Algorithmic Art Workbook #1 Your First Circle

Table of Contents

MIT Licence 4
Introduction to the Curriculum/Tutorial 5
The Joy of Coding 6
! Version Update 7
Screen Arrangement 8
Videos 9
Resources 10
Workbook #1 Quick Content Summary 15
Module A Unit #1: p5.js web editor 17
Question 1 – What machine do I need? 18
Question 2 – What is the editor? 19
Question 3 – What does the update mean? 21
Question 4 – How do you sign in? 24
Question 5 – How do you log in? 25
Question 6 – What is the default sketch? 26
Question 7 – Why does your code look different? 28
Question 8 – What do all the buttons do? 30
Question 9 – How do I run my sketch? 31
Question 10 – Can I change the name of my sketch? 32
Question 11 – Is the auto refresh useful? 33
Question 12 – What do the other settings do? 34
Question 13 – What does the accessibility tab mean? 36
Question 14 – Is the main tab menu important? 38
Question 15 – How do I save my work? 39
Question 16 – How useful is the edit function? 41
Question 17 – How do I add files? 42
Question 18 – Is the Help menu helpful? 43
Question 19 – What is the console for? 44
Final words for the first unit 45
Module A Unit #2: your first circle 47
Sketch A2.1 your first sketch 48
Sketch A2.2 a circle 50
Sketch A2.3 adding another circle 52
Sketch A2.4 making a simple pattern 53
Sketch A2.5 adding some colour 55
Sketch A2.6 different colours 57
Sketch A2.7 new sketch 59
Introduction to variables 61
Sketch A2.8 adding some variables 62

Introduction to Random 64
Symbols we use – Operators 65
Arithmetic Operators 65
Comparison Operators 65
Logical Operators 66
Assignment Operators 66
Maths Functions 67
Sketch A2.9 drawing random circles 68
Sketch A2.10 while() loop circles 70
Sketch A2.11 a few more adjustments 72
Introducing the for() loop 74
Sketch A2.12 using for() loops 75
Module A Unit #3: adding RGB 79
RGB colour 80
RGB transparency 81
Sketch A3.1 RGB 82
Sketch A3.2 alpha 84
Sketch A3.3 random colour alpha value 86
Sketch A3.4 no stroke 88
Sketch A3.5 the weight of the stroke 90
Sketch A3.6 colouring the lines 92
Module A Unit #4: lots of lines 95
Sketch A4.1 lines 96
Sketch A4.2 a row of lines 98
Sketch A4.3 limiting lines 100
Sketch A4.4 colour and thickness 102
Sketch A4.5 incrementing the colour 104
Sketch A4.6 more blue 107
Sketch A4.7 random colour and other stuff 109
Sketch A4.8 newish sketch 111
Sketch A4.9 more random lines 113
Sketch A4.10 we need to count the lines 115
Sketch A4.11 if() statement 116



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



Introduction to the Curriculum/Tutorial

The Algorithmic Art Workbook series is designed to get you coding from the very start. It will take you from zero to competent coder over the coming pages. It will expect no previous experience. It will expect no high levels of maths. It will, however, test your problem-solving skills. It may well frustrate you at times, but you don't learn anything unless you try. This is for all ages, not just for the young.

Above all, you will learn how to code, one small step at a time. My recommendation is to do a bit each day. Enjoy the process. It may feel like a bit of a sharp learning curve at first, especially if you have never coded before. You may feel a bit overwhelmed at first; nevertheless, you will find that the amount of syntax (vocabulary) needed is small, the structure (lines of code) is simple and can be applied repetitively.

You will only learn by trying, and by trying you will learn best when things go wrong. If you make a mistake or it goes horribly wrong, you will learn something (probably patience). Try something new, see what happens, explore, be creative. All you need is a computer (and internet) and a desire to learn. It may well take you out of your comfort zone, but that will be a good thing.

Whatever your motivation for this, I am pretty certain you won't be disappointed. Take the time to play, experiment, and create. See what happens (or in some cases doesn't happen). All this time your brain is being rewired which can only be a good thing. Similar to learning a new language, a musical instrument or learning to draw or paint.

I will be putting very short videos together on my website www.ElegantAI.org that will hopefully help you in this process. There will be other areas of coding that you may be interested in once you have mastered the basics, such as Machine Learning, making Games or Robotics.

These tutorials are spread over several modules. Each module is further split into units. The tutorials take you step by step; it is up to you to decide how fast or slowly you take it.

Creative Coding or Algorithmic Art is the process of creating art using code and some very basic maths. You write lines of code; when these are combined together, they are like a set of instructions called algorithms. A complete programme is called a sketch. Think of this as a recipe where you will have a bunch of ingredients (variables), and a set of instructions (algorithm); then finally, you cook or bake and you have your final product (sketch). Get something in the wrong order, and it may not quite work out as you hoped.

To achieve this, we need a simple, intuitive, user-friendly, and creative coding language. This is where p5.js comes in. It was initially designed for creative people to turn their code into art, although it can be used for machine learning, games, and even robotics.

This course has two primary functions: one, to teach the basics of coding, and two, to give people beginner skills to express their creativity. You can do both with p5.js as you follow this course.

The coding language in question is called p5.js. It is a JavaScript library that has a canvas onto which you create your art. It is what is called open source and completely free. You don't even have to download anything; it is all done in the browser.

This also means you can use any machine that has a browser. A computer, laptop, tablet, Chromebook, even your smartphone or Raspberry Pi. The first unit will help you get started, taking you through the editor you will be using. It is a quick read looking at its main features.

Using p5.js, you can do more than create static images; you can also create dynamic, moving, interactive art. As well as 2D shapes, you can explore 3D shapes as well. You can import images, videos, and music. This is a very flexible and powerful coding language. It is easy to learn, but it becomes as challenging as you want it to be.

To help you, I will include some resources on my website www.ElegantAI.org, which, as the name implies, focusses on Algorithmic Intelligence (Machine Learning) with p5.js.

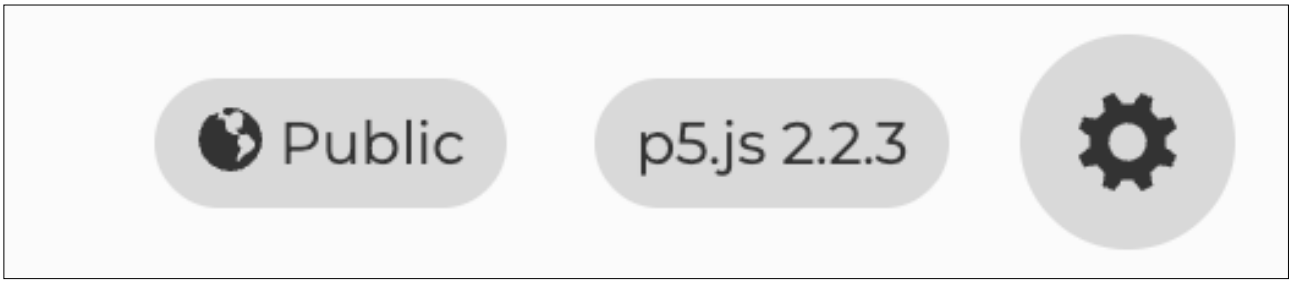
! Version Update

As of writing (September 2026), the default version of `p5.js` is `2.3.x`. There do seem to be a few bugs that on very few occasions don't render the canvas as one would hope or expect. At the moment (until those bugs are fixed), I would highly recommend that you change the version to `2.2.3`.

You can change it by clicking on the middle button shown in the image below. This will give you a drop-down menu, and you can select a different version.

Also, if you click on the `Settings` button (the cog thing on the far right), you will get the settings page. Scroll right down to the bottom and switch the `Infinite Loop Protection` **`off`**. This is on by default every time you change the version.

Figure 1: Version 2.2.3



Screen Arrangement

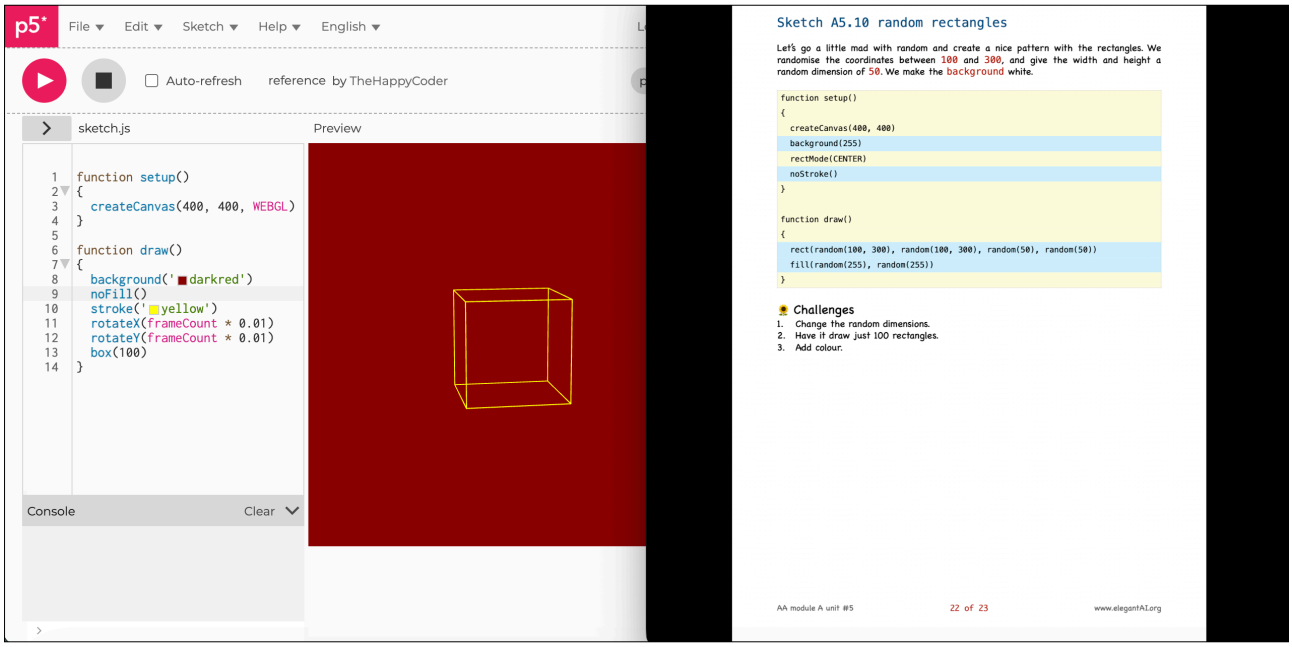
There are a number of ways you can use this workbook:

Suggestion #1: Have it on another device, for instance, a tablet (or even a smartphone), while you work on it on your computer.

Suggestion #2: Split your screen so that you have the editor and the workbook on the same screen using the Kindle app. This depends on the size of your monitor.

Suggestion #3: Download it as a PDF and print it out; this should be possible as I ticked that box to allow it.

Figure 2: Suggested arrangement





Videos

To help with the process, I will include short (less than five minutes) videos on my website that may also help with how to get started. They are very simple. These videos are approximately five minutes long; they give a brief visual explanation to accompany the workbooks.

I will be adding more in the near future.

video #1: Introducing the web editor

video #2: The web editor buttons

video #3: The default code

Resources

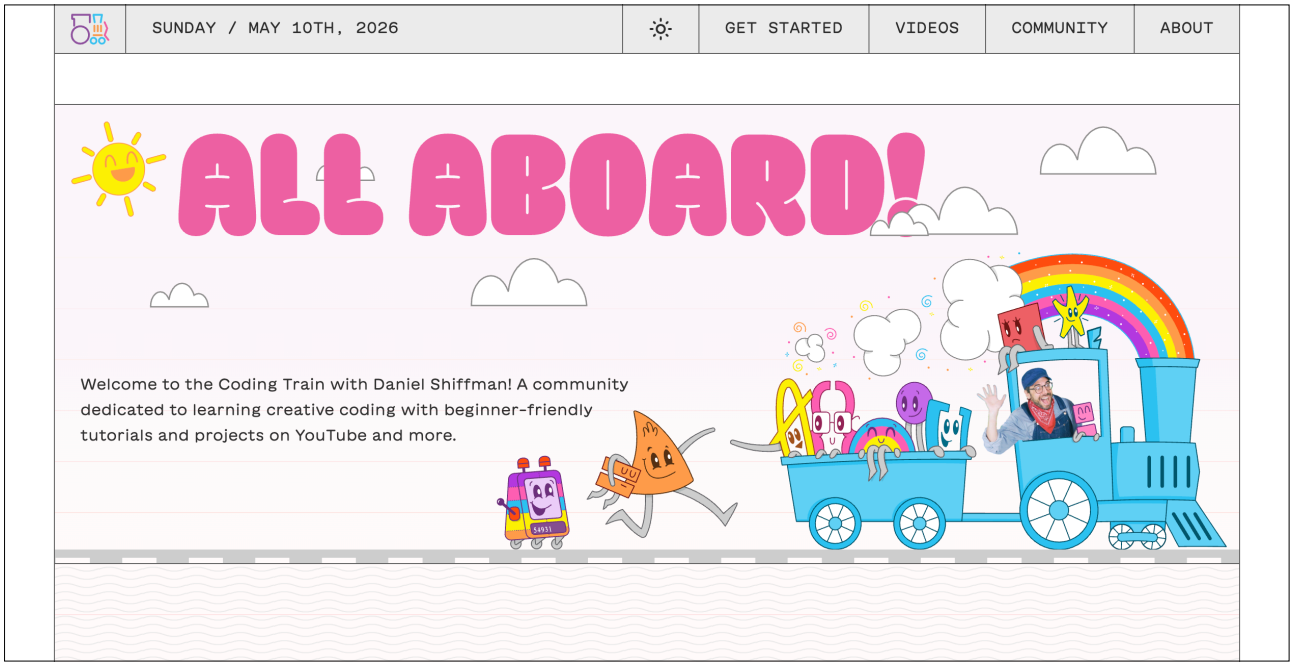
There are a number of additional resources that you may find very useful. The main ones I would encourage you to look for are based around The Coding Train, which is the go-to place for all things p5.js, as well as the Processing Foundation website.

The Coding Train is both a website and a YouTube channel run by Dan Shiffman, who has a huge amount of material available.

Suggestion #1:

Website: https://thecodingtrain.com.

Figure 3: The Coding Train Website



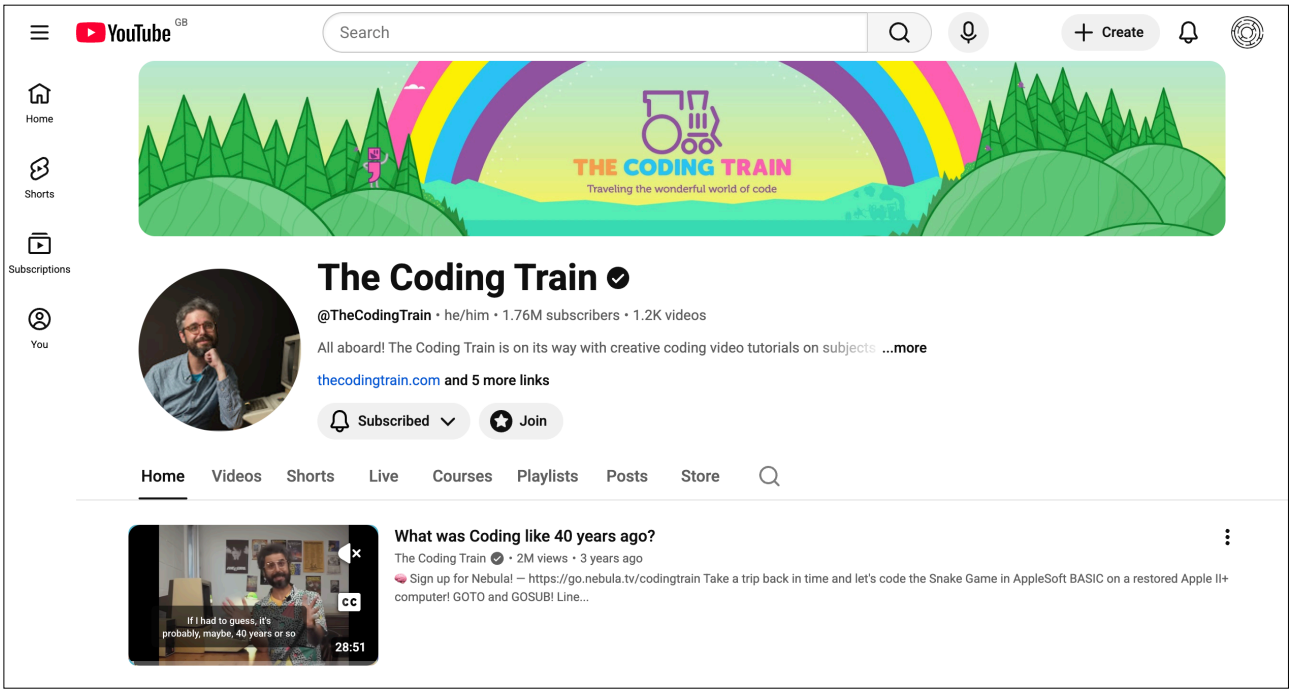
Suggestion #2:

YouTube:

https://www.youtube.com/thecodingtrain

His videos are entertaining as well as very informative. There is a wealth of ideas, challenges, and explanations. Highly recommended.

Figure 4: The Coding Train YouTube Channel



Suggestion #3:

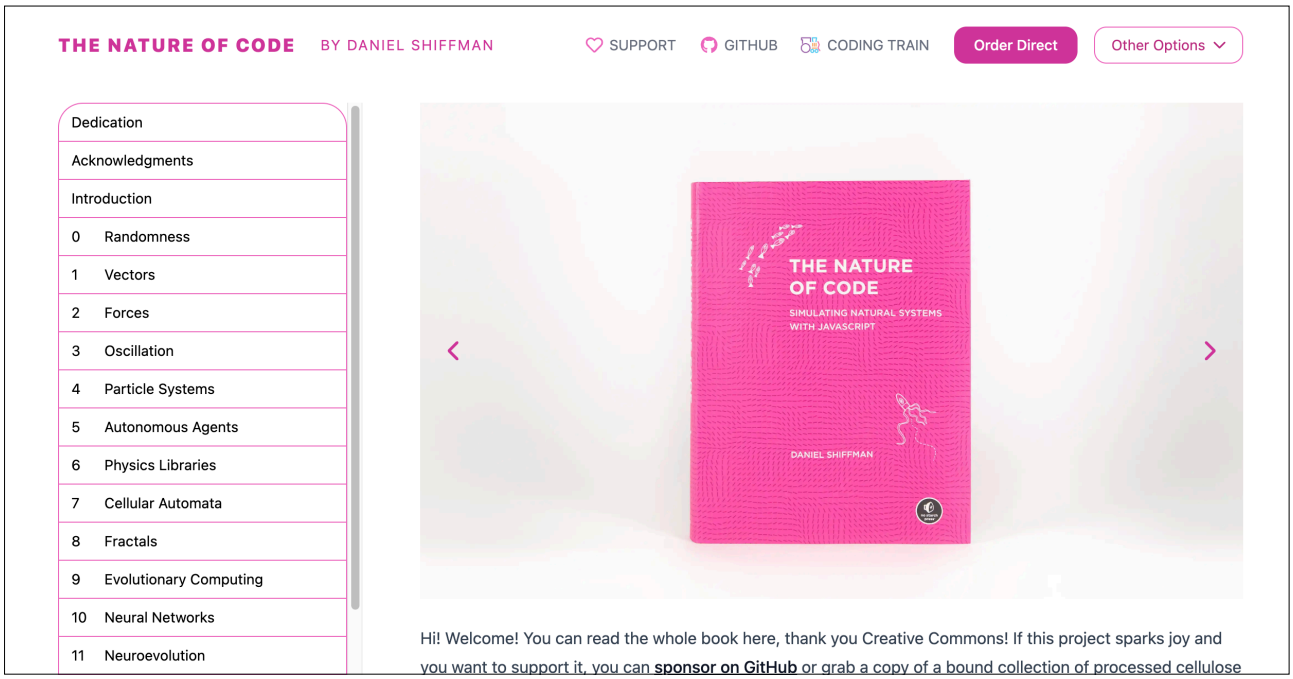
Dan Shiffman's book: The Nature of Code

You can buy this book or use it online at:

https://natureofcode.com

It is all based on p5.js and looks at how you can simulate nature.

Figure 5: The Nature of Code



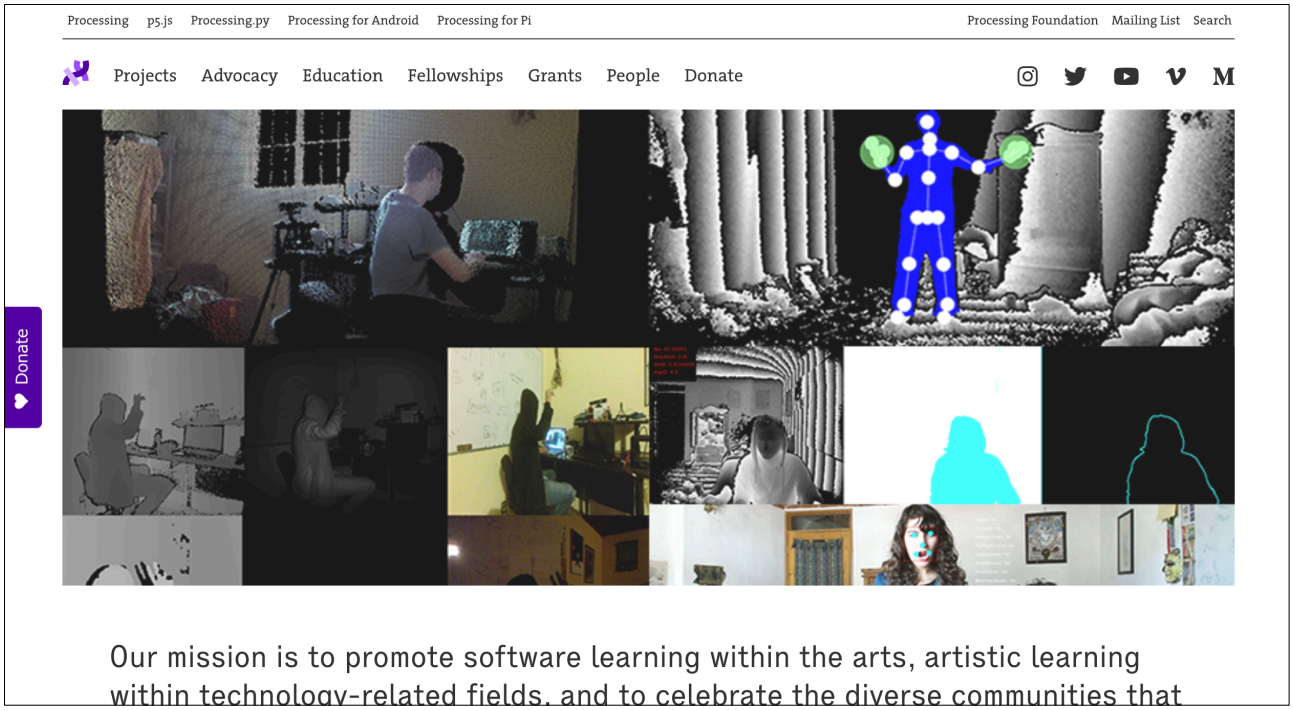
Suggestion #4:

Processing Foundation website at:

https://processingfoundation.org

This is where p5.js originated and is the parent of it, as you might say. It has other resources (the tiny top tab) and is again a fascinating well of information.

Figure 6: The Processing Foundation



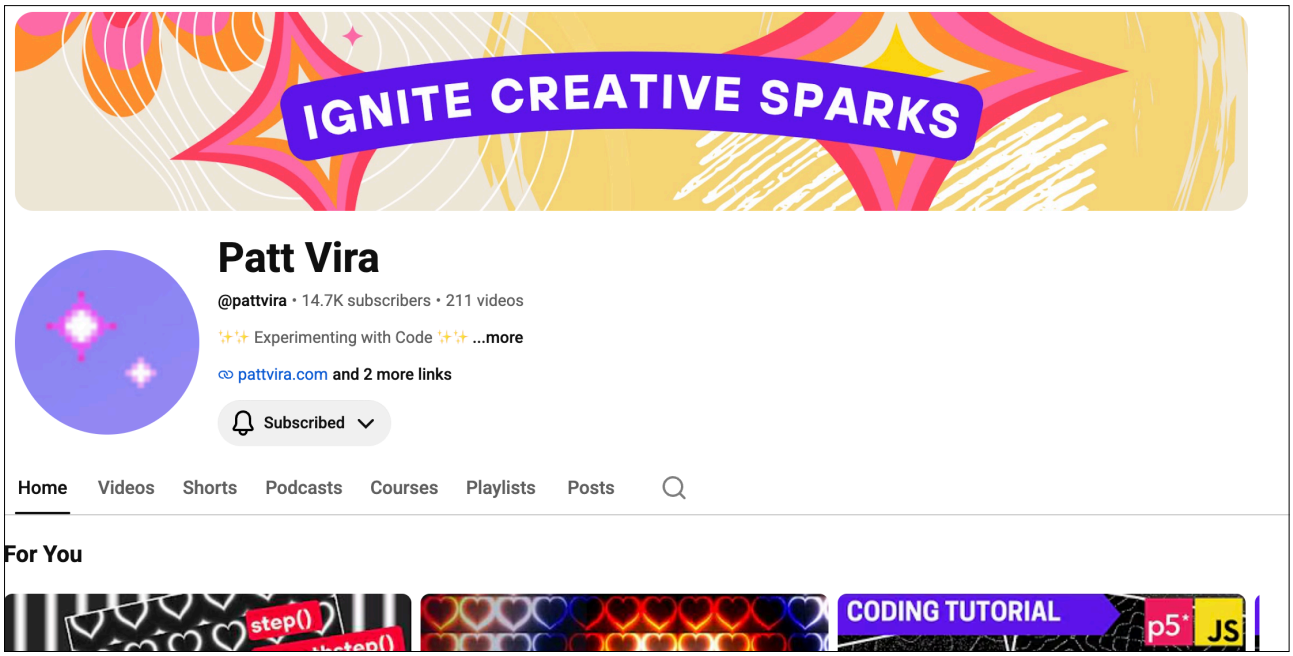
Suggestion #5:

Pat Vira at:

https://www.youtube.com/@pattvira

A really good YouTube channel that explores art with p5.js, some brilliant ideas and demonstrations. She also has a website with a link on her YouTube channel.

