

*Algorithmic
Art
Workbook #12
Sound and
Json Files*



Table of Contents

MIT Licence	5
Workbook #12 Quick Content Summary	6
Module F Unit #4: image files	8
Creating a png (or jpg) image	9
Sketch F4.1 bubbles	10
Sketch F4.2 save canvas as a .png image	11
Sketch F4.3 png object image	12
Uploading the image	14
Sketch F4.4 uploading the bubbles png	16
Creating a gif	18
Sketch F4.5 rotating torus	19
Sketch F4.6 the space bar	21
Sketch F4.7 options and frames	22
Using mp4 capture	24
Sketch F4.8 capture index.html	25
Sketch F4.9 capture main sketch	26
Uploading the video	28
Sketch F4.10 playing the video	31
Commenting out in the index.html file	33
Module F Unit #5: the sound of music	35
Music quick summary	38
Sketch F5.1 uploading	39
Sketch F5.2 play automatically	41
Sketch F5.3 music loop on mouse	42
Sketch F5.4 slider for rate	43
Sketch F5.5 toggle play/pause	45
Sketch F5.6 jump	47
Sketch F5.7 music length	49
Get the volume	51
Sketch F5.8 getting the volume	52
Sketch F5.9 toggle button	54
Sketch F5.10 a bit of style	56
Sketch F5.11 newish sketch	58
Sketch F5.12 an array of data	60
Sketch F5.13 replacing the circle	62
Sketch F5.14 a line of dots	64
Sketch F5.15 a continuous line	66
Sketch F5.16 a rotary version	68

Module F Unit #6: the microphone	72
Sketch F6.1 the microphone	73
Sketch F6.2 the amplitude	75
Sketch F6.3 the volume	76
Sketch F6.4 a circle	78
Sketch F6.5 bouncing ball?	80
Module F Unit #7: telling the time	83
Sketch F7.1 every second counts	84
Time functions	86
Sketch F7.2 a second look	87
Sketch F7.3 what a year	89
Date functions	91
Sketch F7.4 milliseconds of time	92
Sketch F7.5 in the blink of an eye	94
Sketch F7.6 alternative modulo	96
Sketch F7.7 making a clock	98
Sketch F7.8 other radii	100
Sketch F7.9 the angles	102
Sketch F7.10 drawing the hands	104
Module F Unit #8: local storage	108
Sketch F8.1 sliders	109
Sketch F8.2 storing data	111
Sketch F8.3 getting the data	113
Sketch F8.4 replacing all the values	115
Module F Unit #9: using json files	118
Sketch F9.1 a bubble object	119
Creating a json file	121
Sketch F9.2 the json data file	122
Sketch F9.3 adapting the sketch	124
Sketch F9.4 more than one bubble	126
Sketch F9.5 a long winded way	128
Sketch F9.6 a more concise way	130
Using a pre-made .json file	131
Download and upload palette.json file	132
Sketch F9.7 console log	133
Sketch F9.8 palette number 10	135
Sketch F9.9 specific colour	136
Sketch F9.10 fill the circle	137
Sketch F9.11 cycle through the palette	139
Sketch F9.12 saveJSON()	141



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 #12 Quick Content Summary

Continuing the theme of different aspects of coding in p5.js, this workbook, like the previous one, jumps from one topic to another totally unrelated. These are just some of the tools available which you may (or not) find useful, but they are there if you need them.

We explore different media types, such as sounds and music. You will need a microphone for some of this, as well as music files (available on www.ElegantAI.org). There is much to get your teeth into and enjoy.



Algorithmic Art Module F Unit #4 Image Files



Module F Unit #4: image files

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

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



Creating a png (or jpg) image

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



Sketch F4.1 bubbles

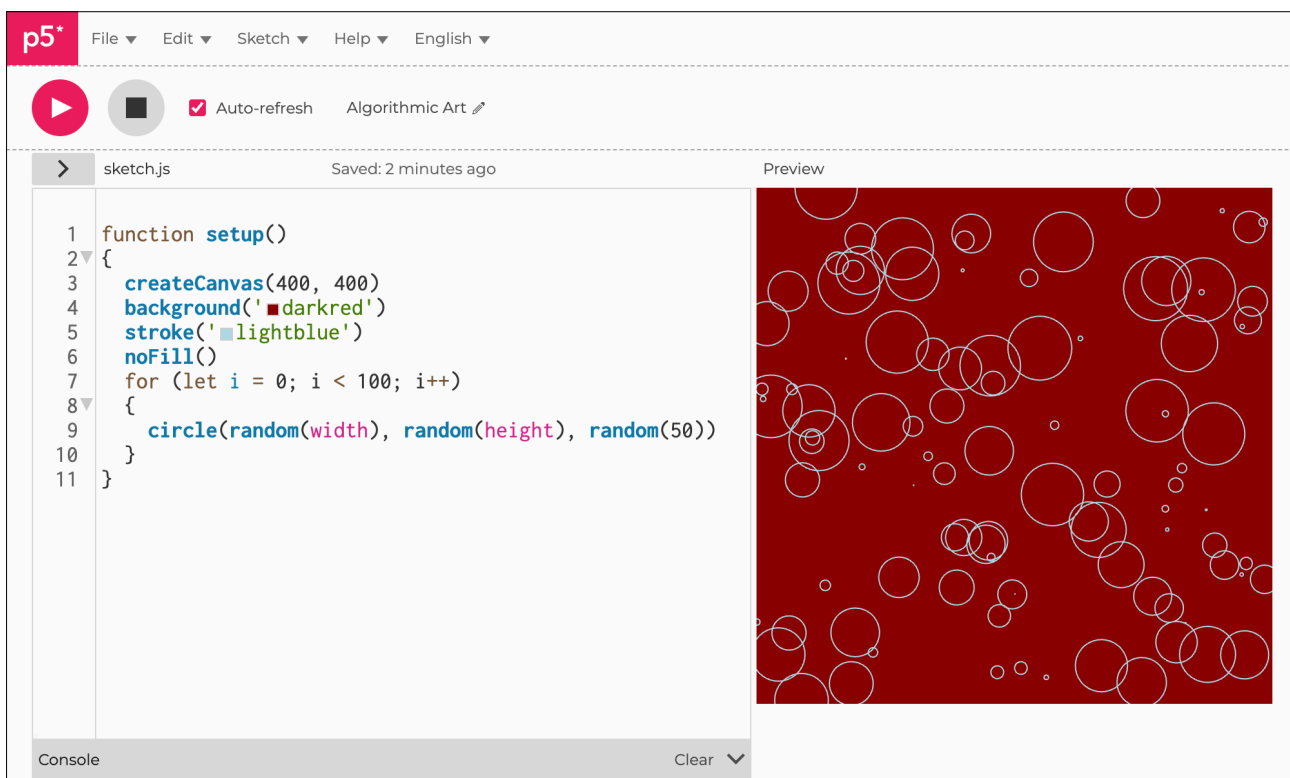
First, we will create a small canvas of bubbles.

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

Notes

100 randomly sized and positioned circles.

Figure F4.1





Sketch F4.2 save canvas as a .png image

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

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

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

Notes

You can also save this as a .jpg image by replacing png with jpg. You don't need to save it just yet.

Challenge

Save as a .jpg



Sketch F4.3 png object image

! We are removing the background.

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

```
let img

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

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

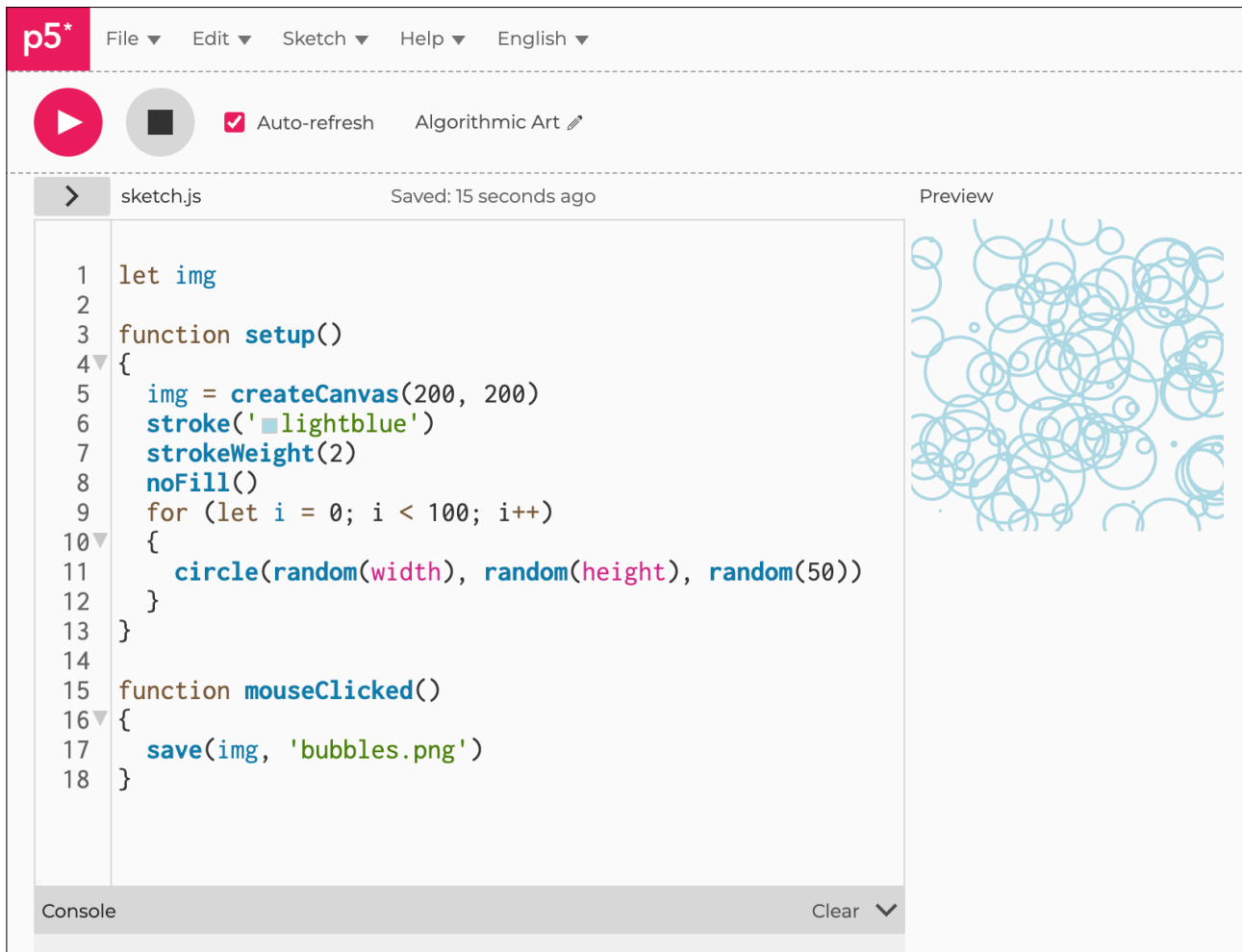
Notes

When you click on the canvas now, you will get another file called bubbles, probably with a suffix (**2** or more). Note where it is. Most likely in the downloads folder. We are now saving the image not the canvas as a whole

Challenge

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

Figure F4.3

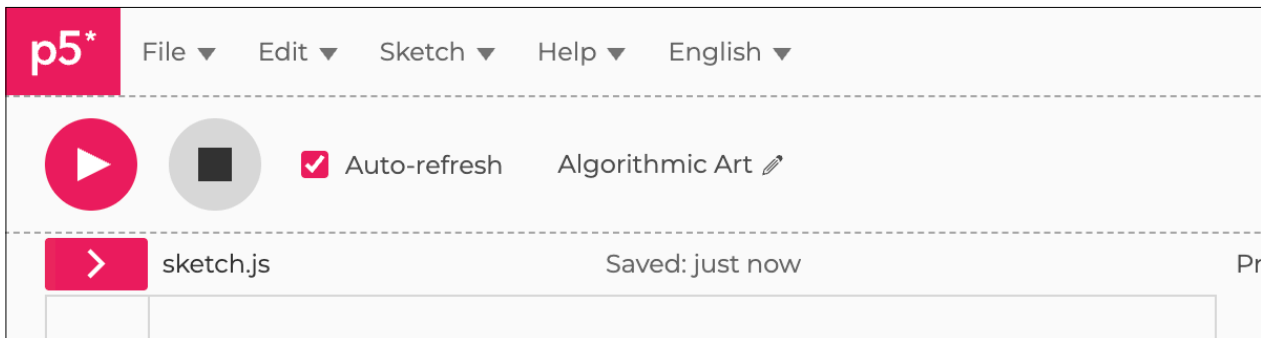




Uploading the image

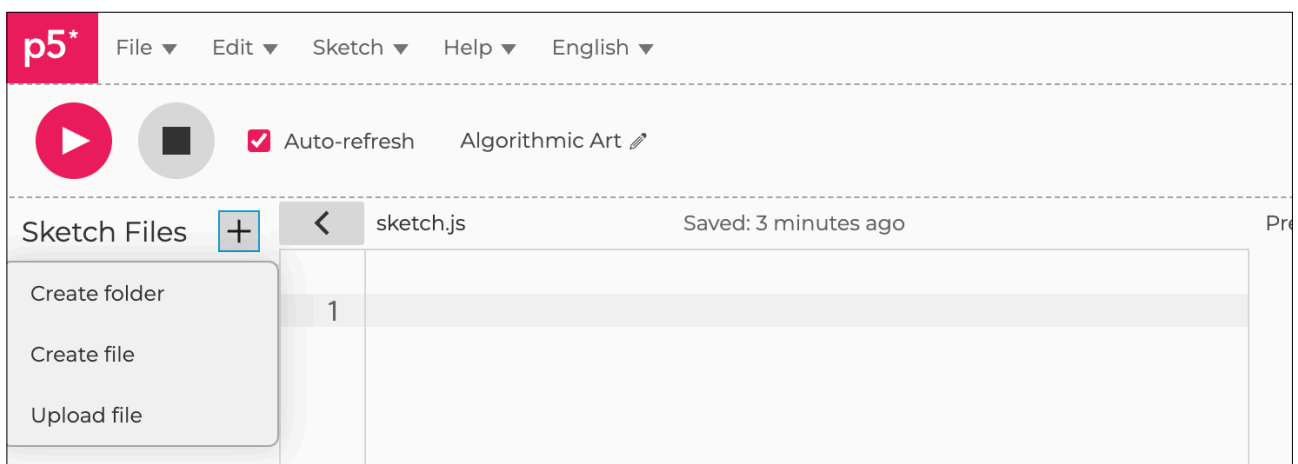
To add the image, we need to upload it as a file. To do that, we click on the side arrow next to where it says sketch.js, shown in Fig.1 below.

Figure 1: rectangular arrow button



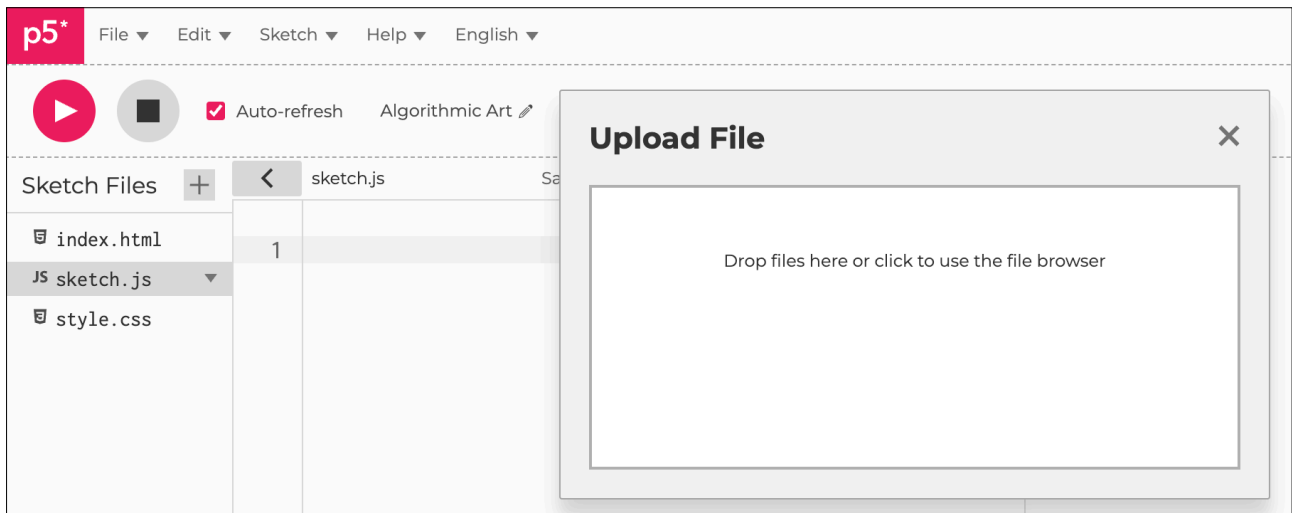
When you click on that arrow, it opens up a list of files. To add anything, you need to click on the + sign at the top, as shown in Figure 2 below.

Figure 2: the + means to add a file



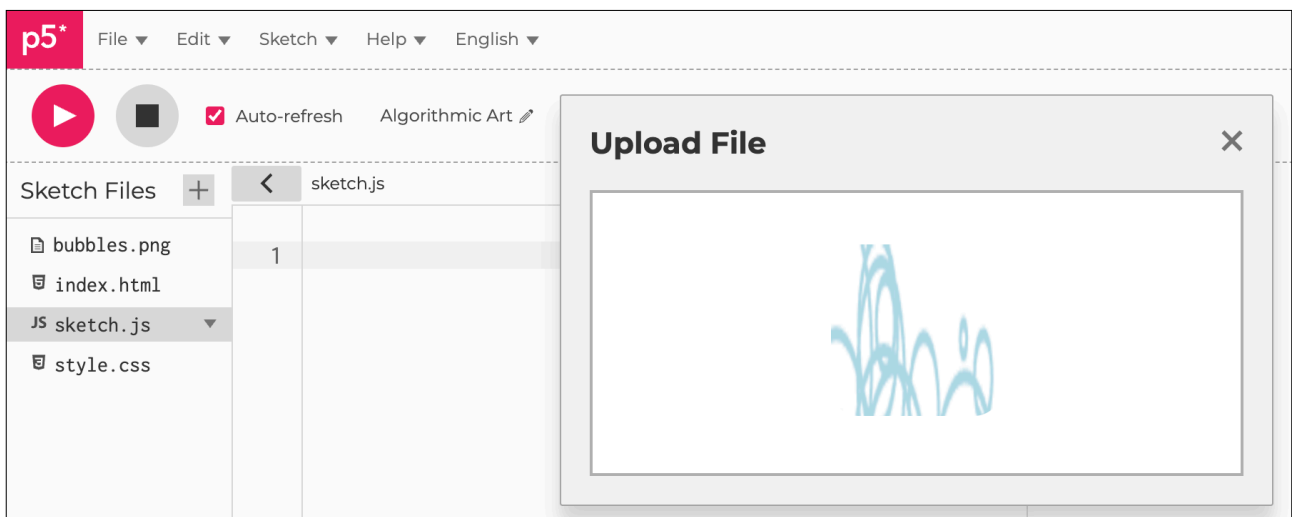
Click on the upload file, and a box appears. You can either click inside that box to browse to where your bubble file is or drag and drop it if it is easily accessible. See Fig. 3 below.

Figure 3: drop or browse window box



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

Figure 4: file (image) uploaded





Sketch F4.4 uploading the bubbles png

! Starting a new sketch after uploading the image **bubbles.png**.

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

```
let img

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

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

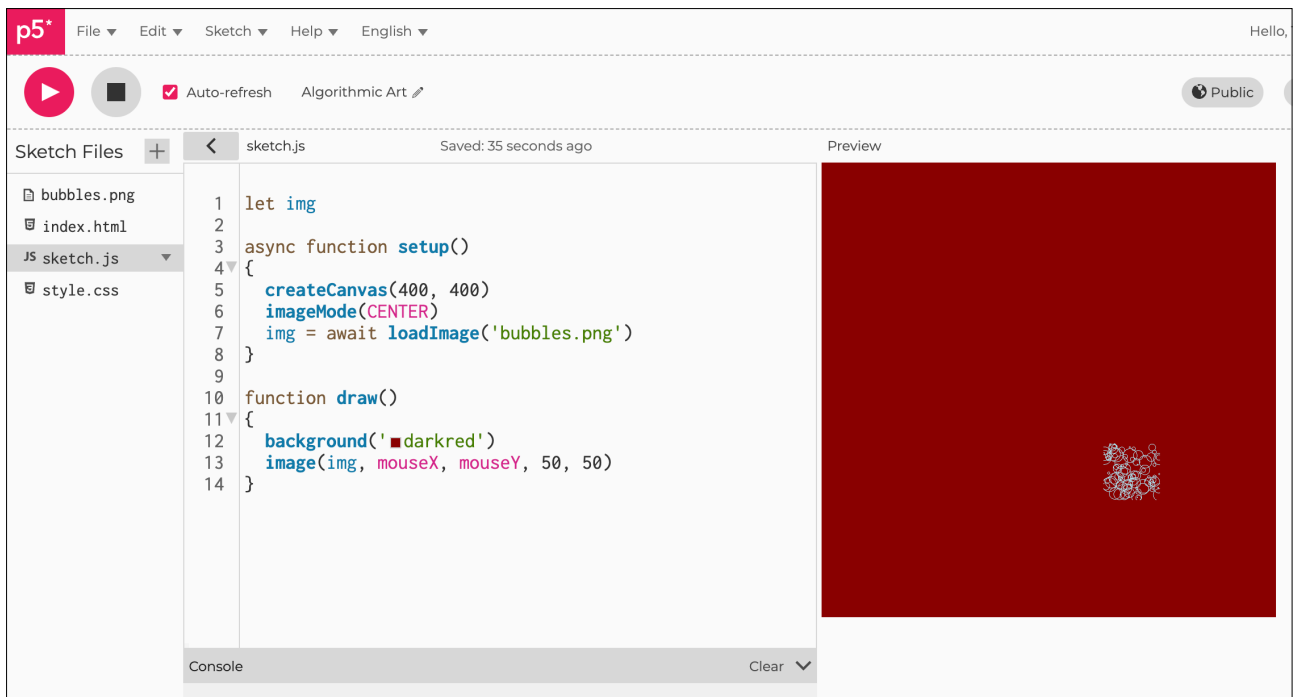
Notes

The main takeaway is that there is no background.

Challenge

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

Figure F4.4





Creating a gif

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



Sketch F4.5 rotating torus

! Of course a new sketch.

This should be pretty familiar to you.

