Algorithmic Art Workbook #11 Video and HTML



Table of Contents

MIT Licence 4
Workbook #11 Quick Content Summary 5
Module F Unit #1: keyboard and mouse 6
Sketch F1.1 KeyIsPressed 7
Sketch F1.2 Key 9
Sketch F1.3 keyPressed() 11
Sketch F1.4 keyReleased() 13
Sketch F1.5 KeyCode 15
Sketch F1.6 keyTyped() 16
Sketch F1.7 KeyIsDown() 17
Sketch F1.8 movedX 19
Sketch F1.9 winMouseX and pwinMouseX 21
Sketch F1.10 mouseIsPressed() 23
Sketch F1.11 mouseButton() 25
Sketch F1.12 mouseMoved() 26
Sketch F1.13 mouseDragged() 27
Sketch F1.14 mousePressed() 28
Sketch F1.15 mouseReleased() 29
Sketch F1.16 mouseClicked() 30
Sketch F1.17 doubleClicked() 31
Sketch F1.18 mouseWheel() 32
Sketch F1.19 easing 34
Module F Unit #2: HTML 37
Sketch F2.1 starting sketch 38
Sketch F2.2 creating html elements 39
Sketch F2.3 position 41
Sketch F2.4 starting sketch 43
Sketch F2.5 background colour 44
Sketch F2.6 adding a button 45
Sketch F2.7 the callback 47
Sketch F2.8 slider 49
Sketch F2.9 text box 51
Sketch F2.10 hovering over text 53
Sketch F2.11 hovering over the canvas 55
Introduction to CSS 57
Sketch F2.12 the index.html file 58
Sketch F2.13 adding a sketch 60
Sketch F2.14 CSS changed 62

Sketch F2.15 submit button 64
Sketch F2.16 adding the button 65
Sketch F2.17 input function 66
Sketch F2.18 have the question 68
Sketch F2.19 simple calculator 71
Sketch F2.20 slider colour circle 73
Sketch F2.21 slider rotate 76
Sketch F2.22 radio buttons colour 78
Module F Unit #3: video capture 81
Sketch F3.1 scale 82
Sketch F3.2 scale with mouse 84
Sketch F3.3 scale negatively 86
Sketch F3.4 video capture 88
Sketch F3.5 creating the canvas 89
Sketch F3.6 video on the canvas 90
Sketch F3.7 size and position 92
Sketch F3.8 hide the streaming video 94
Sketch F3.9 taking a selfie 96
Sketch F3.10 multiple selfies 98
Sketch F3.11 getting pixels 100
Sketch F3.12 returning the video 103
Sketch F3.13 pixelating the image 105
Sketch F3.14 making it grey scale 107
Sketch F3.15 brightness mirror 109
Sketch F3.16 threshold image 111
Sketch F3.17 optional threshold 114
Sketch F3.18 starting sketch 117
Sketch F3.19 array of particles 118
Sketch F3.20 the Particle class 119
Sketch F3.21 show something 120
Sketch F3.22 getting the pixels 122
Sketch F3.23 the 300 124
Sketch F3.24 randomise the position 126
Creating the mirror effect 129
Sketch F3.25 mirror the image #1 130
Sketch F3.26 mirror the image #2 131



MIT Licence

Copyright ©2026 Warren George

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NON-INFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

Brief summary:

Permissions (what you can do)

Commercial use
Modification
Distribution
Private use

Limitations (what is not covered)

Liability
Warranty



Workbook #11 Quick Content Summary

I will be honest, this workbook is a bit of a 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.x.

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 the Chrome browser.

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 F1.1a: no key pressed

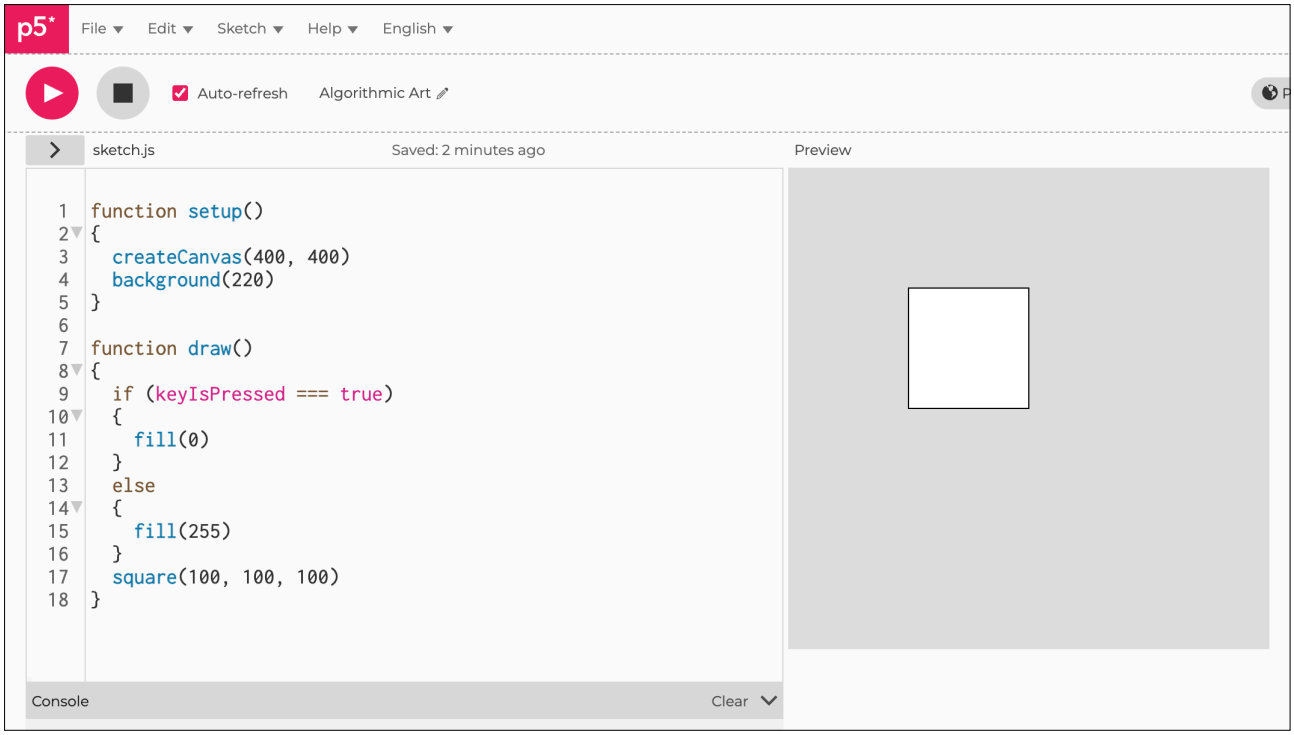


Figure F1.1b: key is pressed



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)
}
```

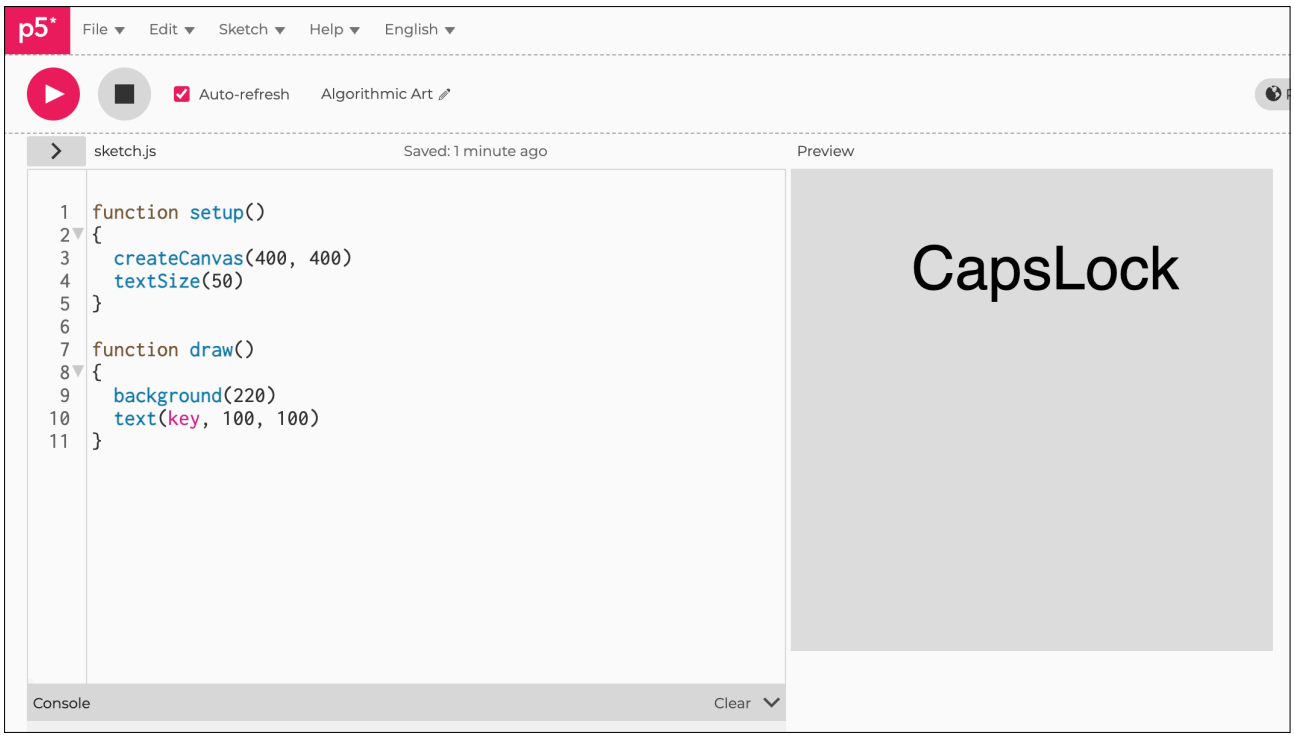
Notes

Showed the pressing of the Caps Lock key.

Code Explanation

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

Figure F1.2


p5*
File
Edit
Sketch
Help
English
Auto-refresh
Algorithmic Art
sketch.js
Saved: 1 minute ago
Preview
function setup()
{
createCanvas(400, 400)
textSize(50)
}
function draw()
{
background(220)
text(key, 100, 100)
}
CapsLock
Console
Clear

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)
}

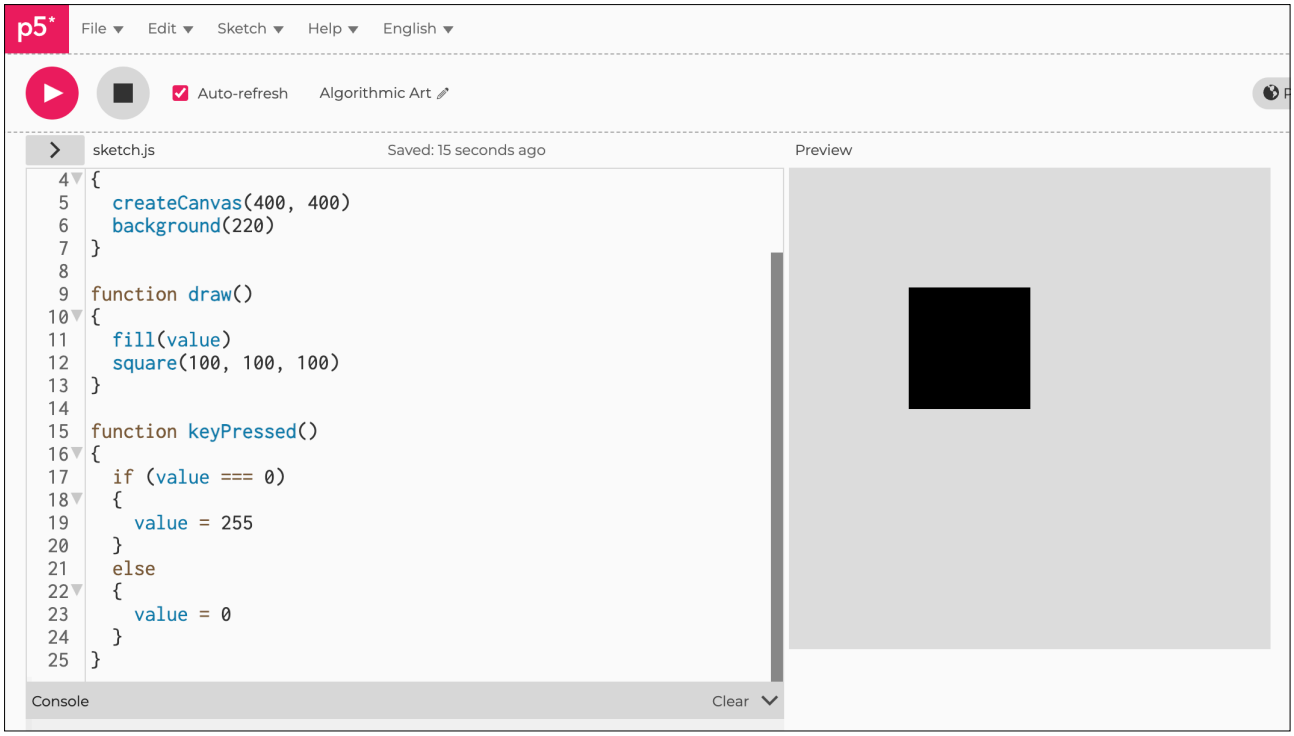
function draw()
{
  fill(value)
  square(100, 100, 100)
}

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


p5*
File
Edit
Sketch
Help
English
Auto-refresh
Algorithmic Art
sketch.js
Saved: 15 seconds ago
Preview
4 {
5 createCanvas(400, 400)
6 background(220)
7 }
8
9 function draw()
10 {
11 fill(value)
12 square(100, 100, 100)
13 }
14
15 function keyPressed()
16 {
17 if (value === 0)
18 {
19 value = 255
20 }
21 else
22 {
23 value = 0
24 }
25 }
Console
Clear

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
  }
}
```

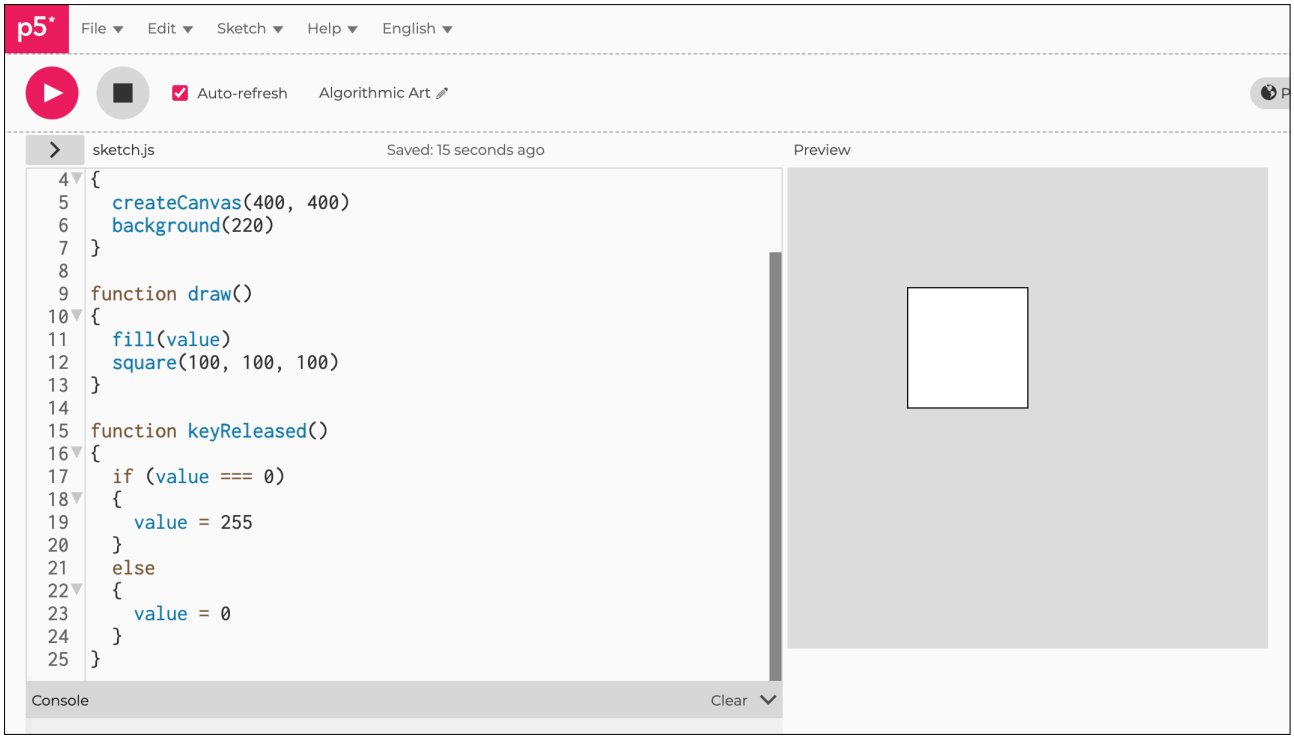
Notes

Only changes at the point of release.

Code Explanation

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

Figure F1.4


p5*
File
Edit
Sketch
Help
English
Auto-refresh
Algorithmic Art
sketch.js
Saved: 15 seconds ago
Preview
4 {
5 createCanvas(400, 400)
6 background(220)
7 }
8
9 function draw()
10 {
11 fill(value)
12 square(100, 100, 100)
13 }
14
15 function keyReleased()
16 {
17 if (value === 0)
18 {
19 value = 255
20 }
21 else
22 {
23 value = 0
24 }
25 }
Console
Clear



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 W 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 === 'w')
  {
    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)
}
```

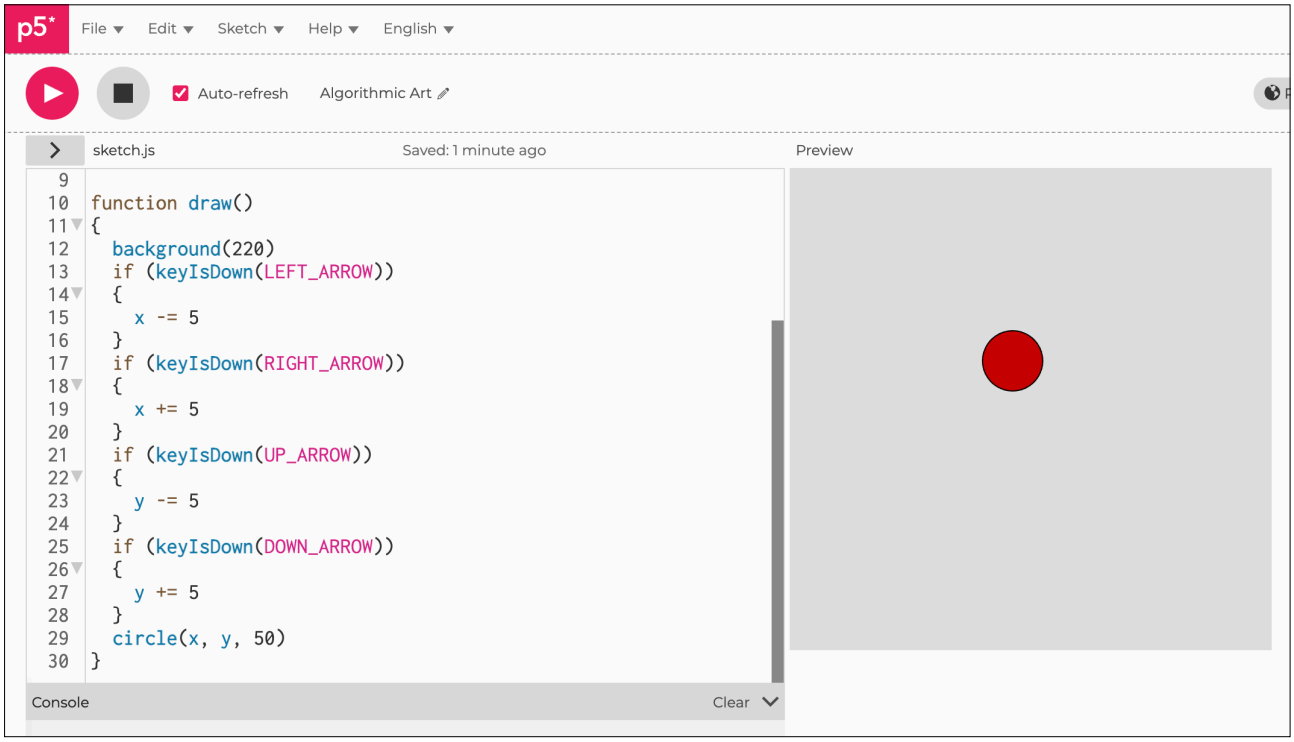
Notes

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

! 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)
}
```

