

The last four elements of the array are the red value (`65`), green value (`21`), blue value (`200`) and the alpha (`157`) of the second pixel and so on. So the pixel array holds information about each pixel in blocks of four, hence why we have to leap over each block in the loops.

So if you have a canvas of (`400`, `400`) with each pixel having 4 channels then the length of the pixel array is `400 x 400 x 4 = 640,000` elements in the array even though there are only `160,000` pixels.

We have another challenge to get the index value for a particular pixel. To do this we need a simple formula. The first row is straight forward. We just add `x` and `y` where `y` is zero. However when `y = 1` for the second

row `x` has already reached `400` (canvas width). So now the formula is `x` + (`y` times `width`). The code will look like thus:

```
let index[i] = x + (y * width)
```

This means that the index value for the pixel on the second row is:

```
0 + (1 * 400) = 400
```

This is correct and the next one is:

```
1 + (1 * 400) = 401
```

...and so on. However when we are looping through to get a particular value from the array, for example the red value for a pixel, we need to jump 4 elements in the array so to get all the red values, or green, or blue or alpha we now use the formula below:

```
let index[i] = x + (y * width) * 4
```

This may seem a bit laboured but hopefully will help as we work through this shortly. Just remember what the array actually represents (one long list of values for every channel and every pixel) and how you are accessing with the nested `for()` loops. It is all very logical.

Sketch D6.1 starting sketch

Our starting Sketch

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

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

Sketch D6.2 loading the pixels

Introducing the two functions. The firs gets all the pixels on the canvas with an array of 640,000 elements. This does nothing except store that information. Next we will access a pixel and do something to it.

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

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

🗒️ Notes

All we have done is load the pixel array and updated the array which means we have done nothing

