

**MANUAL**  
**SONIC PI**

- 1 Welcome to Sonic Pi
  - 1.1 Live Coding
  - 1.2 Exploring the Interface
  - 1.3 Learning through Play
- 2 Synths
  - 2.1 Your First Beeps
  - 2.2 Synth Options
  - 2.3 Switching Synths
  - 2.4 Duration with Envelopes
- 3 Samples
  - 3.1 Triggering Samples
  - 3.2 Sample Parameters
  - 3.3 Stretching Samples
  - 3.4 Enveloped Samples
  - 3.5 Partial Samples
  - 3.6 External Samples
- 4 Randomisation
- 5 Programming Structures
  - 5.1 Blocks
  - 5.2 Iteration and Loops
  - 5.3 Conditionals
  - 5.4 Threads
  - 5.5 Functions
  - 5.6 Variables
  - 5.7 Thread Synchronisation
- 6 FX
  - 6.1 Adding FX
  - 6.2 FX in Practice
- 7 Control
  - 7.1 Controlling Running Synths
  - 7.2 Controlling FX
  - 7.3 Sliding Options
- 8 Data Structures
  - 8.1 Lists
  - 8.2 Chords
  - 8.3 Scales
  - 8.4 Rings
- 9 Live Coding
  - 9.1 Live Coding Fundamentals
  - 9.2 Live Loops
  - 9.3 Multiple Live Loops
  - 9.4 Ticking
- 10 Essential Knowledge
  - 10.1 Using Shortcuts
  - 10.2 Shortcut Cheatsheet
  - 10.3 Sharing
  - 10.4 Performing
- 11 Minecraft Pi
  - 11.1 Basic API
- 12 Conclusions

# 1 - Welcome friend :-)

Welcome to Sonic Pi. Hopefully you're as excited to get started making crazy sounds as I am to show you. It's going to be a really *fun* ride where you'll learn all about music, synthesis, programming, composition, performance and more.

But wait, how rude of me! Let me introduce myself - I'm [Sam Aaron](#) - the chap that created Sonic Pi. You can find me at [@samaaron](#) on Twitter and I'd be more than happy to say hello to you. You might also be interested in finding out more about my [Live Coding Performances](#) where I code with Sonic Pi live in front of audiences.

If you have any thoughts, or ideas for improving Sonic Pi - please pass them on - feedback is so helpful. You never know, your idea might be the next big feature!

This tutorial is divided up into sections grouped by category. Whilst I've written it to have an easy learning progression from start to finish, feel very free just to dip in and out of sections as you see fit. If you feel that there's something missing, do let me know and I'll consider it for a future version.

Finally, watching others live code is a really great way to learn. I regularly stream live on [livecoding.tv/samaaron](#) so please do drop by, say hi and ask me lots of questions :-)

OK, let's get started...

---

## 1.1 - Live Coding

One of the most exciting aspects of Sonic Pi is that it enables you to write and *modify code live* to make music, just like you might perform live with a guitar. This means that given some practice you can take Sonic Pi on stage and gig with it.

### Free your mind

Before we get into the real details of how Sonic Pi works in the rest of this tutorial, I'd like to give you an experience of what it's like to live code. Don't worry if you don't understand much (or any) of this. Just try to hold onto your seats and enjoy...

### A live loop

Let's get started, copy the following code into an empty buffer above:

```
live_loop :flibble do
  sample :bd_haus, rate: 1
  sleep 0.5
end
```

Now, press the **Run** button and you'll hear a nice fast bass drum beating away. If at any time you wish to stop the sound just hit the **Stop** button. Although don't hit it just yet... Instead, follow these steps:

1. Make sure the bass drum sound is still running
2. Change the **sleep** value from **0.5** to something higher like **1**.
3. Press the **Run** button again
4. Notice how the drum speed has changed.
5. Finally, *remember this moment*, this is the first time you've live coded with Sonic Pi and it's unlikely to be your last...

Ok, that was simple enough. Let's add something else into the mix. Above **sample :bd\_haus** add the line **sample :ambi\_choir, rate: 0.3**. Your code should look like this:

```
live_loop :flibble do
  sample :ambi_choir, rate: 0.3
  sample :bd_haus, rate: 1
  sleep 1
end
```

Now, play around. Change the rates - what happens when you use high values, or small values or negative values? See what happens when you change the **rate:** value for the **:ambi\_choir** sample just slightly (say to 0.29). What happens if you choose a really small **sleep** value? See if you can make it go so fast your computer will stop with an error because it can't keep up (if that happens, just choose a bigger **sleep** time and hit **Run** again).

Try commenting one of the **sample** lines out by adding a **#** to the beginning:

```
live_loop :flibble do
  sample :ambi_choir, rate: 0.3
  # sample :bd_haus, rate: 1
  sleep 1
end
```

Notice how it tells the computer to ignore it, so we don't hear it. This is called a comment. In Sonic Pi we can use comments to remove and add things into the mix.

Finally, let me leave you something fun to play with. Take the code below, and copy it into a spare buffer. Now, don't try to understand it too much other than see that there are two loops - so two things going round at the same time. Now, do what you do best - experiment and play around. Here are some suggestions:

- Try changing the blue **rate:** values to hear the sample sound change.
- Try changing the **sleep** times and hear that both loops can spin round at different rates.
- Try uncommenting the sample line (remove the **#**) and enjoy the sound of the guitar played backwards.
- Try changing any of the blue **mix:** values to numbers between **0** (not in the mix) and **1** (fully in the mix).

Remember to press **Run** and you'll hear the change next time the loop goes round. If you end up in a pickle, don't worry - hit **Stop**, delete the code in the buffer and paste a fresh copy in and you're ready to jam again. Making mistakes is how you'll learn the quickest...

```
live_loop :guit do
  with_fx :echo, mix: 0.3, phase: 0.25 do
    sample :guit_em9, rate: 0.5
  end
# sample :guit_em9, rate: -0.5
  sleep 8
end
```

```
live_loop :boom do
  with_fx :reverb, room: 1 do
    sample :bd_boom, amp: 10, rate: 1
  end
end
```

```
end  
sleep 8  
end
```

Now, keep playing and experimenting until your curiosity about how this all actually works kicks in and you start wondering what else you can do with this. You're now ready to read the rest of the tutorial.

So what are you waiting for...

---

## 1.2 - The Sonic Pi Interface

Sonic Pi has a very simple interface for coding music. Let's spend a little time exploring it.



- A - Play Controls
- B - Editor Controls
- C - Info and Help
- D - Code Editor
- E - Prefs Panel
- F - Log Viewer
- G - Help System

### A. Play Controls

These pink buttons are the main controls for starting and stopping sounds. There's the **Run** button for running the code in the editor, **Stop** for stopping all running code, **Save** for saving the code to an external file and **Record** to create a recording (a WAV file) of the sound playing.

### B. Editor Controls

These orange buttons allow you to manipulate the code editor. The **Size +** and **Size -** buttons allow you to make the text bigger and smaller. The **Align** button will neaten the code for you to make it look more professional.

### C. Info and Help

These blue buttons give you access to information, help and preferences. The **Info** button will open up the information window which contains information about Sonic Pi itself - the core team, history, contributors and community. The **Help** button toggles the help system (F) and the **Prefs** button toggles the preferences window which allows you to control some basic system parameters.

### D. Code Editor

This is the area where you'll write your code and compose/perform music. It's a simple text editor where you can write code, delete it, cut and paste, etc. Think of it like a very basic version of Word or Google Docs. The editor will automatically colour words based on their meaning in the code. This may seem strange at first, but you'll soon find it very useful. For example, you'll know something is a number because it is blue.

### E. Prefs Panel

Sonic Pi supports a number of tweakable preferences which can be accessed by toggling the **prefs** button in the Info and Help button set. This will toggle the visibility of the Prefs Panel which includes a number of options to be changed. Examples are forcing mono mode, inverting stereo, Toggling log output verbosity and also a volume slider and audio selector on the Raspberry Pi.

### F. Log Viewer

When you run your code, information about what the program is doing will be displayed in the log viewer. By default, you'll see a message for every sound you create with the exact time the sound was triggered. This is very useful for debugging your code and understanding what your code is doing.

### G. Help System

Finally, one of the most important parts of the Sonic Pi interface is the help system which appears at the bottom of the window. This can be toggled on and off by clicking on the blue **Help** button. The help system contains help and information about all aspects of Sonic Pi including this tutorial, a list of available synths, samples, examples, FX and a full list of all the functions Sonic Pi provides for coding music.

## 1.3 - Learning through Play

Sonic Pi encourages you to learn about both computing and music through play and experimentation. The most important thing is that you're having fun, and before you know it you'll have accidentally learned how to code, compose and perform.

### There are no mistakes

Whilst we're on this subject, let me just give you one piece of advice I've learned over my years of live coding with music - *there are no mistakes, only opportunities*. This is something I've often heard in relation to jazz but it works equally well with live coding. No matter how experienced you are - from a complete beginner to a seasoned Algoraver, you'll run some code that has a completely unexpected outcome. It might sound insanely cool - in which case run with it. However, it might sound totally jarring and out of place. It doesn't matter that it happened - what matters is what you do next with it. Take the sound, manipulate it and morph it into something awesome. The crowd will go *wild*.

### Start Simple

When you're learning, it's tempting to want to do amazing things *now*. However, just hold that thought and see it as a distant goal to reach *later*. For now, instead think of the *simplest* thing you could write which would be fun and rewarding that's a small step towards the amazing thing you have in your head. Once you have an idea about that simple step, then try and build it, play with it and then see what new ideas it gives you. Before long you'll be too busy having fun and making real progress.

Just make sure to share your work with others!

---

## 2 - Synths

OK, enough of the intros - let's get into some sound.

In this section we'll cover the basis of triggering and manipulating synths. Synth is short for synthesiser which is a fancy word for something which creates sounds. Typically synths are quite complicated to use - especially analog synths with many patch wires and modules. However, Sonic Pi gives you much of that power in a very simple and approachable manner.

Don't be fooled by the immediate simplicity of Sonic Pi's interface. You can get very deep into very sophisticated sound manipulation if that's your thing. Hold on to your hats...

---



## 2.1 - Your First Beeps

Take a look at the following code:

```
play 70
```

This is where it all starts. Go ahead, copy and paste it into the code window at the top of the app (the big white space under the Run button). Now, press Run...

### Beep!

Intense. Press it again. And again. *And again...*

Woah, crazy, I'm sure you could keep doing that all day. But wait, before you lose yourself in an infinite stream of beeps, try changing the number:

```
play 75
```

Can you hear the difference? Try a lower number:

```
play 60
```

So, lower numbers make lower pitched beeps and higher numbers make higher pitched beeps. Just like on a piano, the keys at the lower part of the piano (the left hand side) play lower notes and the keys on the higher part of the piano (the right hand side) play higher notes. In fact, the numbers actually relate to notes on the piano. `play 47` actually means play the 47th note on the piano. Which means that `play 48` is one note up (the next note to the right). It just so happens that the 4th octave C is number 60. Go ahead and play it: `play 60`.

*Don't worry* if this means nothing to you - it didn't to me when I first started. All that matters right now is that you know that *low numbers make lower beeps* and *high numbers make higher beeps*.

### Chords

Playing a note is quite fun, but playing many at the same time can be even better. Try it:

```
play 72  
play 75  
play 79
```

Jazzy! So, when you write multiple `plays`, they all play at the same time. Try it for yourself - which numbers sound good together? Which sound terrible? Experiment, explore and find out for yourself.

### Melody

So, playing notes and chords is fun - but how about a melody? What if you wanted to play one note after another and not at the same time? Well, that's easy, you just need to `sleep` between the notes:

```
play 72  
sleep 1  
play 75  
sleep 1  
play 79
```

How lovely, a little arpeggio. So what does the `1` mean in `sleep 1`? Well it means the *duration of the sleep*. It actually means sleep for one beat, but for now we can think about it as sleeping for 1 second. So, what if we wanted to make our arpeggio a little faster? Well, we need to use shorter sleep values. What about a half i.e. `0.5`:

```
play 72  
sleep 0.5  
play 75  
sleep 0.5  
play 79
```

Notice how it plays faster. Now, try for yourself, change the times - use different times and notes.

One thing to try is in-between notes such as `play 52.3` and `play 52.63`. There's absolutely no need to stick to standard whole notes. Play around and have fun.

### Traditional Note Names

For those of you that already know some musical notation (don't worry if you don't - you don't need it to have fun) you might want to write a melody using note names such as C and F# rather than numbers. Sonic Pi has you covered. You can do the following:

```
play :C  
sleep 0.5  
play :D  
sleep 0.5
```

play :E

Remember to put the colon **:** in front of your note name so that it goes pink. Also, you can specify the octave by adding a number after the note name:

```
play :C3
sleep 0.5
play :D3
sleep 0.5
play :E4
```

If you want to make a note sharp, add an **s** after the note name such as **play :Fs3** and if you want to make a note flat, add a **b** such as **play :Eb3**.

Now go *crazy* and have fun making your own tunes.

---

## 2.2 - Synth Options: Amp and Pan

As well as allowing you to control which note to play or which sample to trigger, Sonic Pi provides a whole range of options to craft and control the sounds. We'll be covering many of these in this tutorial and there's extensive documentation for each in the help system. However, for now we'll introduce two of the most useful: *amplitude* and *pan*. First, let's look at what options actually are.

### Options

Sonic Pi supports the notion of options (or opts for short) for its synths. Opts are controls you pass to `play` which modify and control aspects of the sound you hear. Each synth has its own set of opts for finely tuning its sound. However, there are common sets of opts shared by many sounds such as `amp:` and envelope opts (covered in another section).

Opts have two major parts, their name (the name of the control) and their value (the value you want to set the control at). For example, you might have a opt called `cheese:` and want to set it with a value of `1`.

Opts are passed to calls to `play` by using a comma `,` and then the name of the opt such as `amp:` (don't forget the colon `:`) and then a space and the value of the opt. For example:

```
play 50, cheese: 1
```

(Note that `cheese:` isn't a valid opt, we're just using it as an example).

You can pass multiple opts by separating them with a comma:

```
play 50, cheese: 1, beans: 0.5
```

The order of the opts doesn't matter, so the following is identical:

```
play 50, beans: 0.5, cheese: 1
```

Opts that aren't recognised by the synth are just ignored (like `cheese` and `beans` which are clearly ridiculous opt names!)

If you accidentally use the same opt twice with different values, the last one wins. For example, `beans:` here will have the value `2` rather than `0.5`:

```
play 50, beans: 0.5, cheese: 3, eggs: 0.1, beans: 2
```

Many things in Sonic Pi accept opts, so just spend a little time learning how to use them and you'll be set! Let's play with our first opt: `amp:`.

### Amplitude

Amplitude is a computer representation of the loudness of a sound. A *high amplitude produces a loud sound* and a *low amplitude produces a quiet sound*. Just as Sonic Pi uses numbers to represent time and notes, it uses numbers to represent amplitude. An amplitude of `0` is silent (you'll hear nothing) whereas an amplitude of `1` is normal volume. You can even crank up the amplitude higher to `2`, `10`, `100`. However, you should note that when the overall amplitude of all the sounds gets too high, Sonic Pi uses what's called a compressor to squash them all to make sure things aren't too loud for your ears. This can often make the sound muddy and strange. So try to use low amplitudes, i.e. in the range `0` to `0.5` to avoid compression.

### Amp it up

To change the amplitude of a sound, you can use the `amp:` opt. For example, to play at half amplitude pass `0.5`:

```
play 60, amp: 0.5
```

To play at double amplitude pass `2`:

```
play 60, amp: 2
```

The `amp:` opt only modifies the call to `play` it's associated with. So, in this example, the first call to play is at half volume and the second is back to the default (`1`):

```
play 60, amp: 0.5  
sleep 0.5  
play 65
```

Of course, you can use different `amp:` values for each call to play:

```
play 50, amp: 0.1  
sleep 0.25  
play 55, amp: 0.2  
sleep 0.25  
play 57, amp: 0.4  
sleep 0.25  
play 62, amp: 1
```

### Panning

---

Another fun opt to use is `pan`: which controls the panning of a sound in stereo. Panning a sound to the left means that you hear it out of the left speaker, and panning it to the right means you hear it out of your right speaker. For our values, we use a -1 to represent fully left, 0 to represent center and 1 to represent fully right in the stereo field. Of course, we're free to use any value between -1 and 1 to control the exact positioning of our sound.

Let's play a beep out of the left speaker:

```
play 60, pan: -1
```

Now, let's play it out of the right speaker:

```
play 60, pan: 1
```

Finally let's play it back out of the center of both (the default position):

```
play 60, pan: 0
```

Now, go and have fun changing the amplitude and panning of your sounds!

---

## 2.3 - Switching Synths

So far we've had quite a lot of fun making beeps. However, you're probably starting to get bored of the basic beep noise. Is that all Sonic Pi has to offer? Surely there's more to live coding than just playing beeps? Yes there is, and in this section we'll explore the exciting range of sounds that Sonic Pi has to offer.

### Synths

Sonic Pi has a range of instruments it calls synths which is *short for synthesisers*. Whereas samples represent pre-recorded sounds, synths are capable of generating new sounds depending on how you control them (which we'll explore later in this tutorial). Sonic Pi's synths are very powerful and expressive and you'll have a lot of fun exploring and playing with them. First, let's learn how to select the current synth to use.

### Buzzy saws and prophets

A fun sound is the *saw wave* - let's give it a try:

```
use_synth :saw
play 38
sleep 0.25
play 50
sleep 0.25
play 62
sleep 0.25
```

Let's try another sound - the *prophet*:

```
use_synth :prophet
play 38
sleep 0.25
play 50
sleep 0.25
play 62
sleep 0.25
```

How about combining two sounds. First one after another:

```
use_synth :saw
play 38
sleep 0.25
play 50
sleep 0.25
use_synth :prophet
play 57
sleep 0.25
```

Now at the same time:

```
use_synth :tb303
play 38
sleep 0.25
use_synth :dsaw
play 50
sleep 0.25
use_synth :prophet
play 57
sleep 0.25
```

Notice that the `use_synth` command only affects the following calls to `play`. Think of it like a *big switch* - new calls to `play` will play whatever synth it's currently pointing to. You can move the switch to a new synth with `use_synth`.

### Discovering Synths

To see which synths Sonic Pi has for you to play with take a look at the Synths option in the far left vertical menu (above Fx). There are over 20 to choose from. Here are a few of my favourites:

- `:prophet`
- `:dsaw`
- `:fm`
- `:tb303`
- `:pulse`

Now play around with *switching synths during your music*. Have fun combining synths to make new sounds as well as using different synths for different sections of your music.

---

## 2.4 - Duration with Envelopes

In an earlier section, we looked at how we can use the `sleep` command to control when to trigger our sounds. However, we haven't yet been able to control the duration of our sounds.

In order to give us a simple, yet powerful means of *controlling the duration* of our sounds, Sonic Pi provides the notion of an *ADSR amplitude envelope* (we'll cover what ADSR means later in this section). An amplitude envelope offers two useful aspects of control:

- control over the duration of a sound
- control over the amplitude of a sound

### Duration

The duration is the length the sound lasts for. A longer duration means that you hear the sound for longer. Sonic Pi's sounds all have a controllable amplitude envelope, and the total duration of that envelope is the duration of the sound. Therefore, by controlling the envelope you control the duration.

### Amplitude

The ADSR envelope not only controls duration, it also gives you *fine control over the amplitude of the sound*. All audible sounds start and end silent and contain some non-silent part in-between. Envelopes allow you to slide and hold the amplitude of non-silent parts of the sound. It's like giving someone instructions on how to turn up and down the volume of a guitar amplifier. For example you might ask someone to "start at silence, slowly move up to full volume, hold it for a bit, then quickly fall back to silence." Sonic Pi allows you to program exactly this behaviour with envelopes.

Just to recap, as we have seen before, an amplitude of 0 is silence and an amplitude of 1 is normal volume.

Now, let us look at each of the parts of the envelopes in turn.

### Release Phase

The only part of the envelope that's used by default is the release time. This is the time it takes for the synth's sound to fade out. All synths have a release time of 1 which means that by default they have a duration of 1 beat (which at the default BPM of 60 is 1 second):

```
play 70
```

The note will be audible for 1 second. Go ahead and time it :-). This is short hand for the longer more explicit version:

```
play 70, release: 1
```

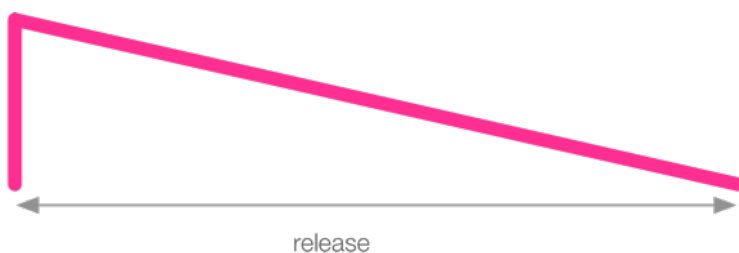
Notice how this sounds exactly the same (the sound lasts for one second). However, it's now very easy to change the duration by modifying the value of the `release:` opt:

```
play 60, release: 2
```

We can make the synth sound for a very short amount of time by using a very small release time:

```
play 60, release: 0.2
```

The duration of the release of the sound is called the *release phase* and by default is a linear transition (i.e. a straight line). The following diagram illustrates this transition:



The vertical line at the far left of the diagram shows that the sound starts at 0 amplitude, but goes up to full amplitude immediately (this is the attack phase which we'll cover next). Once at full amplitude it then moves in a straight line down to zero taking the amount of time specified by `release:`. *Longer release times produce longer synth fade outs.*

You can therefore change the duration of your sound by changing the release time. Have a play adding release times to your music.

### Attack Phase

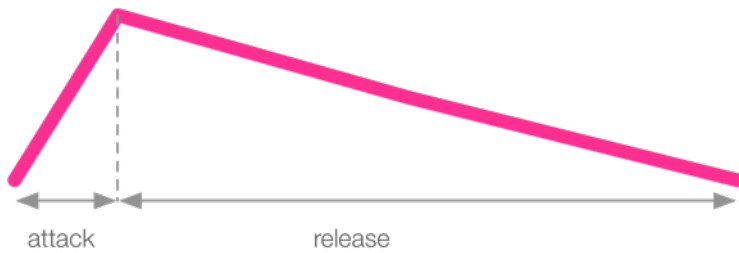
By default, the *attack phase* is 0 for all synths which means they move from 0 amplitude to 1 immediately. This gives the synth an initial percussive sound. However, you may wish to fade your sound in. This can be achieved with the `attack:` opt. Try fading in some sounds:

```
play 60, attack: 2
sleep 3
play 65, attack: 0.5
```

You may use multiple opts at the same time. For example for a short attack and a long release try:

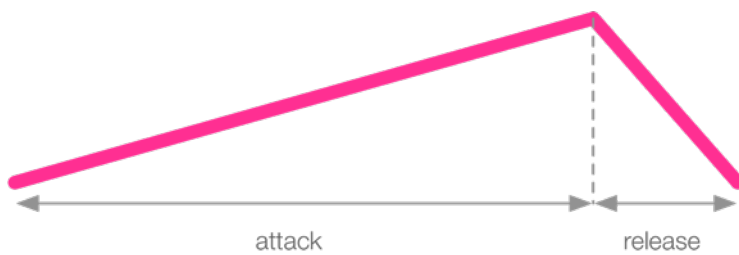
```
play 60, attack: 0.7, release: 4
```

This short attack and long release envelope is illustrated in the following diagram:



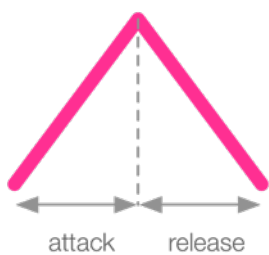
Of course, you may switch things around. Try a long attack and a short release:

```
play 60, attack: 4, release: 0.7
```



Finally, you can also have both short attack and release times for shorter sounds.

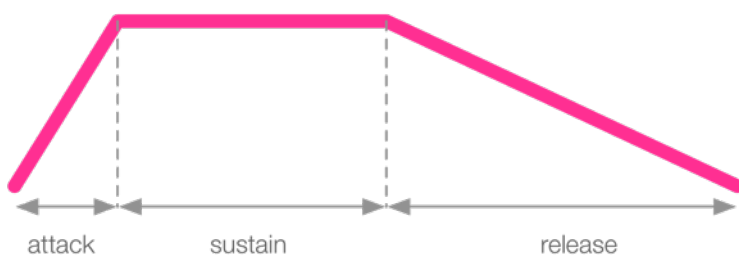
```
play 60, attack: 0.5, release: 0.5
```



## Sustain Phase

In addition to specifying attack and release times, you may also specify a sustain time to control the *sustain phase*. This is the time for which the sound is maintained at full amplitude between the attack and release phases.

```
play 60, attack: 0.3, sustain: 1, release: 1
```

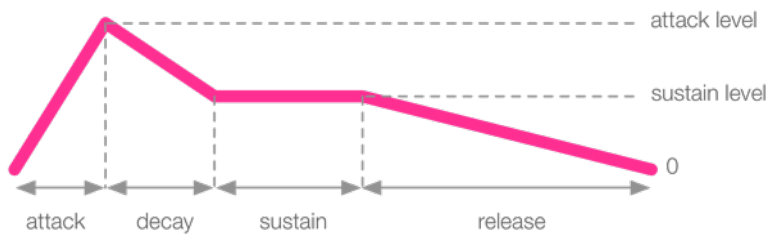


The sustain time is useful for important sounds you wish to give full presence in the mix before entering an optional release phase. Of course, it's totally valid to set both the `attack:` and `release:` opts to 0 and just use the sustain to have absolutely no fade in or fade out to the sound. However, be warned, a release of 0 can produce clicks in the audio and it's often better to use a very small value such as 0.2.

## Decay Phase

For an extra level of control, you can also specify a decay time. This is a phase of the envelope that fits between the attack and sustain phases and specifies a time where the amplitude will drop from the `attack_level:` to the `decay_level:` (which unless you explicitly set it will be set to the `sustain_level:`). By default, the `decay:` opt is 0 and both the attack and sustain levels are 1 so you'll need to specify them for the decay time to have any effect:

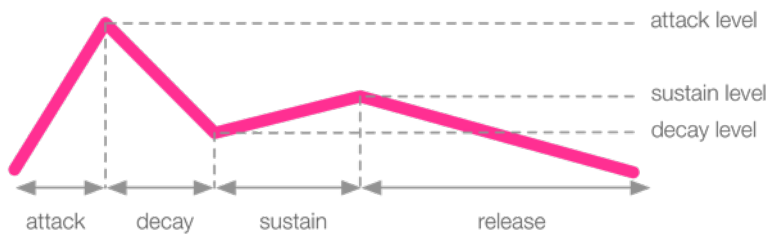
```
play 60, attack: 0.1, attack_level: 1, decay: 0.2, sustain_level: 0.4, sustain: 1, release: 0.5
```



## Decay Level

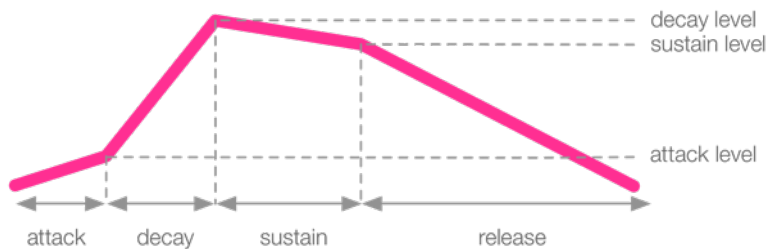
One last trick is that although the `decay_level` opt defaults to be the same value as `sustain_level`: you can explicitly set them to different values for full control over the envelope. This allows you to create envelopes such as the following:

```
play 60, attack: 0.1, attack_level: 1, decay: 0.2, decay_level: 0.3, sustain: 1, sustain_level: 0.4, re
```



It's also possible to set the `decay_level` to be higher than `sustain_level`:

```
play 60, attack: 0.1, attack_level: 0.1, decay: 0.2, decay_level: 1, sustain: 0.5, sustain_level: 0.8,
```



## ADSR Envelopes

So to summarise, Sonic Pi's ADSR envelopes have the following phases:

1. **attack** - time from 0 amplitude to the `attack_level`,
2. **decay** - time to move amplitude from `attack_level` to `decay_level`,
3. **sustain** - time to move the amplitude from `decay_level` to `sustain_level`,
4. **release** - time to move amplitude from `sustain_level` to 0

It's important to note that the duration of a sound is the summation of the times of each of these phases. Therefore the following sound will have a duration of  $0.5 + 1 + 2 + 0.5 = 4$  beats:

```
play 60, attack: 0.5, attack_level: 1, decay: 1, sustain_level: 0.4, sustain: 2, release: 0.5
```

Now go and have a play adding envelopes to your sounds...



## 3 - Samples

Another great way to develop your music is to use pre-recorded sounds. In great hip-hop tradition, we call these pre-recorded sounds *samples*. So, if you take a microphone outside, go and record the gentle sound of rain hitting canvas, you've just created a sample.

Sonic Pi lets you do lots of fun things with samples. Not only does it ship with over 90 public domain samples ready for you to jam with, it lets you play and manipulate your own. Let's get to it...

---

## 3.1 - Triggering Samples

Playing beeps is only the beginning. Something that's a lot of fun is triggering pre-recorded samples. Try it:

```
sample :ambi_lunar_land
```

Sonic Pi includes many samples for you to play with. You can use them just like you use the `play` command. To play multiple samples and notes just write them one after another:

```
play 36
play 48
sample :ambi_lunar_land
sample :ambi_drone
```

If you want to space them out in time, use the `sleep` command:

```
sample :ambi_lunar_land
sleep 1
play 48
sleep 0.5
play 36
sample :ambi_drone
sleep 1
play 36
```

Notice how Sonic Pi doesn't wait for a sound to finish before starting the next sound. The `sleep` command only describes the separation of the *triggering* of the sounds. This allows you to easily layer sounds together creating interesting overlap effects. Later in this tutorial we'll take a look at controlling the *duration* of sounds with envelopes.

## Discovering Samples

There are two ways to discover the range of samples provided in Sonic Pi. First, you can use this help system. Click on Samples in the far left vertical menu, choose your category and then you'll see a list of available sounds.

Alternatively you can use the *auto-completion system*. Simply type the start of a sample group such as: `sample :ambi_` and you'll see a drop-down of sample names appear for you to select. Try the following category prefixes:

- `:ambi_`
- `:bass_`
- `:elec_`
- `:perc_`
- `:guit_`
- `:drum_`
- `:misc_`
- `:bd_`

Now start mixing samples into your compositions!

---

## 3.2 - Sample Parameters: Amp and Pan

As we saw with synths, we can easily control our sounds with parameters. Samples support exactly the same parameterisation mechanism. Let's revisit our friends `amp:` and `pan:`.

### Amping samples

You can change the amplitude of samples with exactly the same approach you used for synths:

```
sample :ambi_lunar_land, amp: 0.5
```

### Panning samples

We're also able to use the `pan:` parameter on samples. For example, here's how we'd play the amen break in the left ear and then half way through play it again through the right ear:

```
sample :loop_amen, pan: -1  
sleep 0.877  
sample :loop_amen, pan: 1
```

Note that 0.877 is half the duration of the `:loop_amen` sample in seconds.

Finally, note that if you set some synth defaults with `use_synth_defaults` (which we will discuss later), these will be ignored by `sample`.

---

## 3.3 - Stretching Samples

Now that we can play a variety of synths and samples to create some music, it's time to learn how to modify both the synths and samples to make the music even more unique and interesting. First, let's explore the ability to *stretch* and *squash* samples.

### Sample Representation

Samples are pre-recorded sounds stored as numbers which represent how to move the speaker cone to reproduce the sound. The speaker cone can move in and out, and so the numbers just need to represent how far in and out the cone needs to be for each moment in time. To be able to faithfully reproduce a recorded sound the sample typically needs to store many thousands of numbers per second! Sonic Pi takes this list of numbers and feeds them at the right speed to move your computer's speaker in and out in just the right way to reproduce the sound. However, it's also fun to change the speed with which the numbers are fed to the speaker to change the sound.

### Changing Rate

Let's play with one of the ambient sounds: `:ambi_choir`. To play it with the default rate, you can pass a `rate:` opt to `sample:`

```
sample :ambi_choir, rate: 1
```

This plays it at normal rate (1), so nothing special yet. However, we're free to change that number to something else. How about `0.5`:

```
sample :ambi_choir, rate: 0.5
```

Woah! What's going on here? Well, two things. Firstly, the sample takes twice as long to play, secondly the sound is an octave lower. Let's explore these things in a little more detail.

### Let's stretch

A sample that's fun to stretch and compress is the Amen Break. At normal rate, we might imagine throwing it into a *drum 'n' bass* track:

```
sample :loop_amen
```

However by changing the rate we can switch up genres. Try half speed for *old school hip-hop*:

```
sample :loop_amen, rate: 0.5
```

If we speed it up, we enter *jungle* territory:

```
sample :loop_amen, rate: 1.5
```

Now for our final party trick - let's see what happens if we use a negative rate:

```
sample :loop_amen, rate: -1
```

Woah! It plays it *backwards*! Now try playing with lots of different samples at different rates. Try very fast rates. Try crazy slow rates. See what interesting sounds you can produce.

### A Simple Explanation of Sample Rate

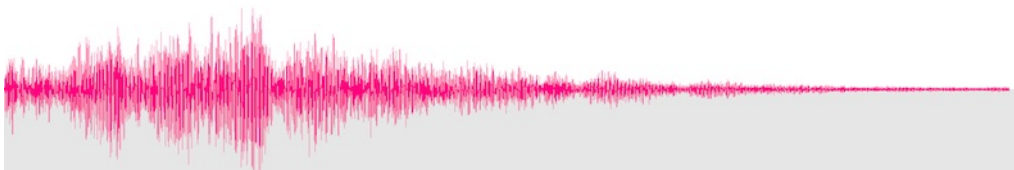
A useful way to think of samples is as springs. Playback rate is like squashing and stretching the spring. If you play the sample at rate 2, you're *squashing the spring* to half its normal length. The sample therefore takes half the amount of time to play as it's shorter. If you play the sample at half rate, you're *stretching the spring* to double its length. The sample therefore takes twice the amount of time to play as it's longer. The more you squash (higher rate), the shorter it gets, the more you stretch (lower rate), the longer it gets.

Compressing a spring increases its density (the number of coils per cm) - this is similar to the sample sounding *higher pitched*. Stretching the spring decreases its density and is similar to the sound having a *lower pitch*.

### The Maths Behind Sample Rate

(This section is provided for those that are interested in the details. Please feel free to skip it...)

As we saw above, a sample is represented by a big long list of numbers representing where the speaker should be through time. We can take this list of numbers and use it to draw a graph which would look similar to this:



You might have seen pictures like this before. It's called the *waveform* of a sample. It's just a graph of numbers. Typically a waveform like this will have 44100 points of data per second (this is due to the Nyquist-Shannon sampling theorem). So, if the sample lasts for 2 seconds, the waveform will be represented by 88200 numbers which we would feed to the speaker at a rate of 44100 points per second. Of course, we could feed it at double rate which would be 88200 points per second. This would therefore take only 1 second to play back. We could also play it back at half rate which would be 22050 points per second taking 4 seconds to play back.

The duration of the sample is affected by the playback rate:

- Doubling the playback rate halves the playback time,
- Halving the playback rate doubles the playback time,
- Using a playback rate of one fourth quadruples the playback time,
- Using a playback rate of 1/10 makes playback last 10 times longer.

We can represent this with the formula:

$$\text{new\_sample\_duration} = (1 / \text{rate}) * \text{sample\_duration}$$

Changing the playback rate also affects the pitch of the sample. The frequency or pitch of a waveform is determined by how fast it moves up and down. Our brains somehow turn fast movement of speakers into high notes and slow movement of speakers into low notes. This is why you can sometimes even see a big bass speaker move as it pumps out super low bass - it's actually moving a lot slower in and out than a speaker producing higher notes.

If you take a waveform and squash it it will move up and down more times per second. This will make it sound higher pitched. It turns out that doubling the amount of up and down movements (oscillations) doubles the frequency. So, *playing your sample at double rate will double the frequency you hear it*. Also, *halving the rate will halve the frequency*. Other rates will affect the frequency accordingly.

---

## 3.4 - Enveloped Samples

It is also possible to modify the *duration* and *amplitude* of a sample using an ADSR envelope. However, this works slightly differently to the ADSR envelope available on synths. Sample envelopes only allow you to reduce the amplitude and duration of a sample - and never to increase it. The sample will stop when either the sample has finished playing or the envelope has completed - whichever is first. So, if you use a very long **release:**, it won't extend the duration of the sample.

### Amen Envelopes

Let's return to our trusty friend the Amen Break:

```
sample :loop_amen
```

With no opts, we hear the full sample at full amplitude. If we want to fade this in over 1 second we can use the **attack:** param:

```
sample :loop_amen, attack: 1
```

For a shorter fade in, choose a shorter attack value:

```
sample :loop_amen, attack: 0.3
```

### Auto Sustain

Where the ADSR envelope's behaviour differs from the standard synth envelope is in the *sustain* value. In the standard synth envelope, the sustain defaulted to 0 unless you set it manually. With samples, the sustain value defaults to an *automagical* value - the time left to play the rest of the sample. This is why we hear the full sample when we pass no defaults. If the attack, decay, sustain and release values were all 0 we'd never hear a peep. Sonic Pi therefore calculates how long the sample is, deducts any attack, decay and release times and uses the result as your sustain time. If the attack, decay and release values add up to more than the duration of the sample, the sustain is simply set to 0.

### Fade Outs

To explore this, let's consider our Amen break in more detail. If we ask Sonic Pi how long the sample is:

```
print sample_duration :loop_amen
```

It will print out **1.753310657596372** which is the length of the sample in seconds. Let's just round that to **1.75** for convenience here. Now, if we set the release to **0.75**, something surprising will happen:

```
sample :loop_amen, release: 0.75
```

It will play the first second of the sample at full amplitude before then fading out over a period of 0.75 seconds. This is the *auto sustain* in action. By default, the release always works from the end of the sample. If our sample was 10.75 seconds long, it would play the first 10 seconds at full amplitude before fading out over 0.75s.

Remember: by default, **release:** fades out at the end of a sample.

### Fade In and Out

We can use both **attack:** and **release:** together with the auto sustain behaviour to fade both in and out over the duration of the sample:

```
sample :loop_amen, attack: 0.75, release: 0.75
```

As the full duration of the sample is 1.75s and our attack and release phases add up to 1.5s, the sustain is automatically set to 0.25s. This allows us to easily fade the sample in and out.

### Explicit sustain

We can easily get back to our normal synth ADSR behaviour by manually setting **sustain:** to a value such as 0:

```
sample :loop_amen, sustain: 0, release: 0.75
```

Now, our sample only plays for 0.75 seconds in total. With the default for **attack:** and **decay:** at 0, the sample jumps straight to full amplitude, sustains there for 0s then releases back down to 0 amplitude over the release period - 0.75s.

### Percussive cymbals

We can use this behaviour to good effect to turn longer sounding samples into shorter, more percussive versions. Consider the sample **:drum\_cymbal\_open**:

```
sample :drum_cymbal_open
```

You can hear the cymbal sound ringing out over a period of time. However, we can use our envelope to make it more percussive:

```
sample :drum_cymbal_open, attack: 0.01, sustain: 0, release: 0.1
```

You can then emulate hitting the cymbal and then dampening it by increasing the sustain period:

```
sample :drum_cymbal_open, attack: 0.01, sustain: 0.3, release: 0.1
```

Now go and have fun putting envelopes over the samples. Try changing the rate too for really interesting results.

---

## 3.5 - Partial Samples

This section will conclude our exploration of Sonic Pi's sample player. Let's do a quick recap. So far we've looked at how we can trigger samples:

```
sample :loop_amen
```

We then looked at how we can change the rate of samples such as playing them at half speed:

```
sample :loop_amen, rate: 0.5
```

Next, we looked at how we could fade a sample in (let's do it at half speed):

```
sample :loop_amen, rate: 0.5, attack: 1
```

We also looked at how we could use the start of a sample percussively by giving **sustain**: an explicit value and setting both the attack and release to be short values:

```
sample :loop_amen, rate: 2, attack: 0.01, sustain: 0, release: 0.35
```

However, wouldn't it be nice if we didn't have to always start at the beginning of the sample? Wouldn't it also be nice if we didn't have to always finish at the end of the sample?

### Choosing a starting point

It is possible to choose an arbitrary starting point in the sample as a value between 0 and 1 where 0 is the start of the sample, 1 is the end and 0.5 is half way through the sample. Let's try playing only the last half of the amen break:

```
sample :loop_amen, start: 0.5
```

How about the last quarter of the sample:

```
sample :loop_amen, start: 0.75
```

### Choosing a finish point

Similarly, it is possible to choose an arbitrary finish point in the sample as a value between 0 and 1. Let's finish the amen break half way through:

```
sample :loop_amen, finish: 0.5
```

### Specifying start and finish

Of course, we can combine these two to play arbitrary segments of the audio file. How about only a small section in the middle:

```
sample :loop_amen, start: 0.4, finish: 0.6
```

What happens if we choose a start position after the finish position?

```
sample :loop_amen, start: 0.6, finish: 0.4
```

Cool! It plays it backwards!

### Combining with rate

We can combine this new ability to play arbitrary segments of audio with our friend **rate**:. For example, we can play a very small section of the middle of the amen break very slowly:

```
sample :loop_amen, start: 0.5, finish: 0.7, rate: 0.2
```

### Combining with envelopes

Finally, we can combine all of this with our ADSR envelopes to produce interesting results:

```
sample :loop_amen, start: 0.5, finish: 0.8, rate: -0.2, attack: 0.3, release: 1
```

Now go and have a play mashing up samples with all of this fun stuff...

---



## 3.6 - External Samples

Whilst the built-in samples can get you up and started quickly, you might wish to experiment with other recorded sounds in your music. Sonic Pi totally supports this. First though, let's have a quick discussion on the portability of your piece.

### Portability

When you compose your piece purely with built-in synths and samples, the code is all you need to faithfully reproduce your music. Think about that for a moment - that's amazing! A simple piece of text you can email around or stick in a [Gist](#) represents everything you need to reproduce your sounds. That makes it *really easy to share* with your friends as they just need to get hold of the code.

However, if you start using your own pre-recorded samples, you lose this portability. This is because to reproduce your music other people not only need your code, they need your samples too. This limits the ability for others to manipulate, mash-up and experiment with your work. Of course this shouldn't stop you from using your own samples, it's just something to consider.

### Local Samples

So how do you play any arbitrary WAV or AIFF file on your computer? All you need to do is pass the path of that file to `sample`:

```
# Raspberry Pi, Mac, Linux
sample "/Users/sam/Desktop/my-sound.wav"
# Windows
sample "C:/Users/sam/Desktop/my-sound.wav"
```

Sonic Pi will automatically load and play the sample. You can also pass all the standard params you're used to passing `sample`:

```
# Raspberry Pi, Mac, Linux
sample "/Users/sam/Desktop/my-sound.wav", rate: 0.5, amp: 0.3
# Windows
sample "C:/Users/sam/Desktop/my-sound.wav", rate: 0.5, amp: 0.3
```

---

## 4 - Randomisation

A great way to add some interest into your music is using some random numbers. Sonic Pi has some great functionality for adding randomness to your music, but before we start we need to learn a shocking truth: in Sonic Pi *random is not truly random*. What on earth does this mean? Well, let's see.

### Repeatability

A really useful random function is `rrand` which will give you a random value between two numbers - a *min* and a *max*. (`rrand` is short for ranged random). Let's try playing a random note:

```
play rrand(50, 95)
```

Ooh, it played a random note. It played note `83.7527`. A nice random note between 50 and 95. Woah, wait, did I just predict the exact random note you got too? Something fishy is going on here. Try running the code again. What? It chose `83.7527` again? That can't be random!

The answer is that it is not truly random, it's pseudo-random. Sonic Pi will give you random-like numbers in a repeatable manner. This is very useful for ensuring that the music you create on your machine sounds identical on everybody else's machine - even if you use some randomness in your composition.

Of course, in a given piece of music, if it 'randomly' chose `83.7527` every time, then it wouldn't be very interesting. However, it doesn't. Try the following:

```
loop do
  play rrand(50, 95)
  sleep 0.5
end
```

Yes! It finally sounds random. Within a given *run* subsequent calls to random functions will return random values. However, the next run will produce exactly the same sequence of random values and sound exactly the same. It's as if all Sonic Pi code went back in time to exactly the same point every time the Run button was pressed. It's the Groundhog Day of music synthesis!

### Haunted Bells

A lovely illustration of randomisation in action is the haunted bells example which loops the `:perc_bell` sample with a random rate and sleep time between bell sounds:

```
loop do
  sample :perc_bell, rate: (rrand 0.125, 1.5)
  sleep rrand(0.2, 2)
end
```

### Random cutoff

Another fun example of randomisation is to modify the cutoff of a synth randomly. A great synth to try this out on is the `:tb303` emulator:

```
use_synth :tb303

loop do
  play 50, release: 0.1, cutoff: rrand(60, 120)
  sleep 0.125
end
```

### Random seeds

So, what if you don't like this particular sequence of random numbers Sonic Pi provides? Well it's totally possible to choose a different starting point via `use_random_seed`. The default seed happens to be 0, so choose a different seed for a different random experience!

Consider the following:

```
5.times do
  play rrand(50, 100)
  sleep 0.5
end
```

Every time you run this code, you'll hear the same sequence of 5 notes. To get a different sequence simply change the seed:

```
use_random_seed 40
5.times do
  play rrand(50, 100)
  sleep 0.5
end
```

This will produce a different sequence of 5 notes. By changing the seed and listening to the results you can find something that you like - and when you share it with others, they will hear exactly what you heard too.

Let's have a look at some other useful random functions.

## choose

A very common thing to do is to choose an item randomly from a list of known items. For example, I may want to play one note from the following: 60, 65 or 72. I can achieve this with `choose` which lets me choose an item from a list. First, I need to put my numbers in a list which is done by wrapping them in square brackets and separating them with commas: `[60, 65, 72]`. Next I just need to pass them to `choose`:

```
choose([60, 65, 72])
```

Let's hear what that sounds like:

```
loop do
  play choose([60, 65, 72])
  sleep 1
end
```

## rrand

We've already seen `rrand`, but let's run over it again. It returns a random number between two values exclusively. That means it will never return either the top or bottom number - always something in between the two. The number will always be a float - meaning it's not a whole number but a fraction of a number. Examples of floats returned by `rrand(20, 110)`:

- 87.5054931640625
- 86.05255126953125
- 61.77825927734375

## rrand\_i

Occasionally you'll want a whole random number, not a float. This is where `rrand_i` comes to the rescue. It works similarly to `rrand` except it may return the min and max values as potential random values (which means it's inclusive rather than exclusive of the range). Examples of numbers returned by `rrand_i(20, 110)` are:

- 88
- 86
- 62

## rand

This will return a random float between 0 (inclusive) and the max value you specify (exclusive). By default it will return a value between 0 and one. It's therefore useful for choosing random `amp` values:

```
loop do
  play 60, amp: rand
  sleep 0.25
end
```

## rand\_i

Similar to the relationship between `rrand_i` and `rrand`, `rand_i` will return a random whole number between 0 and the max value you specify.

## dice

Sometimes you want to emulate a dice throw - this is a special case of `rrand_i` where the lower value is always 1. A call to `dice` requires you to specify the number of sides on the dice. A standard dice has 6 sides, so `dice(6)` will act very similarly - returning values of either 1, 2, 3, 4, 5, or 6. However, just like fantasy role-play games, you might find value in a 4 sided dice, or a 12 sided dice, or a 20 sided dice - perhaps even a 120 sided dice!

## one\_in

Finally you may wish to emulate throwing the top score of a dice such as a 6 in a standard dice. `one_in` therefore returns true with a probability of one in the number of sides on the dice. Therefore `one_in(6)` will return true with a probability of 1 in 6 or false otherwise. True and false values are very useful for `if` statements which we will cover in a subsequent section of this tutorial.

Now, go and jumble up your code with some randomness!

---

## 5 - Programming Structures

Now that you've learned the basics of creating sounds with `play` and `sample` and creating simple melodies and rhythms by `sleeping` between sounds, you might be wondering what else the world of code can offer you...

Well, you're in for an exciting treat! It turns out that basic programming structures such as looping, conditionals, functions and threads give you amazingly powerful tools to express your musical ideas.

Let's get stuck in with the basics...

---

## 5.1 - Blocks

A structure you'll see a lot in Sonic Pi is the block. Blocks allow us to do useful things with large chunks of code. For example, with `synth` and `sample` parameters we were able to change something that happened on a single line. However, sometimes we want to do something meaningful to a number of lines of code. For example, we may wish to loop it, to add reverb to it, to only run it 1 time out of 5, etc. Consider the following code:

```
play 50
sleep 0.5
sample :elec_plip
sleep 0.5
play 62
```

To do something with a chunk of code, we need to tell Sonic Pi where the code block *starts* and where it *ends*. We use `do` for start and `end` for end. For example:

```
do
  play 50
  sleep 0.5
  sample :elec_plip
  sleep 0.5
  play 62
end
```

However, this isn't yet complete and won't work (try it and you'll get an error) as we haven't told Sonic Pi what we want to do with this *do/end block*. We tell Sonic Pi this by writing some special code before the `do`. We'll see a number of these special pieces of code later on in this tutorial. For now, it's important to know that wrapping your code within `do` and `end` tells Sonic Pi you wish to do something special with that chunk of code.

---

## 5.2 - Iteration and Loops

So far we've spent a lot of time looking at the different sounds you can make with `play` and `sample` blocks. We've also learned how to trigger these sounds through time using `sleep`.

As you've probably found out, there's a *lot* of fun you can have with these basic building blocks. However, a whole new dimension of fun opens up when you start using the power of code to structure your music and compositions. In the next few sections we'll explore some of these powerful new tools. First up is iteration and loops.

### Repetition

Have you written some code you'd like to repeat a few times? For example, you might have something like this:

```
play 50
sleep 0.5
sample :elec_blup
sleep 0.5
play 62
sleep 0.25
```

What if we wished to repeat this 3 times? Well, we could do something simple and just copy and paste it three times:

```
play 50
sleep 0.5
sample :elec_blup
sleep 0.5
play 62
sleep 0.25
```

```
play 50
sleep 0.5
sample :elec_blup
sleep 0.5
play 62
sleep 0.25
```

```
play 50
sleep 0.5
sample :elec_blup
sleep 0.5
play 62
sleep 0.25
```

Now that's a lot of code! What happens if you want to change the sample to `:elec_plip`? You're going to have to find all the places with the original `:elec_blup` and switch them over. More importantly, what if you wanted to repeat the original piece of code 50 times or 1000? Now that would be a lot of code, and a lot of lines of code to alter if you wanted to make a change.

### Iteration

In fact, repeating the code should be as easy as saying *do this three times*. Well, it pretty much is. Remember our old friend the code block? We can use it to mark the start and end of the code we'd like to repeat three times. We then use the special code `3.times`. So, instead of writing *do this three times*, we write `3.times do` - that's not too hard. Just remember to write `end` at the end of the code you'd like to repeat:

```
3.times do
  play 50
  sleep 0.5
  sample :elec_blup
  sleep 0.5
  play 62
  sleep 0.25
end
```

Now isn't that much neater than cutting and pasting! We can use this to create lots of nice repeating structures:

```
4.times do
  play 50
  sleep 0.5
end
```

```
8.times do
  play 55, release: 0.2
  sleep 0.25
end
```

```
4.times do
  play 50
  sleep 0.5
end
```

## Nesting Iterations

We can put iterations inside other iterations to create interesting patterns. For example:

```
4.times do
  sample :drum_heavy_kick
  2.times do
    sample :elec_blip2, rate: 2
    sleep 0.25
  end
  sample :elec_snare
  4.times do
    sample :drum_tom_mid_soft
    sleep 0.125
  end
end
```

## Looping

If you want something to repeat a lot of times, you might find yourself using really large numbers such as `1000.times do`. In this case, you're probably better off asking Sonic Pi to repeat forever (at least until you press the stop button!). Let's loop the amen break forever:

```
loop do
  sample :loop_amen
  sleep sample_duration :loop_amen
end
```

The important thing to know about loops is that they act like black holes for code. Once the code enters a loop it can never leave until you press stop - it will just go round and round the loop forever. This means if you have code after the loop you will *never* hear it. For example, the cymbal after this loop will never play:

```
loop do
  play 50
  sleep 1
end
```

```
sample :drum_cymbal_open
```

Now, get structuring your code with iteration and loops!

---

## 5.3 - Conditionals

A common thing you'll likely find yourself wanting to do is to not only play a random note (see the previous section on randomness) but also make a random decision and based on the outcome run some code or some other code. For example, you might want to randomly play a drum or a cymbal. We can achieve this with an `if` statement.

### Flipping a Coin

So, let's flip a coin: if it's heads, play a drum, if it's tails, play a cymbal. Easy. We can emulate a coin flip with our `one_in` function (introduced in the section on randomness) specifying a probability of 1 in 2: `one_in(2)`. We can then use the result of this to decide between two pieces of code, the code to play the drum and the code to play the cymbal:

```
loop do
  if one_in(2)
    sample :drum_heavy_kick
  else
    sample :drum_cymbal_closed
  end

  sleep 0.5
end
```

Notice that `if` statements have three parts:

- The question to ask
- The first choice of code to run (if the answer to the question is yes)
- The second choice of code to run (if the answer to the question is no)

Typically in programming languages, the notion of yes is represented by the term `true` and the notion of no is represented by the term `false`. So we need to find a question that will give us a `true` or `false` answer which is exactly what `one_in` does.

Notice how the first choice is wrapped between the `if` and the `else` and the second choice is wrapped between the `else` and the `end`. Just like `do/end` blocks you can put multiple lines of code in either place. For example:

```
loop do
  if one_in(2)
    sample :drum_heavy_kick
    sleep 0.5
  else
    sample :drum_cymbal_closed
    sleep 0.25
  end
end
```

This time we're sleeping for a different amount of time depending on which choice we make.

### Simple if

Sometimes you want to optionally execute just one line of code. This is possible by placing `if` and then the question at the end. For example:

```
use_synth :dsaw

loop do
  play 50, amp: 0.3, release: 2
  play 53, amp: 0.3, release: 2 if one_in(2)
  play 57, amp: 0.3, release: 2 if one_in(3)
  play 60, amp: 0.3, release: 2 if one_in(4)
  sleep 1.5
end
```

This will play chords of different numbers with the chance of each note playing having a different probability.

---



## 5.4 - Threads

So you've made your killer bassline and a phat beat. How do you play them at the same time? One solution is to weave them together manually - play some bass, then a bit of drums, then more bass... However, the timing soon gets hard to think about, especially when you start weaving in more elements.

What if Sonic Pi could weave things for you automatically? Well, it can, and you do it with a special thing called a *thread*.

### Infinite Loops

To keep this example simple, you'll have to imagine that this is a phat beat and a killer bassline:

```
loop do
  sample :drum_heavy_kick
  sleep 1
end
```

```
loop do
  use_synth :fm
  play 40, release: 0.2
  sleep 0.5
end
```

As we've discussed previously, loops are like *black holes* for the program. Once you enter a loop you can never exit from it until you hit stop. How do we play both loops at the same time? We have to tell Sonic Pi that we want to start something at the same time as the rest of the code. This is where threads come to the rescue.

### Threads to the Rescue

```
in_thread do
  loop do
    sample :drum_heavy_kick
    sleep 1
  end
end
```

```
loop do
  use_synth :fm
  play 40, release: 0.2
  sleep 0.5
end
```

By wrapping the first loop in an *in\_thread* do/end block we tell Sonic Pi to run the contents of the do/end block at *exactly* the same time as the next statement after the do/end block (which happens to be the second loop). Try it and you'll hear both the drums and the bassline weaved together!

Now, what if we wanted to add a synth on top. Something like:

```
in_thread do
  loop do
    sample :drum_heavy_kick
    sleep 1
  end
end
```

```
loop do
  use_synth :fm
  play 40, release: 0.2
  sleep 0.5
end
```

```
loop do
  use_synth :zawa
  play 52, release: 2.5, phase: 2, amp: 0.5
  sleep 2
end
```

Now we have the same problem as before. The first loop is played at the same time as the second loop due to the *in\_thread*. However, *the third loop is never reached*. We therefore need another thread:

```
in_thread do
  loop do
    sample :drum_heavy_kick
    sleep 1
  end
```

```

end

in_thread do
  loop do
    use_synth :fm
    play 40, release: 0.2
    sleep 0.5
  end
end

loop do
  use_synth :zawa
  play 52, release: 2.5, phase: 2, amp: 0.5
  sleep 2
end

```

## Runs as threads

What may surprise you is that when you press the Run button, you're actually creating a new thread for the code to run. This is why pressing it multiple times will layer sounds over each other. As the runs themselves are threads, they will automatically weave the sounds together for you.

## Scope

As you learn how to master Sonic Pi, you'll learn that threads are the most important building blocks for your music. One of the important jobs they have is to isolate the notion of *current settings* from other threads. What does this mean? Well, when you switch synths using `use_synth` you're actually just switching the synth in the *current thread* - no other thread will have their synth switched. Let's see this in action:

```

play 50
sleep 1

in_thread do
  use_synth :tb303
  play 50
end

sleep 1
play 50

```

Notice how the middle sound was different to the others? The `use_synth` statement only affected the thread it was in and not the outer main run thread.

## Inheritance

When you create a new thread with `in_thread`, the new thread will automatically inherit all of the current settings from the current thread. Let's see that:

```

use_synth :tb303
play 50
sleep 1

in_thread do
  play 55
end

```

Notice how the second note is played with the `:tb303` synth even though it was played from a separate thread? Any of the settings modified with the various `use_*` functions will behave in the same way.

When threads are created, they inherit all the settings from their parent but they don't share any changes back.

## Naming Threads

Finally, we can give our threads names:

```

in_thread(name: :bass) do
  loop do
    use_synth :prophet
    play chord(:e2, :m7).choose, release: 0.6
    sleep 0.5
  end
end

in_thread(name: :drums) do
  loop do

```

```
    sample :elec_snare
  sleep 1
end
end
```

Look at the log pane when you run this code. See how the log reports the name of the thread with the message?

```
[Run 36, Time 4.0, Thread :bass]
|- synth :prophet, {release: 0.6, note: 47}
```

## Only One Thread per Name Allowed

One last thing to know about named threads is that only one thread of a given name may be running at the same time. Let's explore this. Consider the following code:

```
in_thread do
  loop do
    sample :loop_amen
    sleep sample_duration :loop_amen
  end
end
```

Go ahead and paste that into a buffer and press the Run button. Press it again a couple of times. Listen to the cacophony of multiple amen breaks looping out of time with each other. Ok, you can press Stop now.

This is the behaviour we've seen again and again - if you press the Run button, sound layers on top of any existing sound. Therefore if you have a loop and press the Run button three times, you'll have three layers of loops playing simultaneously.