Figure 7: Pat Vira YouTube Channel





Workbook #1 Quick Content Summary

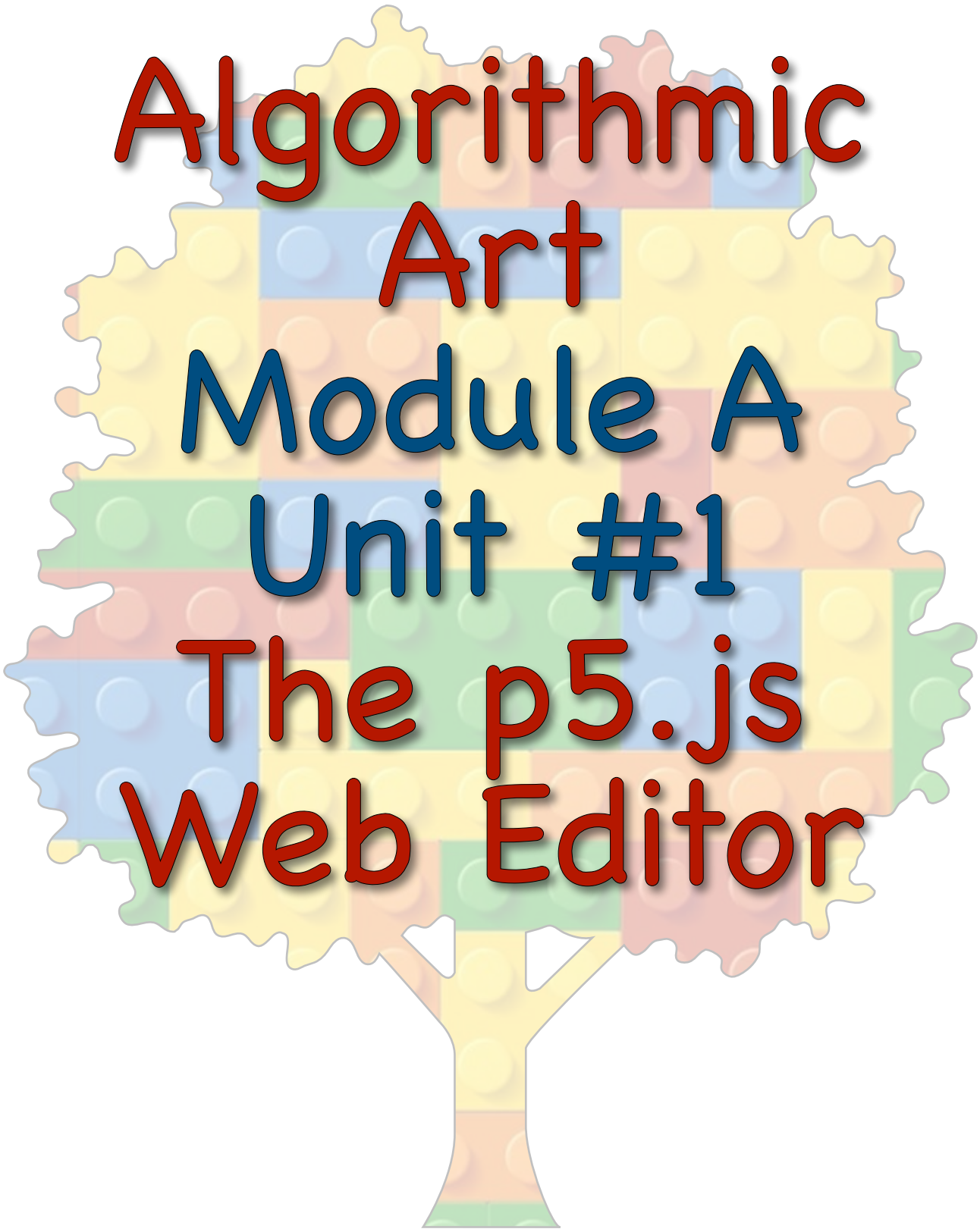
Each unit will explore a specific aspect of coding. The main areas we are going to visit are the range of 2D shapes you can use. How to draw them and colour them. Then we move onto the use of loops. These are integral to coding as they will do a lot of the repetitive coding you would otherwise have to handwrite. One of the unspoken objectives for coders is to write with the fewest lines of code as possible.

You will be facing a lot of new concepts. Please don't worry about whether you understand it straight off. You won't. But as you progress, you will use them time and time again until they become intuitive. The more you practice just typing things in, the more memory muscle you build up. Practice makes perfect.

So, persevere and be intentional. Have a go at the challenges, but they are not obligatory, just to get you thinking.

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.


Algorithmic
Art
Module A
Unit #1
The p5.js
Web Editor



Module A Unit #1: p5.js web editor

The web editor is where you type your code. The code is a set of instructions which you then execute (run). All being well, it should do exactly as you expect. Unfortunately, that may not happen because it will do exactly as you tell it, and this is where the fun comes in. This is a problem-solving activity. It cannot read your mind nor understand your intentions.

The code that you type into the web editor needs to be understood by the computer. It is quite forgiving, but a simple omission such as a comma or a bracket can render the whole code broken. Then you have to find where it all went wrong; usually, it is something very, very simple, but sometimes you may need to rethink your whole approach. It is extraordinarily logical and pedantic at times.

The web editor interprets the code that you type into it. It has been created by someone who wants to make it as human-friendly as possible before translating it into `0s` and `1s` for the computer to understand. In this first unit, we are going to have a good look at the web editor before we use it.

As the same suggests, the web editor is located on the web; you access it with your browser (preferably Chrome). Other editors, you would need to download an IDE (that is what our web editor is) before you can do any coding. This system is not without its drawbacks (need the internet), but it is then accessible on any machine anywhere you go.



Question 1 – What machine do I need?

This is probably the first question you might ask: Which machine should you use or not use for this tutorial? The simple answer is anything with a web browser will work for nearly everything. I do all mine on an iPad, but you can do it on a Chromebook, PC, Mac, laptop, smartphone, tablet, or a Raspberry Pi.

You do not need a state-of-the-art computer or a high-end one. Almost anything will do as the code runs in the browser, not your machine.

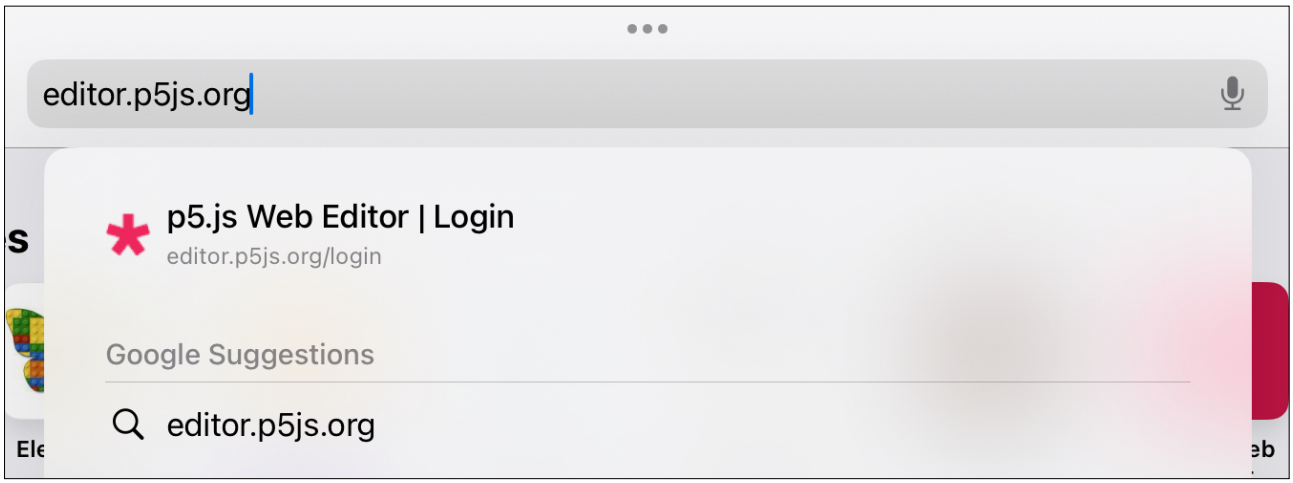
Question 2 – What is the editor?

You don't need to download any software; you simply type:

`editor.p5js.org`

into your web browser, see Fig.8, and off you go. Most web browsers support it, but if unsure, then use Chrome or Safari. If you have a favourite one, then use that and see how you get on.

Figure 8: type `editor.p5js.org`



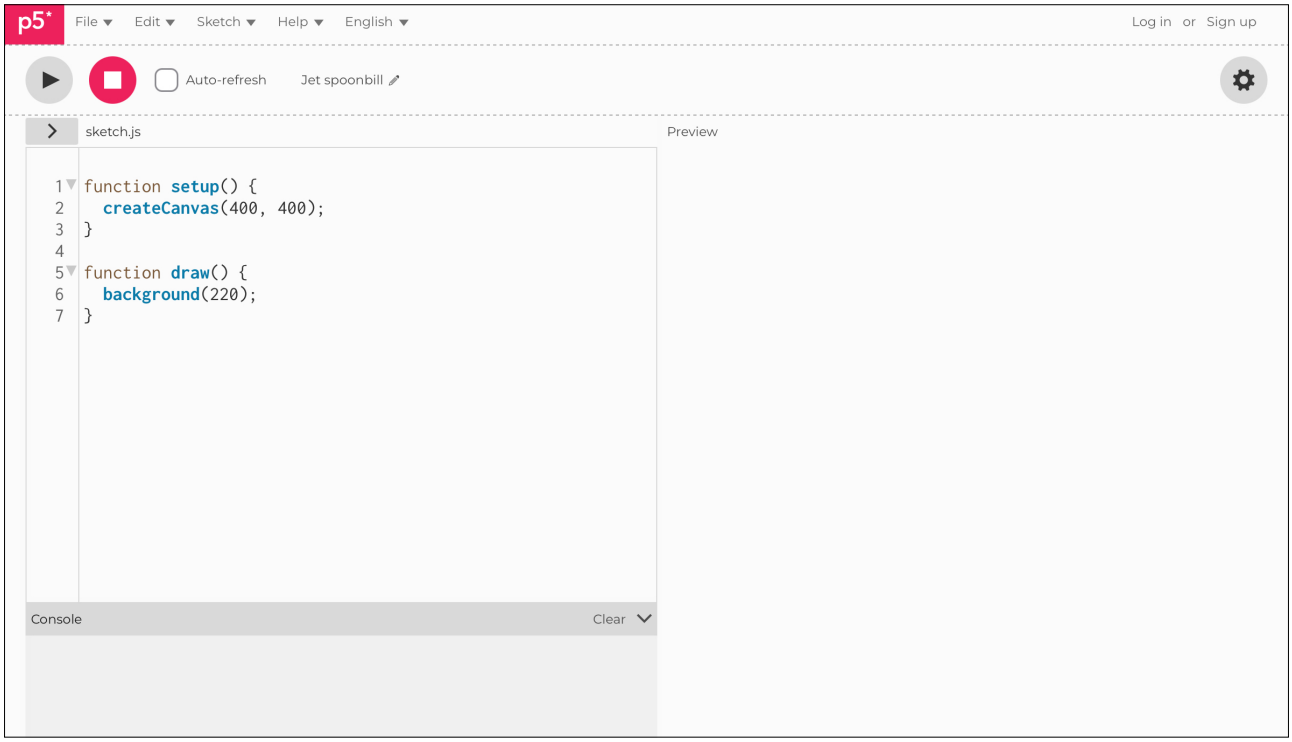
When you first start, you get a page like this shown in Fig.9. You have a `main menu` across the top left-hand side. On the right-hand side, you have `Sign up` and `Log in` tabs. Below that, you have a number of buttons, a checkbox, and a text box.

Below that, you have two main panels. In the first panel (left-hand side), there is a column of numbers which are the line numbers for your code. The second panel is where you type your code. The third panel is the canvas, where stuff happens; the canvas will appear when you start to run your code.

If you press the run button (dark grey triangle in a light grey circle), the canvas should appear as a grey square where it says Preview. Underneath the panel for your code is another one called `console`; this is where your error messages and other information you request (in your code) will be sent and seen.

I will go through all these in a little bit more detail shortly, but for now, click on everything and anything to get a feel for the buttons and tabs. Everything is pretty intuitive, and the best thing to do is learn through playing with the buttons and seeing what happens.

Figure 9: the editor





Question 3 – What does the update mean?

Even as I write this tutorial, there have been some updates. This is true of coding generally; they are always improving it as demand and use change. This is also true for p5.js and the web editor. Every so often, they have a new version, which generally goes unnoticed, but now you will be able to see which version you are using. They now include a tab for selecting earlier versions.

The current and default version is now version 2.3.x. There are some bugs in these versions and so I would recommend changing it to version 2.2.3 which is pretty stable. However, by the time you read this version 2.3.x maybe fine. They are ironing out the wrinkles.

I have made everything compatible with version 2.x, to do that I have had to make very few changes if any. What I recommend at this stage is to use version 2.2.3.

Figure 10: the version

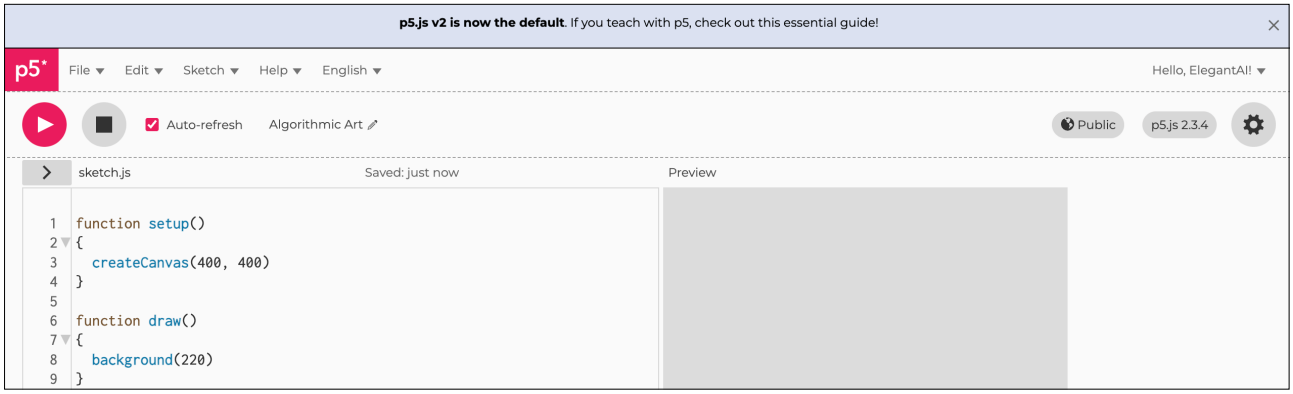


Figure 11: click on the pink highlighted button

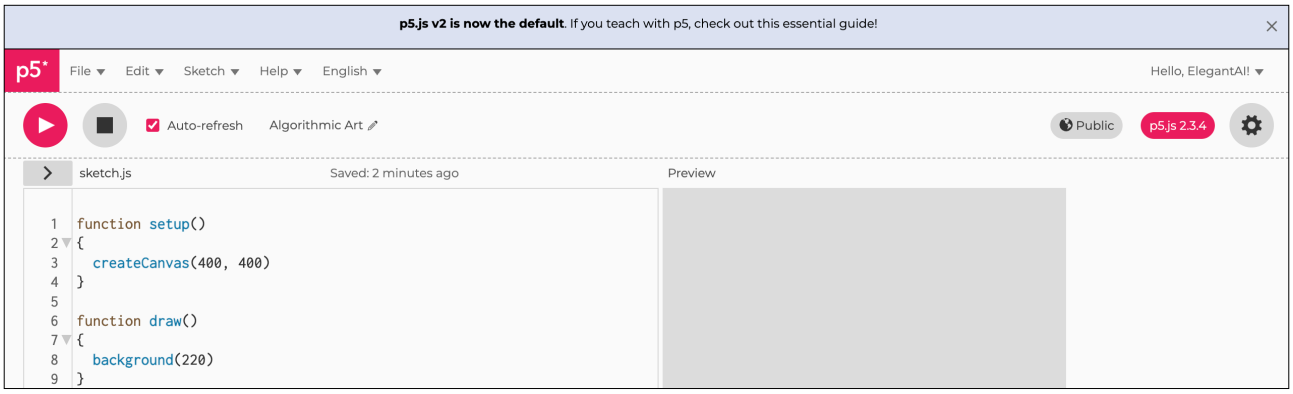
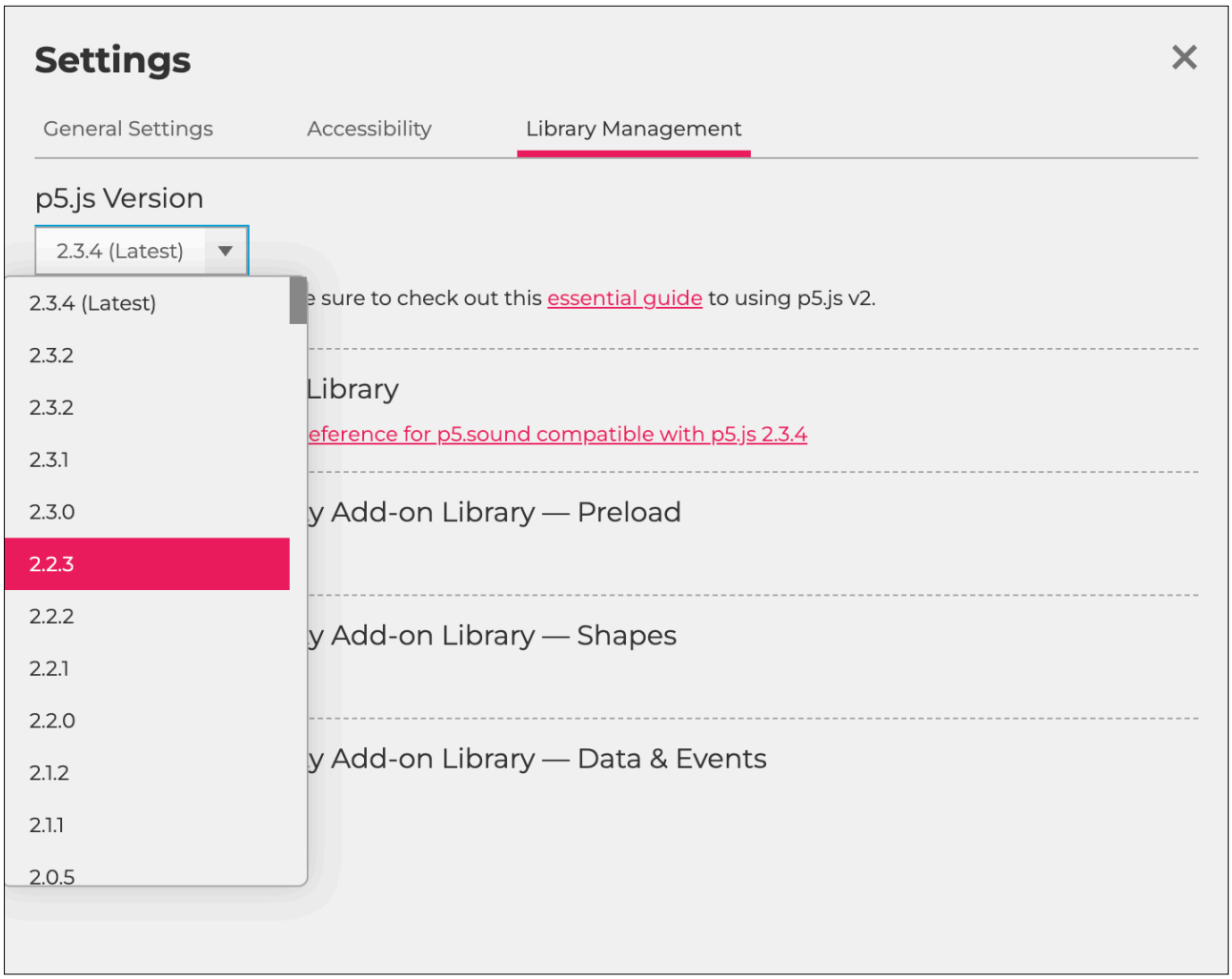


Figure 12: select version 2.2.3

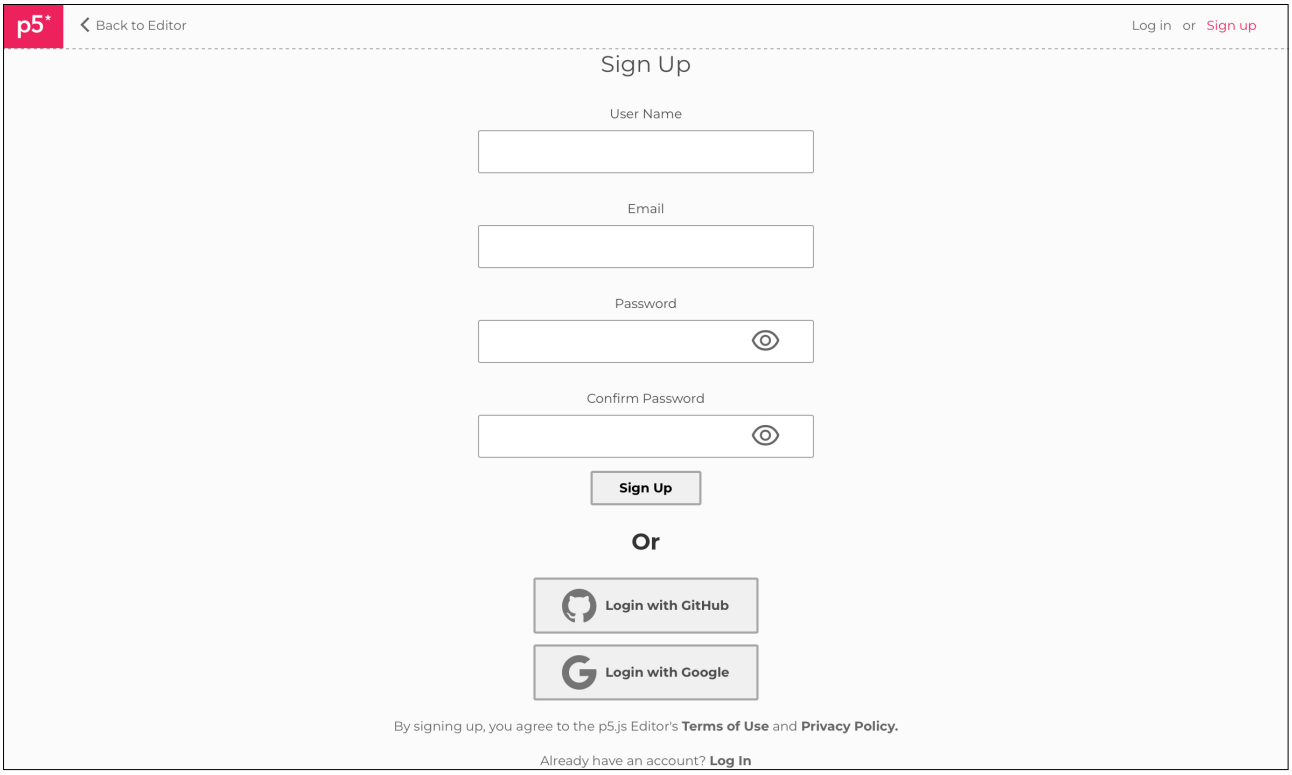


Question 4 – How do you sign in?

Although you don't have to sign up for anything and you can code straight away, you won't be able to save your code unless you do create an account. I would recommend setting one up right from the start. They have never, ever pestered me; it just used to log in.

Use one of the methods shown below, and you are good to go (again). You can log in with your Gmail or sign up with another email account. Test it all out first to make sure it works. I use `Login with Google` (recommended), but you can always use one of the other methods. Obviously, you will need a Gmail account for that. Sign up first and then log in (see Fig.13).

Figure 13: signing up

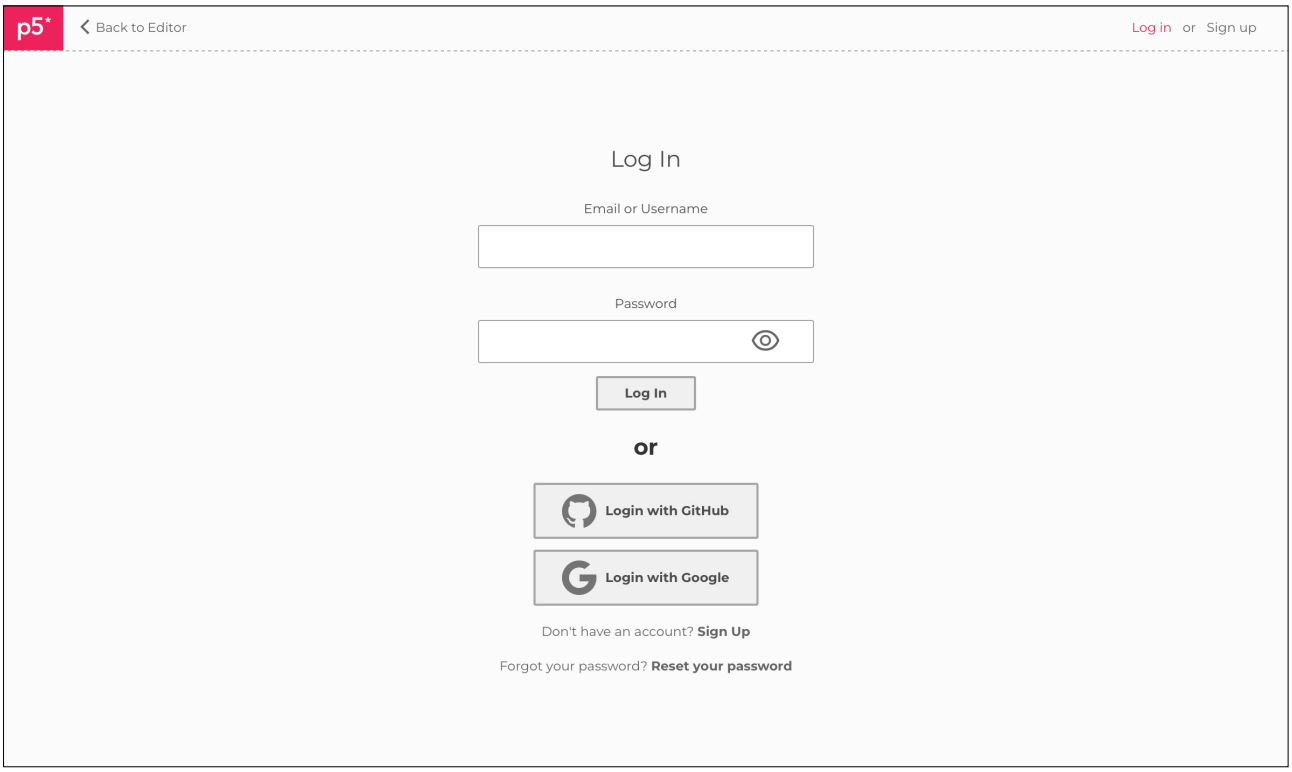




Question 5 – How do you log in?

If you have set up an account then you can simply log in.

Figure 14: logging in





Question 6 – What is the default sketch?

At the very start, you get a default sketch. This includes two functions and some curly brackets. These two functions are important and are built-in functions. The first one, `function setup()`, is to set up how you are going to see and use the sketch; this runs through only once and usually includes the size of the canvas.