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

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

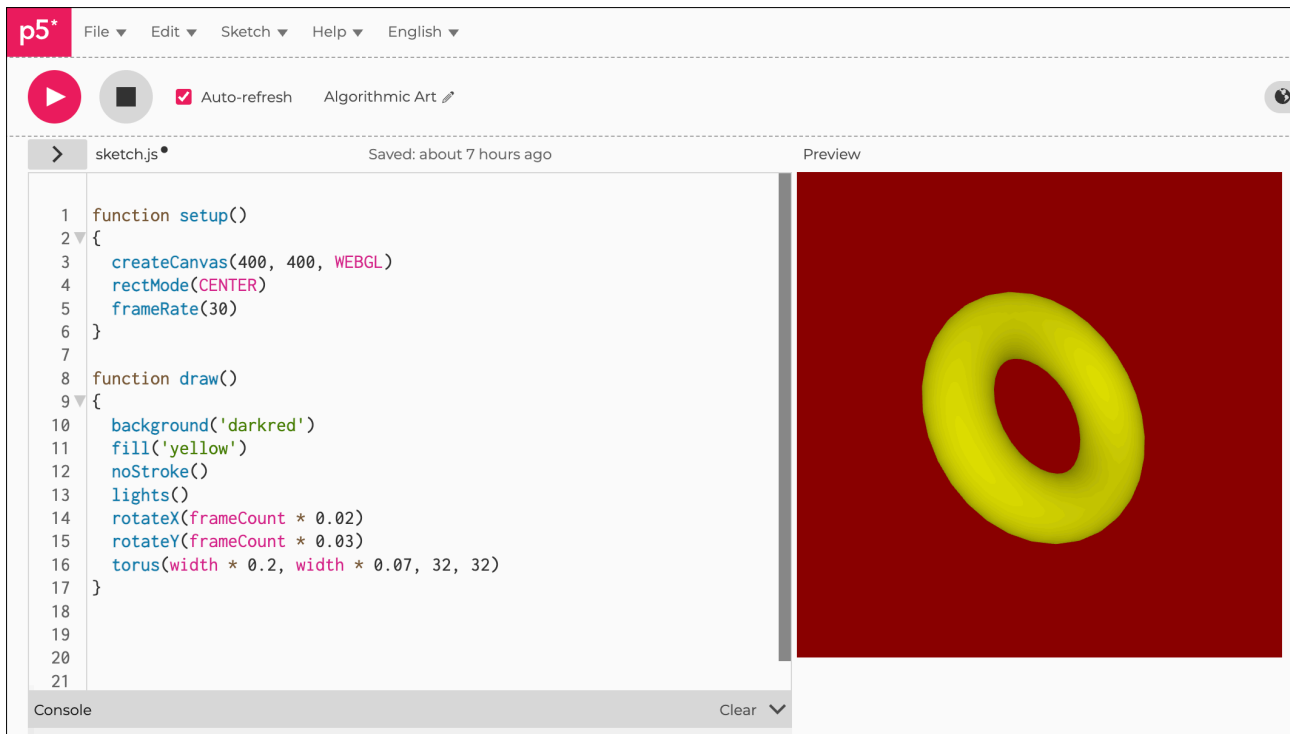
Notes

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

Challenge

Create your own.

Figure F4.5





Sketch F4.6 the space bar

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

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

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

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

Notes

Nothing will happen yet.



Sketch F4.7 options and frames

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

```
let frames = 120

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

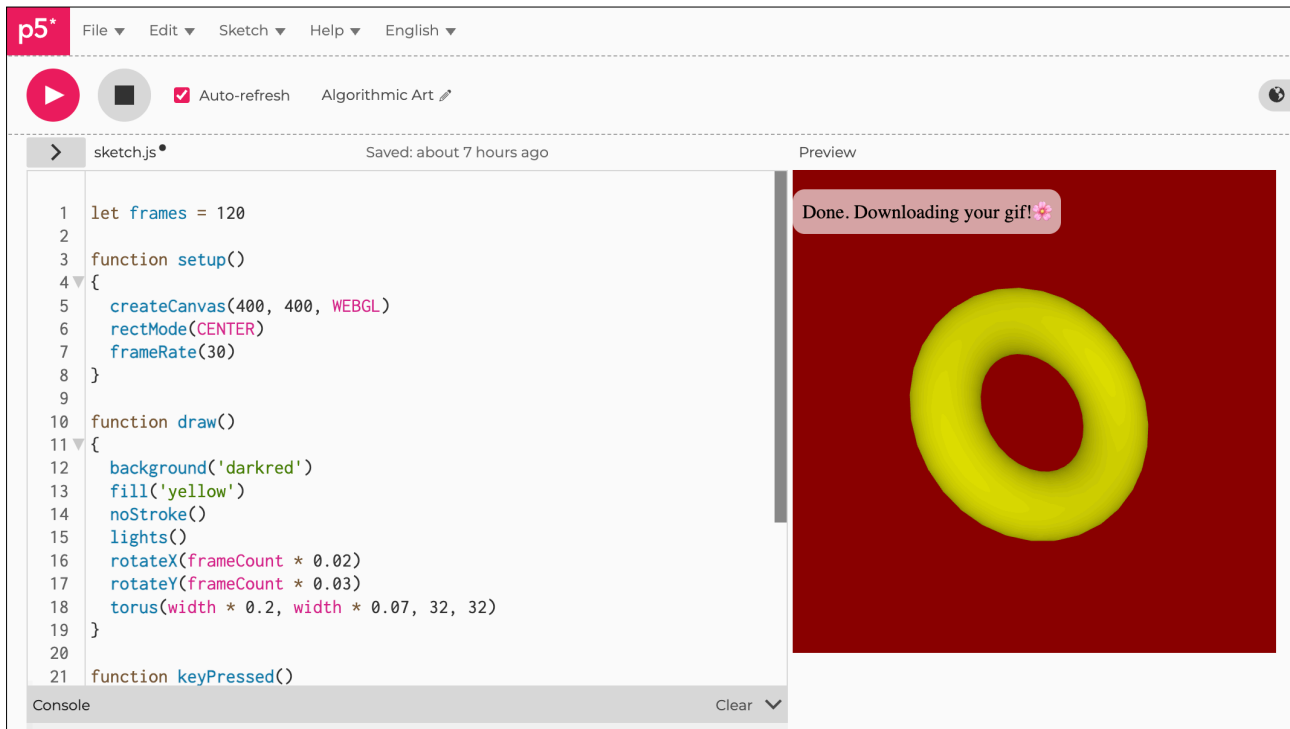
function draw()
{
  background('darkred')
  fill('yellow')
  noStroke()
  lights()
  rotateX(frameCount * 0.02)
  rotateY(frameCount * 0.03)
  torus(width * 0.2, width * 0.07, 32, 32)
}

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

Notes

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

Figure F4.7





Using mp4 capture

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



Sketch F4.8 capture index.html

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

index.html

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

Notes

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

Challenge

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



Sketch F4.9 capture main sketch

! Most of the sketch is the same but remove reference to frames and the function `keyPressed()`.

Using the same sketch as before. You will get a box on top of the canvas. Click on the circular button to commence recording and click on the square to stop. This box will not appear in the final video.

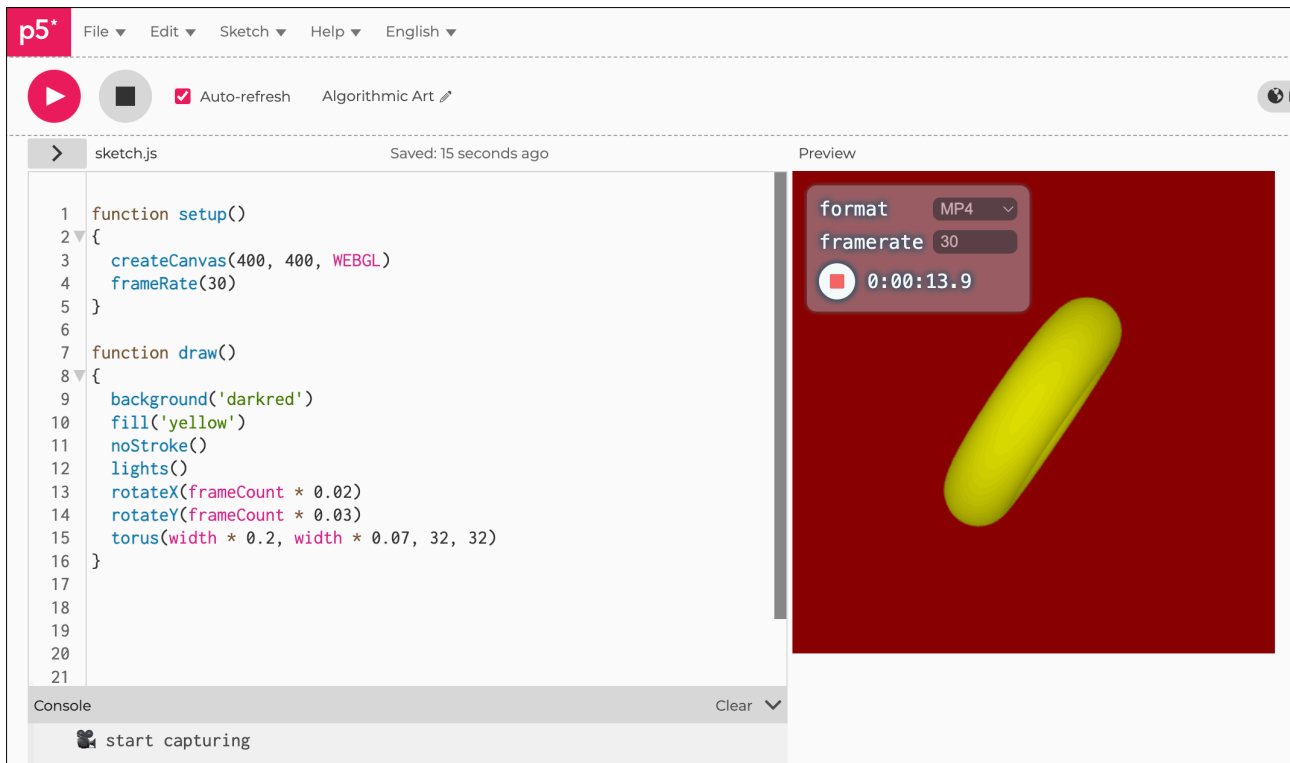
```
function setup()
{
  createCanvas(400, 400, WEBGL)
  frameRate(30)
}

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

Notes

You will get a small box on the canvas; it won't be in the final video. You can change the format, although I recommend doing that in the options setup in the [index.html](#) file. When you stop the recording it will be automatically downloaded to your computer.

Figure F4.9





Uploading the video

Just in case you are unsure, I will run through the process.

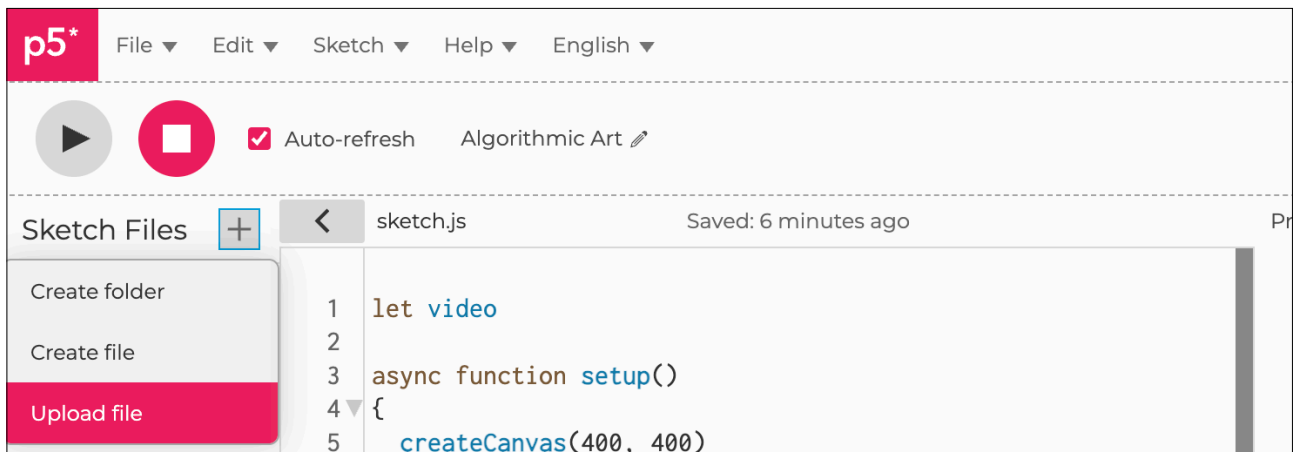
Click on the **+** button

Click on the **+** button



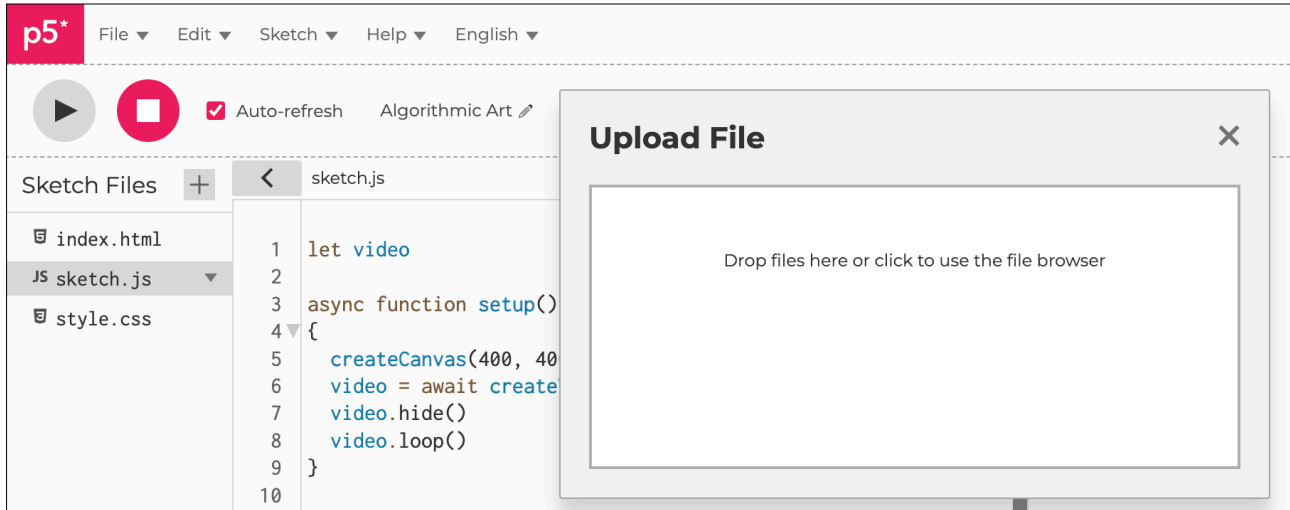
Click on the **Upload file** button

Upload file button



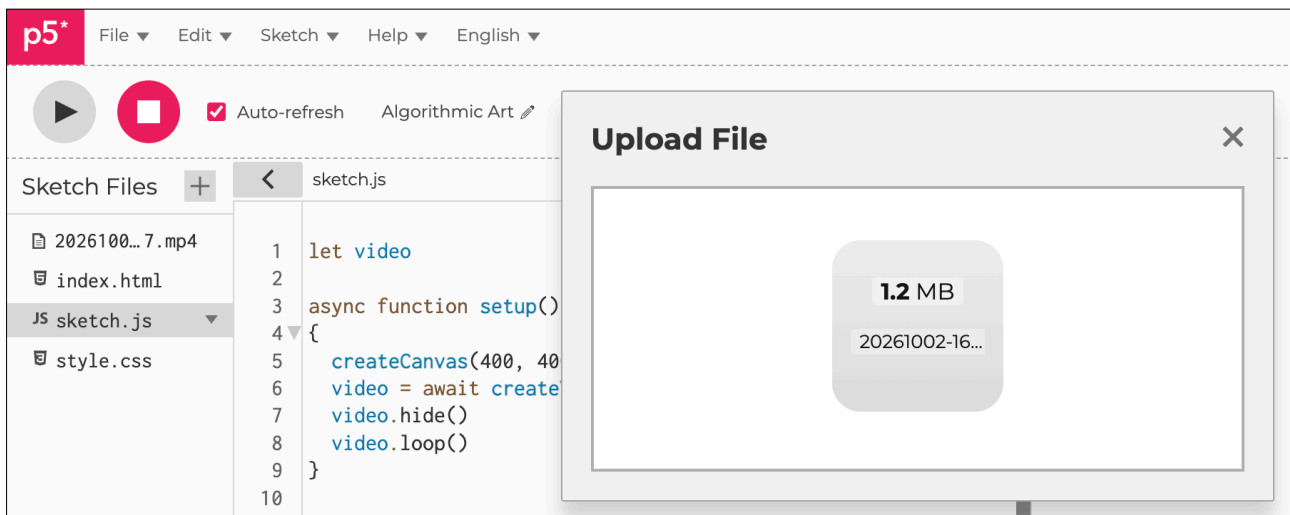
Drag and drop or browse to your mp4 video

Drag and drop or browse



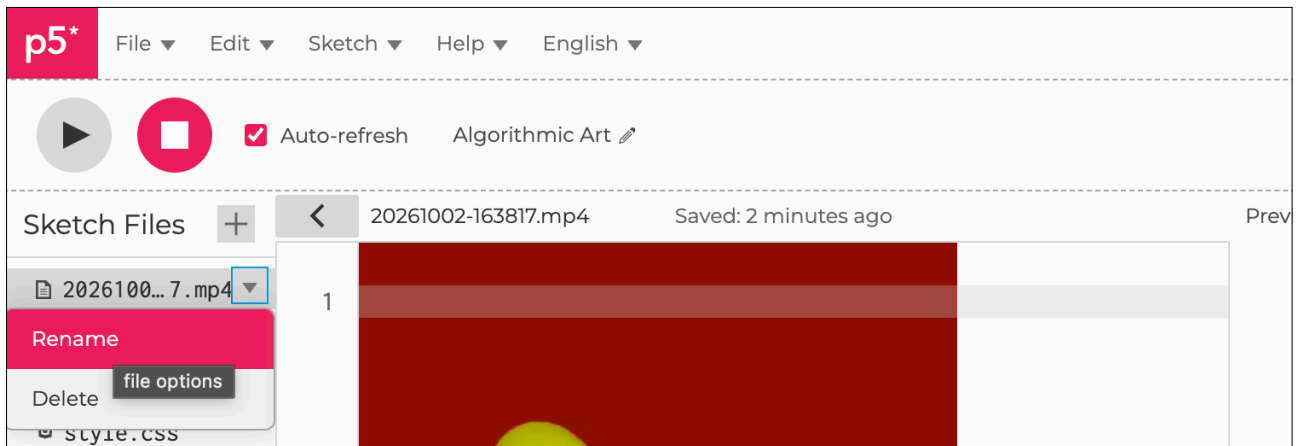
Upload completed

Upload complete



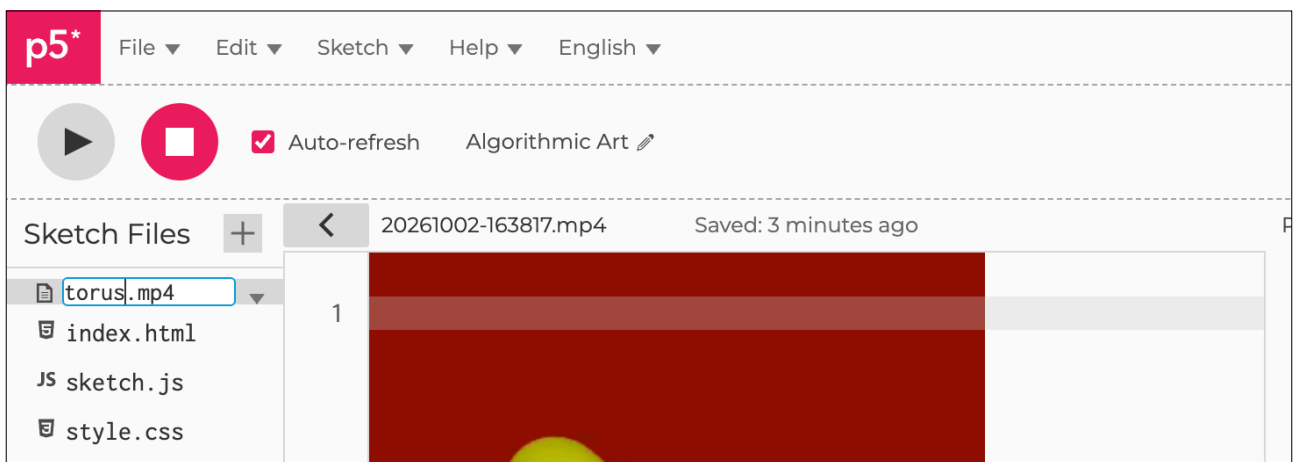
Rename the file

Renaming the file



New name given

torus.mp4



Then back to sketch.js



Sketch F4.10 playing the video

! We are starting a new sketch but first remove all the code in the `index.html` file referring to the `mp4 capture`. You can comment out the code if you want o keep it in for now.