However, with named threads it is different:

```
in_thread(name: :amen) do
  loop do
    sample :loop_amen
    sleep sample_duration :loop_amen
  end
end
```

Try pressing the Run button multiple times with this code. You'll only ever hear one amen break loop. You'll also see this in the log:

```
==> Skipping thread creation: thread with name :amen already exists.
```

Sonic Pi is telling you that a thread with the name `:amen` is already playing, so it's not creating another.

This behaviour may not seem immediately useful to you now - but it will be very handy when we start to live code...

---

## 5.5 - Functions

Once you start writing lots of code, you may wish to find a way to organise and structure things to make them tidier and easier to understand. Functions are a very powerful way to do this. They give us the ability to give a name to a bunch of code. Let's take a look.

### Defining functions

```
define :foo do
  play 50
  sleep 1
  play 55
  sleep 2
end
```

Here, we've defined a new function called `foo`. We do this with our old friend the `do/end` block and the magic word `define` followed by the name we wish to give to our function. We didn't have to call it `foo`, we could have called it anything we want such as `bar`, `baz` or ideally something meaningful to you like `main_section` or `lead_riff`.

Remember to prepend a colon `:` to the name of your function when you define it.

### Calling functions

Once we have defined our function we can call it by just writing its name:

```
define :foo do
  play 50
  sleep 1
  play 55
  sleep 0.5
end

foo

sleep 1

2.times do
  foo
end
```

We can even use `foo` inside iteration blocks or anywhere we may have written `play` or `sample`. This gives us a great way to express ourselves and to create new meaningful words for use in our compositions.

### Functions are remembered across runs

So far, every time you've pressed the Run button, Sonic Pi has started from a completely blank slate. It knows nothing except for what is in the buffer. You can't reference code in another buffer or another thread. However, functions change that. When you define a function, Sonic Pi *remembers* it. Let's try it. Delete all the code in your buffer and replace it with:

```
foo
```

Press the Run button - and hear your function play. Where did the code go? How did Sonic Pi know what to play? Sonic Pi just remembered your function - so even after you deleted it from the buffer, it remembered what you had typed. This behaviour only works with functions created using `define` (and `defonce`).

### Parameterised functions

You might be interested in knowing that just like you can pass min and max values to `rrand`, you can teach your functions to accept arguments. Let's take a look:

```
define :my_player do |n|
  play n
end

my_player 80
sleep 0.5
my_player 90
```

This isn't very exciting, but it illustrates the point. We've created our own version of `play` called `my_player` which is parameterised.

The parameters need to go after the `do` of the `define` `do/end` block, surrounded by vertical goalposts `|` and separated by commas `,`. You may use any words you want for the parameter names.

The magic happens inside the `define` `do/end` block. You may use the parameter names as if they were real values. In this example I'm playing note `n`. You can consider the parameters as a kind of promise that when the code runs, they will be replaced with actual values. You do this by passing a parameter to the function when you call it. I do this with `my_player 80` to play note 80. Inside the function

definition, `n` is now replaced with 80, so `play n` turns into `play 80`. When I call it again with `my_player 90`, `n` is now replaced with 90, so `play n` turns into `play 90`.

Let's see a more interesting example:

```
define :chord_player do |root, repeats|
  repeats.times do
    play chord(root, :minor), release: 0.3
    sleep 0.5
  end
end
```

```
chord_player :e3, 2
sleep 0.5
chord_player :a3, 3
chord_player :g3, 4
sleep 0.5
chord_player :e3, 3
```

Here I used `repeats` as if it was a number in the line `repeats.times do`. I also used `root` as if it was a note name in my call to `play`.

See how we're able to write something very expressive and easy to read by moving a lot of our logic into a function!

---

## 5.6 - Variables

A useful thing to do in your code is to create names for things. Sonic Pi makes this very easy, you write the name you wish to use, an equal sign (=), then the thing you want to remember:

```
sample_name = :loop_amen
```

Here, we've 'remembered' the symbol `:loop_amen` in the variable `sample_name`. We can now use `sample_name` everywhere we might have used `:loop_amen`. For example:

```
sample_name = :loop_amen
sample sample_name
```

There are three main reasons for using variables in Sonic Pi: communicating meaning, managing repetition and capturing the results of things.

### Communicating Meaning

When you write code it's easy to just think you're telling the computer how to do stuff - as long as the computer understands it's OK. However, it's important to remember that it's not just the computer that reads the code. Other people may read it too and try to understand what's going on. Also, you're likely to read your own code in the future and try to understand what's going on. Although it might seem obvious to you now - it might not be so obvious to others or even your future self!

One way to help others understand what your code is doing is to write comments (as we saw in a previous section). Another is to use meaningful variable names. Look at this code:

```
sleep 1.7533
```

Why does it use the number `1.7533`? Where did this number come from? What does it mean? However, look at this code:

```
loop_amen_duration = 1.7533
sleep loop_amen_duration
```

Now, it's much clearer what `1.7533` means: it's the duration of the sample `:loop_amen`! Of course, you might say why not simply write:

```
sleep sample_duration(:loop_amen)
```

Which, of course, is a very nice way of communicating the intent of the code.

### Managing Repetition

Often you see a lot of repetition in your code and when you want to change things, you have to change it in a lot of places. Take a look at this code:

```
sample :loop_amen
sleep sample_duration(:loop_amen)
sample :loop_amen, rate: 0.5
sleep sample_duration(:loop_amen, rate: 0.5)
sample :loop_amen
sleep sample_duration(:loop_amen)
```

We're doing a lot of things with `:loop_amen`! What if we wanted to hear what it sounded like with another loop sample such as `:loop_garzul`? We'd have to find and replace all `:loop_amens` with `:loop_garzul`. That might be fine if you have lots of time - but what if you're performing on stage? Sometimes you don't have the luxury of time - especially if you want to keep people dancing.

What if you'd written your code like this:

```
sample_name = :loop_amen
sample sample_name
sleep sample_duration(sample_name)
sample sample_name, rate: 0.5
sleep sample_duration(sample_name, rate: 0.5)
sample sample_name
sleep sample_duration(sample_name)
```

Now, that does exactly the same as above (try it). It also gives us the ability to just change one line `sample_name = :loop_amen` to `sample_name = :loop_garzul` and we change it in many places through the magic of variables.

### Capturing Results

Finally, a good motivation for using variables is to capture the results of things. For example, you may wish to do things with the duration of a sample:

```
sd = sample_duration(:loop_amen)
```

We can now use `sd` anywhere we need the duration of the `:loop_amen` sample.

Perhaps more importantly, a variable allows us to capture the result of a call to `play` or `sample`:

```
s = play 50, release: 8
```

Now we have caught and remembered `s` as a variable, which allows us to control the synth as it is running:

```
s = play 50, release: 8  
sleep 2  
control s, note: 62
```

We'll look into controlling synths in more detail in a later section.

---

## 5.7 - Thread Synchronisation

Once you have become sufficiently advanced live coding with a number of functions and threads simultaneously, you've probably noticed that it's pretty easy to make a mistake in one of the threads which kills it. That's no big deal, because you can easily restart the thread by hitting Run. However, when you restart the thread it is now *out of time* with the original threads.

### Inherited Time

As we discussed earlier, new threads created with `in_thread` inherit all of the settings from the parent thread. This includes the current time. This means that threads are always in time with each other when started simultaneously.

However, when you start a thread on its own it starts with its own time which is unlikely to be in sync with any of the other currently running threads.

### Cue and Sync

Sonic Pi provides a solution to this problem with the functions `cue` and `sync`.

`cue` allows us to send out heartbeat messages to all other threads. By default the other threads aren't interested and ignore these heartbeat messages. However, you can easily register interest with the `sync` function.

The important thing to be aware of is that `sync` is similar to `sleep` in that it stops the current thread from doing anything for a period of time. However, with `sleep` you specify how long you want to wait while with `sync` you don't know how long you will wait - as `sync` waits for the next `cue` from another thread which may be soon or a long time away.

Let's explore this in a little more detail:

```
in_thread do
  loop do
    cue :tick
    sleep 1
  end
end
```

```
in_thread do
  loop do
    sync :tick
    sample :drum_heavy_kick
  end
end
```

Here we have two threads - one acting like a metronome, not playing any sounds but sending out `:tick` heartbeat messages every beat. The second thread is synchronising on `tick` messages and when it receives one it inherits the time of the `cue` thread and continues running.

As a result, we will hear the `:drum_heavy_kick` sample exactly when the other thread sends the `:tick` message, even if the two threads didn't start their execution at the same time:

```
in_thread do
  loop do
    cue :tick
    sleep 1
  end
end
```

```
sleep(0.3)
```

```
in_thread do
  loop do
    sync :tick
    sample :drum_heavy_kick
  end
end
```

That naughty `sleep` call would typically make the second thread out of phase with the first. However, as we're using `cue` and `sync`, we automatically sync the threads bypassing any accidental timing offsets.

### Cue Names

You are free to use whatever name you'd like for your `cue` messages - not just `:tick`. You just need to ensure that any other threads are `syncing` on the correct name - otherwise they'll be waiting for ever (or at least until you press the Stop button).

Let's play with a few `cue` names:

```
in_thread do
```



```
loop do
  cue [ :foo, :bar, :baz ].choose
  sleep 0.5
end
end

in_thread do
  loop do
    sync :foo
    sample :elec_beep
  end
end

in_thread do
  loop do
    sync :bar
    sample :elec_flip
  end
end

in_thread do
  loop do
    sync :baz
    sample :elec_blup
  end
end
```

Here we have a main **cue** loop which is randomly sending one of the heartbeat names **:foo**, **:bar** or **:baz**. We then also have three loop threads syncing on each of those names independently and then playing a different sample. The net effect is that we hear a sound every 0.5 beats as each of the **sync** threads is randomly synced with the **cue** thread and plays its sample.

This of course also works if you order the threads in reverse as the **sync** threads will simply sit and wait for the next **cue**.

---

## 6 - Studio FX

One of the most rewarding and fun aspects of Sonic Pi is the ability to easily add studio effects to your sounds. For example, you may wish to add some reverb to parts of your piece, or some echo or perhaps even distort or wobble your basslines.

Sonic Pi provides a very simple yet powerful way of adding FX. It even allows you to chain them (so you can pass your sounds through distortion, then echo and then reverb) and also control each individual FX unit with opts (in a similar way to giving params to synths and samples). You can even modify the opts of the FX whilst it's still running. So, for example, you could increase the reverb on your bass throughout the track...

### Guitar Pedals

If all of this sounds a bit complicated, don't worry. Once you play around with it a little, it will all become quite clear. Before you do though, a simple analogy is that of guitar FX pedals. There are many kinds of FX pedals you can buy. Some add reverb, others distort etc. A guitarist will plug his or her guitar into one FX pedal - i.e. distortion -, then take another cable and connect (chain) a reverb pedal. The output of the reverb pedal can then be plugged into the amplifier:

Guitar -> Distortion -> Reverb -> Amplifier

This is called FX chaining. Sonic Pi supports exactly this. Additionally, each pedal often has dials and sliders to allow you to control how much distortion, reverb, echo etc. to apply. Sonic Pi also supports this kind of control. Finally, you can imagine a guitarist playing whilst someone plays with the FX controls whilst they're playing. Sonic Pi also supports this - but instead of needing someone else to control things for you, that's where the computer steps in.

Let's explore FX!

---

## 6.1 - Adding FX

In this section we'll look at a couple of FX: reverb and echo. We'll see how to use them, how to control their opts and how to chain them.

Sonic Pi's FX system uses blocks. So if you haven't read section 5.1 you might want to take a quick look and then head back.

### Reverb

If we want to use reverb we write `with_fx :reverb` as the special code to our block like this:

```
with_fx :reverb do
  play 50
  sleep 0.5
  sample :elec_plip
  sleep 0.5
  play 62
end
```

Now play this code and you'll hear it played with reverb. It sounds good, doesn't it! Everything sounds pretty nice with reverb.

Now let's look what happens if we have code outside the do/end block:

```
with_fx :reverb do
  play 50
  sleep 0.5
  sample :elec_plip
  sleep 0.5
  play 62
end
```

```
sleep 1
play 55
```

Notice how the final `play 55` isn't played with reverb. This is because it is *outside* the do/end block, so it isn't captured by the reverb FX.

Similarly, if you make sounds before the do/end block, they also won't be captured:

```
play 55
sleep 1
```

```
with_fx :reverb do
  play 50
  sleep 0.5
  sample :elec_plip
  sleep 0.5
  play 62
end
```

```
sleep 1
play 55
```

### Echo

There are many FX to choose from. How about some echo?

```
with_fx :echo do
  play 50
  sleep 0.5
  sample :elec_plip
  sleep 0.5
  play 62
end
```

One of the powerful aspects of Sonic Pi's FX blocks is that they may be passed parameters similar to parameters we've already seen with `play` and `sample`. For example a fun echo parameter to play with is `phase`: which represents the duration of a given echo in beats. Let's make the echo slower:

```
with_fx :echo, phase: 0.5 do
  play 50
  sleep 0.5
  sample :elec_plip
  sleep 0.5
  play 62
end
```

Let's also make the echo faster:

```
with_fx :echo, phase: 0.125 do
  play 50
  sleep 0.5
  sample :elec_plip
  sleep 0.5
  play 62
end
```

Let's make the echo take longer to fade away by setting the `decay:` time to 8 beats:

```
with_fx :echo, phase: 0.5, decay: 8 do
  play 50
  sleep 0.5
  sample :elec_plip
  sleep 0.5
  play 62
end
```

## Nesting FX

One of the most powerful aspects of the FX blocks is that you can nest them. This allows you to very easily chain FX together. For example, what if you wanted to play some code with echo and then with reverb? Easy, just put one inside the other:

```
with_fx :reverb do
  with_fx :echo, phase: 0.5, decay: 8 do
    play 50
    sleep 0.5
    sample :elec_blup
    sleep 0.5
    play 62
  end
end
```

Think about the audio flowing from the inside out. The sound of all the code within the inner do/end block such as `play 50` is first sent to the echo FX and the sound of the echo FX is in turn sent out to the reverb FX.

We may use very deep nestings for crazy results. However, be warned, the FX can use a lot of resources and when you nest them you're effectively running multiple FX simultaneously. So be sparing with your use of FX especially on low powered platforms such as the Raspberry Pi.

## Discovering FX

Sonic Pi ships with a large number of FX for you to play with. To find out which ones are available, click on FX in the far left of this help system and you'll see a list of available options. Here's a list of some of my favourites:

- wobble,
- reverb,
- echo,
- distortion,
- slicer

Now go crazy and add FX everywhere for some amazing new sounds!

---

## 6.2 - FX in Practice

Although they look deceptively simple on the outside, FX are actually quite complex beasts internally. Their simplicity often entices people to overuse them in their pieces. This may be fine if you have a powerful machine, but if - like me - you use a Raspberry Pi to jam with, you need to be careful about how much work you ask it to do if you want to ensure the beats keep flowing.

Consider this code:

```
loop do
  with_fx :reverb do
    play 60, release: 0.1
    sleep 0.125
  end
end
```

In this code we're playing note 60 with a very short release time, so it's a short note. We also want reverb so we've wrapped it in a reverb block. All good so far. Except...

Let's look at what the code does. First we have a `loop` which means everything inside of it is repeated forever. Next we have a `with_fx` block. This means we will create a new reverb FX *every time we loop*. This is like having a separate FX reverb pedal for every time you pluck a string on a guitar. It's cool that you can do this, but it's not always what you want. For example, this code will struggle to run nicely on a Raspberry Pi. All the work of creating the reverb and then waiting until it needs to be stopped and removed is all handled by `with_fx` for you, but this takes CPU power which may be precious.

How do we make it more similar to a traditional setup where our guitarist has just *one* reverb pedal which all sounds pass through? Simple:

```
with_fx :reverb do
  loop do
    play 60, release: 0.1
    sleep 0.125
  end
end
```

We put our loop *inside* the `with_fx` block. This way we only create a single reverb for all notes played in our loop. This code is a lot more efficient and would work fine on a Raspberry Pi.

A compromise is to use `with_fx` over an iteration within a loop:

```
loop do
  with_fx :reverb do
    16.times do
      play 60, release: 0.1
      sleep 0.125
    end
  end
end
```

This way we've lifted the `with_fx` out of the inner part of the `loop` and we're now creating a new reverb every 16 notes.

Remember, there are no mistakes, just possibilities. However, each of these approaches will have a different sound and also different performance characteristics. So play around and use the approach that sounds best to you whilst also working within the performance constraints of your platform.

---

## 7 - Controlling running sounds

So far we've looked at how you can trigger synths and samples, and also how to change their default opts such as amplitude, pan, envelope settings and more. Each sound triggered is essentially its own sound with its own list of options set for the duration of the sound.

Wouldn't it also be cool if you could change a sound's opts whilst it's still playing, just like you might bend a string of a guitar whilst it's still vibrating?

You're in luck - this section will show you how to do exactly this.

---

## 7.1 - Controlling Running Synths

So far we've only concerned ourselves with triggering new sounds and FX. However, Sonic Pi gives us the ability to manipulate and control currently running sounds. We do this by using a variable to capture a reference to a synth:

```
s = play 60, release: 5
```

Here, we have a run-local variable `s` which represents the synth playing note 60. Note that this is *run-local* - you can't access it from other runs like functions.

Once we have `s`, we can start controlling it via the `control` function:

```
s = play 60, release: 5
sleep 0.5
control s, note: 65
sleep 0.5
control s, note: 67
sleep 3
control s, note: 72
```

The thing to notice is that we're not triggering 4 different synths here - we're just triggering one synth and then change the pitch 3 times afterwards, while it's playing.

We can pass any of the standard opts to `control`, so you can control things like `amp:`, `cutoff:` or `pan:`.

### Non-controllable Options

Some of the opts can't be controlled once the synth has started. This is the case for all the ADSR envelope parameters. You can find out which opts are controllable by looking at their documentation in the help system. If the documentation says *Can not be changed once set*, you know it's not possible to control the opt after the synth has started.

---

## 7.2 - Controlling FX

It is also possible to control FX, although this is achieved in a slightly different way:

```
with_fx :reverb do |r|
  play 50
  sleep 0.5
  control r, mix: 0.7
  play 55
  sleep 1
  control r, mix: 0.9
  sleep 1
  play 62
end
```

Instead of using a variable, we use the goalpost parameters of the do/end block. Inside the `|` bars, we need to specify a unique name for our running FX which we then reference from the containing do/end block. This behaviour is identical to using parameterised functions.

Now go and control some synths and FX!

---



## 7.3 - Sliding Opts

Whilst exploring the synth and FX opts, you might have noticed that there are a number of opts ending with `_slide`. You might have even tried calling them and seeing no effect. This is because they're not normal parameters, they're special opts that only work when you control synths as introduced in the previous section.

Consider the following example:

```
s = play 60, release: 5
sleep 0.5
control s, note: 65
sleep 0.5
control s, note: 67
sleep 3
control s, note: 72
```

Here, you can hear the synth pitch changing immediately on each `control` call. However, we might want the pitch to slide between changes. As we're controlling the `note:` parameter, to add slide, we need to set the `note_slide` parameter of the synth:

```
s = play 60, release: 5, note_slide: 1
sleep 0.5
control s, note: 65
sleep 0.5
control s, note: 67
sleep 3
control s, note: 72
```

Now we hear the notes being bent between the `control` calls. It sounds nice, doesn't it? You can speed up the slide by using a shorter time such as `note_slide: 0.2` or slow it down by using a longer slide time.

Every parameter that can be controlled has a corresponding `_slide` parameter for you to play with.

### Sliding is sticky

Once you've set a `_slide` parameter on a running synth, it will be remembered and used every time you slide the corresponding parameter. To stop sliding you must set the `_slide` value to 0 before the next `control` call.

### Sliding FX Opts

It is also possible to slide FX opts:

```
with_fx :wobble, phase: 1, phase_slide: 5 do |e|
  use_synth :dsaw
  play 50, release: 5
  control e, phase: 0.025
end
```

Now have fun sliding things around for smooth transitions and flowing control...

---

## 8 - Data Structures

A very useful tool in a programmer's toolkit is a data structure.

Sometimes you may wish to represent and use more than one thing. For example, you may find it useful to have a series of notes to play one after another. Programming languages have data structures to allow you to do exactly this.

There are many exciting and exotic data structures available to programmers - and people are always inventing new ones. However, for now we only really need to consider a very simple data structure - the list.

Let's look at it in more detail. We'll cover its basic form and then also how lists can be used to represent scales and chords.

---

## 8.1 - Lists

In this section we'll take a look at a data structure which is very useful - the list. We met it very briefly before in the section on randomisation when we randomly chose from a list of notes to play:

```
play choose([50, 55, 62])
```

In this section we'll explore using lists to also represent chords and scales. First let's recap how we might play a chord. Remember that if we don't use `sleep`, sounds all happen at the same time:

```
play 52
play 55
play 59
```

Let's look at other ways to represent this code.

### Playing a List

One option is to place all the notes in a list: `[52, 55, 59]`. Our friendly `play` function is smart enough to know how to play a list of notes. Try it:

```
play [52, 55, 59]
```

Ooh, that's already nicer to read. Playing a list of notes doesn't stop you from using any of the parameters as normal:

```
play [52, 55, 59], amp: 0.3
```

Of course, you can also use the traditional note names instead of the MIDI numbers:

```
play [:E3, :G3, :B3]
```

Now those of you lucky enough to have studied some music theory might recognise that chord as *E Minor* played in the 3rd octave.

### Accessing a List

Another very useful feature of a list is the ability to get information out of it. This may sound a bit strange, but it's no more complicated than someone asking you to turn a book to page 23. With a list, you'd say, what's the element at index 23? The only strange thing is that in programming indexes usually start at 0 not 1.

With list indexes we don't count 1, 2, 3... Instead we count 0, 1, 2...

Let's look at this in a little more detail. Take a look at this list:

```
[52, 55, 59]
```

There's nothing especially scary about this. Now, what's the second element in that list? Yes, of course, it's `55`. That was easy. Let's see if we can get the computer to answer it for us too:

```
puts [52, 55, 59][1]
```

OK, that looks a bit weird if you've never seen anything like it before. Trust me though, it's not too hard. There are three parts to the line above: the word `puts`, our list `52, 55, 59` and our index `[1]`. Firstly we're saying `puts` because we want Sonic Pi to print the answer out for us in the log. Next, we're giving it our list, and finally our index is asking for the second element. We need to surround our index with square brackets and because counting starts at `0`, the index for the second element is `1`. Look:

```
# indexes:  0  1  2
           [52, 55, 59]
```

Try running the code `puts [52, 55, 59][1]` and you'll see `55` pop up in the log. Change the index `1` to other indexes, try longer lists and think about how you might use a list in your next code jam. For example, what musical structures might be represented as a series of numbers...

---

## 8.2 - Chords

Sonic Pi has built-in support for chord names which will return lists. Try it for yourself:

```
play chord(:E3, :minor)
```

Now, we're really getting somewhere. That looks a lot more pretty than the raw lists (and is easier to read for other people). So what other chords does Sonic Pi support? Well, a *lot*. Try some of these:

- `chord(:E3, :m7)`
- `chord(:E3, :minor)`
- `chord(:E3, :dim7)`
- `chord(:E3, :dom7)`

## Arpeggios

We can easily turn chords into arpeggios with the function `play_pattern`:

```
play_pattern chord(:E3, :m7)
```

Ok, that's not so fun - it played it really slowly. `play_pattern` will play each note in the list separated with a call to `sleep 1` between each call to `play`. We can use another function `play_pattern_timed` to specify our own timings and speed things up:

```
play_pattern_timed chord(:E3, :m7), 0.25
```

We can even pass a list of times which it will treat as a circle of times:

```
play_pattern_timed chord(:E3, :m13), [0.25, 0.5]
```

This is the equivalent to:

```
play 52
sleep 0.25
play 55
sleep 0.5
play 59
sleep 0.25
play 62
sleep 0.5
play 66
sleep 0.25
play 69
sleep 0.5
play 73
```

Which would you prefer to write?

---

## 8.3 - Scales

Sonic Pi has support for a wide range of scales. How about playing a C3 major scale?

```
play_pattern_timed scale(:c3, :major), 0.125, release: 0.1
```

We can even ask for more octaves:

```
play_pattern_timed scale(:c3, :major, num_octaves: 3), 0.125, release: 0.1
```

How about all the notes in a pentatonic scale?

```
play_pattern_timed scale(:c3, :major_pentatonic, num_octaves: 3), 0.125, release: 0.1
```

## Random notes

Chords and scales are great ways of constraining a random choice to something meaningful. Have a play with this example which picks random notes from the chord E3 minor:

```
use_synth :tb303
loop do
  play choose(chord(:E3, :minor)), release: 0.3, cutoff: rrand(60, 120)
  sleep 0.25
end
```

Try switching in different chord names and cutoff ranges.

## Discovering Chords and Scales

To find out which scales and chords are supported by Sonic Pi simply click the Lang button on the far left of this tutorial and then choose either chord or scale in the API list. In the information in the main panel, scroll down until you see a long list of chords or scales (depending on which you're looking at).

Have fun and remember: there are no mistakes, only opportunities.

---

## 8.4 - Rings

An interesting spin on standard lists are rings. If you know some programming, you might have come across ring buffers or ring arrays. Here, we'll just go for ring - it's short and simple.

In the previous section on lists we saw how we could fetch elements out of them by using the indexing mechanism:

```
puts [52, 55, 59][1]
```

Now, what happens if you want index `100`? Well, there's clearly no element at index 100 as the list has only three elements in it. So Sonic Pi will return you `nil` which means nothing.

However, consider you have a counter such as the current beat which continually increases. Let's create our counter and our list:

```
counter = 0
notes = [52, 55, 59]
```

We can now use our counter to access a note in our list:

```
puts notes[counter]
```

Great, we got `52`. Now, let's increment our counter and get another note:

```
counter = (inc counter)
puts notes[counter]
```

Super, we now get `55` and if we do it again we get `59`. However, if we do it again, we'll run out of numbers in our list and get `nil`. What if we wanted to just loop back round and start at the beginning of the list again? This is what rings are for.

### Creating Rings

We can create rings one of two ways. Either we use the `ring` function with the elements of the ring as parameters:

```
(ring 52, 55, 59)
```

Or we can take a normal list and convert it to a ring by sending it the `.ring` message:

```
[52, 55, 59].ring
```

### Indexing Rings

Once we have a ring, you can use it in exactly the same way you would use a normal list with the exception that you can use indexes that are negative or larger than the size of the ring and they'll wrap round to always point at one of the ring's elements:

```
(ring 52, 55, 59)[0] #=> 52
(ring 52, 55, 59)[1] #=> 55
(ring 52, 55, 59)[2] #=> 59
(ring 52, 55, 59)[3] #=> 52
(ring 52, 55, 59)[-1] #=> 59
```

### Using Rings

Let's say we're using a variable to represent the current beat number. We can use this as an index into our ring to fetch notes to play, or release times or anything useful we've stored in our ring regardless of the beat number we're currently on.

### Scales and Chords are Rings

A useful thing to know is that the lists returned by `scale` and `chord` are also rings and allow you to access them with arbitrary indexes.

### Ring Constructors

In addition to `ring` there are a number of other functions which will construct a ring for us.

- `range` invites you specify a starting point, end point and step size.
- `bools` allows you to use `1s` and `0s` to succinctly represent booleans.
- `knit` allows you to knit a sequence of repeated values.
- `spread` creates a ring of bools with a Euclidean distribution.

Take a look at their respective documentation for more information.

---

## 9 - Live Coding

One of the most exciting aspects of Sonic Pi is that it enables you to write and modify code live to make music, just like you might perform live with a guitar. One advantage of this approach is to give you more feedback whilst composing (get a simple loop running and keep tweaking it till it sounds just perfect). However, the main advantage is that you can take Sonic Pi on stage and gig with it.

In this section we'll cover the fundamentals of turning your static code compositions into dynamic performances.

Hold on to your seats...

---

## 9.1 - Live Coding

Now we've learned enough to really start having some fun. In this section we'll draw from all the previous sections and show you how you can start making your music compositions live and turning them into a performance. For that we'll need 3 main ingredients:

- An ability to write code that makes sounds - CHECK!
- An ability to write functions - CHECK!
- An ability to use (named) threads - CHECK!

Alrighty, let's get started. Let's live code our first sounds. We first need a function containing the code we want to play. Let's start simple. We also want to loop calls to that function in a thread:

```
define :my_loop do
  play 50
  sleep 1
end

in_thread(name: :looper) do
  loop do
    my_loop
  end
end
```

If that looks a little too complicated to you, go back and re-read the sections on functions and threads. It's not too complicated if you've already wrapped your head around these things.

What we have here is a function definition which just plays note 50 and sleeps for a beat. We then define a named thread called `:looper` which just loops around calling `my_loop` repeatedly.

If you run this code, you'll hear note 50 repeating again and again...

### Changing it up

Now, this is where the fun starts. Whilst the code is *still running* change 50 to another number, say 55, then press the Run button again. Woah! It changed! Live!

It didn't add a new layer because we're using named threads which only allow one thread for each name. Also, the sound changed because we *redefined* the function. We gave `:my_loop` a new definition. When the `:looper` thread looped around it simply called the new definition.

Try changing it again, change the note, change the sleep time. How about adding a `use_synth` statement? For example, change it to:

```
define :my_loop do
  use_synth :tb303
  play 50, release: 0.3
  sleep 0.25
end
```

Now it sounds pretty interesting, but we can spice it up further. Instead of playing the same note again and again, try playing a chord:

```
define :my_loop do
  use_synth :tb303
  play chord(:e3, :minor), release: 0.3
  sleep 0.5
end
```

How about playing random notes from the chord:

```
define :my_loop do
  use_synth :tb303
  play choose(chord(:e3, :minor)), release: 0.3
  sleep 0.25
end
```

Or using a random cutoff value:

```
define :my_loop do
  use_synth :tb303
  play choose(chord(:e3, :minor)), release: 0.2, cutoff: rrand(60, 130)
  sleep 0.25
end
```

Finally, add some drums:

```
define :my_loop do
  use_synth :tb303
  sample :drum_bass_hard, rate: rrand(0.5, 2)
end
```



```
  play choose(chord(:e3, :minor)), release: 0.2, cutoff: rrand(60, 130)
  sleep 0.25
end
```

Now things are getting exciting!

However, before you jump up and start live coding with functions and threads, stop what you're doing and read the next section on [live\\_loop](#) which will change the way you code in Sonic Pi forever...

---

## 9.2 - Live Loops

Ok, so this section of the tutorial is the real gem. If you only read one section, it should be this one. If you read the previous section on Live Coding Fundamentals, `live_loop` is a simple way of doing exactly that but without having to write so much.

If you didn't read the previous section, `live_loop` is the best way to jam with Sonic Pi.

Let's play. Write the following in a new buffer:

```
live_loop :foo do
  play 60
  sleep 1
end
```

Now press the Run button. You hear a basic beep every beat. Nothing fun there. However, don't press Stop just yet. Change the `60` to `65` and press Run again.

Woah! It changed *automatically* without missing a beat. This is live coding.

Why not change it to be more bass like? Just update your code whilst it's playing:

```
live_loop :foo do
  use_synth :prophet
  play :e1, release: 8
  sleep 8
end
```

Then hit Run.

Let's make the cutoff move around:

```
live_loop :foo do
  use_synth :prophet
  play :e1, release: 8, cutoff: rand(70, 130)
  sleep 8
end
```

Hit Run again.

Add some drums:

```
live_loop :foo do
  sample :loop_garzul
  use_synth :prophet
  play :e1, release: 8, cutoff: rand(70, 130)
  sleep 8
end
```

Change the note from `e1` to `c1`:

```
live_loop :foo do
  sample :loop_garzul
  use_synth :prophet
  play :c1, release: 8, cutoff: rand(70, 130)
  sleep 8
end
```

Now stop listening to me and play around yourself! Have fun!

---

## 9.3 - Multiple Live Loops

Consider the following live loop:

```
live_loop :foo do
  play 50
  sleep 1
end
```

You may have wondered why it needs the name `:foo`. This name is important because it signifies that this live loop is different from all other live loops.

*There can never be two live loops running with the same name.*

This means that if we want multiple concurrently running live loops, we just need to give them different names:

```
live_loop :foo do
  use_synth :prophet
  play :c1, release: 8, cutoff: rrand(70, 130)
  sleep 8
end
```

```
live_loop :bar do
  sample :bd_haus
  sleep 0.5
end
```

You can now update and change each live loop independently and it all just works.

## Syncing Live Loops

One thing you might have already noticed is that live loops work automatically with the thread cue mechanism we explored previously. Every time the live loop loops, it generates a new `cue` event with the name of the live loop. We can therefore `sync` on these cues to ensure our loops are in sync without having to stop anything.

Consider this badly synced code:

```
live_loop :foo do
  play :e4, release: 0.5
  sleep 0.4
end

live_loop :bar do
  sample :bd_haus
  sleep 1
end
```

Let's see if we can fix the timing and sync without stopping it. First, let's fix the `:foo` loop to make the sleep a factor of 1 - something like `0.5` will do:

```
live_loop :foo do
  play :e4, release: 0.5
  sleep 0.5
end

live_loop :bar do
  sample :bd_haus
  sleep 1
end
```

We're not quite finished yet though - you'll notice that the beats don't quite line up correctly. This is because the loops are *out of phase*. Let's fix that by syncing one to the other:

```
live_loop :foo do
  play :e4, release: 0.5
  sleep 0.5
end

live_loop :bar do
  sync :foo
  sample :bd_haus
  sleep 1
end
```

Wow, everything is now perfectly in time - all without stopping.

Now, go forth and live code with live loops!

---

## 9.4 - Ticking

Something you'll likely find yourself doing a lot when live coding is looping through rings. You'll be putting notes into rings for melodies, sleeps for rhythms, chord progressions, timbral variations, etc. etc.

### Ticking Rings

Sonic Pi provides a *very* handy tool for working with rings within `live_loops`. It's called the tick system. It provides you with the ability to *tick through rings*. Let's look at an example:

```
live_loop :arp do
  play (scale :e3, :minor_pentatonic).tick, release: 0.1
  sleep 0.125
end
```

Here, we're just grabbing the scale E3 minor pentatonic and ticking through each element. This is done by adding `.tick` to the end of the scale declaration. This tick is local to the live loop, so each live loop can have its own independent tick:

```
live_loop :arp do
  play (scale :e3, :minor_pentatonic).tick, release: 0.1
  sleep 0.125
end
```

```
live_loop :arp2 do
  use_synth :dsaw
  play (scale :e2, :minor_pentatonic, num_octaves: 3).tick, release: 0.25
  sleep 0.25
end
```

### Tick

You can also call `tick` as a standard fn and use the value as an index:

```
live_loop :arp do
  idx = tick
  play (scale :e3, :minor_pentatonic)[idx], release: 0.1
  sleep 0.125
end
```

However, it is much nicer to call `.tick` at the end. The `tick` fn is for when you want to do fancy things with the tick value and for when you want to use ticks for other things than indexing into rings.

### Look

The magical thing about tick is that not only does it return a new index (or the value of the ring at that index) it also makes sure that next time you call tick, it's the next value. Take a look at the examples in the docs for `tick` for many ways of working with this. However, for now, it's important to point out that sometimes you'll want to just look at the current tick value and *not increase* it. This is available via the `look` fn. You can call `look` as a standard fn or by adding `.look` to the end of a ring.

### Naming Ticks

Finally, sometimes you'll need more than one tick per live loop. This is achieved by giving your tick a name:

```
live_loop :arp do
  play (scale :e3, :minor_pentatonic).tick(:foo), release: 0.1
  sleep (ring 0.125, 0.25).tick(:bar)
end
```

Here we're using two ticks one for the note to play and another for the sleep time. As they're both in the same live loop, to keep them separate we need to give them unique names. This is exactly the same kind of thing as naming `live_loops` - we just pass a symbol prefixed with a `:`. In the example above we called one tick `:foo` and the other `:bar`. If we want to `look` at these we also need to pass the name of the tick to `look`.

### Don't make it too complicated

Most of the power in the tick system isn't useful when you get started. Don't try and learn everything in this section. Just focus on ticking through a single ring. That'll give you most of the joy and simplicity of ticking through rings in your `live_loops`.

Take a look at the documentation for `tick` where there are many useful examples and happy ticking!

---

## 10 - Essential Knowledge

This section will cover some very useful - in fact *essential* - knowledge for getting the most out of your Sonic Pi experience.

We'll cover how to take advantage of the many keyboard shortcuts available to you, how to share your work and some tips on performing with Sonic Pi.

---

## 10.1 - Using Shortcuts

Sonic Pi is as much an instrument as a coding environment. Shortcuts can therefore make playing Sonic Pi much more *efficient and natural* - especially when you're playing live in front of an audience.

Much of Sonic Pi can be controlled through the keyboard. As you gain more familiarity working and performing with Sonic Pi, you'll likely start using the shortcuts more and more. *I personally touch-type* (I recommend you consider learning too) and find myself frustrated whenever I need to reach for the mouse as it slows me down. I therefore use all of these shortcuts on a very regular basis!

Therefore, if you learn the shortcuts, you'll learn to use your keyboard effectively and you'll be live coding like a pro in no time.

However, *don't try and learn them all at once*, just try and remember the ones you use most and then keep adding more to your practice.

### Consistency across Platforms

Imagine you're learning the clarinet. You'd expect all clarinets of all makes to have similar controls and fingerings. If they didn't, you'd have a tough time switching between different clarinets and you'd be stuck to using just one make.

Unfortunately the three major operating systems (Linux, Mac OS X and Windows) come with their own standard defaults for actions such as cut and paste etc. Sonic Pi will try and honour these standards. However *priority is placed on consistency across platforms* within Sonic Pi rather than attempting to conform to a given platform's standards. This means that when you learn the shortcuts whilst playing with Sonic Pi on your Raspberry Pi, you can move to the Mac or PC and feel right at home.

### Control and Meta

Part of the notion of consistency is the naming of shortcuts. In Sonic Pi we use the names *Control* and *Meta* to refer to the two main combination keys. On all platforms *Control* is the same. However, on Linux and Windows, *Meta* is actually the *Alt* key while on Mac *Meta* is the *Command* key. For consistency we'll use the term *Meta* - just remember to map that to the appropriate key on your operating system.

### Abbreviations

To help keep things simple and readable, we'll use the abbreviations *C-* for *Control* plus another key and *M-* for *Meta* plus another key. For example, if a shortcut requires you to hold down both *Meta* and *r* we'll write that as *M-r*. The *-* just means "at the same time as."

The following are some of the shortcuts I find most useful.

### Stopping and starting

Instead of always reaching for the mouse to run your code, you can simply press *M-r*. Similarly, to stop running code you can press *M-s*.

### Navigation

I'm really lost without the navigation shortcuts. I therefore highly recommend you spend the time to learn them. These shortcuts also work extremely well when you've learned to touch type as they use the standard letters rather than requiring you to move your hand to the mouse or the arrow keys on your keyboard.

You can move to the beginning of the line with *C-a*, the end of the line with *C-e*, up a line with *C-p*, down a line with *C-n*, forward a character with *C-f*, and back a character with *C-b*. You can even delete all the characters from the cursor to the end of the line with *C-k*.

### Tidy Code

To auto-align your code simply press *M-m*.

### Help System

To toggle the help system you can press *M-i*. However, a much more useful shortcut to know is *C-i* which will look up the word underneath the cursor and display the docs if it finds anything. Instant help!

For a full list take a look at section 10.2 Shortcut Cheatsheet.

---

## 10.2 - Shortcut Cheatsheet

The following is a summary of the main shortcuts available within Sonic Pi. Please see Section 10.1 for motivation and background.

### Conventions

In this list, we use the following conventions (where *Meta* is one of *Alt* on Windows/Linux or *Cmd* on Mac):

- **C-a** means hold the *Control* key then press the *a* key whilst holding them both at the same time, then releasing.
- **M-r** means hold the *Meta* key and then press the *r* key whilst holding them both at the same time, then releasing.
- **S-M-z** means hold the *Shift* key, then the *Meta* key, then finally the *z* key all at the same time, then releasing.
- **C-M-f** means hold the *Control* key, then press *Meta* key, finally the *f* key all at the same time, then releasing.

### Main Application Manipulation

- **M-r** - Run code
- **M-s** - Stop code
- **M-i** - Toggle Help System
- **M-p** - Toggle Preferences
- **M-{** - Switch buffer to the left
- **M-}** - Switch buffer to the right
- **M+**  - Increase text size of current buffer
- **M-**  - Decrease text size of current buffer

### Selection/Copy/Paste

- **M-a** - Select all
- **M-c** - Copy selection to paste buffer
- **M-]** - Copy selection to paste buffer
- **M-x** - Cut selection to paste buffer
- **C-]** - Cut selection to paste buffer
- **C-k** - Cut to the end of the line
- **M-v** - Paste from paste buffer to editor
- **C-y** - Paste from paste buffer to editor
- **C-SPACE** - Set mark. Navigation will now manipulate highlighted region. Use **C-g** to escape.

### Text Manipulation

- **M-m** - Align all text
- **Tab** - Align current line/selection (or complete list)
- **C-l** - Centre editor
- **M-/** - Comment/Uncomment current line
- **C-t** - Transpose/swap characters
- **M-u** - Convert next word (or selection) to upper case.
- **M-l** - Convert next word (or selection) to lower case.

### Navigation

- **C-a** - Move to beginning of line
- **C-e** - Move to end of line
- **C-p** - Move to previous line
- **C-n** - Move to next line
- **C-f** - Move forward one character
- **C-b** - Move backward one character
- **M-f** - Move forward one word
- **M-b** - Move backward one word
- **C-M-n** - Move line or selection down
- **C-M-p** - Move line or selection up
- **S-M-u** - Move up 10 lines
- **S-M-d** - Move down 10 lines
- **M-<** - Move to beginning of buffer
- **M->** - Move to end of buffer

### Deletion

- **C-h** - Delete previous character
- **C-d** - Delete next character

### Advanced Editor Features

- **C-i** - Show docs for word under cursor
- **M-z** - Undo



- **S-M-z** - Redo
  - **C-g** - Escape
  - **S-M-f** - Toggle fullscreen mode
  - **S-M-b** - Toggle visibility of buttons
  - **S-M-l** - Toggle visibility of log
  - **S-M-m** - Toggle between light/dark modes
-

## 10.3 - Sharing

Sonic Pi is all about sharing and learning with each other.

Once you've learned how to code music, sharing your compositions is as simple as sending an email containing your code. Please do *share* your code with others so they can *learn* from your work and even use parts in a new *mash-up*.

If you're unsure of the best way to share your work with others I recommend putting your code on [GitHub](#) and your music on [SoundCloud](#). That way you'll be able to easily reach a large audience.

### Code -> GitHub

[GitHub](#) is a site for sharing and working with code. It's used by professional developers as well as artists for sharing and collaborating with code. The simplest way to share a new piece of code (or even an unfinished piece) is to create a [Gist](#). A [Gist](#) is a simple way of uploading your code in a simple way that others can see, copy and share.

### Audio -> SoundCloud

Another important way of sharing your work is to record the audio and upload it to [SoundCloud](#). Once you've uploaded your piece, other users can comment and discuss your work. I also recommend placing a link to a [Gist](#) of your code in the track description.

To record your work, hit the **Rec** button in the toolbar, and recording starts immediately. Hit **Run** to start your code if it isn't already in progress. When you're done recording, press the flashing **Rec** button again, and you'll be prompted to enter a filename. The recording will be saved as a WAV file, which can be edited and converted to MP3 by any number of free programs (try Audacity for instance).

### Hope

I encourage you to share your work and really hope that we'll all teach each other new tricks and moves with Sonic Pi. I'm really excited by what you'll have to show me.

---

## 10.4 - Performing

One of the most exciting aspects of Sonic Pi is that it enables you to use code as a *musical instrument*. This means that writing code live can now be seen as a new way of performing music.

We call this *Live Coding*.

### Show Your Screen

When you live code I recommend you *show your screen* to your audience. Otherwise it's like playing a guitar but hiding your fingers and the strings. When I practice at home I use a Raspberry Pi and a little mini projector on my living room wall. You could use your TV or one of your school/work projectors to give a show. Try it, it's a lot of fun.

### Form a Band

Don't just play on your own - form a live coding band! It's a lot of fun jamming with others. One person could do beats, another ambient background, etc. See what interesting combinations of sounds you can have together.

### TOPLAP

Live coding isn't completely new - a small number of people have been doing it for a few years now, typically using bespoke systems they've built for themselves. A great place to go and find out more about other live coders and systems is [TOPLAP](#).

### Algorave

Another great resource for exploring the live coding world is [Algorave](#). Here you can find all about a specific strand of live coding for making music in nightclubs.

---

## 11 - Minecraft Pi

Sonic Pi now supports a simple API for interacting with Minecraft Pi - the special edition of Minecraft which is installed by default on the Raspberry Pi's Raspbian Linux-based operating system.

### No need to import libraries

The Minecraft Pi integration has been designed to be insanely easy to use. All you need to do is to launch Minecraft Pi and create a world. You're then free to use the `mc_*` fns just like you might use `play` and `synth`. There's no need to import anything or install any libraries - it's all ready to go and works out of the box.

### Automatic Connection

The Minecraft Pi API takes care of managing your connection to the Minecraft Pi application. This means you don't need to worry about a thing. If you try and use the Minecraft Pi API when Minecraft Pi isn't open, Sonic Pi will politely tell you. Similarly, if you close Minecraft Pi whilst you're still running a `live_loop` that uses the API, the live loop will stop and politely tell you that it can't connect. To reconnect, just launch Minecraft Pi again and Sonic Pi will automatically detect and re-create the connection for you.

### Designed to be Live Coded

The Minecraft Pi API has been designed to work seamlessly within `live_loops`. This means it's possible to synchronise modifications in your Minecraft Pi worlds with modifications in your Sonic Pi sounds. Instant Minecraft-based music videos! Note however that Minecraft Pi is alpha software and is known to be slightly buggy. If you encounter any problems simply restart Minecraft Pi and carry on as before. Sonic Pi's automatic connection functionality will take care of things for you.

### Requires a Raspberry Pi 2.0

It is highly recommended that you use a Raspberry Pi 2 if you wish to run both Sonic Pi and Minecraft at the same time - especially if you want to use Sonic Pi's sound capabilities.

### API Support

At this stage, Sonic Pi supports basic block and player manipulations which are detailed in Section 11.1. Support for event callbacks triggered by player interactions in the world is planned for a future release.

---

## 11.1 - Basic Minecraft Pi API

Sonic Pi currently supports the following basic interactions with Minecraft Pi:

- Displaying chat messages
- Setting the position of the user
- Getting the position of the user
- Setting the block type at a given coordinate
- Getting the block type at a given coordinate

Let's look at each of these in turn.

### Displaying chat messages

Let's see just how easy it is to control Minecraft Pi from Sonic Pi. First, make sure you have both Minecraft Pi and Sonic Pi open at the same time and also make sure you've entered a Minecraft world and can walk around.

In a fresh Sonic Pi buffer simply enter the following code:

```
mc_message "Hello from Sonic Pi"
```

When you hit the **Run** button, you'll see your message flash up on the Minecraft window. Congratulations, you've written your first Minecraft code! That was easy wasn't it.

### Setting the position of the user

Now, let's try a little magic. Let's teleport ourselves somewhere! Try the following:

```
mc_teleport 50, 50, 50
```

When you hit **Run** - boom! You're instantantly transported to a new place. Most likely it was somewhere in the sky and you fell down either to dry land or into water. Now, what are those numbers: **50, 50, 50**? They're the coordinates of the location you're trying to teleport to. Let's take a brief moment to explore what coordinates are and how they work because they're really, really important for programming Minecraft.

### Coordinates

Imagine a pirate's map with a big **X** marking the location of some treasure. The exact location of the **X** can be described with two numbers - how far along the map from left to right and how far along the map from bottom to top. For example **10cm** across and **8cm** up. These two numbers **10** and **8** are coordinates. You could easily imagine describing the locations of other stashes of treasure with other pairs of numbers. Perhaps there's a big chest of gold at **2** across and **9** up...

Now, in Minecraft two numbers isn't quite enough. We also need to know how high we are. We therefore need three numbers:

- How far from right to left in the world - **x**
- How far from front to back in the world - **z**
- How high up we are in the world - **y** One more thing - we typically describe these coordinates in this order **x, y, z**.

### Finding your current coordinates

Let's have a play with coordinates. Navigate to a nice place in the Minecraft map and then switch over to Sonic Pi. Now enter the following:

```
puts mc_location
```

When you hit the **Run** button you'll see the coordinates of your current position displayed in the log window. Take a note of them, then move forward in the world and try again. Notice how the coordinates changed! Now, I recommend you spend some time repeating exactly this - move a bit in the world, take a look at the coordinates and repeat. Do this until you start to get a feel for how the coordinates change when you move. Once you've understood how coordinates work, programming with the Minecraft API will be a complete breeze.

### Let's Build!

Now that you know how to find the current position and to teleport using coordinates, you have all the tools you need to start building things in Minecraft with code. Let's say you want to make the block with coordinates **40, 50, 60** to be glass. That's super easy:

```
mc_set_block :glass, 40, 50, 60
```

Haha, it really was that easy. To see your handywork just teleport nearby and take a look:

```
mc_teleport 35, 50, 60
```

Now turn around and you should see your glass block! Try changing it to diamond:

```
mc_set_block :diamond, 40, 50, 60
```

If you were looking in the right direction you might have even seen it change in front of your eyes! This is the start of something exciting...

## Looking at blocks

Let's look at one last thing before we move onto something a bit more involved. Given a set of coordinates we can ask Minecraft what the type of a specific block is. Let's try it with the diamond block you just created:

```
puts mc_get_block 40, 50, 60
```

Yey! It's `:diamond`. Try changing it back to glass and asking again - does it now say `:glass`? I'm sure it does :-)

## Available block types

Before you go on a Minecraft Pi coding rampage, you might find this list of available block types useful:

```
:air
:stone
:grass
:dirt
:cobblestone
:wood_plank
:sapling
:bedrock
:water_flowng
:water
:water_stationary
:lava_flowng
:lava
:lava_stationary
:sand
:gravel
:gold_ore
:iron_ore
:coal_ore
:wood
:leaves
:glass
:lapis
:lapis_lazuli_block
:sandstone
:bed
:cobweb
:grass_tall
:flower_yellow
:flower_cyan
:mushroom_brown
:mushroom_red
:gold_block
:gold
:iron_block
:iron
:stone_slab_double
:stone_slab
:brick
:brick_block
:tnt
:bookshelf
:moss_stone
:obsidian
:torch
:fire
:stairs_wood
:chest
:diamond_ore
:diamond_block
:diamond
:crafting_table
:farmland
:furnace_inactive
:furnace_active
:door_wood
:ladder
:stairs_cobblestone
:door_iron
:redstone_ore
:snow
```

:ice  
:snow\_block  
:cactus  
:clay  
:sugar\_cane  
:fence  
:glowstone\_block  
:bedrock\_invisible  
:stone\_brick  
:glass\_pane  
:melon  
:fence\_gate  
:glowing\_obsidian  
:nether\_reactor\_core

---

## 12 - Conclusions

This concludes the Sonic Pi introductory tutorial. Hopefully you've learned something along the way. Don't worry if you feel you didn't understand everything - just play and have fun and you'll pick things up in your own time. Feel free to dive back in when you have a question that might be covered in one of the sections.

If you have any questions that haven't been covered in the tutorial, then please jump onto the [Sonic Pi forums](#) and ask your question there. You'll find someone friendly and willing to lend a hand.

Finally, I also invite you to take a deeper look at the rest of the documentation in this help system. There are a number of features that haven't been covered in this tutorial that are waiting for your discovery.

So play, have fun, share your code, perform for your friends, show your screens and remember:

*There are no mistakes, only opportunities.*

[Sam Aaron](#)

---



Beep  
Blade  
Bnoise  
Cnoise  
Dark Ambience  
Dsaw  
Dull Bell  
Fm  
Gnoise  
Growl  
Hollow  
Hoover  
Mod Beep  
Mod Dsaw  
Mod Fm  
Mod Pulse  
Mod Saw  
Mod Sine  
Mod Tri  
Noise  
Piano  
Pnoise  
Pretty Bell  
Prophet  
Pulse  
Saw  
Sine  
Square  
Subpulse  
Supersaw  
Tb303  
Tri  
Zawa

# Sine Wave

note:	52	amp:	1	pan:	0	attack:	0	decay:	0	sustain:	0
release:	1	attack_level:	1	decay_level:	sustain_level	sustain_level:	1	env_curve:	2		

`use_synth :beep`

A simple pure sine wave. The sine wave is the simplest, purest sound there is and is the fundamental building block of all noise. The mathematician Fourier demonstrated that any sound could be built out of a number of sine waves (the more complex the sound, the more sine waves needed). Have a play combining a number of sine waves to design your own sounds!

Introduced in v2.0

## Parameters

note:	<p>Note to play. Either a MIDI number or a symbol representing a note. For example:<code>30</code>, <code>52</code>, <code>:C</code>, <code>:C2</code>, <code>:Eb4</code>, or <code>:Ds3</code></p> <p><i>Default: 52</i> <i>Must be zero or greater</i> <i>May be changed whilst playing</i> <a href="#">Has slide parameters to shape changes</a></p>
amp:	<p>The amplitude of the sound. Typically a value between 0 and 1. Higher amplitudes may be used, but won't make the sound louder, they will just reduce the quality of all the sounds currently being played (due to compression.)</p> <p><i>Default: 1</i> <i>Must be zero or greater</i> <i>May be changed whilst playing</i> <a href="#">Has slide parameters to shape changes</a></p>
pan:	<p>Position of sound in stereo. With headphones on, this means how much of the sound is in the left ear, and how much is in the right ear. With a value of -1, the sound is completely in the left ear, a value of 0 puts the sound equally in both ears and a value of 1 puts the sound in the right ear. Values in between -1 and 1 move the sound accordingly.</p> <p><i>Default: 0</i> <i>Must be a value between -1 and 1 inclusively</i> <i>May be changed whilst playing</i> <a href="#">Has slide parameters to shape changes</a></p>
attack:	<p>Amount of time (in beats) for sound to reach full amplitude (<code>attack_level</code>). A short attack (i.e. 0.01) makes the initial part of the sound very percussive like a sharp tap. A longer attack (i.e 1) fades the sound in gently. Full length of sound is <code>attack</code> + <code>decay</code> + <code>sustain</code> + <code>release</code>.</p> <p><i>Default: 0</i> <i>Must be zero or greater</i> <i>Can not be changed once set</i> <i>Scaled with current BPM value</i></p>
decay:	<p>Amount of time (in beats) for the sound to move from full amplitude (<code>attack_level</code>) to the sustain amplitude (<code>sustain_level</code>).</p> <p><i>Default: 0</i> <i>Must be zero or greater</i> <i>Can not be changed once set</i> <i>Scaled with current BPM value</i></p>
sustain:	<p>Amount of time (in beats) for sound to remain at sustain level amplitude. Longer sustain values result in longer sounds. Full length of sound is <code>attack</code> + <code>decay</code> + <code>sustain</code> + <code>release</code>.</p> <p><i>Default: 0</i> <i>Must be zero or greater</i> <i>Can not be changed once set</i> <i>Scaled with current BPM value</i></p>
release:	<p>Amount of time (in beats) for sound to move from sustain level amplitude to silent. A short release (i.e. 0.01) makes the final part of the sound very percussive (potentially resulting in a click). A longer release (i.e 1) fades the sound out gently. Full length of sound is <code>attack</code> + <code>decay</code> + <code>sustain</code> + <code>release</code>.</p> <p><i>Default: 1</i> <i>Must be zero or greater</i> <i>Can not be changed once set</i> <i>Scaled with current BPM value</i></p>
attack_level:	<p>Amplitude level reached after attack phase and immediately before decay phase</p> <p><i>Default: 1</i> <i>Must be zero or greater</i> <i>Can not be changed once set</i></p>
decay_level:	<p>Amplitude level reached after decay phase and immediately before sustain phase. Defaults to <code>sustain_level</code> unless explicitly set</p> <p><i>Default: sustain_level</i> <i>Must be zero or greater</i> <i>Can not be changed once set</i></p>
	<p>Amplitude level reached after decay phase and immediately before release phase</p>

<b>sustain_level:</b>	Amplitude level reached after decay phase and immediately before release phase. <i>Default: 1</i> <i>Must be zero or greater</i> <i>Can not be changed once set</i>
<b>env_curve:</b>	Select the shape of the curve between levels in the envelope. 1=linear, 2=exponential, 3=sine, 4=welch, 6=squared, 7=cubed <i>Default: 2</i> <i>Must be one of the following values: [1, 2, 3, 4, 6, 7]</i> <i>Can not be changed once set</i>

## Slide Parameters

Any parameter that is slidable has three additional parameters named `_slide`, `_slide_curve`, and `_slide_shape`. For example, 'amp' is slidable, so you can also set `amp_slide`, `amp_slide_curve`, and `amp_slide_shape` with the following effects:

<b>_slide:</b>	Amount of time (in beats) for the parameter value to change. A long parameter <code>_slide</code> value means that the parameter takes a long time to slide from the previous value to the new value. A parameter <code>_slide</code> of 0 means that the parameter instantly changes to the new value. <i>Default: 0</i>
<b>_slide_shape:</b>	Shape of curve. 0: step, 1: linear, 3: sine, 4: welch, 5: custom (use <code>*_slide_curve</code> : opt e.g. <code>amp_slide_curve</code> :), 6: squared, 7: cubed. <i>Default: 5</i>
<b>_slide_curve:</b>	Shape of the slide curve (only honoured if slide shape is 5). 0 means linear and positive and negative numbers curve the segment up and down respectively. <i>Default: 0</i>

# Blade Runner style strings

note:	52	amp:	1	pan:	0	attack:	0	decay:	0	sustain:	0
release:	1	attack_level:	1	decay_level:	sustain_level	sustain_level:	1	env_curve:	2	cutoff:	100
vibrato_rate:	6	vibrato_depth:	0.15	vibrato_delay:	0.5	vibrato_onset:	0.1				

`use_synth :blade`

Straight from the 70s, evoking the mists of Blade Runner, this simple electro-style string synth is based on filtered saw waves and a variable vibrato.

