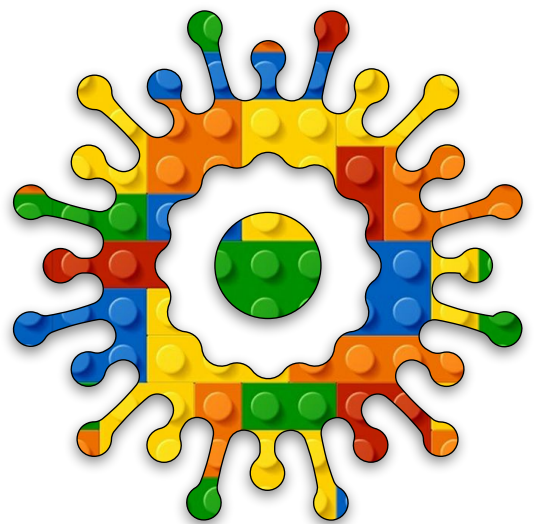


Creative
Coding
Module A
Unit #1
the p5.js
web editor





Module A Unit #1 exploring the p5.js web editor

What machine do I need?

The editor

A quick update

Signing in

The default sketch

Which format?

Looking at the buttons

The run and stop buttons

Naming the sketch

The auto refresh

Making the most of the general settings

Being more accessible

The main menu headings

The File heading

The Edit heading

The Sketch heading

The Help heading

The English heading

The purpose of the console



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.



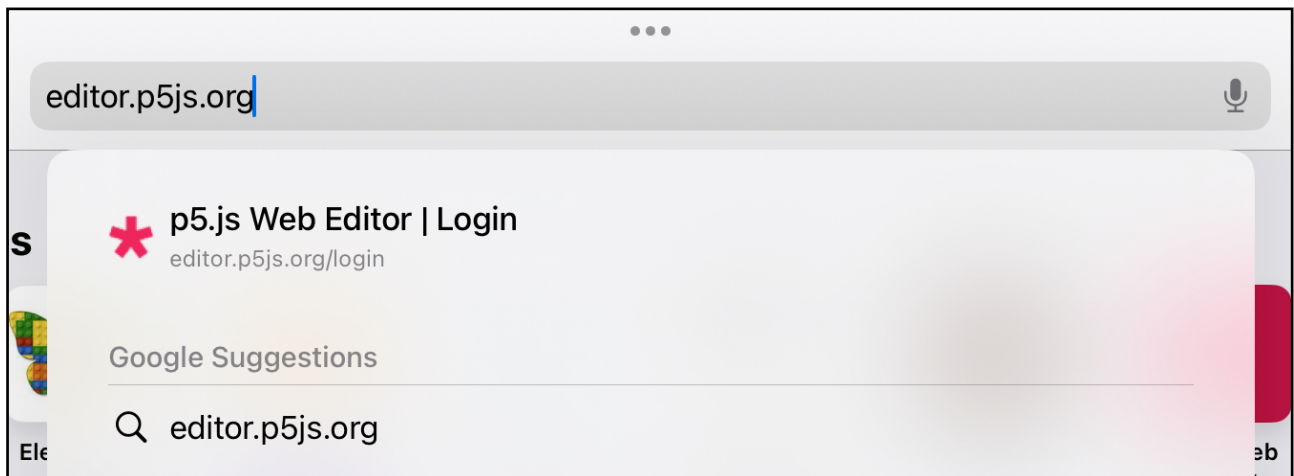
The editor (Fig. 1 & 2)

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

editor.p5js.org

into your web browser, see Fig. 1, 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 1: type **editor.p5js.org**