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

```
let video

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

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

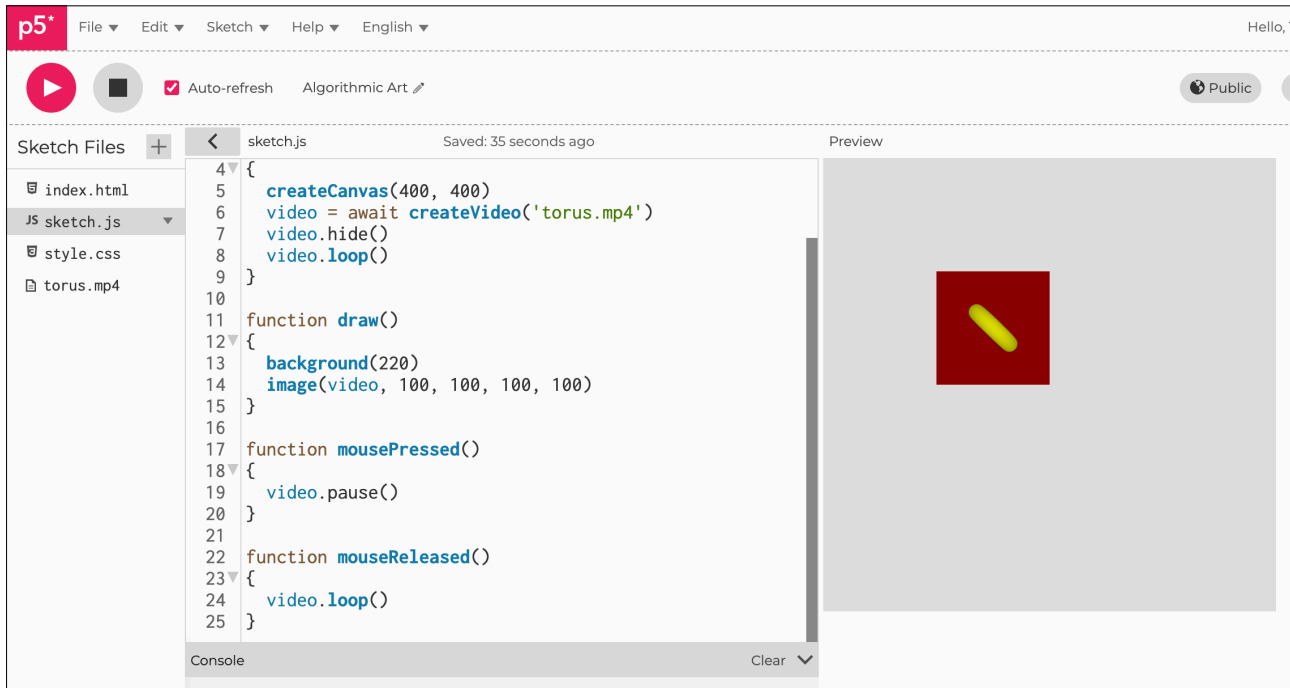
function mousePressed()
{
  video.pause()
}

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

Notes

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

Figure F4.10





Commenting out in the index.html file

If you highlight the code in the `index.html` file that you want to comment out then press `command` button and `/` at the same time it will comment out all that code for you. If you want to reinstate the commented out code then highlight all the code and press the `command` button and `/` simultaneously.

This is one of many short cuts. This one works for the code in in the p5.js sketch as well.

Algorithmic Art

Module F Unit #5

The Sound of Music



Module F Unit #5: the sound of music

The music I have provided is called **jam**. It is an **.mp3** file. You can use your own but be aware that there is a file size limit.

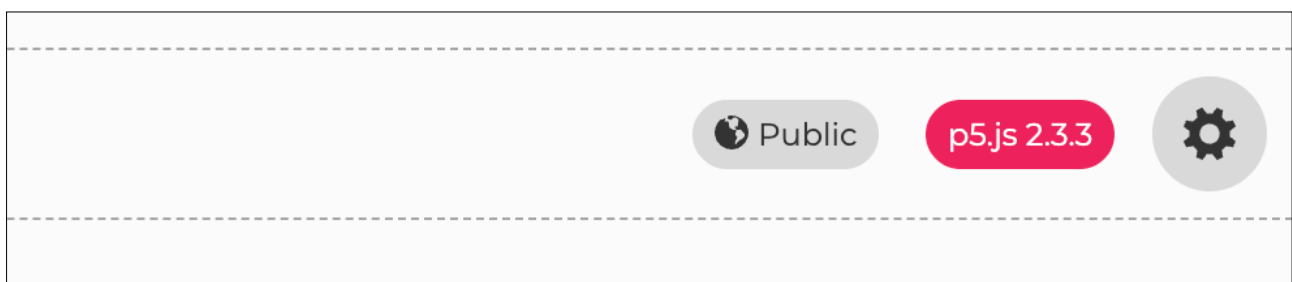
You can download the sound file from my website (www.ElegantAI.org)

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

This unit covers aspects such as visualising music and the microphone sound. Although we will be working with **v2.2.3** of **p5.js**, we do need to add a library that will allow us some compatibility with p5.js version 1. This is easy to do; remember to do it, even especially if you start a new sketch.

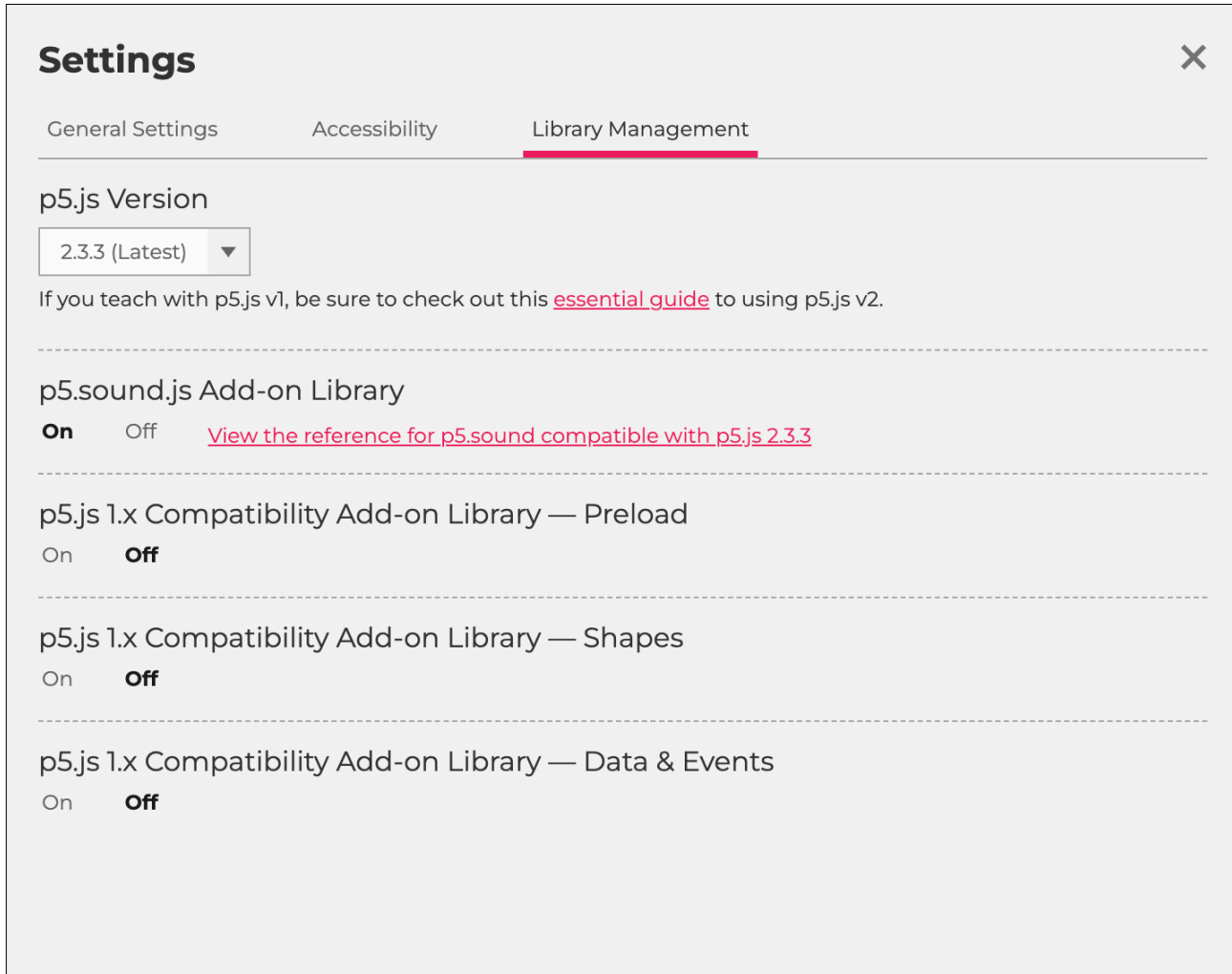
Click on the button that indicates the version you are using; yours might be **2.3.3** or something similar and change the version to **2.2.3**. See Fig.2.

Figure 2: the version button (top left)



Now select the p5.sound.js Add-on Library in **Settings**, as shown in Fig 3, remember to change the version **2.2.3** as well.

Figure 3: settings





Music quick summary

1. Download `jam.mp3` from website
2. Upload to the sketch
3. Change version to `2.2.3`
4. Select `p5.sound.js` Add-on Library



Sketch F5.1 uploading

The piece of music I have provided is approximately **23** seconds long. I have called it **jam** for simplicity. You upload it just the same as a video and an image.

```
let music

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

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

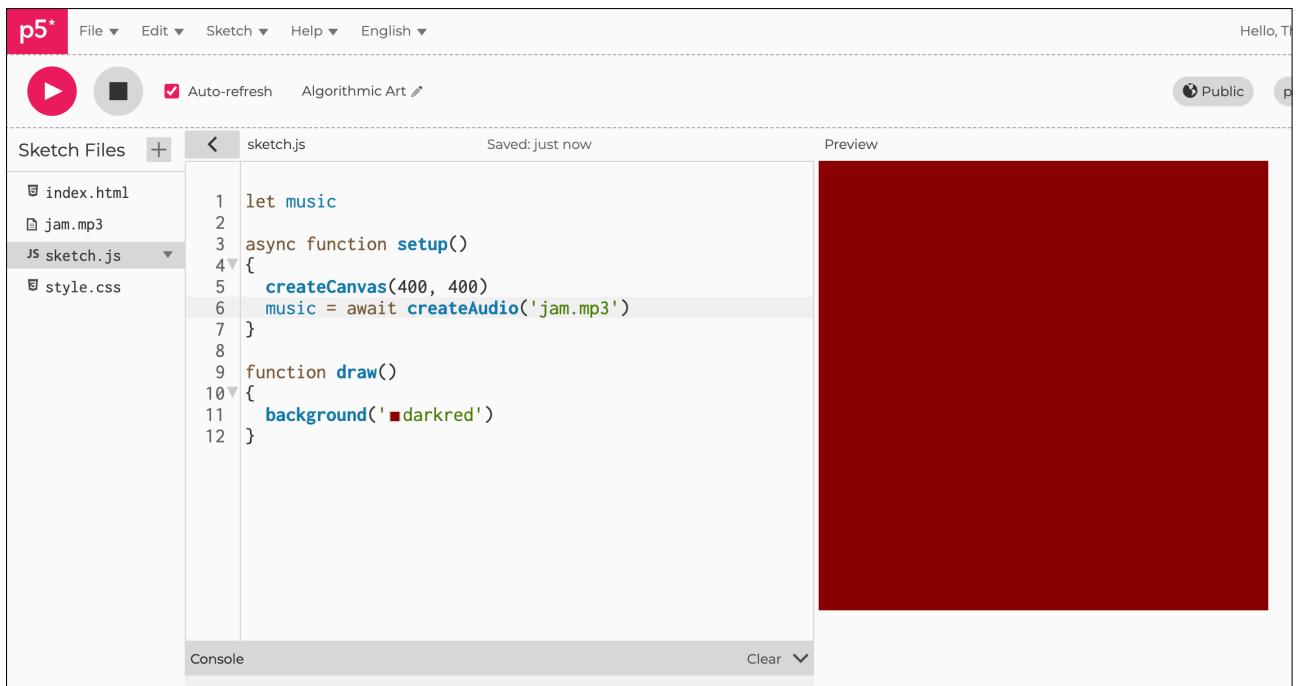
Notes

You can't hear anything yet.

Challenge

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

Figure F5.1





Sketch F5.2 play automatically

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

```
let music

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

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

Notes

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

Challenge

Try it in `draw()`.



Sketch F5.3 music loop on mouse

! Remove `play()` function.

Introducing `loop()` and `stop()`.

```
let music

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

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

function mousePressed()
{
  music.loop()
}

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

Notes

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

Challenge

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



Sketch F5.4 slider for rate

! If you are using the latest version you need to enable the **p5.sound.js** Add-on Library in settings under Library Management. The `loadSound()` function is not recognised in the latest versions of p5.js.

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

```
let music
let sliderRate

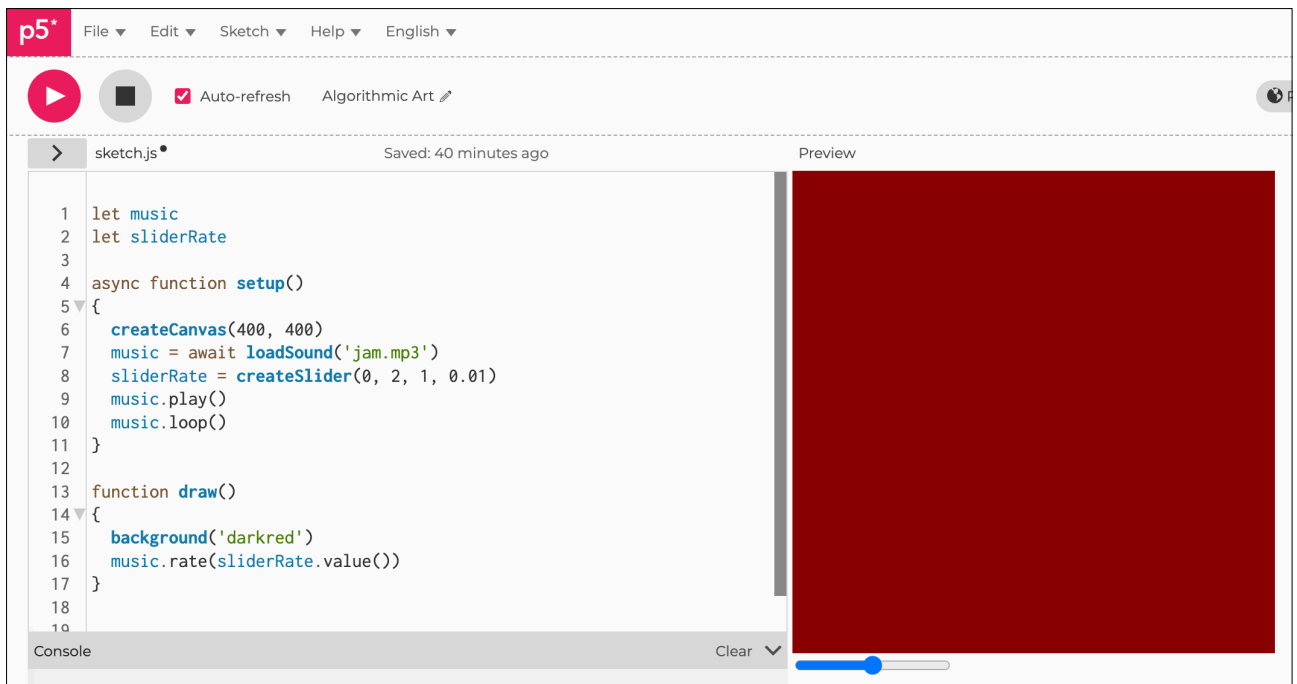
async function setup()
{
  createCanvas(400, 400)
  music = await loadSound('jam.mp3')
  sliderRate = createSlider(0, 2, 1, 0.01)
  music.play()
  music.loop()
}

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

Notes

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

Figure F5.4





Sketch F5.5 toggle play/pause

! Remove slider and the `draw()` function.

Just a simple toggle button.

```
let music
let button

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

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

Notes

Click to play or pause.

Figure F5.5





Sketch F5.6 jump

Jumps to a specific time in seconds. In this case, it jumps to the **20th** second of the **23-second-long** track. So if you jump just after it starts it will take you to near the end. This might be useful if it is a podcast.

```
let music
let button
let jumpButton

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

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

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

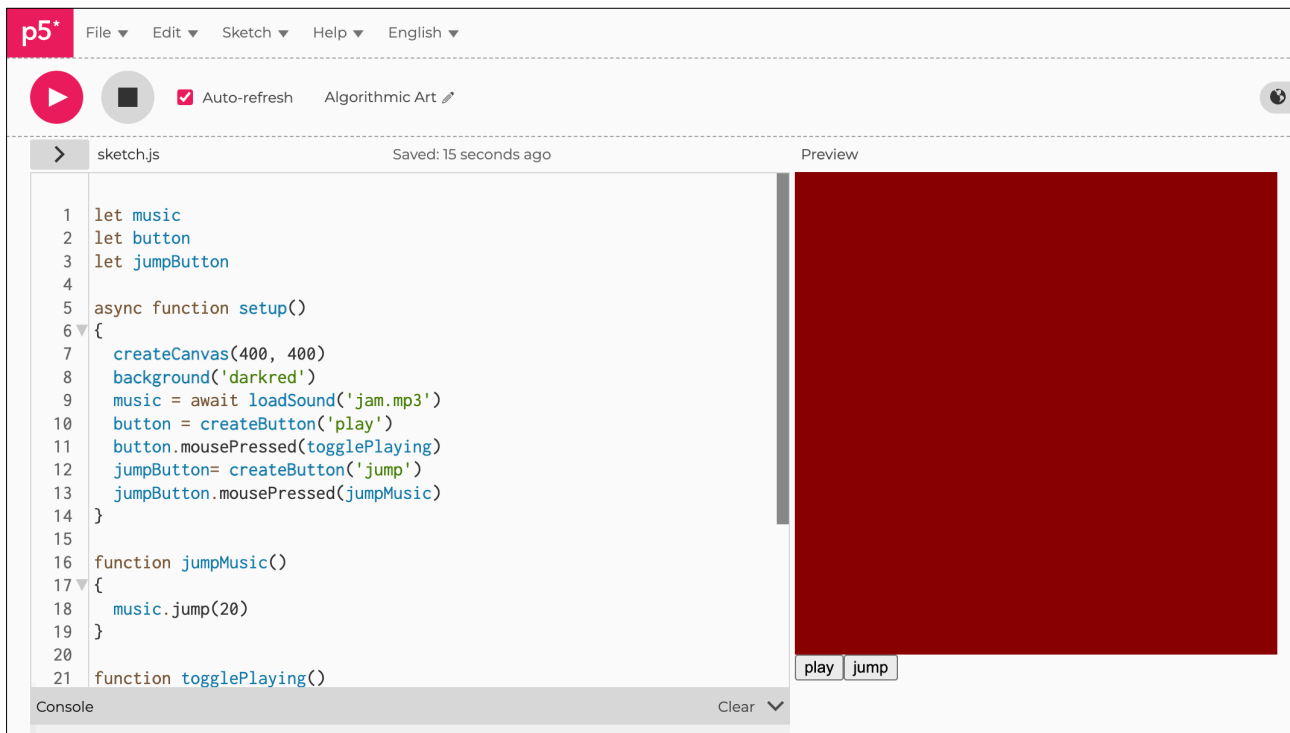
Notes

It might be hard to hear the jump.

Challenge

Try several jump buttons.

Figure F5.6





Sketch F5.7 music length

! Remove all functions except setup().

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

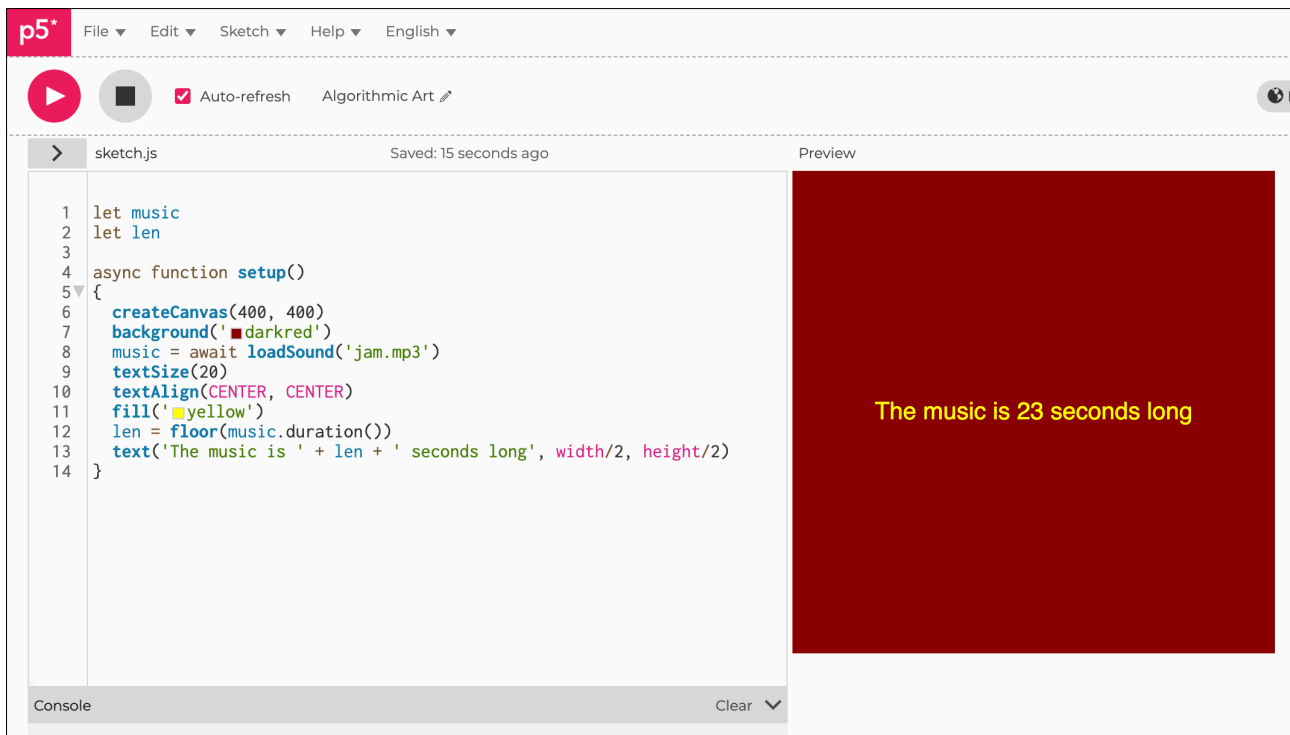
```
let music
let len

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

Notes

You should get the correct value of 23 seconds.

Figure F5.7





Get the volume

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



Sketch F5.8 getting the volume

! Quite a few changes.

