A guide to using Sonic Pi

Made by the Digital Inclusion, Skills and Creativity (DISC) team at the University of York.

Sonic Pi is a coding tool that allows you to "live code" music. Live coding means you can be playing music and changing the code as you go, creating a seamless music experience. You can also use Sonic Pi to learn coding and key features of coding like loops, as it is based on the coding language Ruby.

In some ways, Sonic Pi requires a balance between coding and music, as some features work in a way that will make sense musically and other features work in a way that will make sense for coding, but make less sense if you're used to reading and creating music.

Throughout this guide, we will link to Sonic Pi's own tutorial for further information.

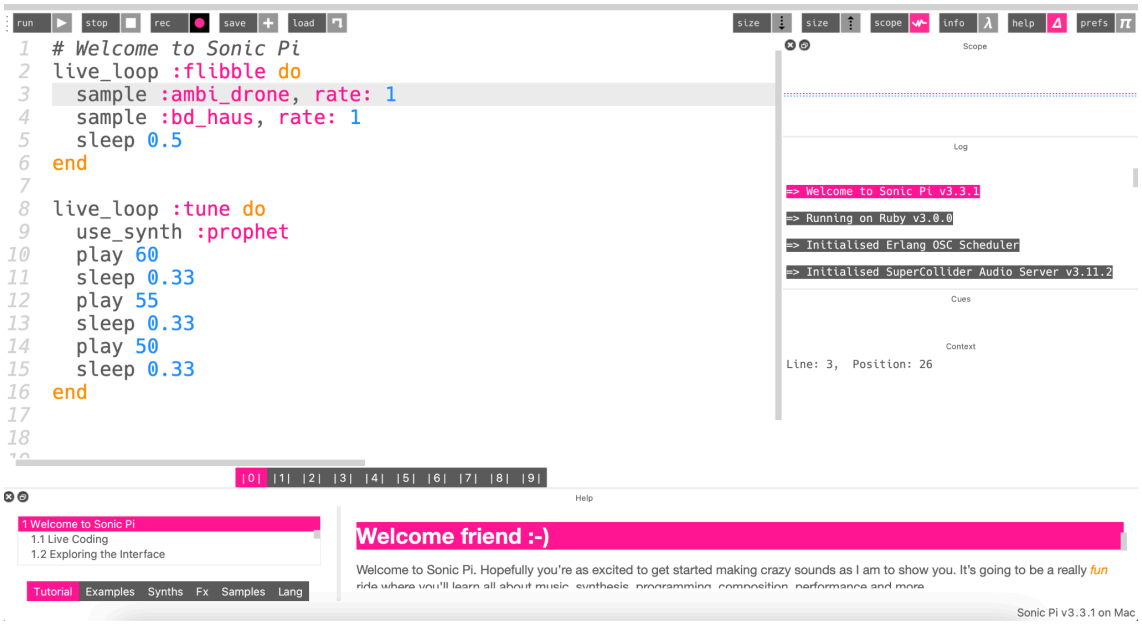
If you'd like to have a go at making things in Sonic Pi, you could also work through our Sonic Pi exercises.

Installing Sonic Pi

Sonic Pi is available for Windows, MacOS and Raspberry Pi OS and can be downloaded from the Sonic Pi website for free.

The Sonic Pi interface

When you open Sonic Pi, you will get a few different sections in the interface. The main part is the code editor, which is the section on the left where you can write your Sonic Pi code. There is also sections for the log of what Sonic Pi is doing (on the right) and a pane that can show the tutorial or lists of examples, synths, FX, and samples (on the bottom left). Here's what it looks like:



Basically, you can write code in the code editor, and then use the run / stop buttons at the top to control if your code plays or not. There are also other buttons along the top you can explore.