Introduced in v2.6

## Parameters

note:	Note to play. Either a MIDI number or a symbol representing a note. For example: <code>30</code> , <code>52</code> , <code>:C</code> , <code>:C2</code> , <code>:Eb4</code> , or <code>:Ds3</code> <i>Default: 52</i> <i>Must be zero or greater</i> <i>May be changed whilst playing</i> <a href="#">Has slide parameters to shape changes</a>
amp:	The amplitude of the sound. Typically a value between 0 and 1. Higher amplitudes may be used, but won't make the sound louder, they will just reduce the quality of all the sounds currently being played (due to compression.) <i>Default: 1</i> <i>Must be zero or greater</i> <i>May be changed whilst playing</i> <a href="#">Has slide parameters to shape changes</a>
pan:	Position of sound in stereo. With headphones on, this means how much of the sound is in the left ear, and how much is in the right ear. With a value of -1, the sound is completely in the left ear, a value of 0 puts the sound equally in both ears and a value of 1 puts the sound in the right ear. Values in between -1 and 1 move the sound accordingly. <i>Default: 0</i> <i>Must be a value between -1 and 1 inclusively</i> <i>May be changed whilst playing</i> <a href="#">Has slide parameters to shape changes</a>
attack:	Amount of time (in beats) for sound to reach full amplitude ( <code>attack_level</code> ). A short attack (i.e. 0.01) makes the initial part of the sound very percussive like a sharp tap. A longer attack (i.e 1) fades the sound in gently. Full length of sound is attack + decay + sustain + release. <i>Default: 0</i> <i>Must be zero or greater</i> <i>Can not be changed once set</i> <i>Scaled with current BPM value</i>
decay:	Amount of time (in beats) for the sound to move from full amplitude ( <code>attack_level</code> ) to the sustain amplitude ( <code>sustain_level</code> ). <i>Default: 0</i> <i>Must be zero or greater</i> <i>Can not be changed once set</i> <i>Scaled with current BPM value</i>
sustain:	Amount of time (in beats) for sound to remain at sustain level amplitude. Longer sustain values result in longer sounds. Full length of sound is attack + decay + sustain + release. <i>Default: 0</i> <i>Must be zero or greater</i> <i>Can not be changed once set</i> <i>Scaled with current BPM value</i>
release:	Amount of time (in beats) for sound to move from sustain level amplitude to silent. A short release (i.e. 0.01) makes the final part of the sound very percussive (potentially resulting in a click). A longer release (i.e 1) fades the sound out gently. Full length of sound is attack + decay + sustain + release. <i>Default: 1</i> <i>Must be zero or greater</i> <i>Can not be changed once set</i> <i>Scaled with current BPM value</i>
attack_level:	Amplitude level reached after attack phase and immediately before decay phase <i>Default: 1</i> <i>Must be zero or greater</i> <i>Can not be changed once set</i>
decay_level:	Amplitude level reached after decay phase and immediately before sustain phase. Defaults to <code>sustain_level</code> unless explicitly set <i>Default: sustain_level</i> <i>Must be zero or greater</i> <i>Can not be changed once set</i>

<b>sustain_level:</b>	Amplitude level reached after decay phase and immediately before release phase.  <i>Default: 1</i> <i>Must be zero or greater</i> <i>Can not be changed once set</i>
<b>env_curve:</b>	Select the shape of the curve between levels in the envelope. 1=linear, 2=exponential, 3=sine, 4=welch, 6=squared, 7=cubed  <i>Default: 2</i> <i>Must be one of the following values: [1, 2, 3, 4, 6, 7]</i> <i>Can not be changed once set</i>
<b>cutoff:</b>	MIDI note representing the highest frequencies allowed to be present in the sound. A low value like 30 makes the sound round and dull, a high value like 100 makes the sound buzzy and crispy.  <i>Default: 100</i> <i>Must be zero or greater, must be a value less than 131</i> <i>May be changed whilst playing</i> <i>Has slide parameters to shape changes</i>
<b>vibrato_rate:</b>	Number of wobbles per second. For realism this should be between 6 and 8, maybe even faster for really high notes.  <i>Default: 6</i> <i>Must be a value greater than or equal to 0.0, must be a value less than or equal to 20.0</i> <i>May be changed whilst playing</i> <i>Has slide parameters to shape changes</i>
<b>vibrato_depth:</b>	Amount of variation around the central note. 1 is the sensible maximum (but you can go up to 5 if you want a special effect), 0 would mean no vibrato. Works well around 0.15 but you can experiment.  <i>Default: 0.15</i> <i>Must be a value greater than or equal to 0.0, must be a value less than or equal to 5.0</i> <i>May be changed whilst playing</i> <i>Has slide parameters to shape changes</i>
<b>vibrato_delay:</b>	How long in seconds before the vibrato kicks in.  <i>Default: 0.5</i> <i>Must be zero or greater</i> <i>Can not be changed once set</i>
<b>vibrato_onset:</b>	How long in seconds before the vibrato reaches full power.  <i>Default: 0.1</i> <i>Must be zero or greater</i> <i>Can not be changed once set</i>

## Slide Parameters

Any parameter that is slidable has three additional parameters named `_slide`, `_slide_curve`, and `_slide_shape`. For example, 'amp' is slidable, so you can also set `amp_slide`, `amp_slide_curve`, and `amp_slide_shape` with the following effects:

<b>_slide:</b>	Amount of time (in beats) for the parameter value to change. A long parameter <code>_slide</code> value means that the parameter takes a long time to slide from the previous value to the new value. A parameter <code>_slide</code> of 0 means that the parameter instantly changes to the new value.  <i>Default: 0</i>
<b>_slide_shape:</b>	Shape of curve. 0: step, 1: linear, 3: sine, 4: welch, 5: custom (use <code>*_slide_curve</code> : opt e.g. <code>amp_slide_curve</code> ), 6: squared, 7: cubed.  <i>Default: 5</i>
<b>_slide_curve:</b>	Shape of the slide curve (only honoured if slide shape is 5). 0 means linear and positive and negative numbers curve the segment up and down respectively.  <i>Default: 0</i>

# Brown Noise

amp:	1	pan:	0	attack:	0	decay:	0	sustain:	0	release:	1
attack_level:	1	decay_level:	sustain_level	sustain_level:	1	env_curve:	2	cutoff:	110	res:	0

`use_synth :bnoise`

Noise whose spectrum falls off in power by 6 dB per octave. Useful for generating percussive sounds such as snares and hand claps. Also useful for simulating wind or sea effects.

Introduced in v2.0

## Parameters

amp:	<p>The amplitude of the sound. Typically a value between 0 and 1. Higher amplitudes may be used, but won't make the sound louder, they will just reduce the quality of all the sounds currently being played (due to compression.)</p> <p><i>Default: 1</i> <i>Must be zero or greater</i> <i>May be changed whilst playing</i> <i>Has slide parameters to shape changes</i></p>
pan:	<p>Position of sound in stereo. With headphones on, this means how much of the sound is in the left ear, and how much is in the right ear. With a value of -1, the sound is completely in the left ear, a value of 0 puts the sound equally in both ears and a value of 1 puts the sound in the right ear. Values in between -1 and 1 move the sound accordingly.</p> <p><i>Default: 0</i> <i>Must be a value between -1 and 1 inclusively</i> <i>May be changed whilst playing</i> <i>Has slide parameters to shape changes</i></p>
attack:	<p>Amount of time (in beats) for sound to reach full amplitude (attack_level). A short attack (i.e. 0.01) makes the initial part of the sound very percussive like a sharp tap. A longer attack (i.e 1) fades the sound in gently. Full length of sound is attack + decay + sustain + release.</p> <p><i>Default: 0</i> <i>Must be zero or greater</i> <i>Can not be changed once set</i> <i>Scaled with current BPM value</i></p>
decay:	<p>Amount of time (in beats) for the sound to move from full amplitude (attack_level) to the sustain amplitude (sustain_level).</p> <p><i>Default: 0</i> <i>Must be zero or greater</i> <i>Can not be changed once set</i> <i>Scaled with current BPM value</i></p>
sustain:	<p>Amount of time (in beats) for sound to remain at sustain level amplitude. Longer sustain values result in longer sounds. Full length of sound is attack + decay + sustain + release.</p> <p><i>Default: 0</i> <i>Must be zero or greater</i> <i>Can not be changed once set</i> <i>Scaled with current BPM value</i></p>
release:	<p>Amount of time (in beats) for sound to move from sustain level amplitude to silent. A short release (i.e. 0.01) makes the final part of the sound very percussive (potentially resulting in a click). A longer release (i.e 1) fades the sound out gently. Full length of sound is attack + decay + sustain + release.</p> <p><i>Default: 1</i> <i>Must be zero or greater</i> <i>Can not be changed once set</i> <i>Scaled with current BPM value</i></p>
attack_level:	<p>Amplitude level reached after attack phase and immediately before decay phase</p> <p><i>Default: 1</i> <i>Must be zero or greater</i> <i>Can not be changed once set</i></p>
decay_level:	<p>Amplitude level reached after decay phase and immediately before sustain phase. Defaults to sustain_level unless explicitly set</p> <p><i>Default: sustain_level</i> <i>Must be zero or greater</i> <i>Can not be changed once set</i></p>
sustain_level:	<p>Amplitude level reached after decay phase and immediately before release phase.</p> <p><i>Default: 1</i> <i>Must be zero or greater</i> <i>Can not be changed once set</i></p>
	<p>Select the shape of the curve between levels in the envelope. 1=linear, 2=exponential, 3=sine, 4=welch, 6=squared, 7=cubed</p>

<b>env_curve:</b>	<p><i>Default: 2</i>  <i>Must be one of the following values: [1, 2, 3, 4, 6, 7]</i>  <i>Can not be changed once set</i></p>
<b>cutoff:</b>	<p>MIDI note representing the highest frequencies allowed to be present in the sound. A low value like 30 makes the sound round and dull, a high value like 100 makes the sound buzzy and crispy.</p> <p><i>Default: 110</i>  <i>Must be zero or greater, must be a value less than 131</i>  <i>May be changed whilst playing</i>  <i>Has slide parameters to shape changes</i></p>
<b>res:</b>	<p>Filter resonance as a value between 0 and 1. Large amounts of resonance (a res: near 1) can create a whistling sound around the cutoff frequency. Smaller values produce less resonance.</p> <p><i>Default: 0</i>  <i>Must be zero or greater, must be a value less than 1</i>  <i>May be changed whilst playing</i>  <i>Has slide parameters to shape changes</i></p>

## Slide Parameters

Any parameter that is slidable has three additional parameters named `_slide`, `_slide_curve`, and `_slide_shape`. For example, 'amp' is slidable, so you can also set `amp_slide`, `amp_slide_curve`, and `amp_slide_shape` with the following effects:

<b>_slide:</b>	<p>Amount of time (in beats) for the parameter value to change. A long parameter_slide value means that the parameter takes a long time to slide from the previous value to the new value. A parameter_slide of 0 means that the parameter instantly changes to the new value.</p> <p><i>Default: 0</i></p>
<b>_slide_shape:</b>	<p>Shape of curve. 0: step, 1: linear, 3: sine, 4: welch, 5: custom (use *_slide_curve: opt e.g. amp_slide_curve:), 6: squared, 7: cubed.</p> <p><i>Default: 5</i></p>
<b>_slide_curve:</b>	<p>Shape of the slide curve (only honoured if slide shape is 5). 0 means linear and positive and negative numbers curve the segment up and down respectively.</p> <p><i>Default: 0</i></p>

# Clip Noise

amp:	1	pan:	0	attack:	0	decay:	0	sustain:	0	release:	1
attack_level:	1	decay_level:	sustain_level	sustain_level:	1	env_curve:	2	cutoff:	110	res:	0

`use_synth :cnoise`

Generates noise whose values are either -1 or 1. This produces the maximum energy for the least peak to peak amplitude. Useful for generating percussive sounds such as snares and hand claps. Also useful for simulating wind or sea effects.

Introduced in v2.0

## Parameters

amp:	<p>The amplitude of the sound. Typically a value between 0 and 1. Higher amplitudes may be used, but won't make the sound louder, they will just reduce the quality of all the sounds currently being played (due to compression.)</p> <p><i>Default: 1</i> <i>Must be zero or greater</i> <i>May be changed whilst playing</i> <i>Has slide parameters to shape changes</i></p>
pan:	<p>Position of sound in stereo. With headphones on, this means how much of the sound is in the left ear, and how much is in the right ear. With a value of -1, the sound is completely in the left ear, a value of 0 puts the sound equally in both ears and a value of 1 puts the sound in the right ear. Values in between -1 and 1 move the sound accordingly.</p> <p><i>Default: 0</i> <i>Must be a value between -1 and 1 inclusively</i> <i>May be changed whilst playing</i> <i>Has slide parameters to shape changes</i></p>
attack:	<p>Amount of time (in beats) for sound to reach full amplitude (attack_level). A short attack (i.e. 0.01) makes the initial part of the sound very percussive like a sharp tap. A longer attack (i.e 1) fades the sound in gently. Full length of sound is attack + decay + sustain + release.</p> <p><i>Default: 0</i> <i>Must be zero or greater</i> <i>Can not be changed once set</i> <i>Scaled with current BPM value</i></p>
decay:	<p>Amount of time (in beats) for the sound to move from full amplitude (attack_level) to the sustain amplitude (sustain_level).</p> <p><i>Default: 0</i> <i>Must be zero or greater</i> <i>Can not be changed once set</i> <i>Scaled with current BPM value</i></p>
sustain:	<p>Amount of time (in beats) for sound to remain at sustain level amplitude. Longer sustain values result in longer sounds. Full length of sound is attack + decay + sustain + release.</p> <p><i>Default: 0</i> <i>Must be zero or greater</i> <i>Can not be changed once set</i> <i>Scaled with current BPM value</i></p>
release:	<p>Amount of time (in beats) for sound to move from sustain level amplitude to silent. A short release (i.e. 0.01) makes the final part of the sound very percussive (potentially resulting in a click). A longer release (i.e 1) fades the sound out gently. Full length of sound is attack + decay + sustain + release.</p> <p><i>Default: 1</i> <i>Must be zero or greater</i> <i>Can not be changed once set</i> <i>Scaled with current BPM value</i></p>
attack_level:	<p>Amplitude level reached after attack phase and immediately before decay phase</p> <p><i>Default: 1</i> <i>Must be zero or greater</i> <i>Can not be changed once set</i></p>
decay_level:	<p>Amplitude level reached after decay phase and immediately before sustain phase. Defaults to sustain_level unless explicitly set</p> <p><i>Default: sustain_level</i> <i>Must be zero or greater</i> <i>Can not be changed once set</i></p>
sustain_level:	<p>Amplitude level reached after decay phase and immediately before release phase.</p> <p><i>Default: 1</i> <i>Must be zero or greater</i> <i>Can not be changed once set</i></p>
	<p>Select the shape of the curve between levels in the envelope. 1=linear, 2=exponential, 3=sine, 4=welch, 6=squared, 7=cubed</p>



<b>env_curve:</b>	<p><i>Default: 2</i>  <i>Must be one of the following values: [1, 2, 3, 4, 6, 7]</i>  <i>Can not be changed once set</i></p>
<b>cutoff:</b>	<p>MIDI note representing the highest frequencies allowed to be present in the sound. A low value like 30 makes the sound round and dull, a high value like 100 makes the sound buzzy and crispy.</p> <p><i>Default: 110</i>  <i>Must be zero or greater, must be a value less than 131</i>  <i>May be changed whilst playing</i>  <i>Has slide parameters to shape changes</i></p>
<b>res:</b>	<p>Filter resonance as a value between 0 and 1. Large amounts of resonance (a res: near 1) can create a whistling sound around the cutoff frequency. Smaller values produce less resonance.</p> <p><i>Default: 0</i>  <i>Must be zero or greater, must be a value less than 1</i>  <i>May be changed whilst playing</i>  <i>Has slide parameters to shape changes</i></p>

## Slide Parameters

Any parameter that is slidable has three additional parameters named `_slide`, `_slide_curve`, and `_slide_shape`. For example, 'amp' is slidable, so you can also set `amp_slide`, `amp_slide_curve`, and `amp_slide_shape` with the following effects:

<b>_slide:</b>	<p>Amount of time (in beats) for the parameter value to change. A long parameter_slide value means that the parameter takes a long time to slide from the previous value to the new value. A parameter_slide of 0 means that the parameter instantly changes to the new value.</p> <p><i>Default: 0</i></p>
<b>_slide_shape:</b>	<p>Shape of curve. 0: step, 1: linear, 3: sine, 4: welch, 5: custom (use *_slide_curve: opt e.g. amp_slide_curve:), 6: squared, 7: cubed.</p> <p><i>Default: 5</i></p>
<b>_slide_curve:</b>	<p>Shape of the slide curve (only honoured if slide shape is 5). 0 means linear and positive and negative numbers curve the segment up and down respectively.</p> <p><i>Default: 0</i></p>

# Dark Ambience

note:	52	amp:	1	pan:	0	attack:	0	decay:	0	sustain:	0
release:	1	attack_level:	1	decay_level:	sustain_level	sustain_level:	1	env_curve:	2	cutoff:	110
res:	0.7	detune1:	12	detune2:	24	noise:	0	ring:	0.2	room:	70
reverb_time:	100										

`use_synth :dark_ambience`

A slow rolling bass with a sparkle of light trying to escape the darkness. Great for an ambient sound.

Introduced in v2.4

## Parameters

note:	<p>Note to play. Either a MIDI number or a symbol representing a note. For example:<code>30</code>, <code>52</code>, <code>:C</code>, <code>:C2</code>, <code>:Eb4</code>, or <code>:Ds3</code></p> <p><i>Default: 52</i> <i>Must be zero or greater</i> <i>May be changed whilst playing</i> <i>Has slide parameters to shape changes</i></p>
amp:	<p>The amplitude of the sound. Typically a value between 0 and 1. Higher amplitudes may be used, but won't make the sound louder, they will just reduce the quality of all the sounds currently being played (due to compression.)</p> <p><i>Default: 1</i> <i>Must be zero or greater</i> <i>May be changed whilst playing</i> <i>Has slide parameters to shape changes</i></p>
pan:	<p>Position of sound in stereo. With headphones on, this means how much of the sound is in the left ear, and how much is in the right ear. With a value of -1, the sound is completely in the left ear, a value of 0 puts the sound equally in both ears and a value of 1 puts the sound in the right ear. Values in between -1 and 1 move the sound accordingly.</p> <p><i>Default: 0</i> <i>Must be a value between -1 and 1 inclusively</i> <i>May be changed whilst playing</i> <i>Has slide parameters to shape changes</i></p>
attack:	<p>Amount of time (in beats) for sound to reach full amplitude (<code>attack_level</code>). A short attack (i.e. 0.01) makes the initial part of the sound very percussive like a sharp tap. A longer attack (i.e 1) fades the sound in gently. Full length of sound is <code>attack</code> + <code>decay</code> + <code>sustain</code> + <code>release</code>.</p> <p><i>Default: 0</i> <i>Must be zero or greater</i> <i>Can not be changed once set</i> <i>Scaled with current BPM value</i></p>
decay:	<p>Amount of time (in beats) for the sound to move from full amplitude (<code>attack_level</code>) to the sustain amplitude (<code>sustain_level</code>).</p> <p><i>Default: 0</i> <i>Must be zero or greater</i> <i>Can not be changed once set</i> <i>Scaled with current BPM value</i></p>
sustain:	<p>Amount of time (in beats) for sound to remain at sustain level amplitude. Longer sustain values result in longer sounds. Full length of sound is <code>attack</code> + <code>decay</code> + <code>sustain</code> + <code>release</code>.</p> <p><i>Default: 0</i> <i>Must be zero or greater</i> <i>Can not be changed once set</i> <i>Scaled with current BPM value</i></p>
release:	<p>Amount of time (in beats) for sound to move from sustain level amplitude to silent. A short release (i.e. 0.01) makes the final part of the sound very percussive (potentially resulting in a click). A longer release (i.e 1) fades the sound out gently. Full length of sound is <code>attack</code> + <code>decay</code> + <code>sustain</code> + <code>release</code>.</p> <p><i>Default: 1</i> <i>Must be zero or greater</i> <i>Can not be changed once set</i> <i>Scaled with current BPM value</i></p>
attack_level:	<p>Amplitude level reached after attack phase and immediately before decay phase</p> <p><i>Default: 1</i> <i>Must be zero or greater</i> <i>Can not be changed once set</i></p>
decay_level:	<p>Amplitude level reached after decay phase and immediately before sustain phase. Defaults to <code>sustain_level</code> unless explicitly set</p> <p><i>Default: sustain_level</i> <i>Must be zero or greater</i> <i>Can not be changed once set</i></p>

<b>sustain_level:</b>	Amplitude level reached after decay phase and immediately before release phase. <i>Default: 1</i> <i>Must be zero or greater</i> <i>Can not be changed once set</i>
<b>env_curve:</b>	Select the shape of the curve between levels in the envelope. 1=linear, 2=exponential, 3=sine, 4=welch, 6=squared, 7=cubed <i>Default: 2</i> <i>Must be one of the following values: [1, 2, 3, 4, 6, 7]</i> <i>Can not be changed once set</i>
<b>cutoff:</b>	MIDI note representing the highest frequencies allowed to be present in the sound. A low value like 30 makes the sound round and dull, a high value like 100 makes the sound buzzy and crispy. <i>Default: 110</i> <i>Must be zero or greater, must be a value less than 131</i> <i>May be changed whilst playing</i> <i>Has slide parameters to shape changes</i>
<b>res:</b>	Filter resonance as a value between 0 and 1. Large amounts of resonance (a res: near 1) can create a whistling sound around the cutoff frequency. Smaller values produce less resonance. <i>Default: 0.7</i> <i>Must be zero or greater, must be a value less than 1</i> <i>May be changed whilst playing</i> <i>Has slide parameters to shape changes</i>
<b>detune1:</b>	Distance (in MIDI notes) between the main note and the second component of sound. Affects thickness, sense of tuning and harmony. <i>Default: 12</i> <i>Can not be changed once set</i> <i>Has slide parameters to shape changes</i>
<b>detune2:</b>	Distance (in MIDI notes) between the main note and the third component of sound. Affects thickness, sense of tuning and harmony. Tiny values such as 0.1 create a thick sound. <i>Default: 24</i> <i>Can not be changed once set</i> <i>Has slide parameters to shape changes</i>
<b>noise:</b>	Noise source. Has a subtle effect on the timbre of the sound. 0=pink noise (the default), 1=brown noise, 2=white noise, 3=clip noise and 4=grey noise <i>Default: 0</i> <i>Must be one of the following values: [0, 1, 2, 3, 4]</i> <i>May be changed whilst playing</i>
<b>ring:</b>	Amount of ring in the sound. Lower values create a more rough sound, higher values produce a sound with more focus. <i>Default: 0.2</i> <i>Must be a value between 0.1 and 50 inclusively</i> <i>May be changed whilst playing</i>
<b>room:</b>	Room size in squared metres used to calculate the reverb. <i>Default: 70</i> <i>Must be a value greater than or equal to 0.1, must be a value less than or equal to 300</i> <i>Can not be changed once set</i>
<b>reverb_time:</b>	How long in beats the reverb should go on for. <i>Default: 100</i> <i>Must be zero or greater</i> <i>Can not be changed once set</i>

## Slide Parameters

Any parameter that is slidable has three additional parameters named `_slide`, `_slide_curve`, and `_slide_shape`. For example, 'amp' is slidable, so you can also set `amp_slide`, `amp_slide_curve`, and `amp_slide_shape` with the following effects:

<b>_slide:</b>	Amount of time (in beats) for the parameter value to change. A long parameter <code>_slide</code> value means that the parameter takes a long time to slide from the previous value to the new value. A parameter <code>_slide</code> of 0 means that the parameter instantly changes to the new value. <i>Default: 0</i>
<b>_slide_shape:</b>	Shape of curve. 0: step, 1: linear, 3: sine, 4: welch, 5: custom (use <code>*_slide_curve</code> : opt e.g. <code>amp_slide_curve</code> ), 6: squared, 7: cubed. <i>Default: 5</i>
<b>_slide_curve:</b>	Shape of the slide curve (only honoured if slide shape is 5). 0 means linear and positive and negative numbers curve the segment up and down respectively.



# Detuned Saw wave

note:	52	amp:	1	pan:	0	attack:	0	decay:	0	sustain:	0
release:	1	attack_level:	1	decay_level:	sustain_level	sustain_level:	1	env_curve:	2	cutoff:	100
detune:	0.1										

`use_synth :dsaw`

A pair of detuned saw waves passed through a low pass filter. Two saw waves with slightly different frequencies generates a nice thick sound which is the basis for a lot of famous synth sounds. Thicken the sound by increasing the detune value, or create an octave-playing synth by choosing a detune of 12 (12 MIDI notes is an octave).

Introduced in v2.0

## Parameters

note:	<p>Note to play. Either a MIDI number or a symbol representing a note. For example:<b>30, 52, :C, :C2, :Eb4</b>, or <b>:Ds3</b></p> <p><i>Default: 52</i> <i>Must be zero or greater</i> <i>May be changed whilst playing</i> <a href="#">Has slide parameters to shape changes</a></p>
amp:	<p>The amplitude of the sound. Typically a value between 0 and 1. Higher amplitudes may be used, but won't make the sound louder, they will just reduce the quality of all the sounds currently being played (due to compression.)</p> <p><i>Default: 1</i> <i>Must be zero or greater</i> <i>May be changed whilst playing</i> <a href="#">Has slide parameters to shape changes</a></p>
pan:	<p>Position of sound in stereo. With headphones on, this means how much of the sound is in the left ear, and how much is in the right ear. With a value of -1, the sound is completely in the left ear, a value of 0 puts the sound equally in both ears and a value of 1 puts the sound in the right ear. Values in between -1 and 1 move the sound accordingly.</p> <p><i>Default: 0</i> <i>Must be a value between -1 and 1 inclusively</i> <i>May be changed whilst playing</i> <a href="#">Has slide parameters to shape changes</a></p>
attack:	<p>Amount of time (in beats) for sound to reach full amplitude (attack_level). A short attack (i.e. 0.01) makes the initial part of the sound very percussive like a sharp tap. A longer attack (i.e 1) fades the sound in gently. Full length of sound is attack + decay + sustain + release.</p> <p><i>Default: 0</i> <i>Must be zero or greater</i> <i>Can not be changed once set</i> <i>Scaled with current BPM value</i></p>
decay:	<p>Amount of time (in beats) for the sound to move from full amplitude (attack_level) to the sustain amplitude (sustain_level).</p> <p><i>Default: 0</i> <i>Must be zero or greater</i> <i>Can not be changed once set</i> <i>Scaled with current BPM value</i></p>
sustain:	<p>Amount of time (in beats) for sound to remain at sustain level amplitude. Longer sustain values result in longer sounds. Full length of sound is attack + decay + sustain + release.</p> <p><i>Default: 0</i> <i>Must be zero or greater</i> <i>Can not be changed once set</i> <i>Scaled with current BPM value</i></p>
release:	<p>Amount of time (in beats) for sound to move from sustain level amplitude to silent. A short release (i.e. 0.01) makes the final part of the sound very percussive (potentially resulting in a click). A longer release (i.e 1) fades the sound out gently. Full length of sound is attack + decay + sustain + release.</p> <p><i>Default: 1</i> <i>Must be zero or greater</i> <i>Can not be changed once set</i> <i>Scaled with current BPM value</i></p>
attack_level:	<p>Amplitude level reached after attack phase and immediately before decay phase</p> <p><i>Default: 1</i> <i>Must be zero or greater</i> <i>Can not be changed once set</i></p>
decay_level:	<p>Amplitude level reached after decay phase and immediately before sustain phase. Defaults to sustain_level unless explicitly set</p> <p><i>Default: sustain_level</i> <i>Must be zero or greater</i> <i>Can not be changed once set</i></p>

<b>sustain_level:</b>	Amplitude level reached after decay phase and immediately before release phase. <i>Default: 1</i> <i>Must be zero or greater</i> <i>Can not be changed once set</i>
<b>env_curve:</b>	Select the shape of the curve between levels in the envelope. 1=linear, 2=exponential, 3=sine, 4=welch, 6=squared, 7=cubed <i>Default: 2</i> <i>Must be one of the following values: [1, 2, 3, 4, 6, 7]</i> <i>Can not be changed once set</i>
<b>cutoff:</b>	MIDI note representing the highest frequencies allowed to be present in the sound. A low value like 30 makes the sound round and dull, a high value like 100 makes the sound buzzy and crispy. <i>Default: 100</i> <i>Must be zero or greater, must be a value less than 131</i> <i>May be changed whilst playing</i> <i>Has slide parameters to shape changes</i>
<b>detune:</b>	Distance (in MIDI notes) between components of sound. Affects thickness, sense of tuning and harmony. Tiny values such as 0.1 create a thick sound. Larger values such as 0.5 make the tuning sound strange. Even bigger values such as 5 create chord-like sounds. <i>Default: 0.1</i> <i>May be changed whilst playing</i> <i>Has slide parameters to shape changes</i>

## Slide Parameters

Any parameter that is slidable has three additional parameters named `_slide`, `_slide_curve`, and `_slide_shape`. For example, 'amp' is slidable, so you can also set `amp_slide`, `amp_slide_curve`, and `amp_slide_shape` with the following effects:

<b>_slide:</b>	Amount of time (in beats) for the parameter value to change. A long <code>parameter_slide</code> value means that the parameter takes a long time to slide from the previous value to the new value. A <code>parameter_slide</code> of 0 means that the parameter instantly changes to the new value. <i>Default: 0</i>
<b>_slide_shape:</b>	Shape of curve. 0: step, 1: linear, 3: sine, 4: welch, 5: custom (use <code>*_slide_curve</code> : opt e.g. <code>amp_slide_curve</code> ), 6: squared, 7: cubed. <i>Default: 5</i>
<b>_slide_curve:</b>	Shape of the slide curve (only honoured if slide shape is 5). 0 means linear and positive and negative numbers curve the segment up and down respectively. <i>Default: 0</i>

# Dull Bell

```
note: 52 amp: 1 pan: 0 attack: 0 decay: 0 sustain: 0  
release: 1 attack_level: 1 decay_level: sustain_level sustain_level: 1 env_curve: 2
```

`use_synth :dull_bell`

A simple dull discordant bell sound.

Introduced in v2.0

## Parameters

<b>note:</b>	Note to play. Either a MIDI number or a symbol representing a note. For example: <code>30</code> , <code>52</code> , <code>:C</code> , <code>:C2</code> , <code>:Eb4</code> , or <code>:Ds3</code> <i>Default: 52</i> <i>Must be zero or greater</i> <i>May be changed whilst playing</i> <i>Has slide parameters to shape changes</i>
<b>amp:</b>	The amplitude of the sound. Typically a value between 0 and 1. Higher amplitudes may be used, but won't make the sound louder, they will just reduce the quality of all the sounds currently being played (due to compression.) <i>Default: 1</i> <i>Must be zero or greater</i> <i>May be changed whilst playing</i> <i>Has slide parameters to shape changes</i>
<b>pan:</b>	Position of sound in stereo. With headphones on, this means how much of the sound is in the left ear, and how much is in the right ear. With a value of -1, the sound is completely in the left ear, a value of 0 puts the sound equally in both ears and a value of 1 puts the sound in the right ear. Values in between -1 and 1 move the sound accordingly. <i>Default: 0</i> <i>Must be a value between -1 and 1 inclusively</i> <i>May be changed whilst playing</i> <i>Has slide parameters to shape changes</i>
<b>attack:</b>	Amount of time (in beats) for sound to reach full amplitude ( <code>attack_level</code> ). A short attack (i.e. 0.01) makes the initial part of the sound very percussive like a sharp tap. A longer attack (i.e 1) fades the sound in gently. Full length of sound is attack + decay + sustain + release. <i>Default: 0</i> <i>Must be zero or greater</i> <i>Can not be changed once set</i> <i>Scaled with current BPM value</i>
<b>decay:</b>	Amount of time (in beats) for the sound to move from full amplitude ( <code>attack_level</code> ) to the sustain amplitude ( <code>sustain_level</code> ). <i>Default: 0</i> <i>Must be zero or greater</i> <i>Can not be changed once set</i> <i>Scaled with current BPM value</i>
<b>sustain:</b>	Amount of time (in beats) for sound to remain at sustain level amplitude. Longer sustain values result in longer sounds. Full length of sound is attack + decay + sustain + release. <i>Default: 0</i> <i>Must be zero or greater</i> <i>Can not be changed once set</i> <i>Scaled with current BPM value</i>
<b>release:</b>	Amount of time (in beats) for sound to move from sustain level amplitude to silent. A short release (i.e. 0.01) makes the final part of the sound very percussive (potentially resulting in a click). A longer release (i.e 1) fades the sound out gently. Full length of sound is attack + decay + sustain + release. <i>Default: 1</i> <i>Must be zero or greater</i> <i>Can not be changed once set</i> <i>Scaled with current BPM value</i>
<b>attack_level:</b>	Amplitude level reached after attack phase and immediately before decay phase <i>Default: 1</i> <i>Must be zero or greater</i> <i>Can not be changed once set</i>
<b>decay_level:</b>	Amplitude level reached after decay phase and immediately before sustain phase. Defaults to <code>sustain_level</code> unless explicitly set <i>Default: sustain_level</i> <i>Must be zero or greater</i> <i>Can not be changed once set</i>
	Amplitude level reached after decay phase and immediately before release phase.

<b>sustain_level:</b>	<i>Default: 1</i> <i>Must be zero or greater</i> <i>Can not be changed once set</i>
<b>env_curve:</b>	Select the shape of the curve between levels in the envelope. 1=linear, 2=exponential, 3=sine, 4=welch, 6=squared, 7=cubed <i>Default: 2</i> <i>Must be one of the following values: [1, 2, 3, 4, 6, 7]</i> <i>Can not be changed once set</i>

## Slide Parameters

Any parameter that is slidable has three additional parameters named `_slide`, `_slide_curve`, and `_slide_shape`. For example, 'amp' is slidable, so you can also set `amp_slide`, `amp_slide_curve`, and `amp_slide_shape` with the following effects:

<b>_slide:</b>	Amount of time (in beats) for the parameter value to change. A long parameter <code>_slide</code> value means that the parameter takes a long time to slide from the previous value to the new value. A parameter <code>_slide</code> of 0 means that the parameter instantly changes to the new value. <i>Default: 0</i>
<b>_slide_shape:</b>	Shape of curve. 0: step, 1: linear, 3: sine, 4: welch, 5: custom (use <code>*_slide_curve</code> : opt e.g. <code>amp_slide_curve</code> :), 6: squared, 7: cubed. <i>Default: 5</i>
<b>_slide_curve:</b>	Shape of the slide curve (only honoured if slide shape is 5). 0 means linear and positive and negative numbers curve the segment up and down respectively. <i>Default: 0</i>



# Basic FM synthesis

note:	52	amp:	1	pan:	0	attack:	0	decay:	0	sustain:	0
release:	1	attack_level:	1	decay_level:	sustain_level	sustain_level:	1	env_curve:	2	cutoff:	100
divisor:	2	depth:	1								

`use_synth :fm`

A sine wave with a fundamental frequency which is modulated at audio rate by another sine wave with a specific modulation, division and depth. Useful for generating a wide range of sounds by playing with the divisor and depth params. Great for deep powerful bass and crazy 70s sci-fi sounds.

Introduced in v2.0

## Parameters

note:	<p>Note to play. Either a MIDI number or a symbol representing a note. For example:<code>30</code>, <code>52</code>, <code>:C</code>, <code>:C2</code>, <code>:Eb4</code>, or <code>:Ds3</code></p> <p><i>Default: 52</i> <i>Must be zero or greater</i> <i>May be changed whilst playing</i> <i>Has slide parameters to shape changes</i></p>
amp:	<p>The amplitude of the sound. Typically a value between 0 and 1. Higher amplitudes may be used, but won't make the sound louder, they will just reduce the quality of all the sounds currently being played (due to compression.)</p> <p><i>Default: 1</i> <i>Must be zero or greater</i> <i>May be changed whilst playing</i> <i>Has slide parameters to shape changes</i></p>
pan:	<p>Position of sound in stereo. With headphones on, this means how much of the sound is in the left ear, and how much is in the right ear. With a value of -1, the sound is completely in the left ear, a value of 0 puts the sound equally in both ears and a value of 1 puts the sound in the right ear. Values in between -1 and 1 move the sound accordingly.</p> <p><i>Default: 0</i> <i>Must be a value between -1 and 1 inclusively</i> <i>May be changed whilst playing</i> <i>Has slide parameters to shape changes</i></p>
attack:	<p>Amount of time (in beats) for sound to reach full amplitude (<code>attack_level</code>). A short attack (i.e. 0.01) makes the initial part of the sound very percussive like a sharp tap. A longer attack (i.e 1) fades the sound in gently. Full length of sound is <code>attack + decay + sustain + release</code>.</p> <p><i>Default: 0</i> <i>Must be zero or greater</i> <i>Can not be changed once set</i> <i>Scaled with current BPM value</i></p>
decay:	<p>Amount of time (in beats) for the sound to move from full amplitude (<code>attack_level</code>) to the sustain amplitude (<code>sustain_level</code>).</p> <p><i>Default: 0</i> <i>Must be zero or greater</i> <i>Can not be changed once set</i> <i>Scaled with current BPM value</i></p>
sustain:	<p>Amount of time (in beats) for sound to remain at sustain level amplitude. Longer sustain values result in longer sounds. Full length of sound is <code>attack + decay + sustain + release</code>.</p> <p><i>Default: 0</i> <i>Must be zero or greater</i> <i>Can not be changed once set</i> <i>Scaled with current BPM value</i></p>
release:	<p>Amount of time (in beats) for sound to move from sustain level amplitude to silent. A short release (i.e. 0.01) makes the final part of the sound very percussive (potentially resulting in a click). A longer release (i.e 1) fades the sound out gently. Full length of sound is <code>attack + decay + sustain + release</code>.</p> <p><i>Default: 1</i> <i>Must be zero or greater</i> <i>Can not be changed once set</i> <i>Scaled with current BPM value</i></p>
attack_level:	<p>Amplitude level reached after attack phase and immediately before decay phase</p> <p><i>Default: 1</i> <i>Must be zero or greater</i> <i>Can not be changed once set</i></p>
decay_level:	<p>Amplitude level reached after decay phase and immediately before sustain phase. Defaults to <code>sustain_level</code> unless explicitly set</p> <p><i>Default: sustain_level</i> <i>Must be zero or greater</i> <i>Can not be changed once set</i></p>

<b>sustain_level:</b>	Amplitude level reached after decay phase and immediately before release phase. <i>Default: 1</i> <i>Must be zero or greater</i> <i>Can not be changed once set</i>
<b>env_curve:</b>	Select the shape of the curve between levels in the envelope. 1=linear, 2=exponential, 3=sine, 4=welch, 6=squared, 7=cubed <i>Default: 2</i> <i>Must be one of the following values: [1, 2, 3, 4, 6, 7]</i> <i>Can not be changed once set</i>
<b>cutoff:</b>	MIDI note representing the highest frequencies allowed to be present in the sound. A low value like 30 makes the sound round and dull, a high value like 100 makes the sound buzzy and crispy. <i>Default: 100</i> <i>Must be zero or greater, must be a value less than 131</i> <i>May be changed whilst playing</i> <i>Has slide parameters to shape changes</i>
<b>divisor:</b>	Modifies the frequency of the modulator oscillator relative to the carrier. Don't worry too much about what this means - just try different numbers out! <i>Default: 2</i> <i>May be changed whilst playing</i> <i>Has slide parameters to shape changes</i>
<b>depth:</b>	Modifies the depth of the carrier wave used to modify fundamental frequency. Don't worry too much about what this means - just try different numbers out! <i>Default: 1</i> <i>May be changed whilst playing</i> <i>Has slide parameters to shape changes</i>

## Slide Parameters

Any parameter that is slidable has three additional parameters named `_slide`, `_slide_curve`, and `_slide_shape`. For example, 'amp' is slidable, so you can also set `amp_slide`, `amp_slide_curve`, and `amp_slide_shape` with the following effects:

<b>_slide:</b>	Amount of time (in beats) for the parameter value to change. A long parameter <code>_slide</code> value means that the parameter takes a long time to slide from the previous value to the new value. A parameter <code>_slide</code> of 0 means that the parameter instantly changes to the new value. <i>Default: 0</i>
<b>_slide_shape:</b>	Shape of curve. 0: step, 1: linear, 3: sine, 4: welch, 5: custom (use <code>*_slide_curve</code> : opt e.g. <code>amp_slide_curve</code> :), 6: squared, 7: cubed. <i>Default: 5</i>
<b>_slide_curve:</b>	Shape of the slide curve (only honoured if slide shape is 5). 0 means linear and positive and negative numbers curve the segment up and down respectively. <i>Default: 0</i>

# Grey Noise

amp:	1	pan:	0	attack:	0	decay:	0	sustain:	0	release:	1
attack_level:	1	decay_level:	sustain_level	sustain_level:	1	env_curve:	2	cutoff:	110	res:	0

`use_synth :gnoise`

Generates noise which results from flipping random bits in a word. The spectrum is emphasised towards lower frequencies. Useful for generating percussive sounds such as snares and hand claps. Also useful for simulating wind or sea effects.

Introduced in v2.0

## Parameters

amp:	<p>The amplitude of the sound. Typically a value between 0 and 1. Higher amplitudes may be used, but won't make the sound louder, they will just reduce the quality of all the sounds currently being played (due to compression.)</p> <p><i>Default: 1</i> <i>Must be zero or greater</i> <i>May be changed whilst playing</i> <i>Has slide parameters to shape changes</i></p>
pan:	<p>Position of sound in stereo. With headphones on, this means how much of the sound is in the left ear, and how much is in the right ear. With a value of -1, the sound is completely in the left ear, a value of 0 puts the sound equally in both ears and a value of 1 puts the sound in the right ear. Values in between -1 and 1 move the sound accordingly.</p> <p><i>Default: 0</i> <i>Must be a value between -1 and 1 inclusively</i> <i>May be changed whilst playing</i> <i>Has slide parameters to shape changes</i></p>
attack:	<p>Amount of time (in beats) for sound to reach full amplitude (attack_level). A short attack (i.e. 0.01) makes the initial part of the sound very percussive like a sharp tap. A longer attack (i.e 1) fades the sound in gently. Full length of sound is attack + decay + sustain + release.</p> <p><i>Default: 0</i> <i>Must be zero or greater</i> <i>Can not be changed once set</i> <i>Scaled with current BPM value</i></p>
decay:	<p>Amount of time (in beats) for the sound to move from full amplitude (attack_level) to the sustain amplitude (sustain_level).</p> <p><i>Default: 0</i> <i>Must be zero or greater</i> <i>Can not be changed once set</i> <i>Scaled with current BPM value</i></p>
sustain:	<p>Amount of time (in beats) for sound to remain at sustain level amplitude. Longer sustain values result in longer sounds. Full length of sound is attack + decay + sustain + release.</p> <p><i>Default: 0</i> <i>Must be zero or greater</i> <i>Can not be changed once set</i> <i>Scaled with current BPM value</i></p>
release:	<p>Amount of time (in beats) for sound to move from sustain level amplitude to silent. A short release (i.e. 0.01) makes the final part of the sound very percussive (potentially resulting in a click). A longer release (i.e 1) fades the sound out gently. Full length of sound is attack + decay + sustain + release.</p> <p><i>Default: 1</i> <i>Must be zero or greater</i> <i>Can not be changed once set</i> <i>Scaled with current BPM value</i></p>
attack_level:	<p>Amplitude level reached after attack phase and immediately before decay phase</p> <p><i>Default: 1</i> <i>Must be zero or greater</i> <i>Can not be changed once set</i></p>
decay_level:	<p>Amplitude level reached after decay phase and immediately before sustain phase. Defaults to sustain_level unless explicitly set</p> <p><i>Default: sustain_level</i> <i>Must be zero or greater</i> <i>Can not be changed once set</i></p>
sustain_level:	<p>Amplitude level reached after decay phase and immediately before release phase.</p> <p><i>Default: 1</i> <i>Must be zero or greater</i> <i>Can not be changed once set</i></p>
	<p>Select the shape of the curve between levels in the envelope. 1=linear, 2=exponential, 3=sine, 4=welch, 6=squared, 7=cubed</p>

<b>env_curve:</b>	<p><i>Default: 2</i>  <i>Must be one of the following values: [1, 2, 3, 4, 6, 7]</i>  <i>Can not be changed once set</i></p>
<b>cutoff:</b>	<p>MIDI note representing the highest frequencies allowed to be present in the sound. A low value like 30 makes the sound round and dull, a high value like 100 makes the sound buzzy and crispy.</p> <p><i>Default: 110</i>  <i>Must be zero or greater, must be a value less than 131</i>  <i>May be changed whilst playing</i>  <i>Has slide parameters to shape changes</i></p>
<b>res:</b>	<p>Filter resonance as a value between 0 and 1. Large amounts of resonance (a res: near 1) can create a whistling sound around the cutoff frequency. Smaller values produce less resonance.</p> <p><i>Default: 0</i>  <i>Must be zero or greater, must be a value less than 1</i>  <i>May be changed whilst playing</i>  <i>Has slide parameters to shape changes</i></p>

## Slide Parameters

Any parameter that is slidable has three additional parameters named `_slide`, `_slide_curve`, and `_slide_shape`. For example, 'amp' is slidable, so you can also set `amp_slide`, `amp_slide_curve`, and `amp_slide_shape` with the following effects:

<b>_slide:</b>	<p>Amount of time (in beats) for the parameter value to change. A long parameter_slide value means that the parameter takes a long time to slide from the previous value to the new value. A parameter_slide of 0 means that the parameter instantly changes to the new value.</p> <p><i>Default: 0</i></p>
<b>_slide_shape:</b>	<p>Shape of curve. 0: step, 1: linear, 3: sine, 4: welch, 5: custom (use *_slide_curve: opt e.g. amp_slide_curve:), 6: squared, 7: cubed.</p> <p><i>Default: 5</i></p>
<b>_slide_curve:</b>	<p>Shape of the slide curve (only honoured if slide shape is 5). 0 means linear and positive and negative numbers curve the segment up and down respectively.</p> <p><i>Default: 0</i></p>

# Growl

note:	52	amp:	1	pan:	0	attack:	0.1	decay:	0	sustain:	0
release:	1	attack_level:	1	decay_level:	sustain_level	sustain_level:	1	env_curve:	2	cutoff:	130
res:	0.7										

`use_synth :growl`

A deep rumbling growl with a bright sine shining through at higher notes.

Introduced in v2.4

## Parameters

note:	<p>Note to play. Either a MIDI number or a symbol representing a note. For example:<code>30</code>, <code>52</code>, <code>:C</code>, <code>:C2</code>, <code>:Eb4</code>, or <code>:Ds3</code></p> <p><i>Default: 52</i> <i>Must be zero or greater</i> <i>May be changed whilst playing</i> <i>Has slide parameters to shape changes</i></p>
amp:	<p>The amplitude of the sound. Typically a value between 0 and 1. Higher amplitudes may be used, but won't make the sound louder, they will just reduce the quality of all the sounds currently being played (due to compression.)</p> <p><i>Default: 1</i> <i>Must be zero or greater</i> <i>May be changed whilst playing</i> <i>Has slide parameters to shape changes</i></p>
pan:	<p>Position of sound in stereo. With headphones on, this means how much of the sound is in the left ear, and how much is in the right ear. With a value of -1, the sound is completely in the left ear, a value of 0 puts the sound equally in both ears and a value of 1 puts the sound in the right ear. Values in between -1 and 1 move the sound accordingly.</p> <p><i>Default: 0</i> <i>Must be a value between -1 and 1 inclusively</i> <i>May be changed whilst playing</i> <i>Has slide parameters to shape changes</i></p>
attack:	<p>Amount of time (in beats) for sound to reach full amplitude (<code>attack_level</code>). A short attack (i.e. 0.01) makes the initial part of the sound very percussive like a sharp tap. A longer attack (i.e 1) fades the sound in gently. Full length of sound is attack + decay + sustain + release.</p> <p><i>Default: 0.1</i> <i>Must be zero or greater</i> <i>Can not be changed once set</i> <i>Scaled with current BPM value</i></p>
decay:	<p>Amount of time (in beats) for the sound to move from full amplitude (<code>attack_level</code>) to the sustain amplitude (<code>sustain_level</code>).</p> <p><i>Default: 0</i> <i>Must be zero or greater</i> <i>Can not be changed once set</i> <i>Scaled with current BPM value</i></p>
sustain:	<p>Amount of time (in beats) for sound to remain at sustain level amplitude. Longer sustain values result in longer sounds. Full length of sound is attack + decay + sustain + release.</p> <p><i>Default: 0</i> <i>Must be zero or greater</i> <i>Can not be changed once set</i> <i>Scaled with current BPM value</i></p>
release:	<p>Amount of time (in beats) for sound to move from sustain level amplitude to silent. A short release (i.e. 0.01) makes the final part of the sound very percussive (potentially resulting in a click). A longer release (i.e 1) fades the sound out gently. Full length of sound is attack + decay + sustain + release.</p> <p><i>Default: 1</i> <i>Must be zero or greater</i> <i>Can not be changed once set</i> <i>Scaled with current BPM value</i></p>
attack_level:	<p>Amplitude level reached after attack phase and immediately before decay phase</p> <p><i>Default: 1</i> <i>Must be zero or greater</i> <i>Can not be changed once set</i></p>
decay_level:	<p>Amplitude level reached after decay phase and immediately before sustain phase. Defaults to <code>sustain_level</code> unless explicitly set</p> <p><i>Default: sustain_level</i> <i>Must be zero or greater</i> <i>Can not be changed once set</i></p>
	<p>Amplitude level reached after decay phase and immediately before release phase.</p>

<b>sustain_level:</b>	<p><i>Default: 1</i>  <i>Must be zero or greater</i>  <i>Can not be changed once set</i></p>
<b>env_curve:</b>	<p>Select the shape of the curve between levels in the envelope. 1=linear, 2=exponential, 3=sine, 4=welch, 6=squared, 7=cubed</p> <p><i>Default: 2</i>  <i>Must be one of the following values: [1, 2, 3, 4, 6, 7]</i>  <i>Can not be changed once set</i></p>
<b>cutoff:</b>	<p>MIDI note representing the highest frequencies allowed to be present in the sound. A low value like 30 makes the sound round and dull, a high value like 100 makes the sound buzzy and crispy.</p> <p><i>Default: 130</i>  <i>Must be zero or greater, must be a value less than 131</i>  <i>May be changed whilst playing</i>  <i>Has slide parameters to shape changes</i></p>
<b>res:</b>	<p>Filter resonance as a value between 0 and 1. Large amounts of resonance (a res: near 1) can create a whistling sound around the cutoff frequency. Smaller values produce less resonance.</p> <p><i>Default: 0.7</i>  <i>Must be zero or greater, must be a value less than 1</i>  <i>May be changed whilst playing</i>  <i>Has slide parameters to shape changes</i></p>

## Slide Parameters

Any parameter that is slidable has three additional parameters named `_slide`, `_slide_curve`, and `_slide_shape`. For example, 'amp' is slidable, so you can also set `amp_slide`, `amp_slide_curve`, and `amp_slide_shape` with the following effects:

<b>_slide:</b>	<p>Amount of time (in beats) for the parameter value to change. A long parameter_slide value means that the parameter takes a long time to slide from the previous value to the new value. A parameter_slide of 0 means that the parameter instantly changes to the new value.</p> <p><i>Default: 0</i></p>
<b>_slide_shape:</b>	<p>Shape of curve. 0: step, 1: linear, 3: sine, 4: welch, 5: custom (use *_slide_curve: opt e.g. amp_slide_curve:), 6: squared, 7: cubed.</p> <p><i>Default: 5</i></p>
<b>_slide_curve:</b>	<p>Shape of the slide curve (only honoured if slide shape is 5). 0 means linear and positive and negative numbers curve the segment up and down respectively.</p> <p><i>Default: 0</i></p>

# Hollow

note:	52	amp:	1	pan:	0	attack:	0	decay:	0	sustain:	0
release:	1	attack_level:	1	decay_level:	sustain_level	sustain_level:	1	env_curve:	2	cutoff:	90
res:	0.99	noise:	1	norm:	0						

`use_synth :hollow`

A hollow breathy sound constructed from random noise

Introduced in v2.4

## Parameters

note:	<p>Note to play. Either a MIDI number or a symbol representing a note. For example:<code>30</code>, <code>52</code>, <code>:C</code>, <code>:C2</code>, <code>:Eb4</code>, or <code>:Ds3</code></p> <p><i>Default: 52</i> <i>Must be zero or greater</i> <i>May be changed whilst playing</i> <i>Has slide parameters to shape changes</i></p>
amp:	<p>The amplitude of the sound. Typically a value between 0 and 1. Higher amplitudes may be used, but won't make the sound louder, they will just reduce the quality of all the sounds currently being played (due to compression.)</p> <p><i>Default: 1</i> <i>Must be zero or greater</i> <i>May be changed whilst playing</i> <i>Has slide parameters to shape changes</i></p>
pan:	<p>Position of sound in stereo. With headphones on, this means how much of the sound is in the left ear, and how much is in the right ear. With a value of -1, the sound is completely in the left ear, a value of 0 puts the sound equally in both ears and a value of 1 puts the sound in the right ear. Values in between -1 and 1 move the sound accordingly.</p> <p><i>Default: 0</i> <i>Must be a value between -1 and 1 inclusively</i> <i>May be changed whilst playing</i> <i>Has slide parameters to shape changes</i></p>
attack:	<p>Amount of time (in beats) for sound to reach full amplitude (<code>attack_level</code>). A short attack (i.e. 0.01) makes the initial part of the sound very percussive like a sharp tap. A longer attack (i.e 1) fades the sound in gently. Full length of sound is <code>attack</code> + <code>decay</code> + <code>sustain</code> + <code>release</code>.</p> <p><i>Default: 0</i> <i>Must be zero or greater</i> <i>Can not be changed once set</i> <i>Scaled with current BPM value</i></p>
decay:	<p>Amount of time (in beats) for the sound to move from full amplitude (<code>attack_level</code>) to the sustain amplitude (<code>sustain_level</code>).</p> <p><i>Default: 0</i> <i>Must be zero or greater</i> <i>Can not be changed once set</i> <i>Scaled with current BPM value</i></p>
sustain:	<p>Amount of time (in beats) for sound to remain at sustain level amplitude. Longer sustain values result in longer sounds. Full length of sound is <code>attack</code> + <code>decay</code> + <code>sustain</code> + <code>release</code>.</p> <p><i>Default: 0</i> <i>Must be zero or greater</i> <i>Can not be changed once set</i> <i>Scaled with current BPM value</i></p>
release:	<p>Amount of time (in beats) for sound to move from sustain level amplitude to silent. A short release (i.e. 0.01) makes the final part of the sound very percussive (potentially resulting in a click). A longer release (i.e 1) fades the sound out gently. Full length of sound is <code>attack</code> + <code>decay</code> + <code>sustain</code> + <code>release</code>.</p> <p><i>Default: 1</i> <i>Must be zero or greater</i> <i>Can not be changed once set</i> <i>Scaled with current BPM value</i></p>
attack_level:	<p>Amplitude level reached after attack phase and immediately before decay phase</p> <p><i>Default: 1</i> <i>Must be zero or greater</i> <i>Can not be changed once set</i></p>
decay_level:	<p>Amplitude level reached after decay phase and immediately before sustain phase. Defaults to <code>sustain_level</code> unless explicitly set</p> <p><i>Default: sustain_level</i> <i>Must be zero or greater</i> <i>Can not be changed once set</i></p>
	<p>Amplitude level reached after decay phase and immediately before release phase.</p>

<b>sustain_level:</b>	<p><i>Default: 1</i>  <i>Must be zero or greater</i>  <i>Can not be changed once set</i></p>
<b>env_curve:</b>	<p>Select the shape of the curve between levels in the envelope. 1=linear, 2=exponential, 3=sine, 4=welch, 6=squared, 7=cubed</p> <p><i>Default: 2</i>  <i>Must be one of the following values: [1, 2, 3, 4, 6, 7]</i>  <i>Can not be changed once set</i></p>
<b>cutoff:</b>	<p>MIDI note representing the highest frequencies allowed to be present in the sound. A low value like 30 makes the sound round and dull, a high value like 100 makes the sound buzzy and crispy.</p> <p><i>Default: 90</i>  <i>Must be zero or greater, must be a value less than 131</i>  <i>May be changed whilst playing</i>  <i>Has slide parameters to shape changes</i></p>
<b>res:</b>	<p>Filter resonance as a value between 0 and 1. Only functional if a cutoff value is specified. Large amounts of resonance (a res: near 1) can create a whistling sound around the cutoff frequency. Smaller values produce less resonance.</p> <p><i>Default: 0.99</i>  <i>Must be zero or greater, must be a value less than 1</i>  <i>May be changed whilst playing</i>  <i>Has slide parameters to shape changes</i></p>
<b>noise:</b>	<p>Noise source. Has a subtle effect on the timbre of the sound. 0=pink noise, 1=brown noise (the default), 2=white noise, 3=clip noise and 4=grey noise</p> <p><i>Default: 1</i>  <i>Must be one of the following values: [0, 1, 2, 3, 4]</i>  <i>May be changed whilst playing</i></p>
<b>norm:</b>	<p>Normalise the audio (make quieter parts of the sample louder and louder parts quieter) - this is similar to the normaliser FX. This may emphasise any clicks caused by clipping.</p> <p><i>Default: 0</i>  <i>Must be one of the following values: [0, 1]</i>  <i>May be changed whilst playing</i></p>

## Slide Parameters

Any parameter that is slidable has three additional parameters named `_slide`, `_slide_curve`, and `_slide_shape`. For example, 'amp' is slidable, so you can also set `amp_slide`, `amp_slide_curve`, and `amp_slide_shape` with the following effects:

<b>_slide:</b>	<p>Amount of time (in beats) for the parameter value to change. A long <code>parameter_slide</code> value means that the parameter takes a long time to slide from the previous value to the new value. A <code>parameter_slide</code> of 0 means that the parameter instantly changes to the new value.</p> <p><i>Default: 0</i></p>
<b>_slide_shape:</b>	<p>Shape of curve. 0: step, 1: linear, 3: sine, 4: welch, 5: custom (use <code>*_slide_curve</code>: opt e.g. <code>amp_slide_curve</code>), 6: squared, 7: cubed.</p> <p><i>Default: 5</i></p>
<b>_slide_curve:</b>	<p>Shape of the slide curve (only honoured if slide shape is 5). 0 means linear and positive and negative numbers curve the segment up and down respectively.</p> <p><i>Default: 0</i></p>



# Hoover

note:	52	amp:	1	pan:	0	attack:	0.05	decay:	0	sustain:	0
release:	1	attack_level:	1	decay_level:	sustain_level	sustain_level:	1	env_curve:	2	cutoff:	130
res:	0.1										

`use_synth :hoover`

Classic early 90's rave synth - 'a sort of slurry chorussy synth line like the classic Dominator by Human Resource'. Based on Dan Stowell's implementation in SuperCollider and Daniel Turczanski's port to Overtone. Works really well with portamento (see docs for the 'control' method).