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

```
let music
let amp
let vol

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

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

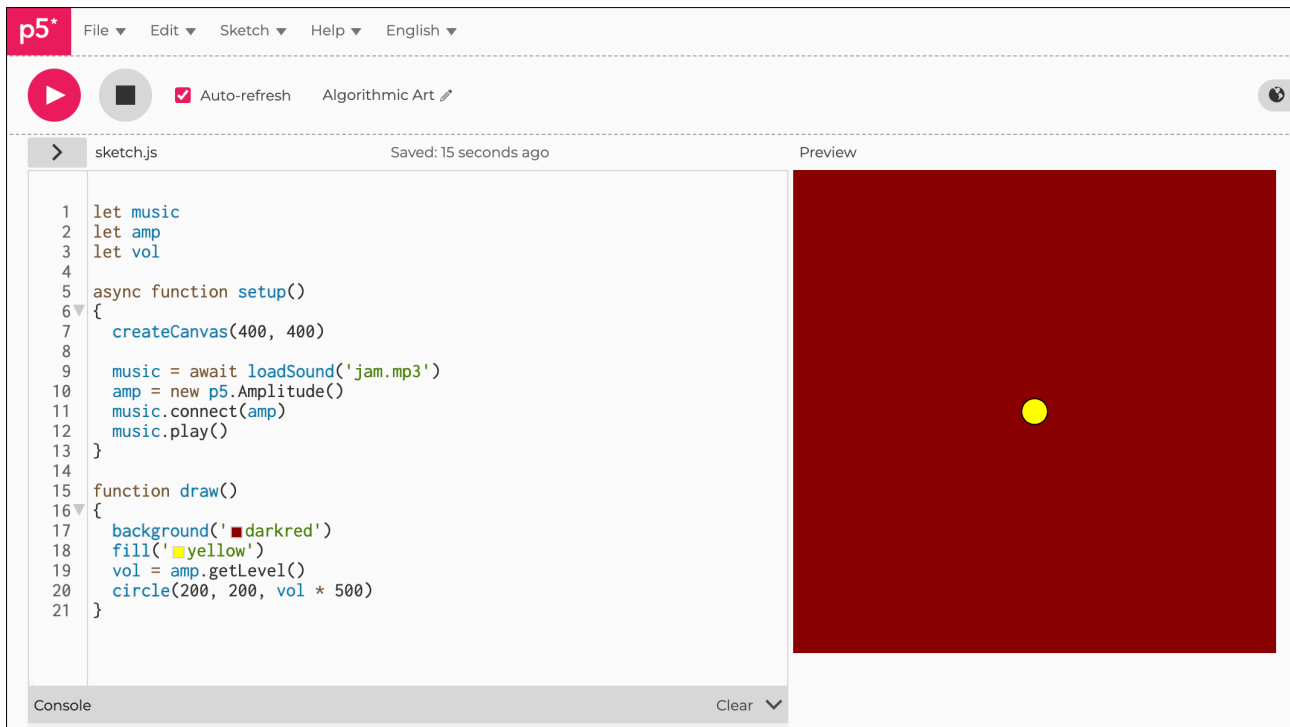
Notes

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

Challenge

Times it by the width instead.

Figure F5.8





Sketch F5.9 toggle button

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

```
let music
let amp
let vol
let button

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

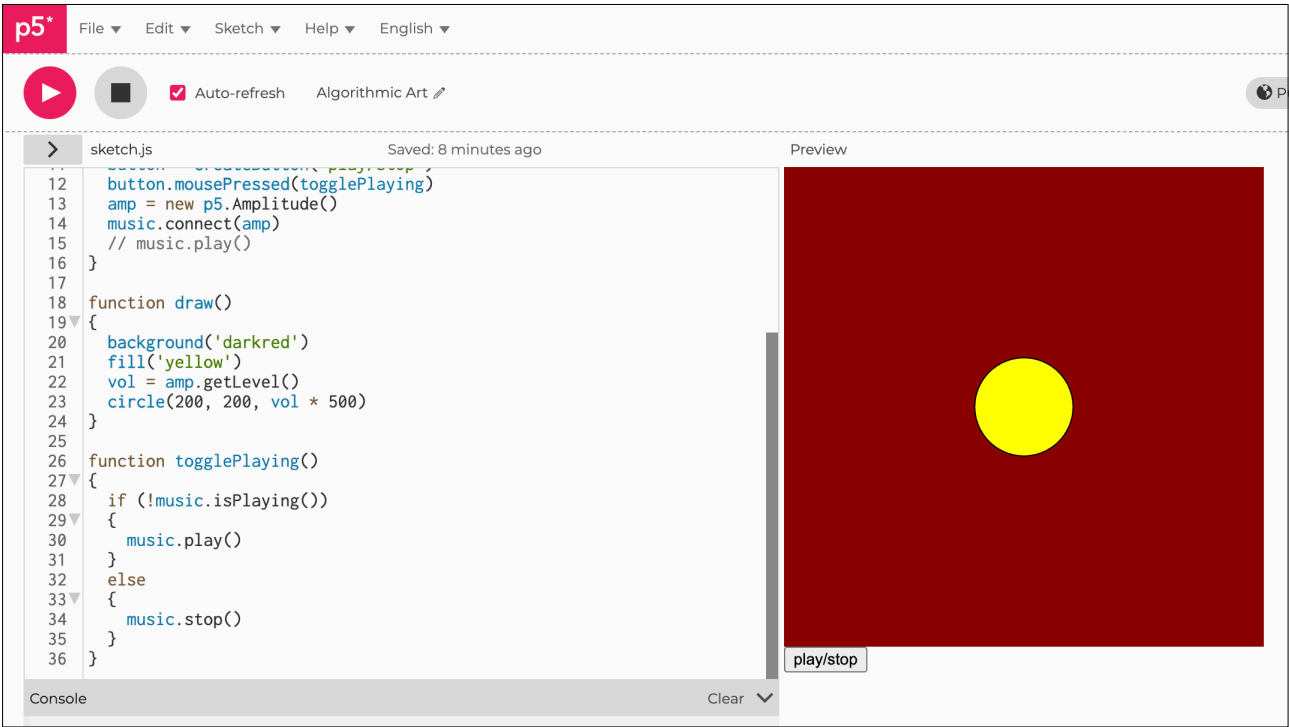
function draw()
{
  background('darkred')
  fill('yellow')
  vol = amp.getLevel()
  circle(200, 200, vol * 500)
}

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

```
}  
}
```

Notes
This is nothing new.

Figure F5.9





Sketch F5.10 a bit of style

! Moving the background into `setup()`.

Adding a bit of style.

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

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

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

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

```

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

```

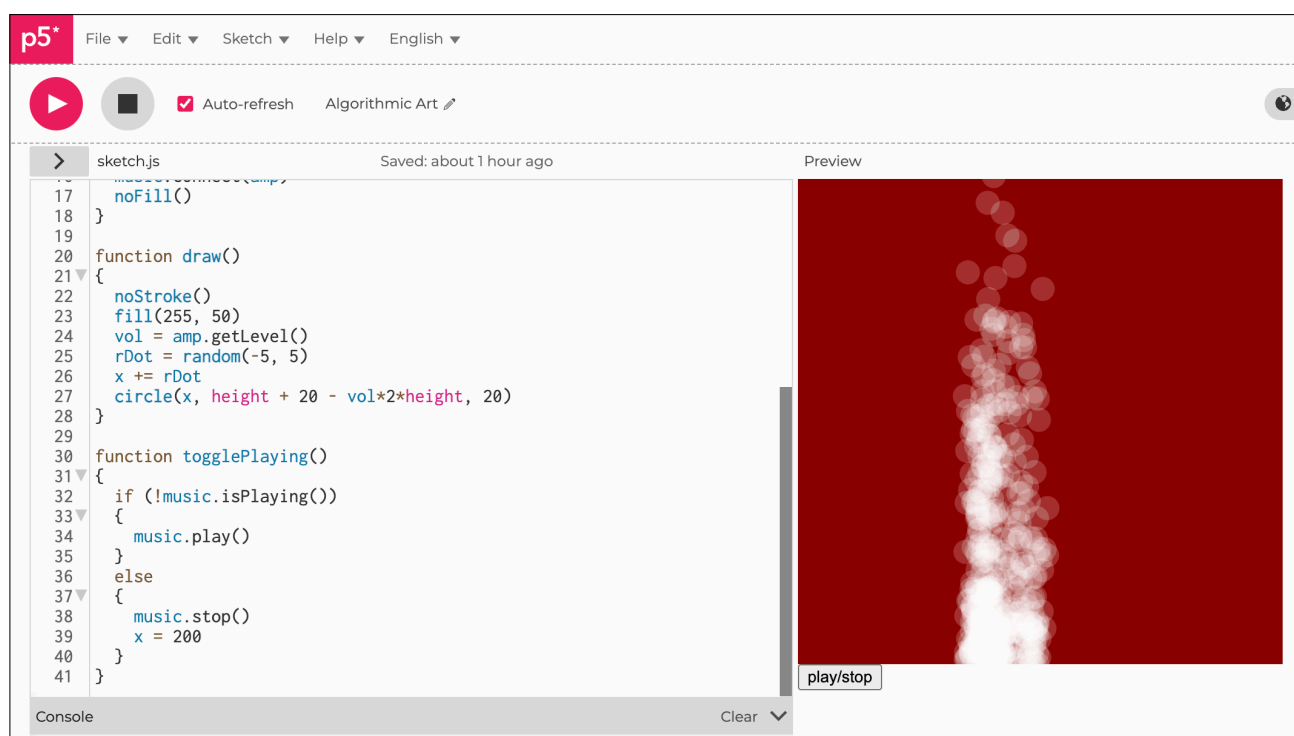
Notes

Should have a nice effect.

Challenge

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

Figure F5.10





Sketch F5.11 newish sketch

! Remove unnecessary code and put background into the `draw()` function. This is what we had previously.

```
let music
let amp
let vol
let button

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

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

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

Notes

The circle will now respond to the music.

Figure F5.11





Sketch F5.12 an array of data

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

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

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

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

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

```

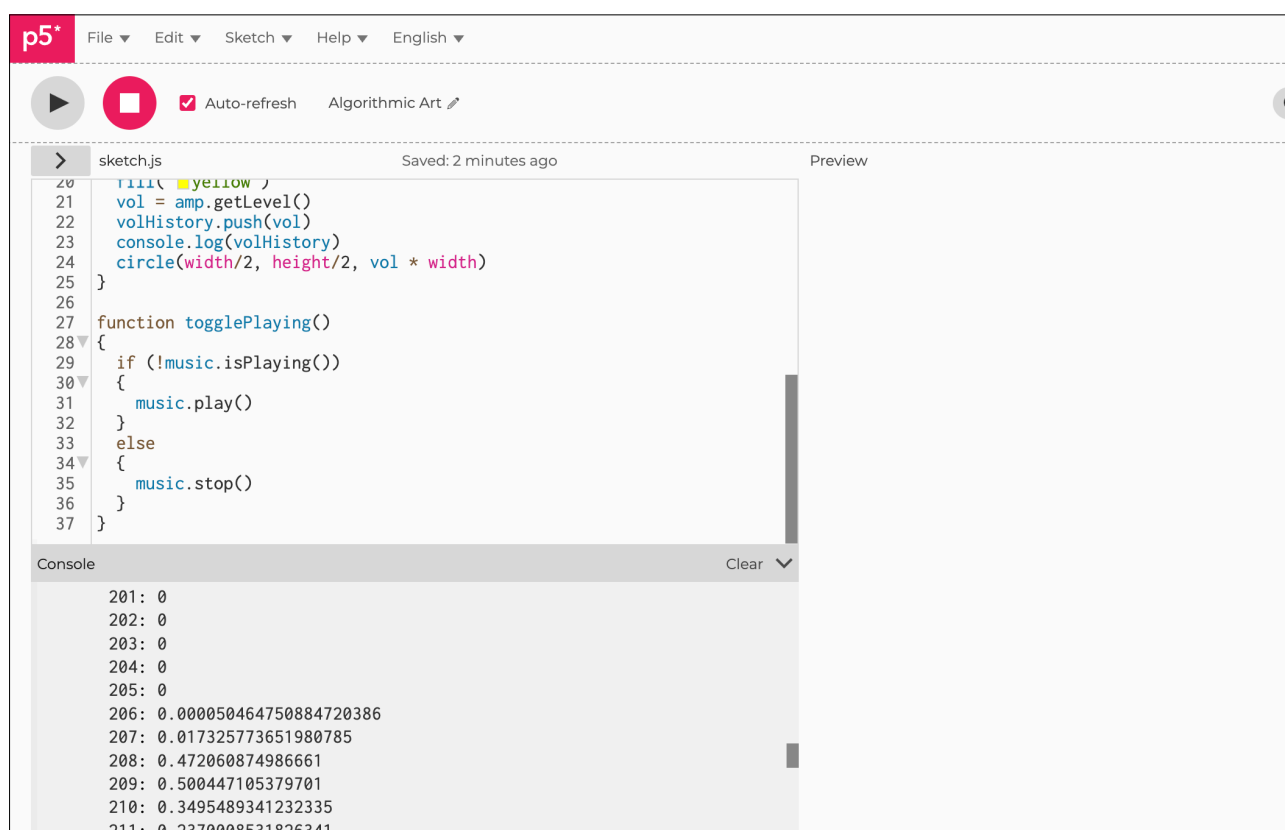
{
  music.stop()
}
}

```

Notes

You will have to pause the sketch and scroll back through the console to see the data. We get a result in the console showing that we are increasing the number of data points; this continues even though the music stops because it will still receive 0.

Figure F5.12





Sketch F5.13 replacing the circle

! Remove the line of code for the circle and the `console.log()`.

Instead, we are going to draw points for the values of the volume.

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

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

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

function togglePlaying()
{

```

```

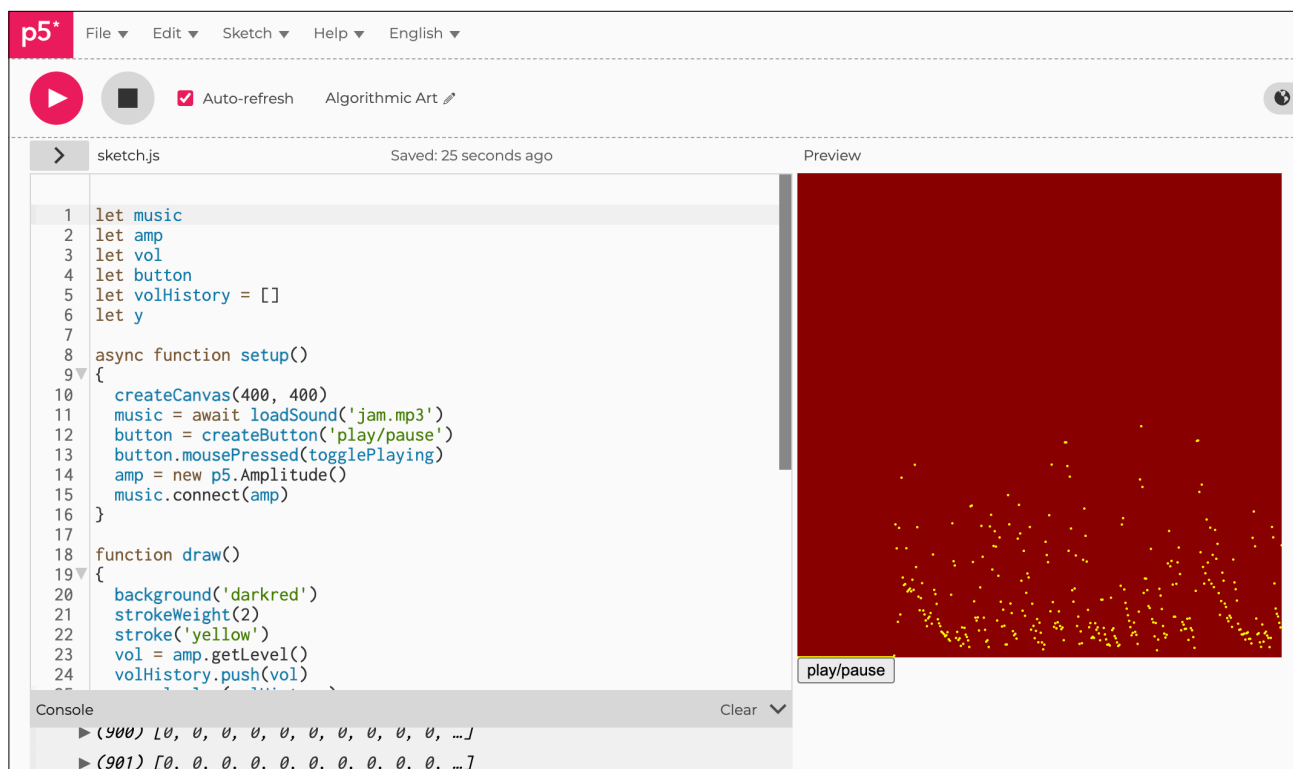
if (!music.isPlaying())
{
  music.play()
}
else
{
  music.stop()
}
}

```

Notes

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

Figure F5.13





Sketch F5.14 a line of dots

! If you haven't already use version **2.2.3** otherwise it will cut off after about **200** pixels on the canvas for some reason.

Replacing the `point()` with `vertex()`, we can draw a line joining the dots.

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

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

function draw()
{
  background('darkred')
  strokeWeight(2)
  stroke('yellow')
  noFill()

  vol = amp.getLevel()
  volHistory.push(vol)

  beginShape()
  for (let i = 0; i < volHistory.length; i++)
  {
    y = map(volHistory[i], 0, 1, height, 0)
    vertex(i, y)
  }
  endShape()
```

```

}

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

```

Notes

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

Figure F5.14





Sketch F5.15 a continuous line

Now the graph scrolls.

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

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

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

```

    volHistory.splice(0, 1)
  }
}

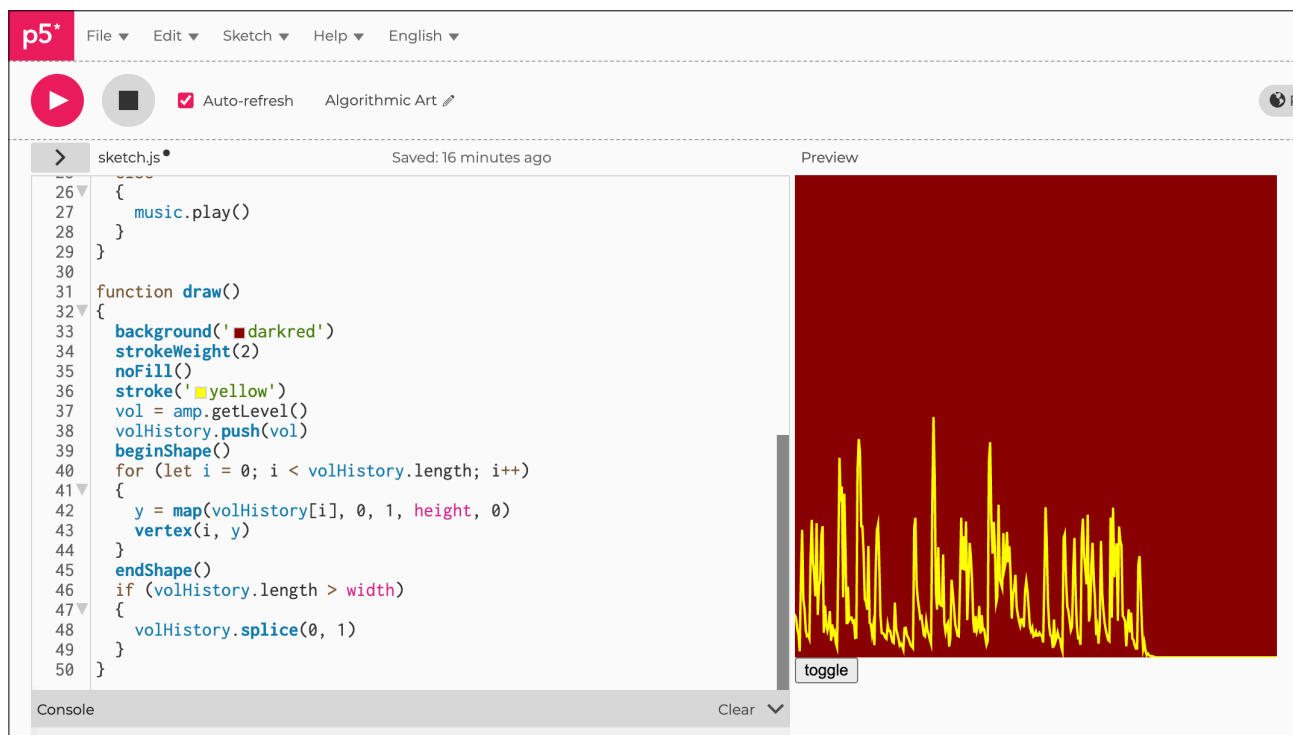
function togglePlaying()
{
  if (!music.isPlaying())
  {
    music.play()
  }
  else
  {
    music.stop()
  }
}

```

Notes

It continues to move even when the music stops.

Figure F5.15





Sketch F5.16 a rotary version

The line is now rotating.

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

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

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

```

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

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

```

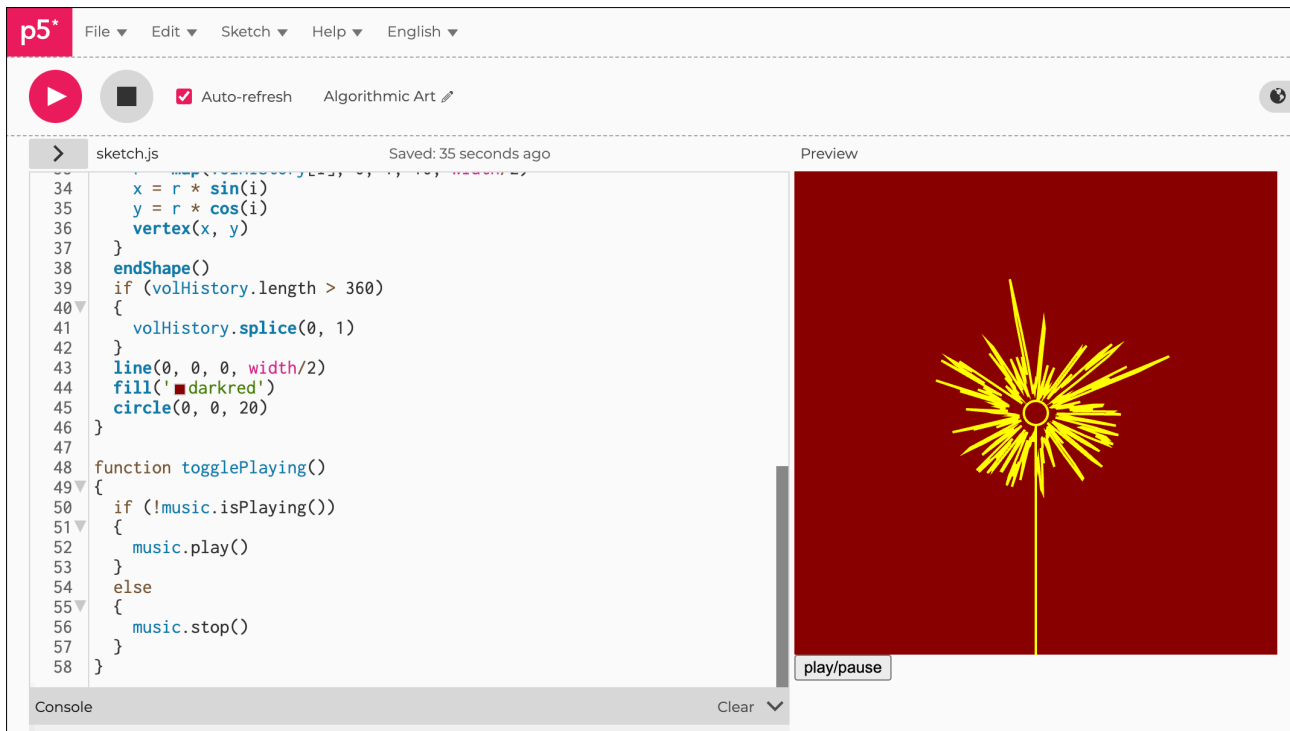
Notes

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

Challenge

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

Figure F5.16



Algorithmic
Art
Module F
Unit #6
Microphone



Module F Unit #6: the microphone

This unit covers aspects such as visualising music and the microphone sound. We still need to use version **2.2.3** and the **p5.sound.js** Add-on Library as before.



Sketch F6.1 the microphone

! Our starting sketch.

You may need to give your computer permission to use the microphone for **p5.js**.