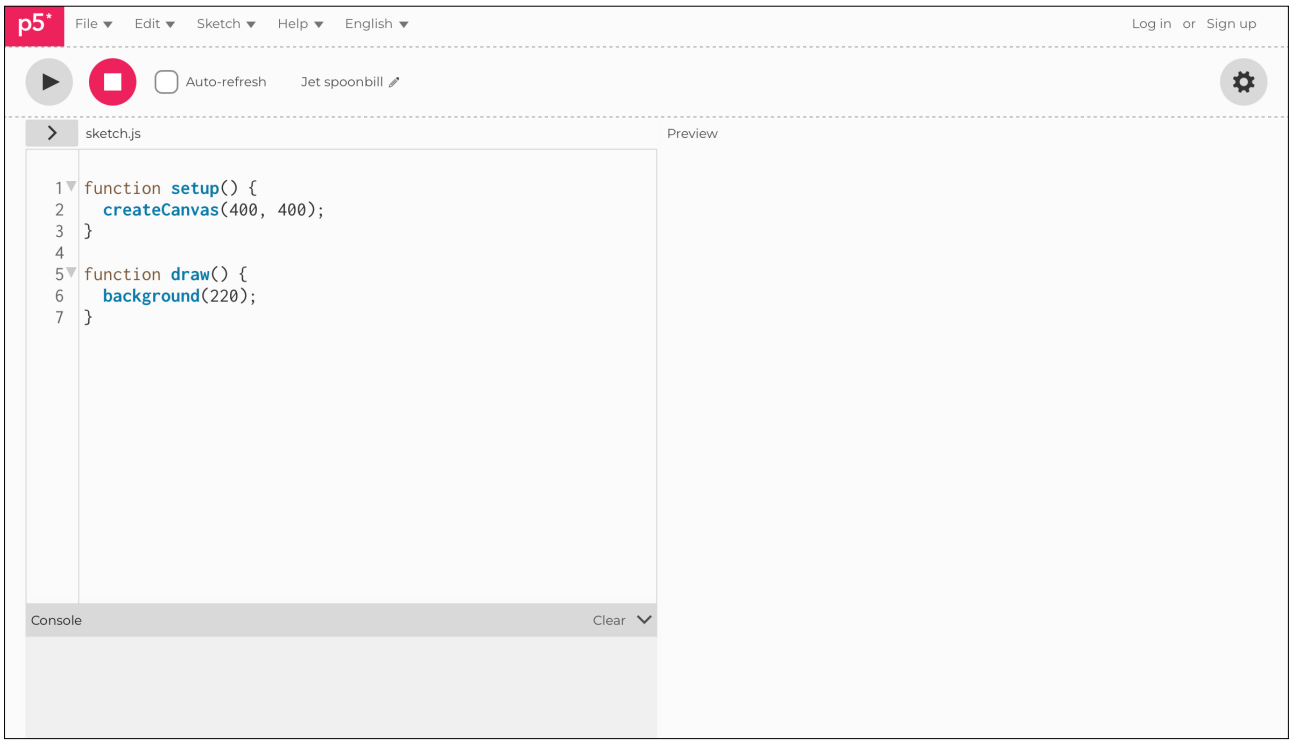
The second one, `function draw()`, is a loop; it draws on the canvas continually. This is where much of your code will go, although in some cases, you won't even have a `draw()` function, but more on that much later.

These two functions are prebuilt and are what are called predefined, and so you cannot use those named words/functions anywhere else.

Inside the curly brackets is where your code sits. At the moment, you have the canvas size of `400` pixels by `400` pixels in the `setup()` function. In the `draw()` function, we have a grey `background()` with a value of `220`. This is the grey value from `0` (black) to `255` (white) and shades of grey in between. We will talk more about colours later.

You will also notice the semicolon (`;`) after some lines of code. Most people who code with p5.js include this to mark the end of a line of code. I have omitted it to keep the code cleaner and easier to read (see Fig.16). This is not essential, but you may want to revert to it later. It has never been a problem in not using the semicolon, but it is up to you to decide whether to use it or not. This is just my preference, and I know I am in the minority here.

Figure 15: default sketch



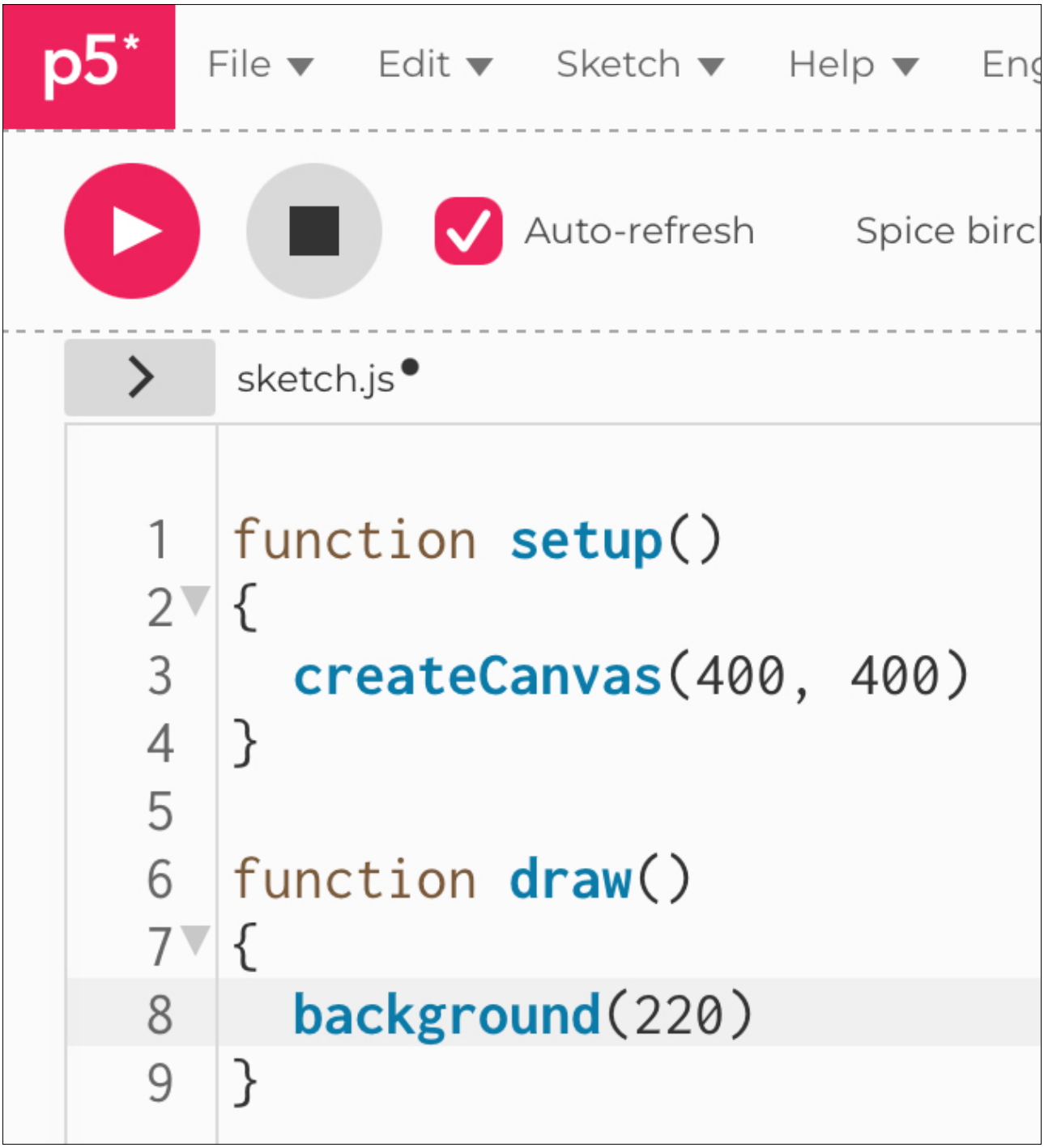


Question 7 – Why does your code look different?

The above is the default format, but the format I will present in my tutorial will be slightly different. You will see that the first curly bracket starts on the same line as the function name; it is how most people will use this. But I want to make the curly brackets { } stand out more, so I change the format to include the first curly bracket on the next (or first) line after the function name. Again, this is just my preference, to improve clarity. See Fig.16.

All the lines of code are placed inside a set of curly brackets { }, there are always in pairs, like bookends. The code inside a set of curly brackets is usually indented with two spaces; this is the default setting and, although not essential, makes the code readable and stands out clearly.

Figure 16: my format

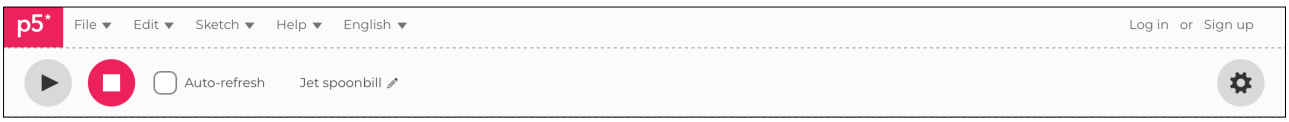


Question 8 – What do all the buttons do?

You will see a row of buttons across the top. It is worth spending the time getting to know them. The first thing to consider is how to save your work as you go along. If you are working on something one day and want to continue the next day, then you need to save your work. Also, you may find that your code may crash (for various reasons). This is why you need to save your work as you go along, not only at the end of a session. You can set it up to save your work automatically as you go along, but be aware there is no undo option.

It is worth exploring these buttons and tabs to familiarise yourself with what they look like and what they offer before you go very far into your coding. I will highlight the most important ones, but they all have their function and purpose.

Figure 17: main buttons



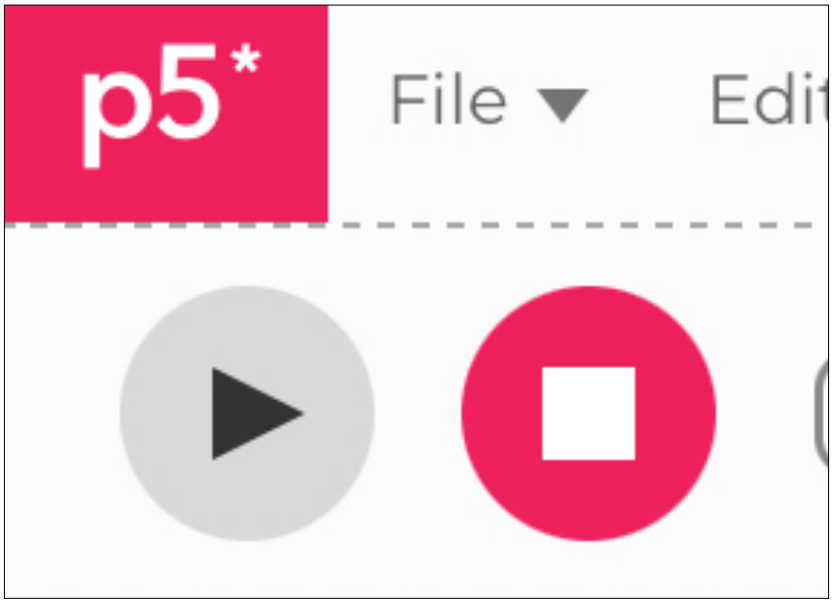
Question 9 – How do I run my sketch?

The first button you come across is the run button. It looks like the play button on a device you might have used. After you have written your code, you will want to run the code to see what happens. When you click on the run button, it will do just that.

To stop the code running, you will need to press the stop button, which is next to it, shown with a square in a circle, again pretty intuitive. The programme will keep on running the code until you stop it.

So every time you write some new code, remember to press the run button each time. You can also set it up to run automatically every time you add new code, but I wouldn't recommend it until you really know what you are doing. This is because it can cause problems as it tries to run incomplete code, at which point you will lose any changes since your last save.

Figure 18: run and stop



Question 10 – Can I change the name of my sketch?

You will have seen a box with a couple of random words in it. This is a randomly generated name for your sketch; in my example below, it generated `Jet spoonbill`. There is a little pencil symbol next to the words. This means you can edit the name to something more meaningful or just leave it as it is if you wish. This is entirely up to you; a new set of words is generated each time you start a new sketch. This is another reason for creating an account so that you can save your sketch under that name and access it again later on, carrying on from where you left off.

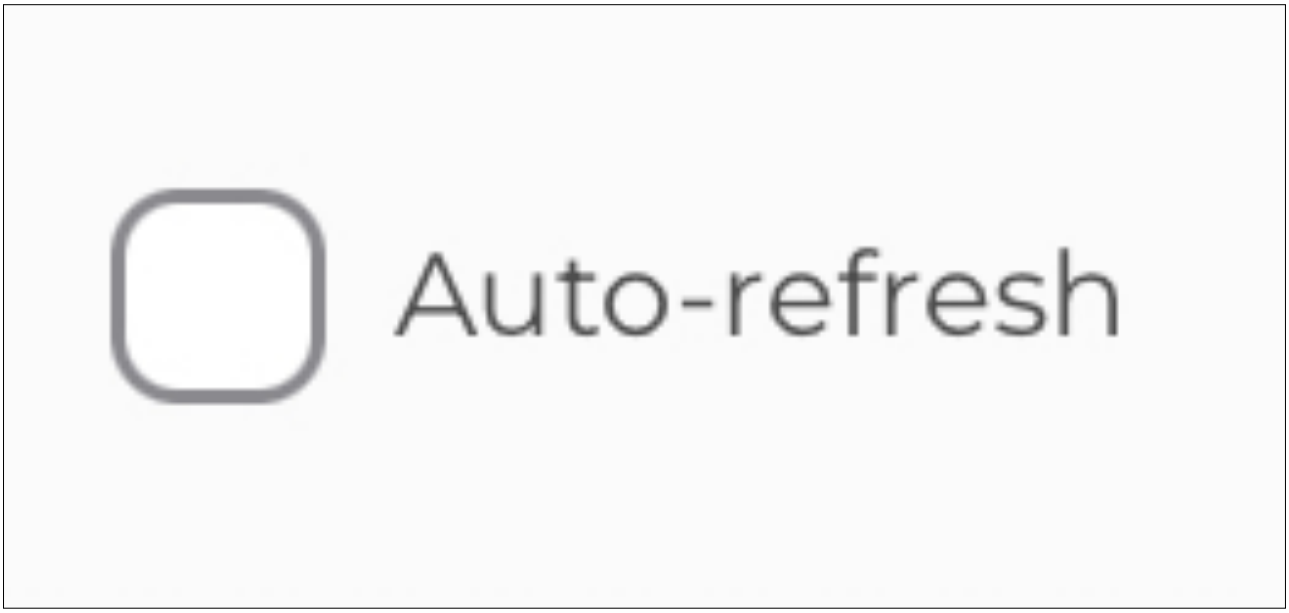
Figure 19: naming sketches



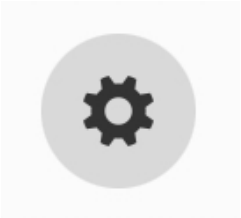
Question 11 – Is the auto refresh useful?

The button before the sketch name box is the auto-refresh button; if that is highlighted or selected, then I recommend that you unselect it by clicking on it. It may be unselected by default, in which case for now leave it as is, but later on when you are more confident in what you are doing, you may like to select that option.

Figure 20: auto refresh



Question 12 – What do the other settings do?



Here we have a button on the far right-hand side which looks like a cog wheel (see opposite). If you click on it, you get a `settings` menu which is split into two parts. The first one is `General Settings` and the second is `Accessibility`.

In `General Settings`, you will have a list of options, most of which (if any) you don't have to alter:

Themes
Depending on what works best for you, choose a theme from a choice of three that is easiest on the eyes. I quite like the high contrast, but for this tutorial, I use the default `light`.

Text Size
Is pretty obvious, but bear in mind that if you make it too big, you will get long lines of code wrapping themselves onto the next line, which I try not to do for this tutorial. Choose the size that works for you.

Auto Save
It might be worth keeping it on for now, but sometimes I want to go back to the last version I saved manually, and if it has saved a later version, it can be an issue, but it is swings and roundabouts whether to have it or not. My preference is to leave it off and to manually save the sketch after each significant change. This is a discipline that is easy to forget. Leave it on if unsure at this stage; or just practise saving as you go along.

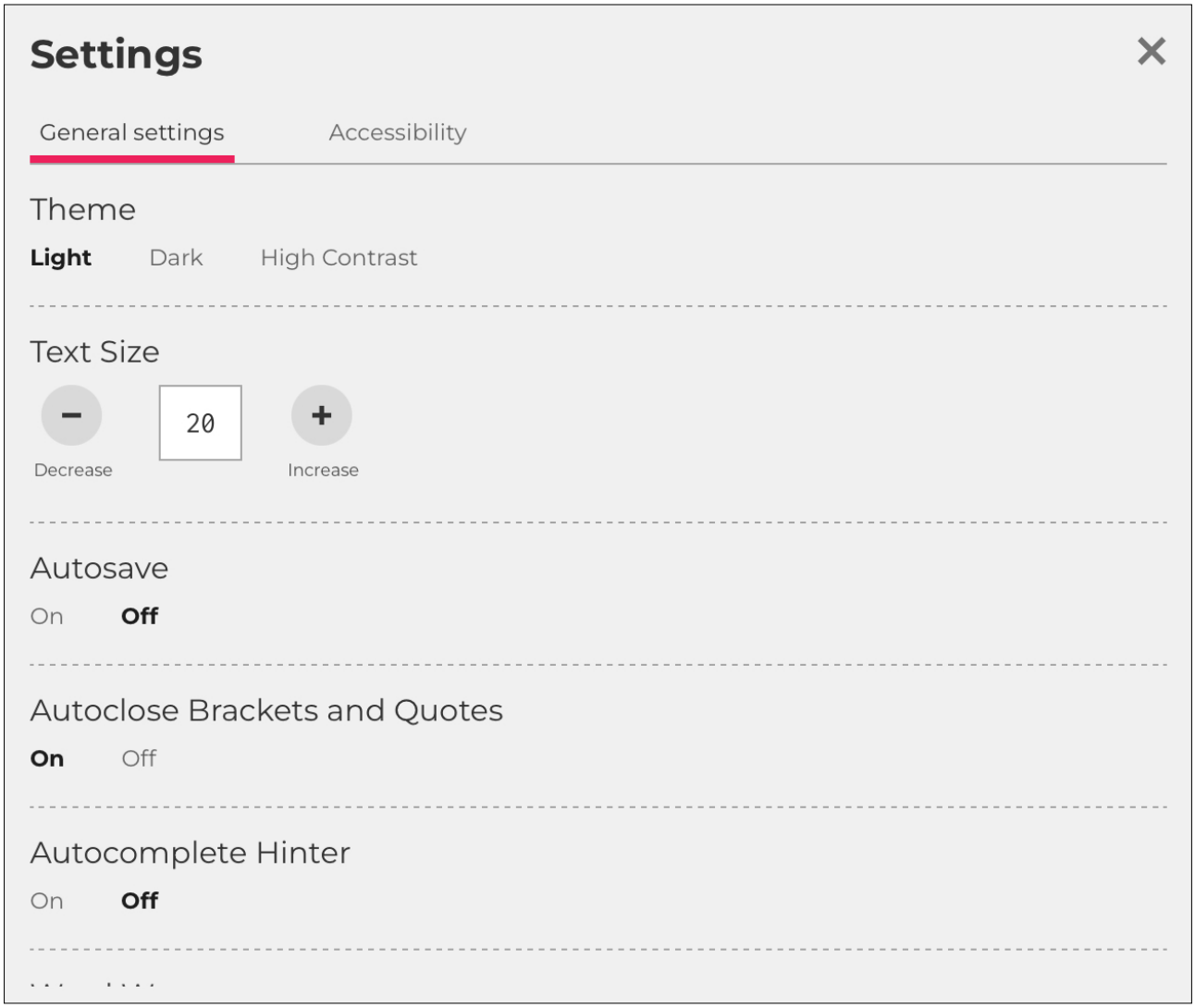
Autoclose Brackets and Quotes
This is a useful one to keep selected; you don't have to, but I would recommend it. You will see the difference if you switch it off; it saves you a bit of time, as you will see.

Autocomplete Hinter
I found this one very annoying if left on. I suspect some coders like it and use it a lot, but I found it got in the way of my coding as it tried to predict what I was about to write; my strong recommendation is to deselect it if it is on by default.

Text Wrap
This is only an issue if the line of code reaches the edge of the panel. Leave it on if you always want to see the code and not disappear.

Loop Protection
This is a very important one. For some of the examples you will be working through, you will find that it stops halfway through and punches out a warning in the console (window box underneath the code) that an infinite loop has been detected. The reality is that there isn't an infinite loop but just a large loop (usually a nested loop). The `Loop Protection` is on by default. If you get the error message, then simply turn it off in settings. I will endeavour to give you a heads up if it is likely to happen.

Figure 21: general settings





Question 13 – What does the accessibility tab mean?

The second menu is Accessibility.

Line Numbers

You can switch the line numbers off or on if you wish; the line numbers are useful if there is an error message, and it can often tell you on which line of code the error message relates to.

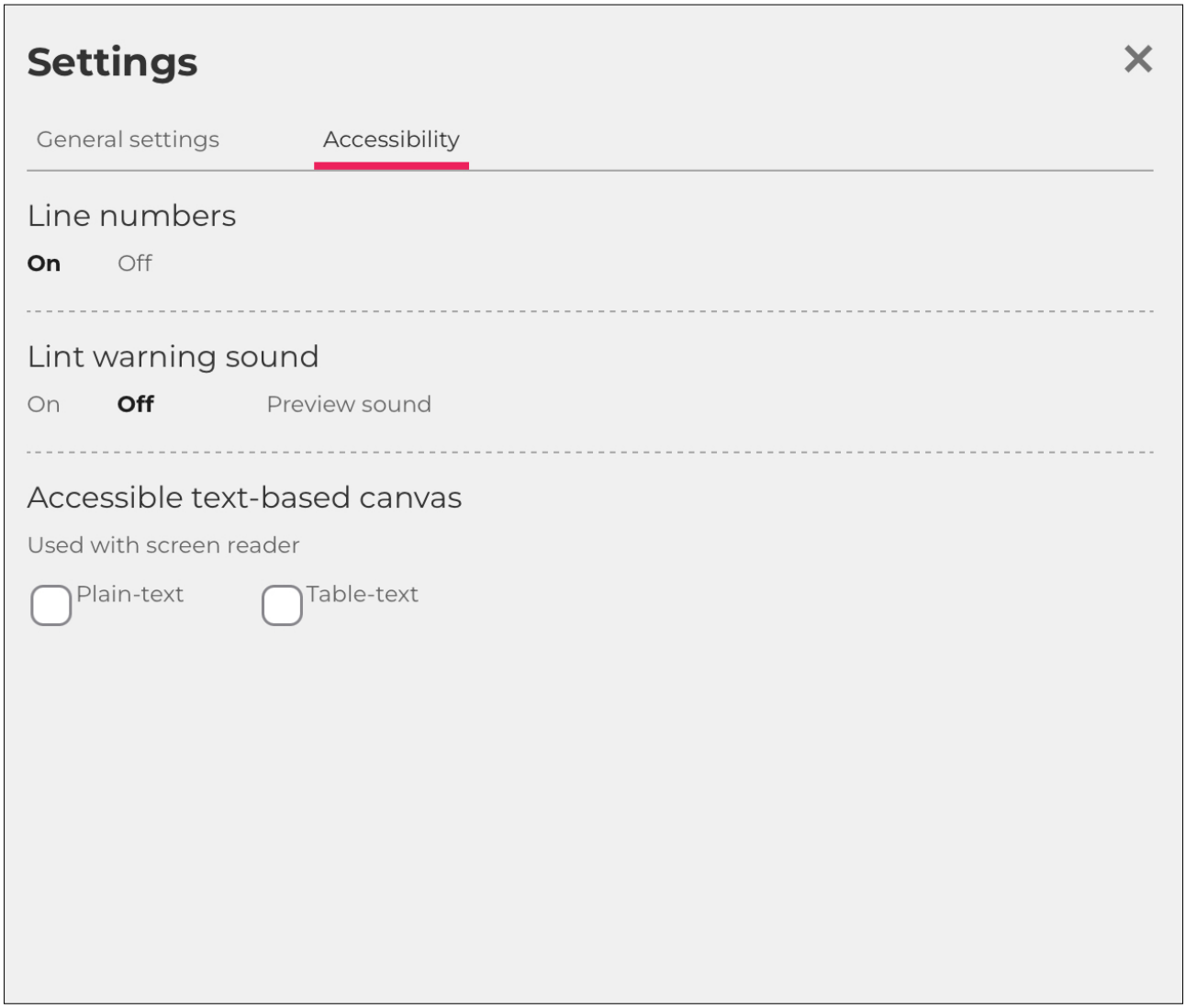
Lint Warning Sound

This gives you an audible sound if it detects an error in the code; again, it is a preference to have or not have it selected; I don't; I think it would drive me mad after a while.

Accessible text-based canvas

Not entirely sure what these do, to be honest, but they may mean something to you.

Figure 22: accessibility



Question 14 – Is the main tab menu important?

We have a list of main menu tabs or headings. If you click on any of the following: File, Edit, Sketch, Help, and English, you will get a subheading list. I will make reference to some of the key subheadings that you are likely to use.

The main heading you will use is File, as it is for managing your sketches. The Edit is also useful for finding words and/or replacing them. If you tidy your code, it will do just that and it will do it in the default format with the curly brackets and the semicolons. The Sketch is where you can add files, folders, run, and stop; there are other ways to do that, which I will show you later (run and stop I have already covered). The Help is where you can get some additional information, and English is where you can change the language used.

Figure 23: main menu





Question 15 – How do I save my work?

Under the File menu, you get either a short menu if you have not saved your sketch, or a longer menu if you have. I am showing you the longer version.

New
This gives you a brand new sketch with the default code and a new randomly generated name.

Save
Use it as often as you can, just in case something goes wrong unexpectedly, so you have a backup up to that point. I generally don't learn many shortcuts, but this is one shortcut worth learning, as you might notice the command symbol plus the letter S; you press both together rather than going through the menu each time. There will be slight variations for different machines and keyboards.

Duplicate
You can find sketches on the internet or one of mine that you want to use. If you have found a sketch that you want to keep, use, or edit for whatever reason, then you can duplicate it (make a copy) and save it under a new name. That way, you can have your own version.

Share
You may want to create a link to your sketch or even embed it in your website. This tab gives you a number of options.

Download
This is something I have never used and am not sure when you would use it.

Open
This takes you to another part of the web editor where you will see a list of all your saved sketches. If you have a lot (and I do), then you can search for specific keywords. Another reason why it is important that you rename your sketch with something meaningful in case one day you want to find it. They are saved chronologically.

You have two other tabs under the Open tab, Collections and Assets:

Collections
This is where you can view any collections you have created as well as creating them.

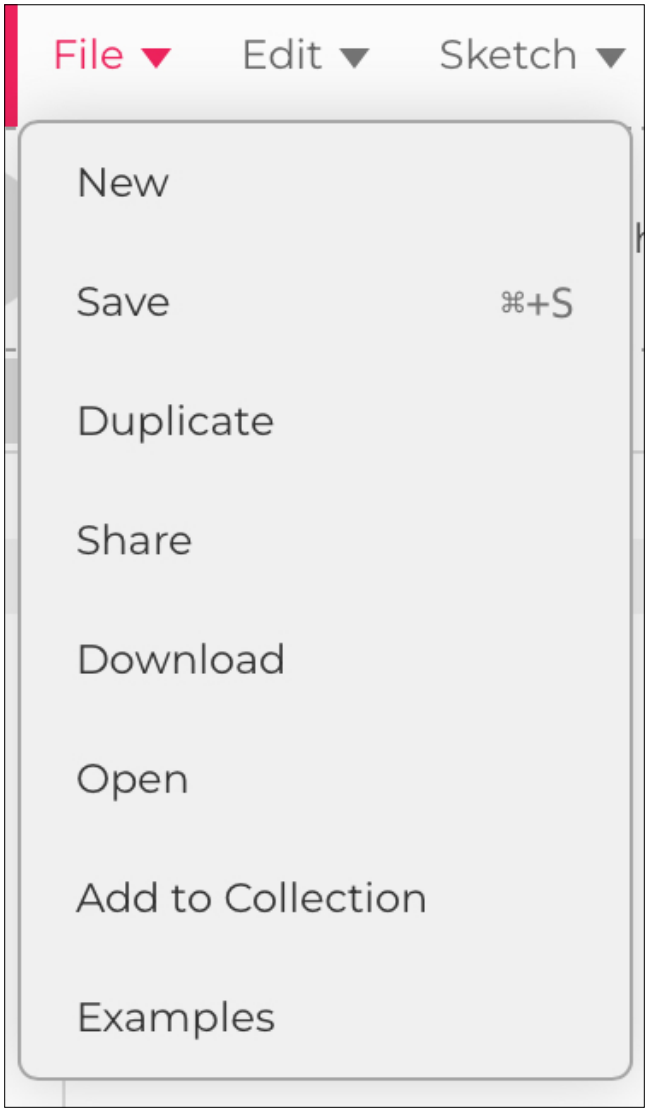
Assets
When you add images, videos, music, etc., this is where you can see them and which files they are linked to. There is a finite amount of memory space to keep them, but there is plenty for many sketches and numerous images and files.