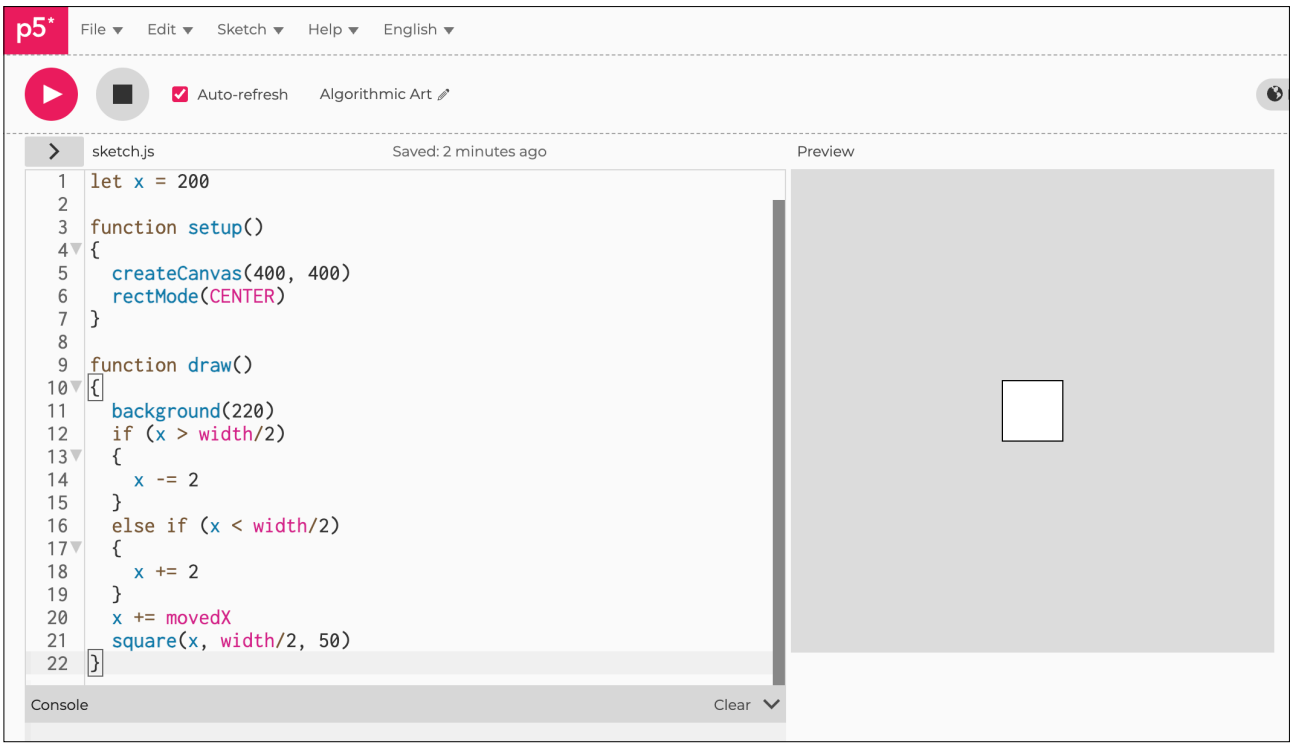
Notes

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


p5*
File
Edit
Sketch
Help
English
Auto-refresh
Algorithmic Art
sketch.js
Saved: 2 minutes ago
Preview
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)
}
Console
Clear

Sketch F1.9 winMouseX and pwinMouseX

! 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)
}
```

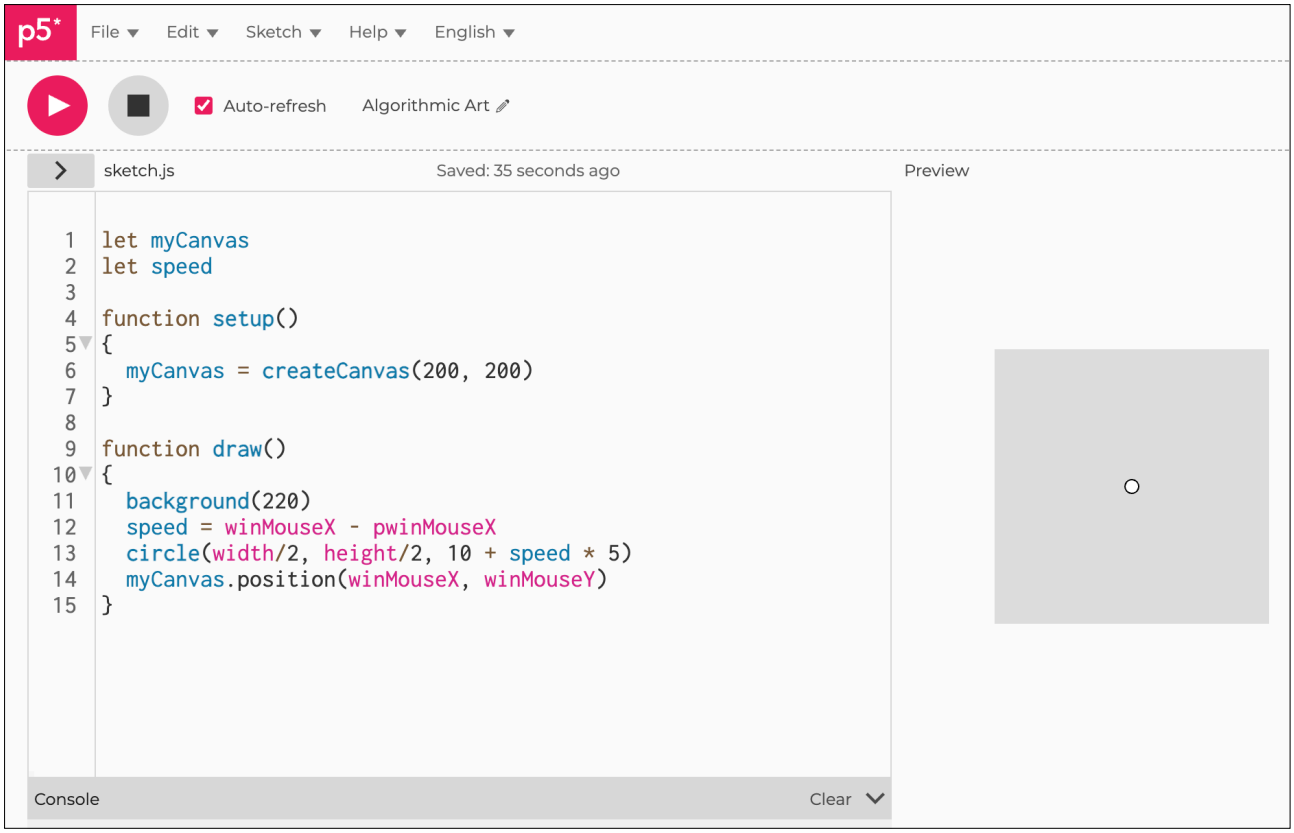
Notes

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


p5*
File
Edit
Sketch
Help
English
Auto-refresh
Algorithmic Art
sketch.js
Saved: 35 seconds ago
Preview
1 let myCanvas
2 let speed
3
4 function setup()
5 {
6 myCanvas = createCanvas(200, 200)
7 }
8
9 function draw()
10 {
11 background(220)
12 speed = winMouseX - pwinMouseX
13 circle(width/2, height/2, 10 + speed * 5)
14 myCanvas.position(winMouseX, winMouseY)
15 }
Console
Clear

Sketch F1.10 mouseIsPressed()

! New sketch.

Simple responds when any mouse button is pressed.

```
let value = 255

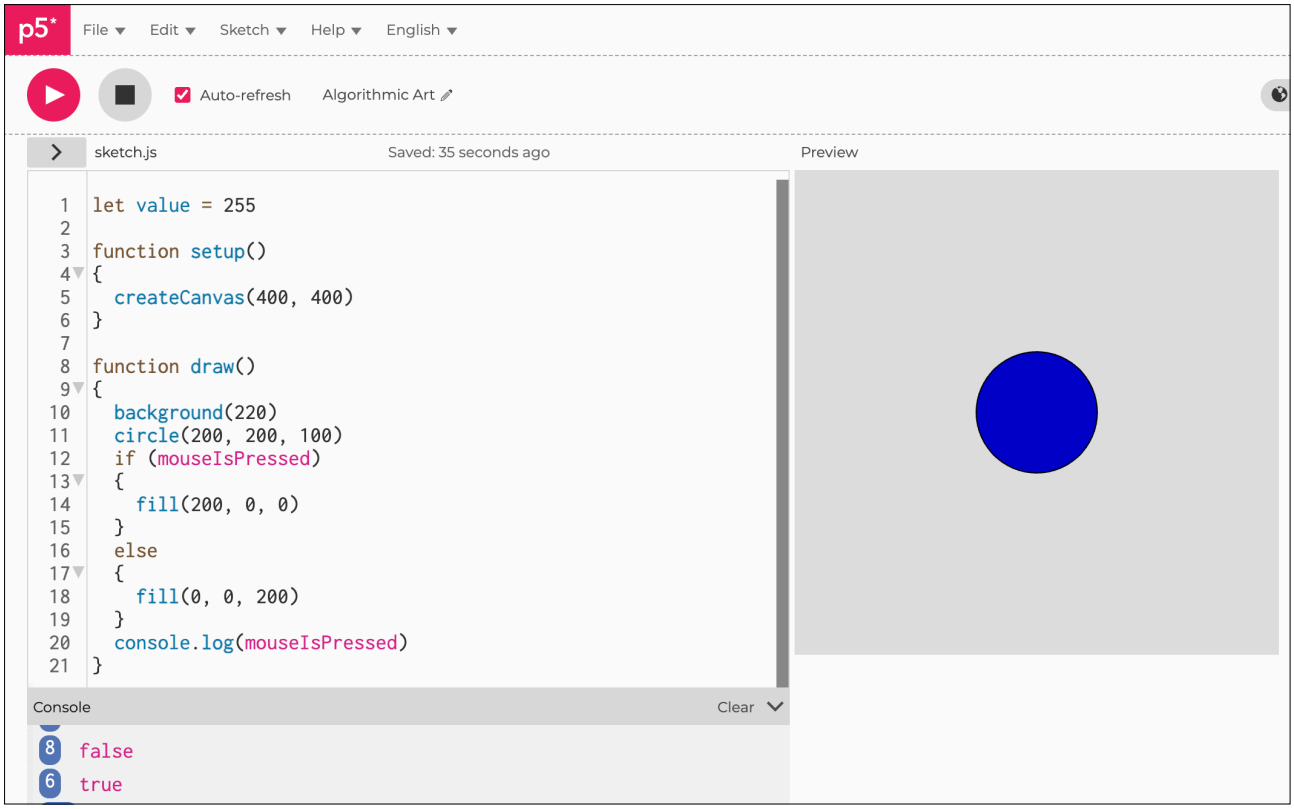
function setup()
{
  createCanvas(400, 400)
}

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


p5*
File
Edit
Sketch
Help
English
Auto-refresh
Algorithmic Art
sketch.js
Saved: 35 seconds ago
Preview
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)
}
Console
Clear
false
true

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)

}

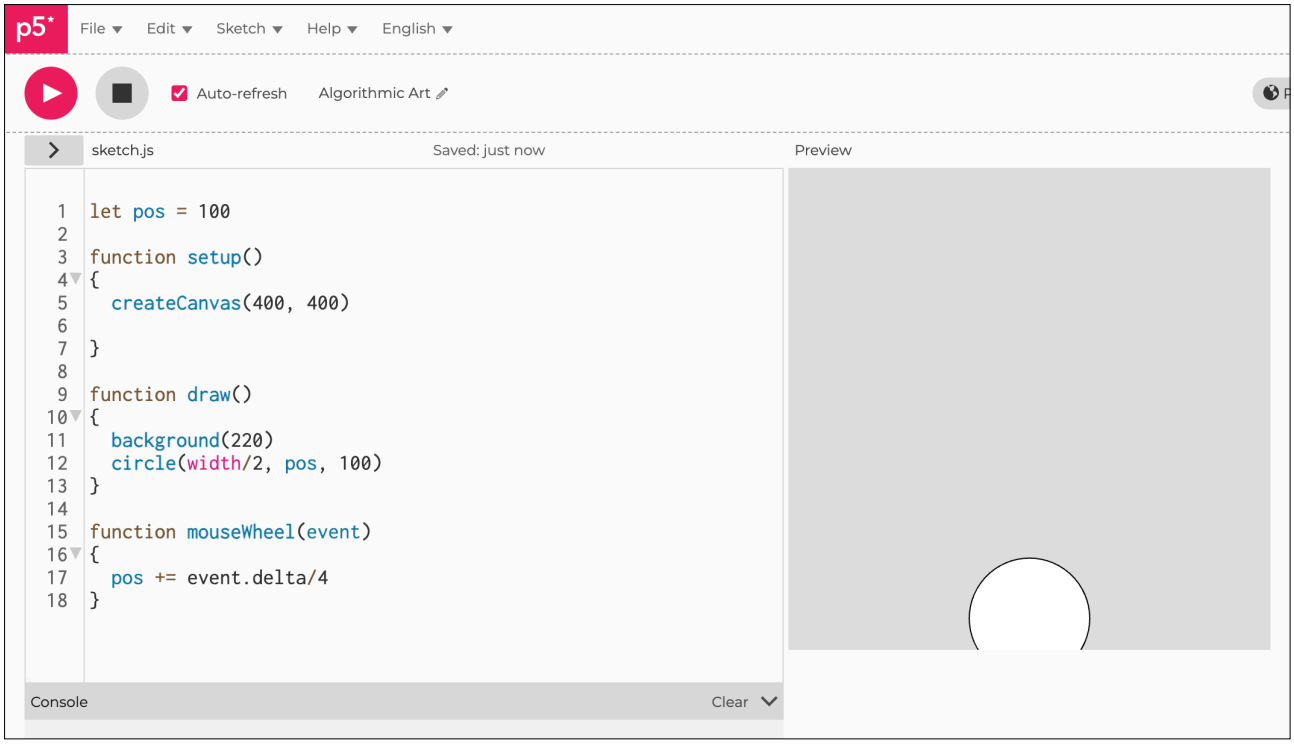
function draw()
{
  background(220)
  circle(width/2, pos, 100)
}

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


p5*
File
Edit
Sketch
Help
English
Auto-refresh
Algorithmic Art
sketch.js
Saved: just now
Preview
let pos = 100
function setup()
{
createCanvas(400, 400)
}
function draw()
{
background(220)
circle(width/2, pos, 100)
}
function mouseWheel(event)
{
pos += event.delta/4
}
Console
Clear

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)
}
```

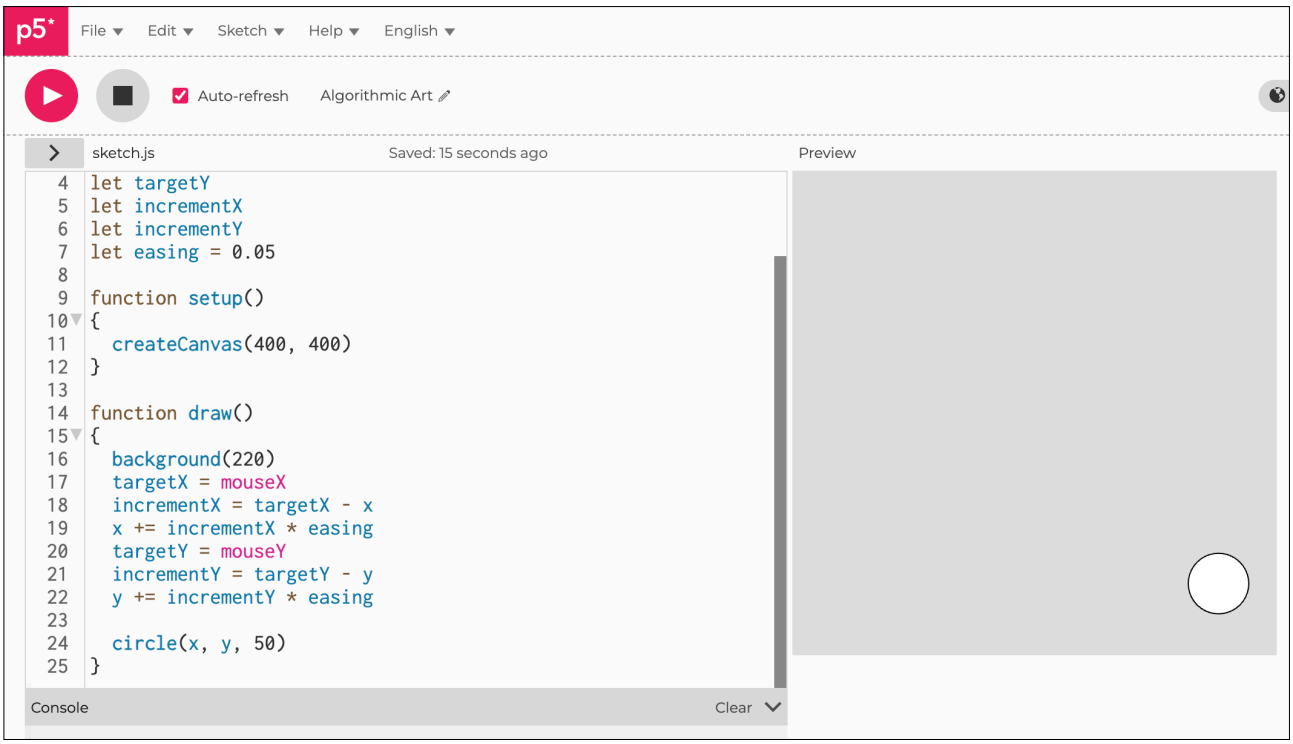
Notes

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


p5*
File
Edit
Sketch
Help
English
Auto-refresh
Algorithmic Art
sketch.js
Saved: 15 seconds ago
Preview
4 let targetY
5 let incrementX
6 let incrementY
7 let easing = 0.05
8
9 function setup()
10 {
11 createCanvas(400, 400)
12 }
13
14 function draw()
15 {
16 background(220)
17 targetX = mouseX
18 incrementX = targetX - x
19 x += incrementX * easing
20 targetY = mouseY
21 incrementY = targetY - y
22 y += incrementY * easing
23
24 circle(x, y, 50)
25 }
Console
Clear


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 give index.html its own heading.

index.html

You can access the list of files as shown previously. 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.

Challenge

Click on the style.css tab and have a peek.

Sketch F2.1 starting sketch

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

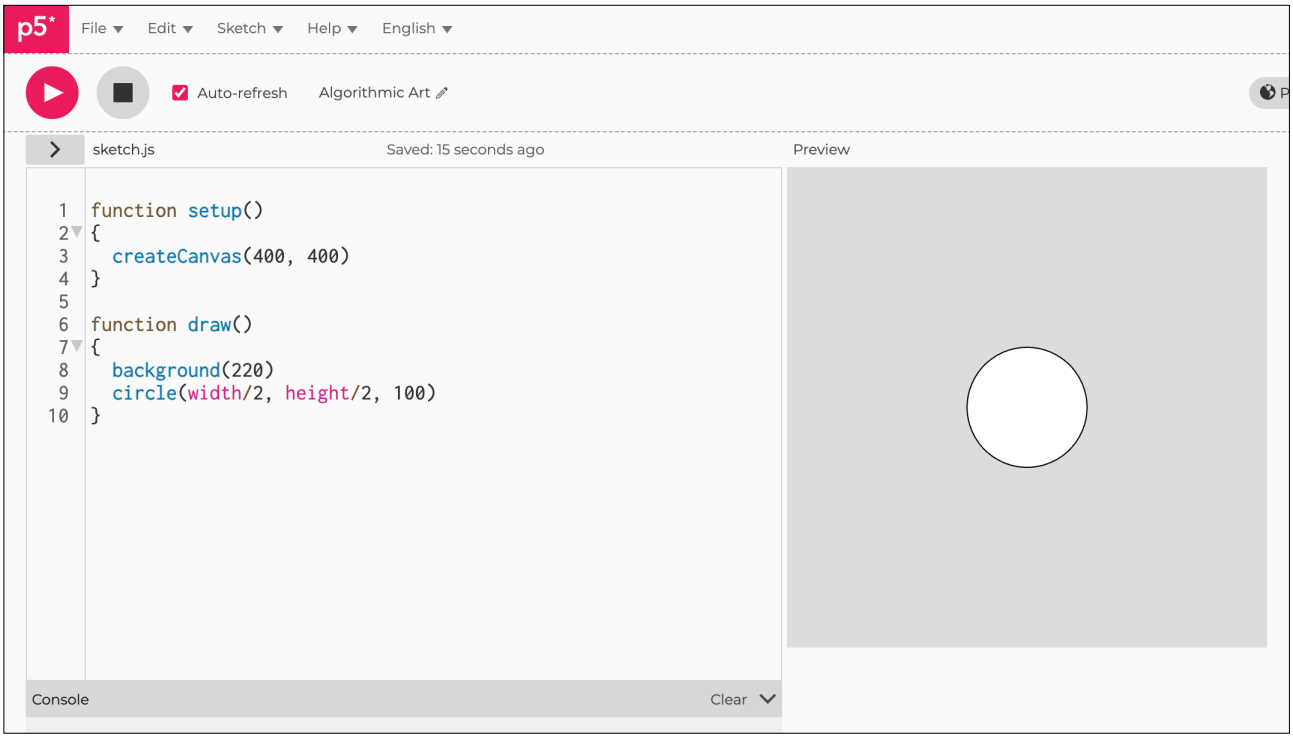
```
function setup()
{
  createCanvas(400, 400)
}