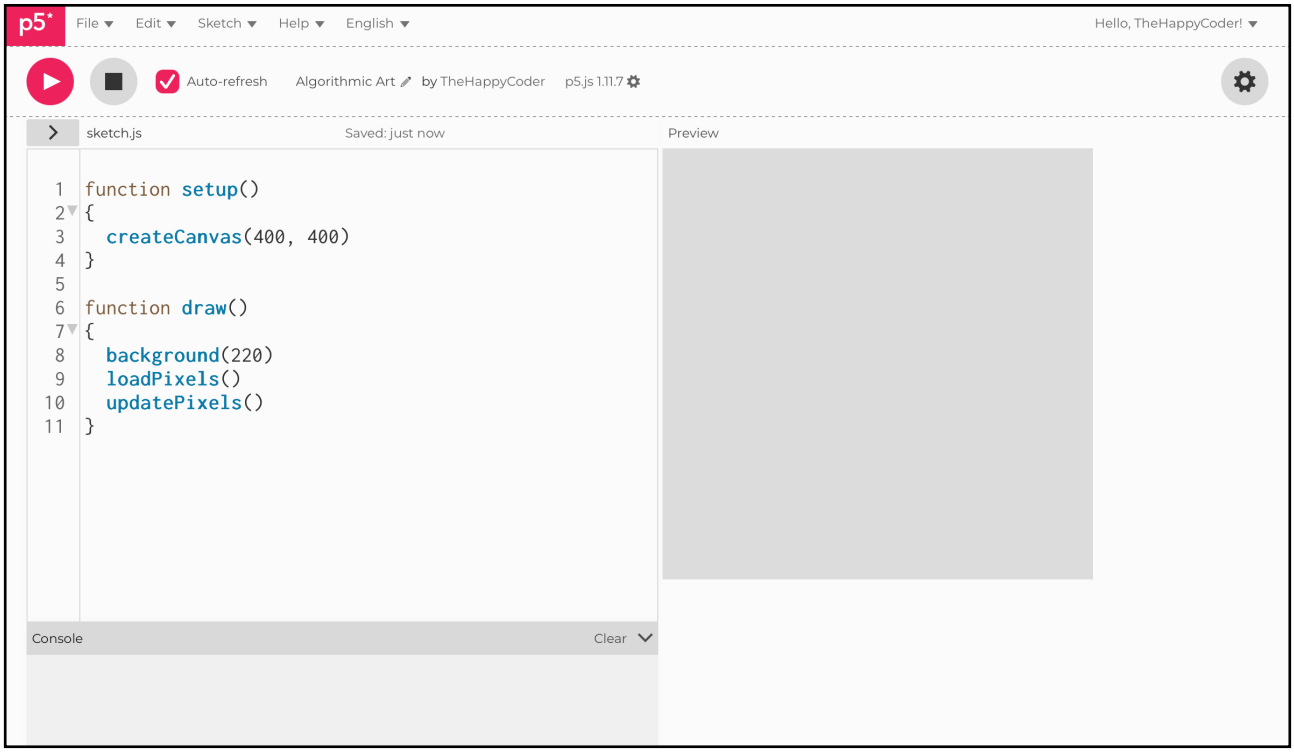
🌻 Challenge

xxxxx

🛠️ Code Explanation

`loadPixels()`	Loads all the pixels (elements) in the canvas
`updatePixels()`	This would update any changes to the pixel arrray

Figure D6.2


p5*
File
Edit
Sketch
Help
English
Hello, TheHappyCoder!
Auto-refresh
Algorithmic Art
by TheHappyCoder
p5.js 1.11.7
sketch.js
Saved: just now
Preview
function setup()
{
createCanvas(400, 400)
}
function draw()
{
background(220)
loadPixels()
updatePixels()
}
Console
Clear

Sketch D6.3 change the pixel

To see what we are doing here we will change the first pixel because we know where each channel is in the array, `index[0]` is red, `index[1]` is green, `index[2]` is blue, and `index[3]` is the alpha. What we have done here is change the very first pixel, making it red. You may struggle to see this but it is there if you zoom in (see Fig.1).