```
let mic

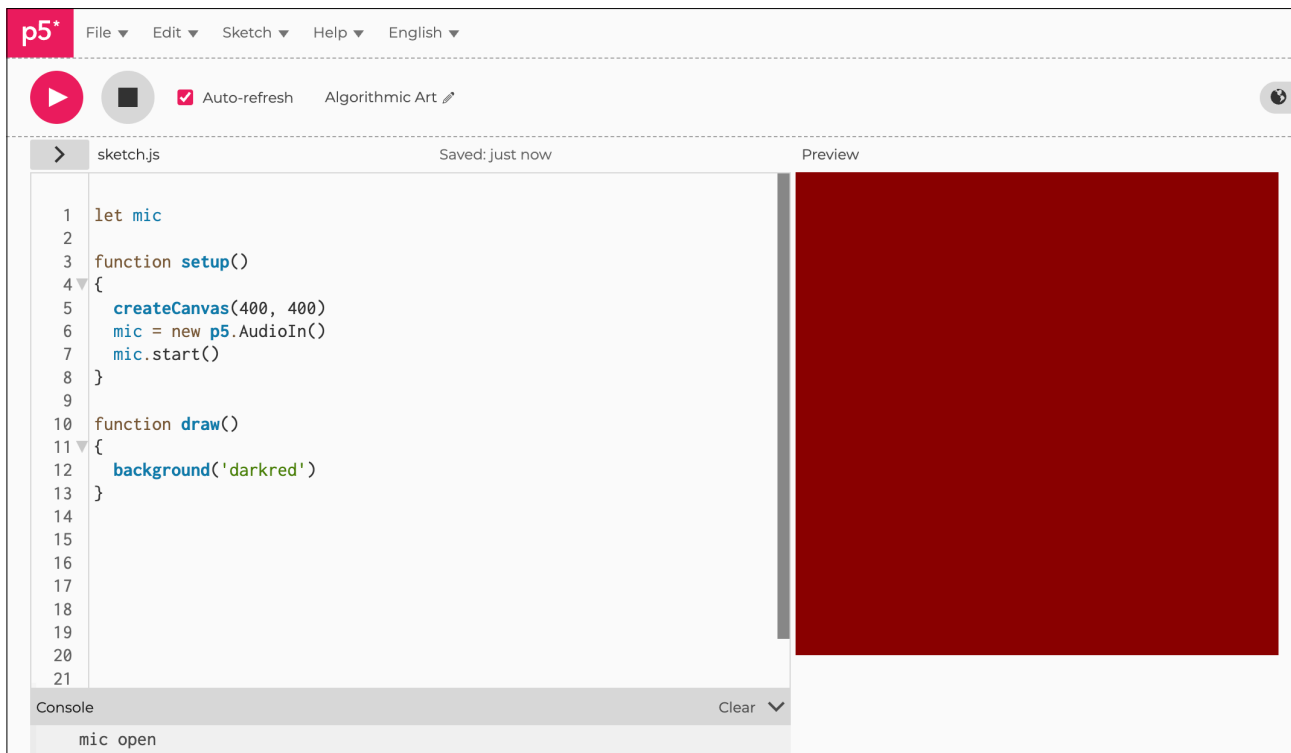
function setup()
{
  createCanvas(400, 400)
  mic = new p5.AudioIn()
  mic.start()
}

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

Notes

We now have the microphone working, hopefully.

Figure F6.1





Sketch F6.2 the amplitude

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

```
let mic
let amp

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

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

Notes

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



Sketch F6.3 the volume

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

```
let mic
let amp
let vol

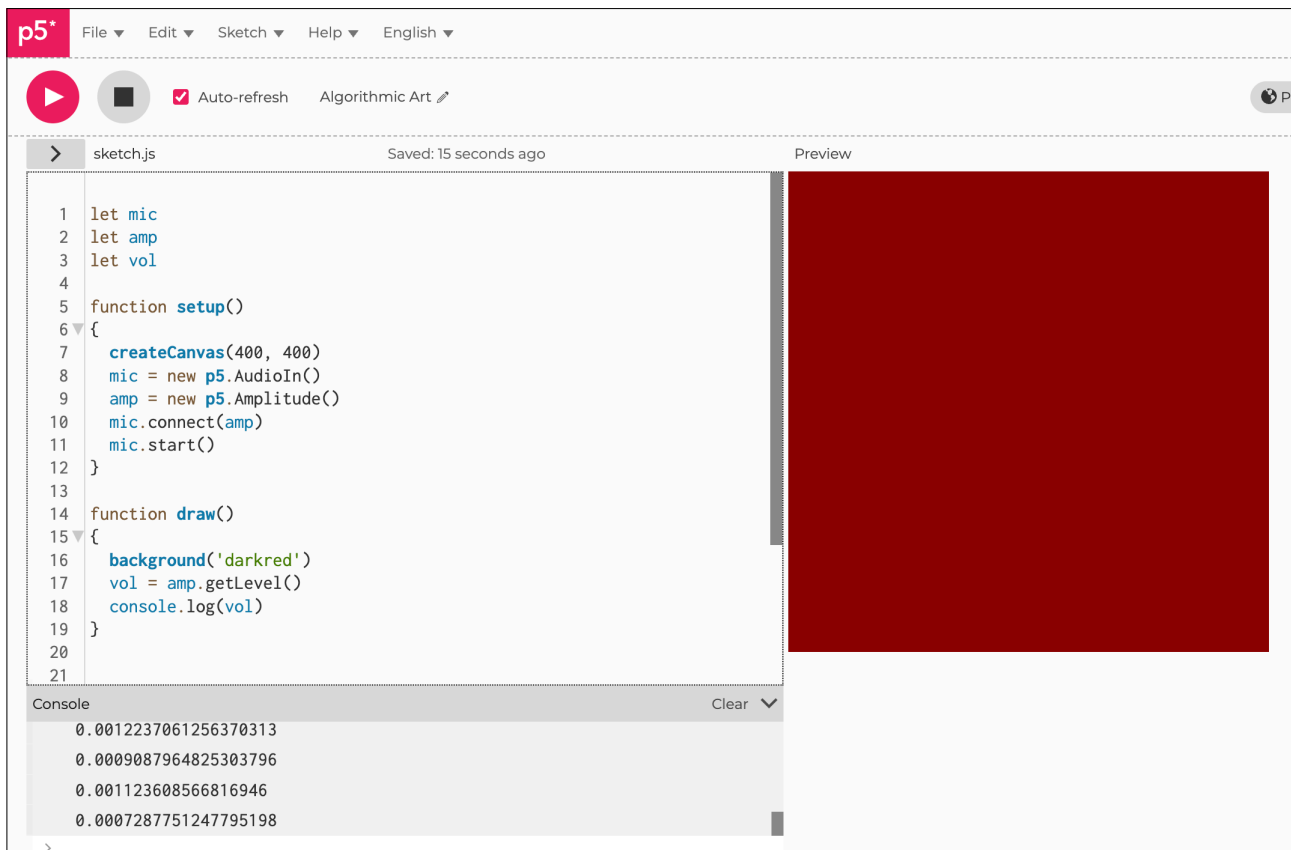
function setup()
{
  createCanvas(400, 400)
  mic = new p5.AudioIn()
  amp = new p5.Amplitude()
  mic.connect(amp)
  mic.start()
}

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

Notes

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

Figure F6.3





Sketch F6.4 a circle

! Remove the `console.log()`.

We can draw a circle that responds to the noise level the microphone picks up.

```
let mic
let amp
let vol

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

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

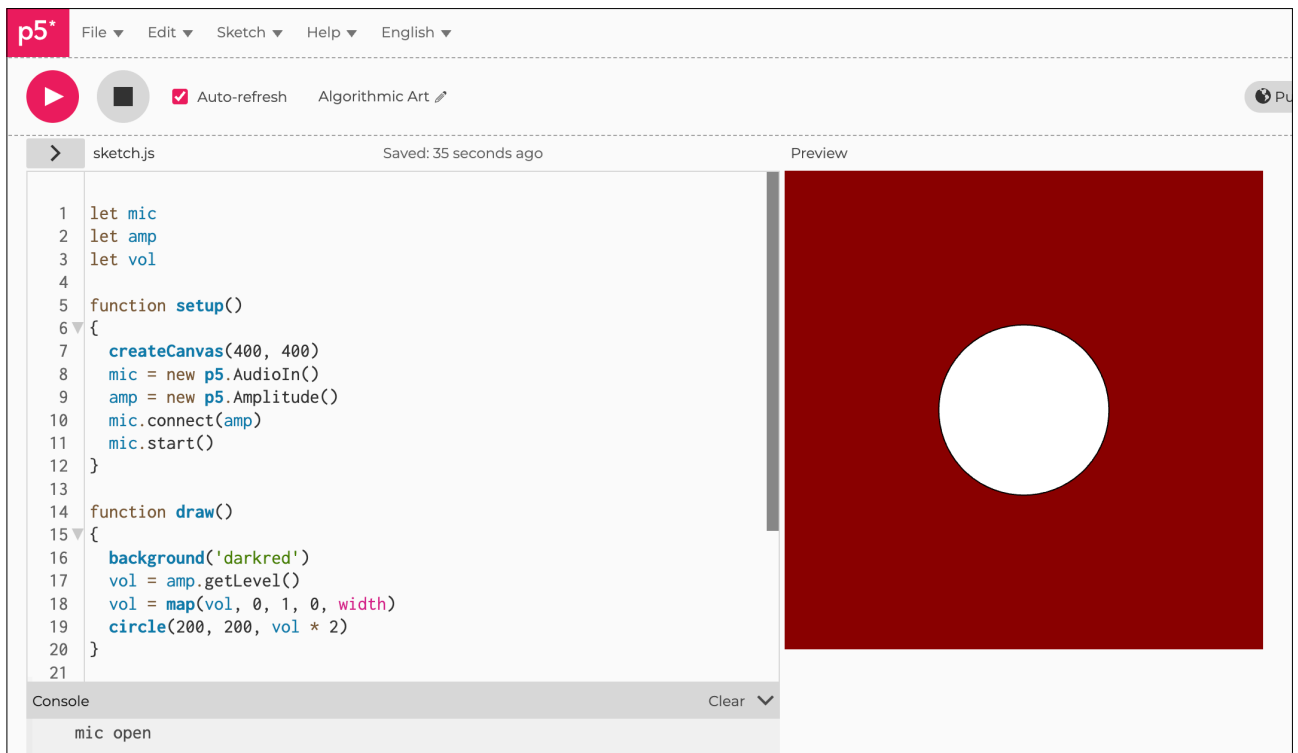
Notes

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

Challenges

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

Figure F6.4





Sketch F6.5 bouncing ball?

! Remove mapping the volume.

We can move the ball with our voice, sort of.

```
let mic
let amp
let vol
let y

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

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

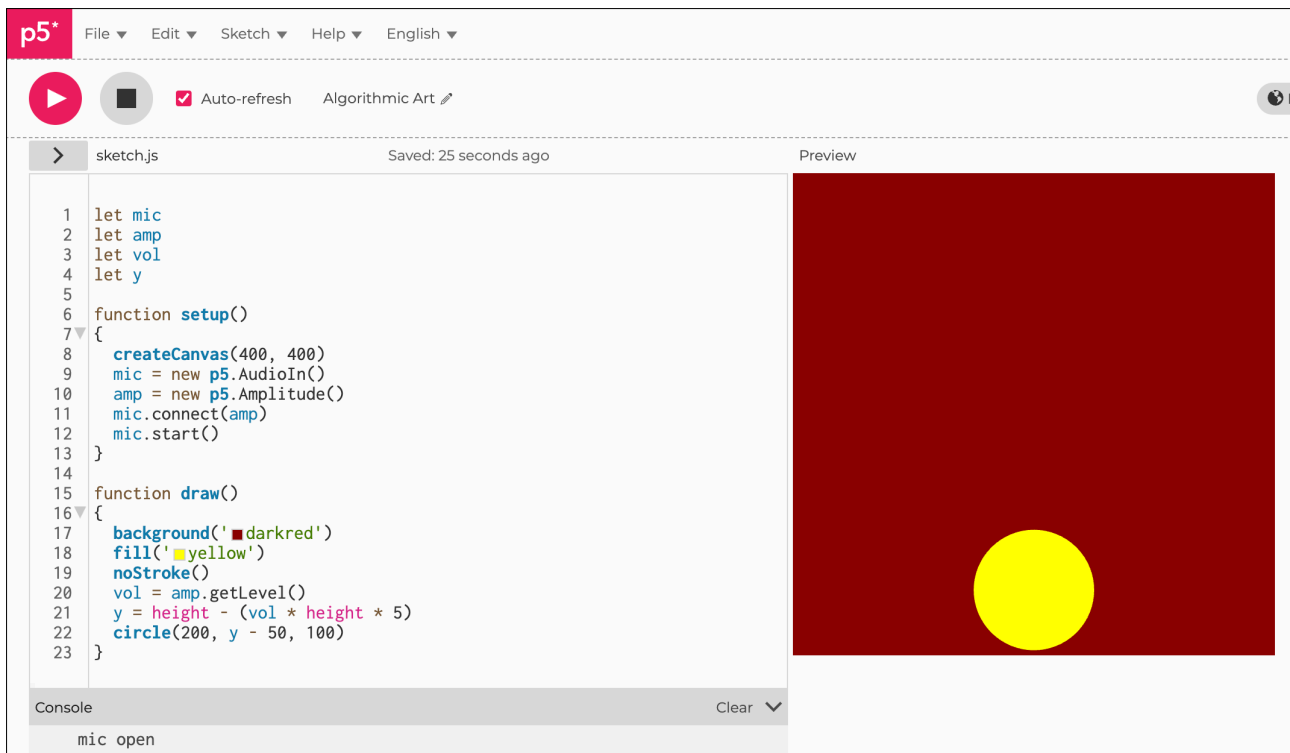
Notes

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

Challenge

Can you create other designs that respond to your voice?

Figure F6.5



Algorithmic
Art
Module F
Unit #7
Telling
the Time



Module F Unit #7: telling the time

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

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



Sketch F7.1 every second counts

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

```
let h
let m
let s

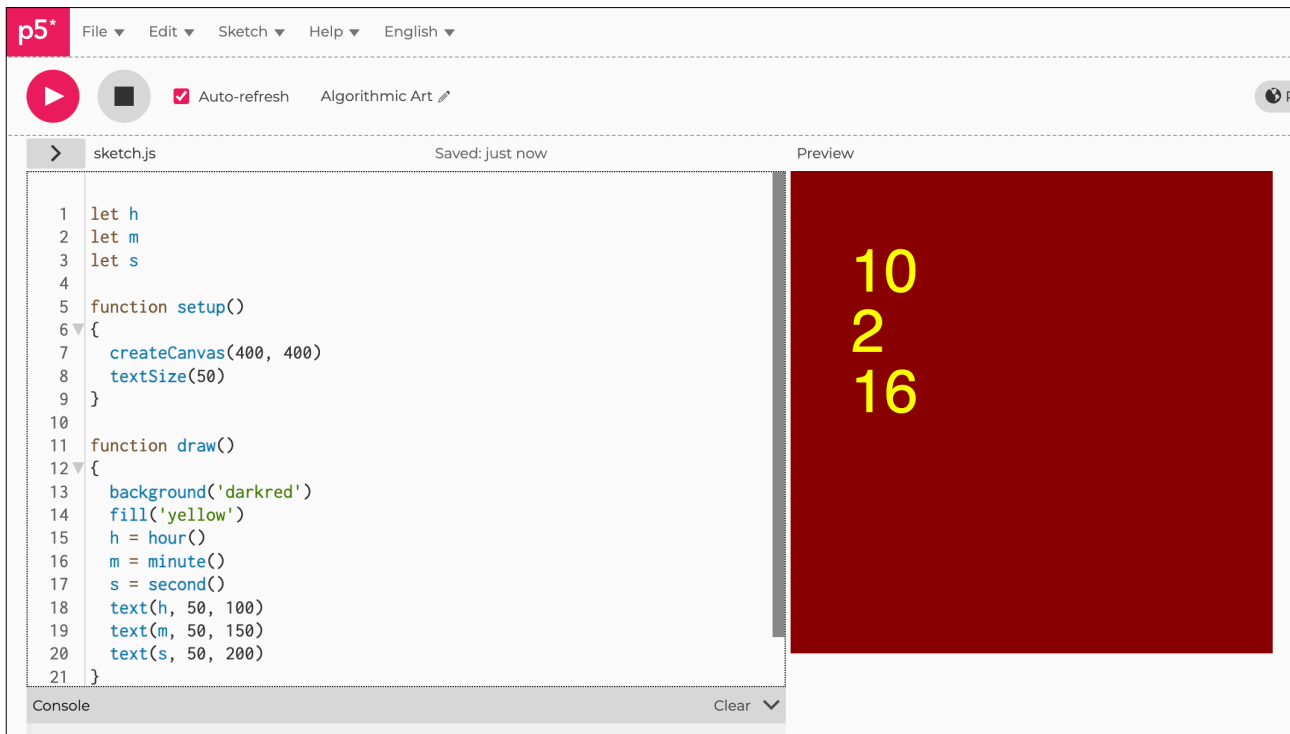
function setup()
{
  createCanvas(400, 400)
  textSize(50)
}

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

Notes

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

Figure F7.1





Time functions

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

Notes

Collects data from the internet.

Challenge

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



Sketch F7.2 a second look

! Replacing `text(m)` and `text(s)`.

A better-looking clock.

```
let h
let m
let s

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

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

```

{
  text(s, 50, 200)
}
}

```

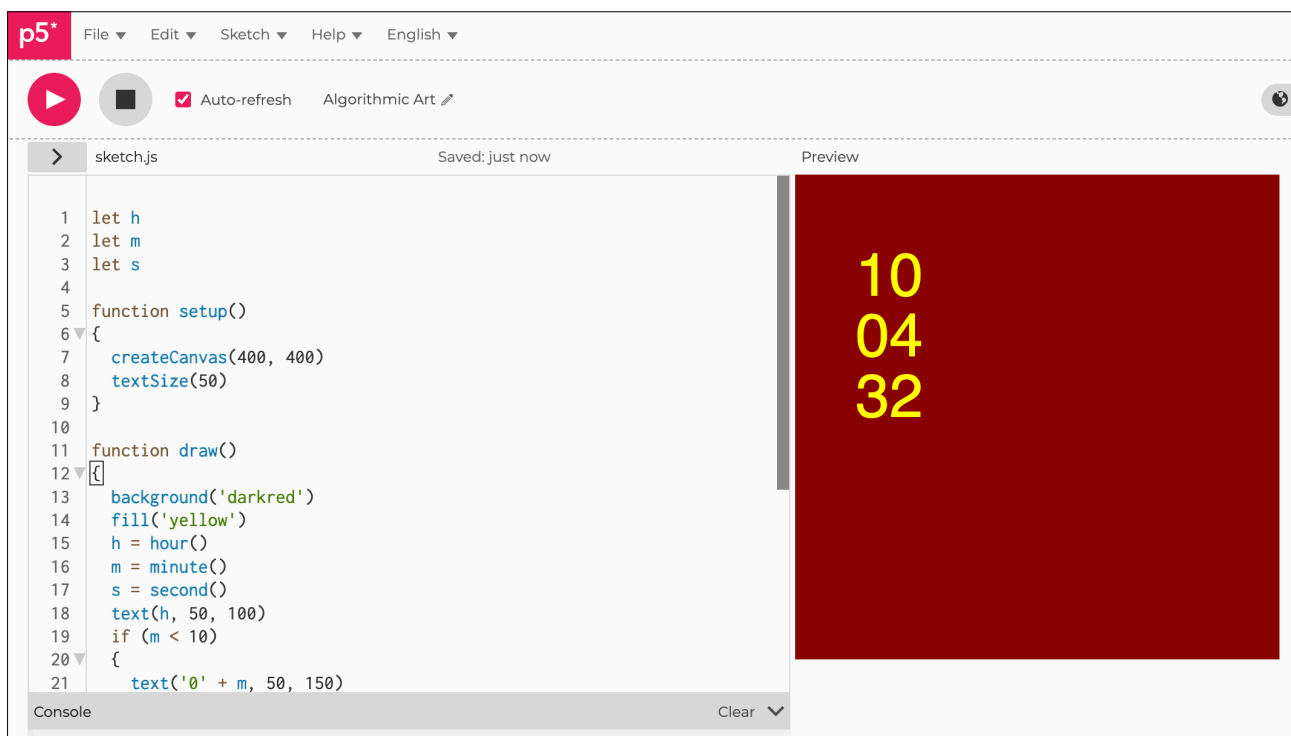
Notes

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

Challenge

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

Figure F7.2





Sketch F7.3 what a year

! Replacing hours, minutes and seconds

Adding the date.

```
let d
let m
let y

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

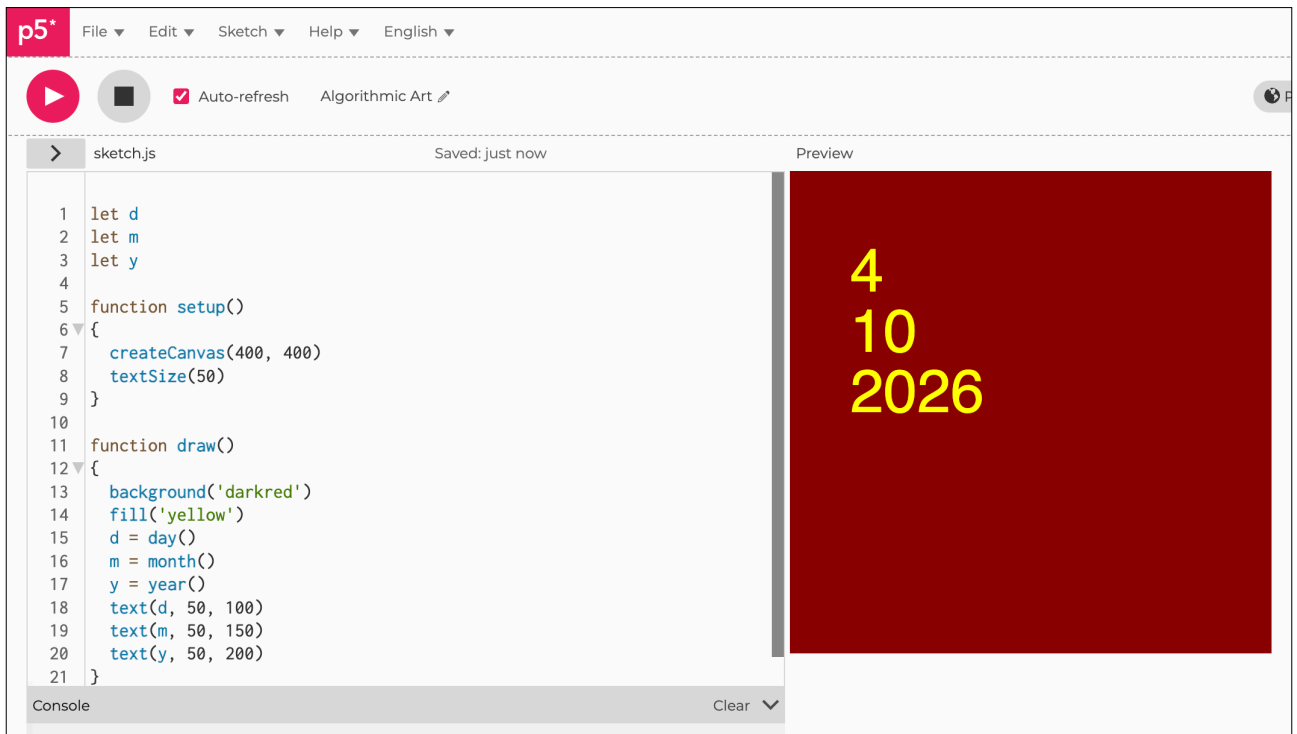
function draw()
{
  background('darkred')
  fill('yellow')

  d = day()
  m = month()
  y = year()
  text(d, 50, 100)
  text(m, 50, 150)
  text(y, 50, 200)
}
```

Notes

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

Figure F7.3





Date functions

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

Notes

These pull the data from the internet.