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.

```
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.

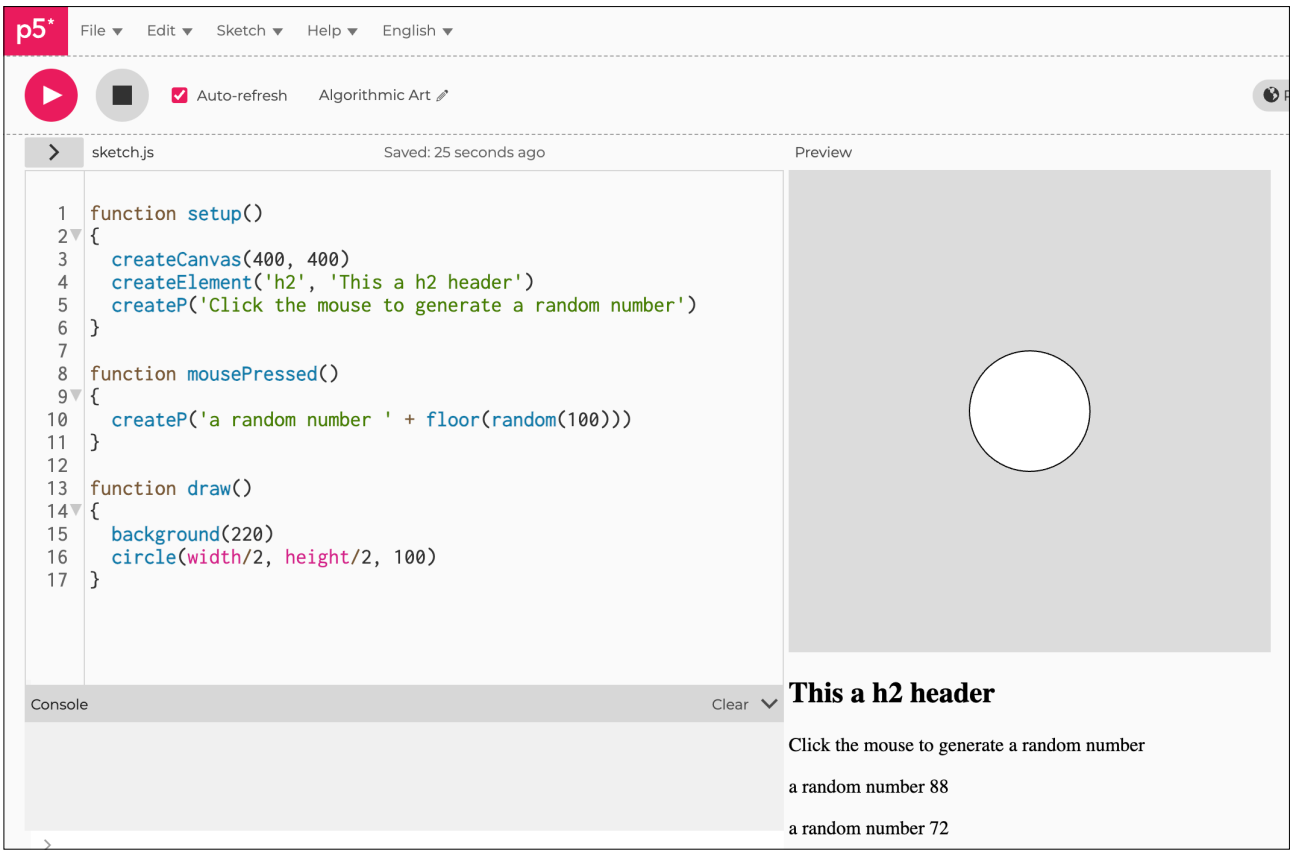
Challenges

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


p5*
File
Edit
Sketch
Help
English
Auto-refresh
Algorithmic Art
sketch.js
Saved: 25 seconds ago
Preview
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)
}
Console
Clear
This a h2 header
Click the mouse to generate a random number
a random number 88
a random number 72



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.

```
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('click on this button')
}

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

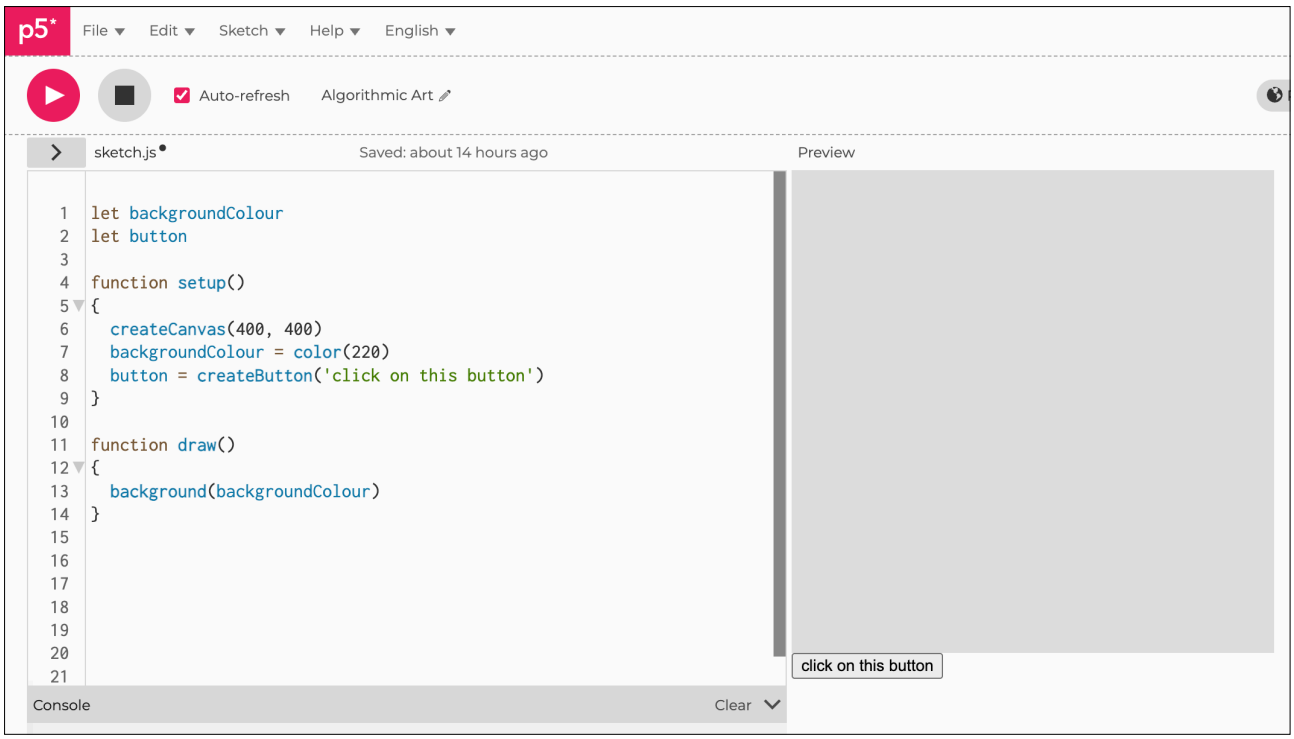
Notes

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


p5*
File
Edit
Sketch
Help
English
Auto-refresh
Algorithmic Art
sketch.js
Saved: about 14 hours ago
Preview
let backgroundColour
let button
function setup()
{
createCanvas(400, 400)
backgroundColour = color(220)
button = createButton('click on this button')
}
function draw()
{
background(backgroundColour)
}
click on this button
Console
Clear

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('click 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).

```
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

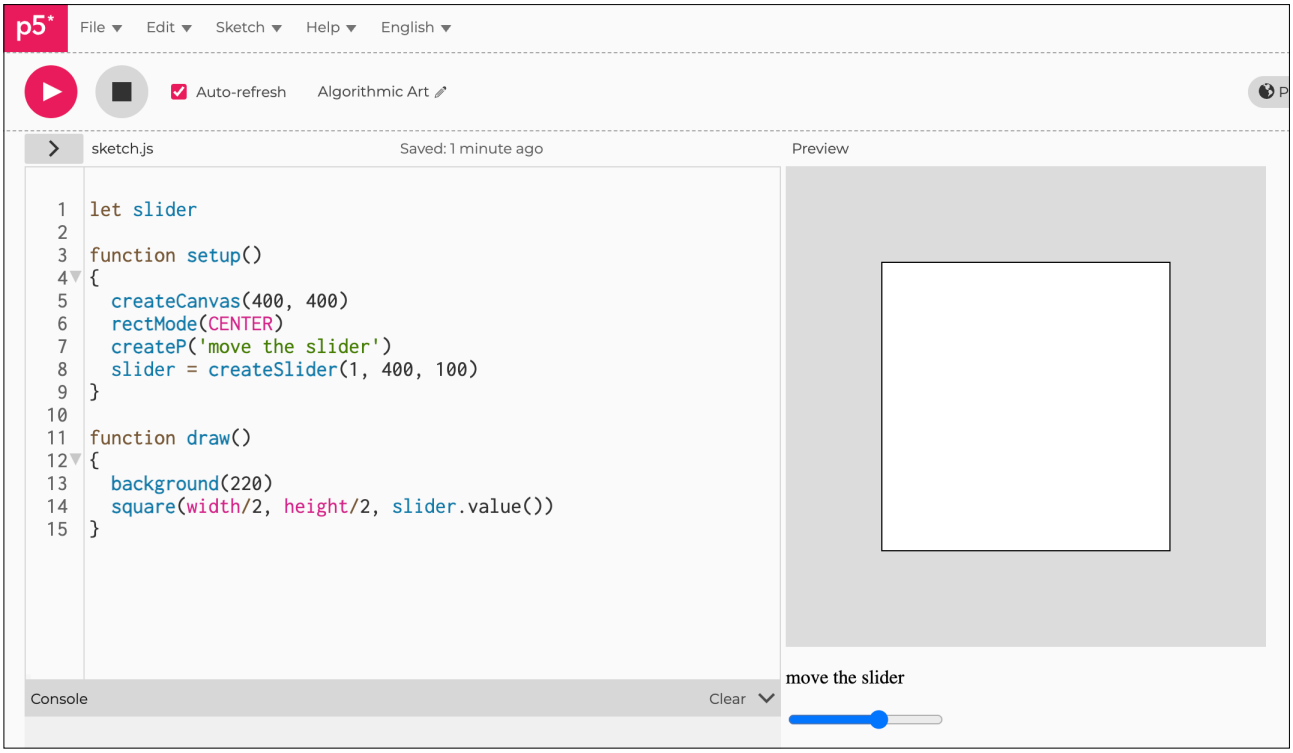
Using a slider DOM element. It has three arguments:
1. the first one is the minimum value,
2. the second is the maximum value, and
3. the third is the starting value.

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


p5*
File
Edit
Sketch
Help
English
Auto-refresh
Algorithmic Art
sketch.js
Saved: 1 minute ago
Preview
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())
}
Console
Clear
move the slider

Sketch F2.9 text box

! New sketch

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

```
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)
}
```

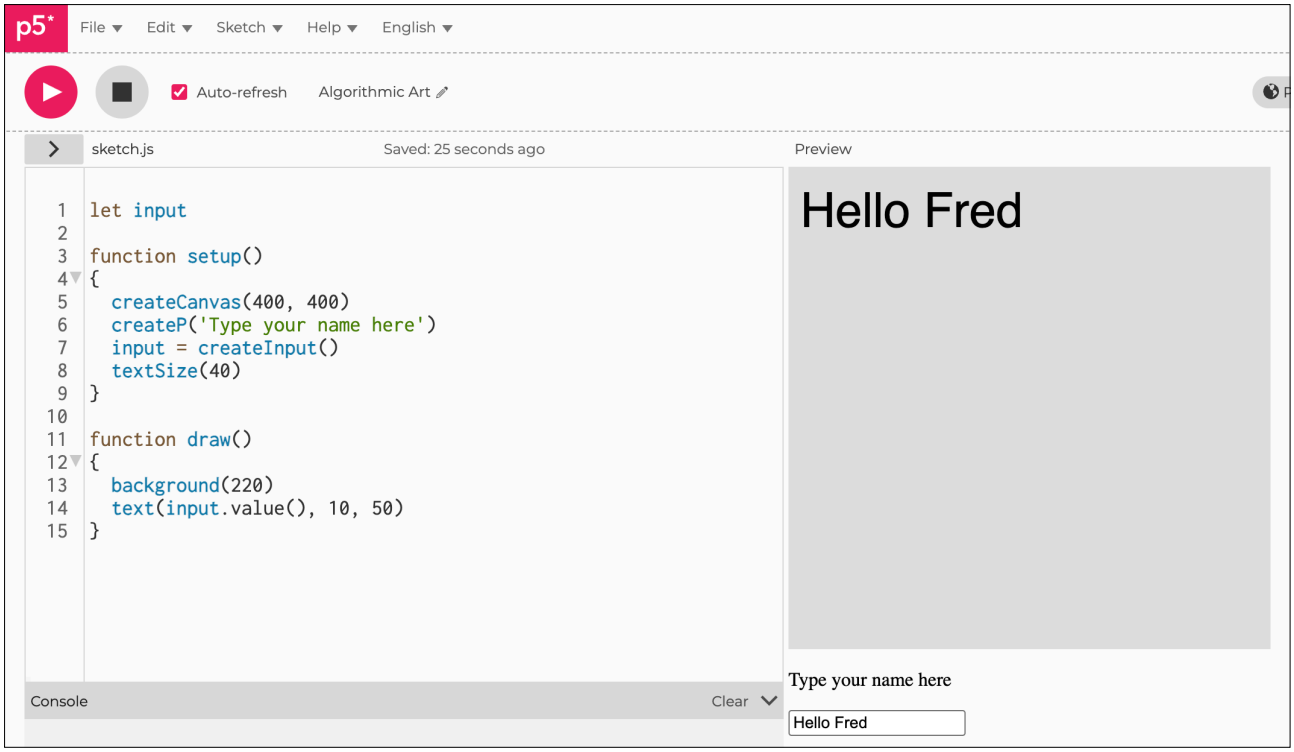
Notes

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


p5*
File
Edit
Sketch
Help
English
Auto-refresh
Algorithmic Art
sketch.js
Saved: 25 seconds ago
Preview
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)
}
Hello Fred
Console
Clear
Type your name here
Hello Fred

Sketch F2.10 hovering over text

! New sketch

Using HTML to change the text.

```
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')
}
```

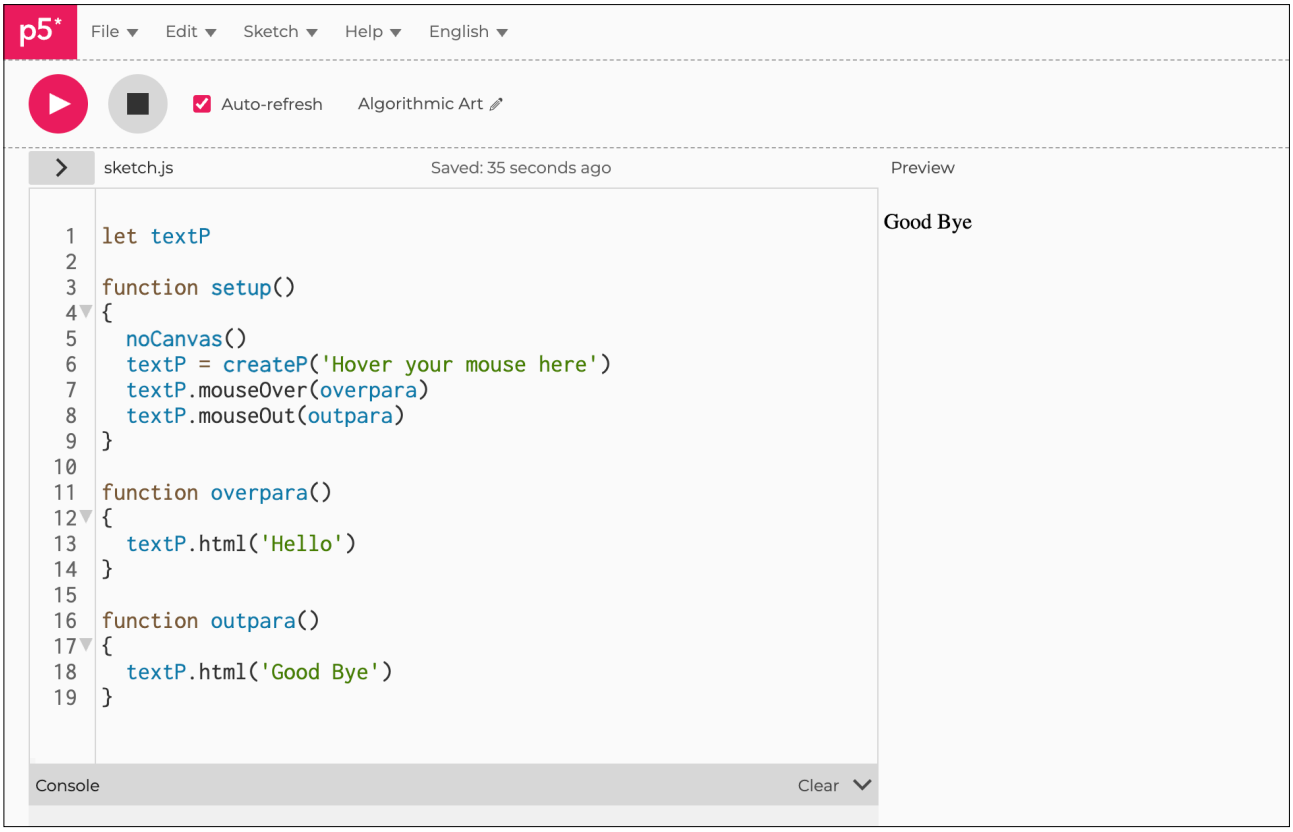
Notes

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


p5*
File
Edit
Sketch
Help
English
Auto-refresh
Algorithmic Art
sketch.js
Saved: 35 seconds ago
Preview
Good Bye
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')
}
Console
Clear



Sketch F2.11 hovering over the canvas

! Start a new sketch.

Three ways to change the colour of the background.