Introduced in v2.6

## Parameters

note:	Note to play. Either a MIDI number or a symbol representing a note. For example: <code>30</code> , <code>52</code> , <code>:C</code> , <code>:C2</code> , <code>:Eb4</code> , or <code>:Ds3</code> <i>Default: 52</i> <i>Must be zero or greater</i> <i>May be changed whilst playing</i> <i>Has slide parameters to shape changes</i>
amp:	The amplitude of the sound. Typically a value between 0 and 1. Higher amplitudes may be used, but won't make the sound louder, they will just reduce the quality of all the sounds currently being played (due to compression.) <i>Default: 1</i> <i>Must be zero or greater</i> <i>May be changed whilst playing</i> <i>Has slide parameters to shape changes</i>
pan:	Position of sound in stereo. With headphones on, this means how much of the sound is in the left ear, and how much is in the right ear. With a value of -1, the sound is completely in the left ear, a value of 0 puts the sound equally in both ears and a value of 1 puts the sound in the right ear. Values in between -1 and 1 move the sound accordingly. <i>Default: 0</i> <i>Must be a value between -1 and 1 inclusively</i> <i>May be changed whilst playing</i> <i>Has slide parameters to shape changes</i>
attack:	Amount of time (in beats) for sound to reach full amplitude ( <code>attack_level</code> ). A short attack (i.e. 0.01) makes the initial part of the sound very percussive like a sharp tap. A longer attack (i.e 1) fades the sound in gently. Full length of sound is <code>attack + decay + sustain + release</code> . <i>Default: 0.05</i> <i>Must be zero or greater</i> <i>Can not be changed once set</i> <i>Scaled with current BPM value</i>
decay:	Amount of time (in beats) for the sound to move from full amplitude ( <code>attack_level</code> ) to the sustain amplitude ( <code>sustain_level</code> ). <i>Default: 0</i> <i>Must be zero or greater</i> <i>Can not be changed once set</i> <i>Scaled with current BPM value</i>
sustain:	Amount of time (in beats) for sound to remain at sustain level amplitude. Longer sustain values result in longer sounds. Full length of sound is <code>attack + decay + sustain + release</code> . <i>Default: 0</i> <i>Must be zero or greater</i> <i>Can not be changed once set</i> <i>Scaled with current BPM value</i>
release:	Amount of time (in beats) for sound to move from sustain level amplitude to silent. A short release (i.e. 0.01) makes the final part of the sound very percussive (potentially resulting in a click). A longer release (i.e 1) fades the sound out gently. Full length of sound is <code>attack + decay + sustain + release</code> . <i>Default: 1</i> <i>Must be zero or greater</i> <i>Can not be changed once set</i> <i>Scaled with current BPM value</i>
attack_level:	Amplitude level reached after attack phase and immediately before decay phase <i>Default: 1</i> <i>Must be zero or greater</i> <i>Can not be changed once set</i>
decay_level:	Amplitude level reached after decay phase and immediately before sustain phase. Defaults to <code>sustain_level</code> unless explicitly set <i>Default: sustain_level</i> <i>Must be zero or greater</i> <i>Can not be changed once set</i>

<b>sustain_level:</b>	Amplitude level reached after decay phase and immediately before release phase. <i>Default: 1</i> <i>Must be zero or greater</i> <i>Can not be changed once set</i>
<b>env_curve:</b>	Select the shape of the curve between levels in the envelope. 1=linear, 2=exponential, 3=sine, 4=welch, 6=squared, 7=cubed <i>Default: 2</i> <i>Must be one of the following values: [1, 2, 3, 4, 6, 7]</i> <i>Can not be changed once set</i>
<b>cutoff:</b>	MIDI note representing the highest frequencies allowed to be present in the sound. A low value like 30 makes the sound round and dull, a high value like 100 makes the sound buzzy and crispy. <i>Default: 130</i> <i>Must be zero or greater, must be a value less than 131</i> <i>May be changed whilst playing</i> <i>Has slide parameters to shape changes</i>
<b>res:</b>	Filter resonance as a value between 0 and 1. Large amounts of resonance (a res: near 1) can create a whistling sound around the cutoff frequency. Smaller values produce less resonance. <i>Default: 0.1</i> <i>Must be zero or greater, must be a value less than 1</i> <i>May be changed whilst playing</i> <i>Has slide parameters to shape changes</i>

## Slide Parameters

Any parameter that is slidable has three additional parameters named `_slide`, `_slide_curve`, and `_slide_shape`. For example, 'amp' is slidable, so you can also set `amp_slide`, `amp_slide_curve`, and `amp_slide_shape` with the following effects:

<b>_slide:</b>	Amount of time (in beats) for the parameter value to change. A long parameter_slide value means that the parameter takes a long time to slide from the previous value to the new value. A parameter_slide of 0 means that the parameter instantly changes to the new value. <i>Default: 0</i>
<b>_slide_shape:</b>	Shape of curve. 0: step, 1: linear, 3: sine, 4: welch, 5: custom (use *_slide_curve: opt e.g. amp_slide_curve:), 6: squared, 7: cubed. <i>Default: 5</i>
<b>_slide_curve:</b>	Shape of the slide curve (only honoured if slide shape is 5). 0 means linear and positive and negative numbers curve the segment up and down respectively. <i>Default: 0</i>

# Modulated Sine Wave

note:	52	amp:	1	pan:	0	attack:	0	decay:	0	sustain:	0
release:	1	attack_level:	1	decay_level:	sustain_level	sustain_level:	1	env_curve:	2	cutoff:	100
mod_phase:	0.25	mod_range:	5	mod_pulse_width:	0.5	mod_phase_offset:	0	mod_invert_wave:	0	mod_wave:	1

`use_synth :mod_beep`

A sine wave passed through a low pass filter which modulates between two separate notes via a variety of control waves.

Introduced in v2.0

## Parameters

note:	<p>Note to play. Either a MIDI number or a symbol representing a note. For example: <b>30</b>, <b>52</b>, <b>:C</b>, <b>:C2</b>, <b>:Eb4</b>, or <b>:Ds3</b></p> <p><i>Default: 52</i> <i>Must be zero or greater</i> <i>May be changed whilst playing</i> <i>Has slide parameters to shape changes</i></p>
amp:	<p>The amplitude of the sound. Typically a value between 0 and 1. Higher amplitudes may be used, but won't make the sound louder, they will just reduce the quality of all the sounds currently being played (due to compression.)</p> <p><i>Default: 1</i> <i>Must be zero or greater</i> <i>May be changed whilst playing</i> <i>Has slide parameters to shape changes</i></p>
pan:	<p>Position of sound in stereo. With headphones on, this means how much of the sound is in the left ear, and how much is in the right ear. With a value of -1, the sound is completely in the left ear, a value of 0 puts the sound equally in both ears and a value of 1 puts the sound in the right ear. Values in between -1 and 1 move the sound accordingly.</p> <p><i>Default: 0</i> <i>Must be a value between -1 and 1 inclusively</i> <i>May be changed whilst playing</i> <i>Has slide parameters to shape changes</i></p>
attack:	<p>Amount of time (in beats) for sound to reach full amplitude (attack_level). A short attack (i.e. 0.01) makes the initial part of the sound very percussive like a sharp tap. A longer attack (i.e. 1) fades the sound in gently. Full length of sound is attack + decay + sustain + release.</p> <p><i>Default: 0</i> <i>Must be zero or greater</i> <i>Can not be changed once set</i> <i>Scaled with current BPM value</i></p>
decay:	<p>Amount of time (in beats) for the sound to move from full amplitude (attack_level) to the sustain amplitude (sustain_level).</p> <p><i>Default: 0</i> <i>Must be zero or greater</i> <i>Can not be changed once set</i> <i>Scaled with current BPM value</i></p>
sustain:	<p>Amount of time (in beats) for sound to remain at sustain level amplitude. Longer sustain values result in longer sounds. Full length of sound is attack + decay + sustain + release.</p> <p><i>Default: 0</i> <i>Must be zero or greater</i> <i>Can not be changed once set</i> <i>Scaled with current BPM value</i></p>
release:	<p>Amount of time (in beats) for sound to move from sustain level amplitude to silent. A short release (i.e. 0.01) makes the final part of the sound very percussive (potentially resulting in a click). A longer release (i.e. 1) fades the sound out gently. Full length of sound is attack + decay + sustain + release.</p> <p><i>Default: 1</i> <i>Must be zero or greater</i> <i>Can not be changed once set</i> <i>Scaled with current BPM value</i></p>
attack_level:	<p>Amplitude level reached after attack phase and immediately before decay phase</p> <p><i>Default: 1</i> <i>Must be zero or greater</i> <i>Can not be changed once set</i></p>
decay_level:	<p>Amplitude level reached after decay phase and immediately before sustain phase. Defaults to sustain_level unless explicitly set</p> <p><i>Default: sustain_level</i> <i>Must be zero or greater</i> <i>Can not be changed once set</i></p>

<b>sustain_level:</b>	Amplitude level reached after decay phase and immediately before release phase.  <i>Default: 1</i> <i>Must be zero or greater</i> <i>Can not be changed once set</i>
<b>env_curve:</b>	Select the shape of the curve between levels in the envelope. 1=linear, 2=exponential, 3=sine, 4=welch, 6=squared, 7=cubed  <i>Default: 2</i> <i>Must be one of the following values: [1, 2, 3, 4, 6, 7]</i> <i>Can not be changed once set</i>
<b>cutoff:</b>	MIDI note representing the highest frequencies allowed to be present in the sound. A low value like 30 makes the sound round and dull, a high value like 100 makes the sound buzzy and crispy.  <i>Default: 100</i> <i>Must be zero or greater, must be a value less than 131</i> <i>May be changed whilst playing</i> <a href="#">Has slide parameters to shape changes</a>
<b>mod_phase:</b>	Phase duration in beats of oscillations between the two notes. Time it takes to switch between the notes.  <i>Default: 0.25</i> <i>Must be greater than zero</i> <i>May be changed whilst playing</i> <a href="#">Has slide parameters to shape changes</a> <i>Scaled with current BPM value</i>
<b>mod_range:</b>	The size of gap between modulation notes. A gap of 12 is one octave.  <i>Default: 5</i> <i>May be changed whilst playing</i> <a href="#">Has slide parameters to shape changes</a>
<b>mod_pulse_width:</b>	The width of the modulated pulse wave as a value between 0 and 1. A width of 0.5 will produce a square wave. Only valid if mod wave is type pulse.  <i>Default: 0.5</i> <i>Must be a value between 0 and 1 exclusively</i> <i>May be changed whilst playing</i> <a href="#">Has slide parameters to shape changes</a>
<b>mod_phase_offset:</b>	Initial modulation phase offset (a value between 0 and 1).  <i>Default: 0</i> <i>Must be a value between 0 and 1 inclusively</i> <i>Can not be changed once set</i>
<b>mod_invert_wave:</b>	Invert mod waveform (i.e. flip it on the y axis). 0=normal wave, 1=inverted wave.  <i>Default: 0</i> <i>Must be one of the following values: [0, 1]</i> <i>May be changed whilst playing</i>
<b>mod_wave:</b>	Wave shape of mod wave. 0=saw wave, 1=pulse, 2=triangle wave and 3=sine wave.  <i>Default: 1</i> <i>Must be one of the following values: [0, 1, 2, 3]</i> <i>May be changed whilst playing</i>

## Slide Parameters

Any parameter that is slidable has three additional parameters named `_slide`, `_slide_curve`, and `_slide_shape`. For example, 'amp' is slidable, so you can also set `amp_slide`, `amp_slide_curve`, and `amp_slide_shape` with the following effects:

<b>_slide:</b>	Amount of time (in beats) for the parameter value to change. A long <code>parameter_slide</code> value means that the parameter takes a long time to slide from the previous value to the new value. A <code>parameter_slide</code> of 0 means that the parameter instantly changes to the new value.  <i>Default: 0</i>
<b>_slide_shape:</b>	Shape of curve. 0: step, 1: linear, 3: sine, 4: welch, 5: custom (use <code>*_slide_curve</code> : opt e.g. <code>amp_slide_curve</code> ); 6: squared, 7: cubed.  <i>Default: 5</i>
<b>_slide_curve:</b>	Shape of the slide curve (only honoured if slide shape is 5). 0 means linear and positive and negative numbers curve the segment up and down respectively.  <i>Default: 0</i>

# Modulated Detuned Saw Waves

note:	52	amp:	1	pan:	0	attack:	0	decay:	0	sustain:	0
release:	1	attack_level:	1	decay_level:	sustain_level	sustain_level:	1	env_curve:	2	cutoff:	100
mod_phase:	0.25	mod_range:	5	mod_pulse_width:	0.5	mod_phase_offset:	0	mod_invert_wave:	0	mod_wave:	1
detune:	0.1										

`use_synth :mod_dsaw`

A pair of detuned saw waves (see the dsaw synth) which are modulated between two fixed notes at a given rate.

Introduced in v2.0

## Parameters

note:	<p>Note to play. Either a MIDI number or a symbol representing a note. For example: <code>30</code>, <code>52</code>, <code>:C</code>, <code>:C2</code>, <code>:Eb4</code>, or <code>:Ds3</code></p> <p><i>Default: 52</i> <i>Must be zero or greater</i> <i>May be changed whilst playing</i> <i>Has slide parameters to shape changes</i></p>
amp:	<p>The amplitude of the sound. Typically a value between 0 and 1. Higher amplitudes may be used, but won't make the sound louder, they will just reduce the quality of all the sounds currently being played (due to compression.)</p> <p><i>Default: 1</i> <i>Must be zero or greater</i> <i>May be changed whilst playing</i> <i>Has slide parameters to shape changes</i></p>
pan:	<p>Position of sound in stereo. With headphones on, this means how much of the sound is in the left ear, and how much is in the right ear. With a value of -1, the sound is completely in the left ear, a value of 0 puts the sound equally in both ears and a value of 1 puts the sound in the right ear. Values in between -1 and 1 move the sound accordingly.</p> <p><i>Default: 0</i> <i>Must be a value between -1 and 1 inclusively</i> <i>May be changed whilst playing</i> <i>Has slide parameters to shape changes</i></p>
attack:	<p>Amount of time (in beats) for sound to reach full amplitude (attack_level). A short attack (i.e. 0.01) makes the initial part of the sound very percussive like a sharp tap. A longer attack (i.e. 1) fades the sound in gently. Full length of sound is attack + decay + sustain + release.</p> <p><i>Default: 0</i> <i>Must be zero or greater</i> <i>Can not be changed once set</i> <i>Scaled with current BPM value</i></p>
decay:	<p>Amount of time (in beats) for the sound to move from full amplitude (attack_level) to the sustain amplitude (sustain_level).</p> <p><i>Default: 0</i> <i>Must be zero or greater</i> <i>Can not be changed once set</i> <i>Scaled with current BPM value</i></p>
sustain:	<p>Amount of time (in beats) for sound to remain at sustain level amplitude. Longer sustain values result in longer sounds. Full length of sound is attack + decay + sustain + release.</p> <p><i>Default: 0</i> <i>Must be zero or greater</i> <i>Can not be changed once set</i> <i>Scaled with current BPM value</i></p>
release:	<p>Amount of time (in beats) for sound to move from sustain level amplitude to silent. A short release (i.e. 0.01) makes the final part of the sound very percussive (potentially resulting in a click). A longer release (i.e. 1) fades the sound out gently. Full length of sound is attack + decay + sustain + release.</p> <p><i>Default: 1</i> <i>Must be zero or greater</i> <i>Can not be changed once set</i> <i>Scaled with current BPM value</i></p>
attack_level:	<p>Amplitude level reached after attack phase and immediately before decay phase</p> <p><i>Default: 1</i> <i>Must be zero or greater</i> <i>Can not be changed once set</i></p>
decay_level:	<p>Amplitude level reached after decay phase and immediately before sustain phase. Defaults to sustain_level unless explicitly set</p> <p><i>Default: sustain_level</i> <i>Must be zero or greater</i> <i>Can not be changed once set</i></p>

<b>sustain_level:</b>	Amplitude level reached after decay phase and immediately before release phase.  <i>Default: 1</i> <i>Must be zero or greater</i> <i>Can not be changed once set</i>
<b>env_curve:</b>	Select the shape of the curve between levels in the envelope. 1=linear, 2=exponential, 3=sine, 4=welch, 6=squared, 7=cubed  <i>Default: 2</i> <i>Must be one of the following values: [1, 2, 3, 4, 6, 7]</i> <i>Can not be changed once set</i>
<b>cutoff:</b>	MIDI note representing the highest frequencies allowed to be present in the sound. A low value like 30 makes the sound round and dull, a high value like 100 makes the sound buzzy and crispy.  <i>Default: 100</i> <i>Must be zero or greater, must be a value less than 131</i> <i>May be changed whilst playing</i> <a href="#">Has slide parameters to shape changes</a>
<b>mod_phase:</b>	Phase duration in beats of oscillations between the two notes. Time it takes to switch between the notes.  <i>Default: 0.25</i> <i>Must be greater than zero</i> <i>May be changed whilst playing</i> <a href="#">Has slide parameters to shape changes</a> <i>Scaled with current BPM value</i>
<b>mod_range:</b>	The size of gap between modulation notes. A gap of 12 is one octave.  <i>Default: 5</i> <i>May be changed whilst playing</i> <a href="#">Has slide parameters to shape changes</a>
<b>mod_pulse_width:</b>	The width of the modulated pulse wave as a value between 0 and 1. A width of 0.5 will produce a square wave. Only valid if mod wave is type pulse.  <i>Default: 0.5</i> <i>Must be a value between 0 and 1 exclusively</i> <i>May be changed whilst playing</i> <a href="#">Has slide parameters to shape changes</a>
<b>mod_phase_offset:</b>	Initial modulation phase offset (a value between 0 and 1).  <i>Default: 0</i> <i>Must be a value between 0 and 1 inclusively</i> <i>Can not be changed once set</i>
<b>mod_invert_wave:</b>	Invert mod waveform (i.e. flip it on the y axis). 0=normal wave, 1=inverted wave.  <i>Default: 0</i> <i>Must be one of the following values: [0, 1]</i> <i>May be changed whilst playing</i>
<b>mod_wave:</b>	Wave shape of mod wave. 0=saw wave, 1=pulse, 2=triangle wave and 3=sine wave.  <i>Default: 1</i> <i>Must be one of the following values: [0, 1, 2, 3]</i> <i>May be changed whilst playing</i>
<b>detune:</b>	Distance (in MIDI notes) between components of sound. Affects thickness, sense of tuning and harmony. Tiny values such as 0.1 create a thick sound. Larger values such as 0.5 make the tuning sound strange. Even bigger values such as 5 create chord-like sounds.  <i>Default: 0.1</i> <i>May be changed whilst playing</i> <a href="#">Has slide parameters to shape changes</a>

## Slide Parameters

Any parameter that is slidable has three additional parameters named `_slide`, `_slide_curve`, and `_slide_shape`. For example, 'amp' is slidable, so you can also set `amp_slide`, `amp_slide_curve`, and `amp_slide_shape` with the following effects:

<b>_slide:</b>	Amount of time (in beats) for the parameter value to change. A long parameter_slide value means that the parameter takes a long time to slide from the previous value to the new value. A parameter_slide of 0 means that the parameter instantly changes to the new value.  <i>Default: 0</i>
<b>_slide_shape:</b>	Shape of curve. 0: step, 1: linear, 3: sine, 4: welch, 5: custom (use <code>*_slide_curve</code> : opt e.g. <code>amp_slide_curve</code> ); 6: squared, 7: cubed.  <i>Default: 5</i>
	Shape of the slide curve (only honoured if slide shape is 5). 0 means linear and positive and negative numbers curve the segment

`_slide_curve:` up and down respectively.

*Default: 0*

---

# Basic FM synthesis with frequency modulation.

note:	52	amp:	1	pan:	0	attack:	0	decay:	0	sustain:	
release:	1	attack_level:	1	decay_level:	sustain_level	sustain_level:	1	env_curve:	2	cutoff:	
divisor:	2	depth:	1	mod_phase:	0.25	mod_range:	5	mod_pulse_width:	0.5	mod_phase_offset:	
mod_invert_wave:	0	mod_wave:	1								

`use_synth :mod_fm`

The FM synth modulating between two notes - the duration of the modulation can be modified using the `mod_phase` arg, the range (number of notes jumped between) by the `mod_range` arg and the width of the jumps by the `mod_width` param. The FM synth is a sine wave with a fundamental frequency which is modulated at audio rate by another sine wave with a specific modulation, division and depth. Useful for generating a wide range of sounds by playing with the `:divisor` and `:depth` params. Great for deep powerful bass and crazy 70s sci-fi sounds.

Introduced in v2.0

## Parameters

<b>note:</b>	Note to play. Either a MIDI number or a symbol representing a note. For example: <code>30</code> , <code>52</code> , <code>:C</code> , <code>:C2</code> , <code>:Eb4</code> , or <code>:Ds3</code>  <i>Default: 52</i> <i>Must be zero or greater</i> <i>May be changed whilst playing</i> <i>Has slide parameters to shape changes</i>
<b>amp:</b>	The amplitude of the sound. Typically a value between 0 and 1. Higher amplitudes may be used, but won't make the sound louder, they will just reduce the quality of all the sounds currently being played (due to compression.)  <i>Default: 1</i> <i>Must be zero or greater</i> <i>May be changed whilst playing</i> <i>Has slide parameters to shape changes</i>
<b>pan:</b>	Position of sound in stereo. With headphones on, this means how much of the sound is in the left ear, and how much is in the right ear. With a value of -1, the sound is completely in the left ear, a value of 0 puts the sound equally in both ears and a value of 1 puts the sound in the right ear. Values in between -1 and 1 move the sound accordingly.  <i>Default: 0</i> <i>Must be a value between -1 and 1 inclusively</i> <i>May be changed whilst playing</i> <i>Has slide parameters to shape changes</i>
<b>attack:</b>	Amount of time (in beats) for sound to reach full amplitude ( <code>attack_level</code> ). A short attack (i.e. 0.01) makes the initial part of the sound very percussive like a sharp tap. A longer attack (i.e 1) fades the sound in gently. Full length of sound is attack + decay + sustain + release.  <i>Default: 0</i> <i>Must be zero or greater</i> <i>Can not be changed once set</i> <i>Scaled with current BPM value</i>
<b>decay:</b>	Amount of time (in beats) for the sound to move from full amplitude ( <code>attack_level</code> ) to the sustain amplitude ( <code>sustain_level</code> ).  <i>Default: 0</i> <i>Must be zero or greater</i> <i>Can not be changed once set</i> <i>Scaled with current BPM value</i>
<b>sustain:</b>	Amount of time (in beats) for sound to remain at sustain level amplitude. Longer sustain values result in longer sounds. Full length of sound is attack + decay + sustain + release.  <i>Default: 0</i> <i>Must be zero or greater</i> <i>Can not be changed once set</i> <i>Scaled with current BPM value</i>
<b>release:</b>	Amount of time (in beats) for sound to move from sustain level amplitude to silent. A short release (i.e. 0.01) makes the final part of the sound very percussive (potentially resulting in a click). A longer release (i.e 1) fades the sound out gently. Full length of sound is attack + decay + sustain + release.  <i>Default: 1</i> <i>Must be zero or greater</i> <i>Can not be changed once set</i> <i>Scaled with current BPM value</i>
<b>attack_level:</b>	Amplitude level reached after attack phase and immediately before decay phase  <i>Default: 1</i> <i>Must be zero or greater</i> <i>Can not be changed once set</i>
	Amplitude level reached after decay phase and immediately before sustain phase. Defaults to <code>sustain_level</code> unless explicitly



<b>decay_level:</b>	set <i>Default: sustain_level</i> <i>Must be zero or greater</i> <i>Can not be changed once set</i>
<b>sustain_level:</b>	Amplitude level reached after decay phase and immediately before release phase. <i>Default: 1</i> <i>Must be zero or greater</i> <i>Can not be changed once set</i>
<b>env_curve:</b>	Select the shape of the curve between levels in the envelope. 1=linear, 2=exponential, 3=sine, 4=welch, 6=squared, 7=cubed <i>Default: 2</i> <i>Must be one of the following values: [1, 2, 3, 4, 6, 7]</i> <i>Can not be changed once set</i>
<b>cutoff:</b>	MIDI note representing the highest frequencies allowed to be present in the sound. A low value like 30 makes the sound round and dull, a high value like 100 makes the sound buzzy and crispy. <i>Default: 100</i> <i>Must be zero or greater, must be a value less than 131</i> <i>May be changed whilst playing</i> <a href="#">Has slide parameters to shape changes</a>
<b>divisor:</b>	Modifies the frequency of the modulator oscillator relative to the carrier. Don't worry too much about what this means - just try different numbers out! <i>Default: 2</i> <i>May be changed whilst playing</i> <a href="#">Has slide parameters to shape changes</a>
<b>depth:</b>	Modifies the depth of the carrier wave used to modify fundamental frequency. Don't worry too much about what this means - just try different numbers out! <i>Default: 1</i> <i>May be changed whilst playing</i> <a href="#">Has slide parameters to shape changes</a>
<b>mod_phase:</b>	Phase duration in beats of oscillations between the two notes. Time it takes to switch between the notes. <i>Default: 0.25</i> <i>Must be greater than zero</i> <i>May be changed whilst playing</i> <i>Scaled with current BPM value</i>
<b>mod_range:</b>	The size of gap between modulation notes. A gap of 12 is one octave. <i>Default: 5</i> <i>May be changed whilst playing</i>
<b>mod_pulse_width:</b>	The width of the modulated pulse wave as a value between 0 and 1. A width of 0.5 will produce a square wave. Only valid if mod wave is type pulse. <i>Default: 0.5</i> <i>Must be a value between 0 and 1 exclusively</i> <i>May be changed whilst playing</i>
<b>mod_phase_offset:</b>	Initial modulation phase offset (a value between 0 and 1). <i>Default: 0</i> <i>Must be a value between 0 and 1 inclusively</i> <i>Can not be changed once set</i>
<b>mod_invert_wave:</b>	Invert mod waveform (i.e. flip it on the y axis). 0=normal wave, 1=inverted wave. <i>Default: 0</i> <i>Must be one of the following values: [0, 1]</i> <i>May be changed whilst playing</i>
<b>mod_wave:</b>	Wave shape of mod wave. 0=saw wave, 1=pulse, 2=triangle wave and 3=sine wave. <i>Default: 1</i> <i>Must be one of the following values: [0, 1, 2, 3]</i> <i>May be changed whilst playing</i>

## Slide Parameters

Any parameter that is slidable has three additional parameters named `_slide`, `_slide_curve`, and `_slide_shape`. For example, 'amp' is slidable, so you can also set `amp_slide`, `amp_slide_curve`, and `amp_slide_shape` with the following effects:

<b>_slide:</b>	Amount of time (in beats) for the parameter value to change. A long <code>parameter_slide</code> value means that the parameter takes a long time to slide from the previous value to the new value. A <code>parameter_slide</code> of 0 means that the parameter instantly changes to the new value.
----------------	---

	<i>Default: 0</i>
<b>_slide_shape:</b>	Shape of curve. 0: step, 1: linear, 3: sine, 4: welch, 5: custom (use *_slide_curve: opt e.g. amp_slide_curve:), 6: squared, 7: cubed. <i>Default: 5</i>
<b>_slide_curve:</b>	Shape of the slide curve (only honoured if slide shape is 5). 0 means linear and positive and negative numbers curve the segment up and down respectively. <i>Default: 0</i>

---

# Modulated Pulse

note:	52	amp:	1	pan:	0	attack:	0	decay:	0	sustain:	0
release:	1	attack_level:	1	decay_level:	sustain_level	sustain_level:	1	env_curve:	2	cutoff:	100
mod_phase:	0.25	mod_range:	5	mod_pulse_width:	0.5	mod_phase_offset:	0	mod_invert_wave:	0	mod_wave:	1
pulse_width:	0.5										

`use_synth :mod_pulse`

A pulse wave with a low pass filter modulating between two notes via a variety of control waves (see `mod_wave`: arg). The pulse wave defaults to a square wave, but the timbre can be changed dramatically by adjusting the `pulse_width` arg between 0 and 1.

Introduced in v2.0

## Parameters

note:	Note to play. Either a MIDI number or a symbol representing a note. For example: <code>30</code> , <code>52</code> , <code>:C</code> , <code>:C2</code> , <code>:Eb4</code> , or <code>:Ds3</code> <i>Default: 52</i> <i>Must be zero or greater</i> <i>May be changed whilst playing</i> <i>Has slide parameters to shape changes</i>
amp:	The amplitude of the sound. Typically a value between 0 and 1. Higher amplitudes may be used, but won't make the sound louder, they will just reduce the quality of all the sounds currently being played (due to compression.) <i>Default: 1</i> <i>Must be zero or greater</i> <i>May be changed whilst playing</i> <i>Has slide parameters to shape changes</i>
pan:	Position of sound in stereo. With headphones on, this means how much of the sound is in the left ear, and how much is in the right ear. With a value of -1, the sound is completely in the left ear, a value of 0 puts the sound equally in both ears and a value of 1 puts the sound in the right ear. Values in between -1 and 1 move the sound accordingly. <i>Default: 0</i> <i>Must be a value between -1 and 1 inclusively</i> <i>May be changed whilst playing</i> <i>Has slide parameters to shape changes</i>
attack:	Amount of time (in beats) for sound to reach full amplitude ( <code>attack_level</code> ). A short attack (i.e. 0.01) makes the initial part of the sound very percussive like a sharp tap. A longer attack (i.e 1) fades the sound in gently. Full length of sound is <code>attack</code> + <code>decay</code> + <code>sustain</code> + <code>release</code> . <i>Default: 0</i> <i>Must be zero or greater</i> <i>Can not be changed once set</i> <i>Scaled with current BPM value</i>
decay:	Amount of time (in beats) for the sound to move from full amplitude ( <code>attack_level</code> ) to the sustain amplitude ( <code>sustain_level</code> ). <i>Default: 0</i> <i>Must be zero or greater</i> <i>Can not be changed once set</i> <i>Scaled with current BPM value</i>
sustain:	Amount of time (in beats) for sound to remain at sustain level amplitude. Longer sustain values result in longer sounds. Full length of sound is <code>attack</code> + <code>decay</code> + <code>sustain</code> + <code>release</code> . <i>Default: 0</i> <i>Must be zero or greater</i> <i>Can not be changed once set</i> <i>Scaled with current BPM value</i>
release:	Amount of time (in beats) for sound to move from sustain level amplitude to silent. A short release (i.e. 0.01) makes the final part of the sound very percussive (potentially resulting in a click). A longer release (i.e 1) fades the sound out gently. Full length of sound is <code>attack</code> + <code>decay</code> + <code>sustain</code> + <code>release</code> . <i>Default: 1</i> <i>Must be zero or greater</i> <i>Can not be changed once set</i> <i>Scaled with current BPM value</i>
attack_level:	Amplitude level reached after attack phase and immediately before decay phase <i>Default: 1</i> <i>Must be zero or greater</i> <i>Can not be changed once set</i>
decay_level:	Amplitude level reached after decay phase and immediately before sustain phase. Defaults to <code>sustain_level</code> unless explicitly set <i>Default: sustain_level</i>

	<p>Must be zero or greater Can not be changed once set</p>
sustain_level:	<p>Amplitude level reached after decay phase and immediately before release phase.</p> <p>Default: 1 Must be zero or greater Can not be changed once set</p>
env_curve:	<p>Select the shape of the curve between levels in the envelope. 1=linear, 2=exponential, 3=sine, 4=welch, 6=squared, 7=cubed</p> <p>Default: 2 Must be one of the following values: [1, 2, 3, 4, 6, 7] Can not be changed once set</p>
cutoff:	<p>MIDI note representing the highest frequencies allowed to be present in the sound. A low value like 30 makes the sound round and dull, a high value like 100 makes the sound buzzy and crispy.</p> <p>Default: 100 Must be zero or greater, must be a value less than 131 May be changed whilst playing <a href="#">Has slide parameters to shape changes</a></p>
mod_phase:	<p>Phase duration in beats of oscillations between the two notes. Time it takes to switch between the notes.</p> <p>Default: 0.25 Must be greater than zero May be changed whilst playing <a href="#">Has slide parameters to shape changes</a> Scaled with current BPM value</p>
mod_range:	<p>The size of gap between modulation notes. A gap of 12 is one octave.</p> <p>Default: 5 May be changed whilst playing <a href="#">Has slide parameters to shape changes</a></p>
mod_pulse_width:	<p>The width of the modulated pulse wave as a value between 0 and 1. A width of 0.5 will produce a square wave. Only valid if mod wave is type pulse.</p> <p>Default: 0.5 Must be a value between 0 and 1 exclusively May be changed whilst playing <a href="#">Has slide parameters to shape changes</a></p>
mod_phase_offset:	<p>Initial modulation phase offset (a value between 0 and 1).</p> <p>Default: 0 Must be a value between 0 and 1 inclusively Can not be changed once set</p>
mod_invert_wave:	<p>Invert mod waveform (i.e. flip it on the y axis). 0=normal wave, 1=inverted wave.</p> <p>Default: 0 Must be one of the following values: [0, 1] May be changed whilst playing</p>
mod_wave:	<p>Wave shape of mod wave. 0=saw wave, 1=pulse, 2=triangle wave and 3=sine wave.</p> <p>Default: 1 Must be one of the following values: [0, 1, 2, 3] May be changed whilst playing</p>
pulse_width:	<p>The width of the pulse wave as a value between 0 and 1. A width of 0.5 will produce a square wave. Different values will change the timbre of the sound. Only valid if wave is type pulse.</p> <p>Default: 0.5 Must be a value between 0 and 1 exclusively May be changed whilst playing <a href="#">Has slide parameters to shape changes</a></p>

## Slide Parameters

Any parameter that is slidable has three additional parameters named `_slide`, `_slide_curve`, and `_slide_shape`. For example, 'amp' is slidable, so you can also set `amp_slide`, `amp_slide_curve`, and `amp_slide_shape` with the following effects:

_slide:	<p>Amount of time (in beats) for the parameter value to change. A long <code>parameter_slide</code> value means that the parameter takes a long time to slide from the previous value to the new value. A <code>parameter_slide</code> of 0 means that the parameter instantly changes to the new value.</p> <p>Default: 0</p>
_slide_shape:	<p>Shape of curve. 0: step, 1: linear, 3: sine, 4: welch, 5: custom (use <code>*_slide_curve</code>: opt e.g. <code>amp_slide_curve</code>); 6: squared, 7: cubed.</p> <p>Default: 5</p>

`_slide_curve:`

Shape of the slide curve (only honoured if slide shape is 5). 0 means linear and positive and negative numbers curve the segment up and down respectively.

*Default: 0*

---

# Modulated Saw Wave

note:	52	amp:	1	pan:	0	attack:	0	decay:	0	sustain:	0
release:	1	attack_level:	1	decay_level:	sustain_level	sustain_level:	1	env_curve:	2	cutoff:	100
mod_phase:	0.25	mod_range:	5	mod_pulse_width:	0.5	mod_phase_offset:	0	mod_invert_wave:	0	mod_wave:	1

`use_synth :mod_saw`

A saw wave passed through a low pass filter which modulates between two separate notes via a variety of control waves.

Introduced in v2.0

## Parameters

note:	<p>Note to play. Either a MIDI number or a symbol representing a note. For example:<b>30</b>, <b>52</b>, <b>:C</b>, <b>:C2</b>, <b>:Eb4</b>, or <b>:Ds3</b></p> <p><i>Default: 52</i> <i>Must be zero or greater</i> <i>May be changed whilst playing</i> <i>Has slide parameters to shape changes</i></p>
amp:	<p>The amplitude of the sound. Typically a value between 0 and 1. Higher amplitudes may be used, but won't make the sound louder, they will just reduce the quality of all the sounds currently being played (due to compression.)</p> <p><i>Default: 1</i> <i>Must be zero or greater</i> <i>May be changed whilst playing</i> <i>Has slide parameters to shape changes</i></p>
pan:	<p>Position of sound in stereo. With headphones on, this means how much of the sound is in the left ear, and how much is in the right ear. With a value of -1, the sound is completely in the left ear, a value of 0 puts the sound equally in both ears and a value of 1 puts the sound in the right ear. Values in between -1 and 1 move the sound accordingly.</p> <p><i>Default: 0</i> <i>Must be a value between -1 and 1 inclusively</i> <i>May be changed whilst playing</i> <i>Has slide parameters to shape changes</i></p>
attack:	<p>Amount of time (in beats) for sound to reach full amplitude (attack_level). A short attack (i.e. 0.01) makes the initial part of the sound very percussive like a sharp tap. A longer attack (i.e. 1) fades the sound in gently. Full length of sound is attack + decay + sustain + release.</p> <p><i>Default: 0</i> <i>Must be zero or greater</i> <i>Can not be changed once set</i> <i>Scaled with current BPM value</i></p>
decay:	<p>Amount of time (in beats) for the sound to move from full amplitude (attack_level) to the sustain amplitude (sustain_level).</p> <p><i>Default: 0</i> <i>Must be zero or greater</i> <i>Can not be changed once set</i> <i>Scaled with current BPM value</i></p>
sustain:	<p>Amount of time (in beats) for sound to remain at sustain level amplitude. Longer sustain values result in longer sounds. Full length of sound is attack + decay + sustain + release.</p> <p><i>Default: 0</i> <i>Must be zero or greater</i> <i>Can not be changed once set</i> <i>Scaled with current BPM value</i></p>
release:	<p>Amount of time (in beats) for sound to move from sustain level amplitude to silent. A short release (i.e. 0.01) makes the final part of the sound very percussive (potentially resulting in a click). A longer release (i.e. 1) fades the sound out gently. Full length of sound is attack + decay + sustain + release.</p> <p><i>Default: 1</i> <i>Must be zero or greater</i> <i>Can not be changed once set</i> <i>Scaled with current BPM value</i></p>
attack_level:	<p>Amplitude level reached after attack phase and immediately before decay phase</p> <p><i>Default: 1</i> <i>Must be zero or greater</i> <i>Can not be changed once set</i></p>
decay_level:	<p>Amplitude level reached after decay phase and immediately before sustain phase. Defaults to sustain_level unless explicitly set</p> <p><i>Default: sustain_level</i> <i>Must be zero or greater</i> <i>Can not be changed once set</i></p>

<b>sustain_level:</b>	Amplitude level reached after decay phase and immediately before release phase.  <i>Default: 1</i> <i>Must be zero or greater</i> <i>Can not be changed once set</i>
<b>env_curve:</b>	Select the shape of the curve between levels in the envelope. 1=linear, 2=exponential, 3=sine, 4=welch, 6=squared, 7=cubed  <i>Default: 2</i> <i>Must be one of the following values: [1, 2, 3, 4, 6, 7]</i> <i>Can not be changed once set</i>
<b>cutoff:</b>	MIDI note representing the highest frequencies allowed to be present in the sound. A low value like 30 makes the sound round and dull, a high value like 100 makes the sound buzzy and crispy.  <i>Default: 100</i> <i>Must be zero or greater, must be a value less than 131</i> <i>May be changed whilst playing</i> <a href="#">Has slide parameters to shape changes</a>
<b>mod_phase:</b>	Phase duration in beats of oscillations between the two notes. Time it takes to switch between the notes.  <i>Default: 0.25</i> <i>Must be greater than zero</i> <i>May be changed whilst playing</i> <a href="#">Has slide parameters to shape changes</a> <i>Scaled with current BPM value</i>
<b>mod_range:</b>	The size of gap between modulation notes. A gap of 12 is one octave.  <i>Default: 5</i> <i>May be changed whilst playing</i> <a href="#">Has slide parameters to shape changes</a>
<b>mod_pulse_width:</b>	The width of the modulated pulse wave as a value between 0 and 1. A width of 0.5 will produce a square wave. Only valid if mod wave is type pulse.  <i>Default: 0.5</i> <i>Must be a value between 0 and 1 exclusively</i> <i>May be changed whilst playing</i> <a href="#">Has slide parameters to shape changes</a>
<b>mod_phase_offset:</b>	Initial modulation phase offset (a value between 0 and 1).  <i>Default: 0</i> <i>Must be a value between 0 and 1 inclusively</i> <i>Can not be changed once set</i>
<b>mod_invert_wave:</b>	Invert mod waveform (i.e. flip it on the y axis). 0=normal wave, 1=inverted wave.  <i>Default: 0</i> <i>Must be one of the following values: [0, 1]</i> <i>May be changed whilst playing</i>
<b>mod_wave:</b>	Wave shape of mod wave. 0=saw wave, 1=pulse, 2=triangle wave and 3=sine wave.  <i>Default: 1</i> <i>Must be one of the following values: [0, 1, 2, 3]</i> <i>May be changed whilst playing</i>

## Slide Parameters

Any parameter that is slidable has three additional parameters named `_slide`, `_slide_curve`, and `_slide_shape`. For example, 'amp' is slidable, so you can also set `amp_slide`, `amp_slide_curve`, and `amp_slide_shape` with the following effects:

<b>_slide:</b>	Amount of time (in beats) for the parameter value to change. A long <code>parameter_slide</code> value means that the parameter takes a long time to slide from the previous value to the new value. A <code>parameter_slide</code> of 0 means that the parameter instantly changes to the new value.  <i>Default: 0</i>
<b>_slide_shape:</b>	Shape of curve. 0: step, 1: linear, 3: sine, 4: welch, 5: custom (use <code>*_slide_curve</code> : opt e.g. <code>amp_slide_curve</code> ); 6: squared, 7: cubed.  <i>Default: 5</i>
<b>_slide_curve:</b>	Shape of the slide curve (only honoured if slide shape is 5). 0 means linear and positive and negative numbers curve the segment up and down respectively.  <i>Default: 0</i>

# Modulated Sine Wave

note:	52	amp:	1	pan:	0	attack:	0	decay:	0	sustain:	0
release:	1	attack_level:	1	decay_level:	sustain_level	sustain_level:	1	env_curve:	2	cutoff:	100
mod_phase:	0.25	mod_range:	5	mod_pulse_width:	0.5	mod_phase_offset:	0	mod_invert_wave:	0	mod_wave:	1

`use_synth :mod_sine`

A sine wave passed through a low pass filter which modulates between two separate notes via a variety of control waves.

Introduced in v2.0

## Parameters

note:	<p>Note to play. Either a MIDI number or a symbol representing a note. For example: <b>30</b>, <b>52</b>, <b>:C</b>, <b>:C2</b>, <b>:Eb4</b>, or <b>:Ds3</b></p> <p><i>Default: 52</i> <i>Must be zero or greater</i> <i>May be changed whilst playing</i> <i>Has slide parameters to shape changes</i></p>
amp:	<p>The amplitude of the sound. Typically a value between 0 and 1. Higher amplitudes may be used, but won't make the sound louder, they will just reduce the quality of all the sounds currently being played (due to compression.)</p> <p><i>Default: 1</i> <i>Must be zero or greater</i> <i>May be changed whilst playing</i> <i>Has slide parameters to shape changes</i></p>
pan:	<p>Position of sound in stereo. With headphones on, this means how much of the sound is in the left ear, and how much is in the right ear. With a value of -1, the sound is completely in the left ear, a value of 0 puts the sound equally in both ears and a value of 1 puts the sound in the right ear. Values in between -1 and 1 move the sound accordingly.</p> <p><i>Default: 0</i> <i>Must be a value between -1 and 1 inclusively</i> <i>May be changed whilst playing</i> <i>Has slide parameters to shape changes</i></p>
attack:	<p>Amount of time (in beats) for sound to reach full amplitude (attack_level). A short attack (i.e. 0.01) makes the initial part of the sound very percussive like a sharp tap. A longer attack (i.e. 1) fades the sound in gently. Full length of sound is attack + decay + sustain + release.</p> <p><i>Default: 0</i> <i>Must be zero or greater</i> <i>Can not be changed once set</i> <i>Scaled with current BPM value</i></p>
decay:	<p>Amount of time (in beats) for the sound to move from full amplitude (attack_level) to the sustain amplitude (sustain_level).</p> <p><i>Default: 0</i> <i>Must be zero or greater</i> <i>Can not be changed once set</i> <i>Scaled with current BPM value</i></p>
sustain:	<p>Amount of time (in beats) for sound to remain at sustain level amplitude. Longer sustain values result in longer sounds. Full length of sound is attack + decay + sustain + release.</p> <p><i>Default: 0</i> <i>Must be zero or greater</i> <i>Can not be changed once set</i> <i>Scaled with current BPM value</i></p>
release:	<p>Amount of time (in beats) for sound to move from sustain level amplitude to silent. A short release (i.e. 0.01) makes the final part of the sound very percussive (potentially resulting in a click). A longer release (i.e. 1) fades the sound out gently. Full length of sound is attack + decay + sustain + release.</p> <p><i>Default: 1</i> <i>Must be zero or greater</i> <i>Can not be changed once set</i> <i>Scaled with current BPM value</i></p>
attack_level:	<p>Amplitude level reached after attack phase and immediately before decay phase</p> <p><i>Default: 1</i> <i>Must be zero or greater</i> <i>Can not be changed once set</i></p>
decay_level:	<p>Amplitude level reached after decay phase and immediately before sustain phase. Defaults to sustain_level unless explicitly set</p> <p><i>Default: sustain_level</i> <i>Must be zero or greater</i> <i>Can not be changed once set</i></p>



<b>sustain_level:</b>	Amplitude level reached after decay phase and immediately before release phase.  <i>Default: 1</i> <i>Must be zero or greater</i> <i>Can not be changed once set</i>
<b>env_curve:</b>	Select the shape of the curve between levels in the envelope. 1=linear, 2=exponential, 3=sine, 4=welch, 6=squared, 7=cubed  <i>Default: 2</i> <i>Must be one of the following values: [1, 2, 3, 4, 6, 7]</i> <i>Can not be changed once set</i>
<b>cutoff:</b>	MIDI note representing the highest frequencies allowed to be present in the sound. A low value like 30 makes the sound round and dull, a high value like 100 makes the sound buzzy and crispy.  <i>Default: 100</i> <i>Must be zero or greater, must be a value less than 131</i> <i>May be changed whilst playing</i> <i>Has slide parameters to shape changes</i>
<b>mod_phase:</b>	Phase duration in beats of oscillations between the two notes. Time it takes to switch between the notes.  <i>Default: 0.25</i> <i>Must be greater than zero</i> <i>May be changed whilst playing</i> <i>Has slide parameters to shape changes</i> <i>Scaled with current BPM value</i>
<b>mod_range:</b>	The size of gap between modulation notes. A gap of 12 is one octave.  <i>Default: 5</i> <i>May be changed whilst playing</i> <i>Has slide parameters to shape changes</i>
<b>mod_pulse_width:</b>	The width of the modulated pulse wave as a value between 0 and 1. A width of 0.5 will produce a square wave. Only valid if mod wave is type pulse.  <i>Default: 0.5</i> <i>Must be a value between 0 and 1 exclusively</i> <i>May be changed whilst playing</i> <i>Has slide parameters to shape changes</i>
<b>mod_phase_offset:</b>	Initial modulation phase offset (a value between 0 and 1).  <i>Default: 0</i> <i>Must be a value between 0 and 1 inclusively</i> <i>Can not be changed once set</i>
<b>mod_invert_wave:</b>	Invert mod waveform (i.e. flip it on the y axis). 0=normal wave, 1=inverted wave.  <i>Default: 0</i> <i>Must be one of the following values: [0, 1]</i> <i>May be changed whilst playing</i>
<b>mod_wave:</b>	Wave shape of mod wave. 0=saw wave, 1=pulse, 2=triangle wave and 3=sine wave.  <i>Default: 1</i> <i>Must be one of the following values: [0, 1, 2, 3]</i> <i>May be changed whilst playing</i>

## Slide Parameters

Any parameter that is slidable has three additional parameters named `_slide`, `_slide_curve`, and `_slide_shape`. For example, 'amp' is slidable, so you can also set `amp_slide`, `amp_slide_curve`, and `amp_slide_shape` with the following effects:

<b>_slide:</b>	Amount of time (in beats) for the parameter value to change. A long <code>parameter_slide</code> value means that the parameter takes a long time to slide from the previous value to the new value. A <code>parameter_slide</code> of 0 means that the parameter instantly changes to the new value.  <i>Default: 0</i>
<b>_slide_shape:</b>	Shape of curve. 0: step, 1: linear, 3: sine, 4: welch, 5: custom (use <code>*_slide_curve</code> : opt e.g. <code>amp_slide_curve</code> ); 6: squared, 7: cubed.  <i>Default: 5</i>
<b>_slide_curve:</b>	Shape of the slide curve (only honoured if slide shape is 5). 0 means linear and positive and negative numbers curve the segment up and down respectively.  <i>Default: 0</i>

# Modulated Triangle Wave

note:	52	amp:	1	pan:	0	attack:	0	decay:	0	sustain:	0
release:	1	attack_level:	1	decay_level:	sustain_level	sustain_level:	1	env_curve:	2	cutoff:	100
mod_phase:	0.25	mod_range:	5	mod_pulse_width:	0.5	mod_phase_offset:	0	mod_invert_wave:	0	mod_wave:	1

`use_synth :mod_tri`

A triangle wave passed through a low pass filter which modulates between two separate notes via a variety of control waves.

Introduced in v2.0

## Parameters

note:	<p>Note to play. Either a MIDI number or a symbol representing a note. For example: <b>30</b>, <b>52</b>, <b>:C</b>, <b>:C2</b>, <b>:Eb4</b>, or <b>:Ds3</b></p> <p><i>Default: 52</i> <i>Must be zero or greater</i> <i>May be changed whilst playing</i> <i>Has slide parameters to shape changes</i></p>
amp:	<p>The amplitude of the sound. Typically a value between 0 and 1. Higher amplitudes may be used, but won't make the sound louder, they will just reduce the quality of all the sounds currently being played (due to compression.)</p> <p><i>Default: 1</i> <i>Must be zero or greater</i> <i>May be changed whilst playing</i> <i>Has slide parameters to shape changes</i></p>
pan:	<p>Position of sound in stereo. With headphones on, this means how much of the sound is in the left ear, and how much is in the right ear. With a value of -1, the sound is completely in the left ear, a value of 0 puts the sound equally in both ears and a value of 1 puts the sound in the right ear. Values in between -1 and 1 move the sound accordingly.</p> <p><i>Default: 0</i> <i>Must be a value between -1 and 1 inclusively</i> <i>May be changed whilst playing</i> <i>Has slide parameters to shape changes</i></p>
attack:	<p>Amount of time (in beats) for sound to reach full amplitude (attack_level). A short attack (i.e. 0.01) makes the initial part of the sound very percussive like a sharp tap. A longer attack (i.e. 1) fades the sound in gently. Full length of sound is attack + decay + sustain + release.</p> <p><i>Default: 0</i> <i>Must be zero or greater</i> <i>Can not be changed once set</i> <i>Scaled with current BPM value</i></p>
decay:	<p>Amount of time (in beats) for the sound to move from full amplitude (attack_level) to the sustain amplitude (sustain_level).</p> <p><i>Default: 0</i> <i>Must be zero or greater</i> <i>Can not be changed once set</i> <i>Scaled with current BPM value</i></p>
sustain:	<p>Amount of time (in beats) for sound to remain at sustain level amplitude. Longer sustain values result in longer sounds. Full length of sound is attack + decay + sustain + release.</p> <p><i>Default: 0</i> <i>Must be zero or greater</i> <i>Can not be changed once set</i> <i>Scaled with current BPM value</i></p>
release:	<p>Amount of time (in beats) for sound to move from sustain level amplitude to silent. A short release (i.e. 0.01) makes the final part of the sound very percussive (potentially resulting in a click). A longer release (i.e. 1) fades the sound out gently. Full length of sound is attack + decay + sustain + release.</p> <p><i>Default: 1</i> <i>Must be zero or greater</i> <i>Can not be changed once set</i> <i>Scaled with current BPM value</i></p>
attack_level:	<p>Amplitude level reached after attack phase and immediately before decay phase</p> <p><i>Default: 1</i> <i>Must be zero or greater</i> <i>Can not be changed once set</i></p>
decay_level:	<p>Amplitude level reached after decay phase and immediately before sustain phase. Defaults to sustain_level unless explicitly set</p> <p><i>Default: sustain_level</i> <i>Must be zero or greater</i> <i>Can not be changed once set</i></p>

<b>sustain_level:</b>	Amplitude level reached after decay phase and immediately before release phase.  <i>Default: 1</i> <i>Must be zero or greater</i> <i>Can not be changed once set</i>
<b>env_curve:</b>	Select the shape of the curve between levels in the envelope. 1=linear, 2=exponential, 3=sine, 4=welch, 6=squared, 7=cubed  <i>Default: 2</i> <i>Must be one of the following values: [1, 2, 3, 4, 6, 7]</i> <i>Can not be changed once set</i>
<b>cutoff:</b>	MIDI note representing the highest frequencies allowed to be present in the sound. A low value like 30 makes the sound round and dull, a high value like 100 makes the sound buzzy and crispy.  <i>Default: 100</i> <i>Must be zero or greater, must be a value less than 131</i> <i>May be changed whilst playing</i> <a href="#">Has slide parameters to shape changes</a>
<b>mod_phase:</b>	Phase duration in beats of oscillations between the two notes. Time it takes to switch between the notes.  <i>Default: 0.25</i> <i>Must be greater than zero</i> <i>May be changed whilst playing</i> <a href="#">Has slide parameters to shape changes</a> <i>Scaled with current BPM value</i>
<b>mod_range:</b>	The size of gap between modulation notes. A gap of 12 is one octave.  <i>Default: 5</i> <i>May be changed whilst playing</i> <a href="#">Has slide parameters to shape changes</a>
<b>mod_pulse_width:</b>	The width of the modulated pulse wave as a value between 0 and 1. A width of 0.5 will produce a square wave. Only valid if mod wave is type pulse.  <i>Default: 0.5</i> <i>Must be a value between 0 and 1 exclusively</i> <i>May be changed whilst playing</i> <a href="#">Has slide parameters to shape changes</a>
<b>mod_phase_offset:</b>	Initial modulation phase offset (a value between 0 and 1).  <i>Default: 0</i> <i>Must be a value between 0 and 1 inclusively</i> <i>Can not be changed once set</i>
<b>mod_invert_wave:</b>	Invert mod waveform (i.e. flip it on the y axis). 0=normal wave, 1=inverted wave.  <i>Default: 0</i> <i>Must be one of the following values: [0, 1]</i> <i>May be changed whilst playing</i>
<b>mod_wave:</b>	Wave shape of mod wave. 0=saw wave, 1=pulse, 2=triangle wave and 3=sine wave.  <i>Default: 1</i> <i>Must be one of the following values: [0, 1, 2, 3]</i> <i>May be changed whilst playing</i>

## Slide Parameters

Any parameter that is slidable has three additional parameters named `_slide`, `_slide_curve`, and `_slide_shape`. For example, 'amp' is slidable, so you can also set `amp_slide`, `amp_slide_curve`, and `amp_slide_shape` with the following effects:

<b>_slide:</b>	Amount of time (in beats) for the parameter value to change. A long parameter_slide value means that the parameter takes a long time to slide from the previous value to the new value. A parameter_slide of 0 means that the parameter instantly changes to the new value.  <i>Default: 0</i>
<b>_slide_shape:</b>	Shape of curve. 0: step, 1: linear, 3: sine, 4: welch, 5: custom (use *_slide_curve: opt e.g. amp_slide_curve:), 6: squared, 7: cubed.  <i>Default: 5</i>
<b>_slide_curve:</b>	Shape of the slide curve (only honoured if slide shape is 5). 0 means linear and positive and negative numbers curve the segment up and down respectively.  <i>Default: 0</i>

# Noise

amp:	1	pan:	0	attack:	0	decay:	0	sustain:	0	release:	1
attack_level:	1	decay_level:	sustain_level	sustain_level:	1	env_curve:	2	cutoff:	110	res:	0

`use_synth :noise`

Noise that contains equal amounts of energy at every frequency - comparable to radio static. Useful for generating percussive sounds such as snares and hand claps. Also useful for simulating wind or sea effects.

Introduced in v2.0

## Parameters

amp:	<p>The amplitude of the sound. Typically a value between 0 and 1. Higher amplitudes may be used, but won't make the sound louder, they will just reduce the quality of all the sounds currently being played (due to compression.)</p> <p><i>Default: 1</i> <i>Must be zero or greater</i> <i>May be changed whilst playing</i> <i>Has slide parameters to shape changes</i></p>
pan:	<p>Position of sound in stereo. With headphones on, this means how much of the sound is in the left ear, and how much is in the right ear. With a value of -1, the sound is completely in the left ear, a value of 0 puts the sound equally in both ears and a value of 1 puts the sound in the right ear. Values in between -1 and 1 move the sound accordingly.</p> <p><i>Default: 0</i> <i>Must be a value between -1 and 1 inclusively</i> <i>May be changed whilst playing</i> <i>Has slide parameters to shape changes</i></p>
attack:	<p>Amount of time (in beats) for sound to reach full amplitude (attack_level). A short attack (i.e. 0.01) makes the initial part of the sound very percussive like a sharp tap. A longer attack (i.e 1) fades the sound in gently. Full length of sound is attack + decay + sustain + release.</p> <p><i>Default: 0</i> <i>Must be zero or greater</i> <i>Can not be changed once set</i> <i>Scaled with current BPM value</i></p>
decay:	<p>Amount of time (in beats) for the sound to move from full amplitude (attack_level) to the sustain amplitude (sustain_level).</p> <p><i>Default: 0</i> <i>Must be zero or greater</i> <i>Can not be changed once set</i> <i>Scaled with current BPM value</i></p>
sustain:	<p>Amount of time (in beats) for sound to remain at sustain level amplitude. Longer sustain values result in longer sounds. Full length of sound is attack + decay + sustain + release.</p> <p><i>Default: 0</i> <i>Must be zero or greater</i> <i>Can not be changed once set</i> <i>Scaled with current BPM value</i></p>
release:	<p>Amount of time (in beats) for sound to move from sustain level amplitude to silent. A short release (i.e. 0.01) makes the final part of the sound very percussive (potentially resulting in a click). A longer release (i.e 1) fades the sound out gently. Full length of sound is attack + decay + sustain + release.</p> <p><i>Default: 1</i> <i>Must be zero or greater</i> <i>Can not be changed once set</i> <i>Scaled with current BPM value</i></p>
attack_level:	<p>Amplitude level reached after attack phase and immediately before decay phase</p> <p><i>Default: 1</i> <i>Must be zero or greater</i> <i>Can not be changed once set</i></p>
decay_level:	<p>Amplitude level reached after decay phase and immediately before sustain phase. Defaults to sustain_level unless explicitly set</p> <p><i>Default: sustain_level</i> <i>Must be zero or greater</i> <i>Can not be changed once set</i></p>
sustain_level:	<p>Amplitude level reached after decay phase and immediately before release phase.</p> <p><i>Default: 1</i> <i>Must be zero or greater</i> <i>Can not be changed once set</i></p>
	<p>Select the shape of the curve between levels in the envelope. 1=linear, 2=exponential, 3=sine, 4=welch, 6=squared, 7=cubed</p>

<b>env_curve:</b>	<p><i>Default: 2</i>  <i>Must be one of the following values: [1, 2, 3, 4, 6, 7]</i>  <i>Can not be changed once set</i></p>
<b>cutoff:</b>	<p>MIDI note representing the highest frequencies allowed to be present in the sound. A low value like 30 makes the sound round and dull, a high value like 100 makes the sound buzzy and crispy.</p> <p><i>Default: 110</i>  <i>Must be zero or greater, must be a value less than 131</i>  <i>May be changed whilst playing</i>  <i>Has slide parameters to shape changes</i></p>
<b>res:</b>	<p>Filter resonance as a value between 0 and 1. Large amounts of resonance (a res: near 1) can create a whistling sound around the cutoff frequency. Smaller values produce less resonance.</p> <p><i>Default: 0</i>  <i>Must be zero or greater, must be a value less than 1</i>  <i>May be changed whilst playing</i>  <i>Has slide parameters to shape changes</i></p>

## Slide Parameters

Any parameter that is slidable has three additional parameters named `_slide`, `_slide_curve`, and `_slide_shape`. For example, 'amp' is slidable, so you can also set `amp_slide`, `amp_slide_curve`, and `amp_slide_shape` with the following effects:

<b>_slide:</b>	<p>Amount of time (in beats) for the parameter value to change. A long parameter_slide value means that the parameter takes a long time to slide from the previous value to the new value. A parameter_slide of 0 means that the parameter instantly changes to the new value.</p> <p><i>Default: 0</i></p>
<b>_slide_shape:</b>	<p>Shape of curve. 0: step, 1: linear, 3: sine, 4: welch, 5: custom (use *_slide_curve: opt e.g. amp_slide_curve:), 6: squared, 7: cubed.</p> <p><i>Default: 5</i></p>
<b>_slide_curve:</b>	<p>Shape of the slide curve (only honoured if slide shape is 5). 0 means linear and positive and negative numbers curve the segment up and down respectively.</p> <p><i>Default: 0</i></p>

# SynthPiano

note:	52	amp:	1	pan:	0	vel:	0.2	attack:	0	decay:	0
sustain:	0	release:	1	attack_level:	1	decay_level:	sustain_level	sustain_level:	1	hard:	0.5
stereo_width:	0										

`use_synth :piano`

A basic piano synthesiser. Note that due to the plucked nature of this synth the envelope opts such as `attack:`, `sustain:` and `release:` do not work as expected. They can only shorten the natural length of the note, not prolong it. Also, the `note:` opt will only honour whole tones.