Add to Collection
One way to sort and manage your sketches is to make a collection and add the sketch to it; this is similar to a folder to hold a specific collection of sketches and can be useful if you start creating sketches for different purposes, for instance, games or art.

Examples

A large repository of useful examples, well worth a quick explore to see some interesting stuff.

Figure 24: file tab



Question 16 – How useful is the edit function?

The Edit menu gives you some useful functions.

Tidy Code

This will tidy your code but will put all the semicolons back in and move the curly brackets to where they are normally kept, so it will change the appearance of your code considerably, assuming that you have adopted my format approach.

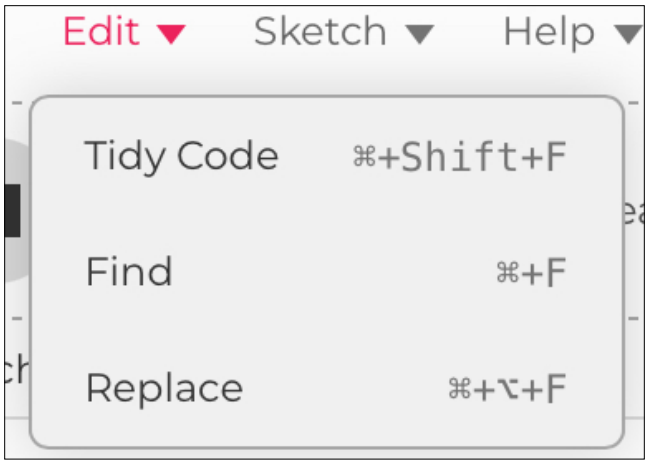
Find

This will search for any specific keyword or phrase in your code.

Replace

This gives you the option of finding a particular word or phrase and an option to replace it. You might use this to change the name of a variable.

Figure 25: edit tab



Question 17 – How do I add files?

The Sketch menu offers an alternative place to run a number of actions.

Add File

This is where you can create another JavaScript file, such as vehicle.js; you can also do this from the sidebar.

Add Folder

This is where you can create a folder to put a collection of files or upload images, videos, models, or data files. You can also do this from the sidebar.

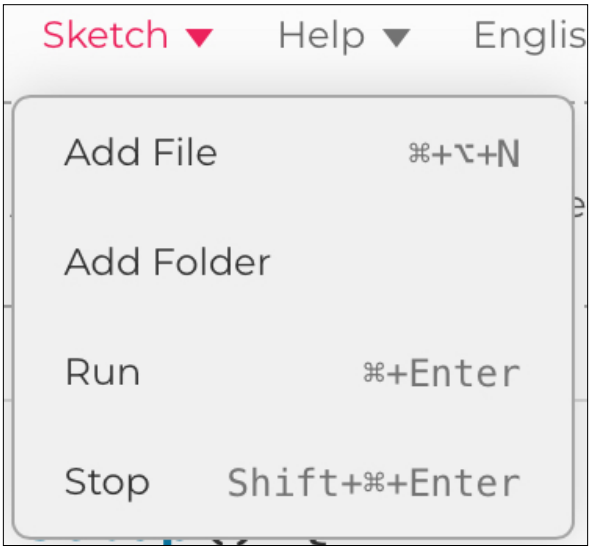
Run

This does the same as the button.

Stop

This also does the same as the button.

Figure 26: sketch tab



Question 18 – Is the Help menu helpful?

The Help menu is quite useful and worth a look at.

Keyboard Shortcuts

There are quite a few, and some are worth learning, such as Save, but others are probably not so useful; it all depends on whether you are generally good at learning shortcuts and would use them.

Reference

This opens up a new tab in your browser. This is an immense resource and can look a bit bewildering. There is a search box, and there are lots of examples and a reference section. It would take too long to go through it, but I would highly recommend having a look at it.

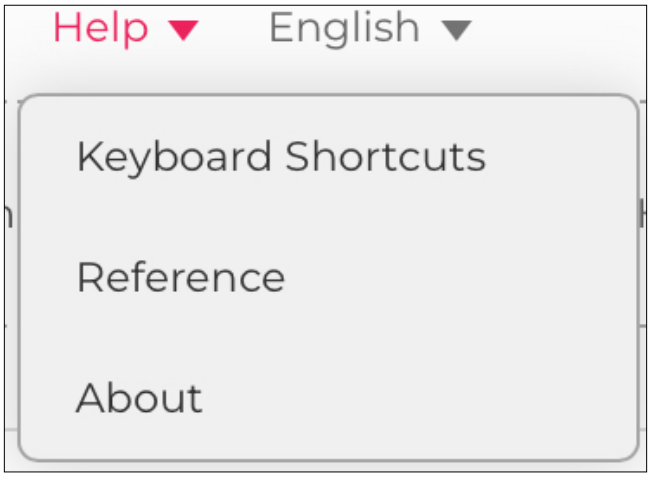
About

This has some links and general information.

English tab

You can change the language to one of many if your first language isn't English.

Figure 27: help tab





Question 19 – What is the console for?

The `console` is the grey-boxed section underneath where you enter your code. This is a very useful function for a number of reasons. It is where you will get error messages and where you can send information. Its main purpose is for debugging, which is another word for problem-solving.

Error messages

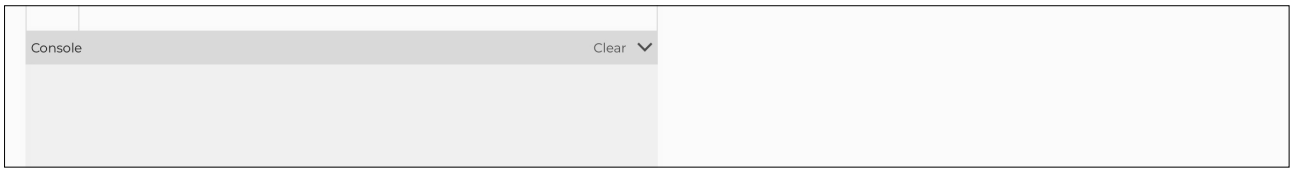
If your editor picks up on some glaring and obvious errors in your code, which can be anything from a missed comma or semicolon to an unknown variable appearing. Mostly you will get something useful and informative, although sometimes it just flags a problem and leaves you to search for it.

Sometimes the error code may be highlighted in red, and it may even give you the line number to identify where the problem is (or the following line number where it does become a problem).

Console log

We can put a line of code in called `console.log()`. This means you are going to log something in the console. Sometimes it can be a piece of text such as `console.log('all done!')`, or it can be the value of a variable such as `console.log(counter)`, which will give you the value of the variable at that time in the code. Another really helpful use is to debug a problem where you are not sure what is happening, for example, to an array.

Figure 28: the console





Final words for the first unit

If you did not understand all the above, please do not worry. I will go through their uses when appropriate. I just wanted to make you aware of them and leave you to play around with them at a later date. I cover some of them in one of my introductory videos.

On the whole, there is a lot to say about the web editor, but most of it is fairly intuitive. The best way is to play and explore, but for now, you can have a read through to see what options are available. The reference section will be very useful later on as you become more confident and competent.

Now, let's get coding and have some creative fun.


Algorithmic
Art
Module A
Unit #2
Your First
Circle



Module A Unit #2: your first circle

This may not seem very challenging, but using the `circle()` function means I can introduce you to many coding concepts that you will use in your journey into creative coding.

Key concepts:

`drawing a circle`
`names of colours`
`variables`
`random`
`operators`
`while() loop`
`for() loop`



Sketch A2.1 your first sketch

First of all, you need to delete the default code and type the following. It is really just the same, but I want to arrange it in a more intuitive way. The semi-colons aren't essential, and the first curly bracket {} doesn't have to start on that line (although it is common to do so).

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

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

Notes

Your screen should look something like the image below.

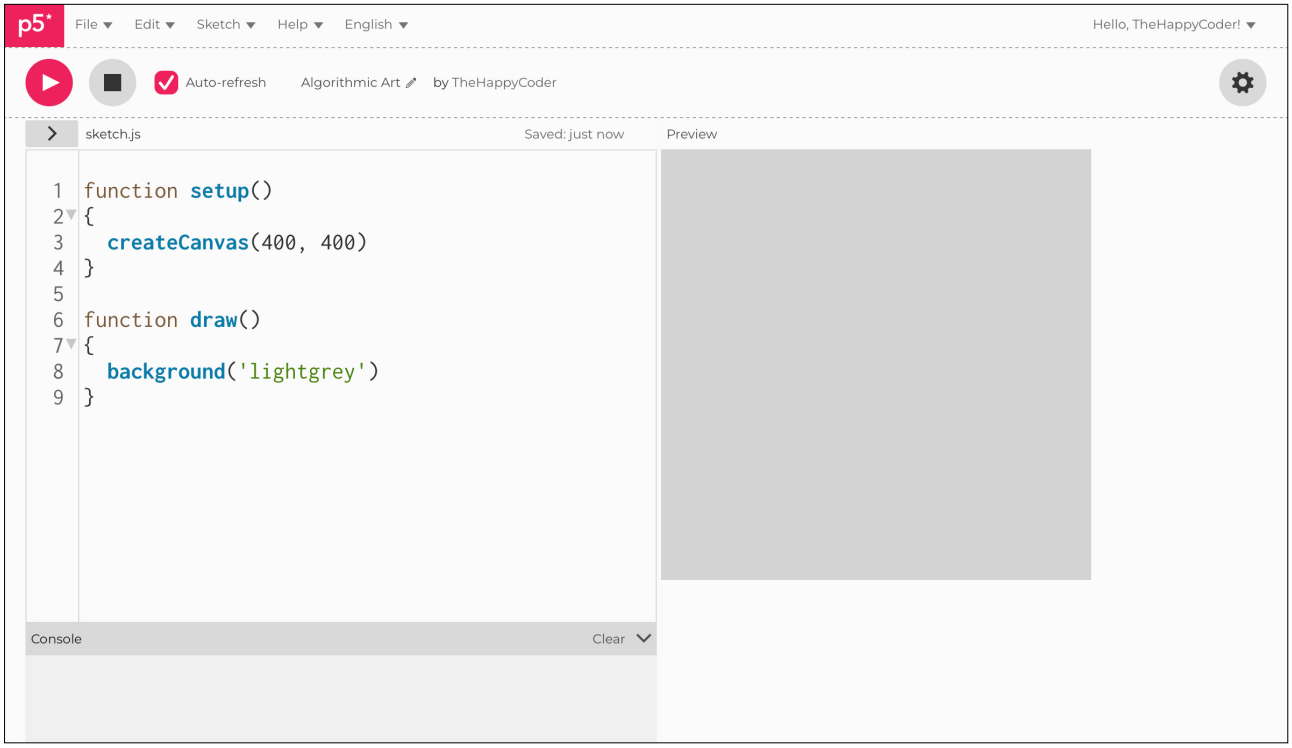
Challenge

Change the size of the canvas from
`createCanvas(400, 400)` to `createCanvas(600, 200)`

Code Explanation

`function setup()`	Everything in the first function only happens once.
`{ }`	All the code goes inside the curly brackets (or braces).
`createCanvas(400, 400)`	We are creating a canvas just like an artist ready to paint. This canvas is 400 pixels wide by 400 pixels high.
`function draw()`	This function operates a continuous loop.
`background('lightgrey')`	We give the background a colour. In this instance, it is a light grey colour (more on colours later).

Figure A2.1


p5*
File
Edit
Sketch
Help
English
Hello, TheHappyCoder!
Auto-refresh
Algorithmic Art
by TheHappyCoder
sketch.js
Saved: just now
Preview
1 function setup()
2 {
3 createCanvas(400, 400)
4 }
5
6 function draw()
7 {
8 background('lightgrey')
9 }
Console
Clear

Sketch A2.2 a circle

When we add or change a line (or lines) of code, they will be highlighted blue. We are going to add your first shape, a circle.

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

function draw()
{
  background('lightgrey')
  circle(200, 200, 100)
}
```

Notes

Because the background and circle are in the `draw()` function, the programme code is drawing them continuously. First, the background, and then the circle.

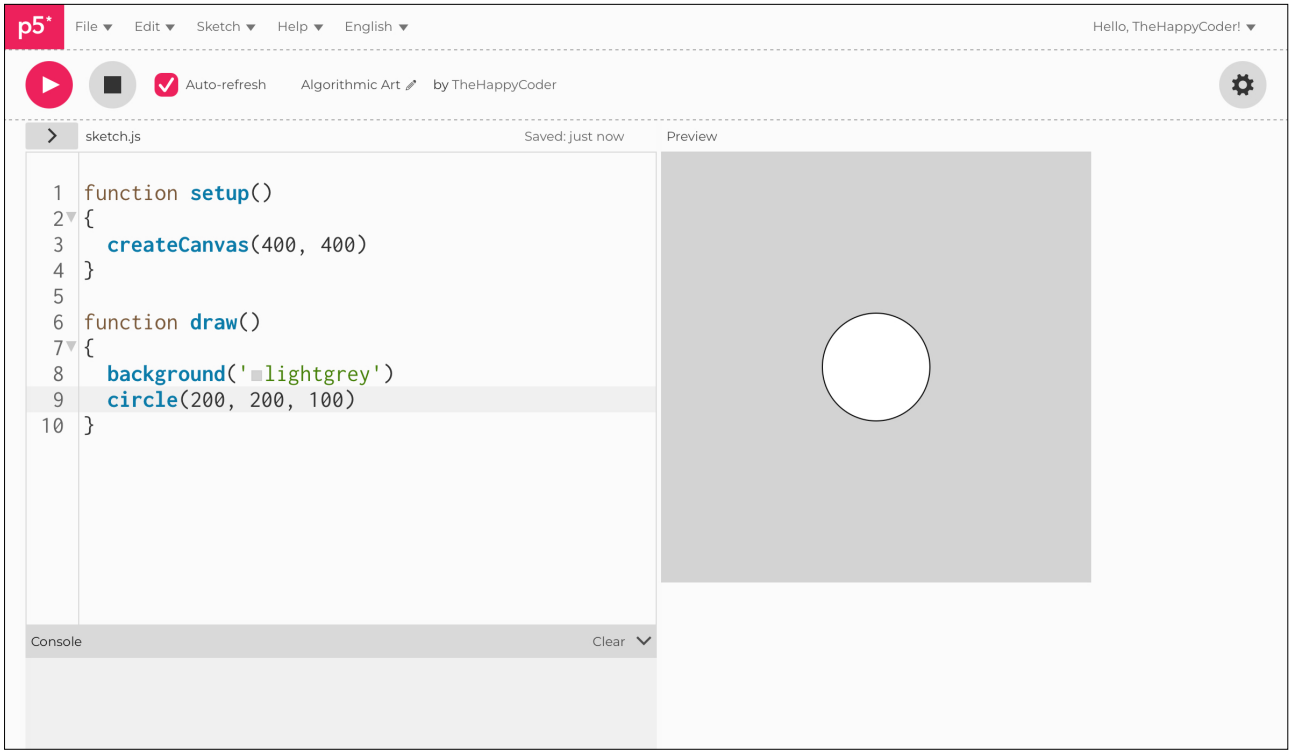
Challenges

1. Change the co-ordinates from: `circle(200, 200, 100)` to `circle(100, 300, 100)`
2. Change the circle diameter from: `circle(100, 300, 100)` to `circle(100, 300, 50)`

Code Explanation

`circle(200, 200, 100)`	The centre of the circle is at position 200 pixels from the left-hand edge of the canvas and 200 pixels from the top of the canvas. The diameter is 100 pixels.

Figure A2.2


p5*
File
Edit
Sketch
Help
English
Hello, TheHappyCoder!
Auto-refresh
Algorithmic Art
by TheHappyCoder
sketch.js
Saved: just now
Preview
1 function setup()
2 {
3 createCanvas(400, 400)
4 }
5
6 function draw()
7 {
8 background('lightgrey')
9 circle(200, 200, 100)
10 }
Console
Clear

Sketch A2.3 adding another circle

We can add more circles (and other shapes)

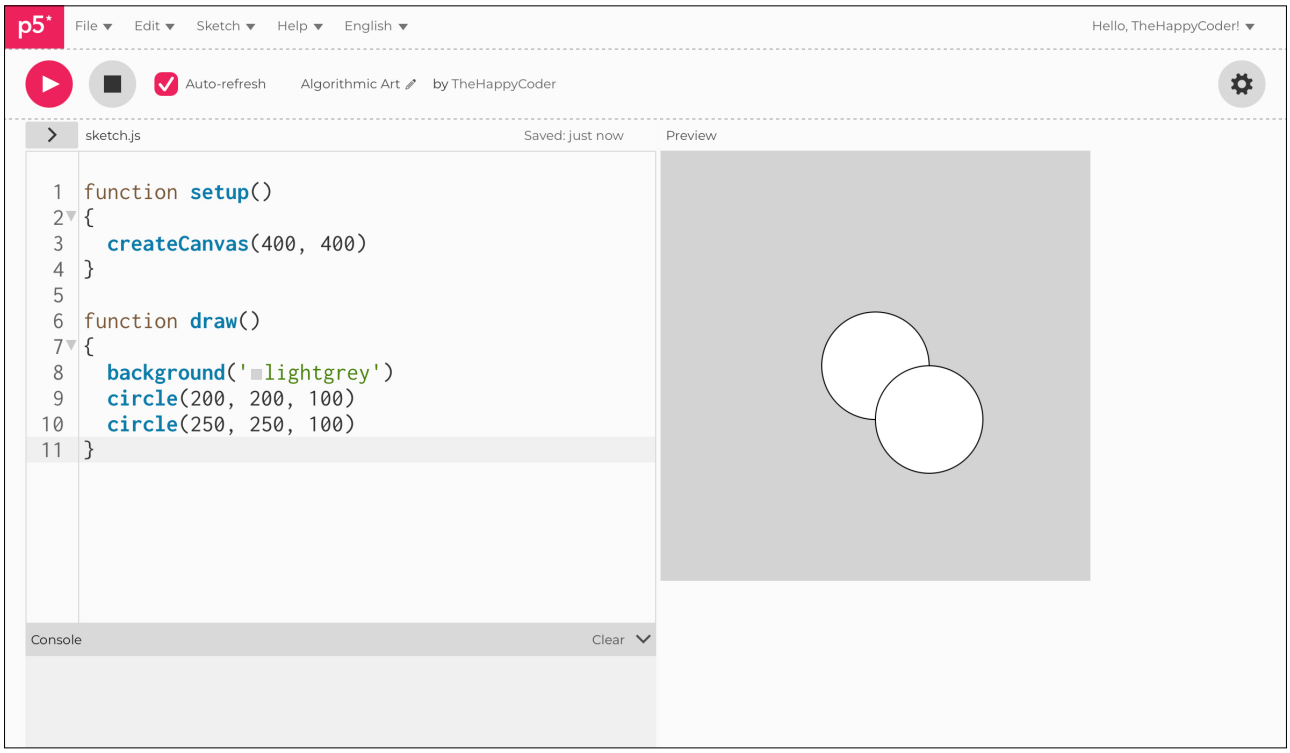
```
function setup()
{
  createCanvas(400, 400)
}

function draw()
{
  background('lightgrey')
  circle(200, 200, 100)
  circle(250, 250, 100)
}
```

Notes

Notice that the second circle overlaps on top of the first circle. The programme works from top to bottom, one line of code at a time. In this instance, it draws the background first, then the first circle, then the second circle, and then repeats.

Figure A2.3



Sketch A2.4 making a simple pattern

We will change the position of the second circle and add three more circles.

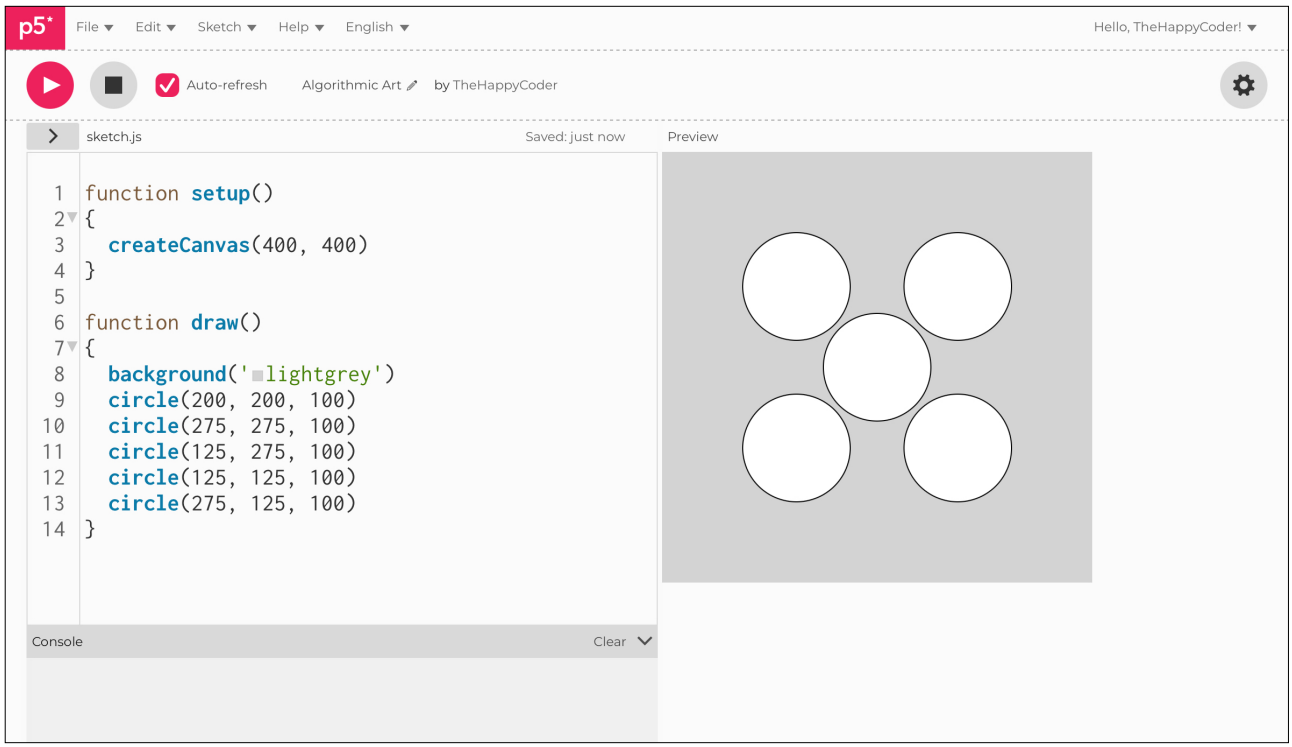
```
function setup()
{
  createCanvas(400, 400)
}

function draw()
{
  background('lightgrey')
  circle(200, 200, 100)
  circle(275, 275, 100)
  circle(125, 275, 100)
  circle(125, 125, 100)
  circle(275, 125, 100)
}
```

Notes

You might notice in the image below that next to `Auto-refresh`, the box is ticked and is highlighted in red. This is useful (as well as dangerous) so that you don't have to keep pressing the run (play) button. It can be dangerous if you are using `for()` loops (more on that later) where you can accidentally get into an infinite loop and crash the programme.

Figure A2.4


p5*
File
Edit
Sketch
Help
English
Hello, TheHappyCoder!
Auto-refresh
Algorithmic Art
by TheHappyCoder
sketch.js
Saved: just now
Preview
1 function setup()
2 {
3 createCanvas(400, 400)
4 }
5
6 function draw()
7 {
8 background('lightgrey')
9 circle(200, 200, 100)
10 circle(275, 275, 100)
11 circle(125, 275, 100)
12 circle(125, 125, 100)
13 circle(275, 125, 100)
14 }
Console
Clear



Sketch A2.5 adding some colour

Before we move onto other shapes and key concepts, we can explore colour. To start with, we can use the names of colours such as `red`, `green`, `blue`, or `orange`, for example (there are more but a limited number). To do this, we use the `fill()` function. First off, let's colour them `red`.

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

function draw()
{
  background('lightgrey')
  fill('red')
  circle(200, 200, 100)
  circle(275, 275, 100)
  circle(125, 275, 100)
  circle(125, 125, 100)
  circle(275, 125, 100)
}
```

Notes

You will notice that in the code, you get a little red box. This just illustrates the colour that you are requesting. There are 140 named colours. You can see them all at: https://www.w3schools.com/colors/colors_names.asp or just search for JavaScript colour names

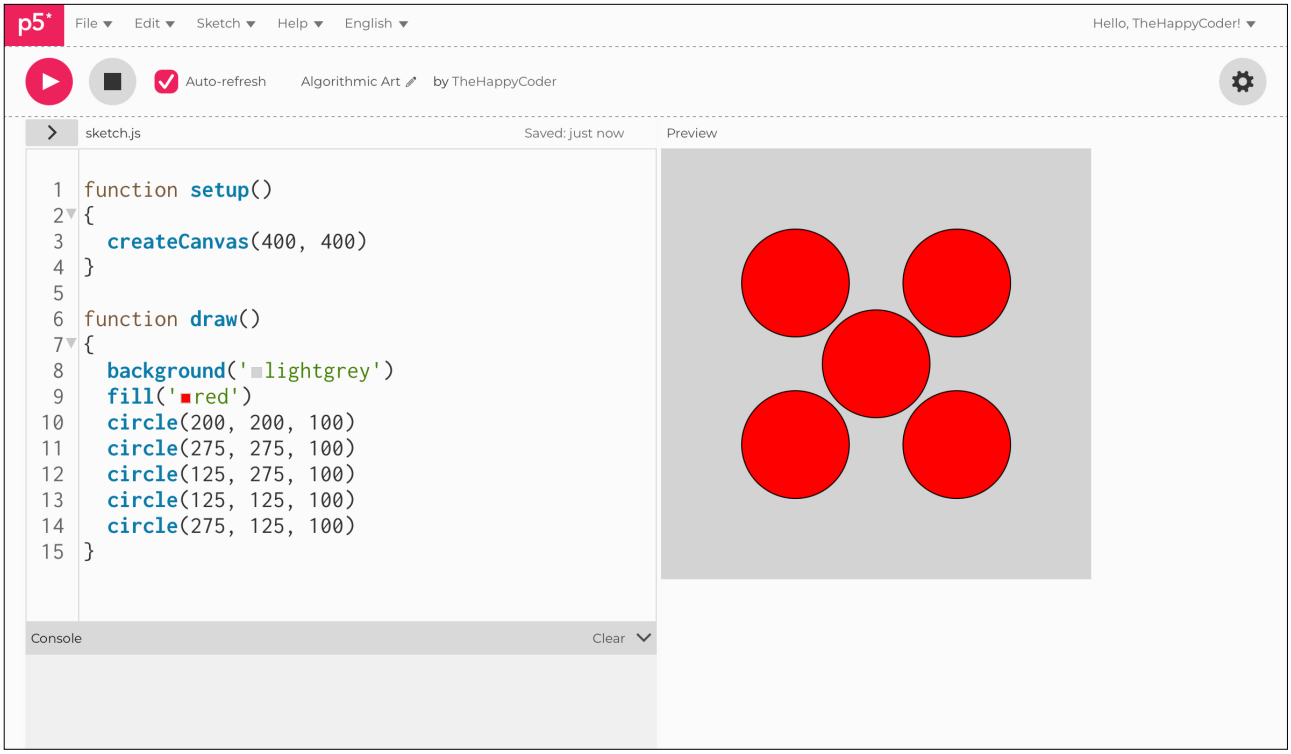
Challenge

Try other names of colours

Code Explanation

`fill('red')`	This is the function to fill any shape any colour you chose. If you use the name of a colour then you have to put it in speech marks (single or double)

Figure A2.5


p5*
File
Edit
Sketch
Help
English
Hello, TheHappyCoder!
Auto-refresh
Algorithmic Art
by TheHappyCoder
sketch.js
Saved: just now
Preview
1 function setup()
2 {
3 createCanvas(400, 400)
4 }
5
6 function draw()
7 {
8 background('lightgrey')
9 fill('red')
10 circle(200, 200, 100)
11 circle(275, 275, 100)
12 circle(125, 275, 100)
13 circle(125, 125, 100)
14 circle(275, 125, 100)
15 }
Console
Clear



Sketch A2.6 different colours

We will give each one a different colour.