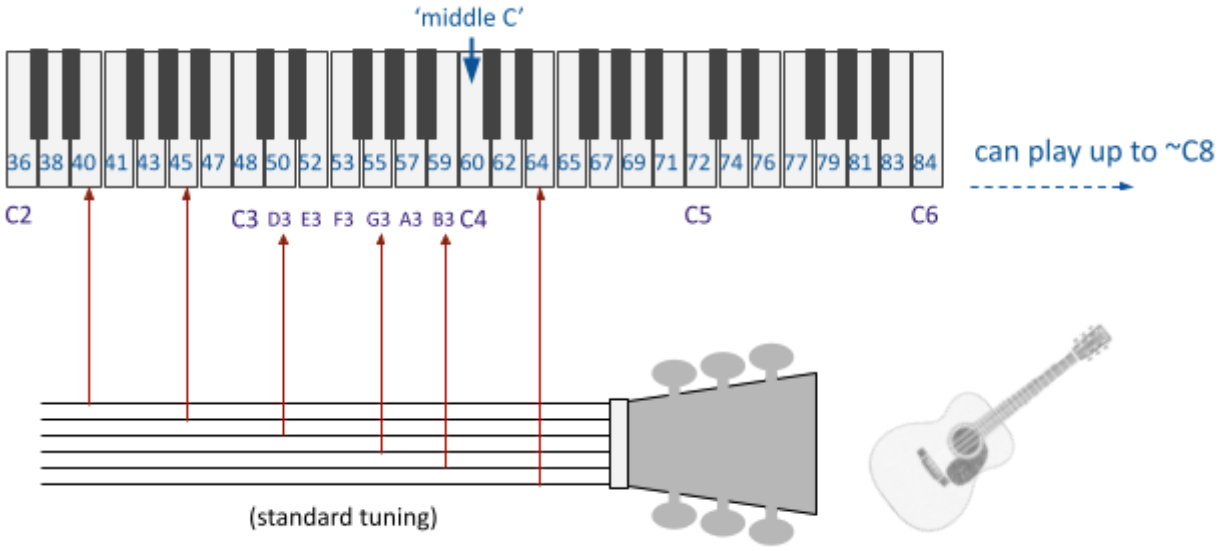
Underneath the code editor are the buffer numbers. Basically, you can have a few different bits of Sonic Pi code open at once and move between them by clicking on the buffer numbers.

[Sonic Pi tutorial: Exploring the interface]

Notes

Notes can be represented either by a number or the note name and octave.

- Using numbers enables you to use calculated or even random notes
- Playing with named notes is easier for coding a definite tune if you are familiar with musical terminology



Use the keyword **play** followed by the note you want - numbers don't need anything else, but named notes need to start with a colon **:** (named notes are not case sensitive) e.g. **play 60** or **play :C4**

To play multiple notes, you can use **play_pattern** and then put the notes in square brackets**[]**, separated by commas (there must be a space between play_pattern and the square brackets), e.g. **play_pattern [60,75,60,41]** or **play_pattern [:c3,:d4:a4,:c4]**

You can also do chords (combinations of notes) can be represented either by several note names/numbers or the chord names, using **play_chord**

If you want to use multiple commands in a row, e.g. multiple play or play_chord commands, you will need to use **sleep** to add a break between them playing, otherwise Sonic Pi will just play notes at the same time. This is strange for music, but it makes sense

in coding - computers will do things immediately after one another unless you tell them otherwise, and this can often appear to be simultaneous because it is so quick.

To use sleep, just write **sleep** and then a time in seconds e.g. **sleep 2** or **sleep 0.5**

use_bpm allows you to set a tempo for a piece in 'beats per minute'.

play_pattern_timed is like play_pattern, but allows you to specify the length of each note afterwards in a separate set of square brackets, e.g. **play_pattern_timed [40,52,50,52],[0.3,0.4,0.4,0.5]** or **play_pattern_timed [:e3,:e4,:d4,:b3], [0.3,0.4,0.4,0.5]**

You can also create rests in play patterns (so, silence) using **:r**, **:rest**, or **nil**.

Synths

Just using the **play** methods, gives relatively plain sounds. To make it more interesting, select a specific synthesiser sound before you play the notes or chords. Type **use_synth** and then a space and you'll get a list of synths you can set, e.g. **use_synth :prophet**

You can change synths later in the code too - just use **use_synth** again!

Sonic Pi tutorial: Switching synths

Samples

The other way to generate a sound is to use **samples**, which are audio clips, and could be music, environmental sounds, speech etc - anything that could be produced and recorded.

To play a sample write **sample** and then do a space and it will suggest built in samples you could use, e.g. **sample :loop_amen**

You can change parameters for the samples and there's a range of parameters you can use. See the information on Sonic Pi tutorial: Samples for the full range, but a handy one to know is changing the **rate** of a sample, which can essentially stretch or squish the sample, making it slower and lower or faster and higher, e.g. **sample :loop_amen, rate: 0.5**

You can even use a rate of -1 to make it play backwards!