Introduced in v2.6

## Parameters

<b>note:</b>	<p>Note to play. Either a MIDI number or a symbol representing a note. For example:<code>30</code>, <code>52</code>, <code>:C</code>, <code>:C2</code>, <code>:Eb4</code>, or <code>:Ds3</code>. Note that the piano synth can only play whole tones such as 60 and does not handle floats such as 60.3</p> <p><i>Default: 52</i> <i>Must be zero or greater</i> <i>May be changed whilst playing</i> <i>Has slide parameters to shape changes</i></p>
<b>amp:</b>	<p>The amplitude of the sound. Typically a value between 0 and 1. Higher amplitudes may be used, but won't make the sound louder, they will just reduce the quality of all the sounds currently being played (due to compression.)</p> <p><i>Default: 1</i> <i>Must be zero or greater</i> <i>May be changed whilst playing</i> <i>Has slide parameters to shape changes</i></p>
<b>pan:</b>	<p>Position of sound in stereo. With headphones on, this means how much of the sound is in the left ear, and how much is in the right ear. With a value of -1, the sound is completely in the left ear, a value of 0 puts the sound equally in both ears and a value of 1 puts the sound in the right ear. Values in between -1 and 1 move the sound accordingly.</p> <p><i>Default: 0</i> <i>Must be a value between -1 and 1 inclusively</i> <i>May be changed whilst playing</i> <i>Has slide parameters to shape changes</i></p>
<b>vel:</b>	<p>Velocity of keypress.</p> <p><i>Default: 0.2</i> <i>Must be a value between 0 and 1 inclusively</i> <i>Can not be changed once set</i></p>
<b>attack:</b>	<p>Amount of time (in beats) for sound to reach full amplitude (<code>attack_level</code>). A short attack (i.e. 0.01) makes the initial part of the sound very percussive like a sharp tap. A longer attack (i.e 1) fades the sound in gently. With the piano synth, this opt can only have the effect of shortening the attack phase, not prolonging it.</p> <p><i>Default: 0</i> <i>Must be zero or greater</i> <i>Can not be changed once set</i> <i>Scaled with current BPM value</i></p>
<b>decay:</b>	<p>Amount of time (in beats) for the sound to move from full amplitude (<code>attack_level</code>) to the sustain amplitude (<code>sustain_level</code>). With the piano synth, this opt can only have the effect of controlling the amp within the natural duration of the note and can not prolong the sound.</p> <p><i>Default: 0</i> <i>Must be zero or greater</i> <i>Can not be changed once set</i> <i>Scaled with current BPM value</i></p>
<b>sustain:</b>	<p>Amount of time (in beats) for sound to remain at sustain level amplitude. Longer sustain values result in longer sounds. With the piano synth, this opt can only have the effect of controlling the amp within the natural duration of the note and can not prolong the sound.</p> <p><i>Default: 0</i> <i>Must be zero or greater</i> <i>Can not be changed once set</i> <i>Scaled with current BPM value</i></p>
<b>release:</b>	<p>Amount of time (in beats) for sound to move from sustain level amplitude to silent. A short release (i.e. 0.01) makes the final part of the sound very percussive (potentially resulting in a click). A longer release (i.e 1) fades the sound out gently. With the piano synth, this opt can only have the effect of controlling the amp within the natural duration of the note and can not prolong the sound.</p> <p><i>Default: 1</i> <i>Must be zero or greater</i> <i>Can not be changed once set</i> <i>Scaled with current BPM value</i></p>

<b>attack_level:</b>	Amplitude level reached after attack phase and immediately before decay phase <i>Default: 1</i> <i>Must be zero or greater</i> <i>Can not be changed once set</i>
<b>decay_level:</b>	Amplitude level reached after decay phase and immediately before sustain phase. Defaults to sustain_level unless explicitly set <i>Default: sustain_level</i> <i>Must be zero or greater</i> <i>Can not be changed once set</i>
<b>sustain_level:</b>	Amplitude level reached after decay phase and immediately before release phase. <i>Default: 1</i> <i>Must be zero or greater</i> <i>Can not be changed once set</i>
<b>hard:</b>	Hardness of keypress. <i>Default: 0.5</i> <i>Must be a value between 0 and 1 inclusively</i> <i>Can not be changed once set</i>
<b>stereo_width:</b>	Width of the stereo effect (which makes low notes sound towards the left, high notes towards the right). 0 to 1. <i>Default: 0</i> <i>Must be a value between 0 and 1 inclusively</i> <i>Can not be changed once set</i>

## Slide Parameters

Any parameter that is slidable has three additional parameters named `_slide`, `_slide_curve`, and `_slide_shape`. For example, 'amp' is slidable, so you can also set `amp_slide`, `amp_slide_curve`, and `amp_slide_shape` with the following effects:

<b>_slide:</b>	Amount of time (in beats) for the parameter value to change. A long parameter_slide value means that the parameter takes a long time to slide from the previous value to the new value. A parameter_slide of 0 means that the parameter instantly changes to the new value. <i>Default: 0</i>
<b>_slide_shape:</b>	Shape of curve. 0: step, 1: linear, 3: sine, 4: welch, 5: custom (use *_slide_curve: opt e.g. amp_slide_curve:), 6: squared, 7: cubed. <i>Default: 5</i>
<b>_slide_curve:</b>	Shape of the slide curve (only honoured if slide shape is 5). 0 means linear and positive and negative numbers curve the segment up and down respectively. <i>Default: 0</i>

# Pink Noise

amp:	1	pan:	0	attack:	0	decay:	0	sustain:	0	release:	1
attack_level:	1	decay_level:	sustain_level	sustain_level:	1	env_curve:	2	cutoff:	110	res:	0

`use_synth :pnoise`

Noise whose spectrum falls off in power by 3 dB per octave. Useful for generating percussive sounds such as snares and hand claps. Also useful for simulating wind or sea effects.

Introduced in v2.0

## Parameters

amp:	<p>The amplitude of the sound. Typically a value between 0 and 1. Higher amplitudes may be used, but won't make the sound louder, they will just reduce the quality of all the sounds currently being played (due to compression.)</p> <p><i>Default: 1</i> <i>Must be zero or greater</i> <i>May be changed whilst playing</i> <i>Has slide parameters to shape changes</i></p>
pan:	<p>Position of sound in stereo. With headphones on, this means how much of the sound is in the left ear, and how much is in the right ear. With a value of -1, the sound is completely in the left ear, a value of 0 puts the sound equally in both ears and a value of 1 puts the sound in the right ear. Values in between -1 and 1 move the sound accordingly.</p> <p><i>Default: 0</i> <i>Must be a value between -1 and 1 inclusively</i> <i>May be changed whilst playing</i> <i>Has slide parameters to shape changes</i></p>
attack:	<p>Amount of time (in beats) for sound to reach full amplitude (attack_level). A short attack (i.e. 0.01) makes the initial part of the sound very percussive like a sharp tap. A longer attack (i.e 1) fades the sound in gently. Full length of sound is attack + decay + sustain + release.</p> <p><i>Default: 0</i> <i>Must be zero or greater</i> <i>Can not be changed once set</i> <i>Scaled with current BPM value</i></p>
decay:	<p>Amount of time (in beats) for the sound to move from full amplitude (attack_level) to the sustain amplitude (sustain_level).</p> <p><i>Default: 0</i> <i>Must be zero or greater</i> <i>Can not be changed once set</i> <i>Scaled with current BPM value</i></p>
sustain:	<p>Amount of time (in beats) for sound to remain at sustain level amplitude. Longer sustain values result in longer sounds. Full length of sound is attack + decay + sustain + release.</p> <p><i>Default: 0</i> <i>Must be zero or greater</i> <i>Can not be changed once set</i> <i>Scaled with current BPM value</i></p>
release:	<p>Amount of time (in beats) for sound to move from sustain level amplitude to silent. A short release (i.e. 0.01) makes the final part of the sound very percussive (potentially resulting in a click). A longer release (i.e 1) fades the sound out gently. Full length of sound is attack + decay + sustain + release.</p> <p><i>Default: 1</i> <i>Must be zero or greater</i> <i>Can not be changed once set</i> <i>Scaled with current BPM value</i></p>
attack_level:	<p>Amplitude level reached after attack phase and immediately before decay phase</p> <p><i>Default: 1</i> <i>Must be zero or greater</i> <i>Can not be changed once set</i></p>
decay_level:	<p>Amplitude level reached after decay phase and immediately before sustain phase. Defaults to sustain_level unless explicitly set</p> <p><i>Default: sustain_level</i> <i>Must be zero or greater</i> <i>Can not be changed once set</i></p>
sustain_level:	<p>Amplitude level reached after decay phase and immediately before release phase.</p> <p><i>Default: 1</i> <i>Must be zero or greater</i> <i>Can not be changed once set</i></p>
	<p>Select the shape of the curve between levels in the envelope. 1=linear, 2=exponential, 3=sine, 4=welch, 6=squared, 7=cubed</p>



<b>env_curve:</b>	<p><i>Default: 2</i>  <i>Must be one of the following values: [1, 2, 3, 4, 6, 7]</i>  <i>Can not be changed once set</i></p>
<b>cutoff:</b>	<p>MIDI note representing the highest frequencies allowed to be present in the sound. A low value like 30 makes the sound round and dull, a high value like 100 makes the sound buzzy and crispy.</p> <p><i>Default: 110</i>  <i>Must be zero or greater, must be a value less than 131</i>  <i>May be changed whilst playing</i>  <i>Has slide parameters to shape changes</i></p>
<b>res:</b>	<p>Filter resonance as a value between 0 and 1. Large amounts of resonance (a res: near 1) can create a whistling sound around the cutoff frequency. Smaller values produce less resonance.</p> <p><i>Default: 0</i>  <i>Must be zero or greater, must be a value less than 1</i>  <i>May be changed whilst playing</i>  <i>Has slide parameters to shape changes</i></p>

## Slide Parameters

Any parameter that is slidable has three additional parameters named `_slide`, `_slide_curve`, and `_slide_shape`. For example, 'amp' is slidable, so you can also set `amp_slide`, `amp_slide_curve`, and `amp_slide_shape` with the following effects:

<b>_slide:</b>	<p>Amount of time (in beats) for the parameter value to change. A long parameter_slide value means that the parameter takes a long time to slide from the previous value to the new value. A parameter_slide of 0 means that the parameter instantly changes to the new value.</p> <p><i>Default: 0</i></p>
<b>_slide_shape:</b>	<p>Shape of curve. 0: step, 1: linear, 3: sine, 4: welch, 5: custom (use *_slide_curve: opt e.g. amp_slide_curve:), 6: squared, 7: cubed.</p> <p><i>Default: 5</i></p>
<b>_slide_curve:</b>	<p>Shape of the slide curve (only honoured if slide shape is 5). 0 means linear and positive and negative numbers curve the segment up and down respectively.</p> <p><i>Default: 0</i></p>

# Pretty Bell

```
note: 52 amp: 1 pan: 0 attack: 0 decay: 0 sustain: 0  
release: 1 attack_level: 1 decay_level: sustain_level sustain_level: 1 env_curve: 2
```

`use_synth :pretty_bell`

A pretty bell sound. Works well with short attacks and long decays.

Introduced in v2.0

## Parameters

<b>note:</b>	<p>Note to play. Either a MIDI number or a symbol representing a note. For example:<code>30</code>, <code>52</code>, <code>:C</code>, <code>:C2</code>, <code>:Eb4</code>, or <code>:Ds3</code></p> <p><i>Default: 52</i> <i>Must be zero or greater</i> <i>May be changed whilst playing</i> <i>Has slide parameters to shape changes</i></p>
<b>amp:</b>	<p>The amplitude of the sound. Typically a value between 0 and 1. Higher amplitudes may be used, but won't make the sound louder, they will just reduce the quality of all the sounds currently being played (due to compression.)</p> <p><i>Default: 1</i> <i>Must be zero or greater</i> <i>May be changed whilst playing</i> <i>Has slide parameters to shape changes</i></p>
<b>pan:</b>	<p>Position of sound in stereo. With headphones on, this means how much of the sound is in the left ear, and how much is in the right ear. With a value of -1, the sound is completely in the left ear, a value of 0 puts the sound equally in both ears and a value of 1 puts the sound in the right ear. Values in between -1 and 1 move the sound accordingly.</p> <p><i>Default: 0</i> <i>Must be a value between -1 and 1 inclusively</i> <i>May be changed whilst playing</i> <i>Has slide parameters to shape changes</i></p>
<b>attack:</b>	<p>Amount of time (in beats) for sound to reach full amplitude (<code>attack_level</code>). A short attack (i.e. 0.01) makes the initial part of the sound very percussive like a sharp tap. A longer attack (i.e 1) fades the sound in gently. Full length of sound is attack + decay + sustain + release.</p> <p><i>Default: 0</i> <i>Must be zero or greater</i> <i>Can not be changed once set</i> <i>Scaled with current BPM value</i></p>
<b>decay:</b>	<p>Amount of time (in beats) for the sound to move from full amplitude (<code>attack_level</code>) to the sustain amplitude (<code>sustain_level</code>).</p> <p><i>Default: 0</i> <i>Must be zero or greater</i> <i>Can not be changed once set</i> <i>Scaled with current BPM value</i></p>
<b>sustain:</b>	<p>Amount of time (in beats) for sound to remain at sustain level amplitude. Longer sustain values result in longer sounds. Full length of sound is attack + decay + sustain + release.</p> <p><i>Default: 0</i> <i>Must be zero or greater</i> <i>Can not be changed once set</i> <i>Scaled with current BPM value</i></p>
<b>release:</b>	<p>Amount of time (in beats) for sound to move from sustain level amplitude to silent. A short release (i.e. 0.01) makes the final part of the sound very percussive (potentially resulting in a click). A longer release (i.e 1) fades the sound out gently. Full length of sound is attack + decay + sustain + release.</p> <p><i>Default: 1</i> <i>Must be zero or greater</i> <i>Can not be changed once set</i> <i>Scaled with current BPM value</i></p>
<b>attack_level:</b>	<p>Amplitude level reached after attack phase and immediately before decay phase</p> <p><i>Default: 1</i> <i>Must be zero or greater</i> <i>Can not be changed once set</i></p>
<b>decay_level:</b>	<p>Amplitude level reached after decay phase and immediately before sustain phase. Defaults to <code>sustain_level</code> unless explicitly set</p> <p><i>Default: sustain_level</i> <i>Must be zero or greater</i> <i>Can not be changed once set</i></p>
	<p>Amplitude level reached after decay phase and immediately before release phase.</p>

<b>sustain_level:</b>	<i>Default: 1</i> <i>Must be zero or greater</i> <i>Can not be changed once set</i>
<b>env_curve:</b>	Select the shape of the curve between levels in the envelope. 1=linear, 2=exponential, 3=sine, 4=welch, 6=squared, 7=cubed <i>Default: 2</i> <i>Must be one of the following values: [1, 2, 3, 4, 6, 7]</i> <i>Can not be changed once set</i>

## Slide Parameters

Any parameter that is slidable has three additional parameters named `_slide`, `_slide_curve`, and `_slide_shape`. For example, 'amp' is slidable, so you can also set `amp_slide`, `amp_slide_curve`, and `amp_slide_shape` with the following effects:

<b>_slide:</b>	Amount of time (in beats) for the parameter value to change. A long parameter <code>_slide</code> value means that the parameter takes a long time to slide from the previous value to the new value. A parameter <code>_slide</code> of 0 means that the parameter instantly changes to the new value. <i>Default: 0</i>
<b>_slide_shape:</b>	Shape of curve. 0: step, 1: linear, 3: sine, 4: welch, 5: custom (use <code>*_slide_curve</code> : opt e.g. <code>amp_slide_curve</code> :), 6: squared, 7: cubed. <i>Default: 5</i>
<b>_slide_curve:</b>	Shape of the slide curve (only honoured if slide shape is 5). 0 means linear and positive and negative numbers curve the segment up and down respectively. <i>Default: 0</i>

# The Prophet

note:	52	amp:	1	pan:	0	attack:	0	decay:	0	sustain:	0
release:	1	attack_level:	1	decay_level:	sustain_level	sustain_level:	1	env_curve:	2	cutoff:	110
res:	0.7										

`use_synth :prophet`

Dark and swirly, this synth uses Pulse Width Modulation (PWM) to create a timbre which continually moves around. This effect is created using the pulse ugen which produces a variable width square wave. We then control the width of the pulses using a variety of LFOs - sin-osc and lf-tri in this case. We use a number of these LFO modulated pulse ugens with varying LFO type and rate (and phase in some cases) to provide the LFO with a different starting point. We then mix all these pulses together to create a thick sound and then feed it through a resonant low pass filter (rlpf). For extra bass, one of the pulses is an octave lower (half the frequency) and its LFO has a little bit of randomisation thrown into its frequency component for that extra bit of variety.

Introduced in v2.0

## Parameters

note:	<p>Note to play. Either a MIDI number or a symbol representing a note. For example:<code>30</code>, <code>52</code>, <code>:C</code>, <code>:C2</code>, <code>:Eb4</code>, or <code>:Ds3</code></p> <p><i>Default: 52</i> <i>Must be zero or greater</i> <i>May be changed whilst playing</i> <i>Has slide parameters to shape changes</i></p>
amp:	<p>The amplitude of the sound. Typically a value between 0 and 1. Higher amplitudes may be used, but won't make the sound louder, they will just reduce the quality of all the sounds currently being played (due to compression.)</p> <p><i>Default: 1</i> <i>Must be zero or greater</i> <i>May be changed whilst playing</i> <i>Has slide parameters to shape changes</i></p>
pan:	<p>Position of sound in stereo. With headphones on, this means how much of the sound is in the left ear, and how much is in the right ear. With a value of -1, the sound is completely in the left ear, a value of 0 puts the sound equally in both ears and a value of 1 puts the sound in the right ear. Values in between -1 and 1 move the sound accordingly.</p> <p><i>Default: 0</i> <i>Must be a value between -1 and 1 inclusively</i> <i>May be changed whilst playing</i> <i>Has slide parameters to shape changes</i></p>
attack:	<p>Amount of time (in beats) for sound to reach full amplitude (attack_level). A short attack (i.e. 0.01) makes the initial part of the sound very percussive like a sharp tap. A longer attack (i.e 1) fades the sound in gently. Full length of sound is attack + decay + sustain + release.</p> <p><i>Default: 0</i> <i>Must be zero or greater</i> <i>Can not be changed once set</i> <i>Scaled with current BPM value</i></p>
decay:	<p>Amount of time (in beats) for the sound to move from full amplitude (attack_level) to the sustain amplitude (sustain_level).</p> <p><i>Default: 0</i> <i>Must be zero or greater</i> <i>Can not be changed once set</i> <i>Scaled with current BPM value</i></p>
sustain:	<p>Amount of time (in beats) for sound to remain at sustain level amplitude. Longer sustain values result in longer sounds. Full length of sound is attack + decay + sustain + release.</p> <p><i>Default: 0</i> <i>Must be zero or greater</i> <i>Can not be changed once set</i> <i>Scaled with current BPM value</i></p>
release:	<p>Amount of time (in beats) for sound to move from sustain level amplitude to silent. A short release (i.e. 0.01) makes the final part of the sound very percussive (potentially resulting in a click). A longer release (i.e 1) fades the sound out gently. Full length of sound is attack + decay + sustain + release.</p> <p><i>Default: 1</i> <i>Must be zero or greater</i> <i>Can not be changed once set</i> <i>Scaled with current BPM value</i></p>
attack_level:	<p>Amplitude level reached after attack phase and immediately before decay phase</p> <p><i>Default: 1</i> <i>Must be zero or greater</i> <i>Can not be changed once set</i></p>
	<p>Amplitude level reached after decay phase and immediately before sustain phase. Defaults to sustain_level unless explicitly set</p>

<b>decay_level:</b>	<i>Default: sustain_level</i> <i>Must be zero or greater</i> <i>Can not be changed once set</i>
<b>sustain_level:</b>	Amplitude level reached after decay phase and immediately before release phase. <i>Default: 1</i> <i>Must be zero or greater</i> <i>Can not be changed once set</i>
<b>env_curve:</b>	Select the shape of the curve between levels in the envelope. 1=linear, 2=exponential, 3=sine, 4=welch, 6=squared, 7=cubed <i>Default: 2</i> <i>Must be one of the following values: [1, 2, 3, 4, 6, 7]</i> <i>Can not be changed once set</i>
<b>cutoff:</b>	MIDI note representing the highest frequencies allowed to be present in the sound. A low value like 30 makes the sound round and dull, a high value like 100 makes the sound buzzy and crispy. <i>Default: 110</i> <i>Must be zero or greater, must be a value less than 131</i> <i>May be changed whilst playing</i> <i>Has slide parameters to shape changes</i>
<b>res:</b>	Filter resonance as a value between 0 and 1. Large amounts of resonance (a res: near 1) can create a whistling sound around the cutoff frequency. Smaller values produce less resonance. <i>Default: 0.7</i> <i>Must be zero or greater, must be a value less than 1</i> <i>May be changed whilst playing</i> <i>Has slide parameters to shape changes</i>

## Slide Parameters

Any parameter that is slidable has three additional parameters named `_slide`, `_slide_curve`, and `_slide_shape`. For example, 'amp' is slidable, so you can also set `amp_slide`, `amp_slide_curve`, and `amp_slide_shape` with the following effects:

<b>_slide:</b>	Amount of time (in beats) for the parameter value to change. A long <code>parameter_slide</code> value means that the parameter takes a long time to slide from the previous value to the new value. A <code>parameter_slide</code> of 0 means that the parameter instantly changes to the new value. <i>Default: 0</i>
<b>_slide_shape:</b>	Shape of curve. 0: step, 1: linear, 3: sine, 4: welch, 5: custom (use <code>*_slide_curve</code> : opt e.g. <code>amp_slide_curve</code> ); 6: squared, 7: cubed. <i>Default: 5</i>
<b>_slide_curve:</b>	Shape of the slide curve (only honoured if slide shape is 5). 0 means linear and positive and negative numbers curve the segment up and down respectively. <i>Default: 0</i>

# Pulse Wave

note:	52	amp:	1	pan:	0	attack:	0	decay:	0	sustain:	0
release:	1	attack_level:	1	decay_level:	sustain_level	sustain_level:	1	env_curve:	2	cutoff:	100
pulse_width:	0.5										

`use_synth :pulse`

A simple pulse wave with a low pass filter. This defaults to a square wave, but the timbre can be changed dramatically by adjusting the `pulse_width` arg between 0 and 1. The pulse wave is thick and heavy with lower notes and is a great ingredient for bass sounds.

Introduced in v2.0

## Parameters

note:	Note to play. Either a MIDI number or a symbol representing a note. For example: <code>30</code> , <code>52</code> , <code>:C</code> , <code>:C2</code> , <code>:Eb4</code> , or <code>:Ds3</code> <i>Default: 52</i> <i>Must be zero or greater</i> <i>May be changed whilst playing</i> <i>Has slide parameters to shape changes</i>
amp:	The amplitude of the sound. Typically a value between 0 and 1. Higher amplitudes may be used, but won't make the sound louder, they will just reduce the quality of all the sounds currently being played (due to compression.) <i>Default: 1</i> <i>Must be zero or greater</i> <i>May be changed whilst playing</i> <i>Has slide parameters to shape changes</i>
pan:	Position of sound in stereo. With headphones on, this means how much of the sound is in the left ear, and how much is in the right ear. With a value of -1, the sound is completely in the left ear, a value of 0 puts the sound equally in both ears and a value of 1 puts the sound in the right ear. Values in between -1 and 1 move the sound accordingly. <i>Default: 0</i> <i>Must be a value between -1 and 1 inclusively</i> <i>May be changed whilst playing</i> <i>Has slide parameters to shape changes</i>
attack:	Amount of time (in beats) for sound to reach full amplitude ( <code>attack_level</code> ). A short attack (i.e. 0.01) makes the initial part of the sound very percussive like a sharp tap. A longer attack (i.e 1) fades the sound in gently. Full length of sound is <code>attack</code> + <code>decay</code> + <code>sustain</code> + <code>release</code> . <i>Default: 0</i> <i>Must be zero or greater</i> <i>Can not be changed once set</i> <i>Scaled with current BPM value</i>
decay:	Amount of time (in beats) for the sound to move from full amplitude ( <code>attack_level</code> ) to the sustain amplitude ( <code>sustain_level</code> ). <i>Default: 0</i> <i>Must be zero or greater</i> <i>Can not be changed once set</i> <i>Scaled with current BPM value</i>
sustain:	Amount of time (in beats) for sound to remain at sustain level amplitude. Longer sustain values result in longer sounds. Full length of sound is <code>attack</code> + <code>decay</code> + <code>sustain</code> + <code>release</code> . <i>Default: 0</i> <i>Must be zero or greater</i> <i>Can not be changed once set</i> <i>Scaled with current BPM value</i>
release:	Amount of time (in beats) for sound to move from sustain level amplitude to silent. A short release (i.e. 0.01) makes the final part of the sound very percussive (potentially resulting in a click). A longer release (i.e 1) fades the sound out gently. Full length of sound is <code>attack</code> + <code>decay</code> + <code>sustain</code> + <code>release</code> . <i>Default: 1</i> <i>Must be zero or greater</i> <i>Can not be changed once set</i> <i>Scaled with current BPM value</i>
attack_level:	Amplitude level reached after attack phase and immediately before decay phase <i>Default: 1</i> <i>Must be zero or greater</i> <i>Can not be changed once set</i>
decay_level:	Amplitude level reached after decay phase and immediately before sustain phase. Defaults to <code>sustain_level</code> unless explicitly set <i>Default: sustain_level</i> <i>Must be zero or greater</i> <i>Can not be changed once set</i>

<b>sustain_level:</b>	Amplitude level reached after decay phase and immediately before release phase.  <i>Default: 1</i> <i>Must be zero or greater</i> <i>Can not be changed once set</i>
<b>env_curve:</b>	Select the shape of the curve between levels in the envelope. 1=linear, 2=exponential, 3=sine, 4=welch, 6=squared, 7=cubed  <i>Default: 2</i> <i>Must be one of the following values: [1, 2, 3, 4, 6, 7]</i> <i>Can not be changed once set</i>
<b>cutoff:</b>	MIDI note representing the highest frequencies allowed to be present in the sound. A low value like 30 makes the sound round and dull, a high value like 100 makes the sound buzzy and crispy.  <i>Default: 100</i> <i>Must be zero or greater, must be a value less than 131</i> <i>May be changed whilst playing</i> <i>Has slide parameters to shape changes</i>
<b>pulse_width:</b>	The width of the pulse wave as a value between 0 and 1. A width of 0.5 will produce a square wave. Different values will change the timbre of the sound. Only valid if wave is type pulse.  <i>Default: 0.5</i> <i>Must be a value between 0 and 1 exclusively</i> <i>May be changed whilst playing</i> <i>Has slide parameters to shape changes</i>

## Slide Parameters

Any parameter that is slidable has three additional parameters named `_slide`, `_slide_curve`, and `_slide_shape`. For example, 'amp' is slidable, so you can also set `amp_slide`, `amp_slide_curve`, and `amp_slide_shape` with the following effects:

<b>_slide:</b>	Amount of time (in beats) for the parameter value to change. A long parameter <code>_slide</code> value means that the parameter takes a long time to slide from the previous value to the new value. A parameter <code>_slide</code> of 0 means that the parameter instantly changes to the new value.  <i>Default: 0</i>
<b>_slide_shape:</b>	Shape of curve. 0: step, 1: linear, 3: sine, 4: welch, 5: custom (use <code>*_slide_curve</code> : opt e.g. <code>amp_slide_curve</code> :), 6: squared, 7: cubed.  <i>Default: 5</i>
<b>_slide_curve:</b>	Shape of the slide curve (only honoured if slide shape is 5). 0 means linear and positive and negative numbers curve the segment up and down respectively.  <i>Default: 0</i>

# Saw Wave

```
note: 52 amp: 1 pan: 0 attack: 0 decay: 0 sustain: 0  
release: 1 attack_level: 1 decay_level: sustain_level sustain_level: 1 env_curve: 2
```

`use_synth :saw`

A saw wave with a low pass filter. Great for using with FX such as the built in low pass filter (available via the cutoff arg) due to the complexity and thickness of the sound.

Introduced in v2.0

## Parameters

<b>note:</b>	Note to play. Either a MIDI number or a symbol representing a note. For example: <code>30</code> , <code>52</code> , <code>:C</code> , <code>:C2</code> , <code>:Eb4</code> , or <code>:Ds3</code> <i>Default: 52</i> <i>Must be zero or greater</i> <i>May be changed whilst playing</i> <i>Has slide parameters to shape changes</i>
<b>amp:</b>	The amplitude of the sound. Typically a value between 0 and 1. Higher amplitudes may be used, but won't make the sound louder, they will just reduce the quality of all the sounds currently being played (due to compression.) <i>Default: 1</i> <i>Must be zero or greater</i> <i>May be changed whilst playing</i> <i>Has slide parameters to shape changes</i>
<b>pan:</b>	Position of sound in stereo. With headphones on, this means how much of the sound is in the left ear, and how much is in the right ear. With a value of -1, the sound is completely in the left ear, a value of 0 puts the sound equally in both ears and a value of 1 puts the sound in the right ear. Values in between -1 and 1 move the sound accordingly. <i>Default: 0</i> <i>Must be a value between -1 and 1 inclusively</i> <i>May be changed whilst playing</i> <i>Has slide parameters to shape changes</i>
<b>attack:</b>	Amount of time (in beats) for sound to reach full amplitude ( <code>attack_level</code> ). A short attack (i.e. 0.01) makes the initial part of the sound very percussive like a sharp tap. A longer attack (i.e 1) fades the sound in gently. Full length of sound is <code>attack</code> + <code>decay</code> + <code>sustain</code> + <code>release</code> . <i>Default: 0</i> <i>Must be zero or greater</i> <i>Can not be changed once set</i> <i>Scaled with current BPM value</i>
<b>decay:</b>	Amount of time (in beats) for the sound to move from full amplitude ( <code>attack_level</code> ) to the sustain amplitude ( <code>sustain_level</code> ). <i>Default: 0</i> <i>Must be zero or greater</i> <i>Can not be changed once set</i> <i>Scaled with current BPM value</i>
<b>sustain:</b>	Amount of time (in beats) for sound to remain at sustain level amplitude. Longer sustain values result in longer sounds. Full length of sound is <code>attack</code> + <code>decay</code> + <code>sustain</code> + <code>release</code> . <i>Default: 0</i> <i>Must be zero or greater</i> <i>Can not be changed once set</i> <i>Scaled with current BPM value</i>
<b>release:</b>	Amount of time (in beats) for sound to move from sustain level amplitude to silent. A short release (i.e. 0.01) makes the final part of the sound very percussive (potentially resulting in a click). A longer release (i.e 1) fades the sound out gently. Full length of sound is <code>attack</code> + <code>decay</code> + <code>sustain</code> + <code>release</code> . <i>Default: 1</i> <i>Must be zero or greater</i> <i>Can not be changed once set</i> <i>Scaled with current BPM value</i>
<b>attack_level:</b>	Amplitude level reached after attack phase and immediately before decay phase <i>Default: 1</i> <i>Must be zero or greater</i> <i>Can not be changed once set</i>
<b>decay_level:</b>	Amplitude level reached after decay phase and immediately before sustain phase. Defaults to <code>sustain_level</code> unless explicitly set <i>Default: sustain_level</i> <i>Must be zero or greater</i> <i>Can not be changed once set</i>
	Amplitude level reached after decay phase and immediately before release phase.



<b>sustain_level:</b>	<i>Default: 1</i> <i>Must be zero or greater</i> <i>Can not be changed once set</i>
<b>env_curve:</b>	Select the shape of the curve between levels in the envelope. 1=linear, 2=exponential, 3=sine, 4=welch, 6=squared, 7=cubed <i>Default: 2</i> <i>Must be one of the following values: [1, 2, 3, 4, 6, 7]</i> <i>Can not be changed once set</i>

## Slide Parameters

Any parameter that is slidable has three additional parameters named `_slide`, `_slide_curve`, and `_slide_shape`. For example, 'amp' is slidable, so you can also set `amp_slide`, `amp_slide_curve`, and `amp_slide_shape` with the following effects:

<b>_slide:</b>	Amount of time (in beats) for the parameter value to change. A long parameter_slide value means that the parameter takes a long time to slide from the previous value to the new value. A parameter_slide of 0 means that the parameter instantly changes to the new value. <i>Default: 0</i>
<b>_slide_shape:</b>	Shape of curve. 0: step, 1: linear, 3: sine, 4: welch, 5: custom (use *_slide_curve: opt e.g. amp_slide_curve:), 6: squared, 7: cubed. <i>Default: 5</i>
<b>_slide_curve:</b>	Shape of the slide curve (only honoured if slide shape is 5). 0 means linear and positive and negative numbers curve the segment up and down respectively. <i>Default: 0</i>

# Sine Wave

note:	52	amp:	1	pan:	0	attack:	0	decay:	0	sustain:	0
release:	1	attack_level:	1	decay_level:	sustain_level	sustain_level:	1	env_curve:	2		

`use_synth :sine`

A simple pure sine wave. The sine wave is the simplest, purest sound there is and is the fundamental building block of all noise. The mathematician Fourier demonstrated that any sound could be built out of a number of sine waves (the more complex the sound, the more sine waves needed). Have a play combining a number of sine waves to design your own sounds!

Introduced in v2.0

## Parameters

note:	<p>Note to play. Either a MIDI number or a symbol representing a note. For example:<code>30</code>, <code>52</code>, <code>:C</code>, <code>:C2</code>, <code>:Eb4</code>, or <code>:Ds3</code></p> <p><i>Default: 52</i> <i>Must be zero or greater</i> <i>May be changed whilst playing</i> <a href="#">Has slide parameters to shape changes</a></p>
amp:	<p>The amplitude of the sound. Typically a value between 0 and 1. Higher amplitudes may be used, but won't make the sound louder, they will just reduce the quality of all the sounds currently being played (due to compression.)</p> <p><i>Default: 1</i> <i>Must be zero or greater</i> <i>May be changed whilst playing</i> <a href="#">Has slide parameters to shape changes</a></p>
pan:	<p>Position of sound in stereo. With headphones on, this means how much of the sound is in the left ear, and how much is in the right ear. With a value of -1, the sound is completely in the left ear, a value of 0 puts the sound equally in both ears and a value of 1 puts the sound in the right ear. Values in between -1 and 1 move the sound accordingly.</p> <p><i>Default: 0</i> <i>Must be a value between -1 and 1 inclusively</i> <i>May be changed whilst playing</i> <a href="#">Has slide parameters to shape changes</a></p>
attack:	<p>Amount of time (in beats) for sound to reach full amplitude (<code>attack_level</code>). A short attack (i.e. 0.01) makes the initial part of the sound very percussive like a sharp tap. A longer attack (i.e 1) fades the sound in gently. Full length of sound is <code>attack</code> + <code>decay</code> + <code>sustain</code> + <code>release</code>.</p> <p><i>Default: 0</i> <i>Must be zero or greater</i> <i>Can not be changed once set</i> <i>Scaled with current BPM value</i></p>
decay:	<p>Amount of time (in beats) for the sound to move from full amplitude (<code>attack_level</code>) to the sustain amplitude (<code>sustain_level</code>).</p> <p><i>Default: 0</i> <i>Must be zero or greater</i> <i>Can not be changed once set</i> <i>Scaled with current BPM value</i></p>
sustain:	<p>Amount of time (in beats) for sound to remain at sustain level amplitude. Longer sustain values result in longer sounds. Full length of sound is <code>attack</code> + <code>decay</code> + <code>sustain</code> + <code>release</code>.</p> <p><i>Default: 0</i> <i>Must be zero or greater</i> <i>Can not be changed once set</i> <i>Scaled with current BPM value</i></p>
release:	<p>Amount of time (in beats) for sound to move from sustain level amplitude to silent. A short release (i.e. 0.01) makes the final part of the sound very percussive (potentially resulting in a click). A longer release (i.e 1) fades the sound out gently. Full length of sound is <code>attack</code> + <code>decay</code> + <code>sustain</code> + <code>release</code>.</p> <p><i>Default: 1</i> <i>Must be zero or greater</i> <i>Can not be changed once set</i> <i>Scaled with current BPM value</i></p>
attack_level:	<p>Amplitude level reached after attack phase and immediately before decay phase</p> <p><i>Default: 1</i> <i>Must be zero or greater</i> <i>Can not be changed once set</i></p>
decay_level:	<p>Amplitude level reached after decay phase and immediately before sustain phase. Defaults to <code>sustain_level</code> unless explicitly set</p> <p><i>Default: sustain_level</i> <i>Must be zero or greater</i> <i>Can not be changed once set</i></p>
	<p>Amplitude level reached after decay phase and immediately before release phase</p>

<b>sustain_level:</b>	Amplitude level reached after decay phase and immediately before release phase. <i>Default: 1</i> <i>Must be zero or greater</i> <i>Can not be changed once set</i>
<b>env_curve:</b>	Select the shape of the curve between levels in the envelope. 1=linear, 2=exponential, 3=sine, 4=welch, 6=squared, 7=cubed <i>Default: 2</i> <i>Must be one of the following values: [1, 2, 3, 4, 6, 7]</i> <i>Can not be changed once set</i>

## Slide Parameters

Any parameter that is slidable has three additional parameters named `_slide`, `_slide_curve`, and `_slide_shape`. For example, 'amp' is slidable, so you can also set `amp_slide`, `amp_slide_curve`, and `amp_slide_shape` with the following effects:

<b>_slide:</b>	Amount of time (in beats) for the parameter value to change. A long parameter_slide value means that the parameter takes a long time to slide from the previous value to the new value. A parameter_slide of 0 means that the parameter instantly changes to the new value. <i>Default: 0</i>
<b>_slide_shape:</b>	Shape of curve. 0: step, 1: linear, 3: sine, 4: welch, 5: custom (use *_slide_curve: opt e.g. amp_slide_curve:), 6: squared, 7: cubed. <i>Default: 5</i>
<b>_slide_curve:</b>	Shape of the slide curve (only honoured if slide shape is 5). 0 means linear and positive and negative numbers curve the segment up and down respectively. <i>Default: 0</i>

# Square Wave

note:	52	amp:	1	pan:	0	attack:	0	decay:	0	sustain:	0
release:	1	attack_level:	1	decay_level:	sustain_level	sustain_level:	1	env_curve:	2	cutoff:	100

`use_synth :square`

A simple square wave with a low pass filter. The square wave is thick and heavy with lower notes and is a great ingredient for bass sounds. If you wish to modulate the width of the square wave see the synth pulse.

Introduced in v2.2

## Parameters

note:	<p>Note to play. Either a MIDI number or a symbol representing a note. For example:<code>30</code>, <code>52</code>, <code>:C</code>, <code>:C2</code>, <code>:Eb4</code>, or <code>:Ds3</code></p> <p><i>Default: 52</i> <i>Must be zero or greater</i> <i>May be changed whilst playing</i> <i>Has slide parameters to shape changes</i></p>
amp:	<p>The amplitude of the sound. Typically a value between 0 and 1. Higher amplitudes may be used, but won't make the sound louder, they will just reduce the quality of all the sounds currently being played (due to compression.)</p> <p><i>Default: 1</i> <i>Must be zero or greater</i> <i>May be changed whilst playing</i> <i>Has slide parameters to shape changes</i></p>
pan:	<p>Position of sound in stereo. With headphones on, this means how much of the sound is in the left ear, and how much is in the right ear. With a value of -1, the sound is completely in the left ear, a value of 0 puts the sound equally in both ears and a value of 1 puts the sound in the right ear. Values in between -1 and 1 move the sound accordingly.</p> <p><i>Default: 0</i> <i>Must be a value between -1 and 1 inclusively</i> <i>May be changed whilst playing</i> <i>Has slide parameters to shape changes</i></p>
attack:	<p>Amount of time (in beats) for sound to reach full amplitude (<code>attack_level</code>). A short attack (i.e. 0.01) makes the initial part of the sound very percussive like a sharp tap. A longer attack (i.e 1) fades the sound in gently. Full length of sound is attack + decay + sustain + release.</p> <p><i>Default: 0</i> <i>Must be zero or greater</i> <i>Can not be changed once set</i> <i>Scaled with current BPM value</i></p>
decay:	<p>Amount of time (in beats) for the sound to move from full amplitude (<code>attack_level</code>) to the sustain amplitude (<code>sustain_level</code>).</p> <p><i>Default: 0</i> <i>Must be zero or greater</i> <i>Can not be changed once set</i> <i>Scaled with current BPM value</i></p>
sustain:	<p>Amount of time (in beats) for sound to remain at sustain level amplitude. Longer sustain values result in longer sounds. Full length of sound is attack + decay + sustain + release.</p> <p><i>Default: 0</i> <i>Must be zero or greater</i> <i>Can not be changed once set</i> <i>Scaled with current BPM value</i></p>
release:	<p>Amount of time (in beats) for sound to move from sustain level amplitude to silent. A short release (i.e. 0.01) makes the final part of the sound very percussive (potentially resulting in a click). A longer release (i.e 1) fades the sound out gently. Full length of sound is attack + decay + sustain + release.</p> <p><i>Default: 1</i> <i>Must be zero or greater</i> <i>Can not be changed once set</i> <i>Scaled with current BPM value</i></p>
attack_level:	<p>Amplitude level reached after attack phase and immediately before decay phase</p> <p><i>Default: 1</i> <i>Must be zero or greater</i> <i>Can not be changed once set</i></p>
decay_level:	<p>Amplitude level reached after decay phase and immediately before sustain phase. Defaults to <code>sustain_level</code> unless explicitly set</p> <p><i>Default: sustain_level</i> <i>Must be zero or greater</i> <i>Can not be changed once set</i></p>
	<p>Amplitude level reached after decay phase and immediately before release phase.</p>

<b>sustain_level:</b>	<p><i>Default: 1</i>  <i>Must be zero or greater</i>  <i>Can not be changed once set</i></p>
<b>env_curve:</b>	<p>Select the shape of the curve between levels in the envelope. 1=linear, 2=exponential, 3=sine, 4=welch, 6=squared, 7=cubed</p> <p><i>Default: 2</i>  <i>Must be one of the following values: [1, 2, 3, 4, 6, 7]</i>  <i>Can not be changed once set</i></p>
<b>cutoff:</b>	<p>MIDI note representing the highest frequencies allowed to be present in the sound. A low value like 30 makes the sound round and dull, a high value like 100 makes the sound buzzy and crispy.</p> <p><i>Default: 100</i>  <i>Must be zero or greater, must be a value less than 131</i>  <i>May be changed whilst playing</i>  <i>Has slide parameters to shape changes</i></p>

## Slide Parameters

Any parameter that is slidable has three additional parameters named `_slide`, `_slide_curve`, and `_slide_shape`. For example, 'amp' is slidable, so you can also set `amp_slide`, `amp_slide_curve`, and `amp_slide_shape` with the following effects:

<b>_slide:</b>	<p>Amount of time (in beats) for the parameter value to change. A long parameter_slide value means that the parameter takes a long time to slide from the previous value to the new value. A parameter_slide of 0 means that the parameter instantly changes to the new value.</p> <p><i>Default: 0</i></p>
<b>_slide_shape:</b>	<p>Shape of curve. 0: step, 1: linear, 3: sine, 4: welch, 5: custom (use *_slide_curve: opt e.g. amp_slide_curve:), 6: squared, 7: cubed.</p> <p><i>Default: 5</i></p>
<b>_slide_curve:</b>	<p>Shape of the slide curve (only honoured if slide shape is 5). 0 means linear and positive and negative numbers curve the segment up and down respectively.</p> <p><i>Default: 0</i></p>

# Pulse Wave with sub

note:	52	amp:	1	pan:	0	attack:	0	decay:	0	sustain:	0
release:	1	attack_level:	1	decay_level:	sustain_level	sustain_level:	1	env_curve:	2	cutoff:	100
pulse_width:	0.5	sub_amp:	1	sub_detune:	-12						

`use_synth :subpulse`

A pulse wave with a sub sine wave passed through a low pass filter. The pulse wave is thick and heavy with lower notes and is a great ingredient for bass sounds - especially with the sub wave.

Introduced in v2.6

## Parameters

note:	Note to play. Either a MIDI number or a symbol representing a note. For example: <code>30</code> , <code>52</code> , <code>:C</code> , <code>:C2</code> , <code>:Eb4</code> , or <code>:Ds3</code> <i>Default: 52</i> <i>Must be zero or greater</i> <i>May be changed whilst playing</i> <i>Has slide parameters to shape changes</i>
amp:	The amplitude of the sound. Typically a value between 0 and 1. Higher amplitudes may be used, but won't make the sound louder, they will just reduce the quality of all the sounds currently being played (due to compression.) <i>Default: 1</i> <i>Must be zero or greater</i> <i>May be changed whilst playing</i> <i>Has slide parameters to shape changes</i>
pan:	Position of sound in stereo. With headphones on, this means how much of the sound is in the left ear, and how much is in the right ear. With a value of -1, the sound is completely in the left ear, a value of 0 puts the sound equally in both ears and a value of 1 puts the sound in the right ear. Values in between -1 and 1 move the sound accordingly. <i>Default: 0</i> <i>Must be a value between -1 and 1 inclusively</i> <i>May be changed whilst playing</i> <i>Has slide parameters to shape changes</i>
attack:	Amount of time (in beats) for sound to reach full amplitude ( <code>attack_level</code> ). A short attack (i.e. 0.01) makes the initial part of the sound very percussive like a sharp tap. A longer attack (i.e 1) fades the sound in gently. Full length of sound is <code>attack</code> + <code>decay</code> + <code>sustain</code> + <code>release</code> . <i>Default: 0</i> <i>Must be zero or greater</i> <i>Can not be changed once set</i> <i>Scaled with current BPM value</i>
decay:	Amount of time (in beats) for the sound to move from full amplitude ( <code>attack_level</code> ) to the sustain amplitude ( <code>sustain_level</code> ). <i>Default: 0</i> <i>Must be zero or greater</i> <i>Can not be changed once set</i> <i>Scaled with current BPM value</i>
sustain:	Amount of time (in beats) for sound to remain at sustain level amplitude. Longer sustain values result in longer sounds. Full length of sound is <code>attack</code> + <code>decay</code> + <code>sustain</code> + <code>release</code> . <i>Default: 0</i> <i>Must be zero or greater</i> <i>Can not be changed once set</i> <i>Scaled with current BPM value</i>
release:	Amount of time (in beats) for sound to move from sustain level amplitude to silent. A short release (i.e. 0.01) makes the final part of the sound very percussive (potentially resulting in a click). A longer release (i.e 1) fades the sound out gently. Full length of sound is <code>attack</code> + <code>decay</code> + <code>sustain</code> + <code>release</code> . <i>Default: 1</i> <i>Must be zero or greater</i> <i>Can not be changed once set</i> <i>Scaled with current BPM value</i>
attack_level:	Amplitude level reached after attack phase and immediately before decay phase <i>Default: 1</i> <i>Must be zero or greater</i> <i>Can not be changed once set</i>
decay_level:	Amplitude level reached after decay phase and immediately before sustain phase. Defaults to <code>sustain_level</code> unless explicitly set <i>Default: sustain_level</i> <i>Must be zero or greater</i> <i>Can not be changed once set</i>

<b>sustain_level:</b>	Amplitude level reached after decay phase and immediately before release phase.  <i>Default: 1</i> <i>Must be zero or greater</i> <i>Can not be changed once set</i>
<b>env_curve:</b>	Select the shape of the curve between levels in the envelope. 1=linear, 2=exponential, 3=sine, 4=welch, 6=squared, 7=cubed  <i>Default: 2</i> <i>Must be one of the following values: [1, 2, 3, 4, 6, 7]</i> <i>Can not be changed once set</i>
<b>cutoff:</b>	MIDI note representing the highest frequencies allowed to be present in the sound. A low value like 30 makes the sound round and dull, a high value like 100 makes the sound buzzy and crispy.  <i>Default: 100</i> <i>Must be zero or greater, must be a value less than 131</i> <i>May be changed whilst playing</i> <i>Has slide parameters to shape changes</i>
<b>pulse_width:</b>	The width of the pulse wave as a value between 0 and 1. A width of 0.5 will produce a square wave. Different values will change the timbre of the sound. Only valid if wave is type pulse.  <i>Default: 0.5</i> <i>Must be a value between 0 and 1 exclusively</i> <i>May be changed whilst playing</i> <i>Has slide parameters to shape changes</i>
<b>sub_amp:</b>	Amplitude for the additional sine wave.  <i>Default: 1</i> <i>May be changed whilst playing</i> <i>Has slide parameters to shape changes</i>
<b>sub_detune:</b>	Amount of detune from the note for the additional sine wave. Default is -12  <i>Default: -12</i> <i>May be changed whilst playing</i> <i>Has slide parameters to shape changes</i>

## Slide Parameters

Any parameter that is slidable has three additional parameters named `_slide`, `_slide_curve`, and `_slide_shape`. For example, 'amp' is slidable, so you can also set `amp_slide`, `amp_slide_curve`, and `amp_slide_shape` with the following effects:

<b>_slide:</b>	Amount of time (in beats) for the parameter value to change. A long <code>parameter_slide</code> value means that the parameter takes a long time to slide from the previous value to the new value. A <code>parameter_slide</code> of 0 means that the parameter instantly changes to the new value.  <i>Default: 0</i>
<b>_slide_shape:</b>	Shape of curve. 0: step, 1: linear, 3: sine, 4: welch, 5: custom (use <code>*_slide_curve</code> : opt e.g. <code>amp_slide_curve</code> ); 6: squared, 7: cubed.  <i>Default: 5</i>
<b>_slide_curve:</b>	Shape of the slide curve (only honoured if slide shape is 5). 0 means linear and positive and negative numbers curve the segment up and down respectively.  <i>Default: 0</i>

# Supersaw

note:	52	amp:	1	pan:	0	attack:	0	decay:	0	sustain:	0
release:	1	attack_level:	1	decay_level:	sustain_level	sustain_level:	1	env_curve:	2	cutoff:	130
res:	0.7										

`use_synth :supersaw`

Thick swirly saw waves sparkling and moving about to create a rich trancy sound.

Introduced in v2.0

## Parameters

note:	<p>Note to play. Either a MIDI number or a symbol representing a note. For example:<code>30</code>, <code>52</code>, <code>:C</code>, <code>:C2</code>, <code>:Eb4</code>, or <code>:Ds3</code></p> <p><i>Default: 52</i> <i>Must be zero or greater</i> <i>May be changed whilst playing</i> <i>Has slide parameters to shape changes</i></p>
amp:	<p>The amplitude of the sound. Typically a value between 0 and 1. Higher amplitudes may be used, but won't make the sound louder, they will just reduce the quality of all the sounds currently being played (due to compression.)</p> <p><i>Default: 1</i> <i>Must be zero or greater</i> <i>May be changed whilst playing</i> <i>Has slide parameters to shape changes</i></p>
pan:	<p>Position of sound in stereo. With headphones on, this means how much of the sound is in the left ear, and how much is in the right ear. With a value of -1, the sound is completely in the left ear, a value of 0 puts the sound equally in both ears and a value of 1 puts the sound in the right ear. Values in between -1 and 1 move the sound accordingly.</p> <p><i>Default: 0</i> <i>Must be a value between -1 and 1 inclusively</i> <i>May be changed whilst playing</i> <i>Has slide parameters to shape changes</i></p>
attack:	<p>Amount of time (in beats) for sound to reach full amplitude (<code>attack_level</code>). A short attack (i.e. 0.01) makes the initial part of the sound very percussive like a sharp tap. A longer attack (i.e 1) fades the sound in gently. Full length of sound is attack + decay + sustain + release.</p> <p><i>Default: 0</i> <i>Must be zero or greater</i> <i>Can not be changed once set</i> <i>Scaled with current BPM value</i></p>
decay:	<p>Amount of time (in beats) for the sound to move from full amplitude (<code>attack_level</code>) to the sustain amplitude (<code>sustain_level</code>).</p> <p><i>Default: 0</i> <i>Must be zero or greater</i> <i>Can not be changed once set</i> <i>Scaled with current BPM value</i></p>
sustain:	<p>Amount of time (in beats) for sound to remain at sustain level amplitude. Longer sustain values result in longer sounds. Full length of sound is attack + decay + sustain + release.</p> <p><i>Default: 0</i> <i>Must be zero or greater</i> <i>Can not be changed once set</i> <i>Scaled with current BPM value</i></p>
release:	<p>Amount of time (in beats) for sound to move from sustain level amplitude to silent. A short release (i.e. 0.01) makes the final part of the sound very percussive (potentially resulting in a click). A longer release (i.e 1) fades the sound out gently. Full length of sound is attack + decay + sustain + release.</p> <p><i>Default: 1</i> <i>Must be zero or greater</i> <i>Can not be changed once set</i> <i>Scaled with current BPM value</i></p>
attack_level:	<p>Amplitude level reached after attack phase and immediately before decay phase</p> <p><i>Default: 1</i> <i>Must be zero or greater</i> <i>Can not be changed once set</i></p>
decay_level:	<p>Amplitude level reached after decay phase and immediately before sustain phase. Defaults to <code>sustain_level</code> unless explicitly set</p> <p><i>Default: sustain_level</i> <i>Must be zero or greater</i> <i>Can not be changed once set</i></p>
	<p>Amplitude level reached after decay phase and immediately before release phase.</p>



<b>sustain_level:</b>	<p><i>Default: 1</i>  <i>Must be zero or greater</i>  <i>Can not be changed once set</i></p>
<b>env_curve:</b>	<p>Select the shape of the curve between levels in the envelope. 1=linear, 2=exponential, 3=sine, 4=welch, 6=squared, 7=cubed</p> <p><i>Default: 2</i>  <i>Must be one of the following values: [1, 2, 3, 4, 6, 7]</i>  <i>Can not be changed once set</i></p>
<b>cutoff:</b>	<p>MIDI note representing the highest frequencies allowed to be present in the sound. A low value like 30 makes the sound round and dull, a high value like 100 makes the sound buzzy and crispy.</p> <p><i>Default: 130</i>  <i>Must be zero or greater, must be a value less than 131</i>  <i>May be changed whilst playing</i>  <i>Has slide parameters to shape changes</i></p>
<b>res:</b>	<p>Filter resonance as a value between 0 and 1. Large amounts of resonance (a res: near 1) can create a whistling sound around the cutoff frequency. Smaller values produce less resonance.</p> <p><i>Default: 0.7</i>  <i>Must be zero or greater, must be a value less than 1</i>  <i>May be changed whilst playing</i>  <i>Has slide parameters to shape changes</i></p>

## Slide Parameters

Any parameter that is slidable has three additional parameters named `_slide`, `_slide_curve`, and `_slide_shape`. For example, 'amp' is slidable, so you can also set `amp_slide`, `amp_slide_curve`, and `amp_slide_shape` with the following effects:

<b>_slide:</b>	<p>Amount of time (in beats) for the parameter value to change. A long parameter_slide value means that the parameter takes a long time to slide from the previous value to the new value. A parameter_slide of 0 means that the parameter instantly changes to the new value.</p> <p><i>Default: 0</i></p>
<b>_slide_shape:</b>	<p>Shape of curve. 0: step, 1: linear, 3: sine, 4: welch, 5: custom (use *_slide_curve: opt e.g. amp_slide_curve:), 6: squared, 7: cubed.</p> <p><i>Default: 5</i></p>
<b>_slide_curve:</b>	<p>Shape of the slide curve (only honoured if slide shape is 5). 0 means linear and positive and negative numbers curve the segment up and down respectively.</p> <p><i>Default: 0</i></p>

# TB-303 Emulation

note:	52	amp:	1	pan:	0	attack:	0
release:	1	attack_level:	1	decay_level:	sustain_level	sustain_level:	1
cutoff_min:	30	cutoff_attack:	attack	cutoff_decay:	decay	cutoff_sustain:	sust
cutoff_decay_level:	cutoff_sustain_level	cutoff_sustain_level:	1	res:	0.9	wave:	0

use\_synth :tb303

Emulation of the classic Roland TB-303 Bass Line synthesiser. Overdrive the res (i.e. use very large values) for that classic late 80s acid sound.