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

function draw()
{
  background('lightgrey')
  fill('red')
  circle(200, 200, 100)
  fill('green')
  circle(275, 275, 100)
  fill('blue')
  circle(125, 275, 100)
  fill('yellow')
  circle(125, 125, 100)
  fill('purple')
  circle(275, 125, 100)
}
```

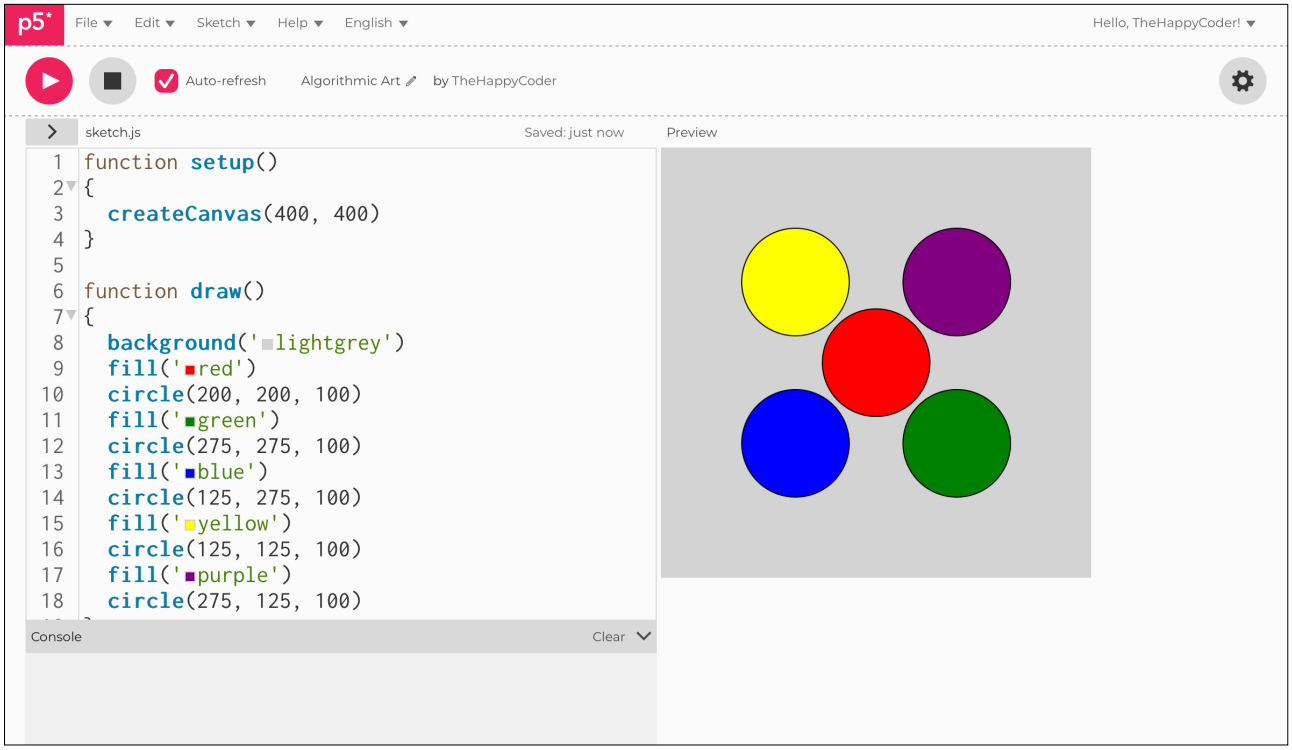
Notes

To get each separate colour to fill the next circle, we have to put the `fill()` function in between the circles. The code runs in a linear fashion; it goes step by step, one line at a time, starting at the top and then works its way down till it reaches the bottom. Then, with the `draw()` function, it goes to the top and starts all over again.

Challenge

Try other colours and see which colours work and which don't

Figure A2.6


p5*
File
Edit
Sketch
Help
English
Hello, TheHappyCoder!
Auto-refresh
Algorithmic Art
by TheHappyCoder
sketch.js
Saved: just now
Preview
function setup()
{
createCanvas(400, 400)
}
function draw()
{
background('lightgrey')
fill('red')
circle(200, 200, 100)
fill('green')
circle(275, 275, 100)
fill('blue')
circle(125, 275, 100)
fill('yellow')
circle(125, 125, 100)
fill('purple')
circle(275, 125, 100)
Console
Clear



Sketch A2.7 new sketch

! Start a brand new sketch.

Writing the code for five circles is not too arduous, but what if you want to draw a hundred or a thousand circles? We are going to need a bit of help. Programming is all about efficiency, and this usually means writing code in the fewest lines as possible. Delete the previous code and write this.

! Notice that the `background()` goes into the `setup()` function.

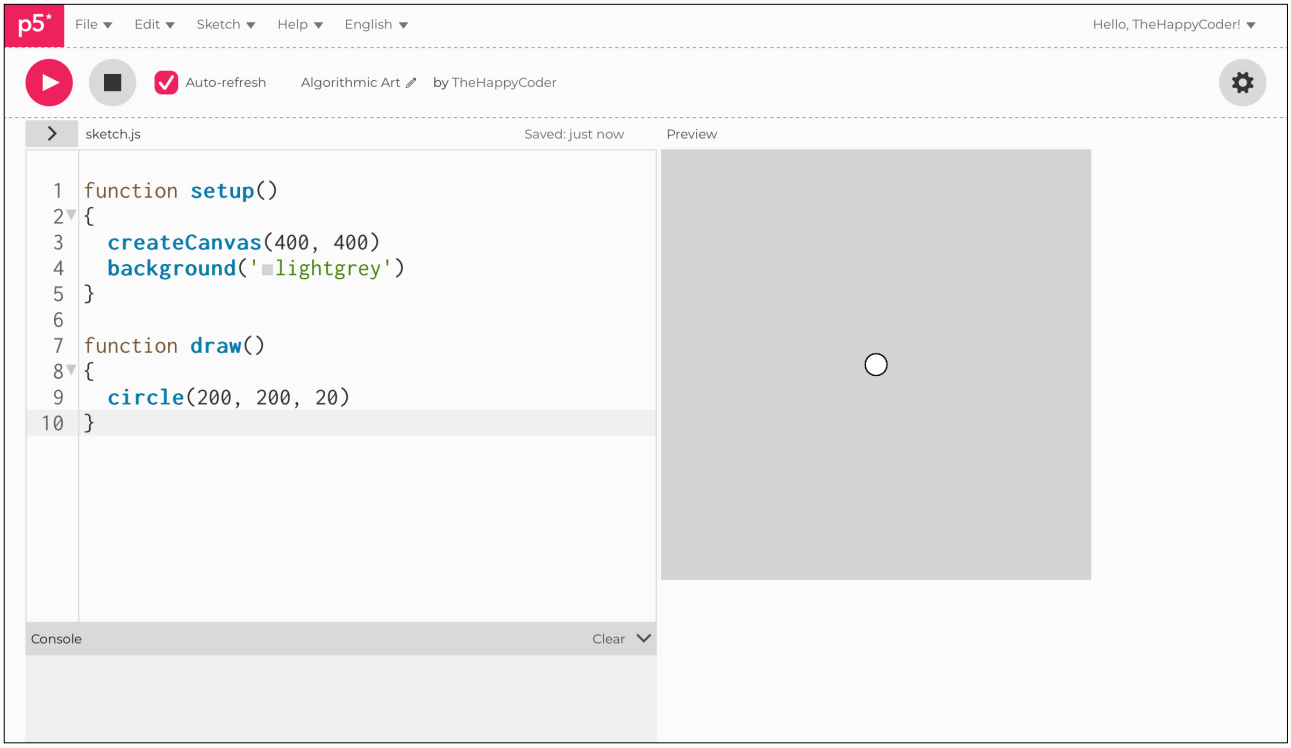
```
function setup()
{
  createCanvas(400, 400)
  background('lightgrey')
}

function draw()
{
  circle(200, 200, 20)
}
```

Notes

We have drawn a small (20-pixel diameter) circle in the centre of the canvas, but before we move onto the next sketch, I need to introduce an important concept: `variables`.

Figure A2.7


p5*
File
Edit
Sketch
Help
English
Hello, TheHappyCoder!
Auto-refresh
Algorithmic Art
by TheHappyCoder
sketch.js
Saved: just now
Preview
1 function setup()
2 {
3 createCanvas(400, 400)
4 background('lightgrey')
5 }
6
7 function draw()
8 {
9 circle(200, 200, 20)
10 }
Console
Clear



Introduction to variables

Variables are very useful for storing data that we may want to access or change later. Variables can be named with a single letter or a name. They can have numbers in them but must never start with a number. Usually, the name of the variables has some relevant meaning.

For instance, if you want to have a variable for speed, you would be best to use the word `speed` rather than just `s` because later on in a long list of code, you may forget what `s` represents. This is especially the case if you have a lot of variables. At the same time, don't make them too long; otherwise, the code will look very messy and difficult to read by anyone else but you.

In this next example, we are going to give the coordinates for the circle names `x` and `y`. So that we can alter them in a later sketch. Variables are very powerful and extremely useful.

To use a variable like `x` and `y`, then we need to define them. We use the key word `let`. You can just define it or initialise it, for example:

`let x`	This defines x as a variable.
`let x = 10`	This gives x an initial value of 10.

Now in the sketch below we are going to create a variable for the `x-coordinate` (which is `200`) and the `y` co-ordinate (which is `300`) of the circle by calling them `let x` and `let y`.

One other important detail, `scope`. It is always a good idea to declare variables at the beginning of a sketch, that way they are available everywhere. If you only declare them between two curly brackets they only live between those curly brackets and nowhere else. You will see that I do both so it is usually OK but something to bear in mind when generating many functions and lines of code.

Mostly we will be using numeric values (integers and floats) but just occasionally we will be using strings (words or letters).



Sketch A2.8 adding some variables

We have replaced the x value and y value with variables of that name. This draws the same small circle in the centre of the canvas.

```
let x = 200
let y = 200

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

function draw()
{
  circle(x, y, 20)
}
```

Notes

This demonstrates how using variables can be very useful, especially when there are many shapes and lots of movement. If you are familiar with data types `let` is a float (also called real) by default. The line of code `circle(x, y, 20)` is just the same as `circle(200, 200, 20)`.

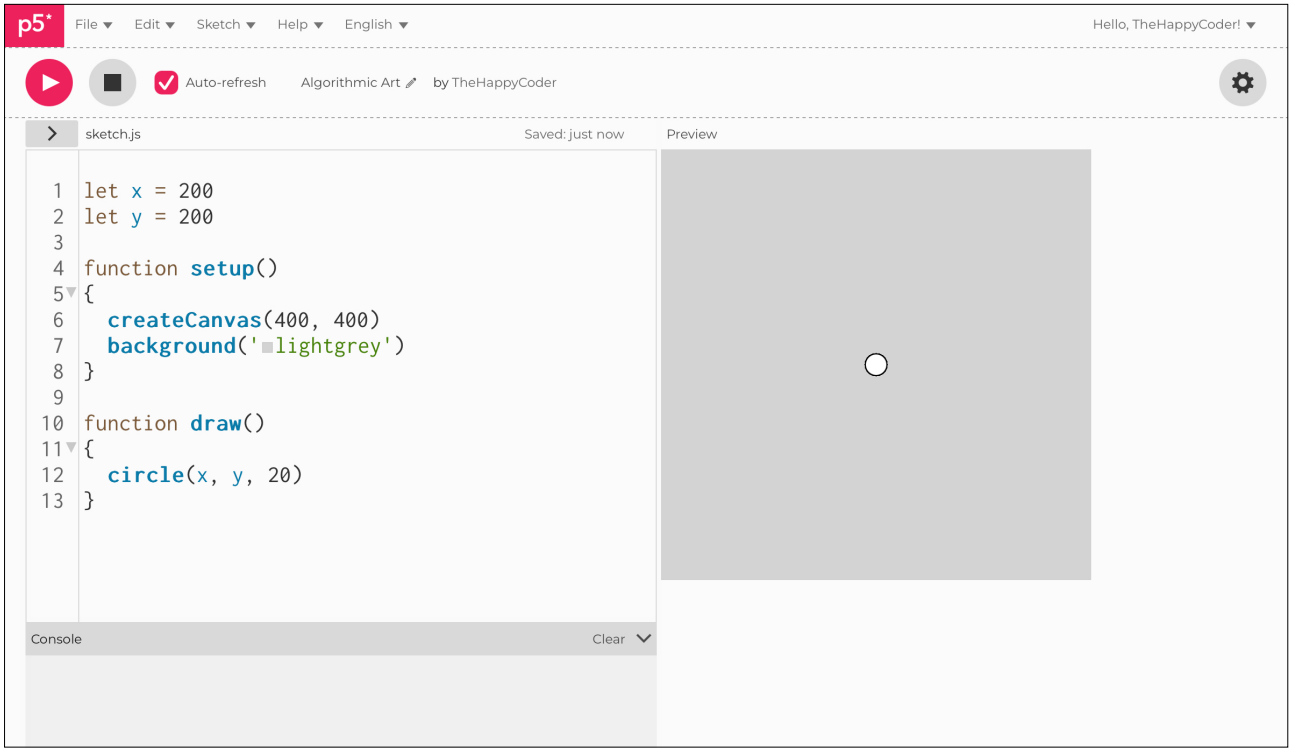
Challenges

1. Use different initialised values for the x and y variable
2. Create another variable for the diameter

Code Explanation

`let x = 200`	Declares the variable x and initialises it to a value of 200.
`let y = 200`	Declares the variable y and initialises it to a value of 200.
`circle(x, y, 20)`	The circle is drawn at the initialised x and y values (200, 200).

Figure A2.8


p5*
File
Edit
Sketch
Help
English
Hello, TheHappyCoder!
Auto-refresh
Algorithmic Art
by TheHappyCoder
sketch.js
Saved: just now
Preview
let x = 200
let y = 200
function setup()
{
createCanvas(400, 400)
background('lightgrey')
}
function draw()
{
circle(x, y, 20)
}
Console
Clear



Introduction to Random

Now we will be making shapes move and introduce random. Different functions as loops will be explored. This is where we can create a random number. It is not really random but for most purposes it is random enough. We use the function `random()` and it works like this...

`random(20)`	It will give you a number between 0 and 19 (but not 20).
`random(12, 37)`	This will give you a value between 12 and 37 (but not 37).

Before we go any further we need to look at notations used with coding which are also used in maths. Just read quickly through the next section to get a feel for the notations and what they mean. As you use them they will make more sense, I include them here and now so that you have a reference.

Symbols we use – Operators

Please just skim read through these; they are for reference and you don't need to learn them just yet. I include them now without any context so that you are aware of them. You may find them a useful reference later on.

In coding (as well as in maths), we use notation, symbols rather than words. Most of the maths used in coding is fairly basic; some symbols, however, may be unfamiliar. We are going to quickly cover the following:

Arithmetic Operators

Simple mathematical operators you will use frequently. They are used throughout all coding and should be quite familiar except for, perhaps, `modulus`.

+	Addition (2 + 4) will give you 6.
-	Subtraction (6 - 3) will give you 3.
/	Division (8 / 2) will give you 4.
*	Multiplication (3 * 5) will give you 15.
%	Modulus gives you the remainder, (10 % 8) will give you 2.
=	Equals (not equals to) 2 + 4 = 6

Comparison Operators

In short, these are conditional statements where a condition needs to be met. These are often used with `while()` loops

==	means equal to (5 == 5) is TRUE, (6 == 5) is FALSE.
<	means less than (6 < 8) is TRUE, (8 < 6) is FALSE.
<=	means less than or equal to (6 <= 8) is TRUE, (6 <= 6) is also TRUE.
>	means greater than (8 > 6) is TRUE, (6 > 8) is FALSE.
>=	means greater than or equal to (8 >= 6) is TRUE, (6 >= 6) is also TRUE.
!=	means not equal to (6 != 5) is TRUE (5 != 5) is FALSE.

Logical Operators

These are used with `if()` statements. The `if()` statement can be similar to the `while()` loop; if something is true or a condition is met, then do something, for example: `if(x < 100 && y > 50)` means if x is `less than` 100 `AND` y is `greater than` 50.

&&	means AND (x < 100 && y > 50).
||	means OR (x < 100 || y y > 50).
!	Means NOT (x < 100 ! y y > 50).

Assignment Operators

They are used often in coding (some much more than others). It is more obvious when you see them in a meaningful context, which often goes for all coding.

++	means increasing by 1 (x++) x is incremented by 1 each time.
−−	mean reducing or subtracting by 1 (y−) y is decremented by 1 each time.
+=	means addition (x += 10) or (x += y), same as x = x + y.
−=	Means subtracting (x −= 10) or (x −= y), same as x = x − y.
*=	means multiplying (x *= 10) or (x *= y) same as x = x * 10.
/=	means division (x /= 2) or (x /= y) same as x = x / 10.



Maths Functions

When using the mathematical notation, these can be very useful. Here are just a few

`floor()`	Calculates the closest integer value that is less than or equal to the value of a number.
`abs()`	Calculates the absolute value of a number. A number's absolute value is its distance from zero on the number line. -5 and 5 are both five units away from zero, so calling **abs(-5)** and **abs(5)** both return 5. The absolute value of a number is always positive.
`round()`	Calculates the integer closest to a number. For example, **round(133.8)** returns the value 134. The second parameter, **decimals**, is optional. It sets the number of decimal places to use when rounding. For example, **round(12.34, 1)** returns 12.3. **It is** zero by default.
`sq()`	Calculates the square of a number. Squaring a number means multiplying the number by itself. For example, **sq(3)** evaluates 3 x 3, which is 9. The **sq(-3)** evaluates -3 x -3, which is also 9. Multiplying two negative numbers produces a positive number. The value returned by **sq()** is always positive.
`squrt()`	Calculates the square root of a number. A number's square root can be multiplied by itself to produce the original number. For example, **sqrt(9)** returns 3 because 3 x 3 = 9. The **sqrt() function** always returns a positive value. **sqrt()** does not work with negative arguments such as **sqrt(-9)**.
`pow`	Calculates exponential expressions such as 2^3. For example, **pow(2, 3)** evaluates the expression 2 x 2 x 2. **pow(2, -3)** evaluates 1 ÷ (2 x 2 x 2).
`ceil()`	Calculates the closest integer value that is greater than or equal to a number. For example, calling **ceil(9.03)** and **ceil(9.97)** both return the value 10.
`exp()`	Calculates the value of Euler's number e (2.71828...) raised to the power of a number.



Sketch A2.9 drawing random circles

In this sketch, you will draw circles in random positions on the canvas. The function `draw()` is a continuous loop and will draw them forever. You don't need to give `x` and `y` an initial value, but it is good practice to do so.

```
let x = 200
let y = 200

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

function draw()
{
  x = random(400)
  y = random(400)
  circle(x, y, 20)
}
```

Notes

A reminder that in computer code it starts counting from 0, not 1, and so it stops when it gets to the 400th number, which is 399. Just something to remember.

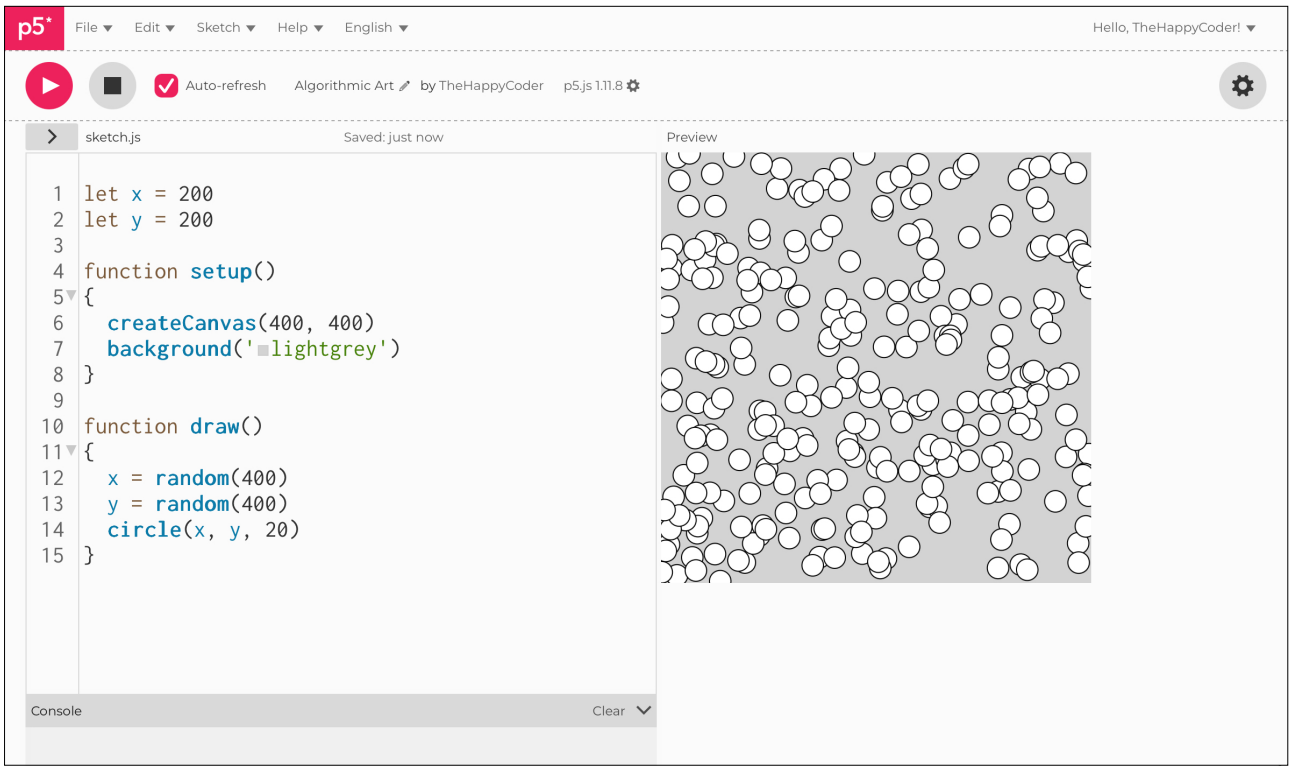
Challenge

Create a random diameter as well.

Code Explanation

`x = random(400)`	It chooses a random number from 0 to 399 for the x value.
`y = random(400)`	It chooses a random number from 0 to 399 for the x value.

Figure A2.9


p5*
File
Edit
Sketch
Help
English
Hello, TheHappyCoder!
Auto-refresh
Algorithmic Art
by TheHappyCoder
p5.js 1.11.8
sketch.js
Saved: just now
Preview
let x = 200
let y = 200
function setup()
{
createCanvas(400, 400)
background('lightgrey')
}
function draw()
{
x = random(400)
y = random(400)
circle(x, y, 20)
}
Console
Clear

Sketch A2.10 while() loop circles

! delete what is in the draw() function, notice we have a loop within draw and so things are double indented.

What we want to do is draw 100 circles and then stop. The while() loop keeps checking the number of circles it has drawn. If the number is less than (<) 100, it keeps going while it is true. Once it has drawn 100 and the next one is therefore more than 100, it then stops because it returns false. To keep track of the number of circles, we add a new variable called count.

! Make sure that Auto-refresh is OFF, otherwise the programme will likely crash as you type it in. It is the tick box near the top.

```
let x = 200
let y = 200
let count = 0

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

function draw()
{
  while (count < 100)
  {
    x = random(400)
    y = random(400)
    circle(x, y, 20)
    count = count + 1
  }
}
```

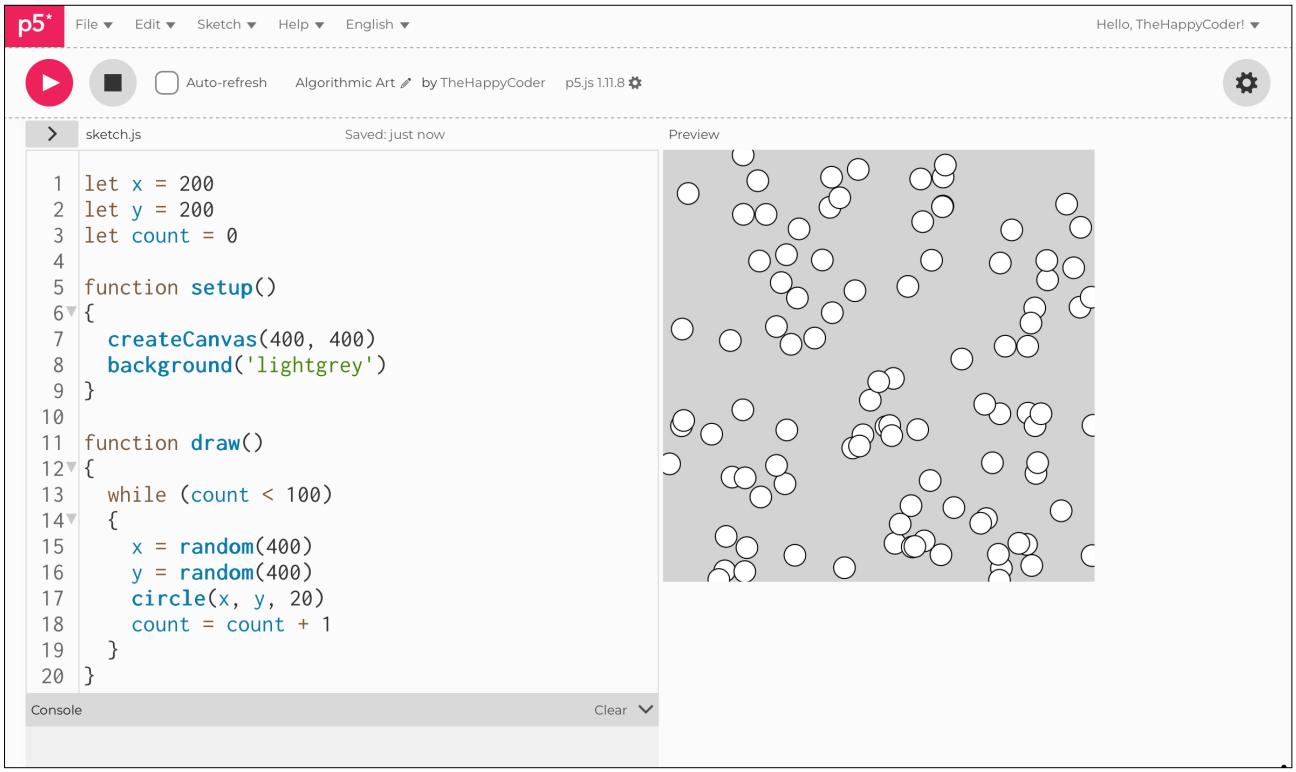
Notes

This sketch draws exactly 100 small circles in random positions every time you press the run button. It makes good use of the while() loop function. Also uses the comparison < less than.

Code Explanation

`while (count < 100)`	This is a while() loop which checks to see if count has reached 100.
`count = count + 1`	We add 1 each time to the count variable.

Figure A2.10





Sketch A2.11 a few more adjustments

We can use the `random()` function to create a border and introduce a very common shorthand for adding each time `1` (`++`). So, `count = count + 1` becomes: `count++`.