You can also use your own samples, but you need WAV, AIFF or FLAC files - .mp3 files don't work. To play a sample stored on your own device, use the **sample** command and then inside double quotes **""** put the file path, which will be slightly different depending on your operating system.

Windows:

sample "C:/Users/name/folder/my-sound.wav"

Mac, Linux, Raspberry Pi:

sample "/Users/name/folder/my-sound.wav"

Comments

Like any other coding language, Sonic Pi allows you to add comments to your code. Comments are parts of code that the computer skips, because they are designed for humans to read, not computers. Comments can be used to add notes and reminders, or even to stop a line of code from running whilst you work out what was wrong. In the context of Sonic Pi, comments can be used to remove a line to see what effect that has on the music you're playing.

To create a comment, use the hash symbol **#** and then anything written afterwards on that line will not be run by the computer.

Loops

Loops are what makes Sonic Pi really powerful. As well as being able to do more traditional kinds of loops for music or coding, like repeating a section a specified number of times using **.times**, you can have a **loop** that runs forever, and, most importantly, a **live_loop** that loops forever but also allows you to edit the loop and hit Run again for it to change into that version of the code seamlessly. This is the feature of Sonic Pi that makes it great for live performances, as you can change the sound of the music without interruption.

All loops in Sonic Pi start with **do** and finish with **end** and then have code in the middle, but what you put before **do** changes what kind of loop it is and how it works.

Using **.times do** with a number in front of the **.** like **2.times do** or **5.times do** will repeat the code within the loop that many times, so

```
4.times do
    play_pattern[60,71,59,71]
end
```

will play that pattern of notes 4 times.

Using **loop do** creates a loop that will run when you hit the Run button until you press Stop. If you press the Run button again before stopping, it will start an additional instance playing, rather than replace it (so you can create so quite horrible sounds by accident).

Using **live_loop :loopName do** creates a loop that will run when you hit the Run button until you press Stop and if you press Run again, it will continue playing the same instance, but incorporate any changes you've made since you last pressed Run. Using a live loop is

crucial to Sonic Pi as it allows you to modify your code live, making it great for performances. You have to give a live_loop a name, but that name can be anything, from :melody to :banana

Sonic Pi tutorial: Iteration and loops

Defining functions

What if you wanted to repeat code, but not immediately? In Sonic Pi, like in most coding languages, you can create what are called 'functions', which is a section of code that is given a name, and then you can make that bit of code run when you write the name. It is similar to having a verse or a chorus in music, and repeating this in difference places in your song.

To create a function you use **define :functionname do** and then put the code, and then close it with **end** so you'll end up with something like

```
define :chorus do
    play 78
    sleep 1
    play_pattern_timed [67,78,54,78,54],[0.4,0.3,0.4,0.4,0.3]
    sleep 2
end

chorus
```

You have to write the name of the function in your code to make it happen, and you can put it inside a loop if you want it to play multiple times (or have loops inside your function).

Sonic Pi tutorial: Functions

Further coding features

There are even more features of coding that are available within Sonic Pi, like conditions and variables. See Sonic Pi tutorial: Programming Structures for these.

Examples

Basic tune in a live loop (called 'twinkie') with the tempo set at 120 bpm

```
live_loop :twinkie do
  use_bpm 120
    play_pattern [:D4,:D4,:A4,:A4,:B4,:B4]
    play :A4
    sleep 2
    play_pattern [67,67,66,66,64,64]
    play chord(:D4, :M)
    sleep 2
 end
```

Use a synth for the sound

```
live_loop :twinkie do
  use_bpm 120
  use_synth :pluck
    play_pattern [:D4,:D4,:A4,:A4,:B4,:B4]
    play :A4
    sleep 2
    play_pattern [67,67,66,66,64,64]
    play chord(:D4, :M)
    sleep 2
 end
```

Same tune using timed pattern, and some effects around the sound

```
live_loop :twinkie do
  use_bpm 120
  use_synth :pluck
  with_fx :reverb, room: 1 do
    play_pattern_timed [:D4,:D4,:A4,:A4,:B4,:B4,:A4], [1,1,1,1,1,1,2]
    play_pattern_timed [67,67,66,66,64,64,60], [1,1,1,1,1,1,2]
```

```
    end
end
```