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

function draw()
{
  background(220)
  loadPixels()
  pixels[0] = 255
  pixels[1] = 0
  pixels[2] = 0
  pixels[3] = 255
  updatePixels()
}
```

Notes

We loaded and changed the first four elements of the pixel array and then updated the array to draw the results on canvas

Challenges

1. Make some other changes, play with the values
2. Could you change one half way across the canvas

🛠️ Code Explanation

`pixels[0] = 255`	The very first element is the red component giving it 255
`pixels[1] = 0`	The second element is the green component value of 0
`pixels[2] = 0`	The third element is the blue which is also 0
`pixels[3] = 255`	The fourth is the alpha of a value 255

Figure D6.3

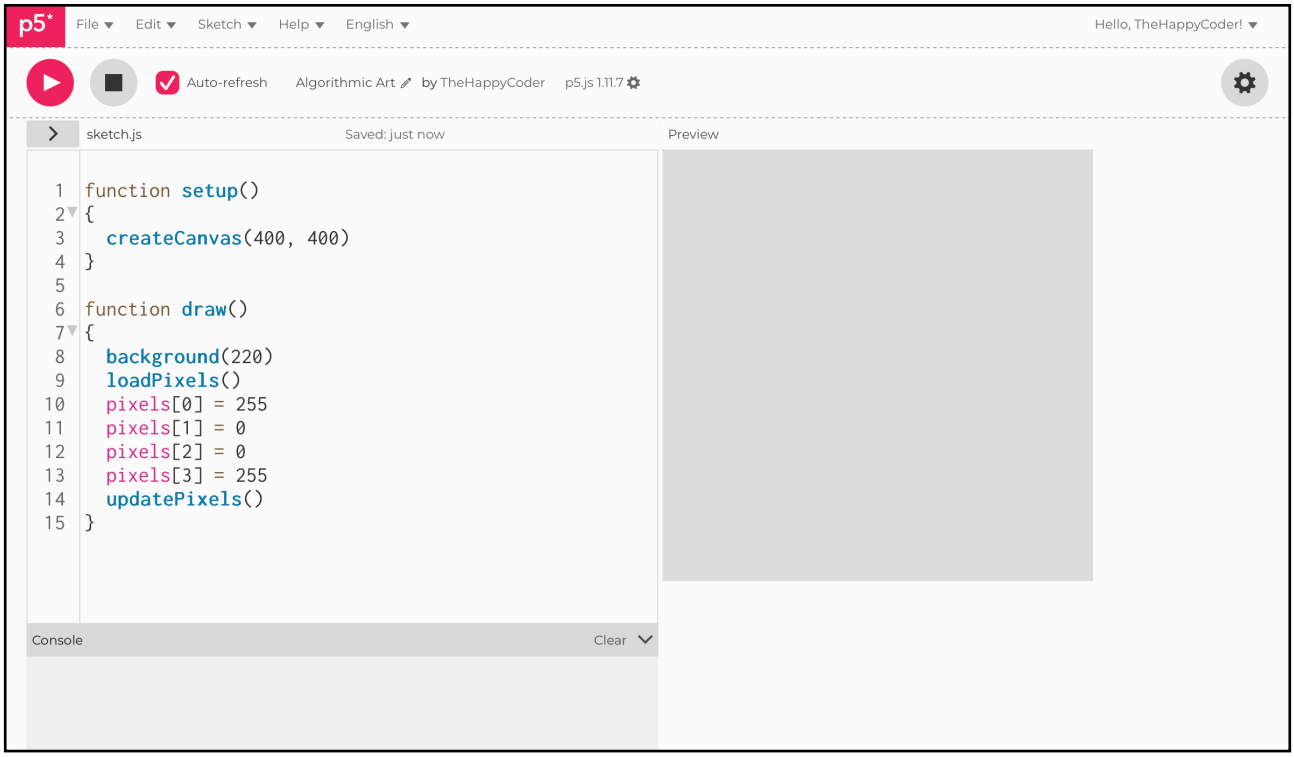
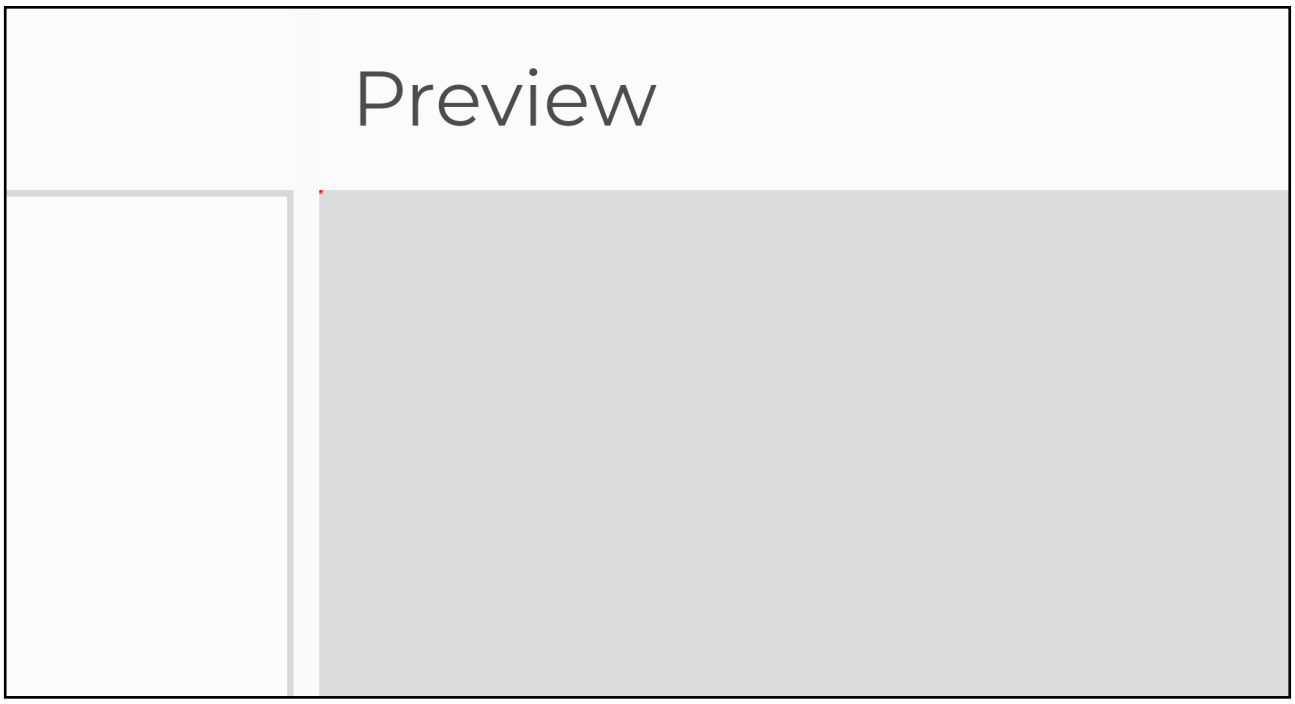

p5*
File
Edit
Sketch
Help
English
Hello, TheHappyCoder!
Auto-refresh
Algorithmic Art
by TheHappyCoder
p5.js 1.11.7
sketch.js
Saved: just now
Preview
function setup()
{
createCanvas(400, 400)
}
function draw()
{
background(220)
loadPixels()
pixels[0] = 255
pixels[1] = 0
pixels[2] = 0
pixels[3] = 255
updatePixels()
}
Console
Clear

Figure 1: a red pixel



Sketch D6.4 a row of pixels

In this instance we are just going to make all the pixels in the first row red. We need to jump every 4 elements in the array and so we need to multiply the width by 4 (there are four times as many elements as pixels). Also I needed to bring in the pixelDensity() otherwise it may not work.