```
let x = 200
let y = 200
let count = 0

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

function draw()
{
  while (count < 100)
  {
    x = random(100, 300)
    y = random(100, 300)
    circle(x, y, 20)
    count++
  }
}
```

Notes

We have now limited the random range to between `100` and `300`; this gives us a nice empty border around the canvas.

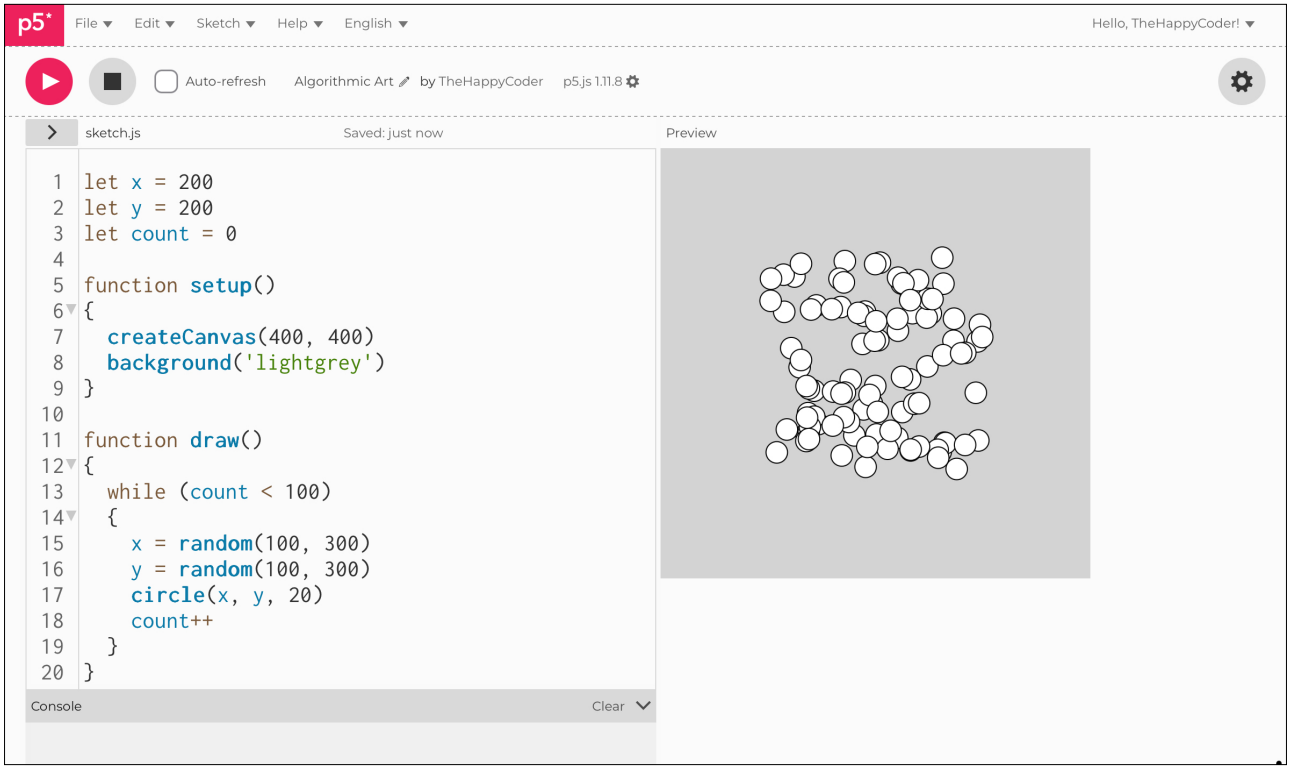
Challenge

Try different values for random.

Code Explanation

`x = random(100, 300)`	Returns a random number between 100 and 300.
`y = random(100, 300)`	Returns a random number between 100 and 300.
`count++`	Shorthand version: adds 1 each time, the same as count = count + 1.

Figure A2.11


p5*
File
Edit
Sketch
Help
English
Hello, TheHappyCoder!
Auto-refresh
Algorithmic Art
by TheHappyCoder
p5.js 1.11.8
sketch.js
Saved: just now
Preview
let x = 200
let y = 200
let count = 0
function setup()
{
createCanvas(400, 400)
background('lightgrey')
}
function draw()
{
while (count < 100)
{
x = random(100, 300)
y = random(100, 300)
circle(x, y, 20)
count++
}
}
Console
Clear



Introducing the for() loop

You have already come across the `while()` loop. There is another called the `if()` statement (coming soon) which is similar to the `while()` loop. There is a third and it is arguably the most important or the most used. It is a little bit more complicated, but as we will be using it a lot, you will get used to it and it does become very intuitive (trust me).

Our example is as follows:

```
for (let i = 0; i < 100; i++)
```

Inside the `for()` loop, brackets are three parts separated by semicolons (`;`), they are:

1. `let i = 0`
2. `i < 100`
3. `i++`

Firstly, we create a variable called `i`. This is just convention; you can call it anything you like, and we give it an initial value of `0`.
Secondly, we put a condition on this variable, in this case, less than `100` (`<`). The `for()` loop is true while `i` is less than `100`.
Thirdly, we increment the loop, in this instance, we simply add `1` to `i` for each loop (`i++`) until `i` has the value of `100` and stops.

Sketch A2.12 using for() loops

Another way to draw 100 circles is to use a for() loop. It loops through as a sort of counter from 0 to 100 in steps of 1. We are still going to use i as the counter, but we initialise it inside the for() loop itself. The noLoop() function stops the draw() function but only after it has drawn 100 circles.

! Remove: all reference to the count variable, I recommend starting a new sketch, it helps memory muscle. The // means it is removed (or commented out), more on that later.

```
let x = 200
let y = 200
// let count = 0

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

function draw()
{
  for (let i = 0; i < 100; i++)
  {
    x = random(100, 300)
    y = random(100, 300)
    circle(x, y, 20)
    // count++
  }
  noLoop()
}
```

Notes

Draws 100 circles in random positions once between `100` and `300` using the `for()` loop. For `(let i = 0; i < 100; i++)` draws `100` circles in random positions once between `100` and `300` using the `for()` loop. There are three parts to it:

Part 1: `let i = 0;`
Part 2: `i < 100;`
Part 3: `i++`

Persevere with the `for()` loop as it will appear a lot in the future. Notice we have the semicolons separating the three parts.

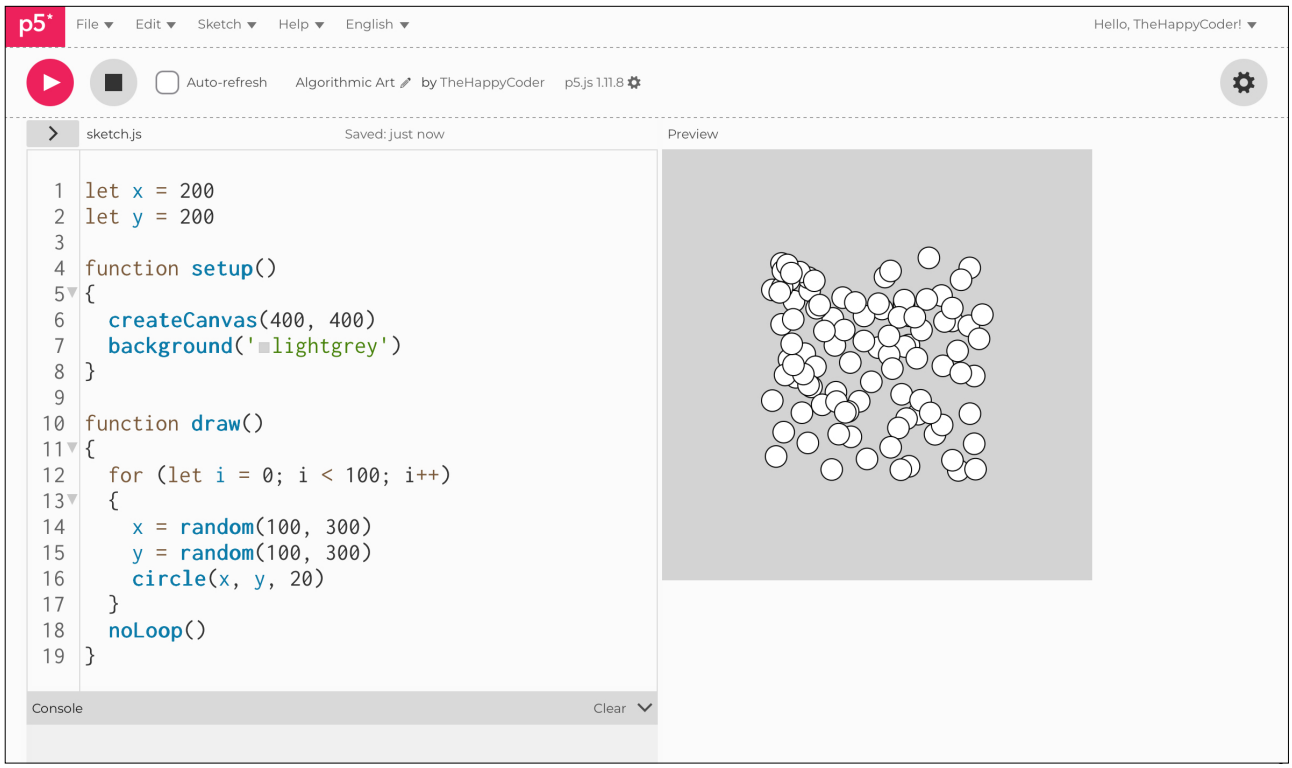
Challenges

1. Change the number from `100` to either `10` or `1000`.
2. Take out the `noLoop()` function.

Code Explanation

`for()`	This is the for() loop; it will count to 10 and stop (actually will then start counting again). The draw function is a loop, so it is a loop within a loop.
`let i = 0`	This declares a variable called i and initialises it to 0.
`i < 100`	This bit checks to see if i is still less than 100.
`i++`	This adds 1 to the i variable every time it does a loop.
`noLoop()`	Stops the draw() loop once the condition has been met.

Figure A2.12


p5*
File
Edit
Sketch
Help
English
Hello, TheHappyCoder!
Auto-refresh
Algorithmic Art
by TheHappyCoder
p5.js 1.11.8
sketch.js
Saved: just now
Preview
let x = 200
let y = 200
function setup()
{
createCanvas(400, 400)
background('lightgrey')
}
function draw()
{
for (let i = 0; i < 100; i++)
{
x = random(100, 300)
y = random(100, 300)
circle(x, y, 20)
}
noLoop()
}
Console
Clear


Algorithmic
Art
Module A
Unit #3
Adding RGB



Module A Unit #3: adding RGB

This unit looks at RGB colour, by that we mean red, green, and blue. This is how you can create a wide range of colours by mixing these primary colours; in fact, you can create over 16 million colours.

Think in terms of light rather than paints. If you have all the red, all the green, and all the blue, you get white; if you have none of them, you get black, and every combination in between. The amount of a single colour is between `0` - `256`.

Key concepts:

`RGB colour`
`transparency`
`noStroke()`
`stroke()`
`strokeWeight()`



RGB colour

We can represent the colour not just in words— red, green, or blue —but also using their RGB values. Each colour has some red, some green, and some blue. One reason for using this system of creating the colours is that we can manipulate the values mathematically. For instance, the following colours are given as RGB values. See Fig. 1 below.

`fill(255, 0, 0)`	This gives us red with no green and no blue.
`fill(0, 255, 0)`	This gives us green with no red or blue.
`fill(0, 0, 255)`	This gives us blue with no red or green.
`fill(255, 255, 255)`	This gives us white, the same as `fill(255)`.
`fill(0, 0, 0)`	This gives us black, same as `fill(0)`.

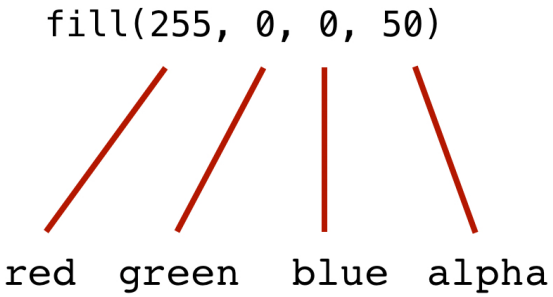


RGB transparency

You can create any colour by mixing the numbers as you do with mixing paint or light. The three numbers are called `arguments`; you can add a fourth argument, which is the `alpha` or transparency value. The range is between `0` and `255`. See `Fig. 1` below.

`fill(255, 0, 0, 255)`	This gives us no transparency at all; even though it should be red, it will appear opaque.
`fill(255, 0, 0, 0)`	This gives us no transparency at all; even though it should be red, it will appear completely transparent.
`fill(255, 0, 0, 50)`	Gives us some red, but you can still see through it.

Figure 1: fill() function



Sketch A3.1 RGB

! Start a new sketch

Drawing three overlapping circles, each with a separate colour. In the previous unit, we used names for the colours used. The background was called `'lightgrey'`. That is one way of using colours in your sketch; alternatively, we can use numerical values. Here we are using the RGB values for the background and the colours of the circles. I have highlighted the code in blue. Notice that I have given the `background()` a value of `220`, which is a single colour value, meaning it is either black (value of `0`) or white (value of `255`) and everything in between as shades of grey.

```
function setup()
{
  createCanvas(400, 400)
  background(220)
}

function draw()
{
  fill(255, 0, 0)
  circle(175, 150, 100)
  fill(0, 255, 0)
  circle(225, 150, 100)
  fill(0, 0, 255)
  circle(200, 200, 100)
}
```

Notes

Each circle will have its own colour fill.

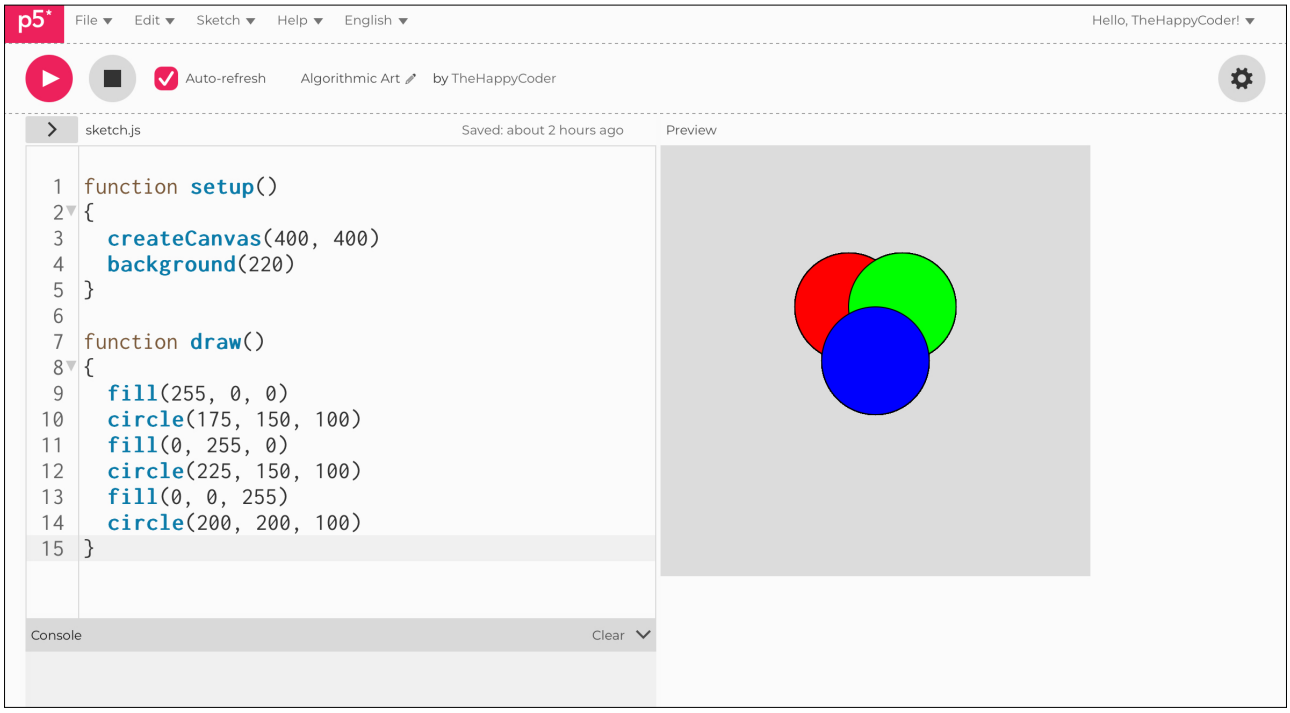
Challenge

Remove one of the `fill()` functions and see what happens.

Code Explanation

`background(220)`	The background has a grey colour.
`fill(255, 0, 0)`	This gives you a red-coloured circle, where the red value = 255, green value = 0, and blue has a value = 0.
`fill(0, 255, 0)`	This gives you a green-coloured circle, where the red value = 0, green value = 255, and blue has a value = 0.
`fill(0, 0, 255)`	This gives you a blue-coloured circle, where the red value = 0, green value = 0, and blue has a value = 255.

Figure A3.1



Sketch A3.2 alpha

Adding a bit of alpha by adding a fourth argument, we give the circles an `alpha` value of `50`.

```
function setup()
{
  createCanvas(400, 400)
  background(220)
}

function draw()
{
  fill(255, 0, 0, 50)
  circle(175, 150, 100)
  fill(0, 255, 0, 50)
  circle(225, 150, 100)
  fill(0, 0, 255, 50)
  circle(200, 200, 100)
}
```

Notes

The overlapping circles are now more translucent, with the help of some alpha.

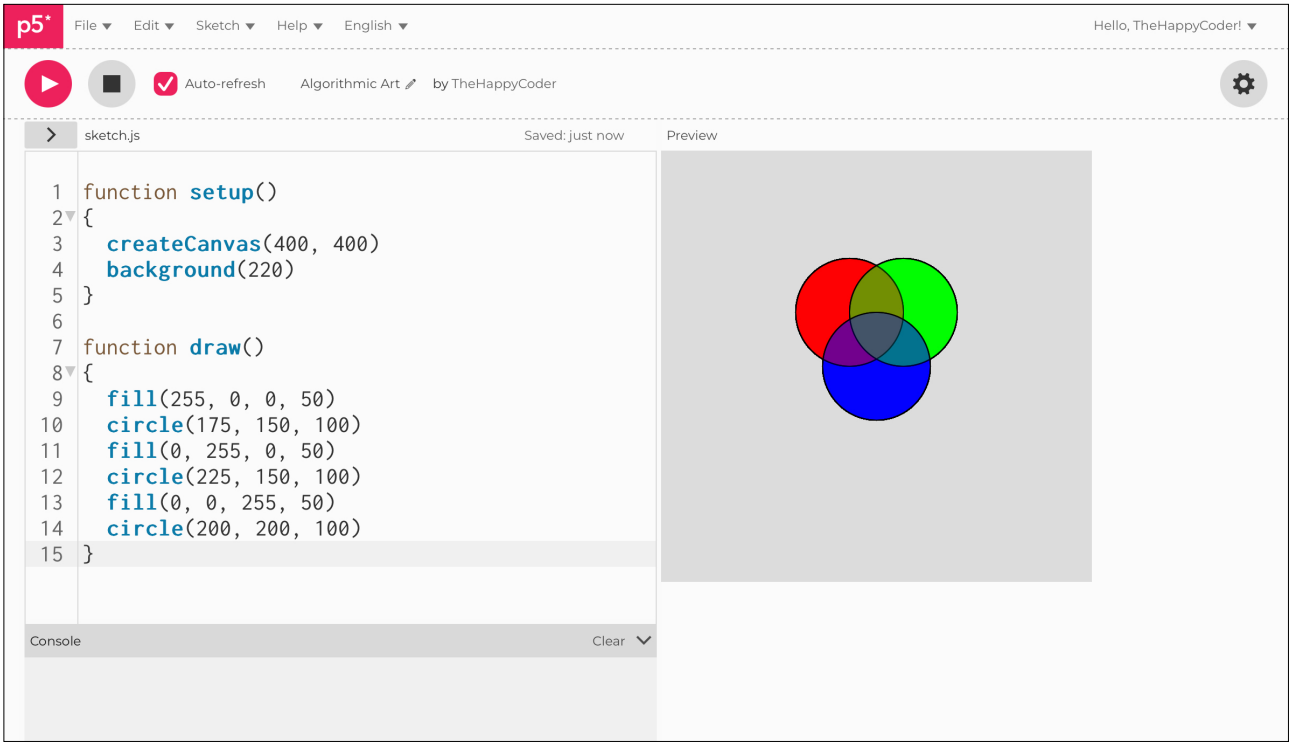
Challenge

Change the values of alpha.

Code Explanation

`fill(255, 0, 0, 50)`	Red with plenty of transparency.
`fill(0, 255, 0, 50)`	Green with plenty of transparency.
`fill(0, 0, 255, 50)`	Blue with plenty of transparency.

Figure A3.2


p5*
File
Edit
Sketch
Help
English
Hello, TheHappyCoder!
Auto-refresh
Algorithmic Art
by TheHappyCoder
sketch.js
Saved: just now
Preview
1 function setup()
2 {
3 createCanvas(400, 400)
4 background(220)
5 }
6
7 function draw()
8 {
9 fill(255, 0, 0, 50)
10 circle(175, 150, 100)
11 fill(0, 255, 0, 50)
12 circle(225, 150, 100)
13 fill(0, 0, 255, 50)
14 circle(200, 200, 100)
15 }
Console
Clear

Sketch A3.3 random colour alpha value

! Start a new sketch

We are now going to go mad with the variables, making variables of nearly everything. The alpha is the fourth argument for colour and it is the amount of transparency from 0 (transparent) to 255 (opaque).

```
let x = 0
let y = 0
let r = 0
let g = 0
let b = 0
let a = 0
let d = 0

function setup()
{
  createCanvas(400, 400)
  background(220)
}

function draw()
{
  x = random(400)
  y = random(400)
  r = random(255)
  g = random(255)
  b = random(255)
  a = random(255)
  d = random(100)
  fill(r, g, b, a)
  circle(x, y, d)
}
```

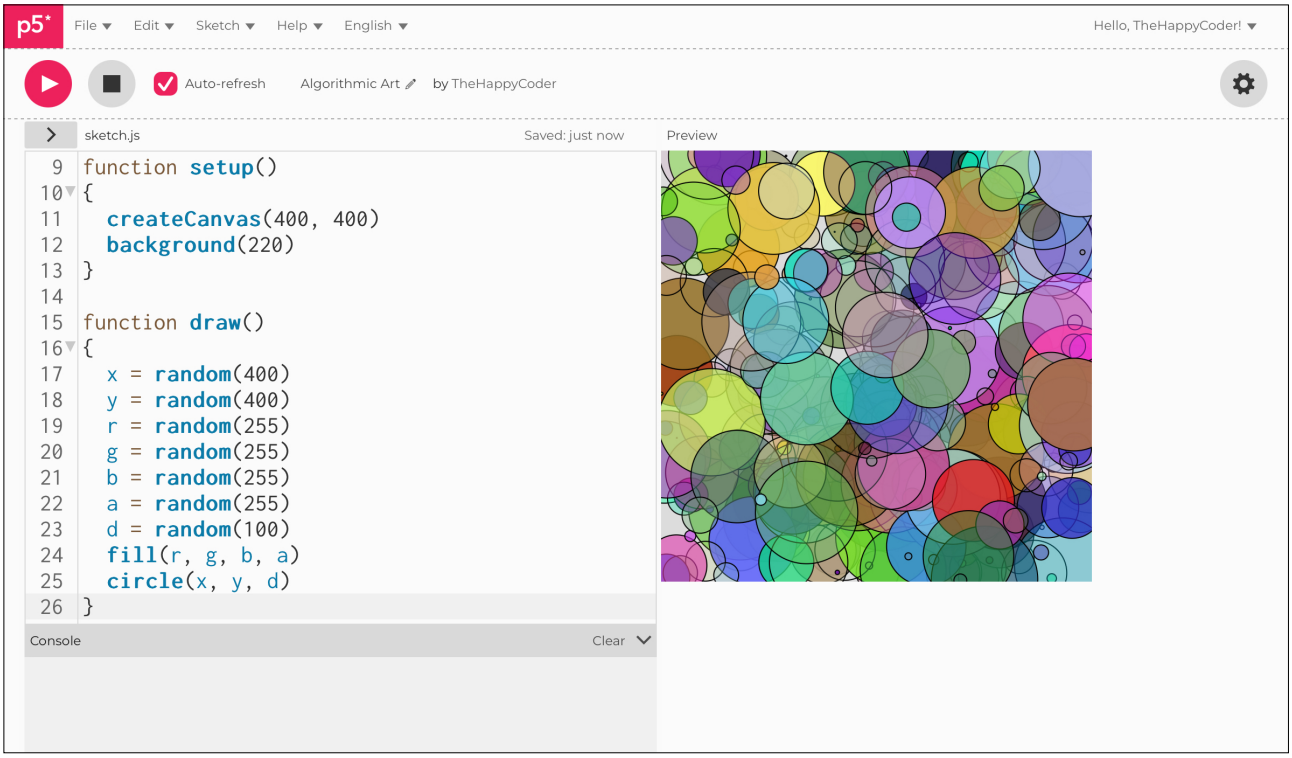
Notes

You wouldn't generally use single letters to name variables, but in this case, it is simple enough and intuitive.

Code Explanation

`x = random(400)`	Random x position of circle.
`y = random(400)`	Random y position of circle.
`r = random(255)`	Random red value.
`g = random(255)`	Random green value.
`b = random(255)`	Random blue value.
`a = random(255)`	Random alpha.
`d = random(100)`	Random diameter.
`fill(r, g, b, a)`	Fill each circle with one set of random values.
`circle(x, y, d)`	Draw a circle on each loop of the draw() function, at random positions and diameters.

Figure A3.3





Sketch A3.4 no stroke

We can remove the line around the circle with the function `noStroke()`.