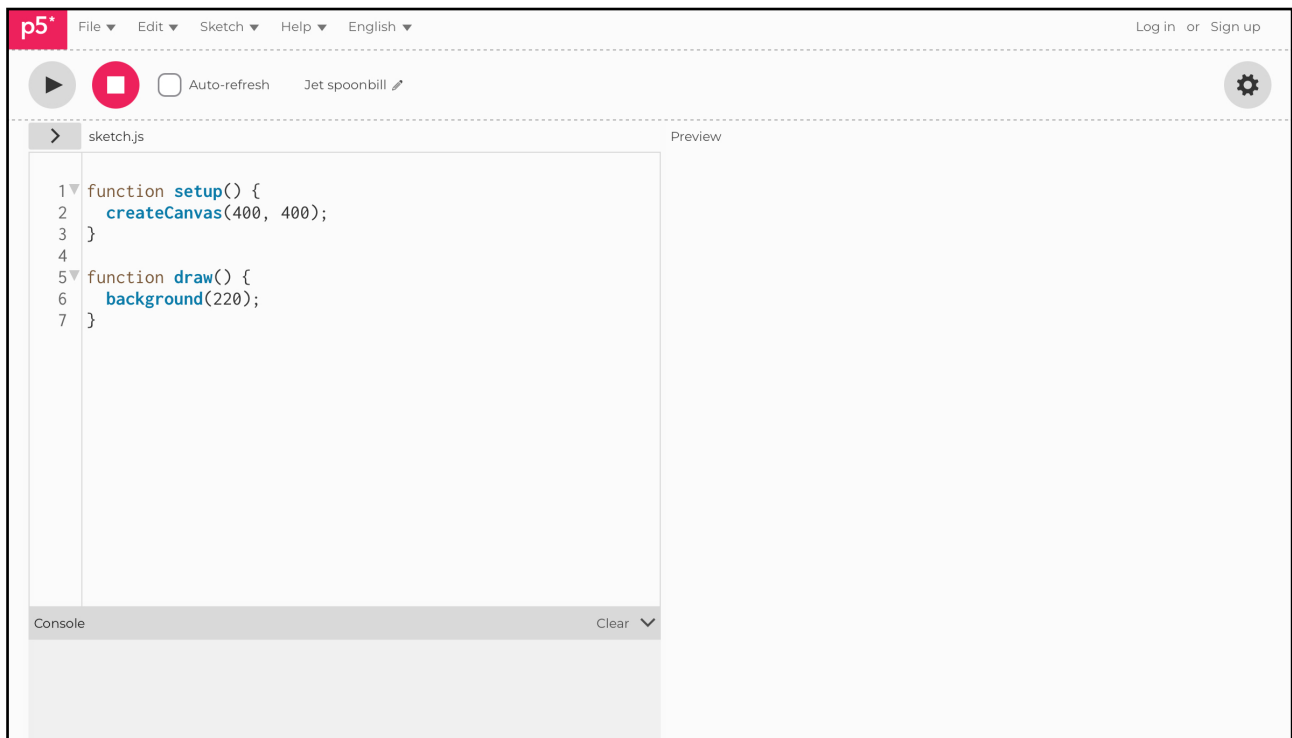
When you first start, you get a page like this shown in Fig. 2. You have a **main menu** across the top left-hand side. On the right-hand side, you have **Sign up** and **Log in** tabs. Below that, you have a number of buttons, a checkbox, and a text box.

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 2: the editor





A quick update (Fig. 3)

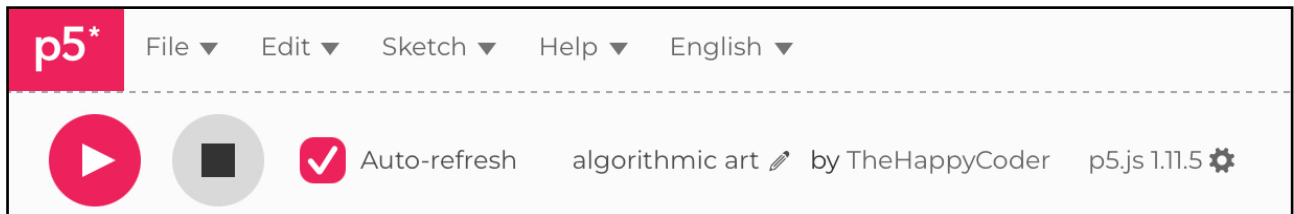
Even as I write this tutorial, there have been some updates. This is true of coding generally; they are always improving it as demand and use change. This is also true for p5.js and the web editor. Every so often, they have a new version, which generally goes unnoticed, but now you will be able to see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

This is in preparation for version 2 coming out next year. You will still be able to access older versions then, but the default will move over to version 2. We will be currently using version 1.11.5 (or higher), which should not be a problem for anything you will be covering.