Introduced in v2.0

## Parameters

note:	Note to play. Either a MIDI number or a symbol representing a note. For example: <b>30, 52, :C, :C2, :Eb4</b> , or <b>:Ds3</b>  <i>Default: 52</i> <i>Must be zero or greater</i> <i>May be changed whilst playing</i> <i>Has slide parameters to shape changes</i>
amp:	The amplitude of the sound. Typically a value between 0 and 1. Higher amplitudes may be used, but won't make the sound louder, they will just reduce the quality of all the sounds currently being played (due to compression.)  <i>Default: 1</i> <i>Must be zero or greater</i> <i>May be changed whilst playing</i> <i>Has slide parameters to shape changes</i>
pan:	Position of sound in stereo. With headphones on, this means how much of the sound is in the left ear, and how much is in the right ear. With a value of -1, the sound is completely in the left ear, a value of 0 puts the sound equally in both ears and a value of 1 puts the sound in the right ear. Values in between -1 and 1 move the sound accordingly.  <i>Default: 0</i> <i>Must be a value between -1 and 1 inclusively</i> <i>May be changed whilst playing</i> <i>Has slide parameters to shape changes</i>
attack:	Amount of time (in beats) for sound to reach full amplitude (attack_level). A short attack (i.e. 0.01) makes the initial part of the sound very percussive like a sharp tap. A longer attack (i.e 1) fades the sound in gently. Full length of sound is attack + decay + sustain + release.  <i>Default: 0</i> <i>Must be zero or greater</i> <i>Can not be changed once set</i> <i>Scaled with current BPM value</i>
decay:	Amount of time (in beats) for the sound to move from full amplitude (attack_level) to the sustain amplitude (sustain_level).  <i>Default: 0</i> <i>Must be zero or greater</i> <i>Can not be changed once set</i> <i>Scaled with current BPM value</i>
sustain:	Amount of time (in beats) for sound to remain at sustain level amplitude. Longer sustain values result in longer sounds. Full length of sound is attack + decay + sustain + release.  <i>Default: 0</i> <i>Must be zero or greater</i> <i>Can not be changed once set</i> <i>Scaled with current BPM value</i>
release:	Amount of time (in beats) for sound to move from sustain level amplitude to silent. A short release (i.e. 0.01) makes the final part of the sound very percussive (potentially resulting in a click). A longer release (i.e 1) fades the sound out gently. Full length of sound is attack + decay + sustain + release.  <i>Default: 1</i> <i>Must be zero or greater</i> <i>Can not be changed once set</i> <i>Scaled with current BPM value</i>
attack_level:	Amplitude level reached after attack phase and immediately before decay phase  <i>Default: 1</i> <i>Must be zero or greater</i> <i>Can not be changed once set</i>
	Amplitude level reached after decay phase and immediately before sustain phase. Defaults to sustain_level unless

<b>decay_level:</b>	explicitly set <i>Default: sustain_level</i> <i>Must be zero or greater</i> <i>Can not be changed once set</i>
<b>sustain_level:</b>	Amplitude level reached after decay phase and immediately before release phase. <i>Default: 1</i> <i>Must be zero or greater</i> <i>Can not be changed once set</i>
<b>env_curve:</b>	Select the shape of the curve between levels in the envelope. 1=linear, 2=exponential, 3=sine, 4=welch, 6=squared, 7=cubed <i>Default: 2</i> <i>Must be one of the following values: [1, 2, 3, 4, 6, 7]</i> <i>Can not be changed once set</i>
<b>cutoff:</b>	The maximum cutoff value as a MIDI note <i>Default: 120</i> <i>Must be a value less than or equal to 130</i> <i>May be changed whilst playing</i> <i>Has slide parameters to shape changes</i>
<b>cutoff_min:</b>	The minimum cutoff value. <i>Default: 30</i> <i>Must be a value less than or equal to 130</i> <i>May be changed whilst playing</i> <i>Has slide parameters to shape changes</i>
<b>cutoff_attack:</b>	Attack time for cutoff filter. Amount of time (in beats) for sound to reach full cutoff value. Default value is set to match amp envelope's attack value. <i>Default: attack</i> <i>Must be zero or greater</i> <i>Can not be changed once set</i> <i>Scaled with current BPM value</i>
<b>cutoff_decay:</b>	Decay time for cutoff filter. Amount of time (in beats) for sound to reach full cutoff value. Default value is set to match amp envelope's decay value. <i>Default: decay</i> <i>Must be zero or greater</i> <i>Can not be changed once set</i> <i>Scaled with current BPM value</i>
<b>cutoff_sustain:</b>	Amount of time for cutoff value to remain at sustain level in beats. Default value is set to match amp envelope's sustain value. <i>Default: sustain</i> <i>Must be zero or greater</i> <i>Can not be changed once set</i> <i>Scaled with current BPM value</i>
<b>cutoff_release:</b>	Amount of time (in beats) for sound to move from cutoff sustain value to cutoff min value. Default value is set to match amp envelope's release value. <i>Default: release</i> <i>Must be zero or greater</i> <i>Can not be changed once set</i> <i>Scaled with current BPM value</i>
<b>cutoff_attack_level:</b>	The peak cutoff (value of cutoff at peak of attack) as a value between 0 and 1 where 0 is the :cutoff_min and 1 is the :cutoff value <i>Default: 1</i> <i>Must be a value between 0 and 1 inclusively</i> <i>Can not be changed once set</i>
<b>cutoff_decay_level:</b>	The level of cutoff after the decay phase as a value between 0 and 1 where 0 is the :cutoff_min and 1 is the :cutoff value <i>Default: cutoff_sustain_level</i> <i>Must be a value between 0 and 1 inclusively</i> <i>Can not be changed once set</i>
<b>cutoff_sustain_level:</b>	The sustain cutoff (value of cutoff at sustain time) as a value between 0 and 1 where 0 is the :cutoff_min and 1 is the :cutoff value. <i>Default: 1</i> <i>Must be a value between 0 and 1 inclusively</i> <i>Can not be changed once set</i>
	Filter resonance as a value between 0 and 1. Large amounts of resonance (a res: near 1) can create a whistling sound around the cutoff frequency. Smaller values produce less resonance.

res:	<p><i>Default: 0.9</i>  <i>Must be zero or greater, must be a value less than 1</i>  <i>May be changed whilst playing</i>  <a href="#">Has slide parameters to shape changes</a></p>
wave:	<p>Wave type - 0 saw, 1 pulse, 2 triangle. Different waves will produce different sounds.</p> <p><i>Default: 0</i>  <i>Must be one of the following values: [0, 1, 2]</i>  <i>May be changed whilst playing</i></p>
pulse_width:	<p>The width of the pulse wave as a value between 0 and 1. A width of 0.5 will produce a square wave. Different values will change the timbre of the sound. Only valid if wave is type pulse.</p> <p><i>Default: 0.5</i>  <i>Must be a value between 0 and 1 exclusively</i>  <i>May be changed whilst playing</i>  <a href="#">Has slide parameters to shape changes</a></p>

## Slide Parameters

Any parameter that is slidable has three additional parameters named `_slide`, `_slide_curve`, and `_slide_shape`. For example, 'amp' is slidable, so you can also set `amp_slide`, `amp_slide_curve`, and `amp_slide_shape` with the following effects:

<code>_slide</code> :	<p>Amount of time (in beats) for the parameter value to change. A long parameter <code>_slide</code> value means that the parameter takes a long time to slide from the previous value to the new value. A parameter <code>_slide</code> of 0 means that the parameter instantly changes to the new value.</p> <p><i>Default: 0</i></p>
<code>_slide_shape</code> :	<p>Shape of curve. 0: step, 1: linear, 3: sine, 4: welch, 5: custom (use <code>*_slide_curve</code>: opt e.g. <code>amp_slide_curve</code>), 6: squared, 7: cubed.</p> <p><i>Default: 5</i></p>
<code>_slide_curve</code> :	<p>Shape of the slide curve (only honoured if slide shape is 5). 0 means linear and positive and negative numbers curve the segment up and down respectively.</p> <p><i>Default: 0</i></p>

# Triangle Wave

note:	52	amp:	1	pan:	0	attack:	0	decay:	0	sustain:	0
release:	1	attack_level:	1	decay_level:	sustain_level	sustain_level:	1	env_curve:	2	cutoff:	100
pulse_width:	0.5										

`use_synth :tri`

A simple triangle wave with a low pass filter.

Introduced in v2.0

## Parameters

note:	<p>Note to play. Either a MIDI number or a symbol representing a note. For example:<code>30</code>, <code>52</code>, <code>:C</code>, <code>:C2</code>, <code>:Eb4</code>, or <code>:Ds3</code></p> <p><i>Default: 52</i> <i>Must be zero or greater</i> <i>May be changed whilst playing</i> <i>Has slide parameters to shape changes</i></p>
amp:	<p>The amplitude of the sound. Typically a value between 0 and 1. Higher amplitudes may be used, but won't make the sound louder, they will just reduce the quality of all the sounds currently being played (due to compression.)</p> <p><i>Default: 1</i> <i>Must be zero or greater</i> <i>May be changed whilst playing</i> <i>Has slide parameters to shape changes</i></p>
pan:	<p>Position of sound in stereo. With headphones on, this means how much of the sound is in the left ear, and how much is in the right ear. With a value of -1, the sound is completely in the left ear, a value of 0 puts the sound equally in both ears and a value of 1 puts the sound in the right ear. Values in between -1 and 1 move the sound accordingly.</p> <p><i>Default: 0</i> <i>Must be a value between -1 and 1 inclusively</i> <i>May be changed whilst playing</i> <i>Has slide parameters to shape changes</i></p>
attack:	<p>Amount of time (in beats) for sound to reach full amplitude (<code>attack_level</code>). A short attack (i.e. 0.01) makes the initial part of the sound very percussive like a sharp tap. A longer attack (i.e 1) fades the sound in gently. Full length of sound is <code>attack</code> + <code>decay</code> + <code>sustain</code> + <code>release</code>.</p> <p><i>Default: 0</i> <i>Must be zero or greater</i> <i>Can not be changed once set</i> <i>Scaled with current BPM value</i></p>
decay:	<p>Amount of time (in beats) for the sound to move from full amplitude (<code>attack_level</code>) to the sustain amplitude (<code>sustain_level</code>).</p> <p><i>Default: 0</i> <i>Must be zero or greater</i> <i>Can not be changed once set</i> <i>Scaled with current BPM value</i></p>
sustain:	<p>Amount of time (in beats) for sound to remain at sustain level amplitude. Longer sustain values result in longer sounds. Full length of sound is <code>attack</code> + <code>decay</code> + <code>sustain</code> + <code>release</code>.</p> <p><i>Default: 0</i> <i>Must be zero or greater</i> <i>Can not be changed once set</i> <i>Scaled with current BPM value</i></p>
release:	<p>Amount of time (in beats) for sound to move from sustain level amplitude to silent. A short release (i.e. 0.01) makes the final part of the sound very percussive (potentially resulting in a click). A longer release (i.e 1) fades the sound out gently. Full length of sound is <code>attack</code> + <code>decay</code> + <code>sustain</code> + <code>release</code>.</p> <p><i>Default: 1</i> <i>Must be zero or greater</i> <i>Can not be changed once set</i> <i>Scaled with current BPM value</i></p>
attack_level:	<p>Amplitude level reached after attack phase and immediately before decay phase</p> <p><i>Default: 1</i> <i>Must be zero or greater</i> <i>Can not be changed once set</i></p>
decay_level:	<p>Amplitude level reached after decay phase and immediately before sustain phase. Defaults to <code>sustain_level</code> unless explicitly set</p> <p><i>Default: sustain_level</i> <i>Must be zero or greater</i> <i>Can not be changed once set</i></p>
	<p>Amplitude level reached after decay phase and immediately before release phase.</p>

<b>sustain_level:</b>	<p><i>Default: 1</i>  <i>Must be zero or greater</i>  <i>Can not be changed once set</i></p>
<b>env_curve:</b>	<p>Select the shape of the curve between levels in the envelope. 1=linear, 2=exponential, 3=sine, 4=welch, 6=squared, 7=cubed</p> <p><i>Default: 2</i>  <i>Must be one of the following values: [1, 2, 3, 4, 6, 7]</i>  <i>Can not be changed once set</i></p>
<b>cutoff:</b>	<p>MIDI note representing the highest frequencies allowed to be present in the sound. A low value like 30 makes the sound round and dull, a high value like 100 makes the sound buzzy and crispy.</p> <p><i>Default: 100</i>  <i>Must be zero or greater, must be a value less than 131</i>  <i>May be changed whilst playing</i>  <i>Has slide parameters to shape changes</i></p>
<b>pulse_width:</b>	<p>The width of the pulse wave as a value between 0 and 1. A width of 0.5 will produce a square wave. Different values will change the timbre of the sound. Only valid if wave is type pulse.</p> <p><i>Default: 0.5</i>  <i>Must be a value between 0 and 1 exclusively</i>  <i>May be changed whilst playing</i>  <i>Has slide parameters to shape changes</i></p>

## Slide Parameters

Any parameter that is slidable has three additional parameters named `_slide`, `_slide_curve`, and `_slide_shape`. For example, 'amp' is slidable, so you can also set `amp_slide`, `amp_slide_curve`, and `amp_slide_shape` with the following effects:

<b>_slide:</b>	<p>Amount of time (in beats) for the parameter value to change. A long <code>parameter_slide</code> value means that the parameter takes a long time to slide from the previous value to the new value. A <code>parameter_slide</code> of 0 means that the parameter instantly changes to the new value.</p> <p><i>Default: 0</i></p>
<b>_slide_shape:</b>	<p>Shape of curve. 0: step, 1: linear, 3: sine, 4: welch, 5: custom (use <code>*_slide_curve</code>: opt e.g. <code>amp_slide_curve</code>); 6: squared, 7: cubed.</p> <p><i>Default: 5</i></p>
<b>_slide_curve:</b>	<p>Shape of the slide curve (only honoured if slide shape is 5). 0 means linear and positive and negative numbers curve the segment up and down respectively.</p> <p><i>Default: 0</i></p>

# Zawa

note:	52	amp:	1	pan:	0	attack:	0	decay:	0	sustain:	0
release:	1	attack_level:	1	decay_level:	sustain_level	sustain_level:	1	cutoff:	100	res:	0.9
phase:	1	phase_offset:	0	wave:	3	invert_wave:	0	range:	24	disable_wave:	0
pulse_width:	0.5										

use\_synth :zawa

Saw wave with oscillating timbre. Produces moving saw waves with a unique character controllable with the control oscillator (usage similar to mod synths).

Introduced in v2.0

## Parameters

note:	<p>Note to play. Either a MIDI number or a symbol representing a note. For example:<b>30</b>, <b>52</b>, <b>:C</b>, <b>:C2</b>, <b>:Eb4</b>, or <b>:Ds3</b></p> <p><i>Default: 52</i> <i>Must be zero or greater</i> <i>May be changed whilst playing</i> <a href="#">Has slide parameters to shape changes</a></p>
amp:	<p>The amplitude of the sound. Typically a value between 0 and 1. Higher amplitudes may be used, but won't make the sound louder, they will just reduce the quality of all the sounds currently being played (due to compression.)</p> <p><i>Default: 1</i> <i>Must be zero or greater</i> <i>May be changed whilst playing</i> <a href="#">Has slide parameters to shape changes</a></p>
pan:	<p>Position of sound in stereo. With headphones on, this means how much of the sound is in the left ear, and how much is in the right ear. With a value of -1, the sound is completely in the left ear, a value of 0 puts the sound equally in both ears and a value of 1 puts the sound in the right ear. Values in between -1 and 1 move the sound accordingly.</p> <p><i>Default: 0</i> <i>Must be a value between -1 and 1 inclusively</i> <i>May be changed whilst playing</i> <a href="#">Has slide parameters to shape changes</a></p>
attack:	<p>Amount of time (in beats) for sound to reach full amplitude (attack_level). A short attack (i.e. 0.01) makes the initial part of the sound very percussive like a sharp tap. A longer attack (i.e 1) fades the sound in gently. Full length of sound is attack + decay + sustain + release.</p> <p><i>Default: 0</i> <i>Must be zero or greater</i> <i>Can not be changed once set</i> <i>Scaled with current BPM value</i></p>
decay:	<p>Amount of time (in beats) for the sound to move from full amplitude (attack_level) to the sustain amplitude (sustain_level).</p> <p><i>Default: 0</i> <i>Must be zero or greater</i> <i>Can not be changed once set</i> <i>Scaled with current BPM value</i></p>
sustain:	<p>Amount of time (in beats) for sound to remain at sustain level amplitude. Longer sustain values result in longer sounds. Full length of sound is attack + decay + sustain + release.</p> <p><i>Default: 0</i> <i>Must be zero or greater</i> <i>Can not be changed once set</i> <i>Scaled with current BPM value</i></p>
release:	<p>Amount of time (in beats) for sound to move from sustain level amplitude to silent. A short release (i.e. 0.01) makes the final part of the sound very percussive (potentially resulting in a click). A longer release (i.e 1) fades the sound out gently. Full length of sound is attack + decay + sustain + release.</p> <p><i>Default: 1</i> <i>Must be zero or greater</i> <i>Can not be changed once set</i> <i>Scaled with current BPM value</i></p>
attack_level:	<p>Amplitude level reached after attack phase and immediately before decay phase</p> <p><i>Default: 1</i> <i>Must be zero or greater</i> <i>Can not be changed once set</i></p>
decay_level:	<p>Amplitude level reached after decay phase and immediately before sustain phase. Defaults to sustain_level unless explicitly set</p> <p><i>Default: sustain_level</i> <i>Must be zero or greater</i></p>

<b>sustain_level:</b>	<p>Can not be changed once set Amplitude level reached after decay phase and immediately before release phase.</p> <p><i>Default: 1</i> <i>Must be zero or greater</i> <i>Can not be changed once set</i></p>
<b>cutoff:</b>	<p>MIDI note representing the highest frequencies allowed to be present in the sound. A low value like 30 makes the sound round and dull, a high value like 100 makes the sound buzzy and crispy.</p> <p><i>Default: 100</i> <i>Must be zero or greater, must be a value less than 131</i> <i>May be changed whilst playing</i> <i>Has slide parameters to shape changes</i></p>
<b>res:</b>	<p>Filter resonance as a value between 0 and 1. Large amounts of resonance (a res: near 1) can create a whistling sound around the cutoff frequency. Smaller values produce less resonance.</p> <p><i>Default: 0.9</i> <i>Must be zero or greater, must be a value less than 1</i> <i>May be changed whilst playing</i> <i>Has slide parameters to shape changes</i></p>
<b>phase:</b>	<p>Phase duration in beats of timbre modulation.</p> <p><i>Default: 1</i> <i>Must be greater than zero</i> <i>May be changed whilst playing</i> <i>Has slide parameters to shape changes</i> <i>Scaled with current BPM value</i></p>
<b>phase_offset:</b>	<p>Initial phase offset of the sync wave (a value between 0 and 1).</p> <p><i>Default: 0</i> <i>Must be a value between 0 and 1 inclusively</i> <i>Can not be changed once set</i></p>
<b>wave:</b>	<p>Wave shape controlling freq sync saw wave. 0=saw wave, 1=pulse, 2=triangle wave and 3=sine wave.</p> <p><i>Default: 3</i> <i>Must be one of the following values: [0, 1, 2, 3]</i> <i>May be changed whilst playing</i></p>
<b>invert_wave:</b>	<p>Invert sync freq control waveform (i.e. flip it on the y axis). 0=uninverted wave, 1=inverted wave.</p> <p><i>Default: 0</i> <i>Must be one of the following values: [0, 1]</i> <i>May be changed whilst playing</i></p>
<b>range:</b>	<p>Range of the associated sync saw in MIDI notes from the main note. Modifies timbre.</p> <p><i>Default: 24</i> <i>Must be a value between 0 and 90 inclusively</i> <i>May be changed whilst playing</i> <i>Has slide parameters to shape changes</i></p>
<b>disable_wave:</b>	<p>Enable and disable sync control wave (setting to 1 will stop timbre movement).</p> <p><i>Default: 0</i> <i>Must be one of the following values: [0, 1]</i> <i>May be changed whilst playing</i></p>
<b>pulse_width:</b>	<p>The width of the pulse wave as a value between 0 and 1. A width of 0.5 will produce a square wave. Different values will change the timbre of the sound. Only valid if wave is type pulse.</p> <p><i>Default: 0.5</i> <i>Must be a value between 0 and 1 exclusively</i> <i>May be changed whilst playing</i> <i>Has slide parameters to shape changes</i></p>

## Slide Parameters

Any parameter that is slidable has three additional parameters named `_slide`, `_slide_curve`, and `_slide_shape`. For example, 'amp' is slidable, so you can also set `amp_slide`, `amp_slide_curve`, and `amp_slide_shape` with the following effects:

<b>_slide:</b>	<p>Amount of time (in beats) for the parameter value to change. A long parameter <code>_slide</code> value means that the parameter takes a long time to slide from the previous value to the new value. A parameter <code>_slide</code> of 0 means that the parameter instantly changes to the new value.</p> <p><i>Default: 0</i></p>
<b>_slide_shape:</b>	<p>Shape of curve. 0: step, 1: linear, 3: sine, 4: welch, 5: custom (use <code>*_slide_curve</code>: opt e.g. <code>amp_slide_curve</code>); 6: squared, 7: cubed.</p> <p><i>Default: 5</i></p>



`_slide_curve:`

Shape of the slide curve (only honoured if slide shape is 5). 0 means linear and positive and negative numbers curve the segment up and down respectively.

*Default: 0*

---

Ambient Sounds  
Bass Drums  
Bass Sounds  
Drum Sounds  
Electric Sounds  
Miscellaneous Sounds  
Percussive Sounds  
Snare Drums  
Sounds featuring guitars  
Sounds for Looping

# Ambient Sounds

amp:	1	pan:	0	attack:	0	decay:	0	sustain:	-1	release:	0
attack_level:	1	decay_level:	sustain_level	sustain_level:	1	env_curve:	2	rate:	1	start:	0
finish:	1	res:	0	cutoff:	0	norm:	0				

sample :ambi\_soft\_buzz  
 sample :ambi\_swoosh  
 sample :ambi\_drone  
 sample :ambi\_glass\_hum  
 sample :ambi\_glass\_rub  
 sample :ambi\_haunted\_hum  
 sample :ambi\_piano  
 sample :ambi\_lunar\_land  
 sample :ambi\_dark\_woosh  
 sample :ambi\_choir

amp:	<p>The amplitude of the sound. Typically a value between 0 and 1. Higher amplitudes may be used, but won't make the sound louder, they will just reduce the quality of all the sounds currently being played (due to compression.)</p> <p><i>Default: 1</i>  <i>must be zero or greater</i>  <i>May be changed whilst playing</i></p>
pan:	<p>Position of sound in stereo. With headphones on, this means how much of the sound is in the left ear, and how much is in the right ear. With a value of -1, the sound is completely in the left ear, a value of 0 puts the sound equally in both ears and a value of 1 puts the sound in the right ear. Values in between -1 and 1 move the sound accordingly.</p> <p><i>Default: 0</i>  <i>must be a value between -1 and 1 inclusively</i>  <i>May be changed whilst playing</i></p>
attack:	<p>Duration of the attack phase of the envelope.</p> <p><i>Default: 0</i>  <i>must be zero or greater</i></p>
decay:	<p>Duration of the decay phase of the envelope.</p> <p><i>Default: 0</i>  <i>must be zero or greater</i></p>
sustain:	<p>Duration of the sustain phase of the envelope.</p> <p><i>Default: -1</i>  <i>must either be a positive value or -1</i></p>
release:	<p>Duration of the release phase of the envelope.</p> <p><i>Default: 0</i>  <i>must be zero or greater</i></p>
attack_level:	<p>Amplitude level reached after attack phase and immediately before decay phase</p> <p><i>Default: 1</i>  <i>must be zero or greater</i></p>
decay_level:	<p>Amplitude level reached after decay phase and immediately before sustain phase. Defaults to sustain_level unless explicitly set</p> <p><i>Default: sustain_level</i>  <i>must be zero or greater</i></p>

<b>sustain_level:</b>	<p>Amplitude level reached after decay phase and immediately before release phase.</p> <p><i>Default: 1</i> <i>must be zero or greater</i></p>
<b>env_curve:</b>	<p>Select the shape of the curve between levels in the envelope. 1=linear, 2=exponential, 3=sine, 4=welch, 6=squared, 7=cubed</p> <p><i>Default: 2</i> <i>must be one of the following values: [1, 2, 3, 4, 6, 7]</i></p>
<b>rate:</b>	<p>Rate with which to play back - default is 1. Playing the sample at rate 2 will play it back at double the normal speed. This will have the effect of doubling the frequencies in the sample and halving the playback time. Use rates lower than 1 to slow the sample down. Negative rates will play the sample in reverse.</p> <p><i>Default: 1</i> <i>must not be zero</i></p>
<b>start:</b>	<p>A fraction (between 0 and 1) representing where in the sample to start playback. 1 represents the end of the sample, 0.5 half-way through etc.</p> <p><i>Default: 0</i> <i>must be a value between 0 and 1 inclusively</i></p>
<b>finish:</b>	<p>A fraction (between 0 and 1) representing where in the sample to finish playback. 1 represents the end of the sample, 0.5 half-way through etc.</p> <p><i>Default: 1</i> <i>must be a value between 0 and 1 inclusively</i></p>
<b>res:</b>	<p>Filter resonance as a value between 0 and 1. Only functional if a cutoff value is specified. Large amounts of resonance (a res: near 1) can create a whistling sound around the cutoff frequency. Smaller values produce less resonance.</p> <p><i>Default: 0</i> <i>must be zero or greater, must be a value less than 1</i> <i>May be changed whilst playing</i></p>
<b>cutoff:</b>	<p>MIDI note representing the highest frequencies allowed to be present in the sound. A low value like 30 makes the sound round and dull, a high value like 100 makes the sound buzzy and crispy.</p> <p><i>Default: 0</i> <i>must be zero or greater, must be a value less than 131</i> <i>May be changed whilst playing</i></p>
<b>norm:</b>	<p>Normalise the audio (make quieter parts of the sample louder and louder parts quieter) - this is similar to the normaliser FX. This may emphasise any clicks caused by clipping.</p> <p><i>Default: 0</i> <i>must be one of the following values: [0, 1]</i></p>

---

# Bass Drums

amp:	1	pan:	0	attack:	0	decay:	0	sustain:	-1	release:	0
attack_level:	1	decay_level:	sustain_level	sustain_level:	1	env_curve:	2	rate:	1	start:	0
finish:	1	res:	0	cutoff:	0	norm:	0				

sample :bd\_ada  
 sample :bd\_pure  
 sample :bd\_808  
 sample :bd\_zum  
 sample :bd\_gas  
 sample :bd\_sone  
 sample :bd\_haus  
 sample :bd\_zome  
 sample :bd\_boom  
 sample :bd\_klub  
 sample :bd\_fat  
 sample :bd\_tek

amp:	<p>The amplitude of the sound. Typically a value between 0 and 1. Higher amplitudes may be used, but won't make the sound louder, they will just reduce the quality of all the sounds currently being played (due to compression.)</p> <p><i>Default: 1</i>  <i>must be zero or greater</i>  <i>May be changed whilst playing</i></p>
pan:	<p>Position of sound in stereo. With headphones on, this means how much of the sound is in the left ear, and how much is in the right ear. With a value of -1, the sound is completely in the left ear, a value of 0 puts the sound equally in both ears and a value of 1 puts the sound in the right ear. Values in between -1 and 1 move the sound accordingly.</p> <p><i>Default: 0</i>  <i>must be a value between -1 and 1 inclusively</i>  <i>May be changed whilst playing</i></p>
attack:	<p>Duration of the attack phase of the envelope.</p> <p><i>Default: 0</i>  <i>must be zero or greater</i></p>
decay:	<p>Duration of the decay phase of the envelope.</p> <p><i>Default: 0</i>  <i>must be zero or greater</i></p>
sustain:	<p>Duration of the sustain phase of the envelope.</p> <p><i>Default: -1</i>  <i>must either be a positive value or -1</i></p>
release:	<p>Duration of the release phase of the envelope.</p> <p><i>Default: 0</i>  <i>must be zero or greater</i></p>
attack_level:	<p>Amplitude level reached after attack phase and immediately before decay phase</p> <p><i>Default: 1</i>  <i>must be zero or greater</i></p>
	<p>Amplitude level reached after decay phase and immediately before sustain phase. Defaults to sustain_level unless explicitly set</p>

<b>decay_level:</b>	<i>Default: sustain_level</i> <i>must be zero or greater</i>
<b>sustain_level:</b>	Amplitude level reached after decay phase and immediately before release phase. <i>Default: 1</i> <i>must be zero or greater</i>
<b>env_curve:</b>	Select the shape of the curve between levels in the envelope. 1=linear, 2=exponential, 3=sine, 4=welch, 6=squared, 7=cubed <i>Default: 2</i> <i>must be one of the following values: [1, 2, 3, 4, 6, 7]</i>
<b>rate:</b>	Rate with which to play back - default is 1. Playing the sample at rate 2 will play it back at double the normal speed. This will have the effect of doubling the frequencies in the sample and halving the playback time. Use rates lower than 1 to slow the sample down. Negative rates will play the sample in reverse. <i>Default: 1</i> <i>must not be zero</i>
<b>start:</b>	A fraction (between 0 and 1) representing where in the sample to start playback. 1 represents the end of the sample, 0.5 half-way through etc. <i>Default: 0</i> <i>must be a value between 0 and 1 inclusively</i>
<b>finish:</b>	A fraction (between 0 and 1) representing where in the sample to finish playback. 1 represents the end of the sample, 0.5 half-way through etc. <i>Default: 1</i> <i>must be a value between 0 and 1 inclusively</i>
<b>res:</b>	Filter resonance as a value between 0 and 1. Only functional if a cutoff value is specified. Large amounts of resonance (a res: near 1) can create a whistling sound around the cutoff frequency. Smaller values produce less resonance. <i>Default: 0</i> <i>must be zero or greater, must be a value less than 1</i> <i>May be changed whilst playing</i>
<b>cutoff:</b>	MIDI note representing the highest frequencies allowed to be present in the sound. A low value like 30 makes the sound round and dull, a high value like 100 makes the sound buzzy and crispy. <i>Default: 0</i> <i>must be zero or greater, must be a value less than 131</i> <i>May be changed whilst playing</i>
<b>norm:</b>	Normalise the audio (make quieter parts of the sample louder and louder parts quieter) - this is similar to the normaliser FX. This may emphasise any clicks caused by clipping. <i>Default: 0</i> <i>must be one of the following values: [0, 1]</i>

---

# Bass Sounds

amp:	1	pan:	0	attack:	0	decay:	0	sustain:	-1	release:	0
attack_level:	1	decay_level:	sustain_level	sustain_level:	1	env_curve:	2	rate:	1	start:	0
finish:	1	res:	0	cutoff:	0	norm:	0				

```

sample :bass_hit_c
sample :bass_hard_c
sample :bass_thick_c
sample :bass_drop_c
sample :bass_woosy_c
sample :bass_voxy_c
sample :bass_voxy_hit_c
sample :bass_dnb_f
  
```

amp:	The amplitude of the sound. Typically a value between 0 and 1. Higher amplitudes may be used, but won't make the sound louder, they will just reduce the quality of all the sounds currently being played (due to compression.)  <i>Default: 1</i> <i>must be zero or greater</i> <i>May be changed whilst playing</i>
pan:	Position of sound in stereo. With headphones on, this means how much of the sound is in the left ear, and how much is in the right ear. With a value of -1, the sound is completely in the left ear, a value of 0 puts the sound equally in both ears and a value of 1 puts the sound in the right ear. Values in between -1 and 1 move the sound accordingly.  <i>Default: 0</i> <i>must be a value between -1 and 1 inclusively</i> <i>May be changed whilst playing</i>
attack:	Duration of the attack phase of the envelope.  <i>Default: 0</i> <i>must be zero or greater</i>
decay:	Duration of the decay phase of the envelope.  <i>Default: 0</i> <i>must be zero or greater</i>
sustain:	Duration of the sustain phase of the envelope.  <i>Default: -1</i> <i>must either be a positive value or -1</i>
release:	Duration of the release phase of the envelope.  <i>Default: 0</i> <i>must be zero or greater</i>
attack_level:	Amplitude level reached after attack phase and immediately before decay phase  <i>Default: 1</i> <i>must be zero or greater</i>
decay_level:	Amplitude level reached after decay phase and immediately before sustain phase. Defaults to sustain_level unless explicitly set  <i>Default: sustain_level</i> <i>must be zero or greater</i>
	Amplitude level reached after decay phase and immediately before release phase.

<b>sustain_level:</b>	<p><i>Default: 1</i> <i>must be zero or greater</i></p>
<b>env_curve:</b>	<p>Select the shape of the curve between levels in the envelope. 1=linear, 2=exponential, 3=sine, 4=welch, 6=squared, 7=cubed</p> <p><i>Default: 2</i> <i>must be one of the following values: [1, 2, 3, 4, 6, 7]</i></p>
<b>rate:</b>	<p>Rate with which to play back - default is 1. Playing the sample at rate 2 will play it back at double the normal speed. This will have the effect of doubling the frequencies in the sample and halving the playback time. Use rates lower than 1 to slow the sample down. Negative rates will play the sample in reverse.</p> <p><i>Default: 1</i> <i>must not be zero</i></p>
<b>start:</b>	<p>A fraction (between 0 and 1) representing where in the sample to start playback. 1 represents the end of the sample, 0.5 half-way through etc.</p> <p><i>Default: 0</i> <i>must be a value between 0 and 1 inclusively</i></p>
<b>finish:</b>	<p>A fraction (between 0 and 1) representing where in the sample to finish playback. 1 represents the end of the sample, 0.5 half-way through etc.</p> <p><i>Default: 1</i> <i>must be a value between 0 and 1 inclusively</i></p>
<b>res:</b>	<p>Filter resonance as a value between 0 and 1. Only functional if a cutoff value is specified. Large amounts of resonance (a res: near 1) can create a whistling sound around the cutoff frequency. Smaller values produce less resonance.</p> <p><i>Default: 0</i> <i>must be zero or greater, must be a value less than 1</i> <i>May be changed whilst playing</i></p>
<b>cutoff:</b>	<p>MIDI note representing the highest frequencies allowed to be present in the sound. A low value like 30 makes the sound round and dull, a high value like 100 makes the sound buzzy and crispy.</p> <p><i>Default: 0</i> <i>must be zero or greater, must be a value less than 131</i> <i>May be changed whilst playing</i></p>
<b>norm:</b>	<p>Normalise the audio (make quieter parts of the sample louder and louder parts quieter) - this is similar to the normaliser FX. This may emphasise any clicks caused by clipping.</p> <p><i>Default: 0</i> <i>must be one of the following values: [0, 1]</i></p>

---



# Drum Sounds

amp:	1	pan:	0	attack:	0	decay:	0	sustain:	-1	release:	0
attack_level:	1	decay_level:	sustain_level	sustain_level:	1	env_curve:	2	rate:	1	start:	0
finish:	1	res:	0	cutoff:	0	norm:	0				

sample :drum\_heavy\_kick  
sample :drum\_tom\_mid\_soft  
sample :drum\_tom\_mid\_hard  
sample :drum\_tom\_lo\_soft  
sample :drum\_tom\_lo\_hard  
sample :drum\_tom\_hi\_soft  
sample :drum\_tom\_hi\_hard  
sample :drum\_splash\_soft  
sample :drum\_splash\_hard  
sample :drum\_snare\_soft  
sample :drum\_snare\_hard  
sample :drum\_cymbal\_soft  
sample :drum\_cymbal\_hard  
sample :drum\_cymbal\_open  
sample :drum\_cymbal\_closed  
sample :drum\_cymbal\_pedal  
sample :drum\_bass\_soft  
sample :drum\_bass\_hard

amp:	<p>The amplitude of the sound. Typically a value between 0 and 1. Higher amplitudes may be used, but won't make the sound louder, they will just reduce the quality of all the sounds currently being played (due to compression.)</p> <p><i>Default: 1</i> <i>must be zero or greater</i> <i>May be changed whilst playing</i></p>
pan:	<p>Position of sound in stereo. With headphones on, this means how much of the sound is in the left ear, and how much is in the right ear. With a value of -1, the sound is completely in the left ear, a value of 0 puts the sound equally in both ears and a value of 1 puts the sound in the right ear. Values in between -1 and 1 move the sound accordingly.</p> <p><i>Default: 0</i> <i>must be a value between -1 and 1 inclusively</i> <i>May be changed whilst playing</i></p>
attack:	<p>Duration of the attack phase of the envelope.</p> <p><i>Default: 0</i> <i>must be zero or greater</i></p>
decay:	<p>Duration of the decay phase of the envelope.</p> <p><i>Default: 0</i> <i>must be zero or greater</i></p>
sustain:	<p>Duration of the sustain phase of the envelope.</p> <p><i>Default: -1</i> <i>must either be a positive value or -1</i></p>
release:	<p>Duration of the release phase of the envelope.</p> <p><i>Default: 0</i> <i>must be zero or greater</i></p>

<b>attack_level:</b>	Amplitude level reached after attack phase and immediately before decay phase <i>Default: 1</i> <i>must be zero or greater</i>
<b>decay_level:</b>	Amplitude level reached after decay phase and immediately before sustain phase. Defaults to sustain_level unless explicitly set <i>Default: sustain_level</i> <i>must be zero or greater</i>
<b>sustain_level:</b>	Amplitude level reached after decay phase and immediately before release phase. <i>Default: 1</i> <i>must be zero or greater</i>
<b>env_curve:</b>	Select the shape of the curve between levels in the envelope. 1=linear, 2=exponential, 3=sine, 4=welch, 6=squared, 7=cubed <i>Default: 2</i> <i>must be one of the following values: [1, 2, 3, 4, 6, 7]</i>
<b>rate:</b>	Rate with which to play back - default is 1. Playing the sample at rate 2 will play it back at double the normal speed. This will have the effect of doubling the frequencies in the sample and halving the playback time. Use rates lower than 1 to slow the sample down. Negative rates will play the sample in reverse. <i>Default: 1</i> <i>must not be zero</i>
<b>start:</b>	A fraction (between 0 and 1) representing where in the sample to start playback. 1 represents the end of the sample, 0.5 half-way through etc. <i>Default: 0</i> <i>must be a value between 0 and 1 inclusively</i>
<b>finish:</b>	A fraction (between 0 and 1) representing where in the sample to finish playback. 1 represents the end of the sample, 0.5 half-way through etc. <i>Default: 1</i> <i>must be a value between 0 and 1 inclusively</i>
<b>res:</b>	Filter resonance as a value between 0 and 1. Only functional if a cutoff value is specified. Large amounts of resonance (a res: near 1) can create a whistling sound around the cutoff frequency. Smaller values produce less resonance. <i>Default: 0</i> <i>must be zero or greater, must be a value less than 1</i> <i>May be changed whilst playing</i>
<b>cutoff:</b>	MIDI note representing the highest frequencies allowed to be present in the sound. A low value like 30 makes the sound round and dull, a high value like 100 makes the sound buzzy and crispy. <i>Default: 0</i> <i>must be zero or greater, must be a value less than 131</i> <i>May be changed whilst playing</i>
<b>norm:</b>	Normalise the audio (make quieter parts of the sample louder and louder parts quieter) - this is similar to the normaliser FX. This may emphasise any clicks caused by clipping. <i>Default: 0</i> <i>must be one of the following values: [0, 1]</i>

# Electric Sounds

amp:	1	pan:	0	attack:	0	decay:	0	sustain:	-1	release:	0
attack_level:	1	decay_level:	sustain_level	sustain_level:	1	env_curve:	2	rate:	1	start:	0
finish:	1	res:	0	cutoff:	0	norm:	0				

sample :elec\_triangle  
sample :elec\_snare  
sample :elec\_lo\_snare  
sample :elec\_hi\_snare  
sample :elec\_mid\_snare  
sample :elec\_cymbal  
sample :elec\_soft\_kick  
sample :elec\_filt\_snare  
sample :elec\_fuzz\_tom  
sample :elec\_chime  
sample :elec\_bong  
sample :elec\_twang  
sample :elec\_wood  
sample :elec\_pop  
sample :elec\_beep  
sample :elec\_blip  
sample :elec\_blip2  
sample :elec\_ping  
sample :elec\_bell  
sample :elec\_flip  
sample :elec\_tick  
sample :elec\_hollow\_kick  
sample :elec\_twip  
sample :elec\_plip  
sample :elec\_blup

amp:	<p>The amplitude of the sound. Typically a value between 0 and 1. Higher amplitudes may be used, but won't make the sound louder, they will just reduce the quality of all the sounds currently being played (due to compression.)</p> <p><i>Default: 1</i> <i>must be zero or greater</i> <i>May be changed whilst playing</i></p>
pan:	<p>Position of sound in stereo. With headphones on, this means how much of the sound is in the left ear, and how much is in the right ear. With a value of -1, the sound is completely in the left ear, a value of 0 puts the sound equally in both ears and a value of 1 puts the sound in the right ear. Values in between -1 and 1 move the sound accordingly.</p> <p><i>Default: 0</i> <i>must be a value between -1 and 1 inclusively</i> <i>May be changed whilst playing</i></p>
attack:	<p>Duration of the attack phase of the envelope.</p> <p><i>Default: 0</i> <i>must be zero or greater</i></p>
decay:	<p>Duration of the decay phase of the envelope.</p> <p><i>Default: 0</i> <i>must be zero or greater</i></p>
	<p>Duration of the sustain phase of the envelope.</p>

<b>sustain:</b>	<i>Default: -1</i> <i>must either be a positive value or -1</i>
<b>release:</b>	Duration of the release phase of the envelope. <i>Default: 0</i> <i>must be zero or greater</i>
<b>attack_level:</b>	Amplitude level reached after attack phase and immediately before decay phase <i>Default: 1</i> <i>must be zero or greater</i>
<b>decay_level:</b>	Amplitude level reached after decay phase and immediately before sustain phase. Defaults to sustain_level unless explicitly set <i>Default: sustain_level</i> <i>must be zero or greater</i>
<b>sustain_level:</b>	Amplitude level reached after decay phase and immediately before release phase. <i>Default: 1</i> <i>must be zero or greater</i>
<b>env_curve:</b>	Select the shape of the curve between levels in the envelope. 1=linear, 2=exponential, 3=sine, 4=welch, 6=squared, 7=cubed <i>Default: 2</i> <i>must be one of the following values: [1, 2, 3, 4, 6, 7]</i>
<b>rate:</b>	Rate with which to play back - default is 1. Playing the sample at rate 2 will play it back at double the normal speed. This will have the effect of doubling the frequencies in the sample and halving the playback time. Use rates lower than 1 to slow the sample down. Negative rates will play the sample in reverse. <i>Default: 1</i> <i>must not be zero</i>
<b>start:</b>	A fraction (between 0 and 1) representing where in the sample to start playback. 1 represents the end of the sample, 0.5 half-way through etc. <i>Default: 0</i> <i>must be a value between 0 and 1 inclusively</i>
<b>finish:</b>	A fraction (between 0 and 1) representing where in the sample to finish playback. 1 represents the end of the sample, 0.5 half-way through etc. <i>Default: 1</i> <i>must be a value between 0 and 1 inclusively</i>
<b>res:</b>	Filter resonance as a value between 0 and 1. Only functional if a cutoff value is specified. Large amounts of resonance (a res: near 1) can create a whistling sound around the cutoff frequency. Smaller values produce less resonance. <i>Default: 0</i> <i>must be zero or greater, must be a value less than 1</i> <i>May be changed whilst playing</i>
<b>cutoff:</b>	MIDI note representing the highest frequencies allowed to be present in the sound. A low value like 30 makes the sound round and dull, a high value like 100 makes the sound buzzy and crispy. <i>Default: 0</i> <i>must be zero or greater, must be a value less than 131</i> <i>May be changed whilst playing</i>
<b>norm:</b>	Normalise the audio (make quieter parts of the sample louder and louder parts quieter) - this is similar to the normaliser FX. This may emphasise any clicks caused by clipping. <i>Default: 0</i>

must be one of the following values: [0, 1]

---

# Miscellaneous Sounds

amp:	1	pan:	0	attack:	0	decay:	0	sustain:	-1	release:	0
attack_level:	1	decay_level:	sustain_level	sustain_level:	1	env_curve:	2	rate:	1	start:	0
finish:	1	res:	0	cutoff:	0	norm:	0				

sample :misc\_burp

amp:	<p>The amplitude of the sound. Typically a value between 0 and 1. Higher amplitudes may be used, but won't make the sound louder, they will just reduce the quality of all the sounds currently being played (due to compression.)</p> <p><i>Default: 1</i> <i>must be zero or greater</i> <i>May be changed whilst playing</i></p>
pan:	<p>Position of sound in stereo. With headphones on, this means how much of the sound is in the left ear, and how much is in the right ear. With a value of -1, the sound is completely in the left ear, a value of 0 puts the sound equally in both ears and a value of 1 puts the sound in the right ear. Values in between -1 and 1 move the sound accordingly.</p> <p><i>Default: 0</i> <i>must be a value between -1 and 1 inclusively</i> <i>May be changed whilst playing</i></p>
attack:	<p>Duration of the attack phase of the envelope.</p> <p><i>Default: 0</i> <i>must be zero or greater</i></p>
decay:	<p>Duration of the decay phase of the envelope.</p> <p><i>Default: 0</i> <i>must be zero or greater</i></p>
sustain:	<p>Duration of the sustain phase of the envelope.</p> <p><i>Default: -1</i> <i>must either be a positive value or -1</i></p>
release:	<p>Duration of the release phase of the envelope.</p> <p><i>Default: 0</i> <i>must be zero or greater</i></p>
attack_level:	<p>Amplitude level reached after attack phase and immediately before decay phase</p> <p><i>Default: 1</i> <i>must be zero or greater</i></p>
decay_level:	<p>Amplitude level reached after decay phase and immediately before sustain phase. Defaults to sustain_level unless explicitly set</p> <p><i>Default: sustain_level</i> <i>must be zero or greater</i></p>
sustain_level:	<p>Amplitude level reached after decay phase and immediately before release phase.</p> <p><i>Default: 1</i> <i>must be zero or greater</i></p>
env_curve:	<p>Select the shape of the curve between levels in the envelope. 1=linear, 2=exponential, 3=sine, 4=welch, 6=squared, 7=cubed</p> <p><i>Default: 2</i> <i>must be one of the following values: [1, 2, 3, 4, 6, 7]</i></p>

<b>rate:</b>	<p>Rate with which to play back - default is 1. Playing the sample at rate 2 will play it back at double the normal speed. This will have the effect of doubling the frequencies in the sample and halving the playback time. Use rates lower than 1 to slow the sample down. Negative rates will play the sample in reverse.</p> <p><i>Default: 1</i> <i>must not be zero</i></p>
<b>start:</b>	<p>A fraction (between 0 and 1) representing where in the sample to start playback. 1 represents the end of the sample, 0.5 half-way through etc.</p> <p><i>Default: 0</i> <i>must be a value between 0 and 1 inclusively</i></p>
<b>finish:</b>	<p>A fraction (between 0 and 1) representing where in the sample to finish playback. 1 represents the end of the sample, 0.5 half-way through etc.</p> <p><i>Default: 1</i> <i>must be a value between 0 and 1 inclusively</i></p>
<b>res:</b>	<p>Filter resonance as a value between 0 and 1. Only functional if a cutoff value is specified. Large amounts of resonance (a res: near 1) can create a whistling sound around the cutoff frequency. Smaller values produce less resonance.</p> <p><i>Default: 0</i> <i>must be zero or greater, must be a value less than 1</i> <i>May be changed whilst playing</i></p>
<b>cutoff:</b>	<p>MIDI note representing the highest frequencies allowed to be present in the sound. A low value like 30 makes the sound round and dull, a high value like 100 makes the sound buzzy and crispy.</p> <p><i>Default: 0</i> <i>must be zero or greater, must be a value less than 131</i> <i>May be changed whilst playing</i></p>
<b>norm:</b>	<p>Normalise the audio (make quieter parts of the sample louder and louder parts quieter) - this is similar to the normaliser FX. This may emphasise any clicks caused by clipping.</p> <p><i>Default: 0</i> <i>must be one of the following values: [0, 1]</i></p>

---

# Percussive Sounds

amp:	1	pan:	0	attack:	0	decay:	0	sustain:	-1	release:	0
attack_level:	1	decay_level:	sustain_level	sustain_level:	1	env_curve:	2	rate:	1	start:	0
finish:	1	res:	0	cutoff:	0	norm:	0				

sample :perc\_bell  
 sample :perc\_snap  
 sample :perc\_snap2

amp:	The amplitude of the sound. Typically a value between 0 and 1. Higher amplitudes may be used, but won't make the sound louder, they will just reduce the quality of all the sounds currently being played (due to compression.)  <i>Default: 1</i> <i>must be zero or greater</i> <i>May be changed whilst playing</i>
pan:	Position of sound in stereo. With headphones on, this means how much of the sound is in the left ear, and how much is in the right ear. With a value of -1, the sound is completely in the left ear, a value of 0 puts the sound equally in both ears and a value of 1 puts the sound in the right ear. Values in between -1 and 1 move the sound accordingly.  <i>Default: 0</i> <i>must be a value between -1 and 1 inclusively</i> <i>May be changed whilst playing</i>
attack:	Duration of the attack phase of the envelope.  <i>Default: 0</i> <i>must be zero or greater</i>
decay:	Duration of the decay phase of the envelope.  <i>Default: 0</i> <i>must be zero or greater</i>
sustain:	Duration of the sustain phase of the envelope.  <i>Default: -1</i> <i>must either be a positive value or -1</i>
release:	Duration of the release phase of the envelope.  <i>Default: 0</i> <i>must be zero or greater</i>
attack_level:	Amplitude level reached after attack phase and immediately before decay phase  <i>Default: 1</i> <i>must be zero or greater</i>
decay_level:	Amplitude level reached after decay phase and immediately before sustain phase. Defaults to sustain_level unless explicitly set  <i>Default: sustain_level</i> <i>must be zero or greater</i>
sustain_level:	Amplitude level reached after decay phase and immediately before release phase.  <i>Default: 1</i> <i>must be zero or greater</i>
	Select the shape of the curve between levels in the envelope. 1=linear, 2=exponential, 3=sine, 4=welch, 6=squared, 7=cubed



<b>env_curve:</b>	<i>Default: 2</i> <i>must be one of the following values: [1, 2, 3, 4, 6, 7]</i>
<b>rate:</b>	Rate with which to play back - default is 1. Playing the sample at rate 2 will play it back at double the normal speed. This will have the effect of doubling the frequencies in the sample and halving the playback time. Use rates lower than 1 to slow the sample down. Negative rates will play the sample in reverse.  <i>Default: 1</i> <i>must not be zero</i>
<b>start:</b>	A fraction (between 0 and 1) representing where in the sample to start playback. 1 represents the end of the sample, 0.5 half-way through etc.  <i>Default: 0</i> <i>must be a value between 0 and 1 inclusively</i>
<b>finish:</b>	A fraction (between 0 and 1) representing where in the sample to finish playback. 1 represents the end of the sample, 0.5 half-way through etc.  <i>Default: 1</i> <i>must be a value between 0 and 1 inclusively</i>
<b>res:</b>	Filter resonance as a value between 0 and 1. Only functional if a cutoff value is specified. Large amounts of resonance (a res: near 1) can create a whistling sound around the cutoff frequency. Smaller values produce less resonance.  <i>Default: 0</i> <i>must be zero or greater, must be a value less than 1</i> <i>May be changed whilst playing</i>
<b>cutoff:</b>	MIDI note representing the highest frequencies allowed to be present in the sound. A low value like 30 makes the sound round and dull, a high value like 100 makes the sound buzzy and crispy.  <i>Default: 0</i> <i>must be zero or greater, must be a value less than 131</i> <i>May be changed whilst playing</i>
<b>norm:</b>	Normalise the audio (make quieter parts of the sample louder and louder parts quieter) - this is similar to the normaliser FX. This may emphasise any clicks caused by clipping.  <i>Default: 0</i> <i>must be one of the following values: [0, 1]</i>

---

# Snare Drums

amp:	1	pan:	0	attack:	0	decay:	0	sustain:	-1	release:	0
attack_level:	1	decay_level:	sustain_level	sustain_level:	1	env_curve:	2	rate:	1	start:	0
finish:	1	res:	0	cutoff:	0	norm:	0				

sample :sn\_dub  
 sample :sn\_dolf  
 sample :sn\_zome

amp:	The amplitude of the sound. Typically a value between 0 and 1. Higher amplitudes may be used, but won't make the sound louder, they will just reduce the quality of all the sounds currently being played (due to compression.)  <i>Default: 1</i> <i>must be zero or greater</i> <i>May be changed whilst playing</i>
pan:	Position of sound in stereo. With headphones on, this means how much of the sound is in the left ear, and how much is in the right ear. With a value of -1, the sound is completely in the left ear, a value of 0 puts the sound equally in both ears and a value of 1 puts the sound in the right ear. Values in between -1 and 1 move the sound accordingly.  <i>Default: 0</i> <i>must be a value between -1 and 1 inclusively</i> <i>May be changed whilst playing</i>
attack:	Duration of the attack phase of the envelope.  <i>Default: 0</i> <i>must be zero or greater</i>
decay:	Duration of the decay phase of the envelope.  <i>Default: 0</i> <i>must be zero or greater</i>
sustain:	Duration of the sustain phase of the envelope.  <i>Default: -1</i> <i>must either be a positive value or -1</i>
release:	Duration of the release phase of the envelope.  <i>Default: 0</i> <i>must be zero or greater</i>
attack_level:	Amplitude level reached after attack phase and immediately before decay phase  <i>Default: 1</i> <i>must be zero or greater</i>
decay_level:	Amplitude level reached after decay phase and immediately before sustain phase. Defaults to sustain_level unless explicitly set  <i>Default: sustain_level</i> <i>must be zero or greater</i>
sustain_level:	Amplitude level reached after decay phase and immediately before release phase.  <i>Default: 1</i> <i>must be zero or greater</i>
	Select the shape of the curve between levels in the envelope. 1=linear, 2=exponential, 3=sine, 4=welch, 6=squared, 7=cubed

<b>env_curve:</b>	<i>Default: 2</i> <i>must be one of the following values: [1, 2, 3, 4, 6, 7]</i>
<b>rate:</b>	Rate with which to play back - default is 1. Playing the sample at rate 2 will play it back at double the normal speed. This will have the effect of doubling the frequencies in the sample and halving the playback time. Use rates lower than 1 to slow the sample down. Negative rates will play the sample in reverse.  <i>Default: 1</i> <i>must not be zero</i>
<b>start:</b>	A fraction (between 0 and 1) representing where in the sample to start playback. 1 represents the end of the sample, 0.5 half-way through etc.  <i>Default: 0</i> <i>must be a value between 0 and 1 inclusively</i>
<b>finish:</b>	A fraction (between 0 and 1) representing where in the sample to finish playback. 1 represents the end of the sample, 0.5 half-way through etc.  <i>Default: 1</i> <i>must be a value between 0 and 1 inclusively</i>
<b>res:</b>	Filter resonance as a value between 0 and 1. Only functional if a cutoff value is specified. Large amounts of resonance (a res: near 1) can create a whistling sound around the cutoff frequency. Smaller values produce less resonance.  <i>Default: 0</i> <i>must be zero or greater, must be a value less than 1</i> <i>May be changed whilst playing</i>
<b>cutoff:</b>	MIDI note representing the highest frequencies allowed to be present in the sound. A low value like 30 makes the sound round and dull, a high value like 100 makes the sound buzzy and crispy.  <i>Default: 0</i> <i>must be zero or greater, must be a value less than 131</i> <i>May be changed whilst playing</i>
<b>norm:</b>	Normalise the audio (make quieter parts of the sample louder and louder parts quieter) - this is similar to the normaliser FX. This may emphasise any clicks caused by clipping.  <i>Default: 0</i> <i>must be one of the following values: [0, 1]</i>

---

# Sounds featuring guitars

amp:	1	pan:	0	attack:	0	decay:	0	sustain:	-1	release:	0
attack_level:	1	decay_level:	sustain_level	sustain_level:	1	env_curve:	2	rate:	1	start:	0
finish:	1	res:	0	cutoff:	0	norm:	0				

sample :guit\_harmonics

sample :guit\_e\_fifths

sample :guit\_e\_slide

sample :guit\_em9

amp:	<p>The amplitude of the sound. Typically a value between 0 and 1. Higher amplitudes may be used, but won't make the sound louder, they will just reduce the quality of all the sounds currently being played (due to compression.)</p> <p><i>Default: 1</i> <i>must be zero or greater</i> <i>May be changed whilst playing</i></p>
pan:	<p>Position of sound in stereo. With headphones on, this means how much of the sound is in the left ear, and how much is in the right ear. With a value of -1, the sound is completely in the left ear, a value of 0 puts the sound equally in both ears and a value of 1 puts the sound in the right ear. Values in between -1 and 1 move the sound accordingly.</p> <p><i>Default: 0</i> <i>must be a value between -1 and 1 inclusively</i> <i>May be changed whilst playing</i></p>
attack:	<p>Duration of the attack phase of the envelope.</p> <p><i>Default: 0</i> <i>must be zero or greater</i></p>
decay:	<p>Duration of the decay phase of the envelope.</p> <p><i>Default: 0</i> <i>must be zero or greater</i></p>
sustain:	<p>Duration of the sustain phase of the envelope.</p> <p><i>Default: -1</i> <i>must either be a positive value or -1</i></p>
release:	<p>Duration of the release phase of the envelope.</p> <p><i>Default: 0</i> <i>must be zero or greater</i></p>
attack_level:	<p>Amplitude level reached after attack phase and immediately before decay phase</p> <p><i>Default: 1</i> <i>must be zero or greater</i></p>
decay_level:	<p>Amplitude level reached after decay phase and immediately before sustain phase. Defaults to sustain_level unless explicitly set</p> <p><i>Default: sustain_level</i> <i>must be zero or greater</i></p>
sustain_level:	<p>Amplitude level reached after decay phase and immediately before release phase.</p> <p><i>Default: 1</i> <i>must be zero or greater</i></p>
	<p>Select the shape of the curve between levels in the envelope. 1=linear, 2=exponential, 3=sine,</p>

<b>env_curve:</b>	4=welch, 6=squared, 7=cubed <i>Default: 2</i> <i>must be one of the following values: [1, 2, 3, 4, 6, 7]</i>
<b>rate:</b>	Rate with which to play back - default is 1. Playing the sample at rate 2 will play it back at double the normal speed. This will have the effect of doubling the frequencies in the sample and halving the playback time. Use rates lower than 1 to slow the sample down. Negative rates will play the sample in reverse. <i>Default: 1</i> <i>must not be zero</i>
<b>start:</b>	A fraction (between 0 and 1) representing where in the sample to start playback. 1 represents the end of the sample, 0.5 half-way through etc. <i>Default: 0</i> <i>must be a value between 0 and 1 inclusively</i>
<b>finish:</b>	A fraction (between 0 and 1) representing where in the sample to finish playback. 1 represents the end of the sample, 0.5 half-way through etc. <i>Default: 1</i> <i>must be a value between 0 and 1 inclusively</i>
<b>res:</b>	Filter resonance as a value between 0 and 1. Only functional if a cutoff value is specified. Large amounts of resonance (a res: near 1) can create a whistling sound around the cutoff frequency. Smaller values produce less resonance. <i>Default: 0</i> <i>must be zero or greater, must be a value less than 1</i> <i>May be changed whilst playing</i>
<b>cutoff:</b>	MIDI note representing the highest frequencies allowed to be present in the sound. A low value like 30 makes the sound round and dull, a high value like 100 makes the sound buzzy and crispy. <i>Default: 0</i> <i>must be zero or greater, must be a value less than 131</i> <i>May be changed whilst playing</i>
<b>norm:</b>	Normalise the audio (make quieter parts of the sample louder and louder parts quieter) - this is similar to the normaliser FX. This may emphasise any clicks caused by clipping. <i>Default: 0</i> <i>must be one of the following values: [0, 1]</i>

---

# Sounds for Looping

amp:	1	pan:	0	attack:	0	decay:	0	sustain:	-1	release:	0
attack_level:	1	decay_level:	sustain_level	sustain_level:	1	env_curve:	2	rate:	1	start:	0
finish:	1	res:	0	cutoff:	0	norm:	0				

```
sample :loop_industrial
sample :loop_compus
sample :loop_amen
sample :loop_amen_full
sample :loop_garzul
sample :loop_mika
```

amp:	<p>The amplitude of the sound. Typically a value between 0 and 1. Higher amplitudes may be used, but won't make the sound louder, they will just reduce the quality of all the sounds currently being played (due to compression.)</p> <p><i>Default: 1</i> <i>must be zero or greater</i> <i>May be changed whilst playing</i></p>
pan:	<p>Position of sound in stereo. With headphones on, this means how much of the sound is in the left ear, and how much is in the right ear. With a value of -1, the sound is completely in the left ear, a value of 0 puts the sound equally in both ears and a value of 1 puts the sound in the right ear. Values in between -1 and 1 move the sound accordingly.</p> <p><i>Default: 0</i> <i>must be a value between -1 and 1 inclusively</i> <i>May be changed whilst playing</i></p>
attack:	<p>Duration of the attack phase of the envelope.</p> <p><i>Default: 0</i> <i>must be zero or greater</i></p>
decay:	<p>Duration of the decay phase of the envelope.</p> <p><i>Default: 0</i> <i>must be zero or greater</i></p>
sustain:	<p>Duration of the sustain phase of the envelope.</p> <p><i>Default: -1</i> <i>must either be a positive value or -1</i></p>
release:	<p>Duration of the release phase of the envelope.</p> <p><i>Default: 0</i> <i>must be zero or greater</i></p>
attack_level:	<p>Amplitude level reached after attack phase and immediately before decay phase</p> <p><i>Default: 1</i> <i>must be zero or greater</i></p>
decay_level:	<p>Amplitude level reached after decay phase and immediately before sustain phase. Defaults to sustain_level unless explicitly set</p> <p><i>Default: sustain_level</i> <i>must be zero or greater</i></p>
sustain_level:	<p>Amplitude level reached after decay phase and immediately before release phase.</p> <p><i>Default: 1</i> <i>must be zero or greater</i></p>

<b>env_curve:</b>	Select the shape of the curve between levels in the envelope. 1=linear, 2=exponential, 3=sine, 4=welch, 6=squared, 7=cubed <i>Default: 2</i> <i>must be one of the following values: [1, 2, 3, 4, 6, 7]</i>
<b>rate:</b>	Rate with which to play back - default is 1. Playing the sample at rate 2 will play it back at double the normal speed. This will have the effect of doubling the frequencies in the sample and halving the playback time. Use rates lower than 1 to slow the sample down. Negative rates will play the sample in reverse. <i>Default: 1</i> <i>must not be zero</i>
<b>start:</b>	A fraction (between 0 and 1) representing where in the sample to start playback. 1 represents the end of the sample, 0.5 half-way through etc. <i>Default: 0</i> <i>must be a value between 0 and 1 inclusively</i>
<b>finish:</b>	A fraction (between 0 and 1) representing where in the sample to finish playback. 1 represents the end of the sample, 0.5 half-way through etc. <i>Default: 1</i> <i>must be a value between 0 and 1 inclusively</i>
<b>res:</b>	Filter resonance as a value between 0 and 1. Only functional if a cutoff value is specified. Large amounts of resonance (a res: near 1) can create a whistling sound around the cutoff frequency. Smaller values produce less resonance. <i>Default: 0</i> <i>must be zero or greater, must be a value less than 1</i> <i>May be changed whilst playing</i>
<b>cutoff:</b>	MIDI note representing the highest frequencies allowed to be present in the sound. A low value like 30 makes the sound round and dull, a high value like 100 makes the sound buzzy and crispy. <i>Default: 0</i> <i>must be zero or greater, must be a value less than 131</i> <i>May be changed whilst playing</i>
<b>norm:</b>	Normalise the audio (make quieter parts of the sample louder and louder parts quieter) - this is similar to the normaliser FX. This may emphasise any clicks caused by clipping. <i>Default: 0</i> <i>must be one of the following values: [0, 1]</i>

---

Band Eq  
Bitcrusher  
BPF  
Compressor  
Distortion  
Echo  
Flanger  
HPF  
Ixi Techno  
Krush  
Level  
LPF  
NBPF  
NHPF  
NLPF  
Normaliser  
NRBPF  
NRHPF  
NRLPF  
Pan  
Panslicer  
Pitch Shift  
RBPF  
Reverb  
RHPF  
Ring Mod  
RLPF  
Slicer  
Wobble



# Band EQ Filter

```
amp: 1 mix: 1 pre_amp: 1 freq: 100 res: 0.6 db: 0.6
```

```
with_fx :band_eq do  
  play 50  
end
```

Attenuate or Boost a frequency band

Introduced in v2.8

## Parameters

amp:	<p>The amplitude of the sound. Typically a value between 0 and 1. Higher amplitudes may be used, but won't make the sound louder, they will just reduce the quality of all the sounds currently being played (due to compression.)</p> <p><i>Default: 1</i> <i>Must be zero or greater</i> <i>May be changed whilst playing</i> <a href="#">Has slide parameters to shape changes</a></p>
mix:	<p>The amount (percentage) of FX present in the resulting sound represented as a value between 0 and 1. For example, a mix of 0 means that only the original sound is heard, a mix of 1 means that only the FX is heard (typically the default) and a mix of 0.5 means that half the original and half of the FX is heard.</p> <p><i>Default: 1</i> <i>Must be a value between 0 and 1 inclusively</i> <i>May be changed whilst playing</i> <a href="#">Has slide parameters to shape changes</a></p>
pre_amp:	<p>Amplification applied to the input signal immediately before it is passed to the FX.</p> <p><i>Default: 1</i> <i>Must be zero or greater</i> <i>May be changed whilst playing</i> <a href="#">Has slide parameters to shape changes</a> <i>Scaled with current BPM value</i></p>
freq:	<p>Centre frequency of the band in MIDI.</p> <p><i>Default: 100</i> <i>Must be greater than zero</i> <i>May be changed whilst playing</i> <a href="#">Has slide parameters to shape changes</a></p>
res:	<p>Width of the band as a value between 0 and 1</p> <p><i>Default: 0.6</i> <i>Must be zero or greater, must be a value less than 1</i> <i>May be changed whilst playing</i> <a href="#">Has slide parameters to shape changes</a></p>
db:	<p>Amount of boost or attenuation of the frequency band. A positive value boosts frequencies in the band, a negative value attenuates them.</p> <p><i>Default: 0.6</i> <i>May be changed whilst playing</i> <a href="#">Has slide parameters to shape changes</a></p>

## Slide Parameters

Any parameter that is slidable has three additional parameters named `_slide`, `_slide_curve`, and

`_slide_shape`. For example, 'amp' is slidable, so you can also set `amp_slide`, `amp_slide_curve`, and `amp_slide_shape` with the following effects:

<code>_slide:</code>	<p>Amount of time (in beats) for the parameter value to change. A long <code>parameter_slide</code> value means that the parameter takes a long time to slide from the previous value to the new value. A <code>parameter_slide</code> of 0 means that the parameter instantly changes to the new value.</p> <p><i>Default: 0</i></p>
<code>_slide_shape:</code>	<p>Shape of curve. 0: step, 1: linear, 3: sine, 4: welch, 5: custom (use <code>*_slide_curve: opt</code> e.g. <code>amp_slide_curve:</code>), 6: squared, 7: cubed.</p> <p><i>Default: 5</i></p>
<code>_slide_curve:</code>	<p>Shape of the slide curve (only honoured if slide shape is 5). 0 means linear and positive and negative numbers curve the segment up and down respectively.</p> <p><i>Default: 0</i></p>

---

# Bitcrusher

```
amp: 1 mix: 1 pre_amp: 1 sample_rate: 10000 bits: 8 cutoff: 0
```

```
with_fx :bitcrusher do  
  play 50  
end
```

Creates lo-fi output by decimating and deconstructing the incoming audio by lowering both the sample rate and bit depth. The default sample rate for CD audio is 44100, so use values less than that for that crunchy chip-tune sound full of artefacts and bitty distortion. Similarly, the default bit depth for CD audio is 16, so use values less than that for lo-fi sound.