Challenge

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



Sketch F7.4 milliseconds of time

! Starting a new sketch.

Milliseconds of time represented differently, as a float, an integer, and as seconds.

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

function draw()
{
  background('darkred')
  fill('yellow')
  text(millis(), 50, 50)
  text(int(millis()), 50, 100)
  text(int(millis()/1000), 50, 150)
}
```

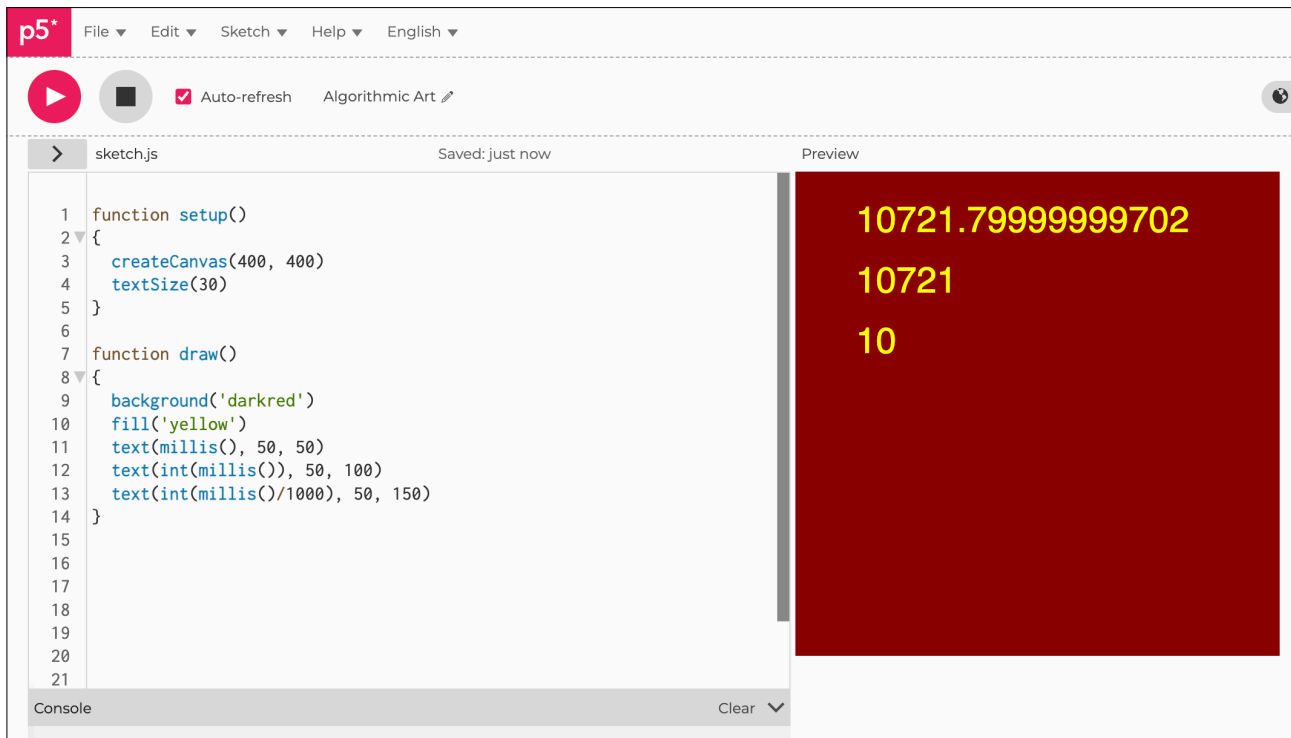
Notes

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

Challenge

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

Figure F7.4





Sketch F7.5 in the blink of an eye

! Start a new sketch.

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

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

function setup()
{
  createCanvas(400, 400)
  background('darkred')
}

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

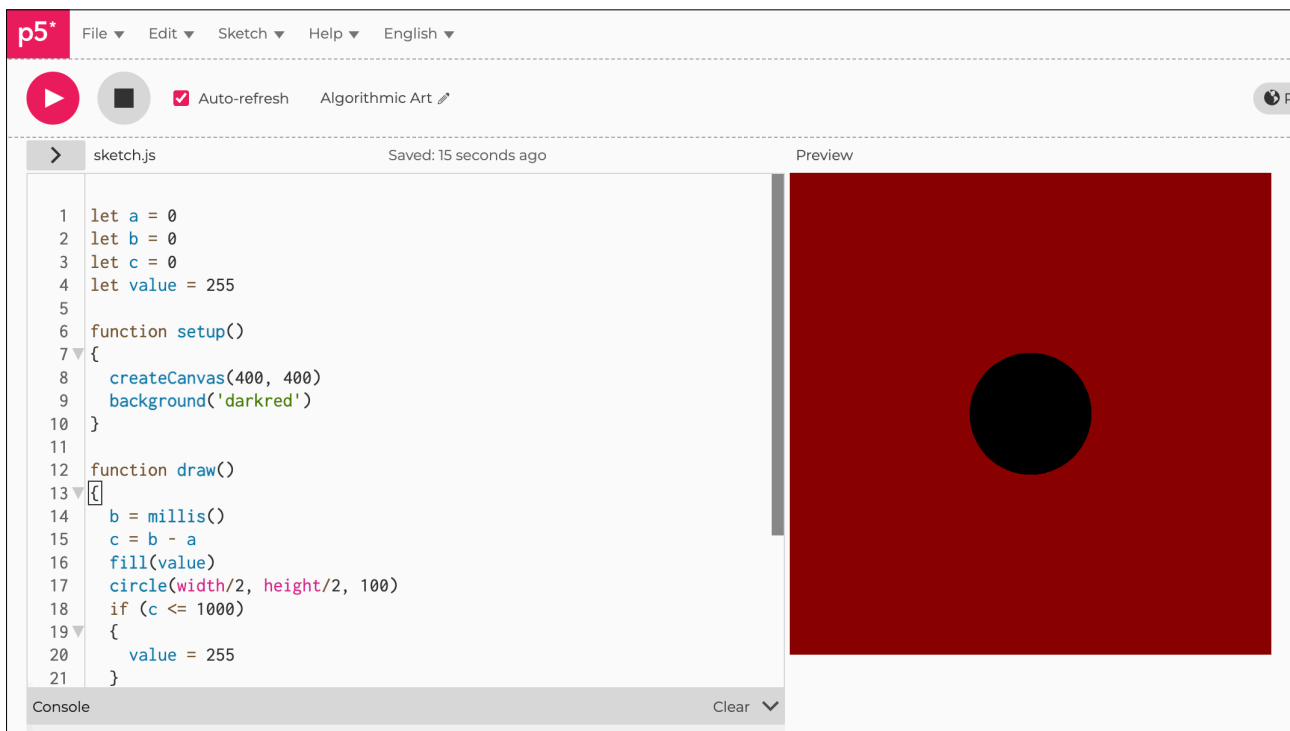
Notes

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

Challenges

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

Figure F7.5





Sketch F7.6 alternative modulo

! Start a new sketch.

This does the same thing where **modulo** returns the remainder.

```
let value = 255

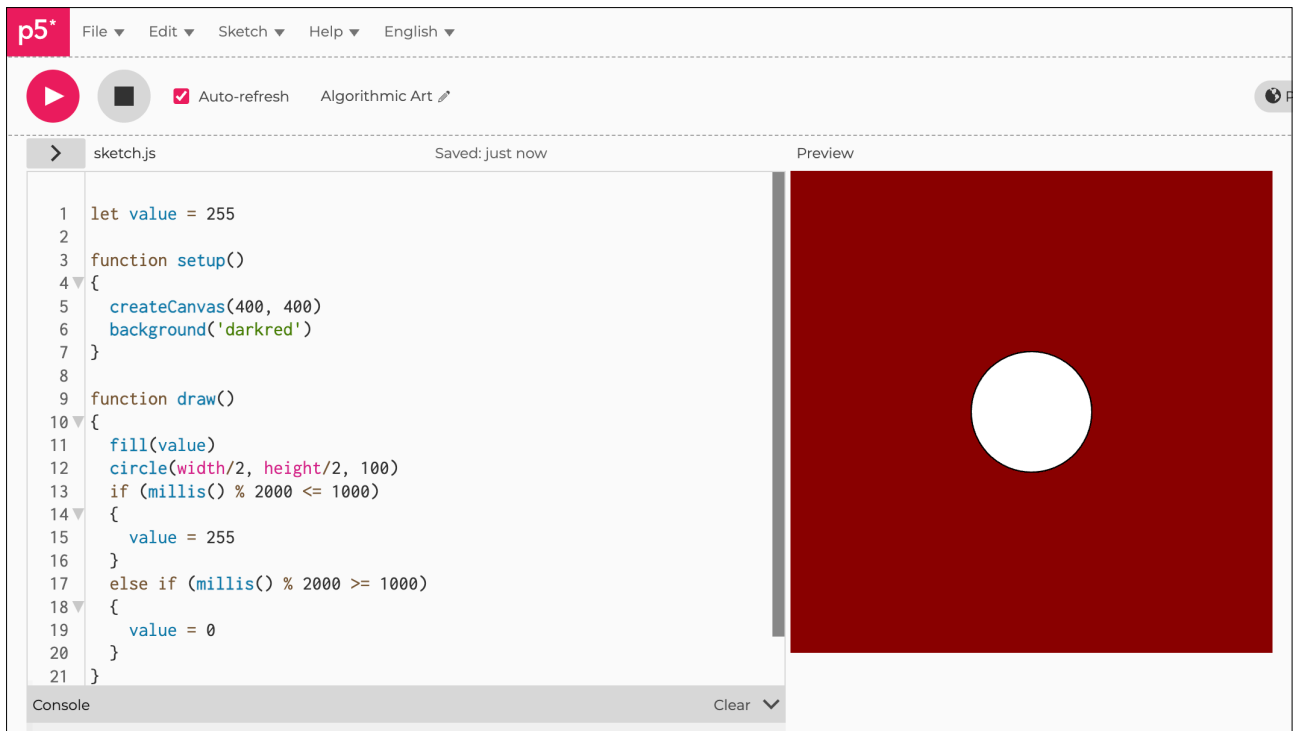
function setup()
{
  createCanvas(400, 400)
  background('darkred')
}

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

Notes

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

Figure F7.6





Sketch F7.7 making a clock

! Start a new sketch.

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

```
let radius = 200
let clockDiameter

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

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

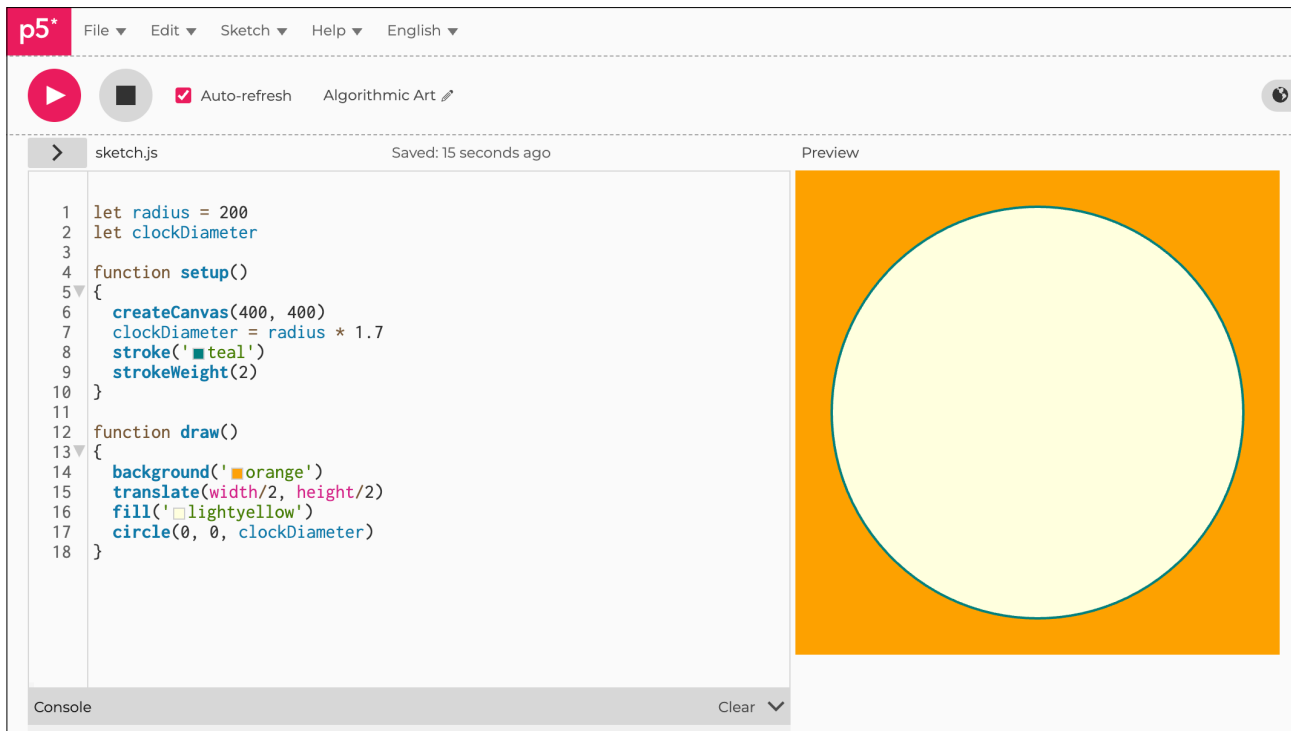
Notes

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

Challenge

Feel free to use other colours.

Figure F7.7





Sketch F7.8 other radii

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

```
let radius = 200
let clockDiameter

let secondsRadius
let minutesRadius
let hoursRadius

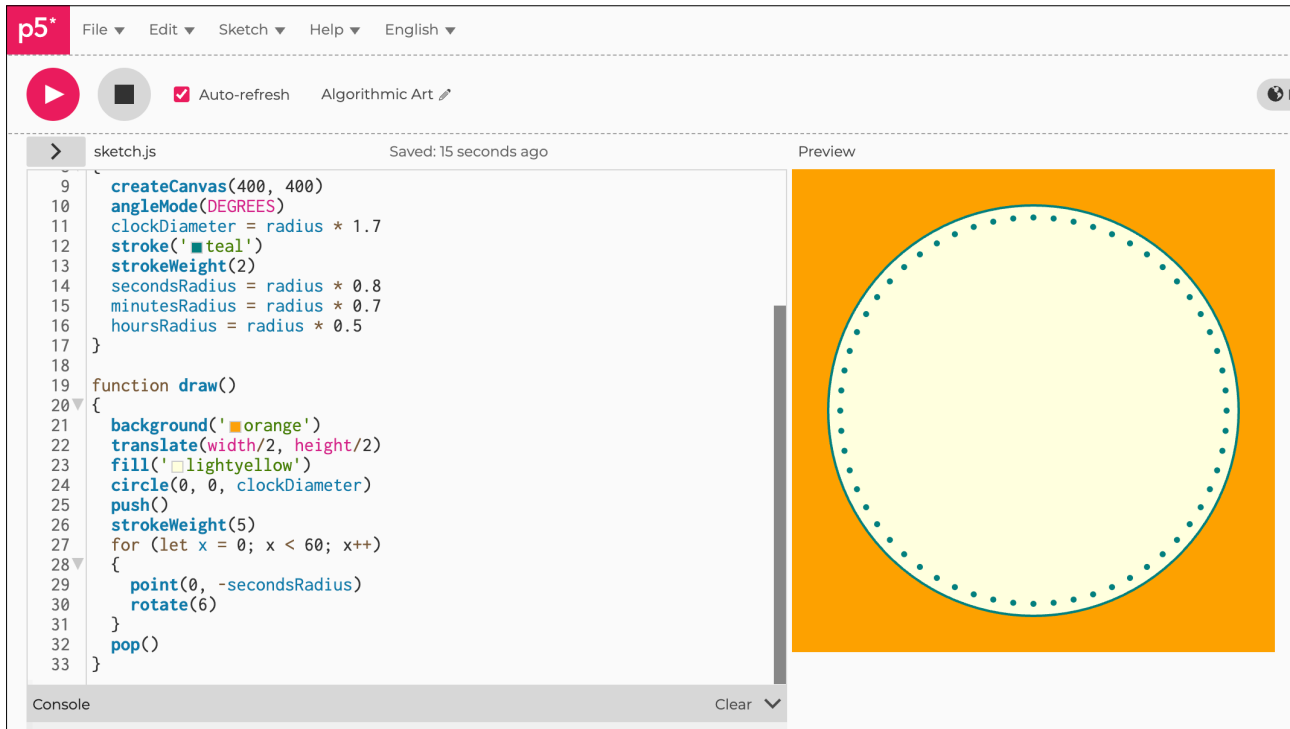
function setup()
{
  createCanvas(400, 400)
  angleMode(DEGREES)
  clockDiameter = radius * 1.7
  stroke('teal')
  strokeWeight(2)
  secondsRadius = radius * 0.8
  minutesRadius = radius * 0.7
  hoursRadius = radius * 0.5
}

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

Notes

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

Figure F7.8





Sketch F7.9 the angles

Calculating the angle from the time.

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

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

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

```
pop()
```

```
secondsAngle = map(second(), 0, 60, 0, 360)
```

```
minutesAngle = map(minute(), 0, 60, 0, 360)
```

```
hoursAngle = map(hour(), 0, 12, 0, 360)
```

```
}
```

Notes

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



Sketch F7.10 drawing the hands

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

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

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

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

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

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

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

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

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

```
}
```

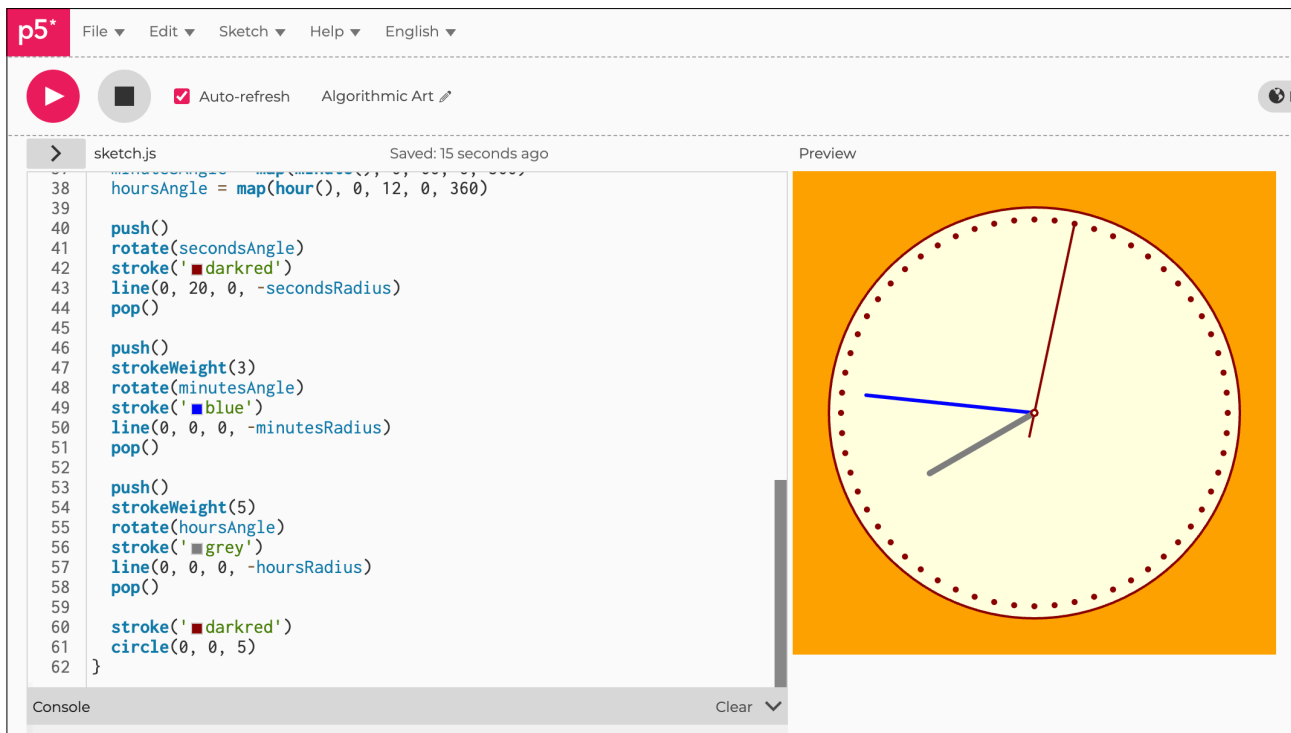
Notes

We have our finished clock.

Challenge

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

Figure F7.10



Algorithmic Art

Module F

Unit #8

Local
Storage



Module F Unit #8: local storage

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



Sketch F8.1 sliders

! Starting a brand new sketch.