For some of this tutorial, the images used were from before this new feature, so it may not be present in part of the tutorial. Fig. 3 shows the new web editor. When you get the chance, click on the version, and you will see a drop-down menu for you to explore at your leisure. You can even try the latest version 2, but until it becomes the default, I won't be using it in this tutorial.

I will check if it works with version 2 and include any changes.

Figure 3: the addition of the version





Signing in or logging in (Fig. 4 & 5)

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. 4).

Figure 4: signing up

The screenshot shows the 'Sign Up' page of the p5.js Editor. At the top left is the p5.js logo and a link to 'Back to Editor'. At the top right are links for 'Log in' and 'Sign up'. The main heading is 'Sign Up'. Below it are four input fields: 'User Name', 'Email', 'Password', and 'Confirm Password'. Each field has a small eye icon to toggle visibility. Below the fields is a 'Sign Up' button. Underneath is the word 'Or', followed by two buttons: 'Login with GitHub' and 'Login with Google'. At the bottom, there is a line of text: 'By signing up, you agree to the p5.js Editor's Terms of Use and Privacy Policy.' and a link: 'Already have an account? Log In'.

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

Figure 5: logging in

[< Back to Editor](#)[Log in](#) or [Sign up](#)

Log In

Email or Username

Password

Log In

or



Login with GitHub



Login with Google

Don't have an account? [Sign Up](#)

Forgot your password? [Reset your password](#)



The default sketch (Fig. 6)

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.

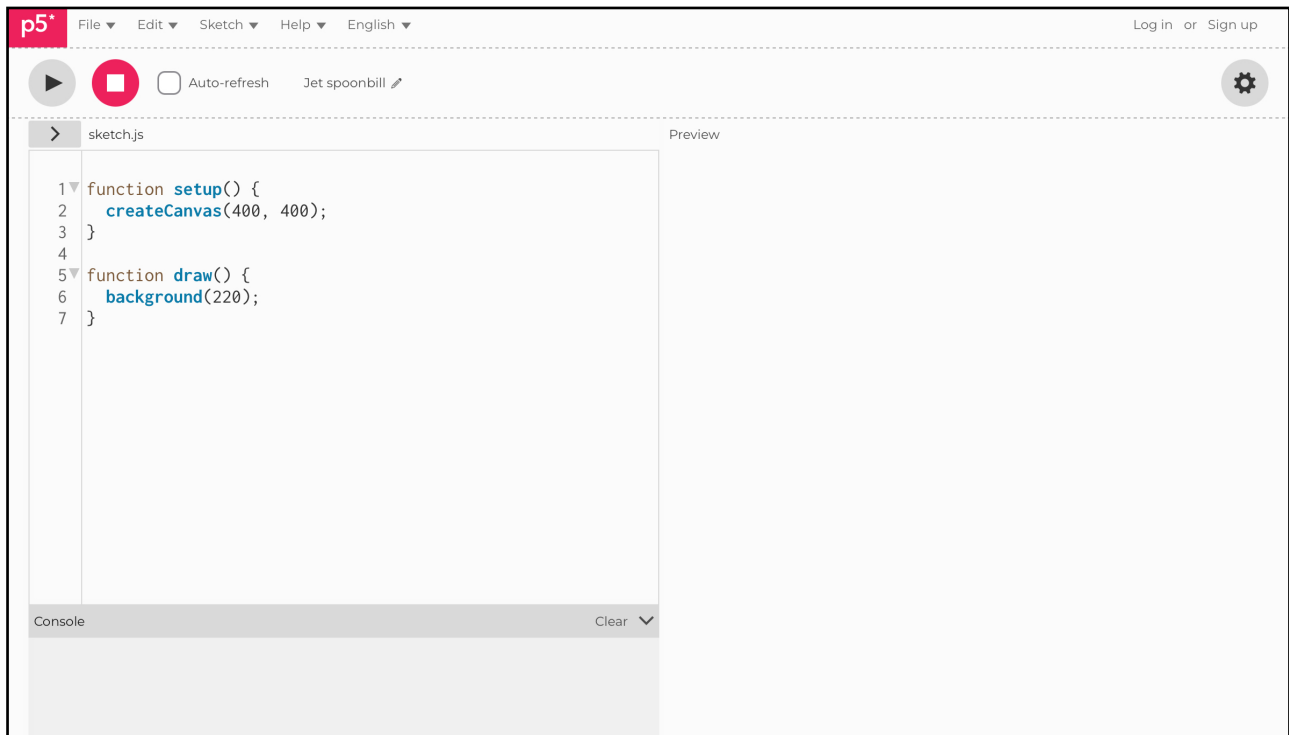
The second one, `function draw()`, is a loop; it draws on the canvas continually. This is where much of your code will go, although in some cases, you won't even have a `draw()` function, but more on that much later.