```
let x = 0
let y = 0
let r = 0
let g = 0
let b = 0
let a = 0
let d = 0

function setup()
{
  createCanvas(400, 400)
  background(220)
}

function draw()
{
  noStroke()
  x = random(400)
  y = random(400)
  r = random(255)
  g = random(255)
  b = random(255)
  a = random(255)
  d = random(100)
  fill(r, g, b, a)
  circle(x, y, d)
}
```

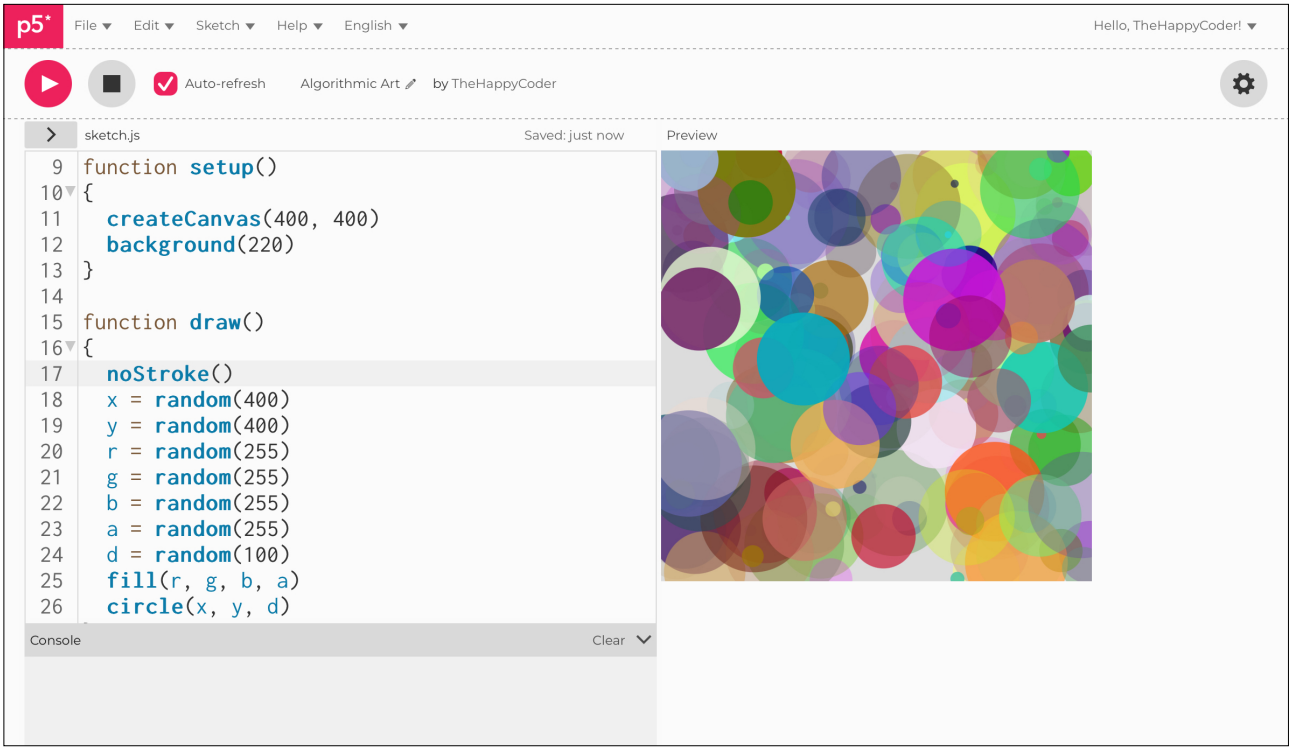
Notes

Having `noStroke()` and `noFill()` will effectively remove the circle from the canvas.

Code Explanation

`noStroke()`	Removes any lines around shapes.

Figure A3.4


p5*
File
Edit
Sketch
Help
English
Hello, TheHappyCoder!
Auto-refresh
Algorithmic Art
by TheHappyCoder
sketch.js
Saved: just now
Preview
9 function setup()
10 {
11 createCanvas(400, 400)
12 background(220)
13 }
14
15 function draw()
16 {
17 noStroke()
18 x = random(400)
19 y = random(400)
20 r = random(255)
21 g = random(255)
22 b = random(255)
23 a = random(255)
24 d = random(100)
25 fill(r, g, b, a)
26 circle(x, y, d)
Console
Clear



Sketch A3.5 the weight of the stroke

! Remove noStroke() and replace with strokeWeight()

We can alter the thickness of the lines with the function strokeWeight(), the default is a thickness of 1 pixel, so you could use any value you like, but I recommend only using a value up to 5.

```
let x = 0
let y = 0
let r = 0
let g = 0
let b = 0
let a = 0
let d = 0

function setup()
{
  createCanvas(400, 400)
  background(220)
}

function draw()
{
  strokeWeight(random(5))
  x = random(400)
  y = random(400)
  r = random(255)
  g = random(255)
  b = random(255)
  a = random(255)
  d = random(100)
  fill(r, g, b, a)
  circle(x, y, d)
}
```

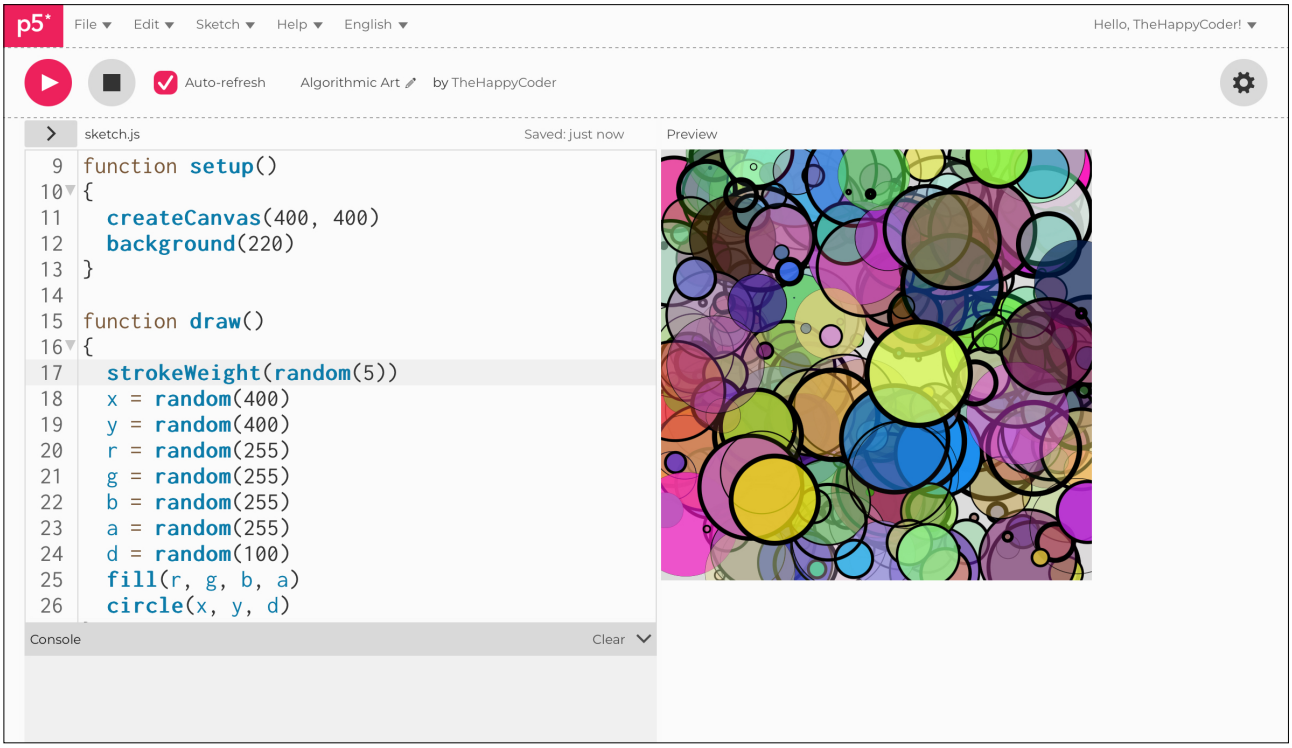
Challenge

Try different ranges of random strokeWeight().

Code Explanation

`strokeWeight(random(5))`	We can put the random function inside the brackets rather than create another variable.

Figure A3.5





Sketch A3.6 colouring the lines

! Replace `strokeWeight()` with a function called `stroke()`

We can also add colour (even add alpha) to the lines using `stroke()`, it works exactly the same as the `fill()` function.

```
let x = 0
let y = 0
let r = 0
let g = 0
let b = 0
let a = 0
let d = 0

function setup()
{
  createCanvas(400, 400)
  background(220)
}

function draw()
{
  stroke(r, g, b)
  x = random(400)
  y = random(400)
  r = random(255)
  g = random(255)
  b = random(255)
  a = random(255)
  d = random(100)
  fill(r, g, b, a)
  circle(x, y, d)
}
```

Notes

The colour of the lines is the colour of the last `fill(r, g, b)` circle drawn.

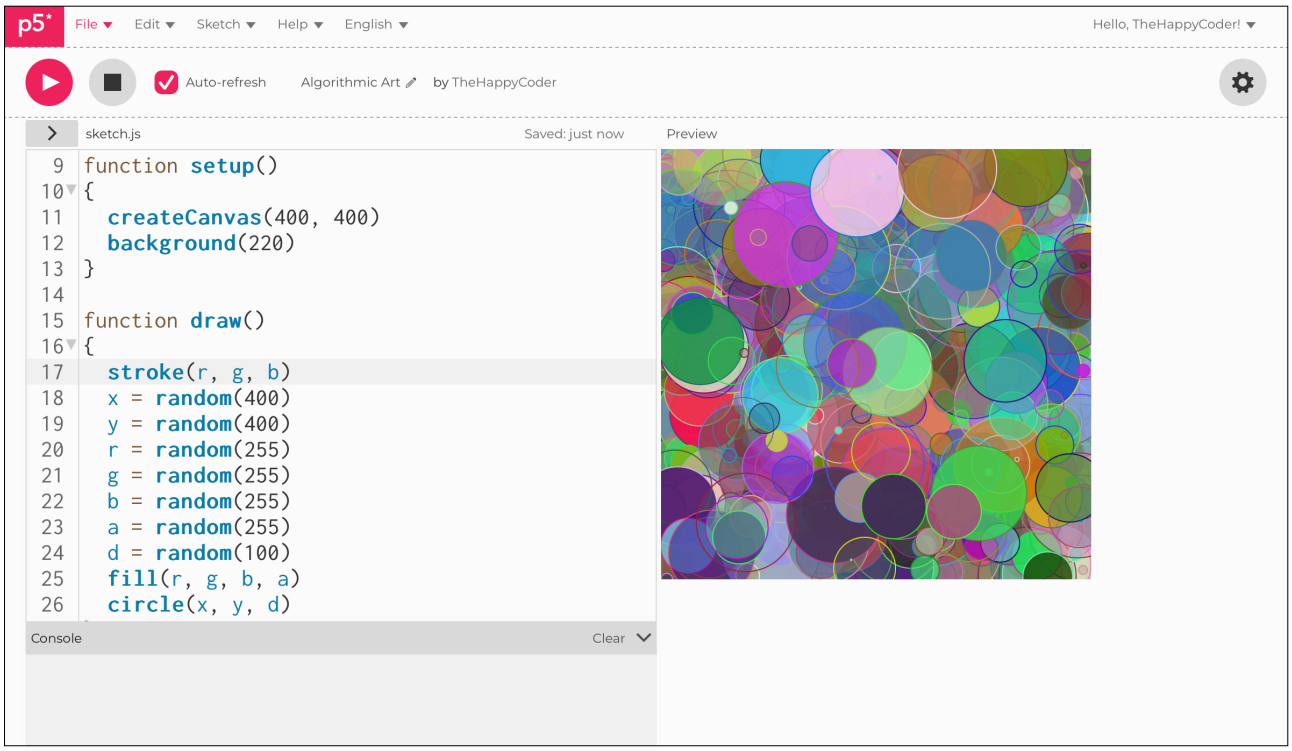
Challenges

1. Add random strokeWeight() as well.
2. Move `stroke(r, g, b)` to the line just before `fill(r, g, b, a)`.

Code Explanation

`stroke(r, g, b)`	As with the fill() function, we can give it up to four arguments: red, green, blue, and alpha.

Figure A3.6





Algorithmic Art Module A Unit #4 Lots of Lines

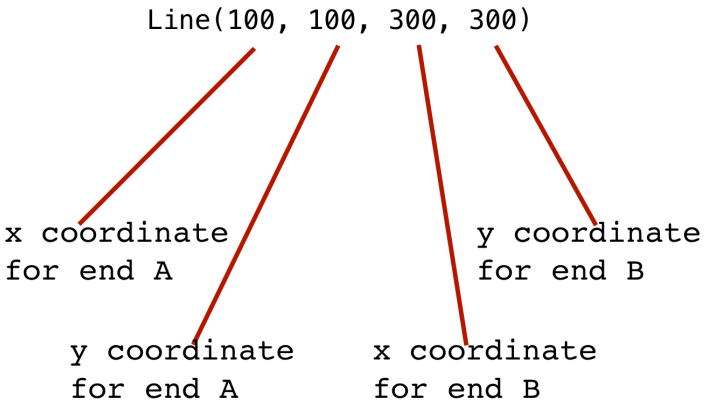
Module A Unit #4: lots of lines

We have drawn circles now to introduce lines. To draw a line, we need four arguments; these are two sets of coordinates. A set for each end of the line; it is that simple. If we have an end A and an end B for the line() function, it will look like below in Fig. 1. This draws a line between the coordinates (100, 100) and (300, 300).

Key concepts:

line() function
if() statement
noLoop()

Figure 1: line() function



Sketch A4.1 lines

! start a new sketch

We can draw lots of other shapes, but in this section, we will draw lines and create simple image using just lines. To reiterate, the line() function has four arguments: the first two are the x and y co-ordinates of one end of the line, and the other two arguments are the x and y co-ordinates of the other end of the line.

```
function setup()
{
  createCanvas(400, 400)
  background(220)
}

function draw()
{
  line(100, 100, 300, 300)
}
```

Notes

Not a very exciting creation but there is plenty we can do with it.

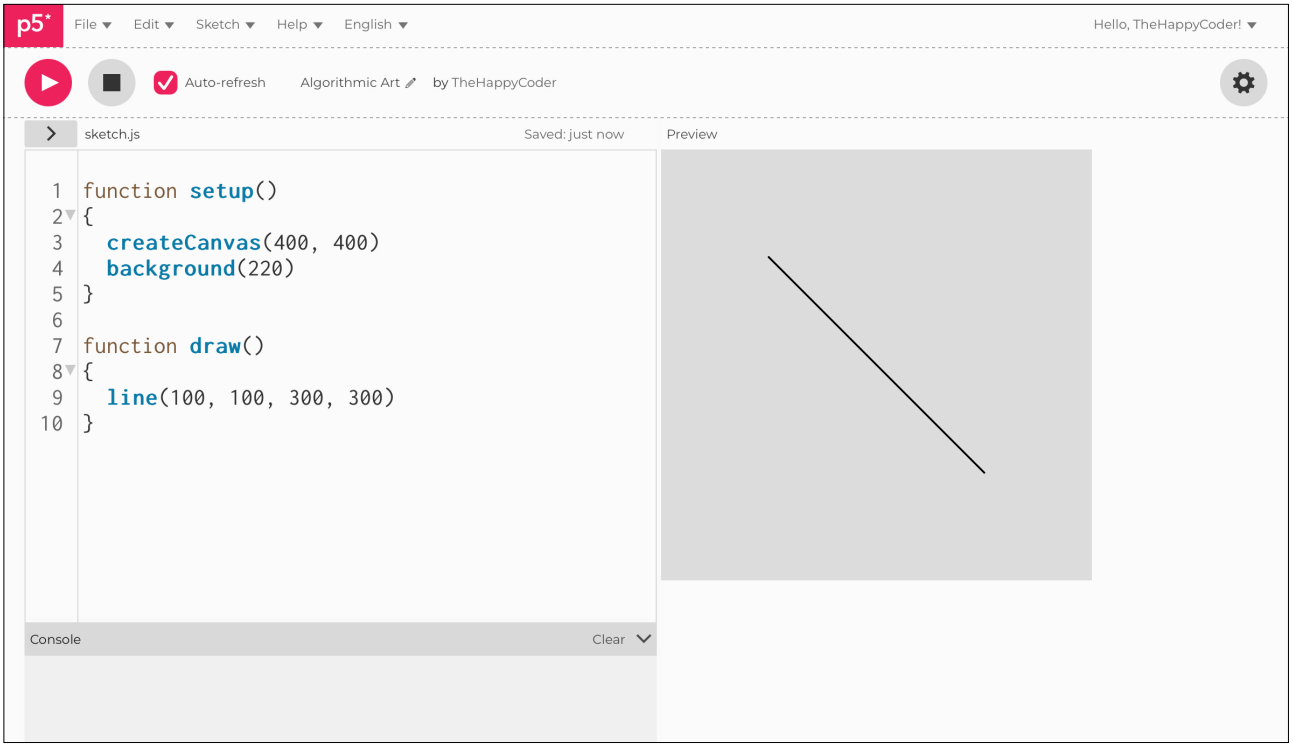
Challenge

Try other values for the coordinates

Code Explanation

`line(100, 100, 300, 300)`	Drawing a line from the top left (100, 100) to the bottom right (300, 300).

Figure A4.1


p5*
File
Edit
Sketch
Help
English
Hello, TheHappyCoder!
Auto-refresh
Algorithmic Art
by TheHappyCoder
sketch.js
Saved: just now
Preview
1 function setup()
2 {
3 createCanvas(400, 400)
4 background(220)
5 }
6
7 function draw()
8 {
9 line(100, 100, 300, 300)
10 }
Console
Clear

Sketch A4.2 a row of lines

Here, we are going to draw lots of vertical lines 10 pixels apart. We first create a variable for the x component of the coordinates and put that in the line() function. We want the same x value for each end.

```
let x = 10

function setup()
{
  createCanvas(400, 400)
  background(220)
}

function draw()
{
  line(x, 100, x, 300)
  x += 10
}
```

Notes

To add 10 each time we have used an abbreviation (+=). We could have written it long hand like so: x = x + 10. We have cheated a little because the programme is still drawing the lines; it never stops, which is never a great idea. In the next bit, we will use a while() loop to stop it.

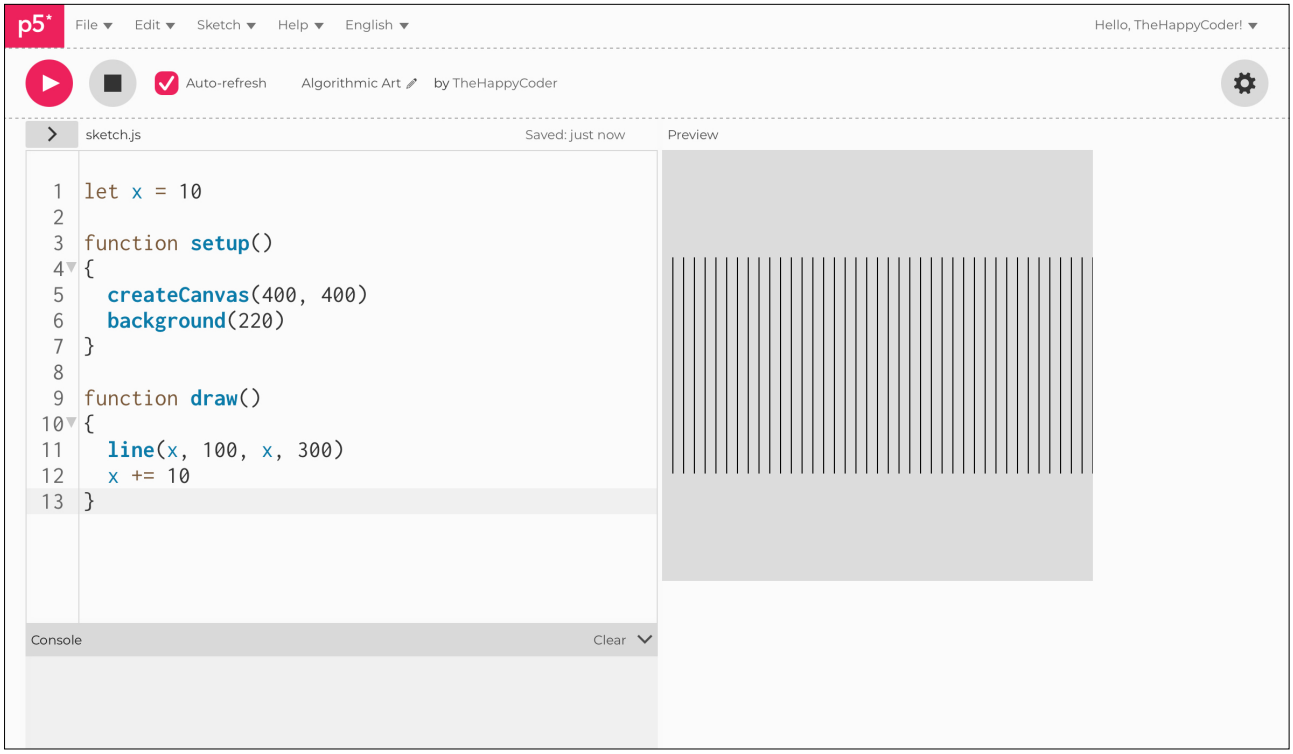
Challenge

Try a different spacing value other than 10

Code Explanation

let x = 10	Create a variable for x and give it a starting value of 10.
line(x, 100, x, 300)	Draws a line with the same x value which starts at 10 and increases by 10; the y values remain the same for each line.
x += 10	Adds 10 each time. We could write x = x + 10.

Figure A4.2


p5*
File
Edit
Sketch
Help
English
Hello, TheHappyCoder!
Auto-refresh
Algorithmic Art
by TheHappyCoder
sketch.js
Saved: just now
Preview
let x = 10
function setup()
{
createCanvas(400, 400)
background(220)
}
function draw()
{
line(x, 100, x, 300)
x += 10
}
Console
Clear

Sketch A4.3 limiting lines

Here we have a `while()` loop. In this example, the lines are drawn while `x` is less than `400`. When `x` reaches `400`, then the loop stops. The loop only works while `x` is less than (`<`) `400`. To put the code inside the curly brackets, you can just cut and paste.

```
let x = 10

function setup()
{
  createCanvas(400, 400)
  background(220)
}

function draw()
{
  while (x < 400)
  {
    line(x, 100, x, 300)
    x += 10
  }
}
```

Notes

The `while()` loop uses a conditional statement (`<`). While `x` is less than `400`, the condition is `true` and it draws the line, adds `10` to `x`, and repeats. The condition is considered to be `false` when the value of `x` is `400` or above (and the function stops).

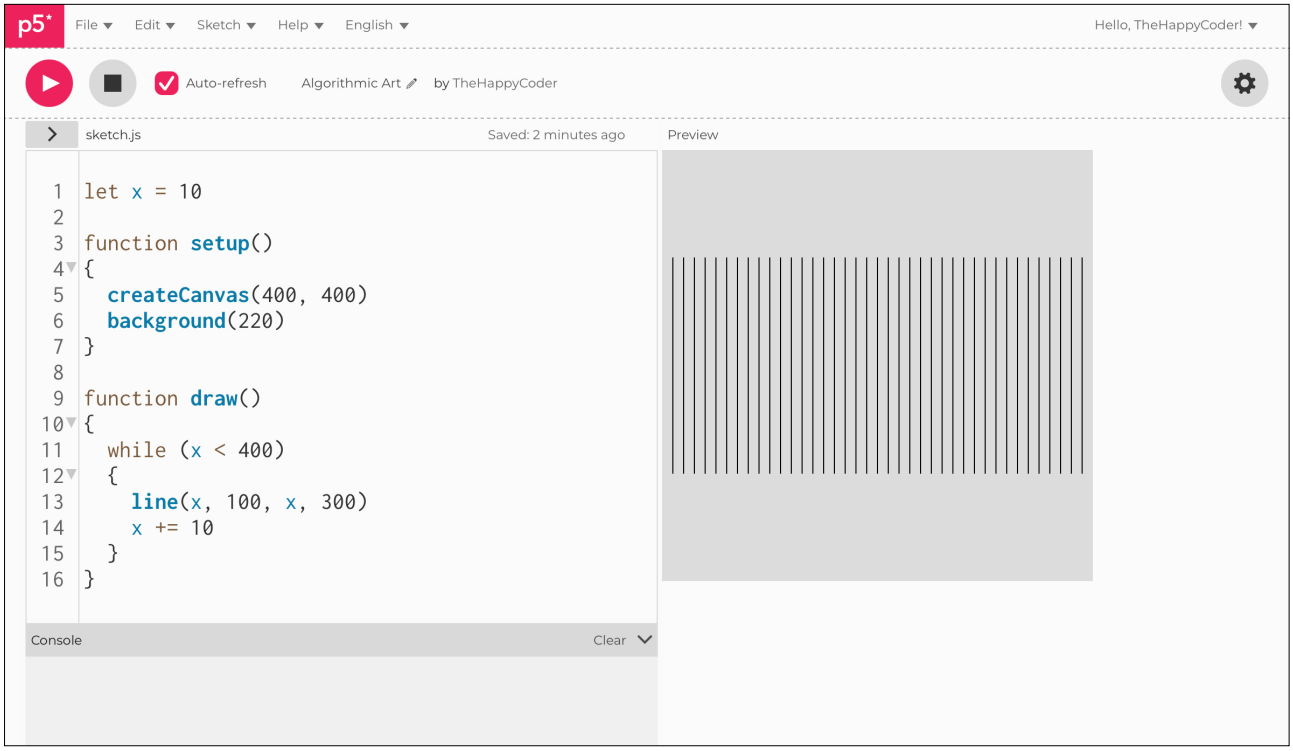
Challenges

Change the conditional value to `100`

Code Explanation

`while (x < 400)`	It checks to see if x has exceeded 400.

Figure A4.3


p5*
File
Edit
Sketch
Help
English
Hello, TheHappyCoder!
Auto-refresh
Algorithmic Art
by TheHappyCoder
sketch.js
Saved: 2 minutes ago
Preview
let x = 10
function setup()
{
createCanvas(400, 400)
background(220)
}
function draw()
{
while (x < 400)
{
line(x, 100, x, 300)
x += 10
}
}
Console
Clear

Sketch A4.4 colour and thickness

Adding some thickness and colour to the lines.