```
function setup()
{
  createCanvas(400, 400)
  pixelDensity(1)
}

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

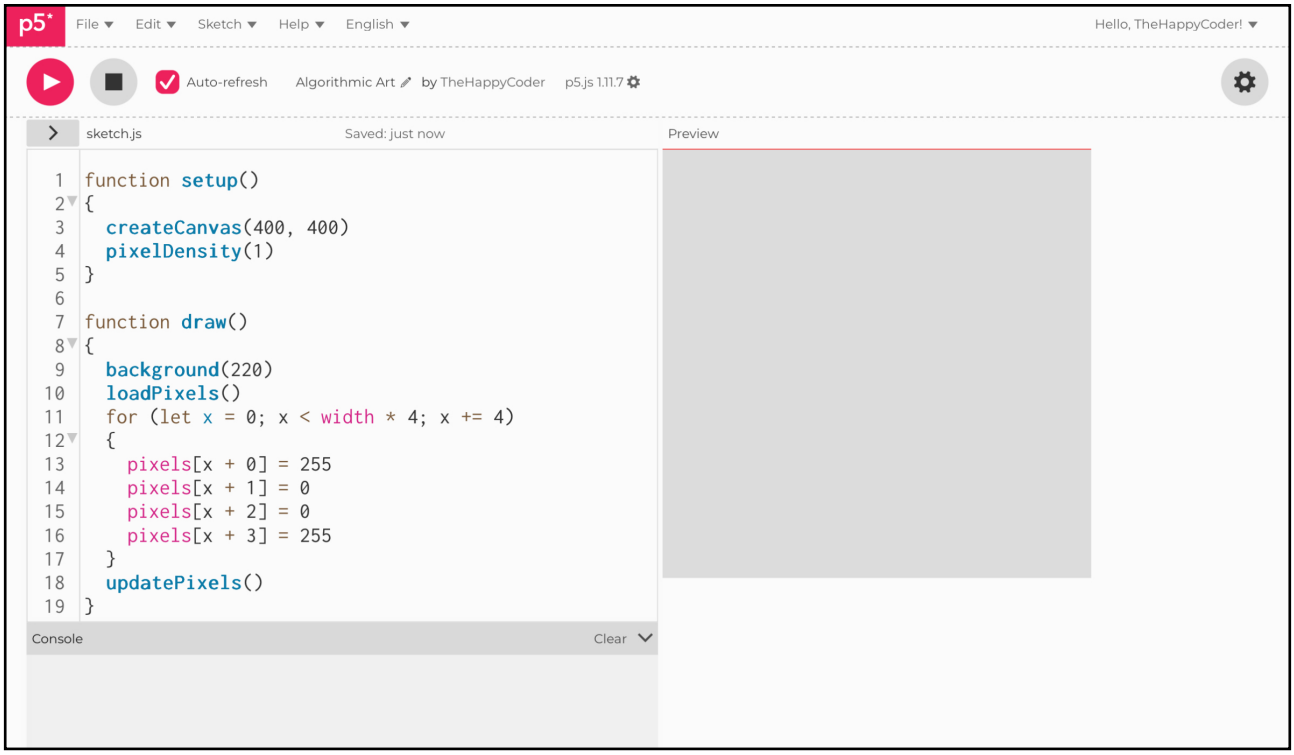
🗒️ Notes

You can see what we have done, created a red line, we multiply by four as the width is just 400

🌻 Challenge

Try it without multiplying by four

Figure D6.4


p5*
File
Edit
Sketch
Help
English
Hello, TheHappyCoder!
Auto-refresh
Algorithmic Art
by TheHappyCoder
p5.js 1.11.7
sketch.js
Saved: just now
Preview
1 function setup()
2 {
3 createCanvas(400, 400)
4 pixelDensity(1)
5 }
6
7 function draw()
8 {
9 background(220)
10 loadPixels()
11 for (let x = 0; x < width * 4; x += 4)
12 {
13 pixels[x + 0] = 255
14 pixels[x + 1] = 0
15 pixels[x + 2] = 0
16 pixels[x + 3] = 255
17 }
18 updatePixels()
19 }
Console
Clear