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. 7). 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 6: default sketch



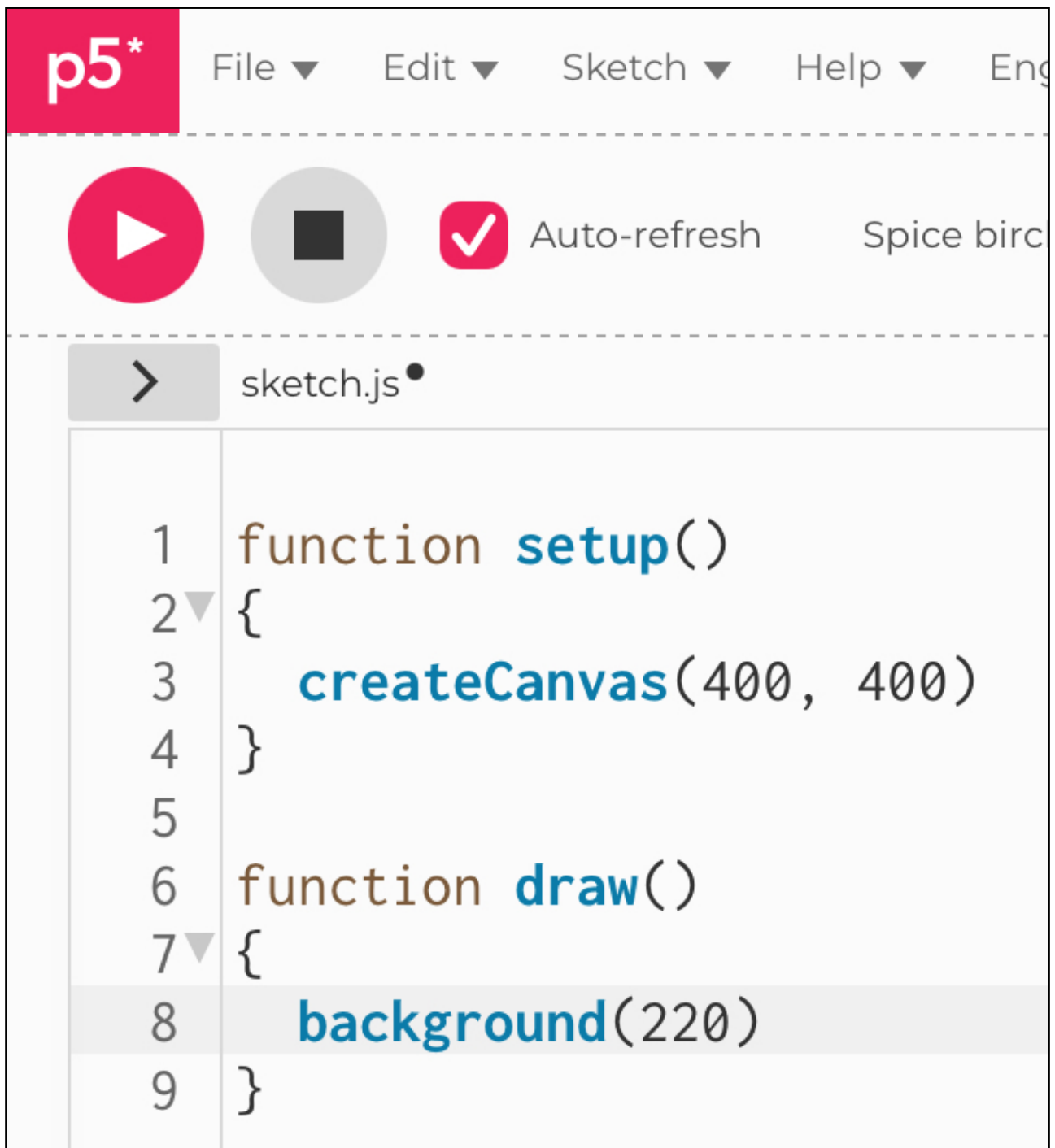


Which format? (Fig. 7)

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. 7.

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 7: my format





Looking at the buttons (Fig. 8)

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 8: main buttons





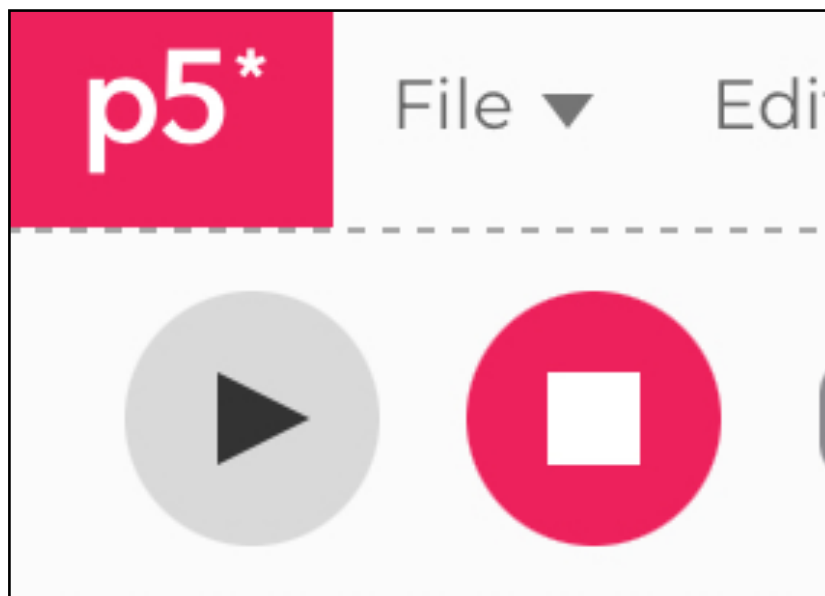
The run and stop buttons (Fig. 9)

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 9: run and stop





Naming the sketch (Fig. 10)