Introduced in v2.3

## Parameters

amp:	<p>The amplitude of the sound. Typically a value between 0 and 1. Higher amplitudes may be used, but won't make the sound louder, they will just reduce the quality of all the sounds currently being played (due to compression.)</p> <p><i>Default: 1</i> <i>Must be zero or greater</i> <i>May be changed whilst playing</i> <a href="#">Has slide parameters to shape changes</a></p>
mix:	<p>The amount (percentage) of FX present in the resulting sound represented as a value between 0 and 1. For example, a mix of 0 means that only the original sound is heard, a mix of 1 means that only the FX is heard (typically the default) and a mix of 0.5 means that half the original and half of the FX is heard.</p> <p><i>Default: 1</i> <i>Must be a value between 0 and 1 inclusively</i> <i>May be changed whilst playing</i> <a href="#">Has slide parameters to shape changes</a></p>
pre_amp:	<p>Amplification applied to the input signal immediately before it is passed to the FX.</p> <p><i>Default: 1</i> <i>Must be zero or greater</i> <i>May be changed whilst playing</i> <a href="#">Has slide parameters to shape changes</a> <i>Scaled with current BPM value</i></p>
sample_rate:	<p>The sample rate the audio will be resampled at.</p> <p><i>Default: 10000</i> <i>Must be greater than zero</i> <i>May be changed whilst playing</i> <a href="#">Has slide parameters to shape changes</a></p>
bits:	<p>The bit depth of the resampled audio.</p> <p><i>Default: 8</i> <i>Must be greater than zero</i> <i>May be changed whilst playing</i> <a href="#">Has slide parameters to shape changes</a></p>
cutoff:	<p>MIDI note representing the highest frequencies allowed to be present in the sound. A low value like 30 makes the sound round and dull, a high value like 100 makes the sound buzzy and crispy.</p> <p><i>Default: 0</i> <i>Must be zero or greater, must be a value less than 131</i> <i>May be changed whilst playing</i> <a href="#">Has slide parameters to shape changes</a></p>

## Slide Parameters

Any parameter that is slidable has three additional parameters named `_slide`, `_slide_curve`, and `_slide_shape`. For example, 'amp' is slidable, so you can also set `amp_slide`, `amp_slide_curve`, and `amp_slide_shape` with the following effects:

<code>_slide:</code>	Amount of time (in beats) for the parameter value to change. A long parameter <code>_slide</code> value means that the parameter takes a long time to slide from the previous value to the new value. A parameter <code>_slide</code> of 0 means that the parameter instantly changes to the new value. <i>Default: 0</i>
<code>_slide_shape:</code>	Shape of curve. 0: step, 1: linear, 3: sine, 4: welch, 5: custom (use <code>*_slide_curve:</code> opt e.g. <code>amp_slide_curve:</code> ), 6: squared, 7: cubed. <i>Default: 5</i>
<code>_slide_curve:</code>	Shape of the slide curve (only honoured if slide shape is 5). 0 means linear and positive and negative numbers curve the segment up and down respectively. <i>Default: 0</i>

---

# Band Pass Filter

```
amp: 1 mix: 1 pre_amp: 1 centre: 100 res: 0.6
```

```
with_fx :bpf do  
  play 50  
end
```

Combines low pass and high pass filters to only allow a 'band' of frequencies through. If the band is very narrow (a low res value like 0.0001) then the BPF will reduce the original sound, almost down to a single frequency (controlled by the centre opt).

With higher values for res we can simulate other filters e.g. telephone lines, by cutting off low and high frequencies.

Use FX `:band_eq` with a negative db for the opposite effect - to attenuate a given band of frequencies.

Introduced in v2.3

## Parameters

amp:	<p>The amplitude of the sound. Typically a value between 0 and 1. Higher amplitudes may be used, but won't make the sound louder, they will just reduce the quality of all the sounds currently being played (due to compression.)</p> <p><i>Default: 1</i> <i>Must be zero or greater</i> <i>May be changed whilst playing</i> <a href="#">Has slide parameters to shape changes</a></p>
mix:	<p>The amount (percentage) of FX present in the resulting sound represented as a value between 0 and 1. For example, a mix of 0 means that only the original sound is heard, a mix of 1 means that only the FX is heard (typically the default) and a mix of 0.5 means that half the original and half of the FX is heard.</p> <p><i>Default: 1</i> <i>Must be a value between 0 and 1 inclusively</i> <i>May be changed whilst playing</i> <a href="#">Has slide parameters to shape changes</a></p>
pre_amp:	<p>Amplification applied to the input signal immediately before it is passed to the FX.</p> <p><i>Default: 1</i> <i>Must be zero or greater</i> <i>May be changed whilst playing</i> <a href="#">Has slide parameters to shape changes</a> <i>Scaled with current BPM value</i></p>
centre:	<p>Centre frequency for the filter as a MIDI note.</p> <p><i>Default: 100</i> <i>Must be a value greater than or equal to 0</i> <i>May be changed whilst playing</i> <a href="#">Has slide parameters to shape changes</a></p>
res:	<p>Filter resonance as a value between 0 and 1. Large amounts of resonance (a res: near 1) can create a whistling sound around the cutoff frequency. Smaller values produce less resonance.</p> <p><i>Default: 0.6</i> <i>Must be zero or greater, must be a value less than 1</i> <i>May be changed whilst playing</i> <a href="#">Has slide parameters to shape changes</a></p>

## Slide Parameters

Any parameter that is slidable has three additional parameters named `_slide`, `_slide_curve`, and `_slide_shape`. For example, 'amp' is slidable, so you can also set `amp_slide`, `amp_slide_curve`, and `amp_slide_shape` with the following effects:

<code>_slide:</code>	<p>Amount of time (in beats) for the parameter value to change. A long parameter <code>_slide</code> value means that the parameter takes a long time to slide from the previous value to the new value. A parameter <code>_slide</code> of 0 means that the parameter instantly changes to the new value.</p> <p><i>Default: 0</i></p>
<code>_slide_shape:</code>	<p>Shape of curve. 0: step, 1: linear, 3: sine, 4: welch, 5: custom (use <code>*_slide_curve: opt</code> e.g. <code>amp_slide_curve:</code>), 6: squared, 7: cubed.</p> <p><i>Default: 5</i></p>
<code>_slide_curve:</code>	<p>Shape of the slide curve (only honoured if slide shape is 5). 0 means linear and positive and negative numbers curve the segment up and down respectively.</p> <p><i>Default: 0</i></p>

---

# Compressor

```
amp: 1 mix: 1 pre_amp: 1 threshold: 0.2 clamp_time: 0.01 slope_above: 0.5  
slope_below: 1 relax_time: 0.01
```

```
with_fx :compressor do  
  play 50  
end
```

Compresses the dynamic range of the incoming signal. Equivalent to automatically turning the amp down when the signal gets too loud and then back up again when it's quiet. Useful for ensuring the containing signal doesn't overwhelm other aspects of the sound. Also a general purpose hard-knee dynamic range processor which can be tuned via the opts to both expand and compress the signal.

Introduced in v2.0

## Parameters

<b>amp:</b>	<p>The amplitude of the sound. Typically a value between 0 and 1. Higher amplitudes may be used, but won't make the sound louder, they will just reduce the quality of all the sounds currently being played (due to compression.)</p> <p><i>Default: 1</i> <i>Must be zero or greater</i> <i>May be changed whilst playing</i> <a href="#">Has slide parameters to shape changes</a></p>
<b>mix:</b>	<p>The amount (percentage) of FX present in the resulting sound represented as a value between 0 and 1. For example, a mix of 0 means that only the original sound is heard, a mix of 1 means that only the FX is heard (typically the default) and a mix of 0.5 means that half the original and half of the FX is heard.</p> <p><i>Default: 1</i> <i>Must be a value between 0 and 1 inclusively</i> <i>May be changed whilst playing</i></p>
<b>pre_amp:</b>	<p>Amplification applied to the input signal immediately before it is passed to the FX.</p> <p><i>Default: 1</i> <i>Must be zero or greater</i> <i>May be changed whilst playing</i> <a href="#">Has slide parameters to shape changes</a> <i>Scaled with current BPM value</i></p>
<b>threshold:</b>	<p>Threshold value determining the break point between slope_below and slope_above.</p> <p><i>Default: 0.2</i> <i>Must be zero or greater</i> <i>May be changed whilst playing</i> <a href="#">Has slide parameters to shape changes</a></p>
<b>clamp_time:</b>	<p>Time taken for the amplitude adjustments to kick in fully (in seconds). This is usually pretty small (not much more than 10 milliseconds). Also known as the time of the attack phase</p> <p><i>Default: 0.01</i> <i>Must be zero or greater</i> <i>May be changed whilst playing</i> <a href="#">Has slide parameters to shape changes</a></p>
<b>slope_above:</b>	<p>Slope of the amplitude curve above the threshold. A value of 1 means that the output of signals with amplitude above the threshold will be unaffected. Greater values will magnify and smaller values will attenuate the signal.</p> <p><i>Default: 0.5</i> <i>May be changed whilst playing</i></p>

	<i>Has slide parameters to shape changes</i>
<b>slope_below:</b>	<p>Slope of the amplitude curve below the threshold. A value of 1 means that the output of signals with amplitude below the threshold will be unaffected. Greater values will magnify and smaller values will attenuate the signal.</p> <p><i>Default: 1</i>  <i>May be changed whilst playing</i>  <i>Has slide parameters to shape changes</i></p>
<b>relax_time:</b>	<p>Time taken for the amplitude adjustments to be released. Usually a little longer than clamp_time. If both times are too short, you can get some (possibly unwanted) artefacts. Also known as the time of the release phase.</p> <p><i>Default: 0.01</i>  <i>Must be zero or greater</i>  <i>May be changed whilst playing</i>  <i>Has slide parameters to shape changes</i></p>

## Slide Parameters

Any parameter that is slidable has three additional parameters named `_slide`, `_slide_curve`, and `_slide_shape`. For example, 'amp' is slidable, so you can also set `amp_slide`, `amp_slide_curve`, and `amp_slide_shape` with the following effects:

<b>_slide:</b>	<p>Amount of time (in beats) for the parameter value to change. A long parameter_slide value means that the parameter takes a long time to slide from the previous value to the new value. A parameter_slide of 0 means that the parameter instantly changes to the new value.</p> <p><i>Default: 0</i></p>
<b>_slide_shape:</b>	<p>Shape of curve. 0: step, 1: linear, 3: sine, 4: welch, 5: custom (use *_slide_curve: opt e.g. amp_slide_curve:), 6: squared, 7: cubed.</p> <p><i>Default: 5</i></p>
<b>_slide_curve:</b>	<p>Shape of the slide curve (only honoured if slide shape is 5). 0 means linear and positive and negative numbers curve the segment up and down respectively.</p> <p><i>Default: 0</i></p>

---



# Distortion

```
amp: 1 mix: 1 pre_amp: 1 distort: 0.5
```

```
with_fx :distortion do  
  play 50  
end
```

Distorts the signal reducing clarity in favour of raw crunchy noise.

Introduced in v2.0

## Parameters

<b>amp:</b>	<p>The amplitude of the sound. Typically a value between 0 and 1. Higher amplitudes may be used, but won't make the sound louder, they will just reduce the quality of all the sounds currently being played (due to compression.)</p> <p><i>Default: 1</i> <i>Must be zero or greater</i> <i>May be changed whilst playing</i> <a href="#">Has slide parameters to shape changes</a></p>
<b>mix:</b>	<p>The amount (percentage) of FX present in the resulting sound represented as a value between 0 and 1. For example, a mix of 0 means that only the original sound is heard, a mix of 1 means that only the FX is heard (typically the default) and a mix of 0.5 means that half the original and half of the FX is heard.</p> <p><i>Default: 1</i> <i>Must be a value between 0 and 1 inclusively</i> <i>May be changed whilst playing</i> <a href="#">Has slide parameters to shape changes</a></p>
<b>pre_amp:</b>	<p>Amplification applied to the input signal immediately before it is passed to the FX.</p> <p><i>Default: 1</i> <i>Must be zero or greater</i> <i>May be changed whilst playing</i> <a href="#">Has slide parameters to shape changes</a> <i>Scaled with current BPM value</i></p>
<b>distort:</b>	<p>Amount of distortion to be applied (as a value between 0 and 1)</p> <p><i>Default: 0.5</i> <i>Must be a value greater than or equal to 0, must be a value less than 1</i> <i>May be changed whilst playing</i> <a href="#">Has slide parameters to shape changes</a></p>

## Slide Parameters

Any parameter that is slidable has three additional parameters named `_slide`, `_slide_curve`, and `_slide_shape`. For example, 'amp' is slidable, so you can also set `amp_slide`, `amp_slide_curve`, and `amp_slide_shape` with the following effects:

<b>_slide:</b>	<p>Amount of time (in beats) for the parameter value to change. A long parameter <code>_slide</code> value means that the parameter takes a long time to slide from the previous value to the new value. A parameter <code>_slide</code> of 0 means that the parameter instantly changes to the new value.</p> <p><i>Default: 0</i></p>
<b>_slide_shape:</b>	<p>Shape of curve. 0: step, 1: linear, 3: sine, 4: welch, 5: custom (use <code>*_slide_curve: opt e.g. amp_slide_curve:</code>), 6: squared, 7: cubed.</p> <p><i>Default: 5</i></p>

`_slide_curve:`

Shape of the slide curve (only honoured if slide shape is 5). 0 means linear and positive and negative numbers curve the segment up and down respectively.

*Default: 0*

---

# Echo

```
amp: 1 mix: 1 pre_amp: 1 phase: 0.25 decay: 2 max_phase: 2
```

```
with_fx :echo do  
  play 50  
end
```

Standard echo with variable phase duration (time between echoes) and decay (length of echo fade out). If you wish to have a phase duration longer than 2s, you need to specify the longest phase duration you'd like with the arg `max_phase`. Be warned, echo FX with very long phases can consume a lot of memory and take longer to initialise.

Introduced in v2.0

## Parameters

<code>amp:</code>	<p>The amplitude of the sound. Typically a value between 0 and 1. Higher amplitudes may be used, but won't make the sound louder, they will just reduce the quality of all the sounds currently being played (due to compression.)</p> <p><i>Default: 1</i> <i>Must be zero or greater</i> <i>May be changed whilst playing</i> <a href="#">Has slide parameters to shape changes</a></p>
<code>mix:</code>	<p>The amount (percentage) of FX present in the resulting sound represented as a value between 0 and 1. For example, a mix of 0 means that only the original sound is heard, a mix of 1 means that only the FX is heard (typically the default) and a mix of 0.5 means that half the original and half of the FX is heard.</p> <p><i>Default: 1</i> <i>Must be a value between 0 and 1 inclusively</i> <i>May be changed whilst playing</i> <a href="#">Has slide parameters to shape changes</a></p>
<code>pre_amp:</code>	<p>Amplification applied to the input signal immediately before it is passed to the FX.</p> <p><i>Default: 1</i> <i>Must be zero or greater</i> <i>May be changed whilst playing</i> <a href="#">Has slide parameters to shape changes</a> <i>Scaled with current BPM value</i></p>
<code>phase:</code>	<p>The time between echoes in beats.</p> <p><i>Default: 0.25</i> <i>Must be greater than zero</i> <i>May be changed whilst playing</i> <a href="#">Has slide parameters to shape changes</a> <i>Scaled with current BPM value</i></p>
<code>decay:</code>	<p>The time it takes for the echoes to fade away in beats.</p> <p><i>Default: 2</i> <i>Must be greater than zero</i> <i>May be changed whilst playing</i> <a href="#">Has slide parameters to shape changes</a> <i>Scaled with current BPM value</i></p>
<code>max_phase:</code>	<p>The maximum phase duration in beats.</p> <p><i>Default: 2</i> <i>Must be greater than zero</i> <i>Can not be changed once set</i></p>

## Slide Parameters

Any parameter that is slidable has three additional parameters named `_slide`, `_slide_curve`, and `_slide_shape`. For example, 'amp' is slidable, so you can also set `amp_slide`, `amp_slide_curve`, and `amp_slide_shape` with the following effects:

<code>_slide:</code>	Amount of time (in beats) for the parameter value to change. A long parameter <code>_slide</code> value means that the parameter takes a long time to slide from the previous value to the new value. A parameter <code>_slide</code> of 0 means that the parameter instantly changes to the new value.  <i>Default: 0</i>
<code>_slide_shape:</code>	Shape of curve. 0: step, 1: linear, 3: sine, 4: welch, 5: custom (use <code>*_slide_curve:</code> opt e.g. <code>amp_slide_curve:</code> ), 6: squared, 7: cubed.  <i>Default: 5</i>
<code>_slide_curve:</code>	Shape of the slide curve (only honoured if slide shape is 5). 0 means linear and positive and negative numbers curve the segment up and down respectively.  <i>Default: 0</i>

---

# Flanger

amp:	1	mix:	1	pre_amp:	1	phase:	4	phase_offset:	0	wave:	4
invert_wave:	0	stereo_invert_wave:	0	delay:	5	max_delay:	20	depth:	5	decay:	2
feedback:	0	invert_flange:	0								

```
with_fx :flanger do  
  play 50  
end
```

Mix the incoming signal with a copy of itself which has a rate modulating faster and slower than the original. Creates a swirling/whooshing effect.

Introduced in v2.3

## Parameters

amp:	<p>The amplitude of the sound. Typically a value between 0 and 1. Higher amplitudes may be used, but won't make the sound louder, they will just reduce the quality of all the sounds currently being played (due to compression.)</p> <p><i>Default: 1</i> <i>Must be zero or greater</i> <i>May be changed whilst playing</i> <a href="#">Has slide parameters to shape changes</a></p>
mix:	<p>The amount (percentage) of FX present in the resulting sound represented as a value between 0 and 1. For example, a mix of 0 means that only the original sound is heard, a mix of 1 means that only the FX is heard (typically the default) and a mix of 0.5 means that half the original and half of the FX is heard.</p> <p><i>Default: 1</i> <i>Must be a value between 0 and 1 inclusively</i> <i>May be changed whilst playing</i> <a href="#">Has slide parameters to shape changes</a></p>
pre_amp:	<p>Amplification applied to the input signal immediately before it is passed to the FX.</p> <p><i>Default: 1</i> <i>Must be zero or greater</i> <i>May be changed whilst playing</i> <a href="#">Has slide parameters to shape changes</a> <i>Scaled with current BPM value</i></p>
phase:	<p>Phase duration in beats of flanger modulation.</p> <p><i>Default: 4</i> <i>Must be greater than zero</i> <i>May be changed whilst playing</i> <a href="#">Has slide parameters to shape changes</a> <i>Scaled with current BPM value</i></p>
phase_offset:	<p>Initial modulation phase offset (a value between 0 and 1).</p> <p><i>Default: 0</i> <i>Must be a value between 0 and 1 inclusively</i> <i>Can not be changed once set</i></p>
wave:	<p>Wave type - 0 saw, 1 pulse, 2 triangle, 3 sine, 4 cubic. Different waves will produce different flanging modulation effects.</p> <p><i>Default: 4</i> <i>Must be one of the following values: [0, 1, 2, 3, 4]</i> <i>May be changed whilst playing</i></p>
	<p>Invert flanger control waveform (i.e. flip it on the y axis). 0=uninverted wave, 1=inverted</p>

<b>invert_wave:</b>	<p>Invert flanger control waveform (i.e. flip it on the y axis). 0=uninverted wave, 1=inverted wave.</p> <p><i>Default: 0</i>  <i>Must be one of the following values: [0, 1]</i>  <i>May be changed whilst playing</i></p>
<b>stereo_invert_wave:</b>	<p>Make the flanger control waveform in the left ear an inversion of the control waveform in the right ear. 0=uninverted wave, 1=inverted wave. This happens after the standard wave inversion with param :invert_wave.</p> <p><i>Default: 0</i>  <i>Must be one of the following values: [0, 1]</i>  <i>May be changed whilst playing</i></p>
<b>delay:</b>	<p>Amount of delay time between original and flanged version of audio.</p> <p><i>Default: 5</i>  <i>May be changed whilst playing</i>  <a href="#">Has slide parameters to shape changes</a></p>
<b>max_delay:</b>	<p>Max delay time. Used to set internal buffer size.</p> <p><i>Default: 20</i>  <i>Must be zero or greater</i>  <i>Can not be changed once set</i></p>
<b>depth:</b>	<p>Flange depth - greater depths produce a more prominent effect.</p> <p><i>Default: 5</i>  <i>May be changed whilst playing</i>  <a href="#">Has slide parameters to shape changes</a></p>
<b>decay:</b>	<p>Flange decay time in ms</p> <p><i>Default: 2</i>  <i>Must be zero or greater</i>  <i>May be changed whilst playing</i>  <a href="#">Has slide parameters to shape changes</a></p>
<b>feedback:</b>	<p>Amount of feedback.</p> <p><i>Default: 0</i>  <i>Must be zero or greater</i>  <i>May be changed whilst playing</i>  <a href="#">Has slide parameters to shape changes</a></p>
<b>invert_flange:</b>	<p>Invert flanger signal. 0=no inversion, 1=inverted signal.</p> <p><i>Default: 0</i>  <i>Must be one of the following values: [0, 1]</i>  <i>May be changed whilst playing</i></p>

## Slide Parameters

Any parameter that is slidable has three additional parameters named `_slide`, `_slide_curve`, and `_slide_shape`. For example, 'amp' is slidable, so you can also set `amp_slide`, `amp_slide_curve`, and `amp_slide_shape` with the following effects:

<b>_slide:</b>	<p>Amount of time (in beats) for the parameter value to change. A long <code>parameter_slide</code> value means that the parameter takes a long time to slide from the previous value to the new value. A <code>parameter_slide</code> of 0 means that the parameter instantly changes to the new value.</p> <p><i>Default: 0</i></p>
<b>_slide_shape:</b>	<p>Shape of curve. 0: step, 1: linear, 3: sine, 4: welch, 5: custom (use <code>*_slide_curve: opt</code> e.g. <code>amp_slide_curve:</code>), 6: squared, 7: cubed.</p>

`_slide_curve:`

*Default: 5*

Shape of the slide curve (only honoured if slide shape is 5). 0 means linear and positive and negative numbers curve the segment up and down respectively.

*Default: 0*

---

# High Pass Filter

```
amp: 1 mix: 1 pre_amp: 1 cutoff: 100
```

```
with_fx :hpf do  
  play 50  
end
```

Dampens the parts of the signal that are lower than the cutoff point (typically the bass of the sound) and keeps the higher parts (typically the crunchy fizzy harmonic overtones). Choose a lower cutoff to keep more of the bass/mid and a higher cutoff to make the sound more light and crispy.

Introduced in v2.0

## Parameters

<b>amp:</b>	<p>The amplitude of the sound. Typically a value between 0 and 1. Higher amplitudes may be used, but won't make the sound louder, they will just reduce the quality of all the sounds currently being played (due to compression.)</p> <p><i>Default: 1</i> <i>Must be zero or greater</i> <i>May be changed whilst playing</i> <a href="#">Has slide parameters to shape changes</a></p>
<b>mix:</b>	<p>The amount (percentage) of FX present in the resulting sound represented as a value between 0 and 1. For example, a mix of 0 means that only the original sound is heard, a mix of 1 means that only the FX is heard (typically the default) and a mix of 0.5 means that half the original and half of the FX is heard.</p> <p><i>Default: 1</i> <i>Must be a value between 0 and 1 inclusively</i> <i>May be changed whilst playing</i> <a href="#">Has slide parameters to shape changes</a></p>
<b>pre_amp:</b>	<p>Amplification applied to the input signal immediately before it is passed to the FX.</p> <p><i>Default: 1</i> <i>Must be zero or greater</i> <i>May be changed whilst playing</i> <a href="#">Has slide parameters to shape changes</a> <i>Scaled with current BPM value</i></p>
<b>cutoff:</b>	<p>MIDI note representing the highest frequencies allowed to be present in the sound. A low value like 30 makes the sound round and dull, a high value like 100 makes the sound buzzy and crispy.</p> <p><i>Default: 100</i> <i>Must be zero or greater, must be a value less than 131</i> <i>May be changed whilst playing</i> <a href="#">Has slide parameters to shape changes</a></p>

## Slide Parameters

Any parameter that is slidable has three additional parameters named `_slide`, `_slide_curve`, and `_slide_shape`. For example, 'amp' is slidable, so you can also set `amp_slide`, `amp_slide_curve`, and `amp_slide_shape` with the following effects:

<b>_slide:</b>	<p>Amount of time (in beats) for the parameter value to change. A long parameter <code>_slide</code> value means that the parameter takes a long time to slide from the previous value to the new value. A parameter <code>_slide</code> of 0 means that the parameter instantly changes to the new value.</p> <p><i>Default: 0</i></p>
	<p>Shape of curve. 0: step, 1: linear, 3: sine, 4: welch, 5: custom (use <code>*_slide_curve</code>: opt e.g.</p>



<b>_slide_shape:</b>	amp_slide_curve:), 6: squared, 7: cubed. <i>Default: 5</i>
<b>_slide_curve:</b>	Shape of the slide curve (only honoured if slide shape is 5). 0 means linear and positive and negative numbers curve the segment up and down respectively. <i>Default: 0</i>

---

# Techno from IXI Lang

```
amp: 1 mix: 1 pre_amp: 1 phase: 4 phase_offset: 0 cutoff_min: 60  
cutoff_max: 120 res: 0.8
```

```
with_fx :ixi_techno do  
  play 50  
end
```

Moving resonant low pass filter between min and max cutoffs. Great for sweeping effects across long synths or samples.

Introduced in v2.0

## Parameters

<b>amp:</b>	<p>The amplitude of the sound. Typically a value between 0 and 1. Higher amplitudes may be used, but won't make the sound louder, they will just reduce the quality of all the sounds currently being played (due to compression.)</p> <p><i>Default: 1</i> <i>Must be zero or greater</i> <i>May be changed whilst playing</i> <a href="#">Has slide parameters to shape changes</a></p>
<b>mix:</b>	<p>The amount (percentage) of FX present in the resulting sound represented as a value between 0 and 1. For example, a mix of 0 means that only the original sound is heard, a mix of 1 means that only the FX is heard (typically the default) and a mix of 0.5 means that half the original and half of the FX is heard.</p> <p><i>Default: 1</i> <i>Must be a value between 0 and 1 inclusively</i> <i>May be changed whilst playing</i> <a href="#">Has slide parameters to shape changes</a></p>
<b>pre_amp:</b>	<p>Amplification applied to the input signal immediately before it is passed to the FX.</p> <p><i>Default: 1</i> <i>Must be zero or greater</i> <i>May be changed whilst playing</i> <a href="#">Has slide parameters to shape changes</a> <i>Scaled with current BPM value</i></p>
<b>phase:</b>	<p>The phase duration (in beats) for filter modulation cycles</p> <p><i>Default: 4</i> <i>Must be greater than zero</i> <i>May be changed whilst playing</i> <a href="#">Has slide parameters to shape changes</a> <i>Scaled with current BPM value</i></p>
<b>phase_offset:</b>	<p>Initial modulation phase offset (a value between 0 and 1).</p> <p><i>Default: 0</i> <i>Must be a value between 0 and 1 inclusively</i> <i>Can not be changed once set</i></p>
<b>cutoff_min:</b>	<p>Minimum (MIDI) note that filter will move to whilst wobbling. Choose a lower note for a higher range of movement. Full range of movement is the distance between cutoff_max and cutoff_min</p> <p><i>Default: 60</i> <i>Must be zero or greater, must be a value less than 130</i> <i>May be changed whilst playing</i> <a href="#">Has slide parameters to shape changes</a></p>

<b>cutoff_max:</b>	<p>Maximum (MIDI) note that filter will move to whilst wobbling. Choose a higher note for a higher range of movement. Full range of movement is the distance between cutoff_max and cutoff_min</p> <p><i>Default: 120</i>  <i>Must be zero or greater, must be a value less than 130</i>  <i>May be changed whilst playing</i>  <a href="#">Has slide parameters to shape changes</a></p>
<b>res:</b>	<p>Filter resonance as a value between 0 and 1. Large amounts of resonance (a res: near 1) can create a whistling sound around the cutoff frequency. Smaller values produce less resonance.</p> <p><i>Default: 0.8</i>  <i>Must be zero or greater, must be a value less than 1</i>  <i>May be changed whilst playing</i>  <a href="#">Has slide parameters to shape changes</a></p>

## Slide Parameters

Any parameter that is slidable has three additional parameters named `_slide`, `_slide_curve`, and `_slide_shape`. For example, 'amp' is slidable, so you can also set `amp_slide`, `amp_slide_curve`, and `amp_slide_shape` with the following effects:

<b>_slide:</b>	<p>Amount of time (in beats) for the parameter value to change. A long parameter_slide value means that the parameter takes a long time to slide from the previous value to the new value. A parameter_slide of 0 means that the parameter instantly changes to the new value.</p> <p><i>Default: 0</i></p>
<b>_slide_shape:</b>	<p>Shape of curve. 0: step, 1: linear, 3: sine, 4: welch, 5: custom (use *_slide_curve: opt e.g. amp_slide_curve:), 6: squared, 7: cubed.</p> <p><i>Default: 5</i></p>
<b>_slide_curve:</b>	<p>Shape of the slide curve (only honoured if slide shape is 5). 0 means linear and positive and negative numbers curve the segment up and down respectively.</p> <p><i>Default: 0</i></p>

---

# krush

```
amp: 1 mix: 1 pre_amp: 1 gain: 5 cutoff: 100 res: 0
```

```
with_fx :krush do  
  play 50  
end
```

Krush that sound!

Introduced in v2.6

## Parameters

amp:	<p>The amplitude of the sound. Typically a value between 0 and 1. Higher amplitudes may be used, but won't make the sound louder, they will just reduce the quality of all the sounds currently being played (due to compression.)</p> <p><i>Default: 1</i> <i>Must be zero or greater</i> <i>May be changed whilst playing</i> <a href="#">Has slide parameters to shape changes</a></p>
mix:	<p>The amount (percentage) of FX present in the resulting sound represented as a value between 0 and 1. For example, a mix of 0 means that only the original sound is heard, a mix of 1 means that only the FX is heard (typically the default) and a mix of 0.5 means that half the original and half of the FX is heard.</p> <p><i>Default: 1</i> <i>Must be a value between 0 and 1 inclusively</i> <i>May be changed whilst playing</i> <a href="#">Has slide parameters to shape changes</a></p>
pre_amp:	<p>Amplification applied to the input signal immediately before it is passed to the FX.</p> <p><i>Default: 1</i> <i>Must be zero or greater</i> <i>May be changed whilst playing</i> <a href="#">Has slide parameters to shape changes</a> <i>Scaled with current BPM value</i></p>
gain:	<p>Amount of crushing to serve</p> <p><i>Default: 5</i> <i>Must be greater than zero</i> <i>May be changed whilst playing</i> <a href="#">Has slide parameters to shape changes</a></p>
cutoff:	<p>MIDI note representing the highest frequencies allowed to be present in the sound. A low value like 30 makes the sound round and dull, a high value like 100 makes the sound buzzy and crispy.</p> <p><i>Default: 100</i> <i>Must be zero or greater, must be a value less than 131</i> <i>May be changed whilst playing</i> <a href="#">Has slide parameters to shape changes</a></p>
res:	<p>Filter resonance as a value between 0 and 1. Large amounts of resonance (a res: near 1) can create a whistling sound around the cutoff frequency. Smaller values produce less resonance.</p> <p><i>Default: 0</i> <i>Must be zero or greater, must be a value less than 1</i> <i>May be changed whilst playing</i> <a href="#">Has slide parameters to shape changes</a></p>

## Slide Parameters

Any parameter that is slidable has three additional parameters named `_slide`, `_slide_curve`, and `_slide_shape`. For example, 'amp' is slidable, so you can also set `amp_slide`, `amp_slide_curve`, and `amp_slide_shape` with the following effects:

<code>_slide:</code>	<p>Amount of time (in beats) for the parameter value to change. A long parameter <code>_slide</code> value means that the parameter takes a long time to slide from the previous value to the new value. A parameter <code>_slide</code> of 0 means that the parameter instantly changes to the new value.</p> <p><i>Default: 0</i></p>
<code>_slide_shape:</code>	<p>Shape of curve. 0: step, 1: linear, 3: sine, 4: welch, 5: custom (use <code>*_slide_curve: opt</code> e.g. <code>amp_slide_curve:</code>), 6: squared, 7: cubed.</p> <p><i>Default: 5</i></p>
<code>_slide_curve:</code>	<p>Shape of the slide curve (only honoured if slide shape is 5). 0 means linear and positive and negative numbers curve the segment up and down respectively.</p> <p><i>Default: 0</i></p>

---

# Level Amplifier

```
amp: 1
```

```
with_fx :level do  
  play 50  
end
```

Amplitude modifier. All FX have their own amp built in, so it may be the case that you don't specifically need an isolated amp FX. However, it is useful to be able to control the overall amplitude of a number of running synths. All sounds created in the FX block will have their amplitudes multiplied by the amp level of this FX. For example, use an amp of 0 to silence all internal synths.

Introduced in v2.0

## Parameters

	The amplitude of the sound. Typically a value between 0 and 1. Higher amplitudes may be used, but won't make the sound louder, they will just reduce the quality of all the sounds currently being played (due to compression.)
amp:	<i>Default: 1</i> <i>Must be zero or greater</i> <i>May be changed whilst playing</i> <i>Has slide parameters to shape changes</i>

## Slide Parameters

Any parameter that is slidable has three additional parameters named `_slide`, `_slide_curve`, and `_slide_shape`. For example, 'amp' is slidable, so you can also set `amp_slide`, `amp_slide_curve`, and `amp_slide_shape` with the following effects:

<code>_slide:</code>	Amount of time (in beats) for the parameter value to change. A long <code>parameter_slide</code> value means that the parameter takes a long time to slide from the previous value to the new value. A <code>parameter_slide</code> of 0 means that the parameter instantly changes to the new value. <i>Default: 0</i>
<code>_slide_shape:</code>	Shape of curve. 0: step, 1: linear, 3: sine, 4: welch, 5: custom (use <code>*_slide_curve:</code> opt e.g. <code>amp_slide_curve:</code> ), 6: squared, 7: cubed. <i>Default: 5</i>
<code>_slide_curve:</code>	Shape of the slide curve (only honoured if slide shape is 5). 0 means linear and positive and negative numbers curve the segment up and down respectively. <i>Default: 0</i>

---

# Low Pass Filter

```
amp: 1 mix: 1 pre_amp: 1 cutoff: 100
```

```
with_fx :lpf do  
  play 50  
end
```

Dampens the parts of the signal that are above than the cutoff point (typically the crunchy fizzy harmonic overtones) and keeps the lower parts (typically the bass/mid of the sound). Choose a higher cutoff to keep more of the high frequencies/treble of the sound and a lower cutoff to make the sound more dull and only keep the bass.

Introduced in v2.0

## Parameters

<b>amp:</b>	<p>The amplitude of the sound. Typically a value between 0 and 1. Higher amplitudes may be used, but won't make the sound louder, they will just reduce the quality of all the sounds currently being played (due to compression.)</p> <p><i>Default: 1</i> <i>Must be zero or greater</i> <i>May be changed whilst playing</i> <a href="#">Has slide parameters to shape changes</a></p>
<b>mix:</b>	<p>The amount (percentage) of FX present in the resulting sound represented as a value between 0 and 1. For example, a mix of 0 means that only the original sound is heard, a mix of 1 means that only the FX is heard (typically the default) and a mix of 0.5 means that half the original and half of the FX is heard.</p> <p><i>Default: 1</i> <i>Must be a value between 0 and 1 inclusively</i> <i>May be changed whilst playing</i> <a href="#">Has slide parameters to shape changes</a></p>
<b>pre_amp:</b>	<p>Amplification applied to the input signal immediately before it is passed to the FX.</p> <p><i>Default: 1</i> <i>Must be zero or greater</i> <i>May be changed whilst playing</i> <a href="#">Has slide parameters to shape changes</a> <i>Scaled with current BPM value</i></p>
<b>cutoff:</b>	<p>MIDI note representing the highest frequencies allowed to be present in the sound. A low value like 30 makes the sound round and dull, a high value like 100 makes the sound buzzy and crispy.</p> <p><i>Default: 100</i> <i>Must be zero or greater, must be a value less than 131</i> <i>May be changed whilst playing</i> <a href="#">Has slide parameters to shape changes</a></p>

## Slide Parameters

Any parameter that is slidable has three additional parameters named `_slide`, `_slide_curve`, and `_slide_shape`. For example, 'amp' is slidable, so you can also set `amp_slide`, `amp_slide_curve`, and `amp_slide_shape` with the following effects:

<b>_slide:</b>	<p>Amount of time (in beats) for the parameter value to change. A long <code>parameter_slide</code> value means that the parameter takes a long time to slide from the previous value to the new value. A <code>parameter_slide</code> of 0 means that the parameter instantly changes to the new value.</p> <p><i>Default: 0</i></p>
----------------	---

**\_slide\_shape:**

Shape of curve. 0: step, 1: linear, 3: sine, 4: welch, 5: custom (use  $\ast\_slide\_curve$ : opt e.g.  $amp\_slide\_curve$ ), 6: squared, 7: cubed.

*Default: 5*

**\_slide\_curve:**

Shape of the slide curve (only honoured if slide shape is 5). 0 means linear and positive and negative numbers curve the segment up and down respectively.

*Default: 0*

---



# Normalised Band Pass Filter

```
amp: 1 mix: 1 pre_amp: 1 centre: 100 res: 0.6
```

```
with_fx :nbpf do  
  play 50  
end
```

Like the Band Pass Filter but normalised. The normaliser is useful here as some volume is lost when filtering the original signal.

Introduced in v2.3

## Parameters

amp:	<p>The amplitude of the sound. Typically a value between 0 and 1. Higher amplitudes may be used, but won't make the sound louder, they will just reduce the quality of all the sounds currently being played (due to compression.)</p> <p><i>Default: 1</i> <i>Must be zero or greater</i> <i>May be changed whilst playing</i> <a href="#">Has slide parameters to shape changes</a></p>
mix:	<p>The amount (percentage) of FX present in the resulting sound represented as a value between 0 and 1. For example, a mix of 0 means that only the original sound is heard, a mix of 1 means that only the FX is heard (typically the default) and a mix of 0.5 means that half the original and half of the FX is heard.</p> <p><i>Default: 1</i> <i>Must be a value between 0 and 1 inclusively</i> <i>May be changed whilst playing</i> <a href="#">Has slide parameters to shape changes</a></p>
pre_amp:	<p>Amplification applied to the input signal immediately before it is passed to the FX.</p> <p><i>Default: 1</i> <i>Must be zero or greater</i> <i>May be changed whilst playing</i> <a href="#">Has slide parameters to shape changes</a> <i>Scaled with current BPM value</i></p>
centre:	<p>Centre frequency for the filter as a MIDI note.</p> <p><i>Default: 100</i> <i>Must be a value greater than or equal to 0</i> <i>May be changed whilst playing</i> <a href="#">Has slide parameters to shape changes</a></p>
res:	<p>Filter resonance as a value between 0 and 1. Large amounts of resonance (a res: near 1) can create a whistling sound around the cutoff frequency. Smaller values produce less resonance.</p> <p><i>Default: 0.6</i> <i>Must be zero or greater, must be a value less than 1</i> <i>May be changed whilst playing</i> <a href="#">Has slide parameters to shape changes</a></p>

## Slide Parameters

Any parameter that is slidable has three additional parameters named `_slide`, `_slide_curve`, and `_slide_shape`. For example, 'amp' is slidable, so you can also set `amp_slide`, `amp_slide_curve`, and `amp_slide_shape` with the following effects:

Amount of time (in beats) for the parameter value to change. A long parameter slide value

<code>_slide:</code>	means that the parameter takes a long time to slide from the previous value to the new value. A parameter <code>_slide</code> of 0 means that the parameter instantly changes to the new value. <i>Default: 0</i>
<code>_slide_shape:</code>	Shape of curve. 0: step, 1: linear, 3: sine, 4: welch, 5: custom (use <code>*_slide_curve:</code> opt e.g. <code>amp_slide_curve:</code> ), 6: squared, 7: cubed. <i>Default: 5</i>
<code>_slide_curve:</code>	Shape of the slide curve (only honoured if slide shape is 5). 0 means linear and positive and negative numbers curve the segment up and down respectively. <i>Default: 0</i>

---

# Normalised High Pass Filter

```
amp: 1 mix: 1 pre_amp: 1 cutoff: 100 res: 0.5
```

```
with_fx :nhpf do  
  play 50  
end
```

A high pass filter chained to a normaliser. Ensures that the signal is both filtered by a standard high pass filter and then normalised to ensure the amplitude of the final output is constant. A high pass filter will reduce the amplitude of the resulting signal (as some of the sound has been filtered out) the normaliser can compensate for this loss (although will also have the side effect of flattening all dynamics). See doc for hpf.

Introduced in v2.0

## Parameters

amp:	<p>The amplitude of the sound. Typically a value between 0 and 1. Higher amplitudes may be used, but won't make the sound louder, they will just reduce the quality of all the sounds currently being played (due to compression.)</p> <p><i>Default: 1</i> <i>Must be zero or greater</i> <i>May be changed whilst playing</i> <a href="#">Has slide parameters to shape changes</a></p>
mix:	<p>The amount (percentage) of FX present in the resulting sound represented as a value between 0 and 1. For example, a mix of 0 means that only the original sound is heard, a mix of 1 means that only the FX is heard (typically the default) and a mix of 0.5 means that half the original and half of the FX is heard.</p> <p><i>Default: 1</i> <i>Must be a value between 0 and 1 inclusively</i> <i>May be changed whilst playing</i> <a href="#">Has slide parameters to shape changes</a></p>
pre_amp:	<p>Amplification applied to the input signal immediately before it is passed to the FX.</p> <p><i>Default: 1</i> <i>Must be zero or greater</i> <i>May be changed whilst playing</i> <a href="#">Has slide parameters to shape changes</a> <i>Scaled with current BPM value</i></p>
cutoff:	<p>MIDI note representing the highest frequencies allowed to be present in the sound. A low value like 30 makes the sound round and dull, a high value like 100 makes the sound buzzy and crispy.</p> <p><i>Default: 100</i> <i>Must be zero or greater, must be a value less than 131</i> <i>May be changed whilst playing</i> <a href="#">Has slide parameters to shape changes</a></p>
res:	<p>Filter resonance as a value between 0 and 1. Large amounts of resonance (a res: near 1) can create a whistling sound around the cutoff frequency. Smaller values produce less resonance.</p> <p><i>Default: 0.5</i> <i>Must be zero or greater, must be a value less than 1</i> <i>May be changed whilst playing</i> <a href="#">Has slide parameters to shape changes</a></p>

## Slide Parameters

Any parameter that is slidable has three additional parameters named `_slide`, `_slide_curve`, and `_slide_shape`. For example, 'amp' is slidable, so you can also set `amp_slide`, `amp_slide_curve`, and

amp\_slide\_shape with the following effects:

<b>_slide:</b>	Amount of time (in beats) for the parameter value to change. A long parameter_slide value means that the parameter takes a long time to slide from the previous value to the new value. A parameter_slide of 0 means that the parameter instantly changes to the new value.  <i>Default: 0</i>
<b>_slide_shape:</b>	Shape of curve. 0: step, 1: linear, 3: sine, 4: welch, 5: custom (use *_slide_curve: opt e.g. amp_slide_curve:), 6: squared, 7: cubed.  <i>Default: 5</i>
<b>_slide_curve:</b>	Shape of the slide curve (only honoured if slide shape is 5). 0 means linear and positive and negative numbers curve the segment up and down respectively.  <i>Default: 0</i>

---

# Normalised Low Pass Filter.

```
amp: 1 mix: 1 pre_amp: 1 cutoff: 100
```

```
with_fx :nlpf do  
  play 50  
end
```

A low pass filter chained to a normaliser. Ensures that the signal is both filtered by a standard low pass filter and then normalised to ensure the amplitude of the final output is constant. A low pass filter will reduce the amplitude of the resulting signal (as some of the sound has been filtered out) the normaliser can compensate for this loss (although will also have the side effect of flattening all dynamics). See doc for lpf.

Introduced in v2.0

## Parameters

<b>amp:</b>	<p>The amplitude of the sound. Typically a value between 0 and 1. Higher amplitudes may be used, but won't make the sound louder, they will just reduce the quality of all the sounds currently being played (due to compression.)</p> <p><i>Default: 1</i> <i>Must be zero or greater</i> <i>May be changed whilst playing</i> <a href="#">Has slide parameters to shape changes</a></p>
<b>mix:</b>	<p>The amount (percentage) of FX present in the resulting sound represented as a value between 0 and 1. For example, a mix of 0 means that only the original sound is heard, a mix of 1 means that only the FX is heard (typically the default) and a mix of 0.5 means that half the original and half of the FX is heard.</p> <p><i>Default: 1</i> <i>Must be a value between 0 and 1 inclusively</i> <i>May be changed whilst playing</i> <a href="#">Has slide parameters to shape changes</a></p>
<b>pre_amp:</b>	<p>Amplification applied to the input signal immediately before it is passed to the FX.</p> <p><i>Default: 1</i> <i>Must be zero or greater</i> <i>May be changed whilst playing</i> <a href="#">Has slide parameters to shape changes</a> <i>Scaled with current BPM value</i></p>
<b>cutoff:</b>	<p>MIDI note representing the highest frequencies allowed to be present in the sound. A low value like 30 makes the sound round and dull, a high value like 100 makes the sound buzzy and crispy.</p> <p><i>Default: 100</i> <i>Must be zero or greater, must be a value less than 131</i> <i>May be changed whilst playing</i> <a href="#">Has slide parameters to shape changes</a></p>

## Slide Parameters

Any parameter that is slidable has three additional parameters named `_slide`, `_slide_curve`, and `_slide_shape`. For example, 'amp' is slidable, so you can also set `amp_slide`, `amp_slide_curve`, and `amp_slide_shape` with the following effects:

<b>_slide:</b>	<p>Amount of time (in beats) for the parameter value to change. A long parameter_slide value means that the parameter takes a long time to slide from the previous value to the new value. A parameter_slide of 0 means that the parameter instantly changes to the new value.</p> <p><i>Default: 0</i></p>
----------------	---

**\_slide\_shape:**

Shape of curve. 0: step, 1: linear, 3: sine, 4: welch, 5: custom (use  $\ast\_slide\_curve$ : opt e.g.  $amp\_slide\_curve$ ), 6: squared, 7: cubed.

*Default: 5*

**\_slide\_curve:**

Shape of the slide curve (only honoured if slide shape is 5). 0 means linear and positive and negative numbers curve the segment up and down respectively.

*Default: 0*

---

# Normaliser

```
amp: 1 mix: 1 pre_amp: 1 level: 1
```

```
with_fx :normaliser do  
  play 50  
end
```

Raise or lower amplitude of sound to a specified level. Evens out the amplitude of incoming sound across the frequency spectrum by flattening all dynamics.

Introduced in v2.0

## Parameters

<b>amp:</b>	<p>The amplitude of the sound. Typically a value between 0 and 1. Higher amplitudes may be used, but won't make the sound louder, they will just reduce the quality of all the sounds currently being played (due to compression.)</p> <p><i>Default: 1</i> <i>Must be zero or greater</i> <i>May be changed whilst playing</i> <a href="#">Has slide parameters to shape changes</a></p>
<b>mix:</b>	<p>The amount (percentage) of FX present in the resulting sound represented as a value between 0 and 1. For example, a mix of 0 means that only the original sound is heard, a mix of 1 means that only the FX is heard (typically the default) and a mix of 0.5 means that half the original and half of the FX is heard.</p> <p><i>Default: 1</i> <i>Must be a value between 0 and 1 inclusively</i> <i>May be changed whilst playing</i> <a href="#">Has slide parameters to shape changes</a></p>
<b>pre_amp:</b>	<p>Amplification applied to the input signal immediately before it is passed to the FX.</p> <p><i>Default: 1</i> <i>Must be zero or greater</i> <i>May be changed whilst playing</i> <a href="#">Has slide parameters to shape changes</a> <i>Scaled with current BPM value</i></p>
<b>level:</b>	<p>The peak output amplitude level at which to normalise the input.</p> <p><i>Default: 1</i> <i>Must be a value greater than or equal to 0</i> <i>May be changed whilst playing</i> <a href="#">Has slide parameters to shape changes</a></p>

## Slide Parameters

Any parameter that is slidable has three additional parameters named `_slide`, `_slide_curve`, and `_slide_shape`. For example, 'amp' is slidable, so you can also set `amp_slide`, `amp_slide_curve`, and `amp_slide_shape` with the following effects:

<b>_slide:</b>	<p>Amount of time (in beats) for the parameter value to change. A long parameter <code>_slide</code> value means that the parameter takes a long time to slide from the previous value to the new value. A parameter <code>_slide</code> of 0 means that the parameter instantly changes to the new value.</p> <p><i>Default: 0</i></p>
<b>_slide_shape:</b>	<p>Shape of curve. 0: step, 1: linear, 3: sine, 4: welch, 5: custom (use <code>*_slide_curve: opt e.g. amp_slide_curve:</code>), 6: squared, 7: cubed.</p>

`_slide_curve:`

*Default: 5*

Shape of the slide curve (only honoured if slide shape is 5). 0 means linear and positive and negative numbers curve the segment up and down respectively.

*Default: 0*

---



# Normalised Resonant Band Pass Filter

```
amp: 1 mix: 1 pre_amp: 1 centre: 100 res: 0.5
```

```
with_fx :nrbpf do  
  play 50  
end
```

Like the Band Pass Filter but normalised, with a resonance (slight volume boost) around the target frequency. This can produce an interesting whistling effect, especially when used with smaller values for the res opt.

The normaliser is useful here as some volume is lost when filtering the original signal.