Sketch D6.5 every pixel array

To colour every pixel on the canvas red we need to build the nested loop. We start with the y value as the outside of the nested loops because we want to move from top to bottom and on each row left to right. We add in the formula for the index.

```
function setup()
{
  createCanvas(400, 400)
  pixelDensity(1)
}

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

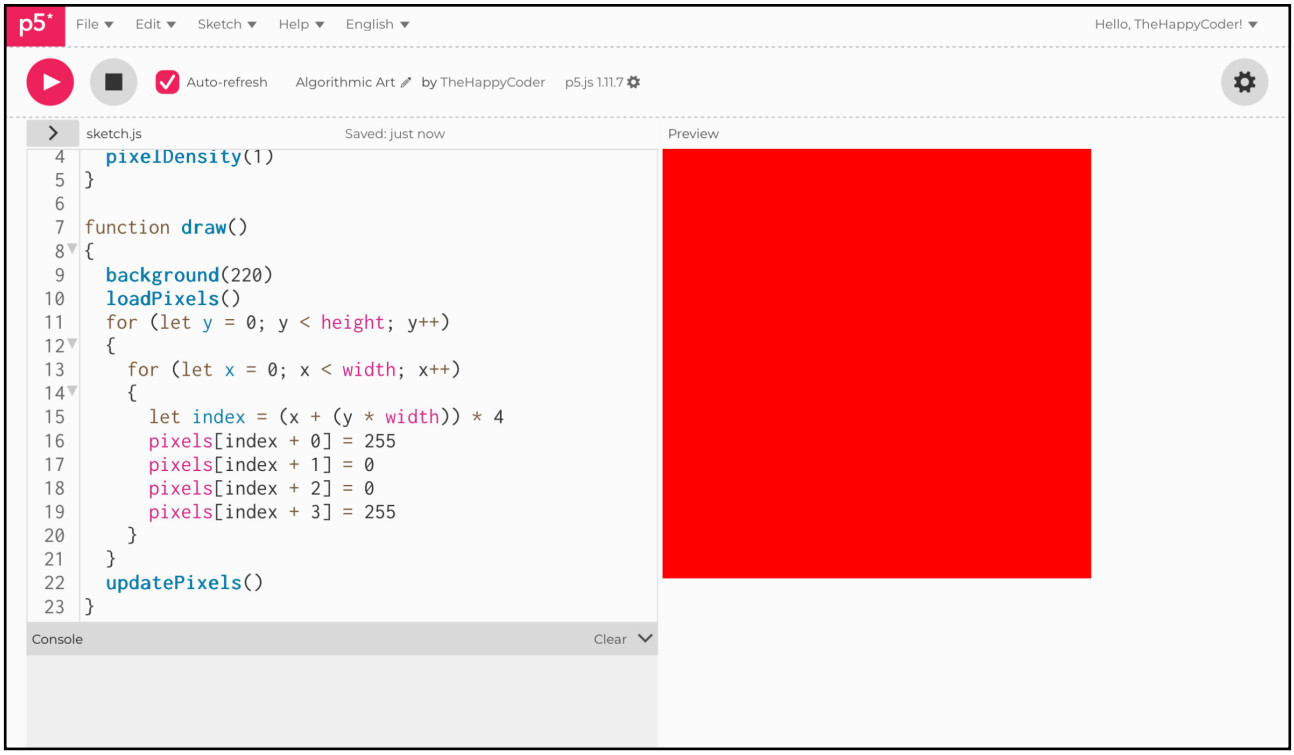
Notes

A nested loop covers the entire canvas

🌻 Challenge

Try going from left to right rather than top to bottom

Figure D6.5


p5*
File
Edit
Sketch
Help
English
Hello, TheHappyCoder!
Auto-refresh
Algorithmic Art
by TheHappyCoder
p5.js 1.11.7
sketch.js
Saved: just now
Preview
pixelDensity(1)
}
function draw()
{
background(220)
loadPixels()
for (let y = 0; y < height; y++)
{
for (let x = 0; x < width; x++)
{
let index = (x + (y * width)) * 4
pixels[index + 0] = 255
pixels[index + 1] = 0
pixels[index + 2] = 0
pixels[index + 3] = 255
}
}
updatePixels()
}
Console
Clear

Sketch D6.6 the pixel values

We can play with these values and do interesting stuff. Play around with the variables yourself.

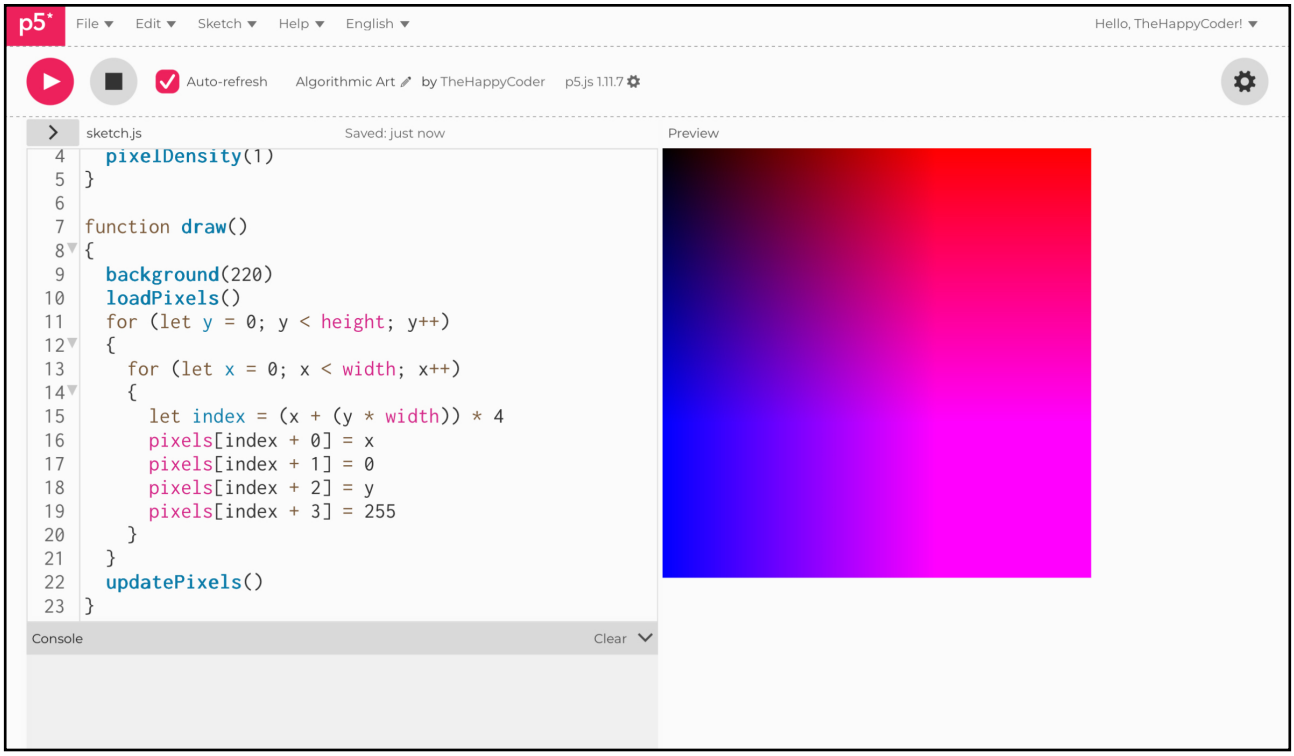
```
function setup()
{
  createCanvas(400, 400)
  pixelDensity(1)
}