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 10: naming sketches

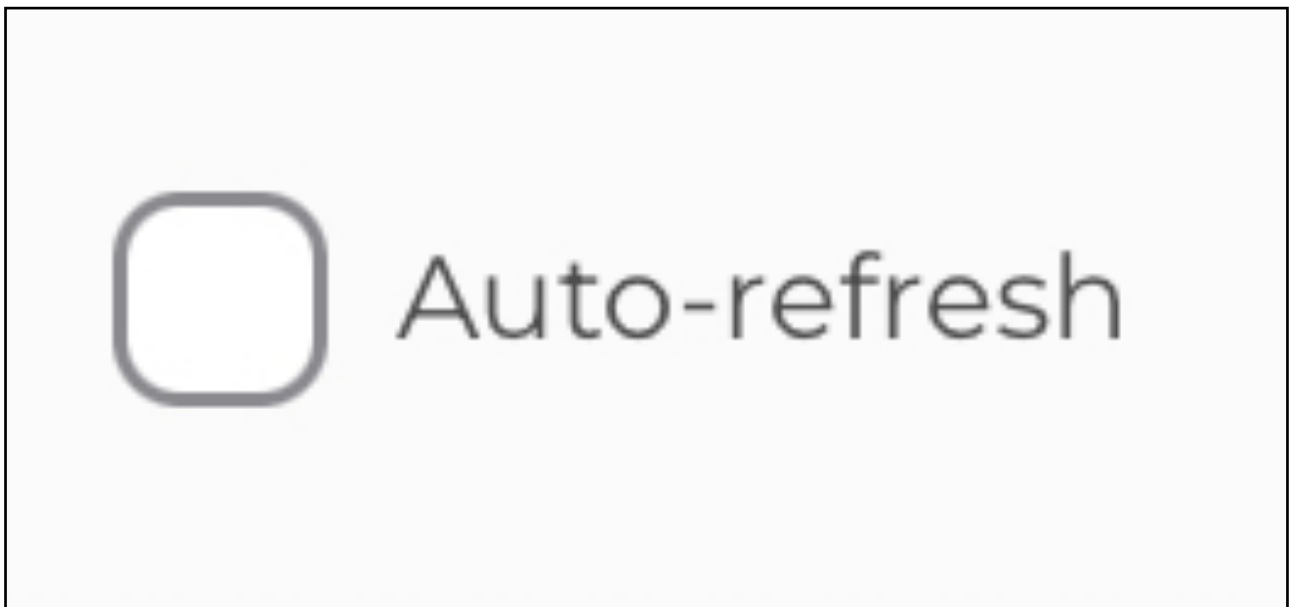
Jet spoonbill 



The auto refresh (Fig. 11)

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.

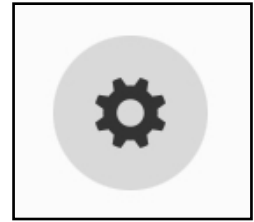
Figure 11: auto refresh





Making the most of the general settings (Fig. 12)

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 Hinder

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.

Figure 12: general settings

Settings

General settings