```
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)
}
```

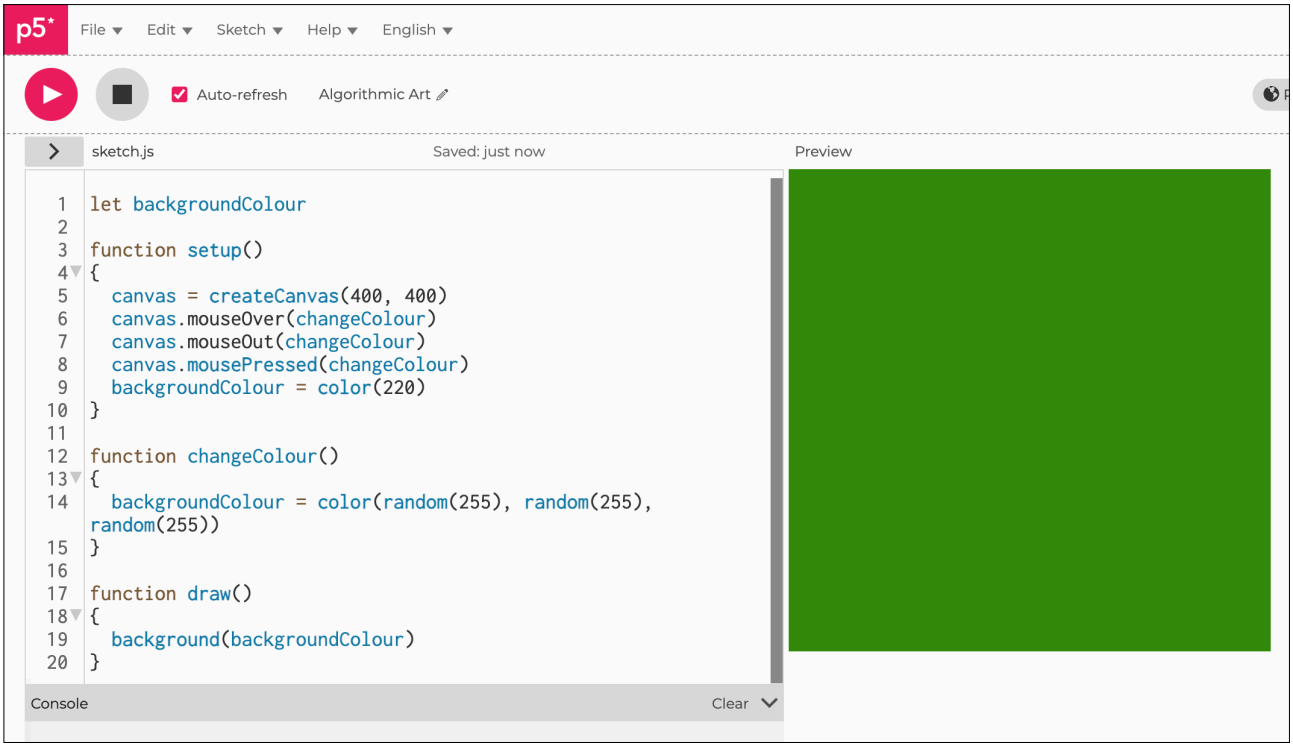
Notes

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


p5*
File
Edit
Sketch
Help
English
Auto-refresh
Algorithmic Art
sketch.js
Saved: just now
Preview
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)
}
Console
Clear



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 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

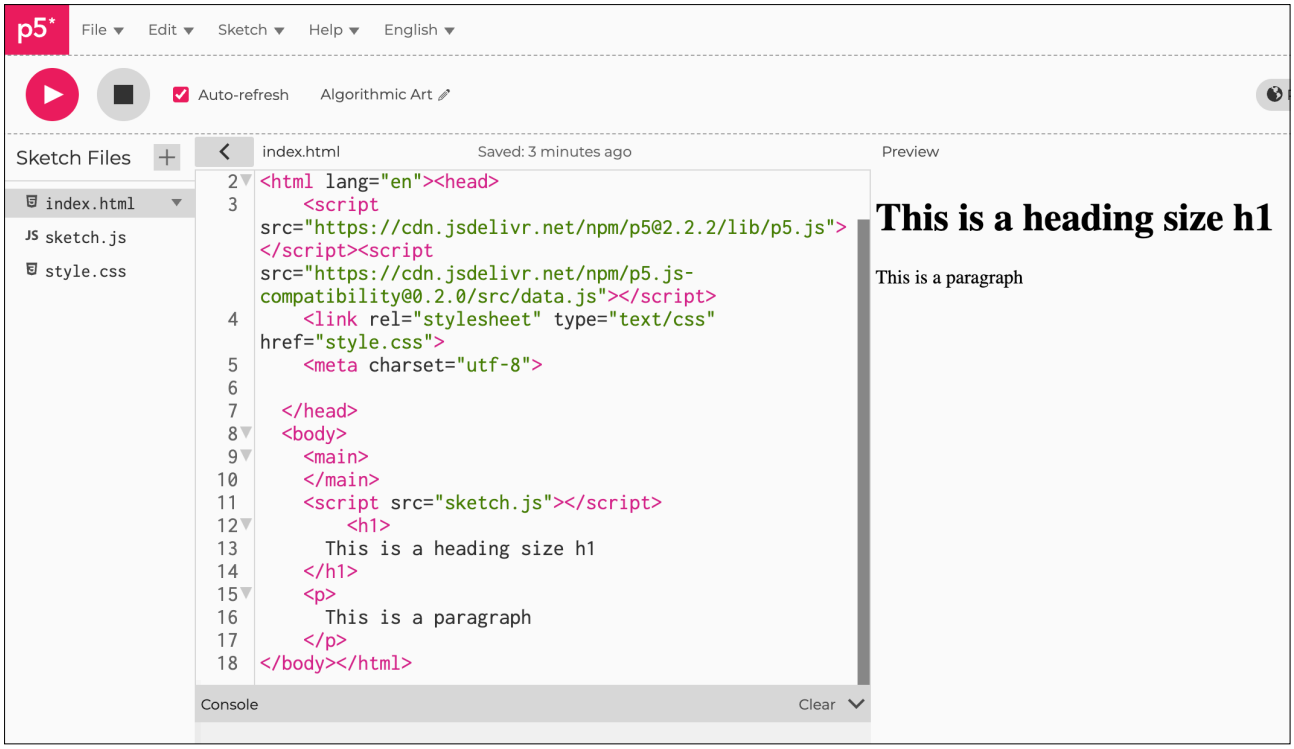
```
<!DOCTYPE html>
<html lang="en"><head>
    <script src="https://cdn.jsdelivr.net/npm/p5@2.2.2/lib/p5.js"></
script><script src="https://cdn.jsdelivr.net/npm/p5.js-compatibility@0.2.0/
src/data.js"></script>
    <link rel="stylesheet" type="text/css" href="style.css">
    <meta charset="utf-8">


  </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


p5*
File
Edit
Sketch
Help
English
Auto-refresh
Algorithmic Art
Sketch Files
index.html
sketch.js
style.css
index.html
Saved: 3 minutes ago
2 <html lang="en"><head>
3 <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>
4 <link rel="stylesheet" type="text/css"
href="style.css">
5 <meta charset="utf-8">
6
7 </head>
8 <body>
9 <main>
10 </main>
11 <script src="sketch.js"></script>
12 <h1>
13 This is a heading size h1
14 </h1>
15
16 This is a paragraph
17
18 </body></html>
Console
Clear
Preview
This is a heading size h1
This is a paragraph

Sketch F2.13 adding a sketch

! Move over to sketch.js.

The corresponding sketch.

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

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

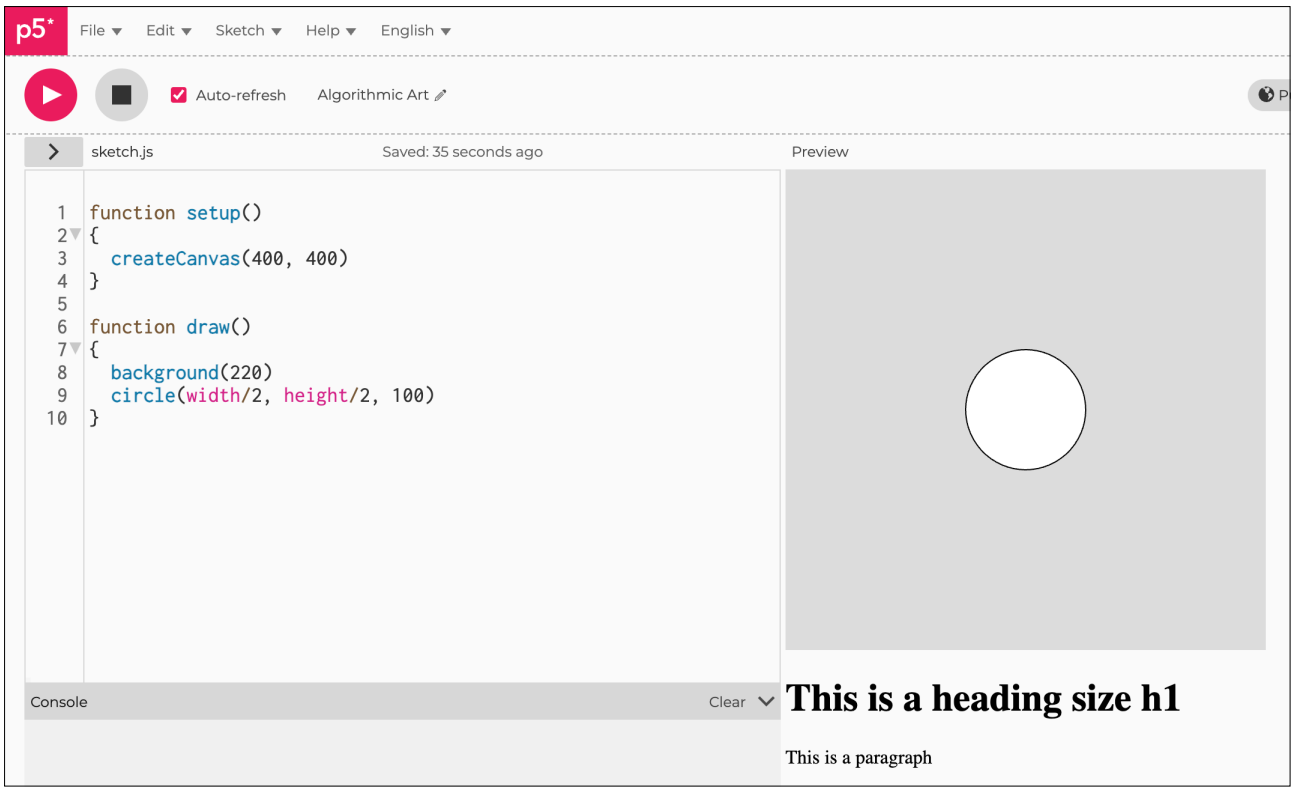
Notes

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   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


p5*
File
Edit
Sketch
Help
English
Auto-refresh
Algorithmic Art
sketch.js
Saved: 35 seconds ago
Preview
function setup()
{
createCanvas(400, 400)
}
function draw()
{
background(220)
circle(width/2, height/2, 100)
}
Console
Clear
This is a heading size h1
This is a paragraph

Sketch F2.14 CSS changed

! Remove the `draw()` function, in other words start a new sketch.

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). We have to use a variable name other than already used by p5.js, so I cannot use the word `text` for instance, instead I went for `words`.

```
let words

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

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

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

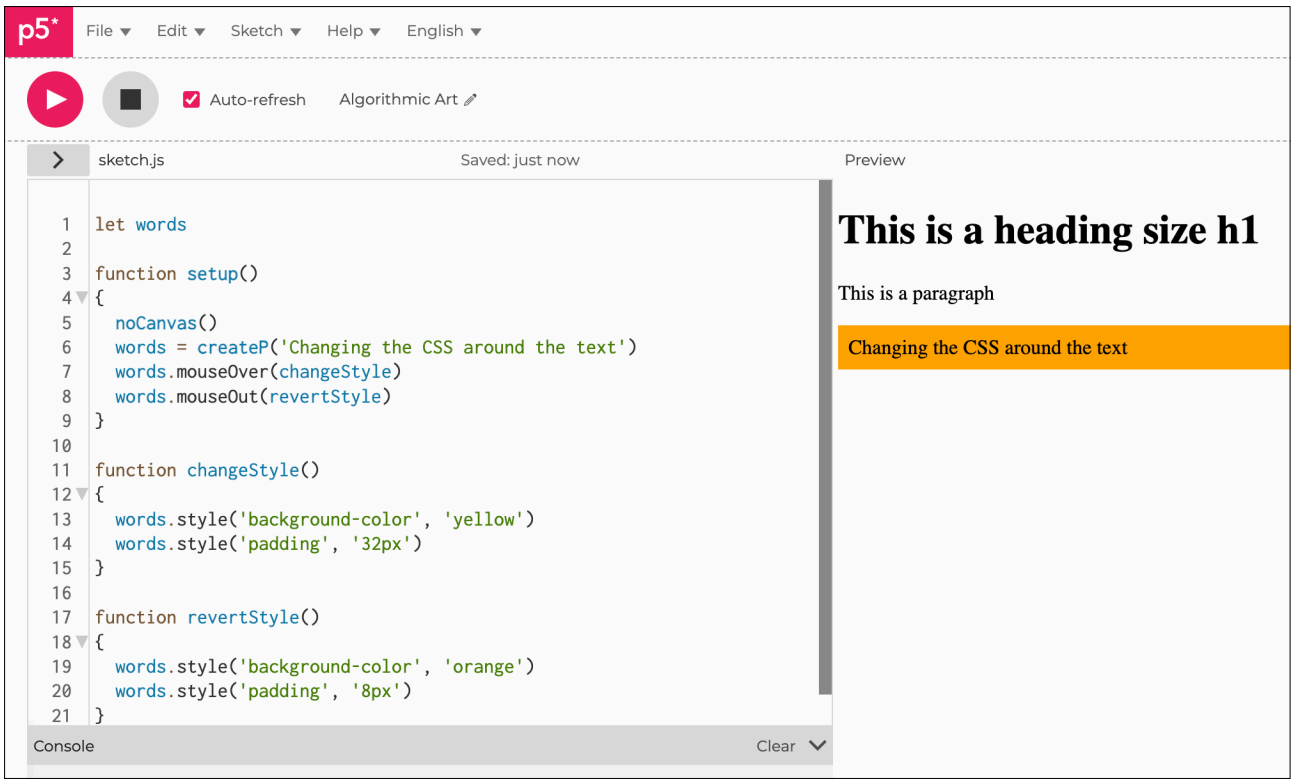
Notes

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


p5*
File
Edit
Sketch
Help
English
Auto-refresh
Algorithmic Art
sketch.js
Saved: just now
Preview
let words
function setup()
{
noCanvas()
words = createP('Changing the CSS around the text')
words.mouseOver(changeStyle)
words.mouseOut(revertStyle)
}
function changeStyle()
{
words.style('background-color', 'yellow')
words.style('padding', '32px')
}
function revertStyle()
{
words.style('background-color', 'orange')
words.style('padding', '8px')
}
Console
Clear
This is a heading size h1
This is a paragraph
Changing the CSS around the text

Sketch F2.15 submit button

! Start a new sketch, and also remove the HTML tags in the index.html file.

In this next bit, we are going to create a submit button that will take data (your name) and push it onto the canvas. First, we create an input box to input our name.

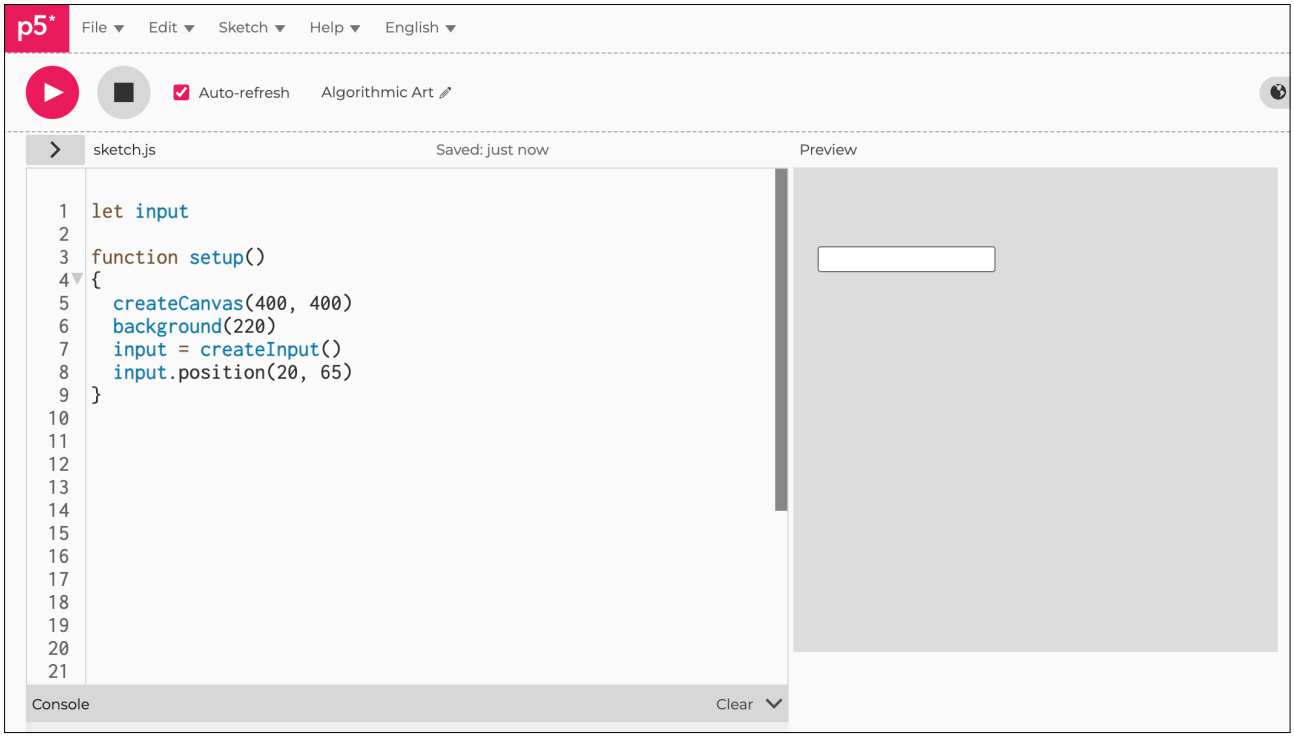
```
let input


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.

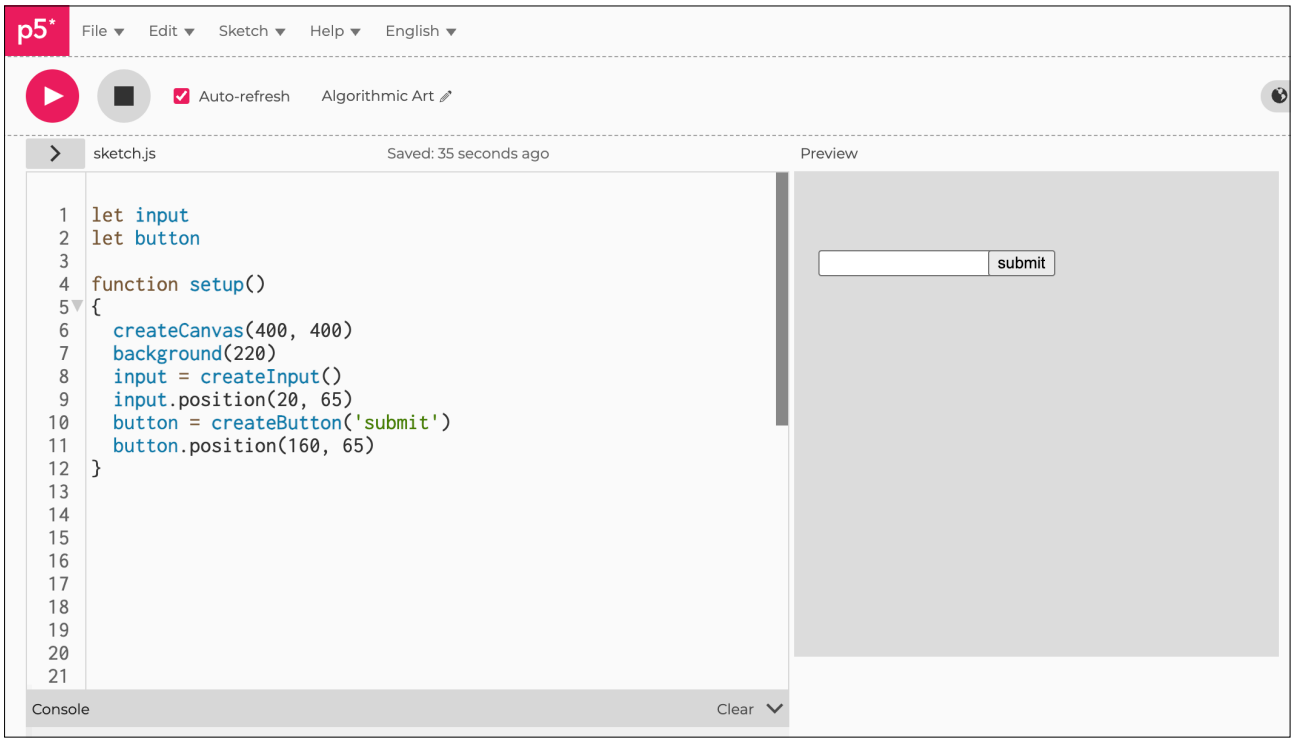
```
let input
let button

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.