By sliding the **sliders**, we can change the background.

```
let rSlider
let gSlider
let bSlider

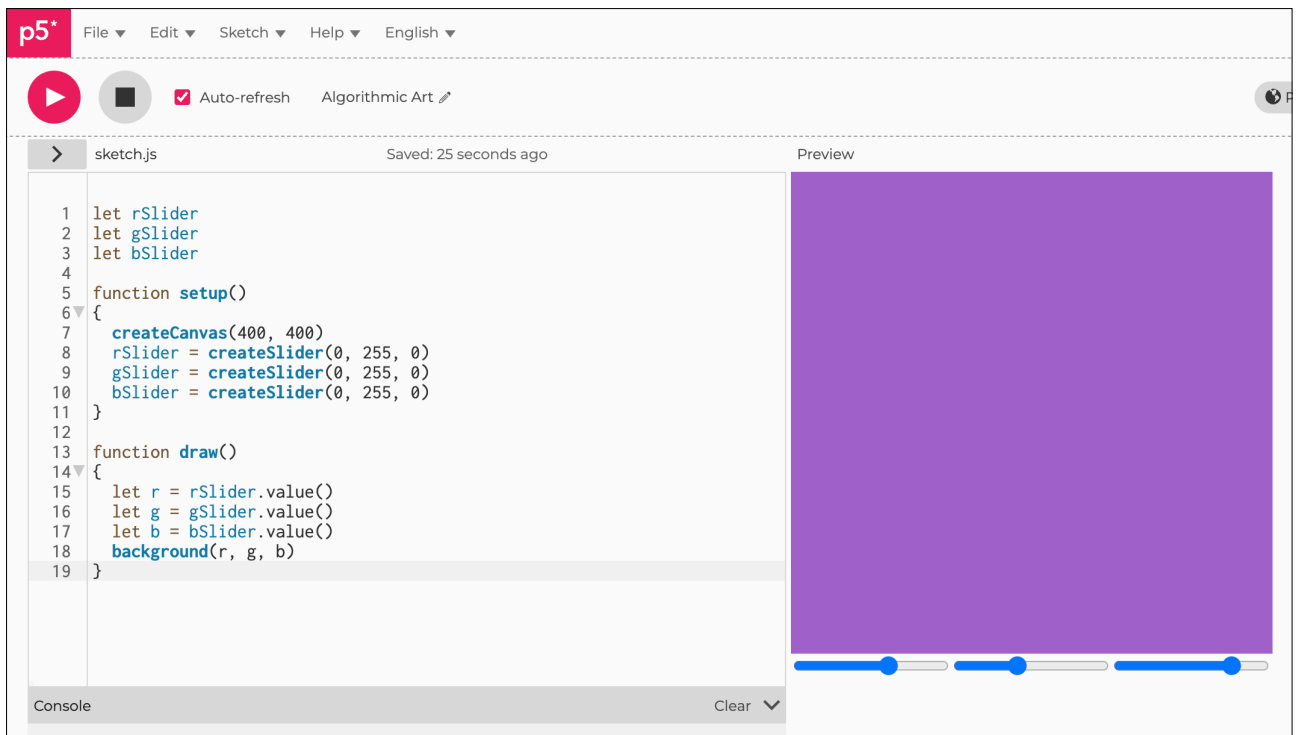
function setup()
{
  createCanvas(400, 400)
  rSlider = createSlider(0, 255, 0)
  gSlider = createSlider(0, 255, 0)
  bSlider = createSlider(0, 255, 0)
}

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

Notes

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

Figure F8.1





Sketch F8.2 storing data

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

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

```
let rSlider
let gSlider
let bSlider

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

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

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

Notes

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

<code>storeItem()</code>	which, as it suggests, stores the data we want stored
<code>getItem()</code>	this retrieves the said data



Sketch F8.3 getting the data

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

```
let rSlider
let gSlider
let bSlider

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

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

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

Notes

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

Challenge

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



Sketch F8.4 replacing all the values

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

```
let rSlider
let gSlider
let bSlider

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

```

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

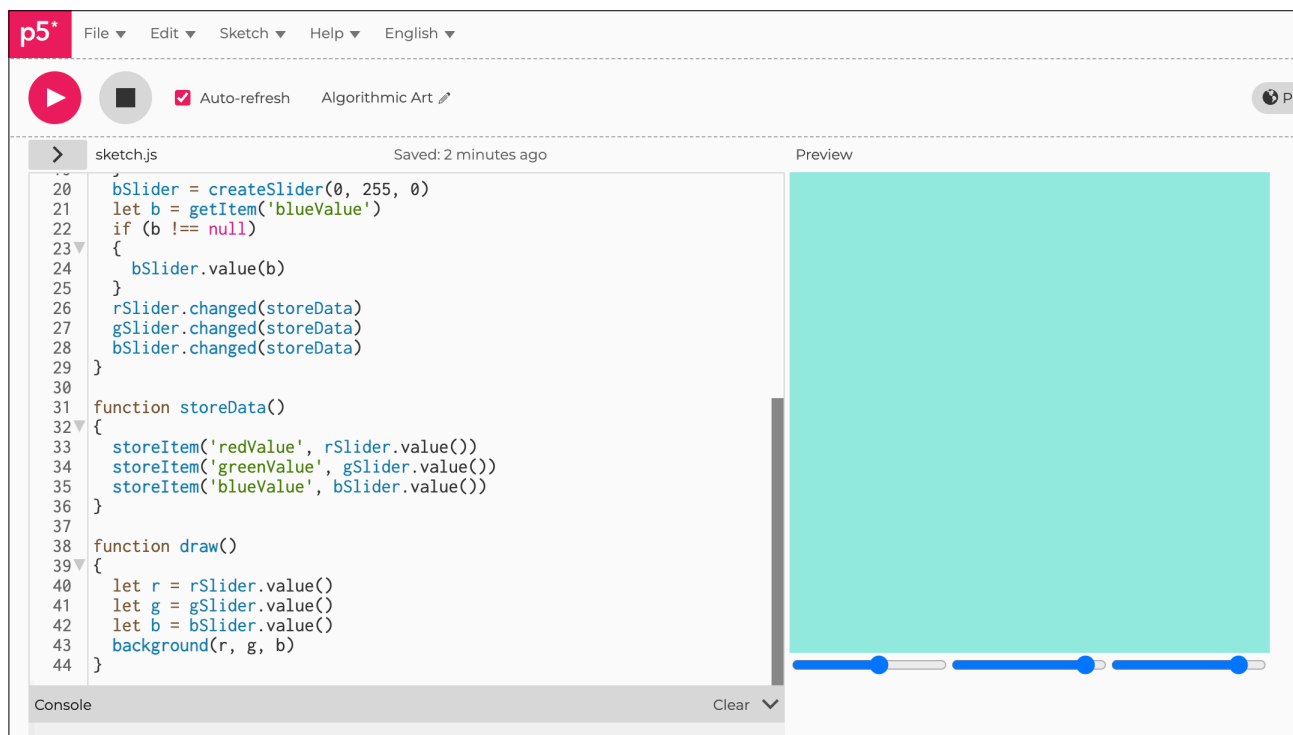
function draw()
{
  let r = rSlider.value()
  let g = gSlider.value()
  let b = bSlider.value()
  background(r, g, b)
}

```

Notes

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

Figure F8.4



Algorithmic Art

Module F Unit #9

Using json Files



Module F Unit #9: using json files

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

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



Sketch F9.1 a bubble object

! Starting a new sketch

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

```
let bubble

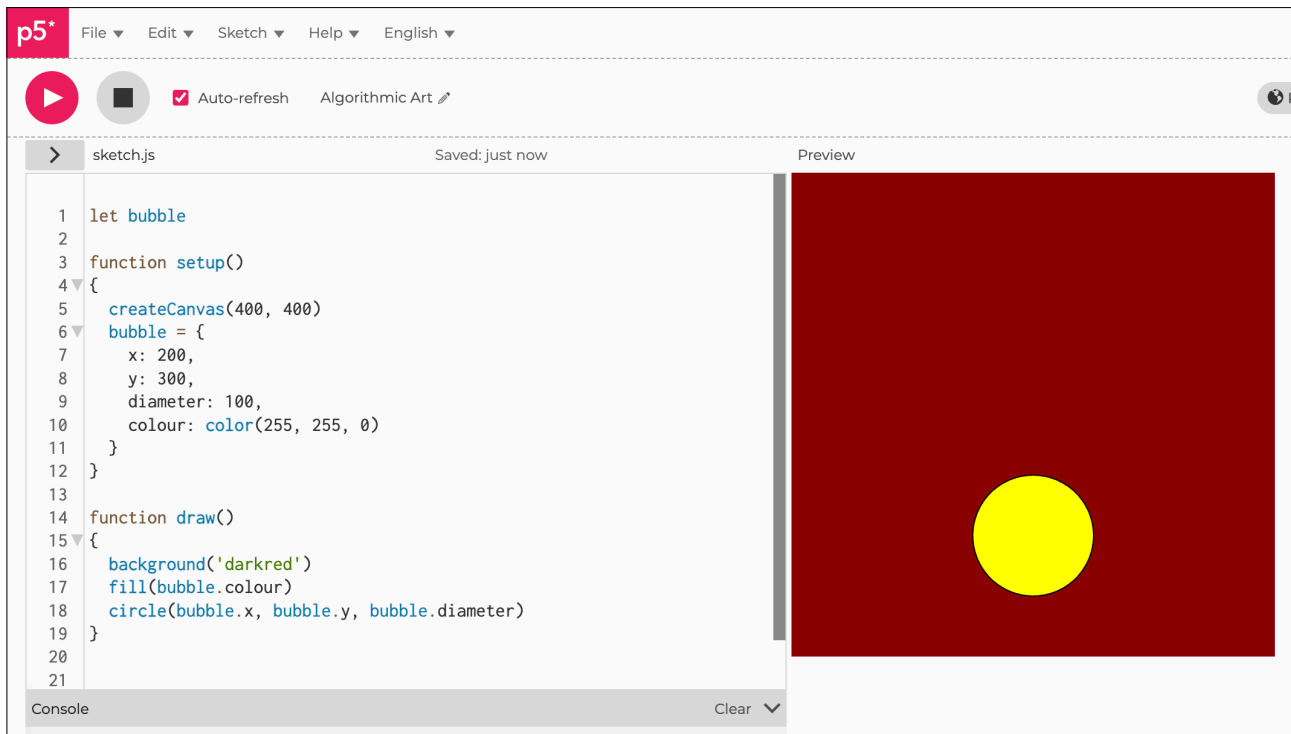
function setup()
{
  createCanvas(400, 400)
  bubble = {
    x: 200,
    y: 300,
    diameter: 100,
    colour: color(255, 255, 0)
  }
}

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

Notes

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

Figure F9.1

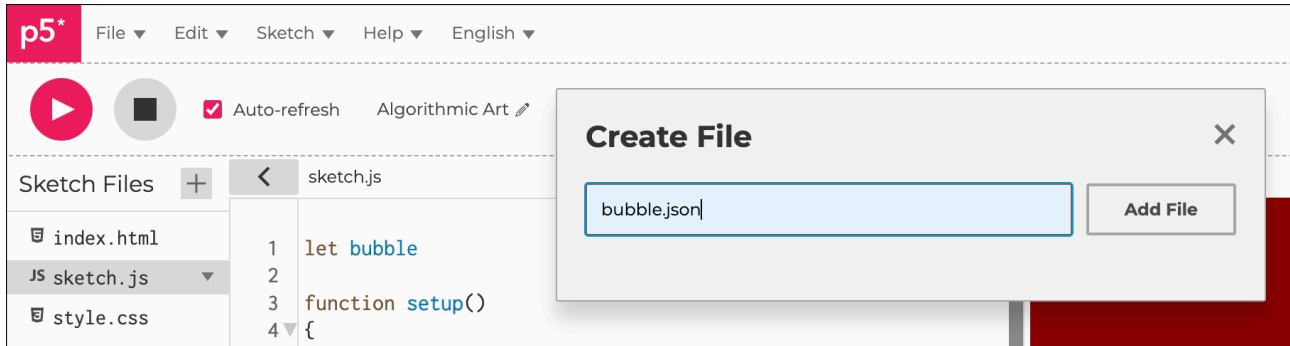




Creating a json file

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

Figure 1: creating the bubble.json file





Sketch F9.2 the json data file

! The **.json** file. Type this into our new file (that should be empty)

bubble.json

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

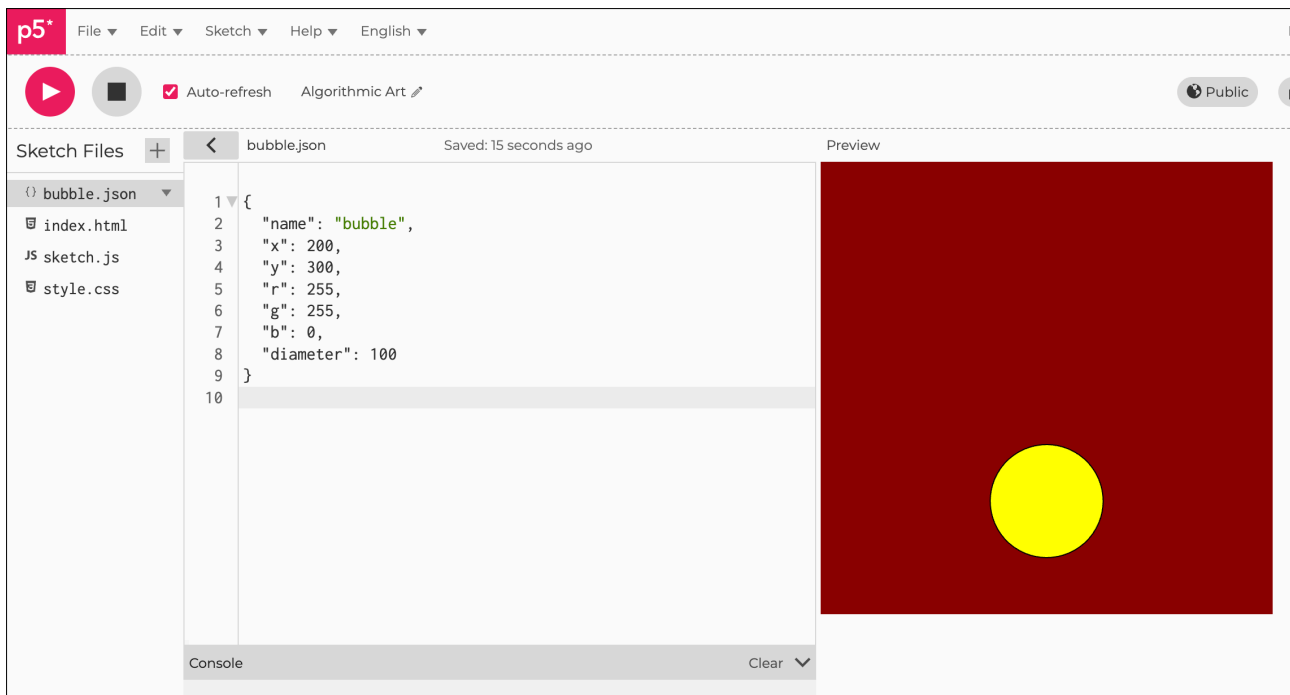
Notes

You can check if the format is correct because it does need to be. The following website will do that by you copying and pasting it into the website, and it will return the result:

<https://jsonformatter.curiousconcept.com/#>

is a way of checking if the JSON is OK.

Figure F9.2





Sketch F9.3 adapting the sketch

! Back to sketch.js. Remove the bubble info from the sketch.

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

```
let bubble

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

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

Notes

We load the data and extract it accordingly.

Figure F9.3





Sketch F9.4 more than one bubble

! Revisiting the bubble.json file.

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

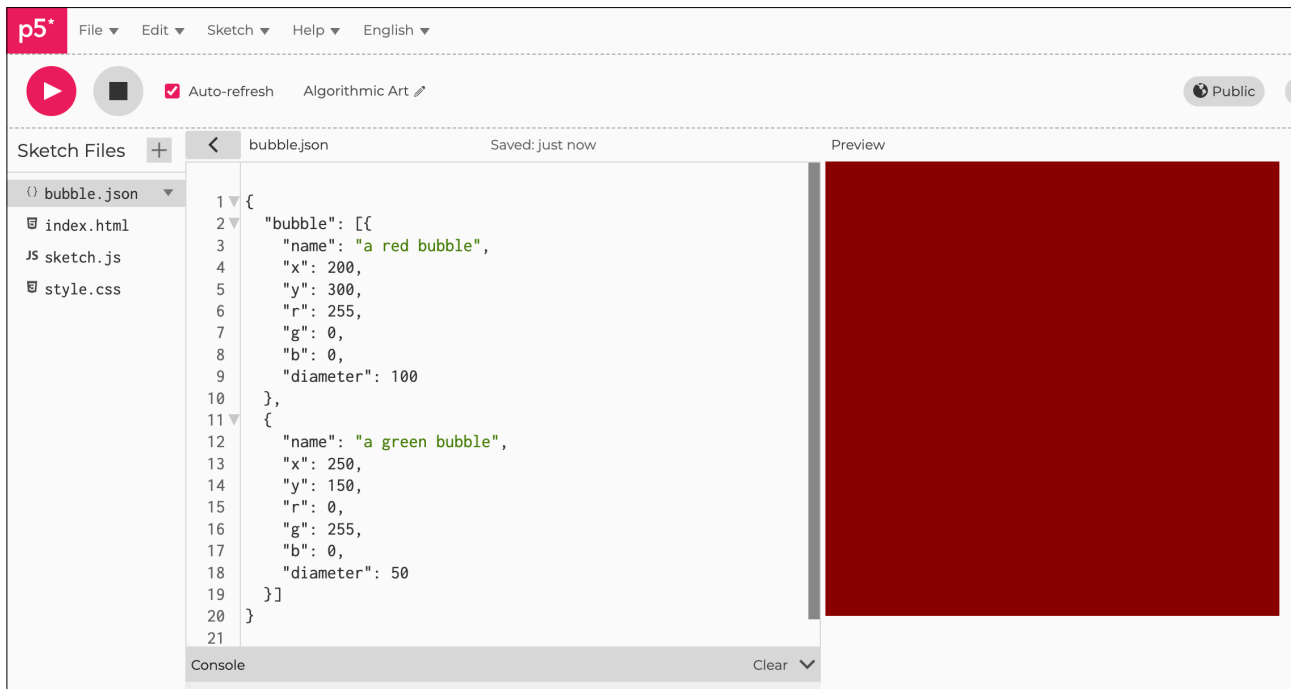
bubble.json

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

Notes

This array has two bubble elements.

Figure F9.4





Sketch F9.5 a long winded way

! In the sketch.js.

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

```
let data

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

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

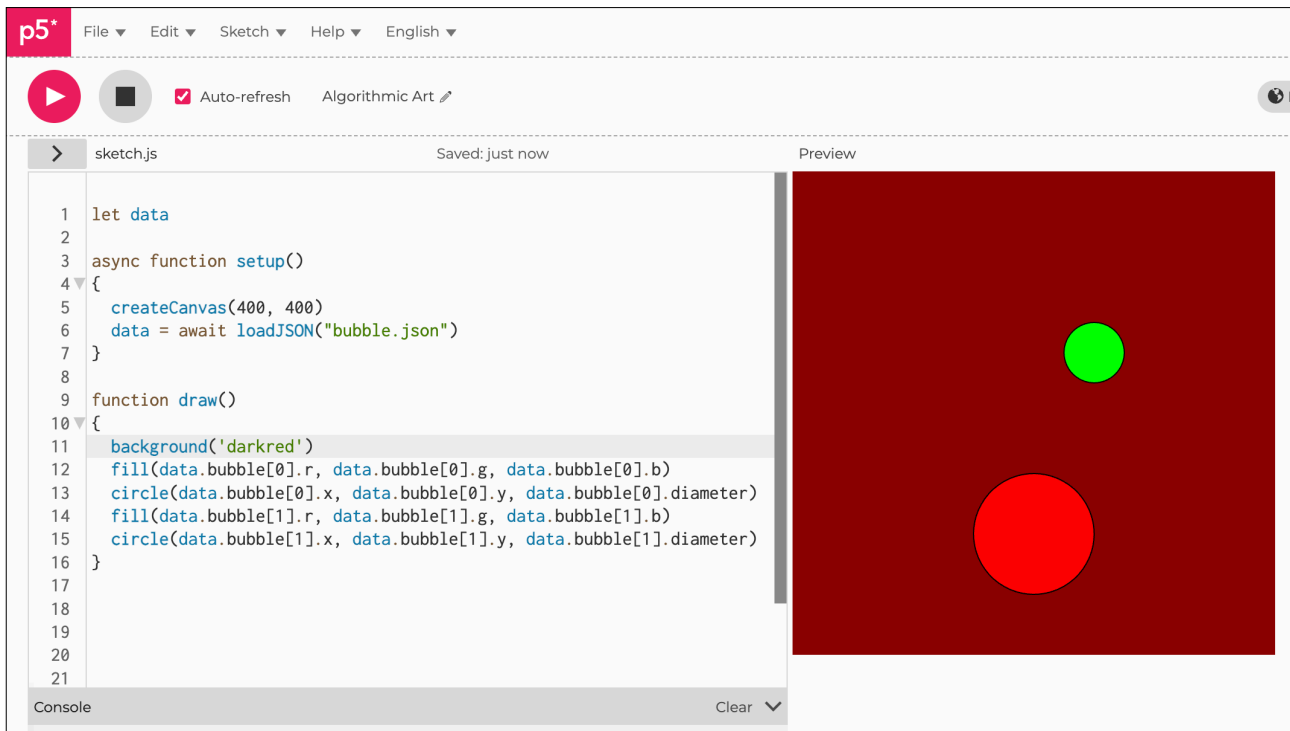
Notes

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

Challenges

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

Figure F9.5





Sketch F9.6 a more concise way

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

```
let data

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

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

Notes

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



Using a pre-made .json file

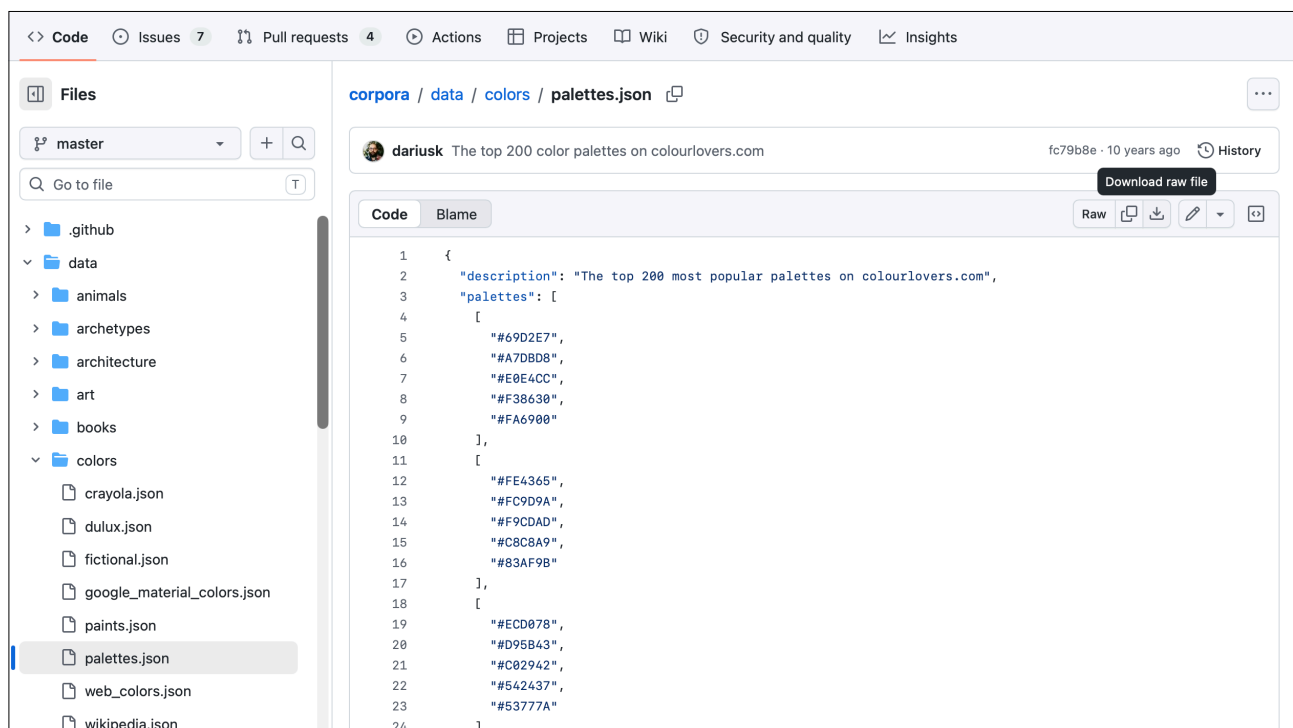
We are going to use a pre-made **JSON** file that has a collection of some of the favourite colour palettes in hex values. We can access this file and use the five colours in each set of palettes; there are **200** palettes altogether.

To get the **JSON** file, follow the link here or use the link on the website to download the file:

<https://github.com/dariusk/corpora/blob/master/data/colors/palettes.json>

You may notice there are other collections that might be of interest.