Accessibility

Theme

Light

Dark

High Contrast

Text Size

−

Decrease

20

+

Increase

Autosave

On

Off

Autoclose Brackets and Quotes

On

Off

Autocomplete Hinder

On

Off

...

...



Being more accessible (Fig. 13)

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

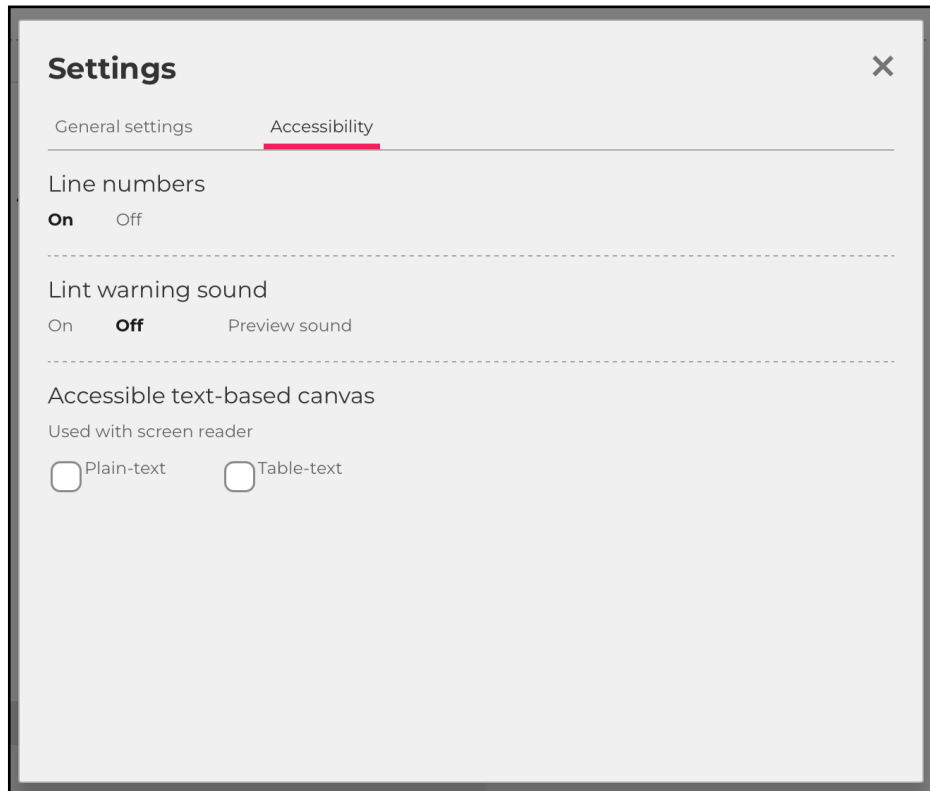
This gives you an audible sound if it detects an error in the code; again, it is a preference to have or not have it selected; I don't; I think it would drive me mad after a while.