```
let input
let button
let name

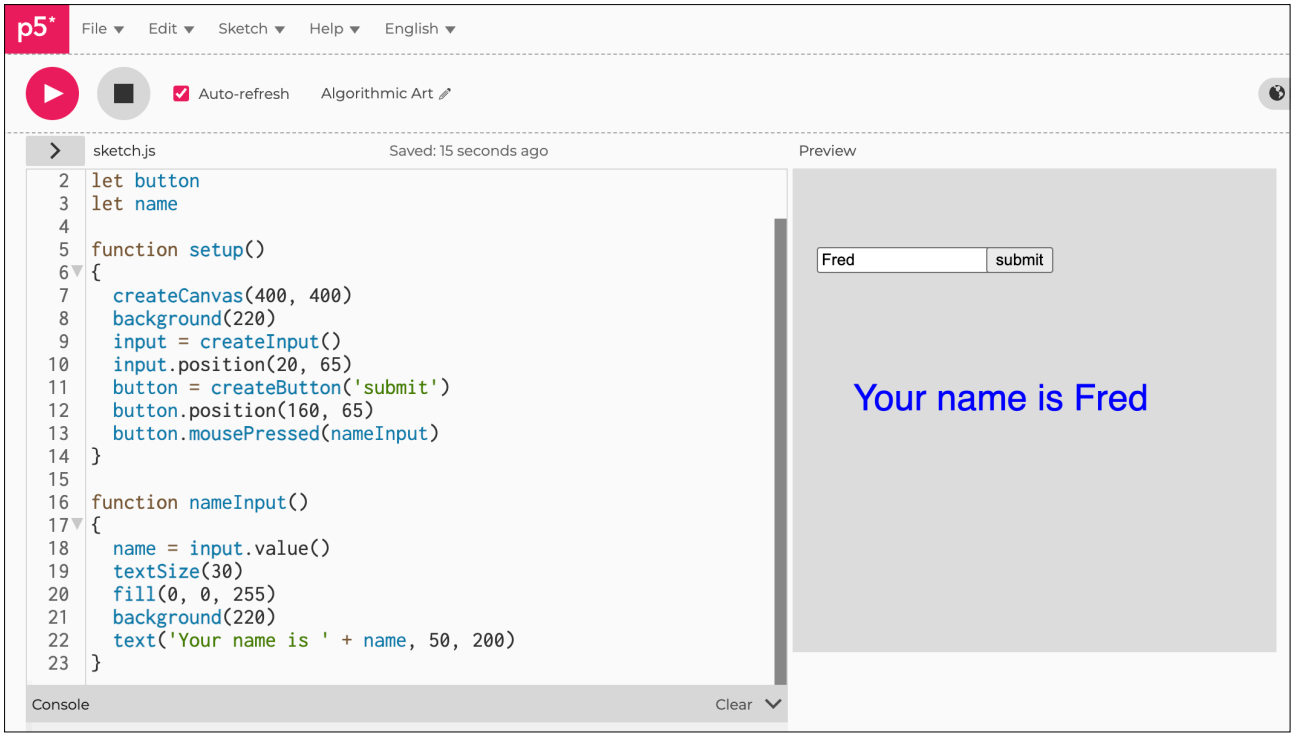
function setup()
{
  createCanvas(400, 400)
  background(220)
  input = createInput()
  input.position(20, 65)
  button = createButton('submit')
  button.position(160, 65)
  button.mousePressed(nameInput)
}

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


p5*
File
Edit
Sketch
Help
English
Auto-refresh
Algorithmic Art
sketch.js
Saved: 15 seconds ago
Preview
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)
}
Fred
submit
Your name is Fred
Console
Clear

Sketch F2.18 have the question

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

```
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)
}
```

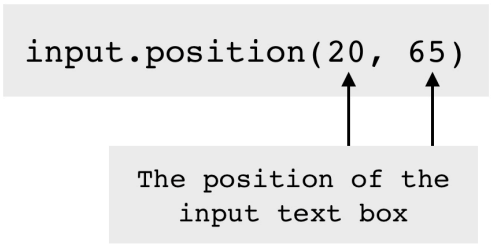
Notes

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.



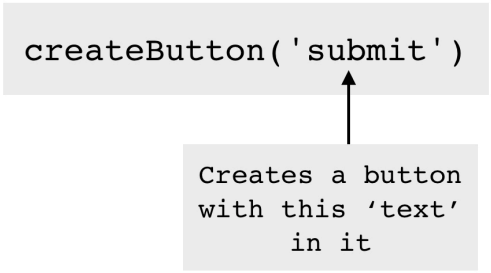
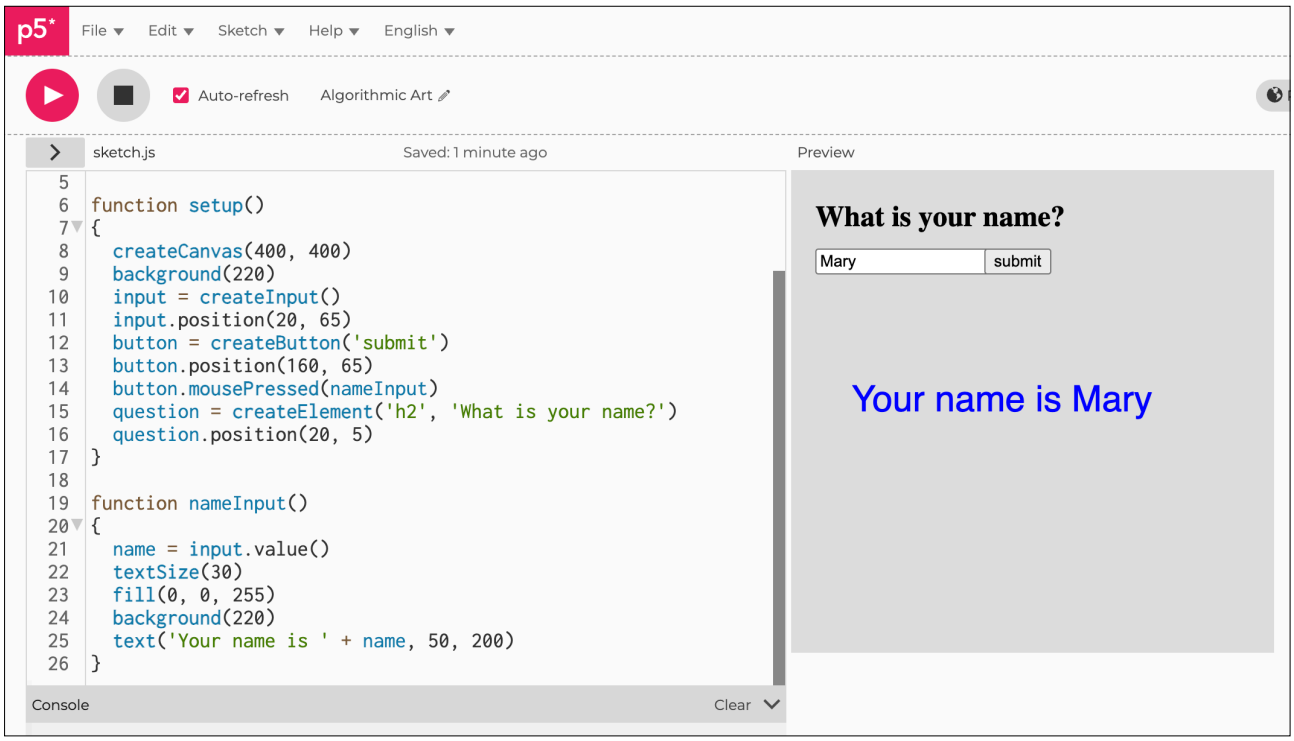


Figure F2.18


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.

```
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.

```
createElement('h2', 'Multiplying two numbers')
```

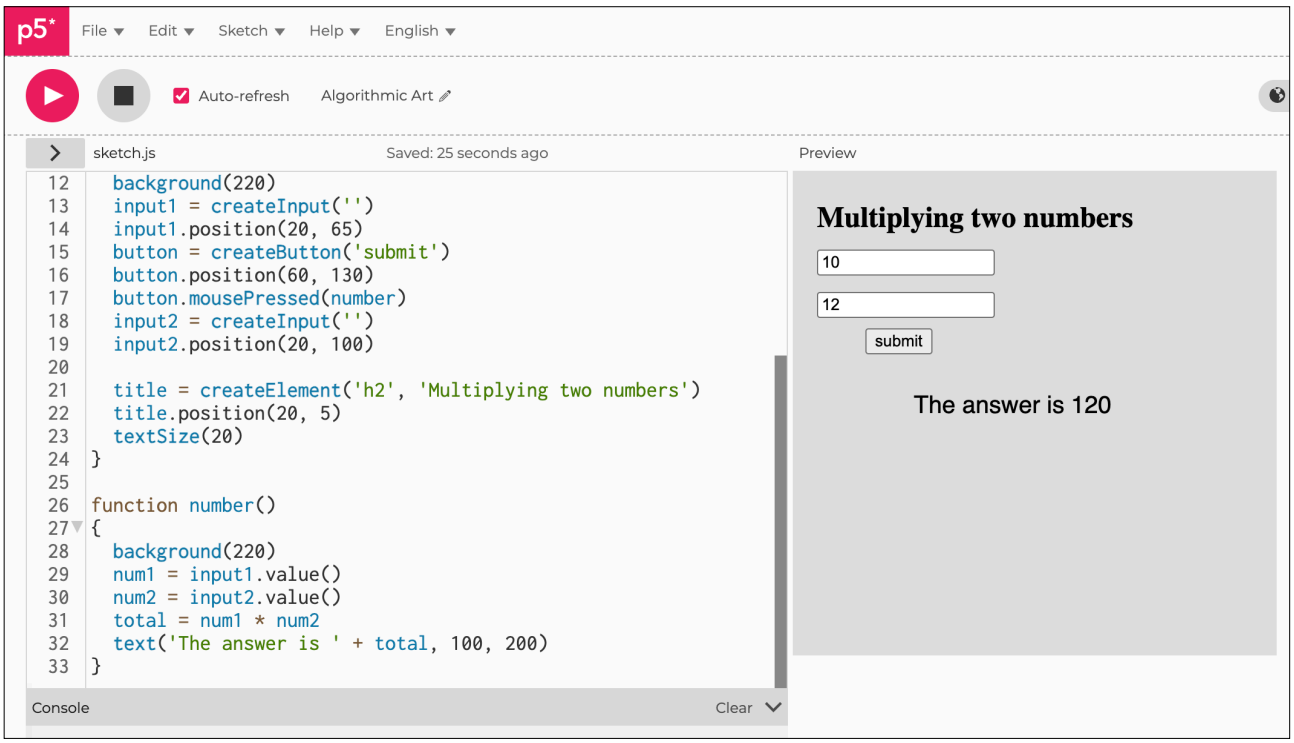
Heading size h2

Text to go inside the element

Challenge

Add more functionality with addition, subtraction, or division.

Figure F2.19



Sketch F2.20 slider colour circle

! Another new sketch.

Three RGB sliders so that you can create your own colour chart.

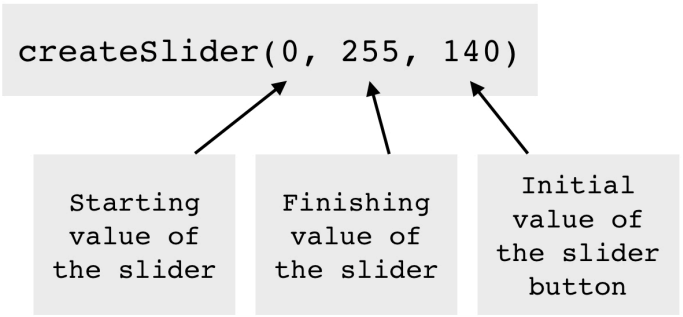
```
let sliderRed
let sliderGreen
let sliderBlue

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

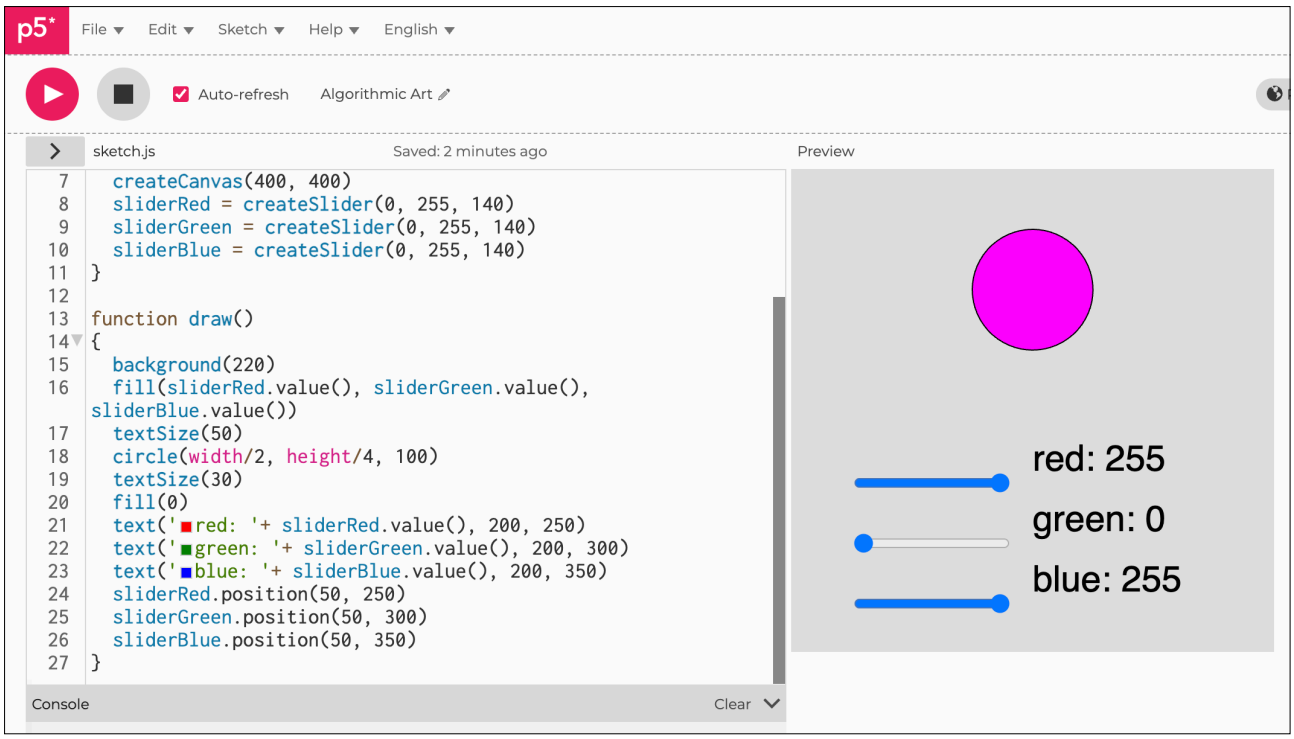
Here we can create a slider to change the colour of the circle. The line `createSlider(0, 255, 140)` gives us three arguments. The first one `(0)` is the minimum value, the second `(255)` is the maximum value, and the third `(140)` is the starting position of the slider button between the max and the 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)
}
red: 255
green: 0
blue: 255
Console
Clear

Sketch F2.21 slider rotate

! Yet another new sketch.

Rotating a baton with the slider, a nice demonstration.

```
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()
}
```

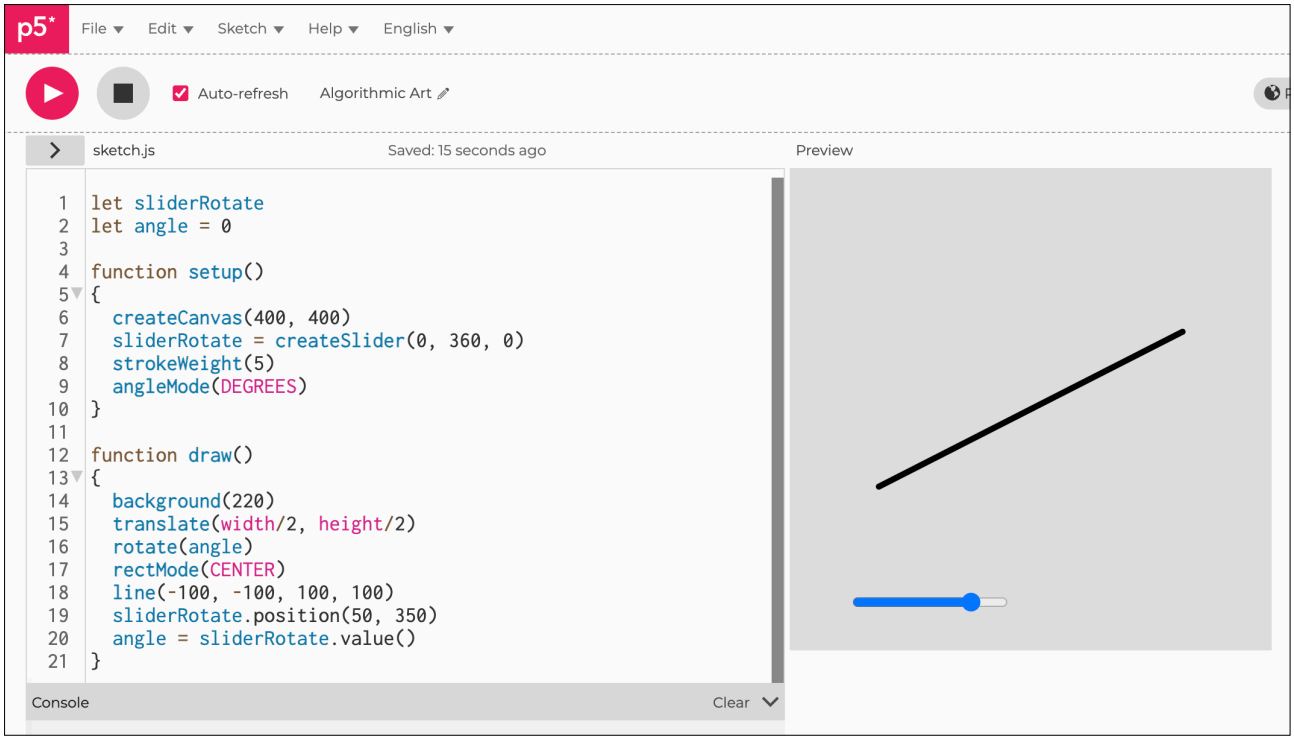
Notes

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


p5*
File
Edit
Sketch
Help
English
Auto-refresh
Algorithmic Art
sketch.js
Saved: 15 seconds ago
Preview
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()
}
Console
Clear

Sketch F2.22 radio buttons colour

! Another new sketch

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

```
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)
  }
}
```

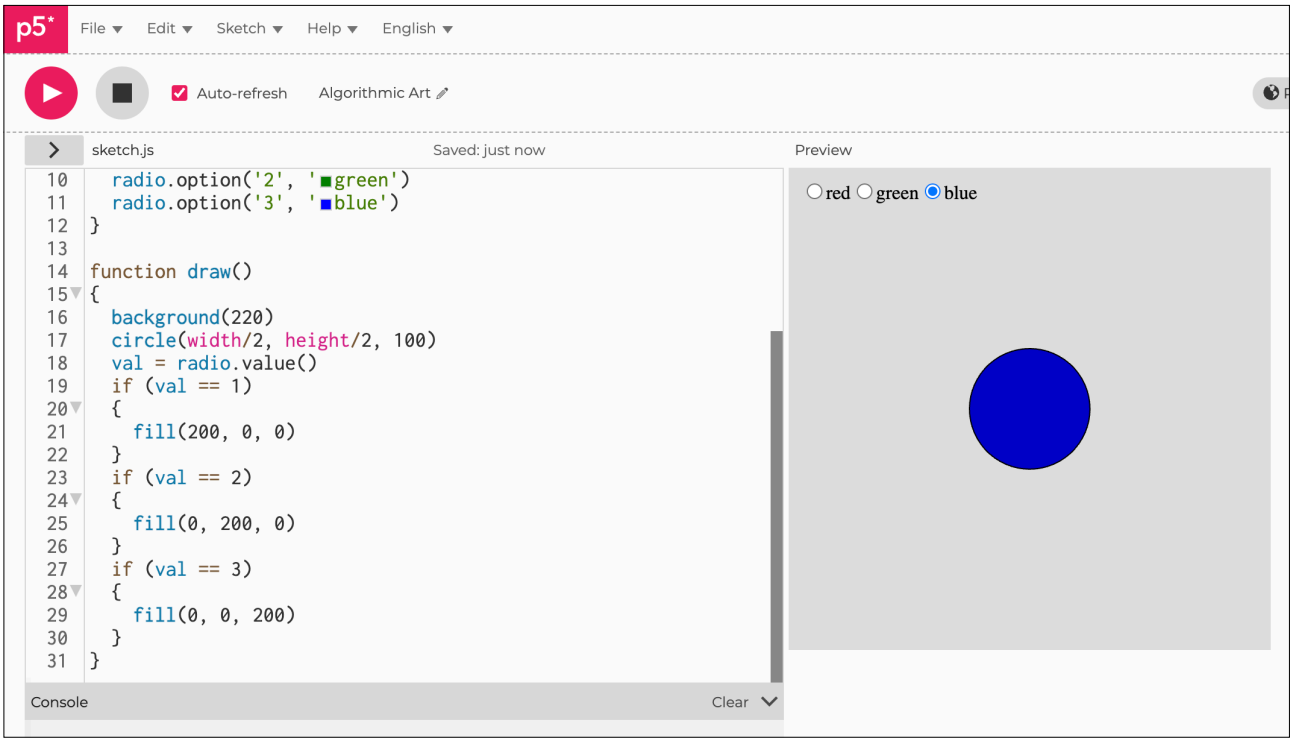
Notes

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 video input; most machines have them built in. Using the webcam, you can create and manipulate images and video.

When you run the programme, you will most likely need to give `p5.js` permission to access the webcam.

Sketch F3.1 scale

! This is our starting sketch with the main bits highlighted.

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`.