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

Notes

You are linking the pixel element value to their position on the canvas

Figure D6.6


p5*
File
Edit
Sketch
Help
English
Hello, TheHappyCoder!
Auto-refresh
Algorithmic Art
by TheHappyCoder
p5.js 1.11.7
sketch.js
Saved: just now
Preview
pixelDensity(1)
}
function draw()
{
background(220)
loadPixels()
for (let y = 0; y < height; y++)
{
for (let x = 0; x < width; x++)
{
let index = (x + (y * width)) * 4
pixels[index + 0] = x
pixels[index + 1] = 0
pixels[index + 2] = y
pixels[index + 3] = 255
}
}
updatePixels()
}
Console
Clear

Sketch D6.7 another effect

And yet more playing incorporating the y value

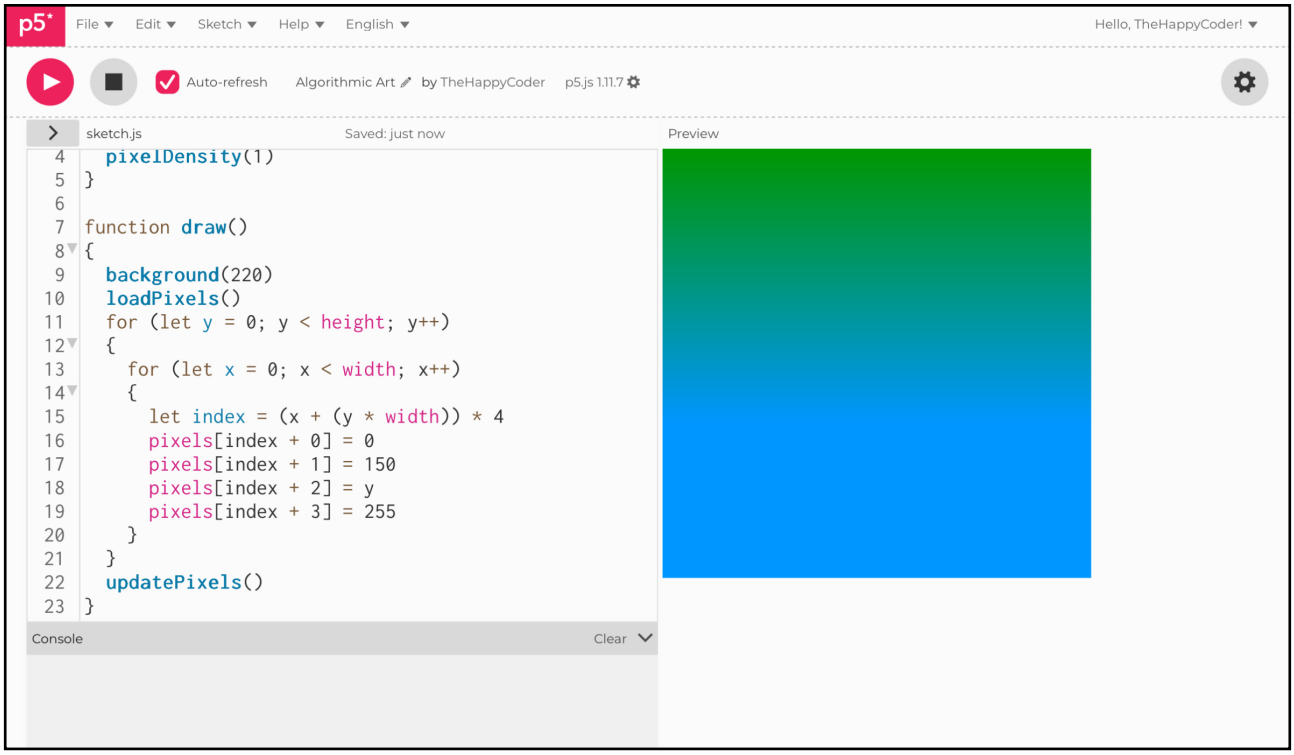
```
function setup()
{
  createCanvas(400, 400)
  pixelDensity(1)
}

function draw()
{
  background(220)
  loadPixels()
  for (let y = 0; y < height; y++)
  {
    for (let x = 0; x < width; x++)
    {
      let index = (x + (y * width)) * 4
      pixels[index + 0] = 0
      pixels[index + 1] = 150
      pixels[index + 2] = y
      pixels[index + 3] = 255
    }
  }
  updatePixels()
}
```

Challenge

Consider using mouseX and mouseY

Figure D6.7


p5*
File
Edit
Sketch
Help
English
Hello, TheHappyCoder!
Auto-refresh
Algorithmic Art
by TheHappyCoder
p5.js 1.11.7
sketch.js
Saved: just now
Preview
pixelDensity(1)
}
function draw()
{
background(220)
loadPixels()
for (let y = 0; y < height; y++)
{
for (let x = 0; x < width; x++)
{
let index = (x + (y * width)) * 4
pixels[index + 0] = 0
pixels[index + 1] = 150
pixels[index + 2] = y
pixels[index + 3] = 255
}
}
updatePixels()
}
Console
Clear