Accessible text-based canvas

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

Figure 13: accessibility





The main menu tabs (Fig. 14)

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 14: main menu





The File menu (Fig. 15)

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.

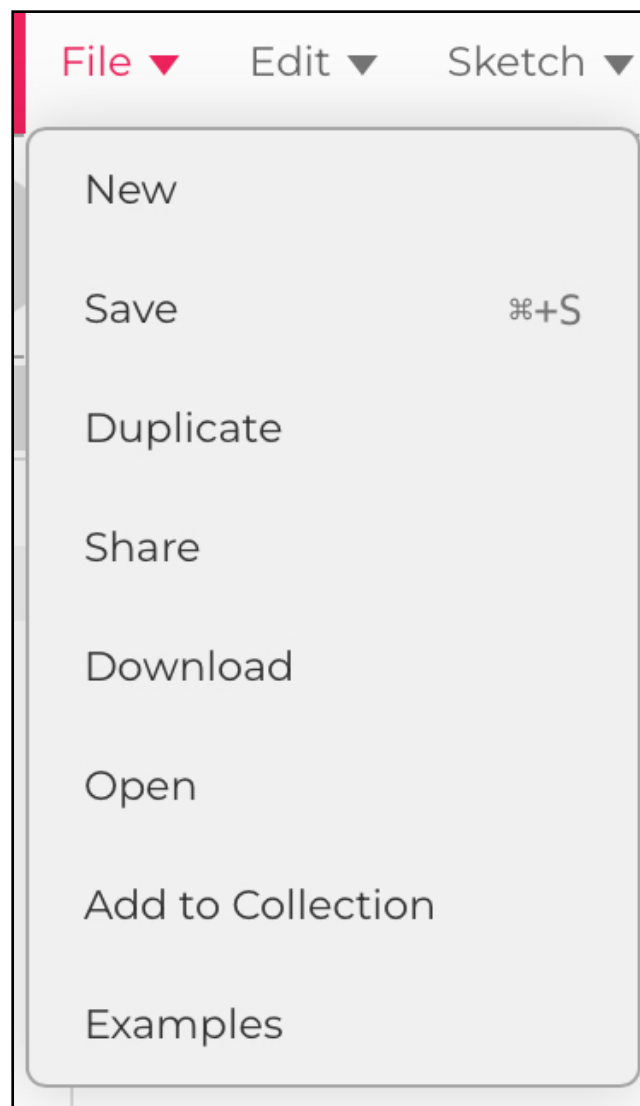
Add to Collection

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 15: file tab





The Edit menu (Fig. 16)