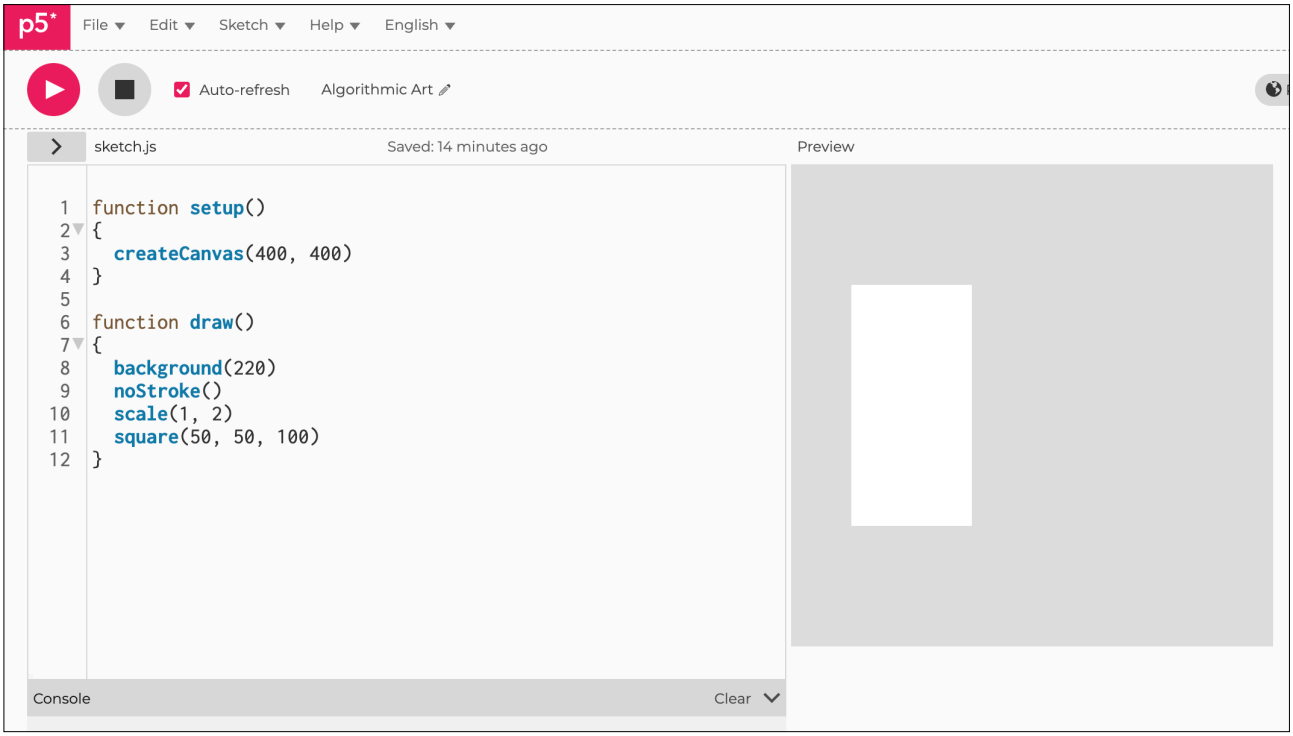
Challenges

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


p5*
File
Edit
Sketch
Help
English
Auto-refresh
Algorithmic Art
sketch.js
Saved: 14 minutes ago
Preview
1 function setup()
2 {
3 createCanvas(400, 400)
4 }
5
6 function draw()
7 {
8 background(220)
9 noStroke()
10 scale(1, 2)
11 square(50, 50, 100)
12 }
Console
Clear



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)
}
```

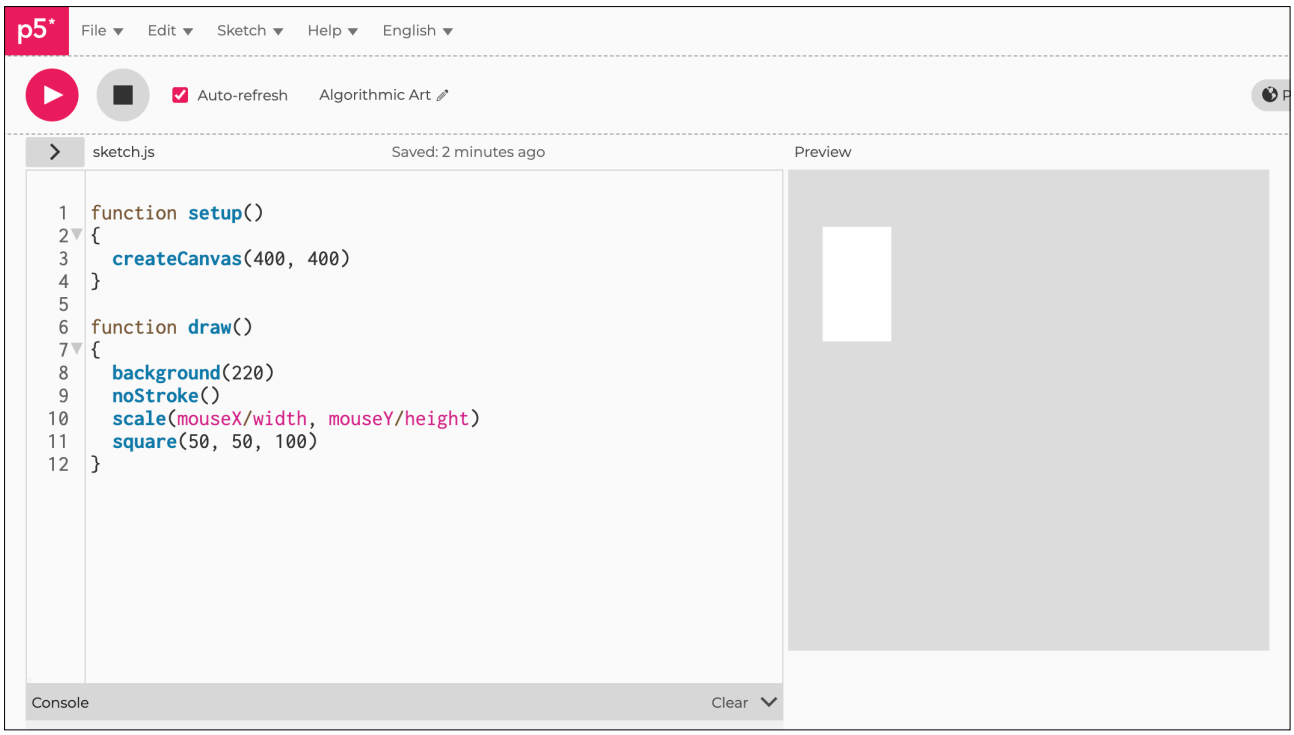
Notes

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


p5*
File
Edit
Sketch
Help
English
Auto-refresh
Algorithmic Art
sketch.js
Saved: 2 minutes ago
Preview
function setup()
{
createCanvas(400, 400)
}
function draw()
{
background(220)
noStroke()
scale(mouseX/width, mouseY/height)
square(50, 50, 100)
}
Console
Clear

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


p5*
File
Edit
Sketch
Help
English
Auto-refresh
Algorithmic Art
sketch.js
Saved: 15 seconds ago
Preview
1 function setup()
2 {
3 createCanvas(400, 400)
4 }
5
6 function draw()
7 {
8 background(220)
9 noStroke()
10 translate(width/2, height/2)
11 fill(255)
12 rect(0, 0, 100, 50)
13 fill(0)
14 scale(-1, -3)
15 rect(0, 0, 100, 50)
16 }
Console
Clear

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.

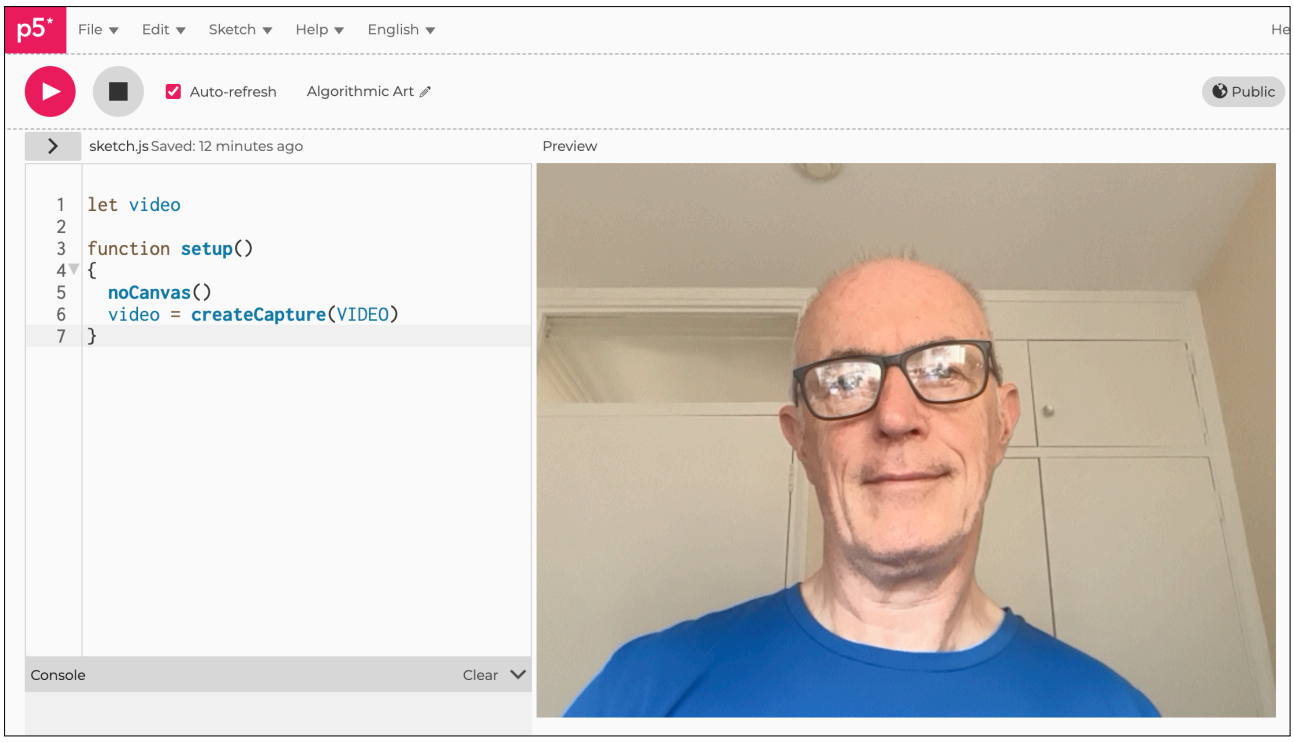
```
let video

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)`.

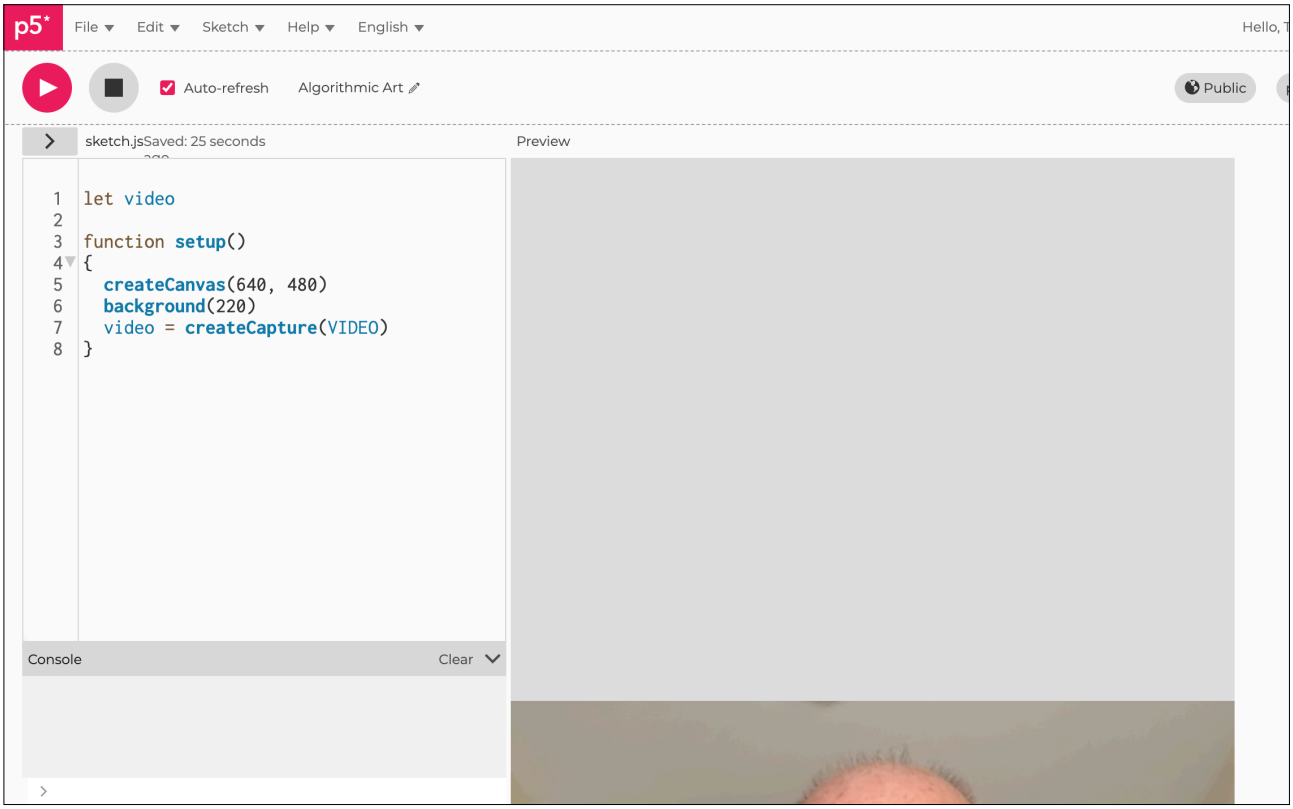
```
let video

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)
}
```

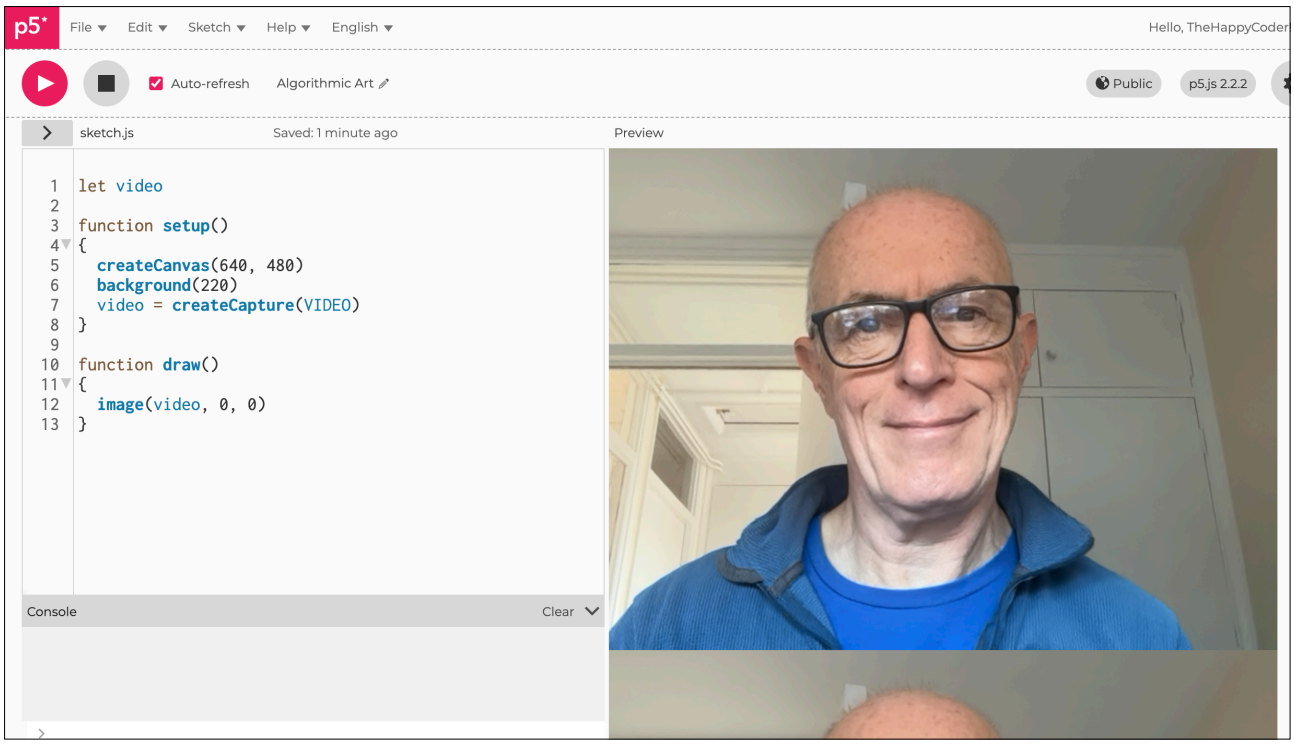
Notes

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


p5*
File
Edit
Sketch
Help
English
Hello, TheHappyCoder
Auto-refresh
Algorithmic Art
Public
p5.js 2.2.2
sketch.js
Saved: 1 minute ago
Preview
let video
function setup()
{
createCanvas(640, 480)
background(220)
video = createCapture(VIDEO)
}
function draw()
{
image(video, 0, 0)
}
Console
Clear

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)
}
```

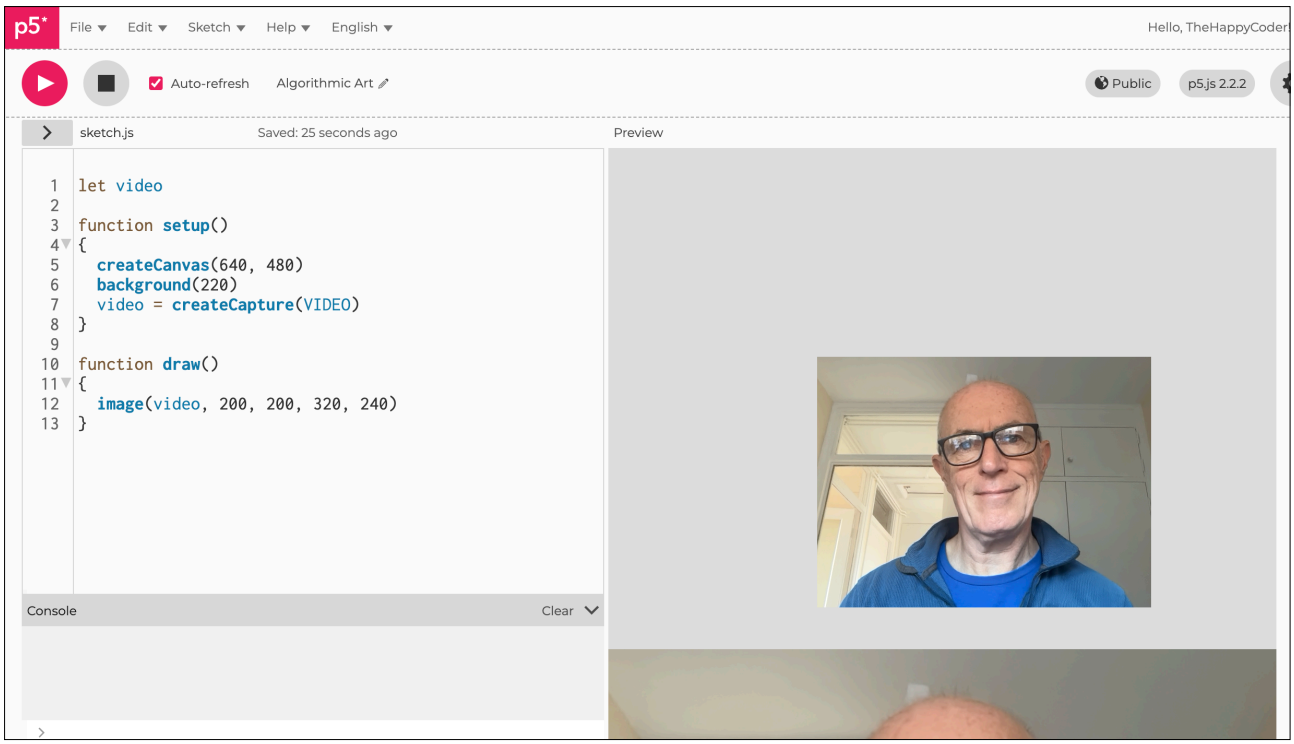
Notes

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


p5*
File
Edit
Sketch
Help
English
Hello, TheHappyCoder
Auto-refresh
Algorithmic Art
Public
p5.js 2.2.2
sketch.js
Saved: 25 seconds ago
Preview
let video
function setup()
{
createCanvas(640, 480)
background(220)
video = createCapture(VIDEO)
}
function draw()
{
image(video, 200, 200, 320, 240)
}
Console
Clear



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()`.

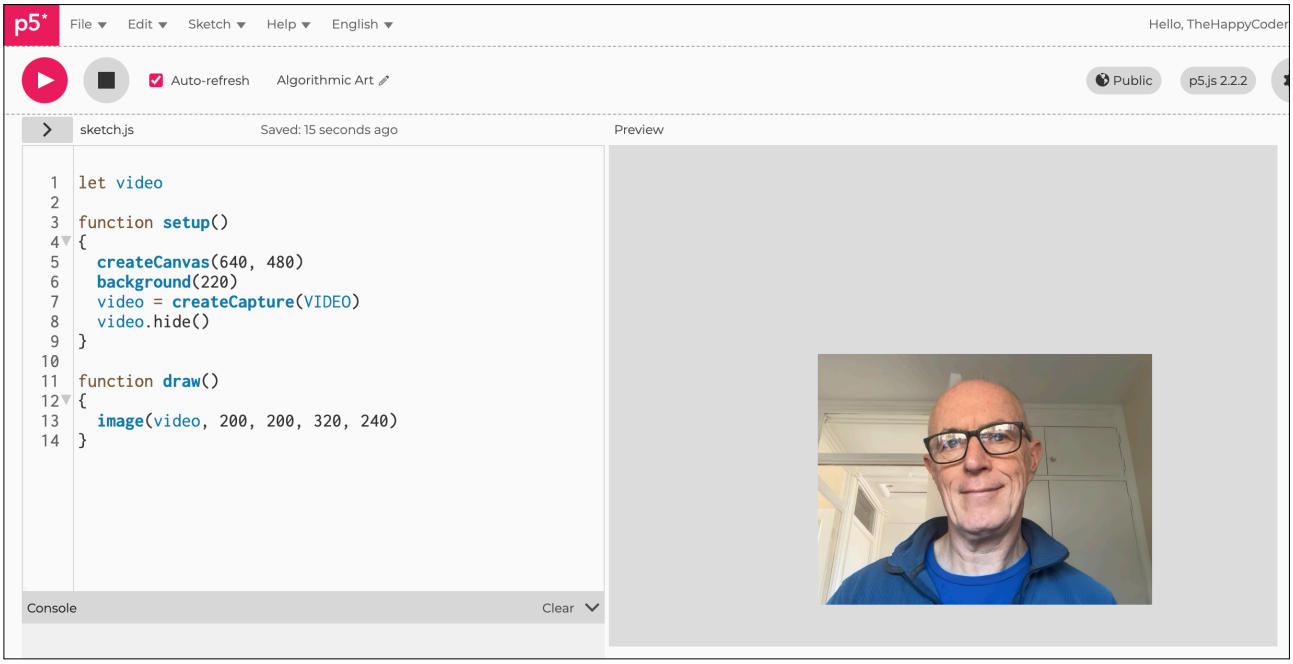
Challenges

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


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 video
function setup()
{
createCanvas(640, 480)
background(220)
video = createCapture(VIDEO)
video.hide()
}
function draw()
{
image(video, 200, 200, 320, 240)
}
Console
Clear

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)
}
```