Figure 2: download the file



```
1 {
2   "description": "The top 200 most popular palettes on colourlovers.com",
3   "palettes": [
4     [
5       "#69D2E7",
6       "#A7DBD8",
7       "#E0E4CC",
8       "#F3B630",
9       "#FA6900"
10    ],
11    [
12      "#FE4365",
13      "#FC9D9A",
14      "#F9CDAD",
15      "#C8C8A9",
16      "#83AF9B"
17    ],
18    [
19      "#ECD078",
20      "#D95B43",
21      "#C02942",
22      "#542437",
23      "#53777A"
24    ],
25    ...
26  ]
27 }
```



Download and upload palette.json file

Click on the uploaded file to see it.

Figure 3: the download tab

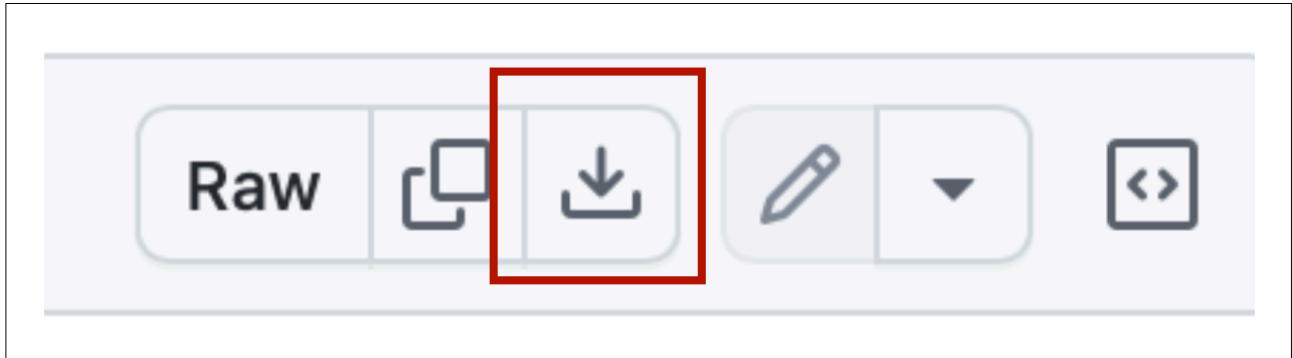
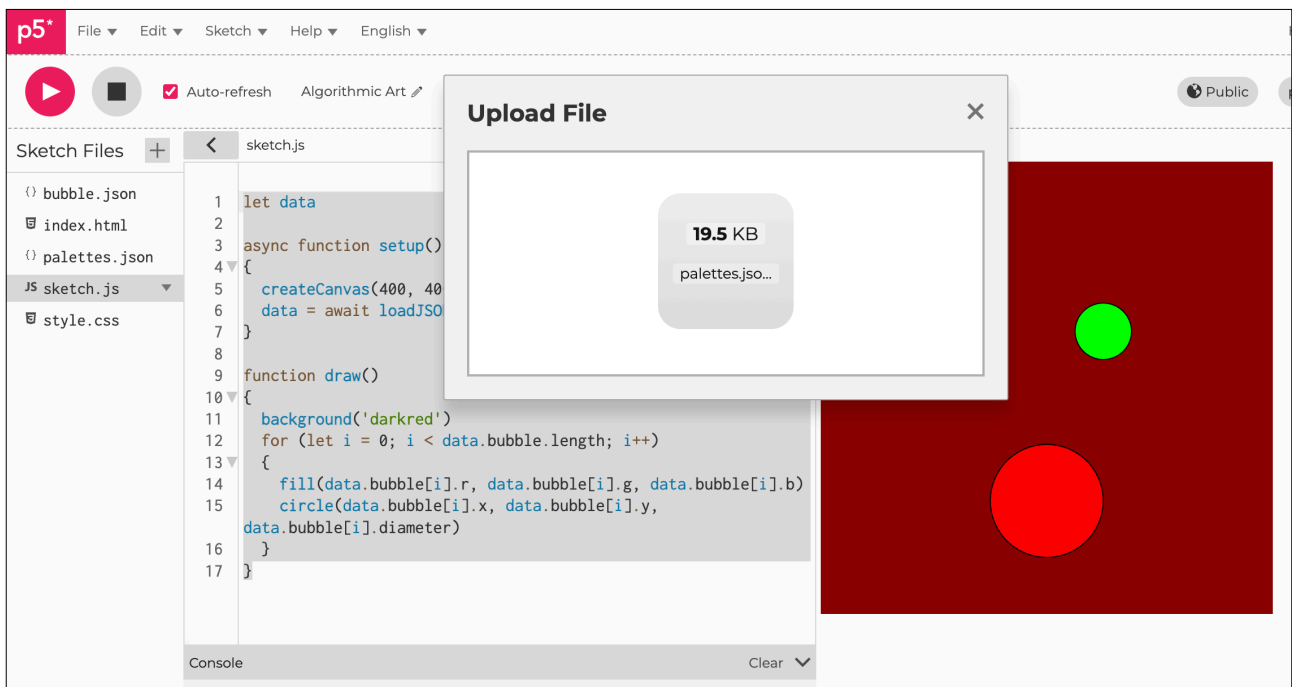


Figure 4: uploading the file





Sketch F9.7 console log

! Start a new sketch.

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

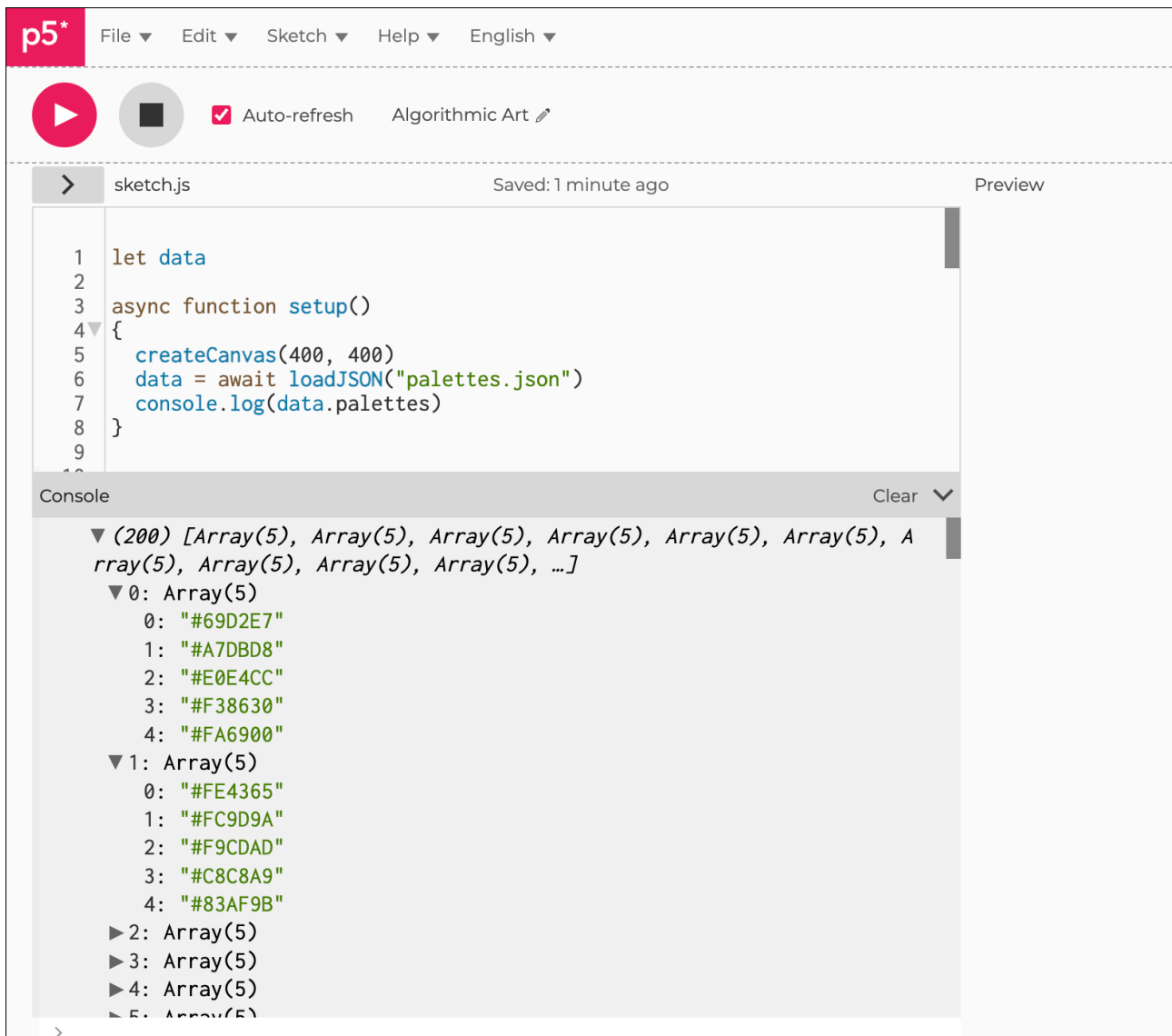
```
let data

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

Notes

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

Figure F9.7





Sketch F9.8 palette number 10

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

```
let data

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

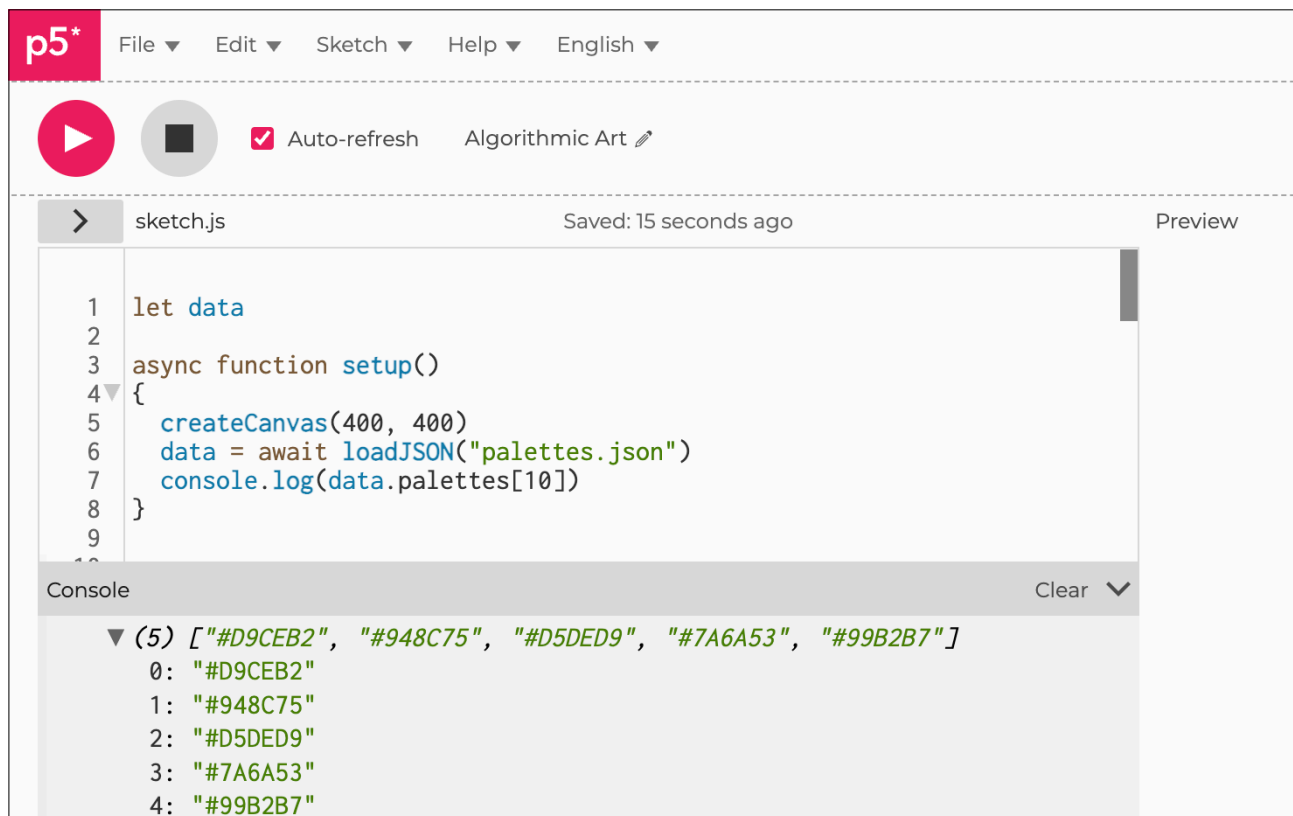
Notes

Randomly picked a palette.

Challenge

Select your own palette.

Figure F9.8





Sketch F9.9 specific colour

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

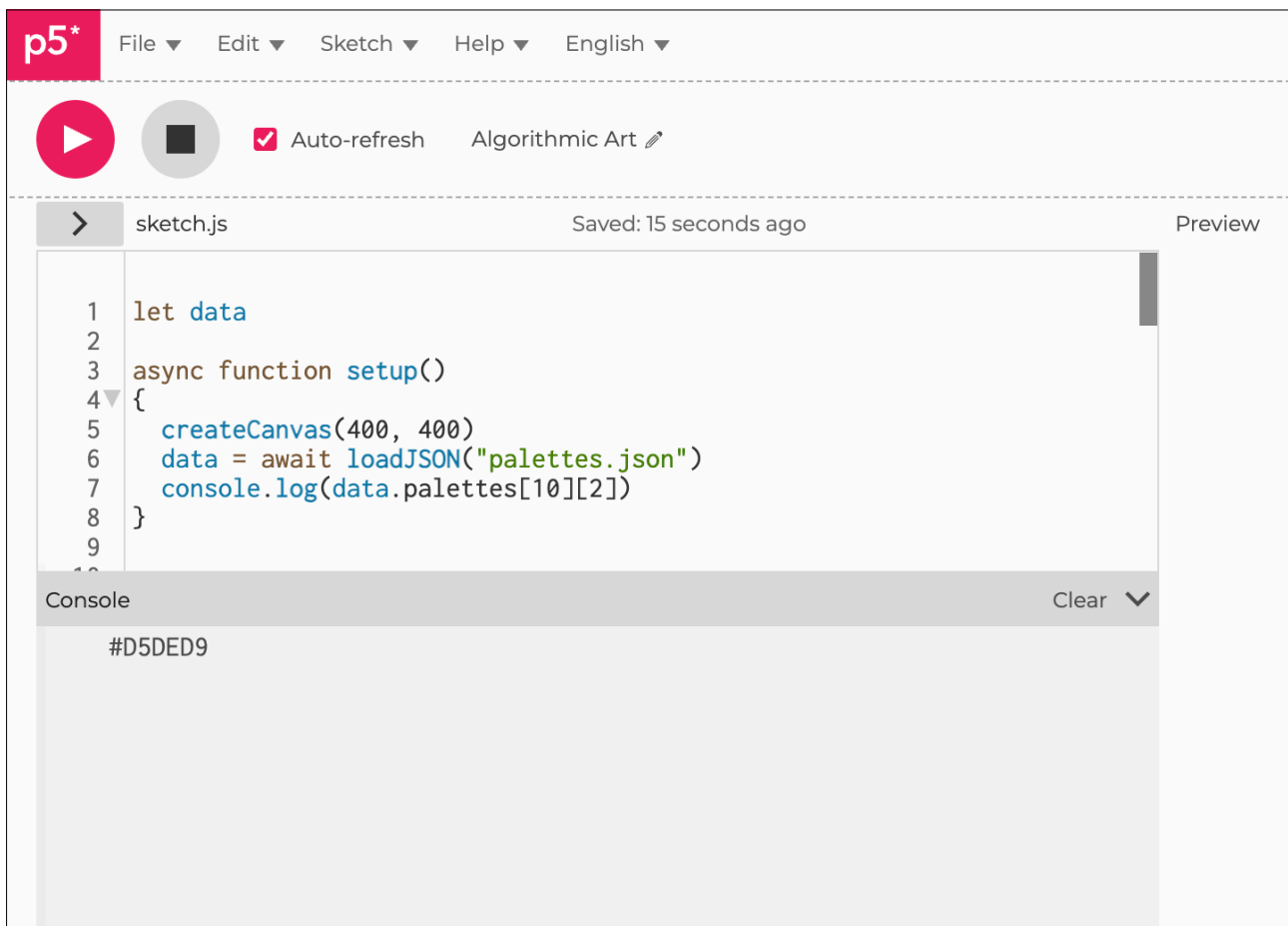
```
let data

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

Notes

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

Figure F9.9





Sketch F9.10 fill the circle

Let's have a look at it.

! Remove console log.

```
let data

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

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

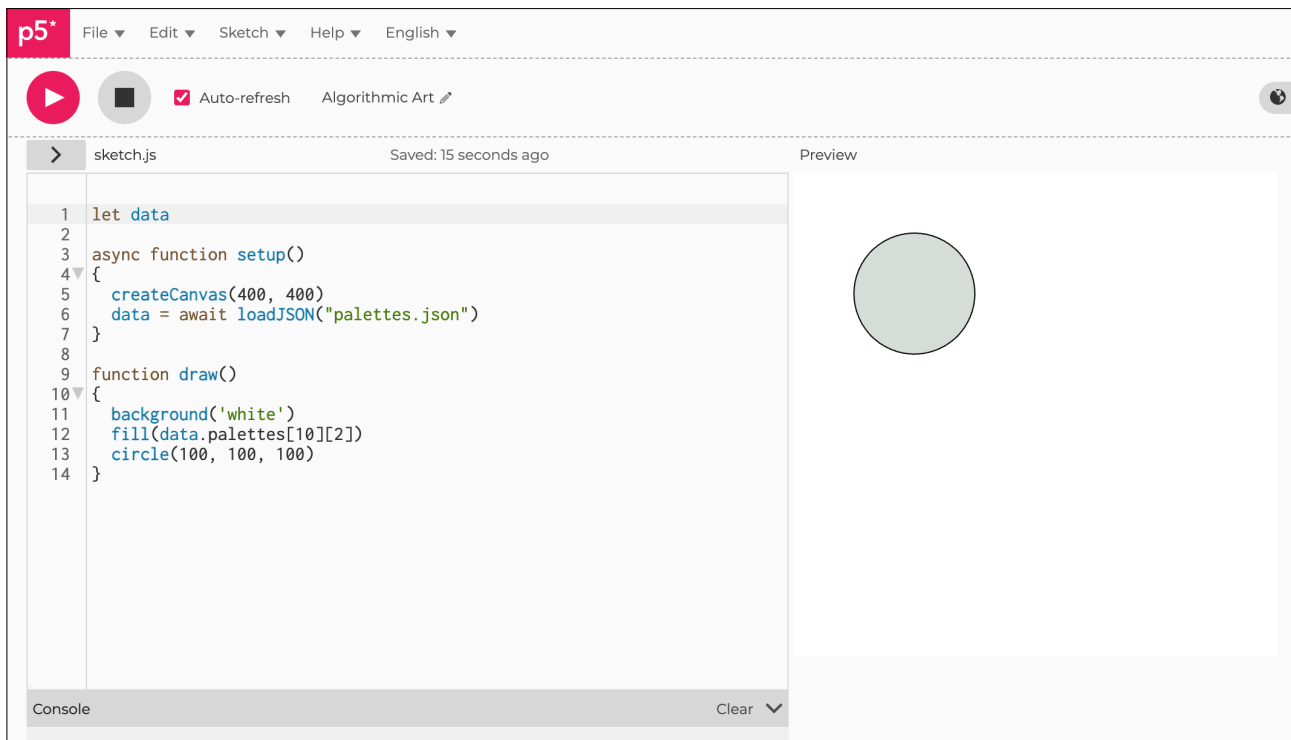
Notes

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

Challenges

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

Figure F9.10





Sketch F9.11 cycle through the palette

The main function is to iterate through each palette one at a time as you click the mouse.

```
let data

let i = 0

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

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

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

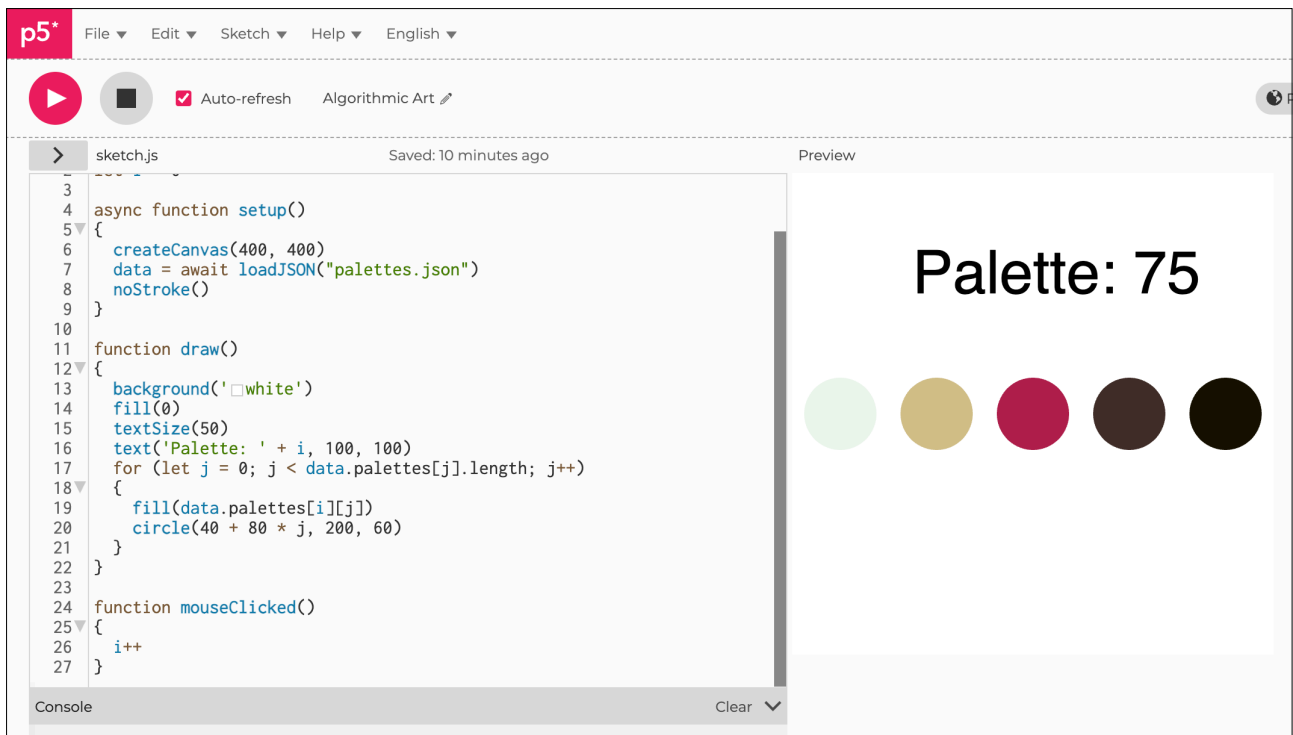
Notes

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

Challenge

How would you reset the counter when it reaches the end of the array?

Figure F9.11





Sketch F9.12 saveJSON()

! Starting a new sketch.

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

```
let data

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

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

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

Notes

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

Challenge

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

Figure F9.12