```
let x = 10

function setup()
{
  createCanvas(400, 400)
  background(220)
}

function draw()
{
  strokeWeight(5)
  stroke(200, 0, 0)
  while (x < 400)
  {
    line(x, 100, x, 300)
    x += 10
  }
}
```

Notes

We can give the line extra weight (thickness) as well as colour. This means we can manipulate the lines in many creative and interesting ways.

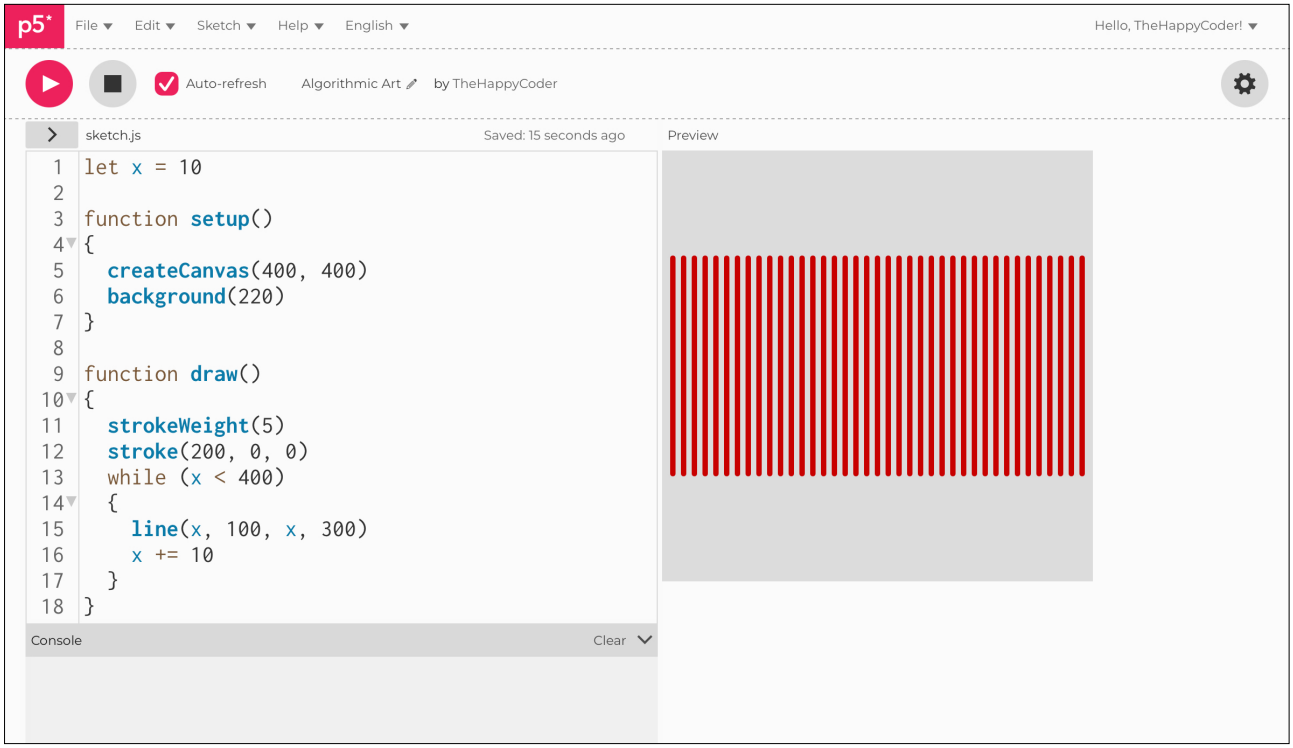
Challenges

1. Try alpha as well
2. Make it full height

Code Explanation

`strokeWeight(5)`	Determines how thick the line is.
`stroke(200, 0, 0)`	Determines the colour of the line.

Figure A4.4


p5*
File
Edit
Sketch
Help
English
Hello, TheHappyCoder!
Auto-refresh
Algorithmic Art
by TheHappyCoder
sketch.js
Saved: 15 seconds ago
Preview
let x = 10
function setup()
{
createCanvas(400, 400)
background(220)
}
function draw()
{
strokeWeight(5)
stroke(200, 0, 0)
while (x < 400)
{
line(x, 100, x, 300)
x += 10
}
}
Console
Clear

Sketch A4.5 incrementing the colour

We can introduce another variable to reduce the amount of red. We will call this `increment` and subtract it from the initial value of `200` in steps of `5`.

! we move the `stroke()` function inside the `while()` loop

```
let x = 10
let increment = 5

function setup()
{
  createCanvas(400, 400)
  background(220)
}

function draw()
{
  strokeWeight(5)
  // stroke(200, 0, 0)
  while (x < 400)
  {
    stroke(200 - increment, 0, 0)
    line(x, 100, x, 300)
    x += 10
    increment += 5
  }
}
```

Notes

We are decreasing the red element of the RGB by `5` on each iteration inside the `while()` loop, thus making it darker.

Challenge

1. Can you think of another way to do it (hint: use `-=` instead)
2. Try to achieve the opposite, from dark red to a light red

Code Explanation

`let increment = 5`	Create a variable called increment.
`stroke(200 - increment, 0, 0)`	Reduce the amount of red by that amount.

`increment += 5`	Keep reducing the red with each iteration of the loop.

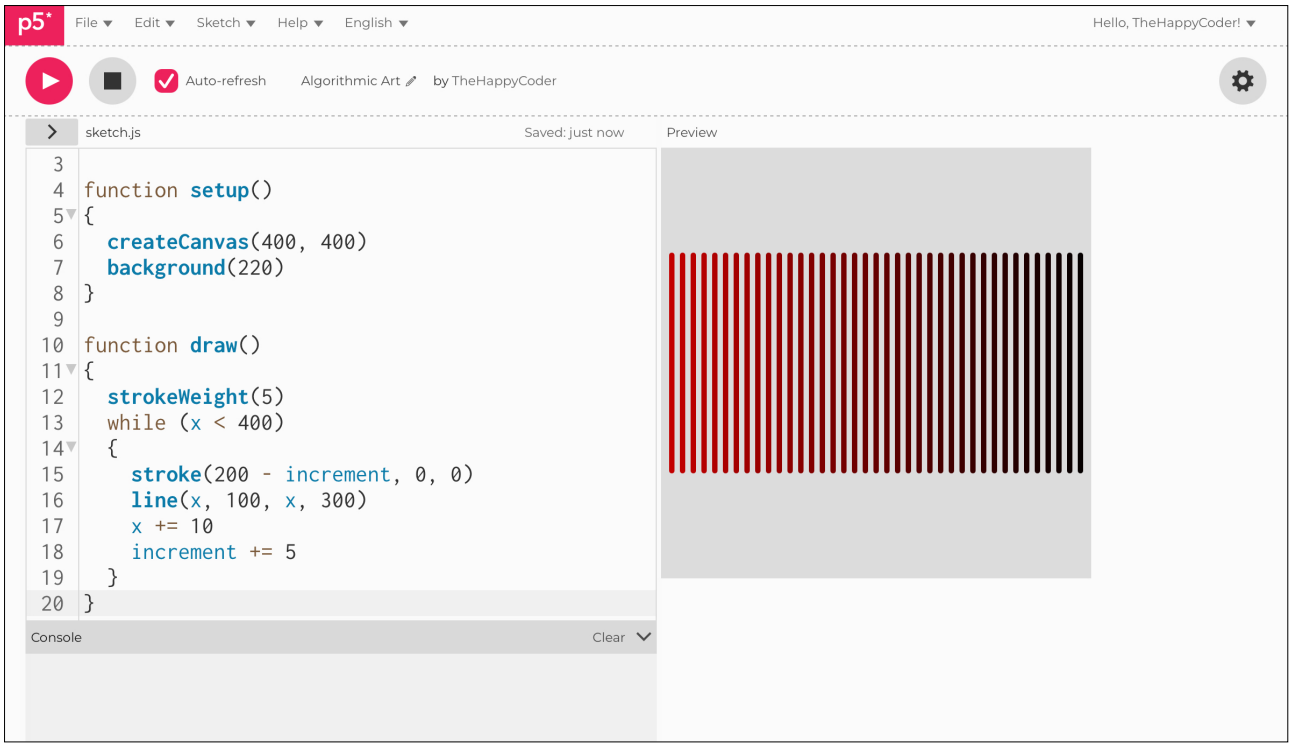
Hint solution:

```
let x = 10
let increment = 200

function setup()
{
  createCanvas(400, 400)
  background(220)
}

function draw()
{
  strokeWeight(5)
  while (x < 400)
  {
    stroke(increment, 0, 0)
    line(x, 100, x, 300)
    x += 10
    increment -= 5
  }
}
```

Figure A4.5



Sketch A4.6 more blue

Let's add the increment to the amount of blue as the red value decreases.

```
let x = 10
let increment = 5

function setup()
{
  createCanvas(400, 400)
  background(220)
}

function draw()
{
  strokeWeight(5)
  while (x < 400)
  {
    stroke(200 - increment, 0, increment)
    line(x, 100, x, 300)
    x += 10
    increment += 5
  }
}
```

Notes

The power of variables

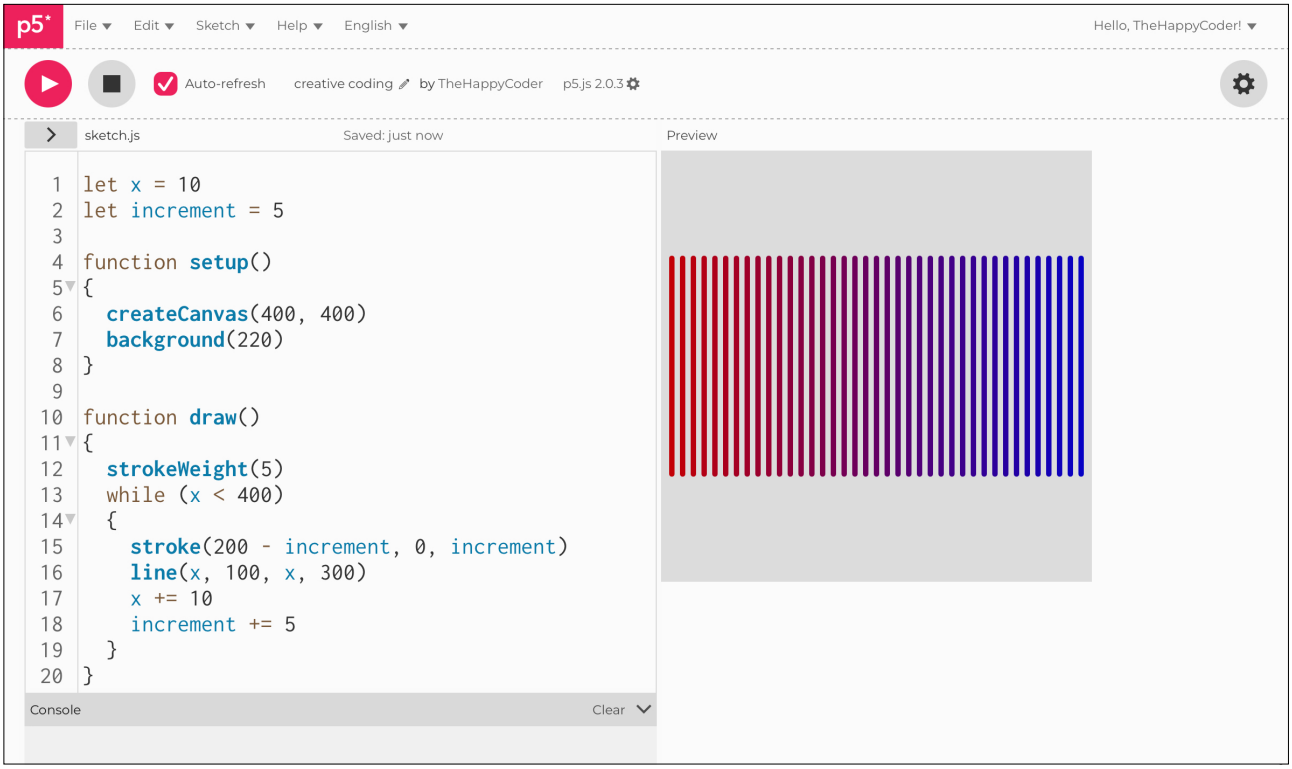
Challenges

1. Try other colours
2. Try alpha

Code Explanation

`stroke(200 - increment, 0, increment)`	Decreases the red and increases the blue at the same time.

Figure A4.6


p5*
File
Edit
Sketch
Help
English
Hello, TheHappyCoder!
Auto-refresh
creative coding
by TheHappyCoder
p5.js 2.0.3
sketch.js
Saved: just now
Preview
let x = 10
let increment = 5
function setup()
{
createCanvas(400, 400)
background(220)
}
function draw()
{
strokeWeight(5)
while (x < 400)
{
stroke(200 - increment, 0, increment)
line(x, 100, x, 300)
x += 10
increment += 5
}
}
Console
Clear

Sketch A4.7 random colour and other stuff

We can start to play around a bit. Notice that you don't always have to create variables when using random. Where appropriate, you can put them straight into the function.

! Move the `strokeWeight()` into the `while()` loop and randomise it, also notice we aren't using `increment` anymore.

```
let x = 10
let increment = 5

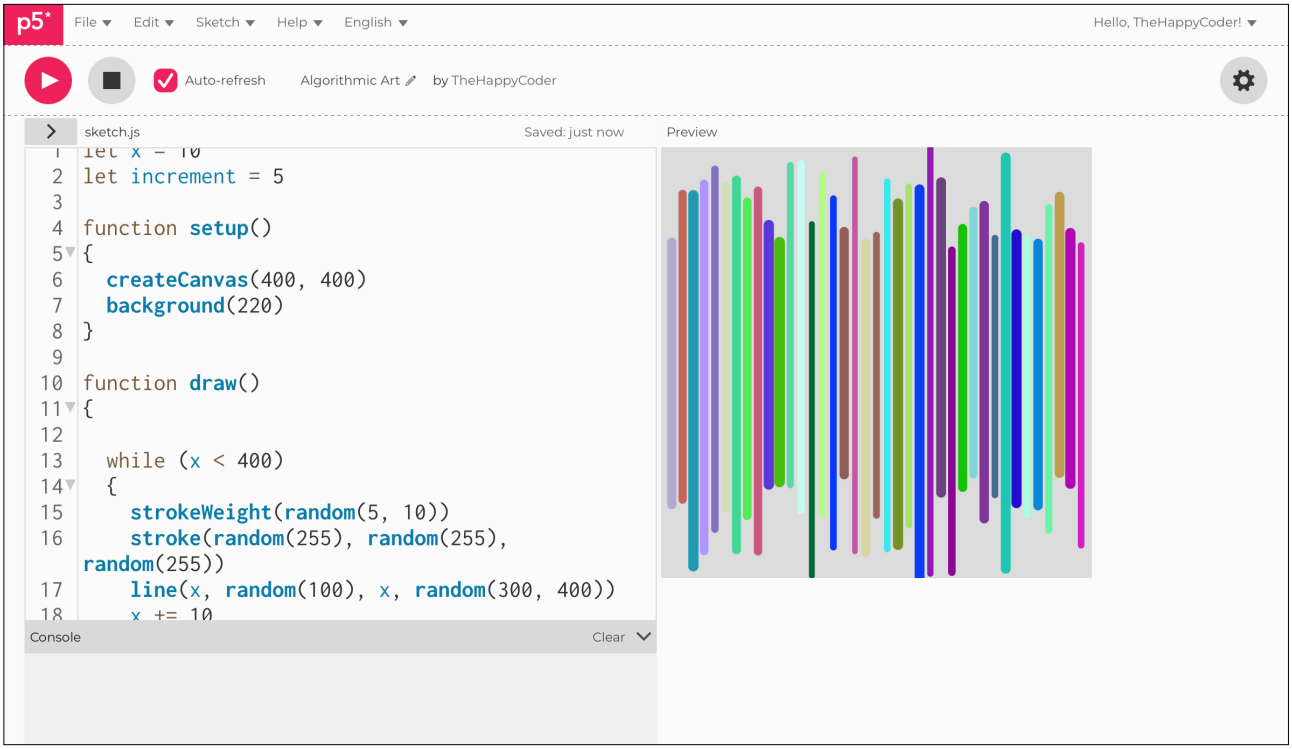
function setup()
{
  createCanvas(400, 400)
  background(220)
}

function draw()
{
  while (x < 400)
  {
    strokeWeight(random(5, 10))
    stroke(random(255), random(255), random(255))
    line(x, random(100), x, random(300, 400))
    x += 10
    increment += 5
  }
}
```

Notes

The code should be fairly obvious to you now, creates a pleasing effect

Figure A4.7


p5*
File
Edit
Sketch
Help
English
Hello, TheHappyCoder!
Auto-refresh
Algorithmic Art
by TheHappyCoder
sketch.js
Saved: just now
Preview
let x = 10
let increment = 5
function setup()
{
createCanvas(400, 400)
background(220)
}
function draw()
{
while (x < 400)
{
strokeWeight(random(5, 10))
stroke(random(255), random(255), random(255))
line(x, random(100), x, random(300, 400))
x += 10
Console
Clear

Sketch A4.8 newish sketch

! start a new sketch.

Here we have a faint line drawn randomly from the left side to the right side.

```
function setup()
{
  createCanvas(400, 400)
  background(255)
}

function draw()
{
  strokeWeight(0.1)
  line(0, random(400), 400, random(400))
}
```

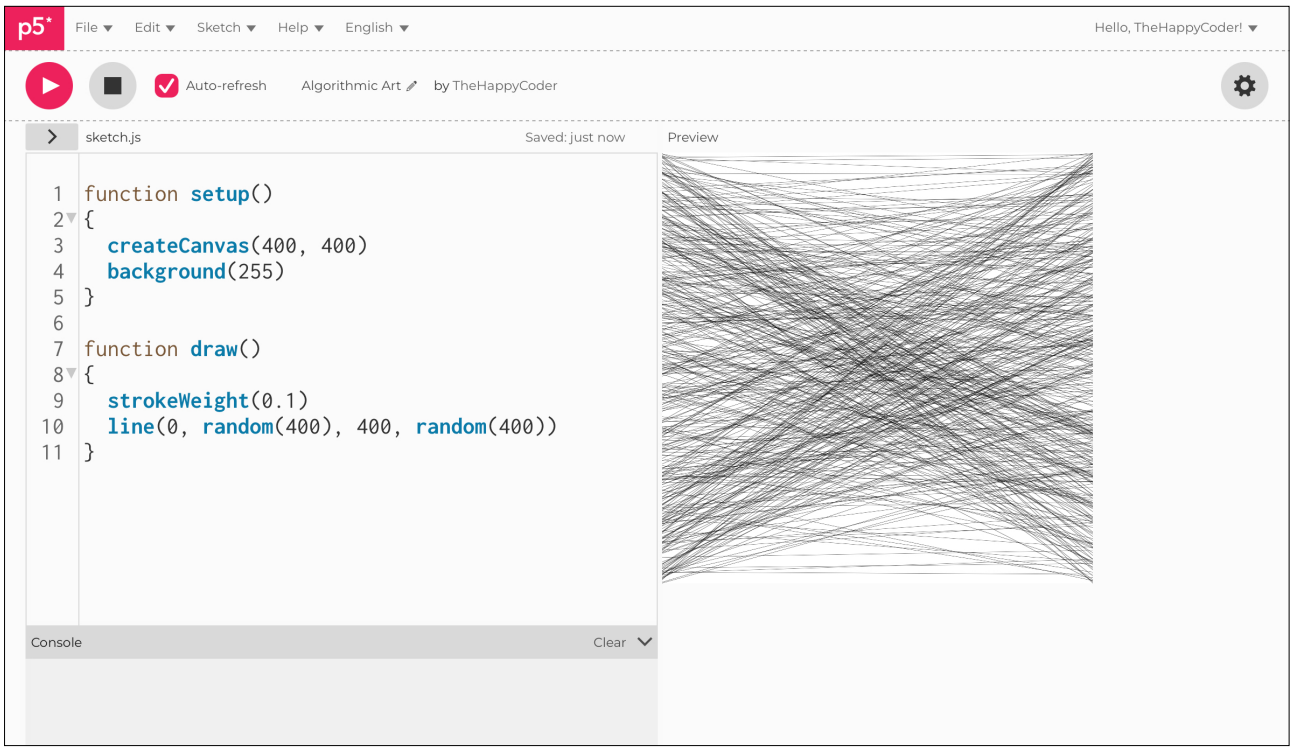
Notes

This will continue infinitum

Code Explanation

`strokeWeight(0.1)`	Gives a very thin weight to the line.

Figure A4.8


p5*
File
Edit
Sketch
Help
English
Hello, TheHappyCoder!
Auto-refresh
Algorithmic Art
by TheHappyCoder
sketch.js
Saved: just now
Preview
function setup()
{
createCanvas(400, 400)
background(255)
}
function draw()
{
strokeWeight(0.1)
line(0, random(400), 400, random(400))
}
Console
Clear



Sketch A4.9 more random lines

Let's add another random line generator. This time, top to bottom.

```
function setup()
{
  createCanvas(400, 400)
  background(255)
}

function draw()
{
  strokeWeight(0.1)
  line(0, random(400), 400, random(400))
  line(random(400), 0, random(400), 400)
}
```

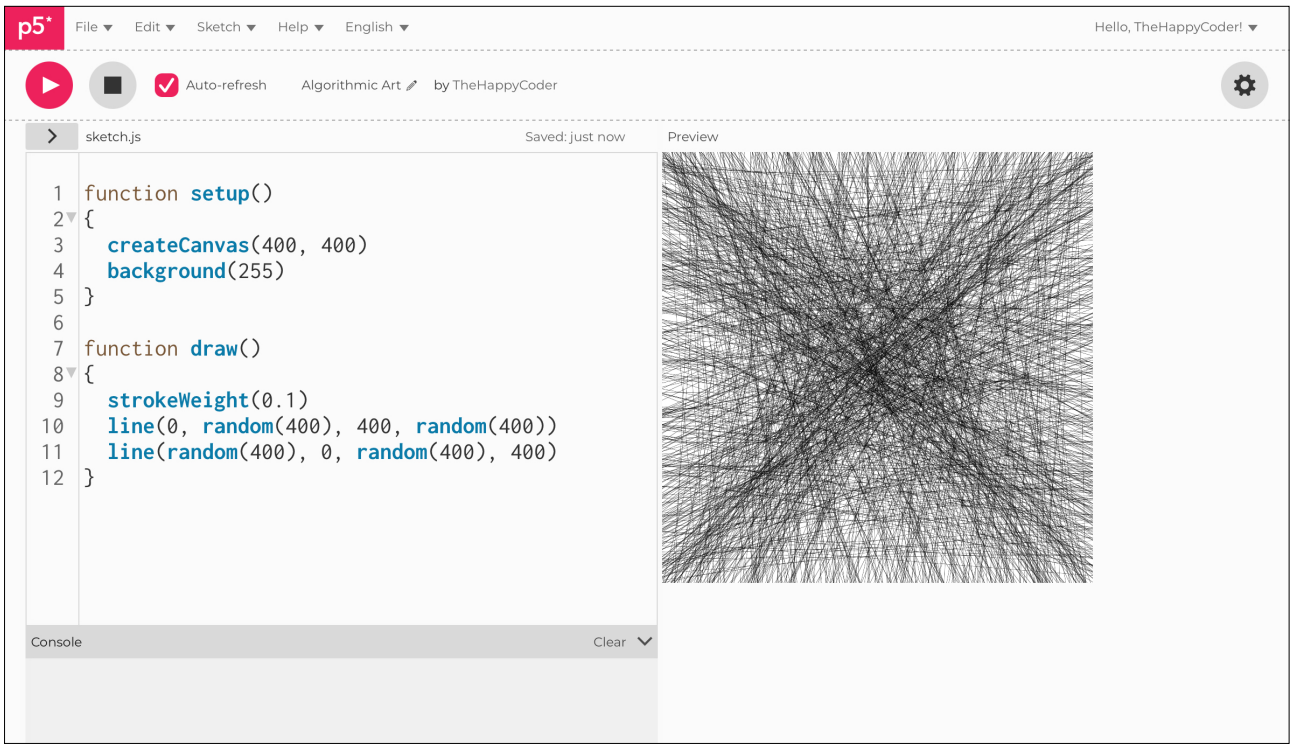
Notes

The mesh of lines fills up the canvas rapidly. What we would like to do is stop it before it is completely blacked out.

Challenge

Can you think of a way to stop it after, say, 300 lines?

Figure A4.9


p5*
File
Edit
Sketch
Help
English
Hello, TheHappyCoder!
Auto-refresh
Algorithmic Art
by TheHappyCoder
sketch.js
Saved: just now
Preview
function setup()
{
createCanvas(400, 400)
background(255)
}
function draw()
{
strokeWeight(0.1)
line(0, random(400), 400, random(400))
line(random(400), 0, random(400), 400)
}
Console
Clear



Sketch A4.10 we need to count the lines

We can create a random mesh. First, let's create a variable called `count` to count the number of lines.

```
let count = 0

function setup()
{
  createCanvas(400, 400)
  background(255)
}

function draw()
{
  strokeWeight(0.1)
  line(0, random(400), 400, random(400))
  line(random(400), 0, random(400), 400)
  count++
}
```

Notes

This does nothing except count the lines (two lines for every `count` increase) on each iteration of the `draw()` loop.

Sketch A4.11 if() statement

Another solution to limiting the number of lines is to use something called an `if()` statement. It has a condition attached; in this case, it is the `==` sign, which means `if it is equal to`. When that condition is `true`, then it activates the code inside the curly brackets `{...}`. In this instance, we want to stop drawing any more lines, so we use a function called `noLoop()`, which is pretty obvious what it does.

```
let count = 0

function setup()
{
  createCanvas(400, 400)
  background(255)
}

function draw()
{
  strokeWeight(0.1)
  line(0, random(400), 400, random(400))
  line(random(400), 0, random(400), 400)
  count++
  if (count == 300)
  {
    noLoop()
  }
}
```

Notes

The `if()` statement is similar to the `while()` loop; it is used more often than the `while()` loop. You can string lots of `if()` statements together and have what is called `if()...else()` statements (more on that later).

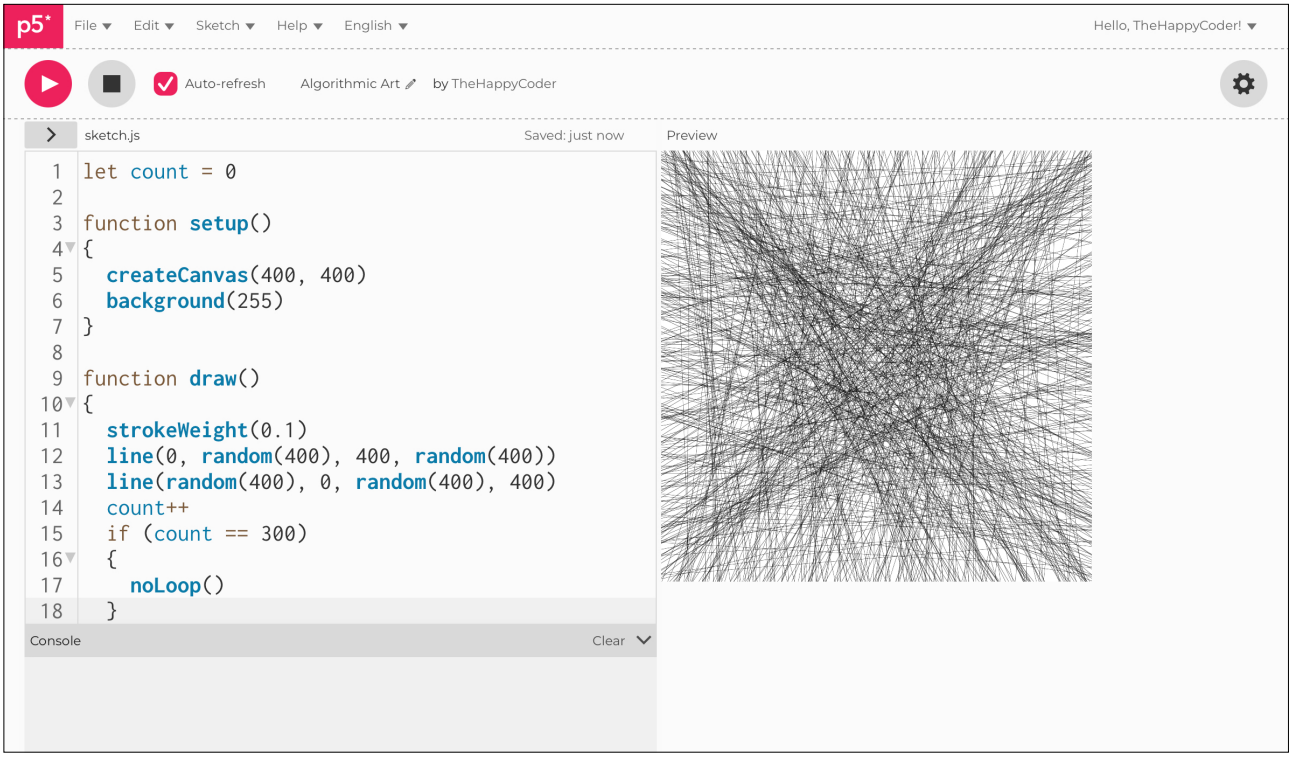
Challenges

1. Change the number of lines.
2. Change the weight and add alpha instead.
3. Have different colours.

Code Explanation

`if (count == 300)`	Checks to see if this condition is `true`.
`noLoop()`	If it is `true`, then it does this.

Figure A4.11


p5*
File
Edit
Sketch
Help
English
Hello, TheHappyCoder!
Auto-refresh
Algorithmic Art
by TheHappyCoder
sketch.js
Saved: just now
Preview
let count = 0
function setup()
{
createCanvas(400, 400)
background(255)
}
function draw()
{
strokeWeight(0.1)
line(0, random(400), 400, random(400))
line(random(400), 0, random(400), 400)
count++
if (count == 300)
{
noLoop()
}
Console
Clear