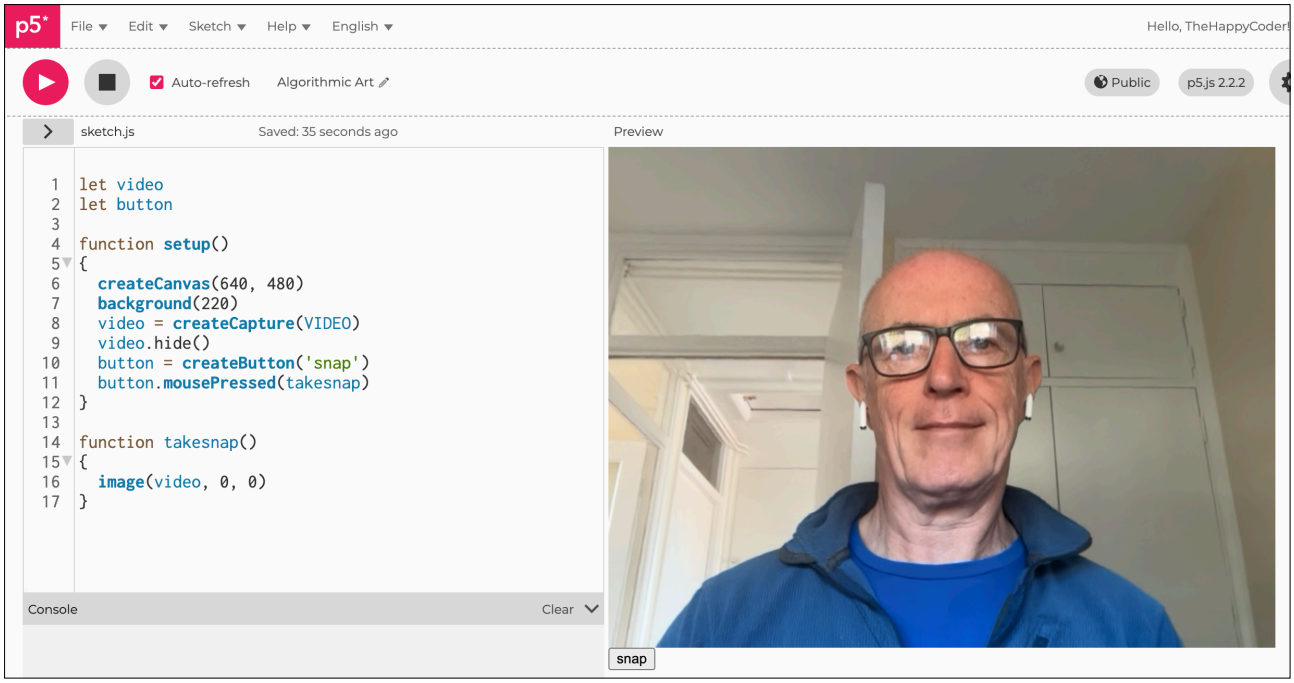
Notes

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


p5*
File
Edit
Sketch
Help
English
Hello, TheHappyCoder!
Auto-refresh
Algorithmic Art
Public
p5.js 2.2.2
sketch.js
Saved: 35 seconds ago
Preview
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)
}
Console
Clear
snap



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

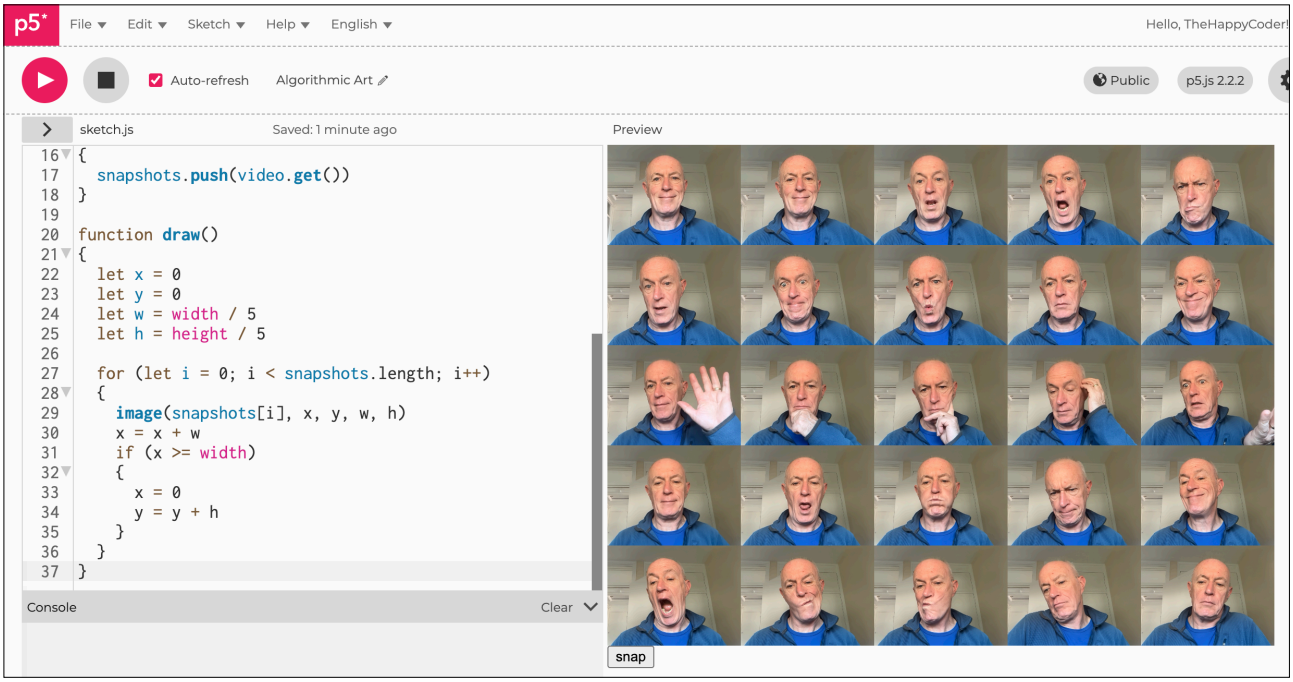
We create an empty array and call it `snapshots[]`. Every time the button is clicked, it adds an image to be stored in the array. The `get()` function gets the image, and the `push()` function pushes it into the array. By default, the `get()` function gets the whole image; you can specify the actual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

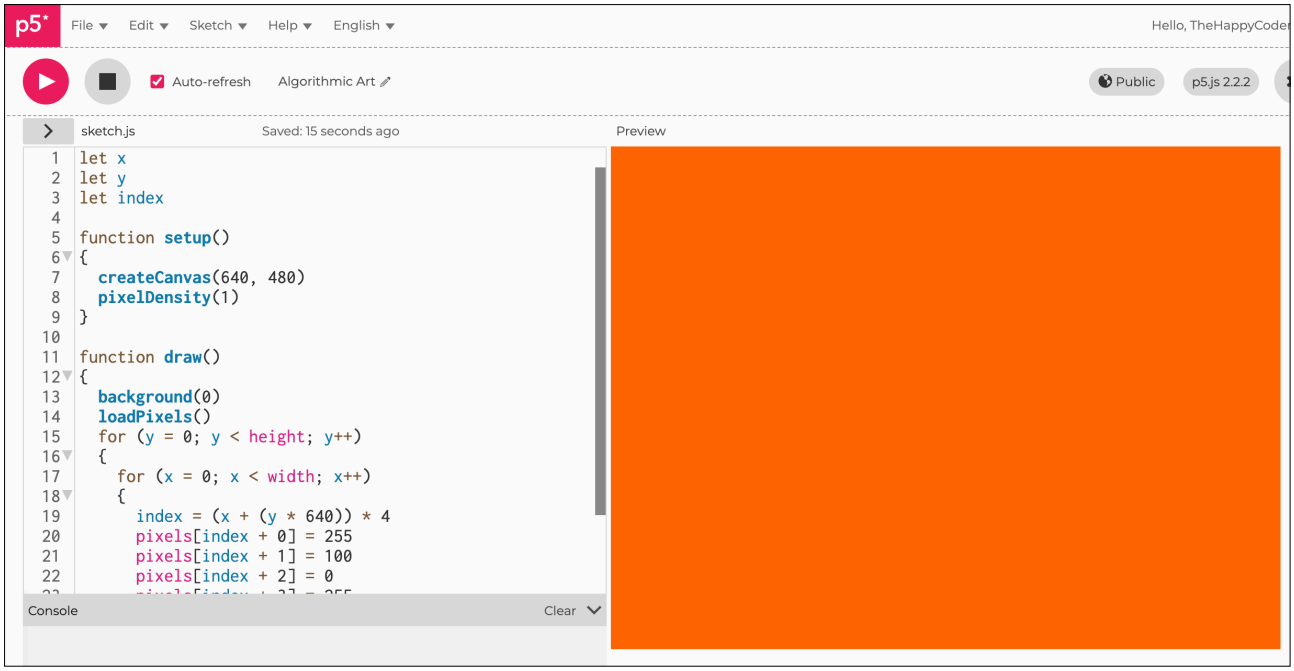
The line of code…
`index = (x + (y * 640)) * 4`
Goes through every fourth index. When it gets to the end of the row, it has done `640` of them, and so the next line of pixels starts with `640`. It sounds complicated, but this is how it works: through every pixel one at a time and then when it is at each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

! You will need to switch Loop Protection off otherwise you get an infinite loop warning and the sketch is terminated.

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()
}
```

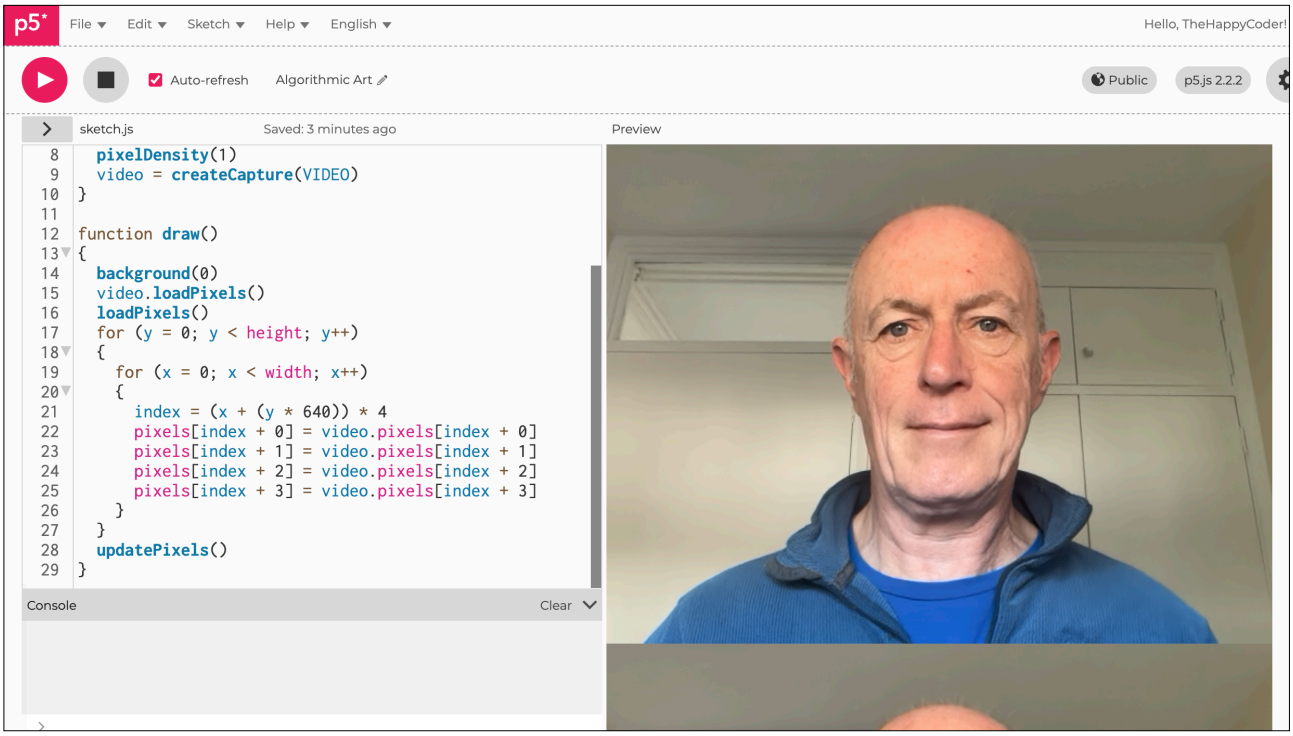
Notes

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)
    }
  }
}
```

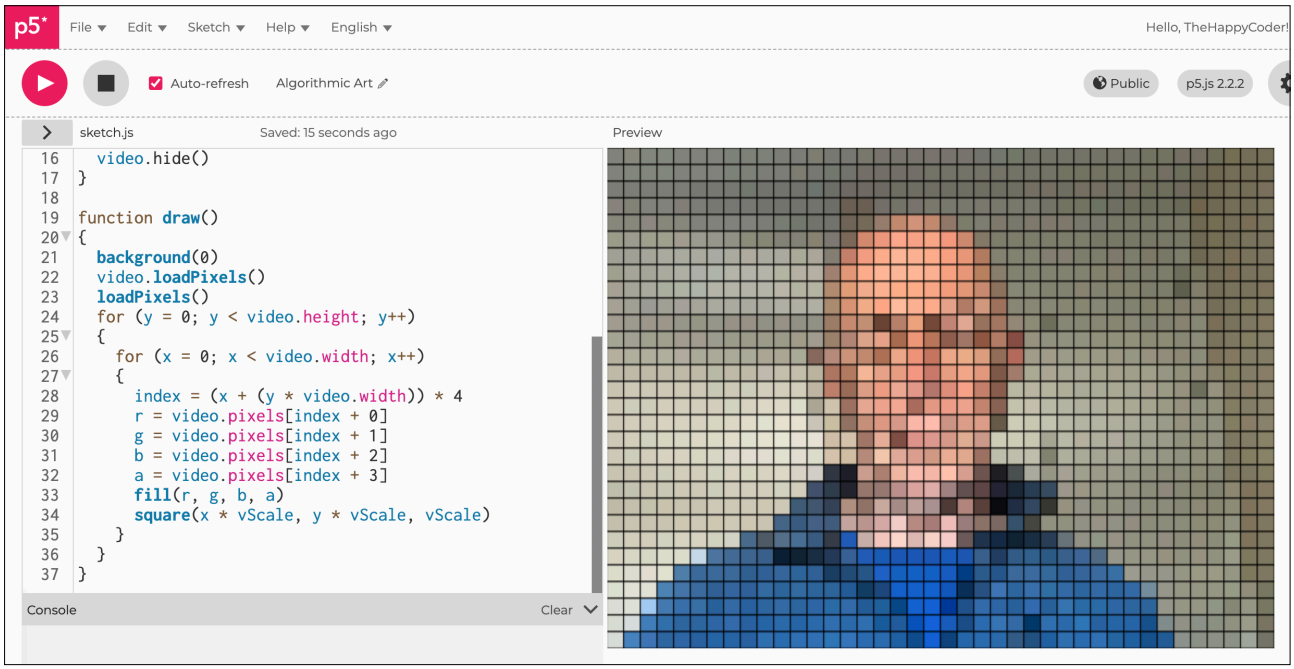
Notes

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()
}

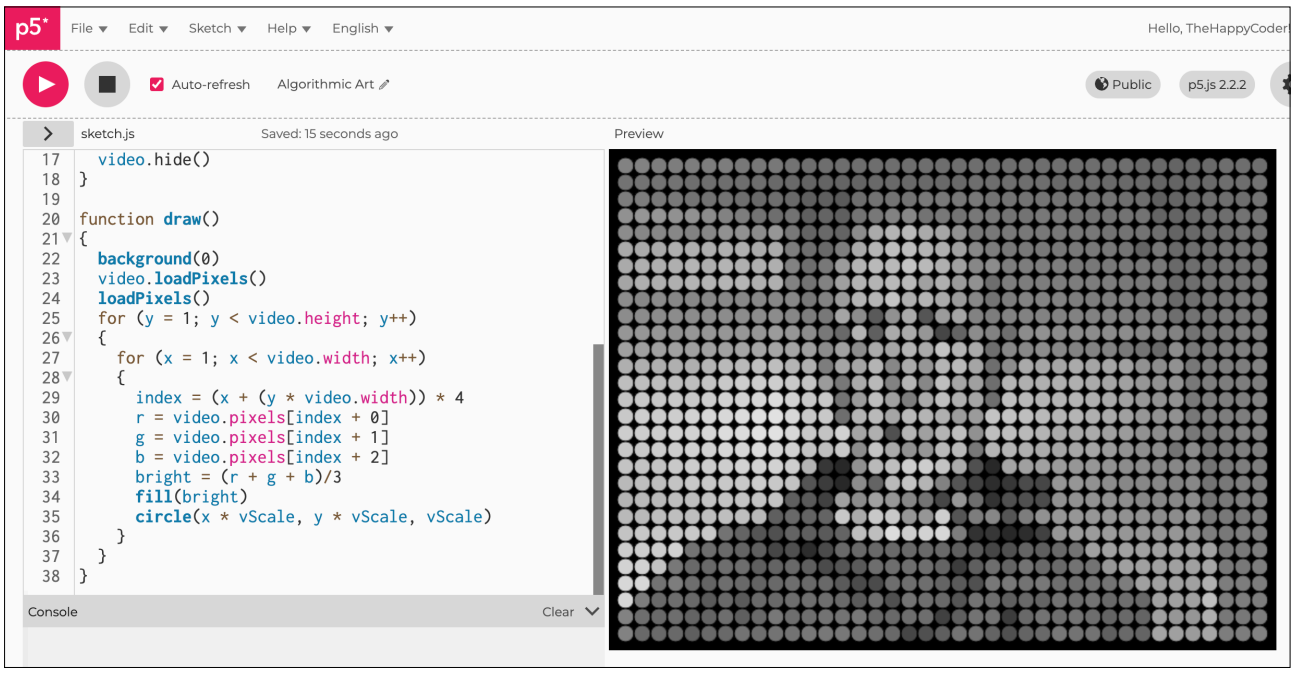
function draw()
{
  background(0)
  video.loadPixels()
  loadPixels()
  for (y = 1; y < video.height; y++)
  {
    for (x = 1; x < video.width; x++)
    {
      index = (x + (y * video.width)) * 4
      r = video.pixels[index + 0]
      g = video.pixels[index + 1]
      b = video.pixels[index + 2]
 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