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

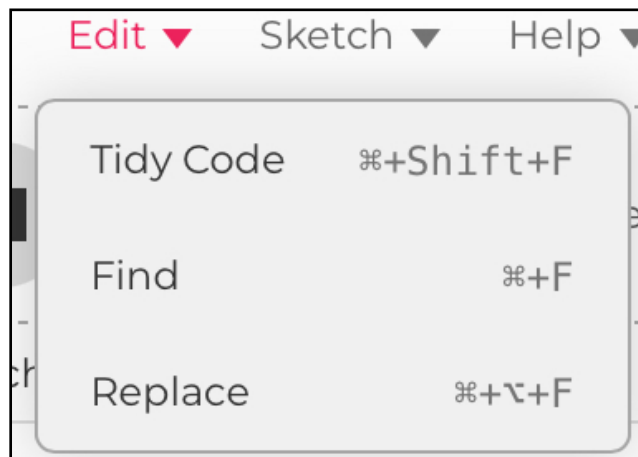
This will search for any specific keyword or phrase in your code.



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 16: edit tab





The Sketch menu (Fig. 17)

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

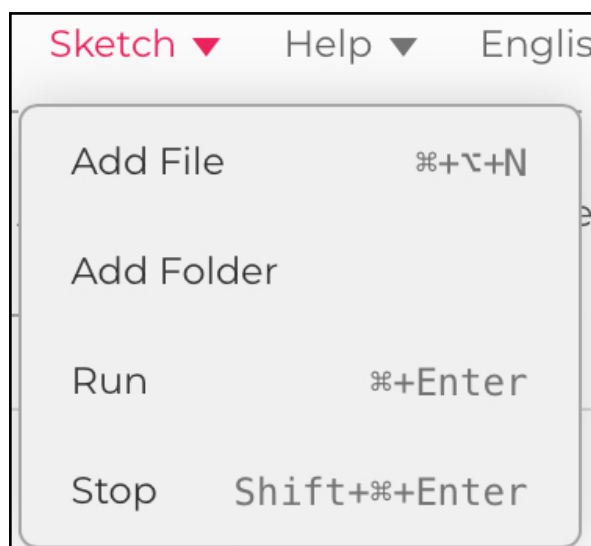
This does the same as the button.



Stop

This also does the same as the button.

Figure 17: sketch tab





The Help menu (Fig:18)

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

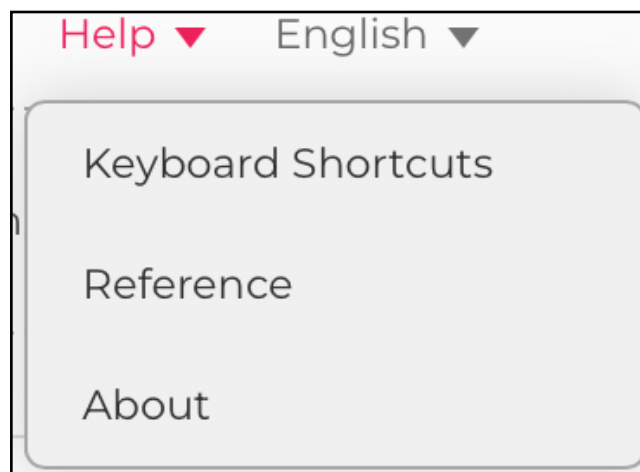
This has some links and general information.



English tab

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

Figure 18: help tab





The purpose of the console (Fig. 19)

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.

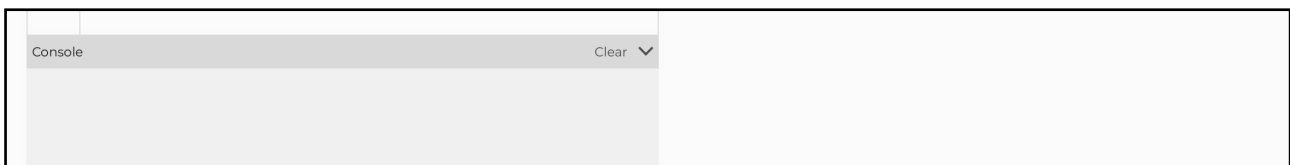
Sometimes the error code may be highlighted in red, and it may even give you the line number to identify where the problem is (or the following line number where it does become a problem).



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 19: the console





Final words on this unit

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 and the use they may or may not be.

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