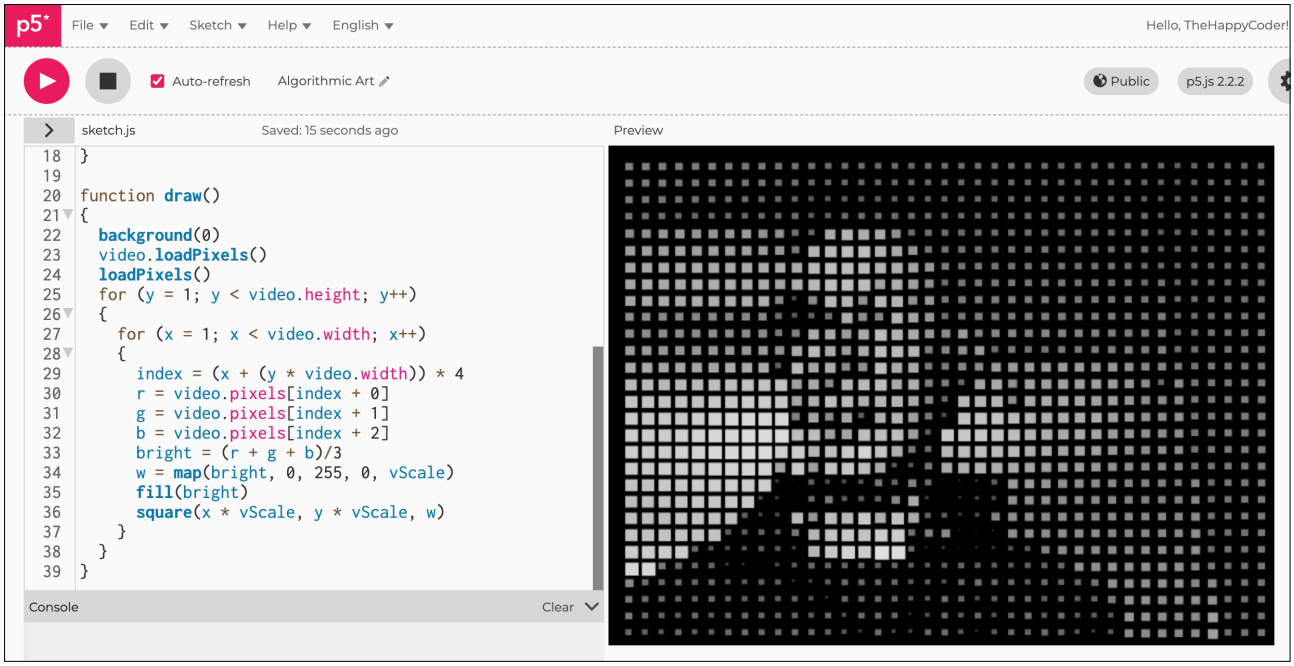
```
      w = map(bright, 0, 255, 0, vScale)
      fill(bright)
      square(x * vScale, y * vScale, w)
    }
  }
}
```

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 w
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)
    }
  }
}
```

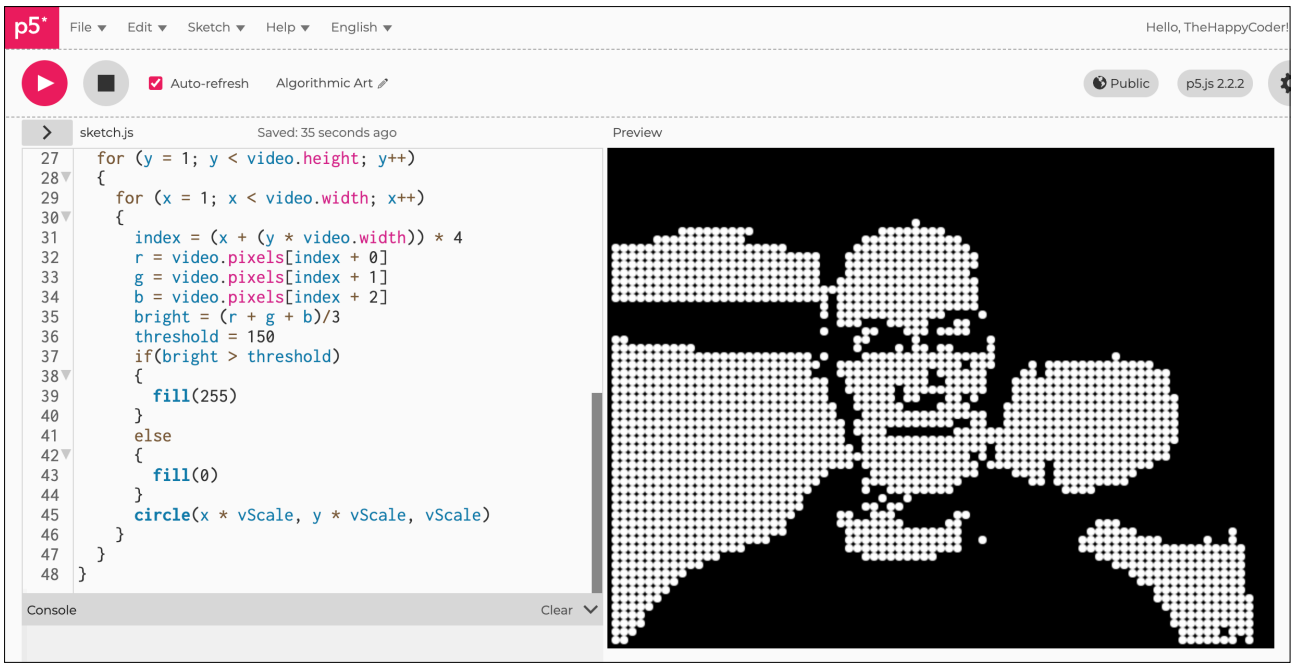
Notes

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


p5*
File
Edit
Sketch
Help
English
Hello, TheHappyCoder!
Auto-refresh
Algorithmic Art
Public
p5.js 2.2.2
sketch.js
Saved: 35 seconds ago
Preview
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)
}
}
Console
Clear

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
let w
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
      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)
    }
  }
}
```

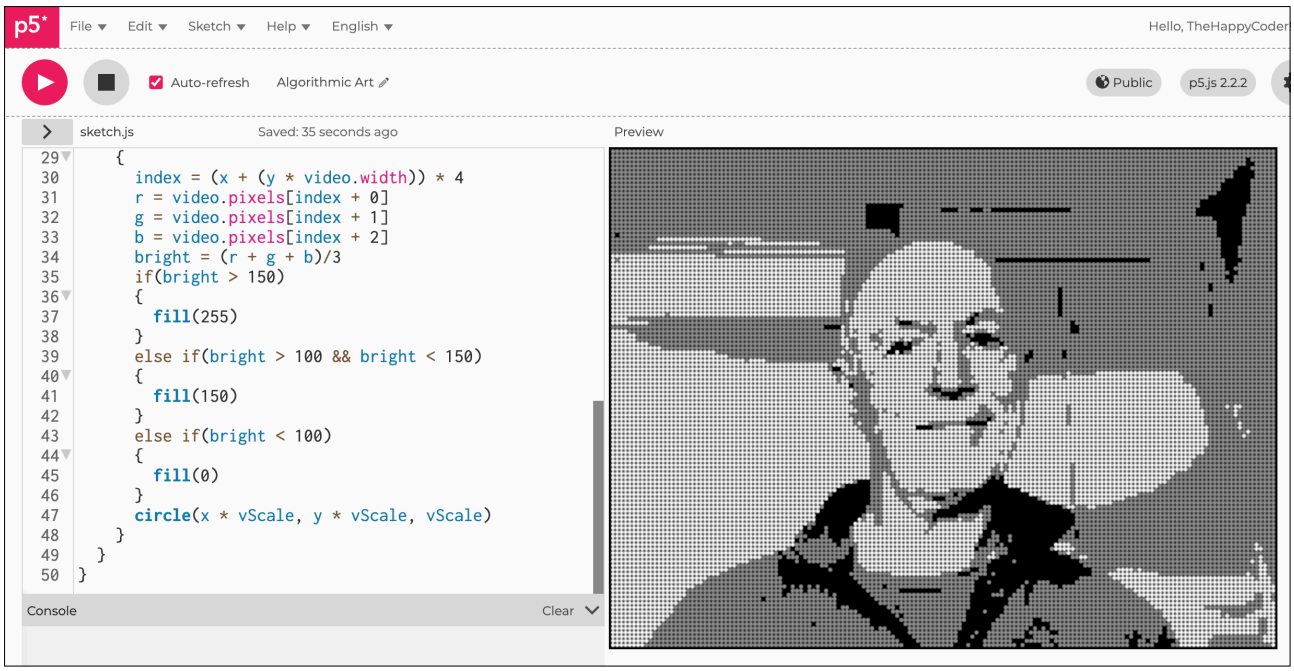
Notes

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


p5*
File
Edit
Sketch
Help
English
Hello, TheHappyCoder
Auto-refresh
Algorithmic Art
Public
p5.js 2.2.2
sketch.js
Saved: 35 seconds ago
Preview
{
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)
}
}
}
Console
Clear

Sketch F3.18 starting sketch

! We are going to start a new sketch.

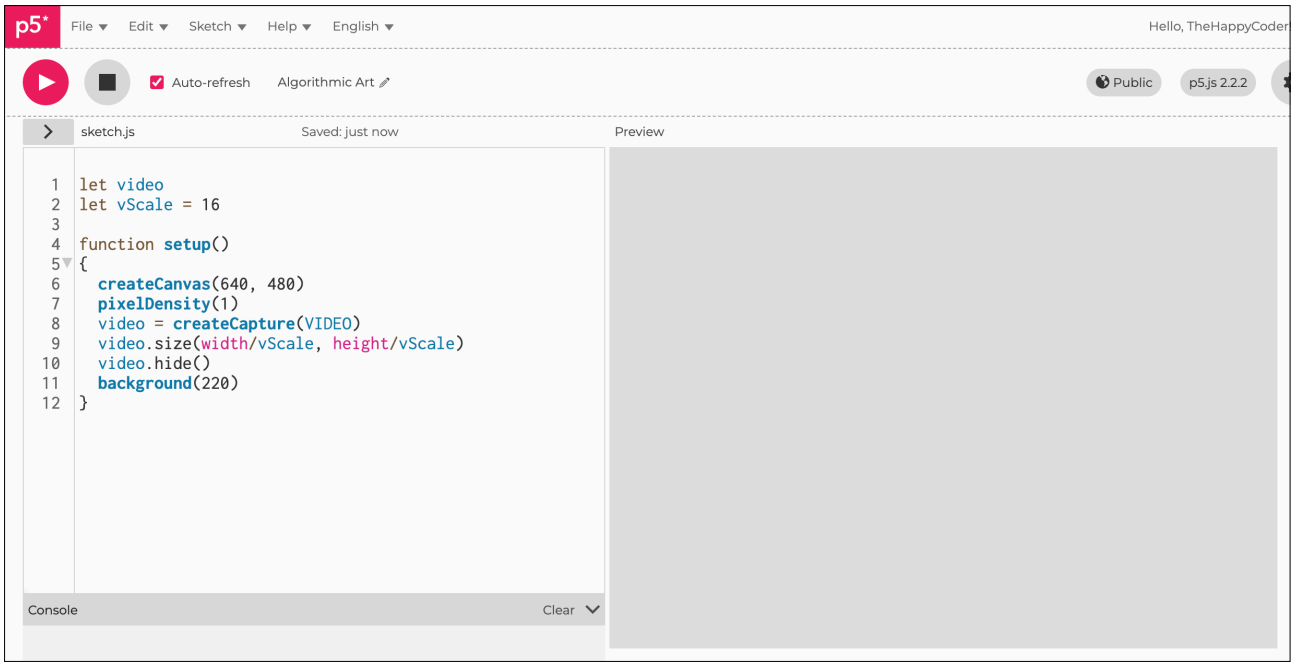
```
let video
let vScale = 16

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()
  }
}

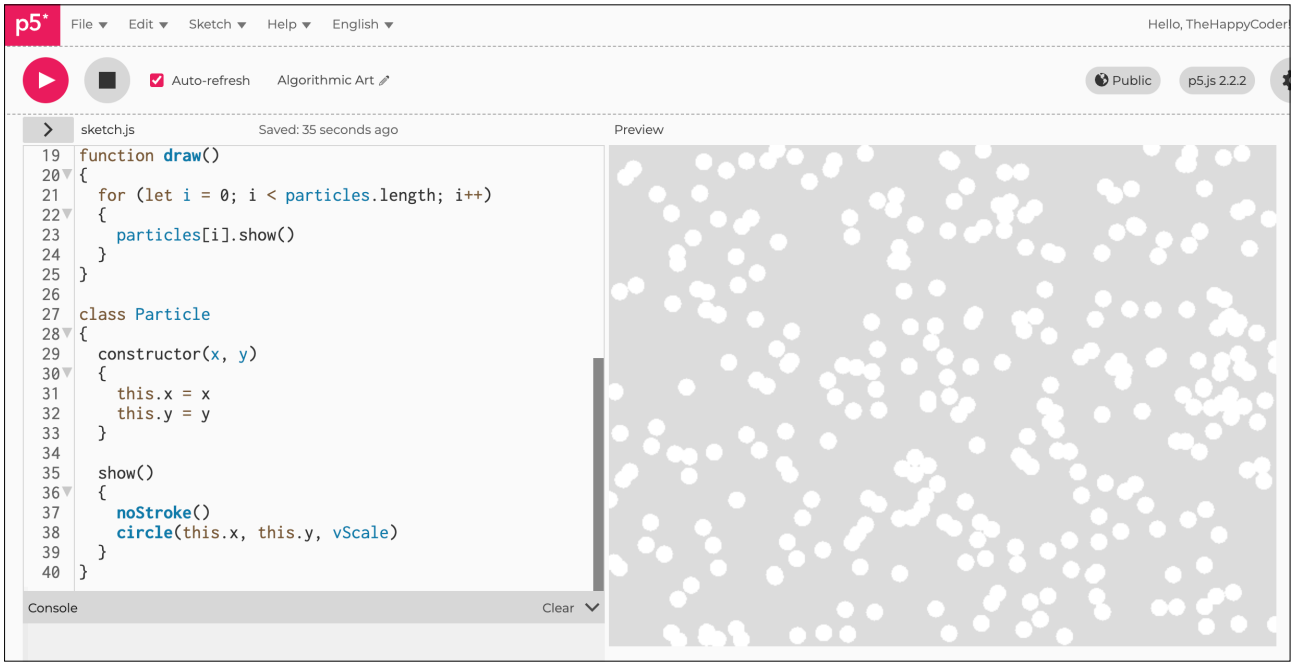
class Particle
{
  constructor(x, y)
  {
    this.x = x
    this.y = y
  }
```

```
  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()
  }
}

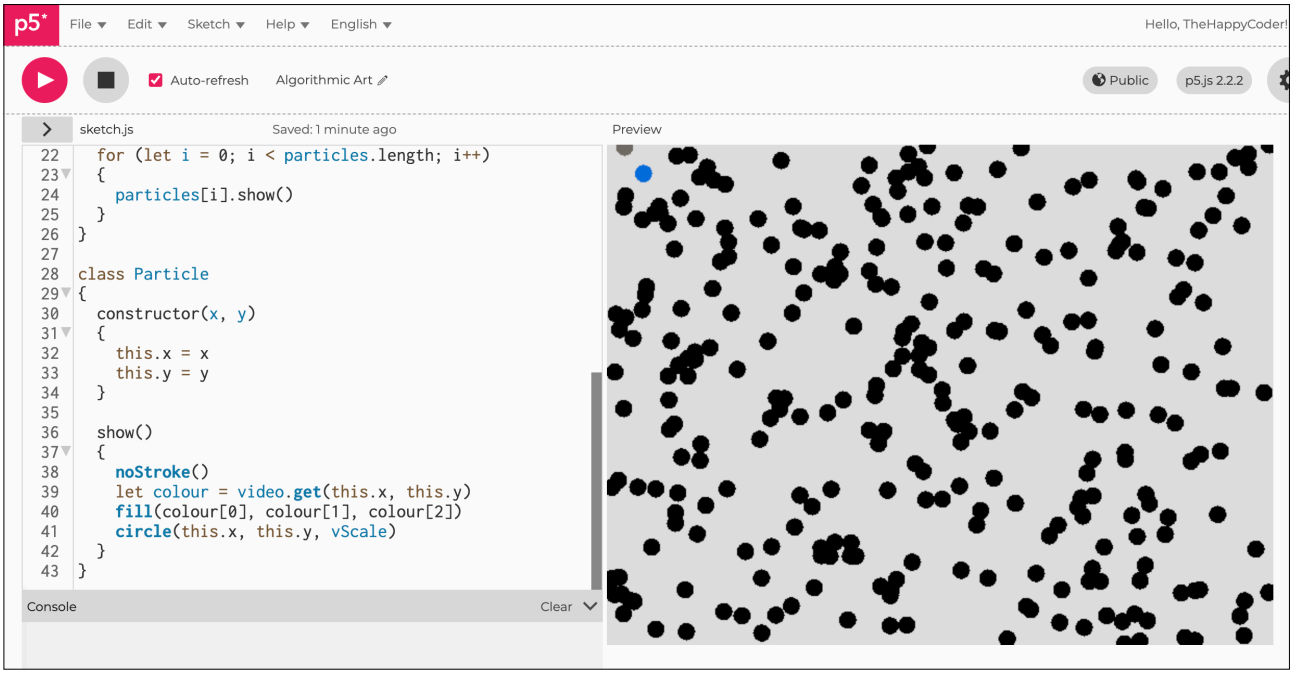
class Particle
{
  constructor(x, y)
  {
    this.x = x
    this.y = y
  }
```

```
  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
  }

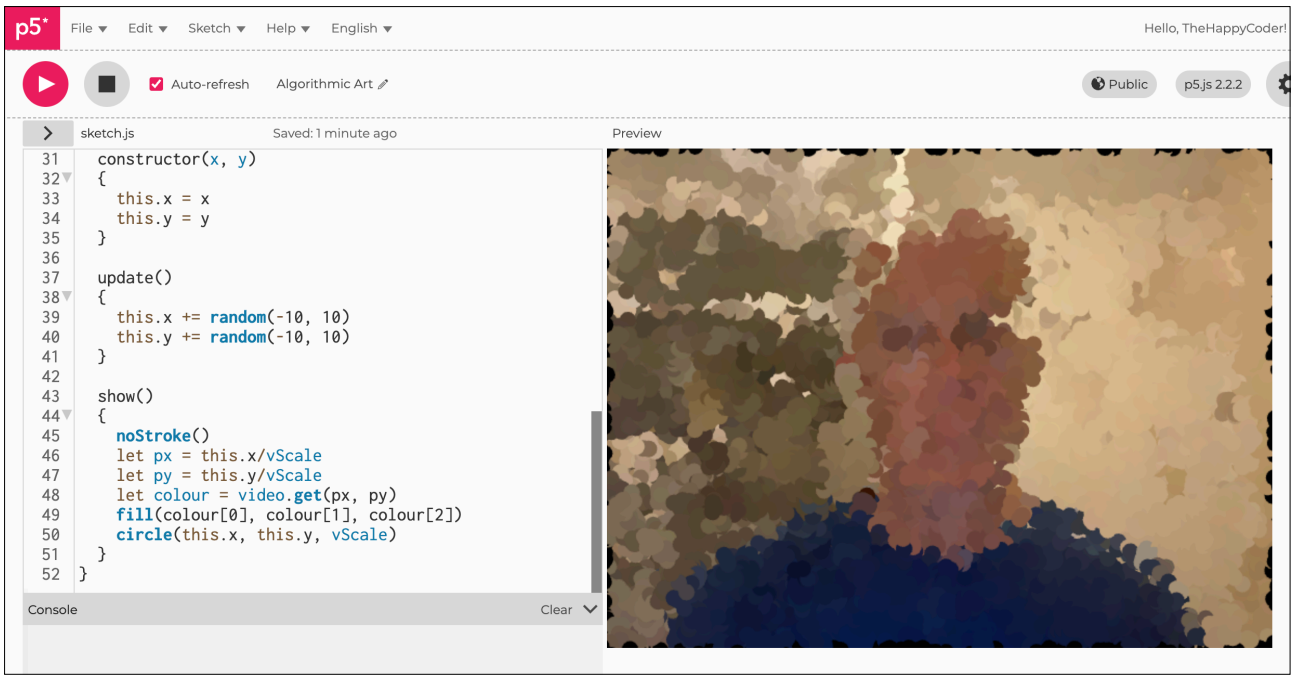
  update()
  {
    this.x += random(-10, 10)
    this.y += random(-10, 10)
  }

  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


p5*
File
Edit
Sketch
Help
English
Hello, TheHappyCoder!
Auto-refresh
Algorithmic Art
Public
p5.js 2.2.2
sketch.js
Saved: 1 minute ago
Preview
31 constructor(x, y)
32 {
33 this.x = x
34 this.y = y
35 }
36
37 update()
38 {
39 this.x += random(-10, 10)
40 this.y += random(-10, 10)
41 }
42
43 show()
44 {
45 noStroke()
46 let px = this.x/vScale
47 let py = this.y/vScale
48 let colour = video.get(px, py)
49 fill(colour[0], colour[1], colour[2])
50 circle(this.x, this.y, vScale)
51 }
52 }
Console
Clear



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

! Remove the scaling function.

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()
{
  // scale(-1, 1)
  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).