Introduced in v2.3

## Parameters

amp:	<p>The amplitude of the sound. Typically a value between 0 and 1. Higher amplitudes may be used, but won't make the sound louder, they will just reduce the quality of all the sounds currently being played (due to compression.)</p> <p><i>Default: 1</i> <i>Must be zero or greater</i> <i>May be changed whilst playing</i> <a href="#">Has slide parameters to shape changes</a></p>
mix:	<p>The amount (percentage) of FX present in the resulting sound represented as a value between 0 and 1. For example, a mix of 0 means that only the original sound is heard, a mix of 1 means that only the FX is heard (typically the default) and a mix of 0.5 means that half the original and half of the FX is heard.</p> <p><i>Default: 1</i> <i>Must be a value between 0 and 1 inclusively</i> <i>May be changed whilst playing</i> <a href="#">Has slide parameters to shape changes</a></p>
pre_amp:	<p>Amplification applied to the input signal immediately before it is passed to the FX.</p> <p><i>Default: 1</i> <i>Must be zero or greater</i> <i>May be changed whilst playing</i> <a href="#">Has slide parameters to shape changes</a> <i>Scaled with current BPM value</i></p>
centre:	<p>Centre frequency for the filter as a MIDI note.</p> <p><i>Default: 100</i> <i>Must be a value greater than or equal to 0</i> <i>May be changed whilst playing</i> <a href="#">Has slide parameters to shape changes</a></p>
res:	<p>Filter resonance as a value between 0 and 1. Large amounts of resonance (a res: near 1) can create a whistling sound around the cutoff frequency. Smaller values produce less resonance.</p> <p><i>Default: 0.5</i> <i>Must be zero or greater, must be a value less than 1</i> <i>May be changed whilst playing</i> <a href="#">Has slide parameters to shape changes</a></p>

## Slide Parameters

Any parameter that is slidable has three additional parameters named `_slide`, `_slide_curve`, and `_slide_shape`. For example, 'amp' is slidable, so you can also set `amp_slide`, `amp_slide_curve`, and

amp\_slide\_shape with the following effects:

<b>_slide:</b>	Amount of time (in beats) for the parameter value to change. A long parameter_slide value means that the parameter takes a long time to slide from the previous value to the new value. A parameter_slide of 0 means that the parameter instantly changes to the new value.  <i>Default: 0</i>
<b>_slide_shape:</b>	Shape of curve. 0: step, 1: linear, 3: sine, 4: welch, 5: custom (use *_slide_curve: opt e.g. amp_slide_curve:), 6: squared, 7: cubed.  <i>Default: 5</i>
<b>_slide_curve:</b>	Shape of the slide curve (only honoured if slide shape is 5). 0 means linear and positive and negative numbers curve the segment up and down respectively.  <i>Default: 0</i>

---

# Normalised Resonant High Pass Filter

```
amp: 1 mix: 1 pre_amp: 1 cutoff: 100 res: 0.5
```

```
with_fx :nrhpf do  
  play 50  
end
```

Dampens the parts of the signal that are above than the cutoff point (typically the crunchy fizzy harmonic overtones) and keeps the lower parts (typically the bass/mid of the sound). behaviour, The resonant part of the resonant low pass filter emphasises/resonates the frequencies around the cutoff point. The amount of emphasis is controlled by the res param with a lower res resulting in greater resonance. High amounts of resonance (rq ~0) can create a whistling sound around the cutoff frequency.

Choose a higher cutoff to keep more of the high frequencies/treble of the sound and a lower cutoff to make the sound more dull and only keep the bass.

Introduced in v2.0

## Parameters

amp:	<p>The amplitude of the sound. Typically a value between 0 and 1. Higher amplitudes may be used, but won't make the sound louder, they will just reduce the quality of all the sounds currently being played (due to compression.)</p> <p><i>Default: 1</i> <i>Must be zero or greater</i> <i>May be changed whilst playing</i> <a href="#">Has slide parameters to shape changes</a></p>
mix:	<p>The amount (percentage) of FX present in the resulting sound represented as a value between 0 and 1. For example, a mix of 0 means that only the original sound is heard, a mix of 1 means that only the FX is heard (typically the default) and a mix of 0.5 means that half the original and half of the FX is heard.</p> <p><i>Default: 1</i> <i>Must be a value between 0 and 1 inclusively</i> <i>May be changed whilst playing</i> <a href="#">Has slide parameters to shape changes</a></p>
pre_amp:	<p>Amplification applied to the input signal immediately before it is passed to the FX.</p> <p><i>Default: 1</i> <i>Must be zero or greater</i> <i>May be changed whilst playing</i> <a href="#">Has slide parameters to shape changes</a> <i>Scaled with current BPM value</i></p>
cutoff:	<p>MIDI note representing the highest frequencies allowed to be present in the sound. A low value like 30 makes the sound round and dull, a high value like 100 makes the sound buzzy and crispy.</p> <p><i>Default: 100</i> <i>Must be zero or greater, must be a value less than 131</i> <i>May be changed whilst playing</i> <a href="#">Has slide parameters to shape changes</a></p>
res:	<p>Filter resonance as a value between 0 and 1. Large amounts of resonance (a res: near 1) can create a whistling sound around the cutoff frequency. Smaller values produce less resonance.</p> <p><i>Default: 0.5</i> <i>Must be zero or greater, must be a value less than 1</i> <i>May be changed whilst playing</i> <a href="#">Has slide parameters to shape changes</a></p>

## Slide Parameters

Any parameter that is slidable has three additional parameters named `_slide`, `_slide_curve`, and `_slide_shape`. For example, 'amp' is slidable, so you can also set `amp_slide`, `amp_slide_curve`, and `amp_slide_shape` with the following effects:

<code>_slide:</code>	Amount of time (in beats) for the parameter value to change. A long parameter <code>_slide</code> value means that the parameter takes a long time to slide from the previous value to the new value. A parameter <code>_slide</code> of 0 means that the parameter instantly changes to the new value. <i>Default: 0</i>
<code>_slide_shape:</code>	Shape of curve. 0: step, 1: linear, 3: sine, 4: welch, 5: custom (use <code>*_slide_curve:</code> opt e.g. <code>amp_slide_curve:</code> ), 6: squared, 7: cubed. <i>Default: 5</i>
<code>_slide_curve:</code>	Shape of the slide curve (only honoured if slide shape is 5). 0 means linear and positive and negative numbers curve the segment up and down respectively. <i>Default: 0</i>

---

# Normalised Resonant Low Pass Filter

```
amp: 1 mix: 1 pre_amp: 1 cutoff: 100 res: 0.5
```

```
with_fx :nrlpf do  
  play 50  
end
```

Dampens the parts of the signal that are above than the cutoff point (typically the crunchy fizzy harmonic overtones) and keeps the lower parts (typically the bass/mid of the sound). behaviour, The resonant part of the resonant low pass filter emphasises/resonates the frequencies around the cutoff point. The amount of emphasis is controlled by the res param with a lower res resulting in greater resonance. High amounts of resonance (rq ~0) can create a whistling sound around the cutoff frequency.

Choose a higher cutoff to keep more of the high frequencies/treble of the sound and a lower cutoff to make the sound more dull and only keep the bass.

Introduced in v2.0

## Parameters

amp:	<p>The amplitude of the sound. Typically a value between 0 and 1. Higher amplitudes may be used, but won't make the sound louder, they will just reduce the quality of all the sounds currently being played (due to compression.)</p> <p><i>Default: 1</i> <i>Must be zero or greater</i> <i>May be changed whilst playing</i> <a href="#">Has slide parameters to shape changes</a></p>
mix:	<p>The amount (percentage) of FX present in the resulting sound represented as a value between 0 and 1. For example, a mix of 0 means that only the original sound is heard, a mix of 1 means that only the FX is heard (typically the default) and a mix of 0.5 means that half the original and half of the FX is heard.</p> <p><i>Default: 1</i> <i>Must be a value between 0 and 1 inclusively</i> <i>May be changed whilst playing</i> <a href="#">Has slide parameters to shape changes</a></p>
pre_amp:	<p>Amplification applied to the input signal immediately before it is passed to the FX.</p> <p><i>Default: 1</i> <i>Must be zero or greater</i> <i>May be changed whilst playing</i> <a href="#">Has slide parameters to shape changes</a> <i>Scaled with current BPM value</i></p>
cutoff:	<p>MIDI note representing the highest frequencies allowed to be present in the sound. A low value like 30 makes the sound round and dull, a high value like 100 makes the sound buzzy and crispy.</p> <p><i>Default: 100</i> <i>Must be zero or greater, must be a value less than 131</i> <i>May be changed whilst playing</i> <a href="#">Has slide parameters to shape changes</a></p>
res:	<p>Filter resonance as a value between 0 and 1. Large amounts of resonance (a res: near 1) can create a whistling sound around the cutoff frequency. Smaller values produce less resonance.</p> <p><i>Default: 0.5</i> <i>Must be zero or greater, must be a value less than 1</i> <i>May be changed whilst playing</i> <a href="#">Has slide parameters to shape changes</a></p>

## Slide Parameters

Any parameter that is slidable has three additional parameters named `_slide`, `_slide_curve`, and `_slide_shape`. For example, 'amp' is slidable, so you can also set `amp_slide`, `amp_slide_curve`, and `amp_slide_shape` with the following effects:

<code>_slide:</code>	Amount of time (in beats) for the parameter value to change. A long parameter <code>_slide</code> value means that the parameter takes a long time to slide from the previous value to the new value. A parameter <code>_slide</code> of 0 means that the parameter instantly changes to the new value. <i>Default: 0</i>
<code>_slide_shape:</code>	Shape of curve. 0: step, 1: linear, 3: sine, 4: welch, 5: custom (use * <code>_slide_curve</code> : opt e.g. <code>amp_slide_curve</code> ), 6: squared, 7: cubed. <i>Default: 5</i>
<code>_slide_curve:</code>	Shape of the slide curve (only honoured if slide shape is 5). 0 means linear and positive and negative numbers curve the segment up and down respectively. <i>Default: 0</i>

---

# Pan

```
amp: 1 mix: 1 pre_amp: 1 pan: 0
```

```
with_fx :pan do  
  play 50  
end
```

Specify where in the stereo field the sound should be heard. A value of -1 for pan will put the sound in the left speaker, a value of 1 will put the sound in the right speaker and values in between will shift the sound accordingly.

Introduced in v2.0

## Parameters

amp:	<p>The amplitude of the sound. Typically a value between 0 and 1. Higher amplitudes may be used, but won't make the sound louder, they will just reduce the quality of all the sounds currently being played (due to compression.)</p> <p><i>Default: 1</i> <i>Must be zero or greater</i> <i>May be changed whilst playing</i> <a href="#">Has slide parameters to shape changes</a></p>
mix:	<p>The amount (percentage) of FX present in the resulting sound represented as a value between 0 and 1. For example, a mix of 0 means that only the original sound is heard, a mix of 1 means that only the FX is heard (typically the default) and a mix of 0.5 means that half the original and half of the FX is heard.</p> <p><i>Default: 1</i> <i>Must be a value between 0 and 1 inclusively</i> <i>May be changed whilst playing</i> <a href="#">Has slide parameters to shape changes</a></p>
pre_amp:	<p>Amplification applied to the input signal immediately before it is passed to the FX.</p> <p><i>Default: 1</i> <i>Must be zero or greater</i> <i>May be changed whilst playing</i> <a href="#">Has slide parameters to shape changes</a> <i>Scaled with current BPM value</i></p>
pan:	<p>Position of sound in stereo. With headphones on, this means how much of the sound is in the left ear, and how much is in the right ear. With a value of -1, the sound is completely in the left ear, a value of 0 puts the sound equally in both ears and a value of 1 puts the sound in the right ear. Values in between -1 and 1 move the sound accordingly.</p> <p><i>Default: 0</i> <i>Must be a value between -1 and 1 inclusively</i> <i>May be changed whilst playing</i> <a href="#">Has slide parameters to shape changes</a></p>

## Slide Parameters

Any parameter that is slidable has three additional parameters named `_slide`, `_slide_curve`, and `_slide_shape`. For example, 'amp' is slidable, so you can also set `amp_slide`, `amp_slide_curve`, and `amp_slide_shape` with the following effects:

<code>_slide:</code>	<p>Amount of time (in beats) for the parameter value to change. A long parameter <code>_slide</code> value means that the parameter takes a long time to slide from the previous value to the new value. A parameter <code>_slide</code> of 0 means that the parameter instantly changes to the new value.</p> <p><i>Default: 0</i></p>
----------------------	---

**\_slide\_shape:**

Shape of curve. 0: step, 1: linear, 3: sine, 4: welch, 5: custom (use \*\_slide\_curve: opt e.g. amp\_slide\_curve:), 6: squared, 7: cubed.

*Default: 5*

**\_slide\_curve:**

Shape of the slide curve (only honoured if slide shape is 5). 0 means linear and positive and negative numbers curve the segment up and down respectively.

*Default: 0*

---



# Pan Slicer

amp:	1	mix:	1	pre_amp:	1	phase:	0.25	pan_min:	-1	pan_max:	1
pulse_width:	0.5	phase_offset:	0	wave:	1	invert_wave:	0	probability:	0	prob_pos:	0
seed:	0	smooth:	0	smooth_up:	0	smooth_down:	0				

```
with_fx :panslicer do
  play 50
end
```

Slice the pan automatically from left to right. Behaves similarly to slicer and wobble FX but modifies stereo panning of sound in left and right speakers. Default slice wave form is square (hard slicing between left and right) however other wave forms can be set with the **wave**: opt.

Introduced in v2.6

## Parameters

amp:	<p>The amplitude of the resulting effect.</p> <p><i>Default: 1</i> <i>Must be zero or greater</i> <i>May be changed whilst playing</i> <i>Has slide parameters to shape changes</i></p>
mix:	<p>The amount (percentage) of FX present in the resulting sound represented as a value between 0 and 1. For example, a mix of 0 means that only the original sound is heard, a mix of 1 means that only the FX is heard (typically the default) and a mix of 0.5 means that half the original and half of the FX is heard.</p> <p><i>Default: 1</i> <i>Must be a value between 0 and 1 inclusively</i> <i>May be changed whilst playing</i> <i>Has slide parameters to shape changes</i></p>
pre_amp:	<p>Amplification applied to the input signal immediately before it is passed to the FX.</p> <p><i>Default: 1</i> <i>Must be zero or greater</i> <i>May be changed whilst playing</i> <i>Has slide parameters to shape changes</i> <i>Scaled with current BPM value</i></p>
phase:	<p>The phase duration (in beats) of the slices</p> <p><i>Default: 0.25</i> <i>Must be greater than zero</i> <i>May be changed whilst playing</i> <i>Has slide parameters to shape changes</i> <i>Scaled with current BPM value</i></p>
pan_min:	<p>Minimum pan value (-1 is the left speaker only)</p> <p><i>Default: -1</i> <i>Must be zero or greater</i> <i>May be changed whilst playing</i> <i>Has slide parameters to shape changes</i></p>
pan_max:	<p>Maximum pan value (+1 is the right speaker only)</p> <p><i>Default: 1</i> <i>Must be zero or greater</i> <i>May be changed whilst playing</i> <i>Has slide parameters to shape changes</i></p>

<b>pulse_width:</b>	<p>The width of the pulse wave as a value between 0 and 1. A width of 0.5 will produce a square wave. Different values will change the timbre of the sound. Only valid if wave is type pulse.</p> <p><i>Default: 0.5</i>  <i>Must be a value between 0 and 1 exclusively</i>  <i>May be changed whilst playing</i>  <a href="#">Has slide parameters to shape changes</a></p>
<b>phase_offset:</b>	<p>Initial phase offset.</p> <p><i>Default: 0</i>  <i>Must be a value between 0 and 1 inclusively</i>  <i>Can not be changed once set</i></p>
<b>wave:</b>	<p>Control waveform used to modulate the amplitude. 0=saw, 1=pulse, 2=tri, 3=sine</p> <p><i>Default: 1</i>  <i>Must be one of the following values: [0, 1, 2, 3]</i>  <i>May be changed whilst playing</i></p>
<b>invert_wave:</b>	<p>Invert control waveform (i.e. flip it on the y axis). 0=uninverted wave, 1=inverted wave.</p> <p><i>Default: 0</i>  <i>Must be one of the following values: [0, 1]</i>  <i>May be changed whilst playing</i></p>
<b>probability:</b>	<p>Probability (as a value between 0 and 1) that a given slice will be replaced by the value of the prob_pos opt (which defaults to 0, i.e. silence)</p> <p><i>Default: 0</i>  <i>Must be a value between 0 and 1 inclusively</i>  <i>May be changed whilst playing</i>  <a href="#">Has slide parameters to shape changes</a></p>
<b>prob_pos:</b>	<p>Position of the slicer that will be jumped to when the probability test passes as a value between 0 and 1</p> <p><i>Default: 0</i>  <i>Must be a value between 0 and 1 inclusively</i>  <i>May be changed whilst playing</i>  <a href="#">Has slide parameters to shape changes</a></p>
<b>seed:</b>	<p>Seed value for rand num generator used for probability test</p> <p><i>Default: 0</i>  <i>Can not be changed once set</i></p>
<b>smooth:</b>	<p>Amount of time in seconds to transition from the current value to the next. Allows you to round off harsh edges in the slicer wave which may cause clicks.</p> <p><i>Default: 0</i>  <i>Must be zero or greater</i>  <i>May be changed whilst playing</i>  <a href="#">Has slide parameters to shape changes</a></p>
<b>smooth_up:</b>	<p>Amount of time in seconds to transition from the current value to the next only when the value is going up. This smoothing happens before the main smooth mechanism.</p> <p><i>Default: 0</i>  <i>Must be zero or greater</i>  <i>May be changed whilst playing</i>  <a href="#">Has slide parameters to shape changes</a></p>
<b>smooth_down:</b>	<p>Amount of time in seconds to transition from the current value to the next only when the value is going down. This smoothing happens before the main smooth mechanism.</p> <p><i>Default: 0</i>  <i>Must be zero or greater</i></p>

May be changed whilst playing  
Has slide parameters to shape changes

## Slide Parameters

Any parameter that is slidable has three additional parameters named `_slide`, `_slide_curve`, and `_slide_shape`. For example, 'amp' is slidable, so you can also set `amp_slide`, `amp_slide_curve`, and `amp_slide_shape` with the following effects:

<code>_slide:</code>	Amount of time (in beats) for the parameter value to change. A long parameter <code>_slide</code> value means that the parameter takes a long time to slide from the previous value to the new value. A parameter <code>_slide</code> of 0 means that the parameter instantly changes to the new value. <i>Default: 0</i>
<code>_slide_shape:</code>	Shape of curve. 0: step, 1: linear, 3: sine, 4: welch, 5: custom (use <code>*_slide_curve:</code> opt e.g. <code>amp_slide_curve:</code> ), 6: squared, 7: cubed. <i>Default: 5</i>
<code>_slide_curve:</code>	Shape of the slide curve (only honoured if slide shape is 5). 0 means linear and positive and negative numbers curve the segment up and down respectively. <i>Default: 0</i>

---

# Pitch shift

```
amp: 1 pre_amp: 1 mix: 1 window_size: 0.2 pitch: 0 pitch_dis: 0.0  
time_dis: 0.0
```

```
with_fx :pitch_shift do  
  play 50  
end
```

Changes the pitch of a signal without affecting tempo. Does this mainly through the pitch parameter which takes a midi number to transpose by. You can also play with the other params to produce some interesting textures and sounds.

Introduced in v2.5

## Parameters

amp:	<p>The amplitude of the sound. Typically a value between 0 and 1. Higher amplitudes may be used, but won't make the sound louder, they will just reduce the quality of all the sounds currently being played (due to compression.)</p> <p><i>Default: 1</i> <i>Must be zero or greater</i> <i>May be changed whilst playing</i> <a href="#">Has slide parameters to shape changes</a></p>
pre_amp:	<p>Amplification applied to the input signal immediately before it is passed to the FX.</p> <p><i>Default: 1</i> <i>Must be zero or greater</i> <i>May be changed whilst playing</i> <a href="#">Has slide parameters to shape changes</a> <i>Scaled with current BPM value</i></p>
mix:	<p>The amount (percentage) of FX present in the resulting sound represented as a value between 0 and 1. For example, a mix of 0 means that only the original sound is heard, a mix of 1 means that only the FX is heard (typically the default) and a mix of 0.5 means that half the original and half of the FX is heard.</p> <p><i>Default: 1</i> <i>Must be a value between 0 and 1 inclusively</i> <i>May be changed whilst playing</i> <a href="#">Has slide parameters to shape changes</a></p>
window_size:	<p>Pitch shift works by chopping the input into tiny slices, then playing these slices at a higher or lower rate. If we make the slices small enough and overlap them, it sounds like the original sound with the pitch changed.</p> <p>The window_size is the length of the slices and is measured in seconds. It needs to be around 0.2 (200ms) or greater for pitched sounds like guitar or bass, and needs to be around 0.02 (20ms) or lower for percussive sounds like drum loops. You can experiment with this to get the best sound for your input.</p> <p><i>Default: 0.2</i> <i>Must be a value greater than 5.0e-05</i> <i>May be changed whilst playing</i> <a href="#">Has slide parameters to shape changes</a></p>
pitch:	<p>Pitch adjustment in semitones. 1 is up a semitone, 12 is up an octave, -12 is down an octave etc. Maximum upper limit of 24 (up 2 octaves). Lower limit of -72 (down 6 octaves). Decimal numbers can be used for fine tuning.</p> <p><i>Default: 0</i> <i>Must be a value greater than or equal to -72, must be a value less than or equal to 24</i> <i>May be changed whilst playing</i></p>

	<i>Has slide parameters to shape changes</i>
<b>pitch_dis:</b>	<p>Pitch dispersion - how much random variation in pitch to add. Using a low value like 0.001 can help to “soften up” the metallic sounds, especially on drum loops. To be really technical, pitch_dispersion is the maximum random deviation of the pitch from the pitch ratio (which is set by the pitch param)</p> <p><i>Default: 0.0</i>  <i>Must be a value greater than or equal to 0</i>  <i>May be changed whilst playing</i>  <i>Has slide parameters to shape changes</i></p>
<b>time_dis:</b>	<p>Time dispersion - how much random delay before playing each grain (measured in seconds). Again, low values here like 0.001 can help to soften up metallic sounds introduced by the effect. Large values are also fun as they can make soundscapes and textures from the input, although you will most likely lose the rhythm of the original. NB - This won't have an effect if it's larger than window_size.</p> <p><i>Default: 0.0</i>  <i>Must be a value greater than or equal to 0</i>  <i>May be changed whilst playing</i>  <i>Has slide parameters to shape changes</i></p>

## Slide Parameters

Any parameter that is slidable has three additional parameters named `_slide`, `_slide_curve`, and `_slide_shape`. For example, 'amp' is slidable, so you can also set `amp_slide`, `amp_slide_curve`, and `amp_slide_shape` with the following effects:

<b>_slide:</b>	<p>Amount of time (in beats) for the parameter value to change. A long parameter_slide value means that the parameter takes a long time to slide from the previous value to the new value. A parameter_slide of 0 means that the parameter instantly changes to the new value.</p> <p><i>Default: 0</i></p>
<b>_slide_shape:</b>	<p>Shape of curve. 0: step, 1: linear, 3: sine, 4: welch, 5: custom (use *_slide_curve: opt e.g. amp_slide_curve:), 6: squared, 7: cubed.</p> <p><i>Default: 5</i></p>
<b>_slide_curve:</b>	<p>Shape of the slide curve (only honoured if slide shape is 5). 0 means linear and positive and negative numbers curve the segment up and down respectively.</p> <p><i>Default: 0</i></p>

# Resonant Band Pass Filter

```
amp: 1 mix: 1 pre_amp: 1 centre: 100 res: 0.5
```

```
with_fx :rbpf do  
  play 50  
end
```

Like the Band Pass Filter but with a resonance (slight volume boost) around the target frequency. This can produce an interesting whistling effect, especially when used with smaller values for the res opt.

Introduced in v2.3

## Parameters

amp:	<p>The amplitude of the sound. Typically a value between 0 and 1. Higher amplitudes may be used, but won't make the sound louder, they will just reduce the quality of all the sounds currently being played (due to compression.)</p> <p><i>Default: 1</i> <i>Must be zero or greater</i> <i>May be changed whilst playing</i> <a href="#">Has slide parameters to shape changes</a></p>
mix:	<p>The amount (percentage) of FX present in the resulting sound represented as a value between 0 and 1. For example, a mix of 0 means that only the original sound is heard, a mix of 1 means that only the FX is heard (typically the default) and a mix of 0.5 means that half the original and half of the FX is heard.</p> <p><i>Default: 1</i> <i>Must be a value between 0 and 1 inclusively</i> <i>May be changed whilst playing</i> <a href="#">Has slide parameters to shape changes</a></p>
pre_amp:	<p>Amplification applied to the input signal immediately before it is passed to the FX.</p> <p><i>Default: 1</i> <i>Must be zero or greater</i> <i>May be changed whilst playing</i> <a href="#">Has slide parameters to shape changes</a> <i>Scaled with current BPM value</i></p>
centre:	<p>Centre frequency for the filter as a MIDI note.</p> <p><i>Default: 100</i> <i>Must be a value greater than or equal to 0</i> <i>May be changed whilst playing</i> <a href="#">Has slide parameters to shape changes</a></p>
res:	<p>Filter resonance as a value between 0 and 1. Large amounts of resonance (a res: near 1) can create a whistling sound around the cutoff frequency. Smaller values produce less resonance.</p> <p><i>Default: 0.5</i> <i>Must be zero or greater, must be a value less than 1</i> <i>May be changed whilst playing</i> <a href="#">Has slide parameters to shape changes</a></p>

## Slide Parameters

Any parameter that is slidable has three additional parameters named `_slide`, `_slide_curve`, and `_slide_shape`. For example, 'amp' is slidable, so you can also set `amp_slide`, `amp_slide_curve`, and `amp_slide_shape` with the following effects:

Amount of time (in beats) for the parameter value to change. A long parameter slide value

<b>_slide:</b>	means that the parameter takes a long time to slide from the previous value to the new value. A parameter_slide of 0 means that the parameter instantly changes to the new value. <i>Default: 0</i>
<b>_slide_shape:</b>	Shape of curve. 0: step, 1: linear, 3: sine, 4: welch, 5: custom (use *_slide_curve: opt e.g. amp_slide_curve:), 6: squared, 7: cubed. <i>Default: 5</i>
<b>_slide_curve:</b>	Shape of the slide curve (only honoured if slide shape is 5). 0 means linear and positive and negative numbers curve the segment up and down respectively. <i>Default: 0</i>

---

# Reverb

```
amp: 1 mix: 0.4 pre_amp: 1 room: 0.6 damp: 0.5
```

```
with_fx :reverb do  
  play 50  
end
```

Make the incoming signal sound more spacious or distant as if it were played in a large room or cave. Signal may also be dampened by reducing the amplitude of the higher frequencies.

Introduced in v2.0

## Parameters

<b>amp:</b>	<p>The amplitude of the sound. Typically a value between 0 and 1. Higher amplitudes may be used, but won't make the sound louder, they will just reduce the quality of all the sounds currently being played (due to compression.)</p> <p><i>Default: 1</i> <i>Must be zero or greater</i> <i>May be changed whilst playing</i> <a href="#">Has slide parameters to shape changes</a></p>
<b>mix:</b>	<p>The amount (percentage) of FX present in the resulting sound represented as a value between 0 and 1. For example, a mix of 0 means that only the original sound is heard, a mix of 1 means that only the FX is heard (typically the default) and a mix of 0.5 means that half the original and half of the FX is heard.</p> <p><i>Default: 0.4</i> <i>Must be a value between 0 and 1 inclusively</i> <i>May be changed whilst playing</i> <a href="#">Has slide parameters to shape changes</a></p>
<b>pre_amp:</b>	<p>Amplification applied to the input signal immediately before it is passed to the FX.</p> <p><i>Default: 1</i> <i>Must be zero or greater</i> <i>May be changed whilst playing</i> <a href="#">Has slide parameters to shape changes</a> <i>Scaled with current BPM value</i></p>
<b>room:</b>	<p>The room size - a value between 0 (no reverb) and 1 (maximum reverb).</p> <p><i>Default: 0.6</i> <i>Must be a value between 0 and 1 inclusively</i> <i>May be changed whilst playing</i> <a href="#">Has slide parameters to shape changes</a></p>
<b>damp:</b>	<p>High frequency dampening - a value between 0 (no dampening) and 1 (maximum dampening)</p> <p><i>Default: 0.5</i> <i>Must be a value between 0 and 1 inclusively</i> <i>May be changed whilst playing</i> <a href="#">Has slide parameters to shape changes</a></p>

## Slide Parameters

Any parameter that is slidable has three additional parameters named `_slide`, `_slide_curve`, and `_slide_shape`. For example, 'amp' is slidable, so you can also set `amp_slide`, `amp_slide_curve`, and `amp_slide_shape` with the following effects:

Amount of time (in beats) for the parameter value to change. A long `parameter_slide` value means that the parameter takes a long time to slide from the previous value to the new value. A



<code>_slide:</code>	parameter_slide of 0 means that the parameter instantly changes to the new value. <i>Default: 0</i>
<code>_slide_shape:</code>	Shape of curve. 0: step, 1: linear, 3: sine, 4: welch, 5: custom (use *_slide_curve: opt e.g. amp_slide_curve:), 6: squared, 7: cubed. <i>Default: 5</i>
<code>_slide_curve:</code>	Shape of the slide curve (only honoured if slide shape is 5). 0 means linear and positive and negative numbers curve the segment up and down respectively. <i>Default: 0</i>

---

# Resonant High Pass Filter

```
amp: 1 mix: 1 pre_amp: 1 cutoff: 100 res: 0.5
```

```
with_fx :rhpf do  
  play 50  
end
```

Dampens the parts of the signal that are lower than the cutoff point (typically the bass of the sound) and keeps the higher parts (typically the crunchy fizzy harmonic overtones). The resonant part of the resonant high pass filter emphasises/resonates the frequencies around the cutoff point. The amount of emphasis is controlled by the res param with a lower res resulting in greater resonance. High amounts of resonance (rq ~0) can create a whistling sound around the cutoff frequency.

Choose a lower cutoff to keep more of the bass/mid and a higher cutoff to make the sound more light and crispy.

Introduced in v2.0

## Parameters

amp:	<p>The amplitude of the sound. Typically a value between 0 and 1. Higher amplitudes may be used, but won't make the sound louder, they will just reduce the quality of all the sounds currently being played (due to compression.)</p> <p><i>Default: 1</i> <i>Must be zero or greater</i> <i>May be changed whilst playing</i> <a href="#">Has slide parameters to shape changes</a></p>
mix:	<p>The amount (percentage) of FX present in the resulting sound represented as a value between 0 and 1. For example, a mix of 0 means that only the original sound is heard, a mix of 1 means that only the FX is heard (typically the default) and a mix of 0.5 means that half the original and half of the FX is heard.</p> <p><i>Default: 1</i> <i>Must be a value between 0 and 1 inclusively</i> <i>May be changed whilst playing</i> <a href="#">Has slide parameters to shape changes</a></p>
pre_amp:	<p>Amplification applied to the input signal immediately before it is passed to the FX.</p> <p><i>Default: 1</i> <i>Must be zero or greater</i> <i>May be changed whilst playing</i> <a href="#">Has slide parameters to shape changes</a> <i>Scaled with current BPM value</i></p>
cutoff:	<p>MIDI note representing the highest frequencies allowed to be present in the sound. A low value like 30 makes the sound round and dull, a high value like 100 makes the sound buzzy and crispy.</p> <p><i>Default: 100</i> <i>Must be zero or greater, must be a value less than 131</i> <i>May be changed whilst playing</i> <a href="#">Has slide parameters to shape changes</a></p>
res:	<p>Filter resonance as a value between 0 and 1. Large amounts of resonance (a res: near 1) can create a whistling sound around the cutoff frequency. Smaller values produce less resonance.</p> <p><i>Default: 0.5</i> <i>Must be zero or greater, must be a value less than 1</i> <i>May be changed whilst playing</i> <a href="#">Has slide parameters to shape changes</a></p>

## Slide Parameters

Any parameter that is slidable has three additional parameters named `_slide`, `_slide_curve`, and `_slide_shape`. For example, 'amp' is slidable, so you can also set `amp_slide`, `amp_slide_curve`, and `amp_slide_shape` with the following effects:

<code>_slide:</code>	Amount of time (in beats) for the parameter value to change. A long parameter <code>_slide</code> value means that the parameter takes a long time to slide from the previous value to the new value. A parameter <code>_slide</code> of 0 means that the parameter instantly changes to the new value. <i>Default: 0</i>
<code>_slide_shape:</code>	Shape of curve. 0: step, 1: linear, 3: sine, 4: welch, 5: custom (use <code>*_slide_curve:</code> opt e.g. <code>amp_slide_curve:</code> ), 6: squared, 7: cubed. <i>Default: 5</i>
<code>_slide_curve:</code>	Shape of the slide curve (only honoured if slide shape is 5). 0 means linear and positive and negative numbers curve the segment up and down respectively. <i>Default: 0</i>

---

# Ring Modulator

```
freq: 30 amp: 1 mix: 1 pre_amp: 1 mod_amp: 1
```

```
with_fx :ring_mod do  
  play 50  
end
```

Attack of the Daleks! Ring mod is a classic effect often used on soundtracks to evoke robots or aliens as it sounds hollow or metallic. We take a 'carrier' signal (a sine wave controlled by the freq opt) and modulate its amplitude using the signal given inside the fx block. This produces a wide variety of sounds - the best way to learn is to experiment!

Introduced in v2.3

## Parameters

freq:	Frequency of the carrier signal (as a midi note). <i>Default: 30</i> <i>Must be greater than zero</i> <i>May be changed whilst playing</i> <a href="#">Has slide parameters to shape changes</a>
amp:	The amplitude of the sound. Typically a value between 0 and 1. Higher amplitudes may be used, but won't make the sound louder, they will just reduce the quality of all the sounds currently being played (due to compression.) <i>Default: 1</i> <i>Must be zero or greater</i> <i>May be changed whilst playing</i> <a href="#">Has slide parameters to shape changes</a>
mix:	The amount (percentage) of FX present in the resulting sound represented as a value between 0 and 1. For example, a mix of 0 means that only the original sound is heard, a mix of 1 means that only the FX is heard (typically the default) and a mix of 0.5 means that half the original and half of the FX is heard. <i>Default: 1</i> <i>Must be a value between 0 and 1 inclusively</i> <i>May be changed whilst playing</i> <a href="#">Has slide parameters to shape changes</a>
pre_amp:	Amplification applied to the input signal immediately before it is passed to the FX. <i>Default: 1</i> <i>Must be zero or greater</i> <i>May be changed whilst playing</i> <a href="#">Has slide parameters to shape changes</a> Scaled with current BPM value
mod_amp:	Amplitude of the modulation <i>Default: 1</i> <i>Must be zero or greater</i> <i>May be changed whilst playing</i> <a href="#">Has slide parameters to shape changes</a>

## Slide Parameters

Any parameter that is slidable has three additional parameters named `_slide`, `_slide_curve`, and `_slide_shape`. For example, 'amp' is slidable, so you can also set `amp_slide`, `amp_slide_curve`, and `amp_slide_shape` with the following effects:

<b>_slide:</b>	Amount of time (in beats) for the parameter value to change. A long parameter_slide value means that the parameter takes a long time to slide from the previous value to the new value. A parameter_slide of 0 means that the parameter instantly changes to the new value. <i>Default: 0</i>
<b>_slide_shape:</b>	Shape of curve. 0: step, 1: linear, 3: sine, 4: welch, 5: custom (use *_slide_curve: opt e.g. amp_slide_curve:), 6: squared, 7: cubed. <i>Default: 5</i>
<b>_slide_curve:</b>	Shape of the slide curve (only honoured if slide shape is 5). 0 means linear and positive and negative numbers curve the segment up and down respectively. <i>Default: 0</i>

---

# Resonant Low Pass Filter

```
amp: 1 mix: 1 pre_amp: 1 cutoff: 100 res: 0.5
```

```
with_fx :rlpf do  
  play 50  
end
```

Dampens the parts of the signal that are above than the cutoff point (typically the crunchy fizzy harmonic overtones) and keeps the lower parts (typically the bass/mid of the sound). behaviour, The resonant part of the resonant low pass filter emphasises/resonates the frequencies around the cutoff point. The amount of emphasis is controlled by the res param with a lower res resulting in greater resonance. High amounts of resonance (rq ~0) can create a whistling sound around the cutoff frequency.

Choose a higher cutoff to keep more of the high frequencies/treble of the sound and a lower cutoff to make the sound more dull and only keep the bass.

Introduced in v2.0

## Parameters

amp:	<p>The amplitude of the sound. Typically a value between 0 and 1. Higher amplitudes may be used, but won't make the sound louder, they will just reduce the quality of all the sounds currently being played (due to compression.)</p> <p><i>Default: 1</i> <i>Must be zero or greater</i> <i>May be changed whilst playing</i> <a href="#">Has slide parameters to shape changes</a></p>
mix:	<p>The amount (percentage) of FX present in the resulting sound represented as a value between 0 and 1. For example, a mix of 0 means that only the original sound is heard, a mix of 1 means that only the FX is heard (typically the default) and a mix of 0.5 means that half the original and half of the FX is heard.</p> <p><i>Default: 1</i> <i>Must be a value between 0 and 1 inclusively</i> <i>May be changed whilst playing</i> <a href="#">Has slide parameters to shape changes</a></p>
pre_amp:	<p>Amplification applied to the input signal immediately before it is passed to the FX.</p> <p><i>Default: 1</i> <i>Must be zero or greater</i> <i>May be changed whilst playing</i> <a href="#">Has slide parameters to shape changes</a> <i>Scaled with current BPM value</i></p>
cutoff:	<p>MIDI note representing the highest frequencies allowed to be present in the sound. A low value like 30 makes the sound round and dull, a high value like 100 makes the sound buzzy and crispy.</p> <p><i>Default: 100</i> <i>Must be zero or greater, must be a value less than 131</i> <i>May be changed whilst playing</i> <a href="#">Has slide parameters to shape changes</a></p>
res:	<p>Filter resonance as a value between 0 and 1. Large amounts of resonance (a res: near 1) can create a whistling sound around the cutoff frequency. Smaller values produce less resonance.</p> <p><i>Default: 0.5</i> <i>Must be zero or greater, must be a value less than 1</i> <i>May be changed whilst playing</i> <a href="#">Has slide parameters to shape changes</a></p>

## Slide Parameters

Any parameter that is slidable has three additional parameters named `_slide`, `_slide_curve`, and `_slide_shape`. For example, 'amp' is slidable, so you can also set `amp_slide`, `amp_slide_curve`, and `amp_slide_shape` with the following effects:

<code>_slide:</code>	Amount of time (in beats) for the parameter value to change. A long parameter <code>_slide</code> value means that the parameter takes a long time to slide from the previous value to the new value. A parameter <code>_slide</code> of 0 means that the parameter instantly changes to the new value. <i>Default: 0</i>
<code>_slide_shape:</code>	Shape of curve. 0: step, 1: linear, 3: sine, 4: welch, 5: custom (use <code>*_slide_curve:</code> opt e.g. <code>amp_slide_curve:</code> ), 6: squared, 7: cubed. <i>Default: 5</i>
<code>_slide_curve:</code>	Shape of the slide curve (only honoured if slide shape is 5). 0 means linear and positive and negative numbers curve the segment up and down respectively. <i>Default: 0</i>

---

# Slicer

amp:	1	mix:	1	pre_amp:	1	phase:	0.25	amp_min:	0	amp_max:	1
pulse_width:	0.5	phase_offset:	0	wave:	1	invert_wave:	0	probability:	0	prob_pos:	0
seed:	0	smooth:	0	smooth_up:	0	smooth_down:	0				

```
with_fx :slicer do
  play 50
end
```

Modulates the amplitude of the input signal with a specific control wave and phase duration. With the default pulse wave, slices the signal in and out, with the triangle wave, fades the signal in and out and with the saw wave, phases the signal in and then dramatically out. Control wave may be inverted with the arg `invert_wave` for more variety.

Introduced in v2.0

## Parameters

amp:	<p>The amplitude of the resulting effect.</p> <p><i>Default: 1</i> <i>Must be zero or greater</i> <i>May be changed whilst playing</i> <i>Has slide parameters to shape changes</i></p>
mix:	<p>The amount (percentage) of FX present in the resulting sound represented as a value between 0 and 1. For example, a mix of 0 means that only the original sound is heard, a mix of 1 means that only the FX is heard (typically the default) and a mix of 0.5 means that half the original and half of the FX is heard.</p> <p><i>Default: 1</i> <i>Must be a value between 0 and 1 inclusively</i> <i>May be changed whilst playing</i> <i>Has slide parameters to shape changes</i></p>
pre_amp:	<p>Amplification applied to the input signal immediately before it is passed to the FX.</p> <p><i>Default: 1</i> <i>Must be zero or greater</i> <i>May be changed whilst playing</i> <i>Has slide parameters to shape changes</i> <i>Scaled with current BPM value</i></p>
phase:	<p>The phase duration (in beats) of the slices</p> <p><i>Default: 0.25</i> <i>Must be greater than zero</i> <i>May be changed whilst playing</i> <i>Has slide parameters to shape changes</i> <i>Scaled with current BPM value</i></p>
amp_min:	<p>Minimum amplitude of the slicer</p> <p><i>Default: 0</i> <i>Must be zero or greater</i> <i>May be changed whilst playing</i> <i>Has slide parameters to shape changes</i></p>
amp_max:	<p>Maximum amplitude of the slicer</p> <p><i>Default: 1</i> <i>Must be zero or greater</i> <i>May be changed whilst playing</i></p>



	<a href="#">Has slide parameters to shape changes</a>
<b>pulse_width:</b>	<p>The width of the pulse wave as a value between 0 and 1. A width of 0.5 will produce a square wave. Different values will change the timbre of the sound. Only valid if wave is type pulse.</p> <p><i>Default: 0.5</i>  <i>Must be a value between 0 and 1 exclusively</i>  <i>May be changed whilst playing</i>  <a href="#">Has slide parameters to shape changes</a></p>
<b>phase_offset:</b>	<p>Initial phase offset.</p> <p><i>Default: 0</i>  <i>Must be a value between 0 and 1 inclusively</i>  <i>Can not be changed once set</i></p>
<b>wave:</b>	<p>Control waveform used to modulate the amplitude. 0=saw, 1=pulse, 2=tri, 3=sine</p> <p><i>Default: 1</i>  <i>Must be one of the following values: [0, 1, 2, 3]</i>  <i>May be changed whilst playing</i></p>
<b>invert_wave:</b>	<p>Invert control waveform (i.e. flip it on the y axis). 0=uninverted wave, 1=inverted wave.</p> <p><i>Default: 0</i>  <i>Must be one of the following values: [0, 1]</i>  <i>May be changed whilst playing</i></p>
<b>probability:</b>	<p>Probability (as a value between 0 and 1) that a given slice will be replaced by the value of the prob_pos opt (which defaults to 0, i.e. silence)</p> <p><i>Default: 0</i>  <i>Must be a value between 0 and 1 inclusively</i>  <i>May be changed whilst playing</i>  <a href="#">Has slide parameters to shape changes</a></p>
<b>prob_pos:</b>	<p>Position of the slicer that will be jumped to when the probability test passes as a value between 0 and 1</p> <p><i>Default: 0</i>  <i>Must be a value between 0 and 1 inclusively</i>  <i>May be changed whilst playing</i>  <a href="#">Has slide parameters to shape changes</a></p>
<b>seed:</b>	<p>Seed value for rand num generator used for probability test</p> <p><i>Default: 0</i>  <i>Can not be changed once set</i></p>
<b>smooth:</b>	<p>Amount of time in seconds to transition from the current value to the next. Allows you to round off harsh edges in the slicer wave which may cause clicks.</p> <p><i>Default: 0</i>  <i>Must be zero or greater</i>  <i>May be changed whilst playing</i>  <a href="#">Has slide parameters to shape changes</a></p>
<b>smooth_up:</b>	<p>Amount of time in seconds to transition from the current value to the next only when the value is going up. This smoothing happens before the main smooth mechanism.</p> <p><i>Default: 0</i>  <i>Must be zero or greater</i>  <i>May be changed whilst playing</i>  <a href="#">Has slide parameters to shape changes</a></p>
<b>smooth_down:</b>	<p>Amount of time in seconds to transition from the current value to the next only when the value is going down. This smoothing happens before the main smooth mechanism.</p> <p><i>Default: 0</i></p>

Must be zero or greater  
May be changed whilst playing  
*Has slide parameters to shape changes*

## Slide Parameters

Any parameter that is slidable has three additional parameters named `_slide`, `_slide_curve`, and `_slide_shape`. For example, 'amp' is slidable, so you can also set `amp_slide`, `amp_slide_curve`, and `amp_slide_shape` with the following effects:

<code>_slide:</code>	Amount of time (in beats) for the parameter value to change. A long parameter <code>_slide</code> value means that the parameter takes a long time to slide from the previous value to the new value. A parameter <code>_slide</code> of 0 means that the parameter instantly changes to the new value.  <i>Default: 0</i>
<code>_slide_shape:</code>	Shape of curve. 0: step, 1: linear, 3: sine, 4: welch, 5: custom (use <code>*_slide_curve: opt</code> e.g. <code>amp_slide_curve:</code> ), 6: squared, 7: cubed.  <i>Default: 5</i>
<code>_slide_curve:</code>	Shape of the slide curve (only honoured if slide shape is 5). 0 means linear and positive and negative numbers curve the segment up and down respectively.  <i>Default: 0</i>

---

# Wobble

amp:	1	mix:	1	pre_amp:	1	phase:	0.5	cutoff_min:	60	cutoff_max:	120
res:	0.8	phase_offset:	0	wave:	0	invert_wave:	0	pulse_width:	0.5	filter:	0
probability:	0	prob_pos:	0	seed:	0	smooth:	0	smooth_up:	0	smooth_down:	0

```
with_fx :wobble do
  play 50
end
```

Versatile wobble FX. Will repeatedly modulate a range of filters (rlpf, rhpf) between two cutoff values using a range of control wave forms (saw, pulse, tri, sine). You may alter the phase duration of the wobble, and the resonance of the filter. Combines well with the dsaw synth for crazy dub wobbles. Cutoff value is at cutoff\_min at the start of phase

Introduced in v2.0

## Parameters

amp:	<p>The amplitude of the sound. Typically a value between 0 and 1. Higher amplitudes may be used, but won't make the sound louder, they will just reduce the quality of all the sounds currently being played (due to compression.)</p> <p><i>Default: 1</i> <i>Must be zero or greater</i> <i>May be changed whilst playing</i> <a href="#">Has slide parameters to shape changes</a></p>
mix:	<p>The amount (percentage) of FX present in the resulting sound represented as a value between 0 and 1. For example, a mix of 0 means that only the original sound is heard, a mix of 1 means that only the FX is heard (typically the default) and a mix of 0.5 means that half the original and half of the FX is heard.</p> <p><i>Default: 1</i> <i>Must be a value between 0 and 1 inclusively</i> <i>May be changed whilst playing</i> <a href="#">Has slide parameters to shape changes</a></p>
pre_amp:	<p>Amplification applied to the input signal immediately before it is passed to the FX.</p> <p><i>Default: 1</i> <i>Must be zero or greater</i> <i>May be changed whilst playing</i> <a href="#">Has slide parameters to shape changes</a> <i>Scaled with current BPM value</i></p>
phase:	<p>The phase duration (in beats) for filter modulation cycles</p> <p><i>Default: 0.5</i> <i>Must be greater than zero</i> <i>May be changed whilst playing</i> <a href="#">Has slide parameters to shape changes</a> <i>Scaled with current BPM value</i></p>
cutoff_min:	<p>Minimum (MIDI) note that filter will move to whilst wobbling. Choose a lower note for a higher range of movement. Full range of movement is the distance between cutoff_max and cutoff_min</p> <p><i>Default: 60</i> <i>Must be zero or greater, must be a value less than 130</i> <i>May be changed whilst playing</i> <a href="#">Has slide parameters to shape changes</a></p>
	<p>Maximum (MIDI) note that filter will move to whilst wobbling. Choose a higher note for a higher range of movement. Full range of movement is the distance between cutoff_max and cutoff_min</p>

<b>cutoff_max:</b>	<p>Default: 120          Must be zero or greater, must be a value less than 130          May be changed whilst playing  <a href="#">Has slide parameters to shape changes</a></p>
<b>res:</b>	<p>Filter resonance as a value between 0 and 1. Large amounts of resonance (a res: near 1) can create a whistling sound around the cutoff frequency. Smaller values produce less resonance.</p> <p>Default: 0.8          Must be zero or greater, must be a value less than 1          May be changed whilst playing  <a href="#">Has slide parameters to shape changes</a></p>
<b>phase_offset:</b>	<p>Initial modulation phase offset (a value between 0 and 1).</p> <p>Default: 0          Must be a value between 0 and 1 inclusively          Can not be changed once set</p>
<b>wave:</b>	<p>Wave shape of wobble. Use 0 for saw wave, 1 for pulse, 2 for triangle wave and 3 for a sine wave.</p> <p>Default: 0          Must be one of the following values: [0, 1, 2, 3]          May be changed whilst playing</p>
<b>invert_wave:</b>	<p>Invert control waveform (i.e. flip it on the y axis). 0=uninverted wave, 1=inverted wave.</p> <p>Default: 0          Must be one of the following values: [0, 1]          May be changed whilst playing</p>
<b>pulse_width:</b>	<p>Only valid if wave is type pulse.</p> <p>Default: 0.5          Must be zero or greater          May be changed whilst playing  <a href="#">Has slide parameters to shape changes</a></p>
<b>filter:</b>	<p>Filter used for wobble effect. Use 0 for a resonant low pass filter or 1 for a resonant high pass filter</p> <p>Default: 0          Must be one of the following values: [0, 1]          May be changed whilst playing</p>
<b>probability:</b>	<p>Probability (as a value between 0 and 1) that a given wobble will be replaced by the value of the prob_pos opt (which defaults to 0, i.e. min_cutoff)</p> <p>Default: 0          Must be a value between 0 and 1 inclusively          May be changed whilst playing  <a href="#">Has slide parameters to shape changes</a></p>
<b>prob_pos:</b>	<p>Position of the wobble that will be jumped to when the probability test passes as a value between 0 and 1</p> <p>Default: 0          Must be a value between 0 and 1 inclusively          May be changed whilst playing  <a href="#">Has slide parameters to shape changes</a></p>
<b>seed:</b>	<p>Seed value for rand num generator used for probability test</p> <p>Default: 0          Can not be changed once set</p>
	<p>Amount of time in seconds to transition from the current value to the next. Allows you to round off harsh edges in the slicer wave which may cause clicks.</p>

<b>smooth:</b>	<p><i>Default: 0</i>  <i>Must be zero or greater</i>  <i>May be changed whilst playing</i>  <a href="#">Has slide parameters to shape changes</a></p>
<b>smooth_up:</b>	<p>Amount of time in seconds to transition from the current value to the next only when the value is going up. This smoothing happens before the main smooth mechanism.</p> <p><i>Default: 0</i>  <i>Must be zero or greater</i>  <i>May be changed whilst playing</i>  <a href="#">Has slide parameters to shape changes</a></p>
<b>smooth_down:</b>	<p>Amount of time in seconds to transition from the current value to the next only when the value is going down. This smoothing happens before the main smooth mechanism.</p> <p><i>Default: 0</i>  <i>Must be zero or greater</i>  <i>May be changed whilst playing</i>  <a href="#">Has slide parameters to shape changes</a></p>

## Slide Parameters

Any parameter that is slidable has three additional parameters named `_slide`, `_slide_curve`, and `_slide_shape`. For example, 'amp' is slidable, so you can also set `amp_slide`, `amp_slide_curve`, and `amp_slide_shape` with the following effects:

<b><code>_slide:</code></b>	<p>Amount of time (in beats) for the parameter value to change. A long <code>parameter_slide</code> value means that the parameter takes a long time to slide from the previous value to the new value. A <code>parameter_slide</code> of 0 means that the parameter instantly changes to the new value.</p> <p><i>Default: 0</i></p>
<b><code>_slide_shape:</code></b>	<p>Shape of curve. 0: step, 1: linear, 3: sine, 4: welch, 5: custom (use <code>*_slide_curve:</code> opt e.g. <code>amp_slide_curve:</code>), 6: squared, 7: cubed.</p> <p><i>Default: 5</i></p>
<b><code>_slide_curve:</code></b>	<p>Shape of the slide curve (only honoured if slide shape is 5). 0 means linear and positive and negative numbers curve the segment up and down respectively.</p> <p><i>Default: 0</i></p>

---

all\_sample\_names  
at  
bools  
choose  
chord  
chord\_degree  
chord\_names  
comment  
control  
cue  
current\_arg\_checks  
current\_beat\_duration  
current\_bpm  
current\_debug  
current\_sample\_defaults  
current\_sample\_pack  
current\_sample\_pack\_aliases  
current\_sched\_ahead\_time  
current\_synth  
current\_synth\_defaults  
current\_transpose  
current\_volume  
dec  
define  
defonce  
degree  
density  
dice  
factor?  
hz\_to\_midi  
in\_thread  
inc  
invert\_chord  
kill  
knit  
line  
live\_loop  
load\_sample  
load\_samples  
load\_synthdefs  
look  
mc\_block\_id  
mc\_block\_ids  
mc\_block\_name  
mc\_block\_names  
mc\_camera\_fixed  
mc\_camera\_normal  
mc\_camera\_set\_location  
mc\_camera\_third\_person  
mc\_chat\_post  
mc\_checkpoint\_restore  
mc\_checkpoint\_save  
mc\_get\_block  
mc\_get\_height  
mc\_get\_pos  
mc\_get\_tile  
mc\_ground\_height  
mc\_location  
mc\_message  
mc\_set\_area  
mc\_set\_block  
mc\_set\_pos  
mc\_set\_tile  
mc\_surface\_teleport  
mc\_teleport  
midi\_notes  
midi\_to\_hz  
ndefine  
note  
note\_info  
note\_range  
one\_in  
pitch\_to\_ratio  
play  
play\_chord  
play\_pattern  
play\_pattern\_timed

print  
puts  
quantise  
ramp  
rand  
rand\_back  
rand\_i  
rand\_reset  
rand\_skip  
range  
ratio\_to\_pitch  
rdist  
rest?  
ring  
rrand  
rrand\_i  
rt  
sample  
sample\_buffer  
sample\_duration  
sample\_groups  
sample\_info  
sample\_loaded?  
sample\_names  
scale  
scale\_names  
set\_control\_delta!  
set\_mixer\_control!  
set\_sched\_ahead\_time!  
set\_volume!  
shuffle  
sleep  
spark  
spark\_graph  
spread  
status  
stop  
stretch  
sync  
synth  
tick  
tick\_reset  
tick\_reset\_all  
tick\_set  
uncomment  
use\_arg\_bpm\_scaling  
use\_arg\_checks  
use\_bpm  
use\_bpm\_mul  
use\_cue\_logging  
use\_debug  
use\_merged\_synth\_defaults  
use\_random\_seed  
use\_sample\_bpm  
use\_sample\_defaults  
use\_sample\_pack  
use\_sample\_pack\_as  
use\_synth  
use\_synth\_defaults  
use\_transpose  
use\_tuning  
vector  
version  
vt  
wait  
with\_arg\_bpm\_scaling  
with\_arg\_checks  
with\_bpm  
with\_bpm\_mul  
with\_cue\_logging  
with\_debug  
with\_fx  
with\_merged\_synth\_defaults  
with\_random\_seed  
with\_sample\_bpm  
with\_sample\_defaults  
with\_sample\_pack

with\_sample\_pack\_as  
with\_synth  
with\_synth\_defaults  
with\_transpose  
with\_tuning

## Get all sample names

`all_sample_names`

Return a list of all the sample names available

Introduced in v2.0

## Asynchronous Time. Run a block at the given time(s)

`at times (list), params (list)`

Given a list of times, run the block once after waiting each given time. If passed an optional params list, will pass each param individually to each block call. If size of params list is smaller than the times list, the param values will act as rings (rotate through). If the block is given 1 arg, the times are fed through. If the block is given 2 args, both the times and the params are fed through. A third block arg will receive the index of the time.

Introduced in v2.1

## Examples

<pre># Example 1</pre>	
<pre>at [1, 2, 4] do   play 75 end</pre>	<pre># plays a note after waiting 1 beat, # then after 1 more beat, # then after 2 more beats (4 beats total)</pre>
<pre># Example 2</pre>	
<pre>at [1, 2, 3], [75, 76, 77] do  n    play n end</pre>	<pre># plays 3 different notes</pre>
<pre># Example 3</pre>	
<pre>at [1, 2, 3],   [[:amp=&gt;0.5], [:amp=&gt; 0.8]] do  p    sample :drum_cymbal_open, p end</pre>	<pre># alternate soft and loud # cymbal hits three times</pre>
<pre># Example 4</pre>	
<pre>at [0, 1, 2] do  t    puts t end</pre>	<pre># when no params are given to at, the times are fed through to the block #=&gt; prints 0, 1, then 2</pre>
<pre># Example 5</pre>	
<pre>at [0, 1, 2], [:a, :b] do  t, b    puts [t, b] end</pre>	<pre>#If you specify the block with 2 args, it will pass through both the tim #=&gt; prints out [0, :a], [1, :b], then [2, :a]</pre>
<pre># Example 6</pre>	
<pre>at [0, 0.5, 2] do  t, idx    puts [t, idx] end</pre>	<pre>#If you specify the block with 2 args, and no param list to at, it will #=&gt; prints out [0, 0], [0.5, 1], then [2, 2]</pre>



### # Example 7

```
at [0, 0.5, 2], [:a, :b] do |t, b, idx|
  puts [t, b, idx]
end
```

```
#If you specify the block with 3 args, it will pass through the time, th
#=> prints out [0, :a, 0], [0.5, :b, 1], then [2, :a, 2]
```

## Create a ring of boolean values

`bools list` (array)

Create a new ring of booleans values from 1s, and 0s which can be easier to write and manipulate in a live setting.

Introduced in v2.2

### Examples

#### # Example 1

```
(bools 1, 0)
```

```
#=> (ring true, false)
```

#### # Example 2

```
(bools 1, 0, true, false, nil)
```

```
#=> (ring true, false, true, false, false)
```

## Random list selection

`choose list` (array)

Choose an element at random from a list (array).

Introduced in v2.0

### Example

#### # Example 1

```
loop do
  play choose([60, 64, 67])
  sleep 1
  play chord(:c, :major).choose
  sleep 1
end
```

```
#=> plays one of 60, 64 or 67 at random
```

```
#=> You can also call .choose on the list
```

## Create chord

`chord tonic` (symbol), `name` (symbol)

Creates a ring of Midi note numbers when given a tonic note and a chord type

Introduced in v2.0

### Options

`invert:`

Apply the specified num inversions to chord. See the fn `invert_chord`.

`num_octaves:`

Create an arpeggio of the chord over n octaves

## Examples

### # Example 1

```
puts chord(:e, :minor) # returns a list of midi notes - [64, 67, 71]
```

### # Example 2

```
play chord(:e, :minor) # Play all the notes together
```

### # Example 3

```
loop do
  play chord(:e, :minor).choose
  sleep 0.2
end

# looping over arpeggios can sound good
# Here we use choose to pick a random note from the chord
```

### # Example 4

```
use_bpm 150
play chord(:C, '1')
sleep 1
play chord(:C, '5')
sleep 1
play chord(:C, '+5')
sleep 1
play chord(:C, 'm+5')
sleep 1
play chord(:C, :sus2)
sleep 1
play chord(:C, :sus4)
sleep 1
play chord(:C, '6')
sleep 1
play chord(:C, :m6)
sleep 1
play chord(:C, '7sus2')
sleep 1
play chord(:C, '7sus4')
sleep 1
play chord(:C, '7-5')
sleep 1
play chord(:C, 'm7-5')
sleep 1
play chord(:C, '7+5')
sleep 1
play chord(:C, 'm7+5')
sleep 1
play chord(:C, '9')
sleep 1
play chord(:C, :m9)
sleep 1
play chord(:C, 'm7+9')
sleep 1
play chord(:C, :maj9)
sleep 1
play chord(:C, '9sus4')
sleep 1
play chord(:C, '6*9')
sleep 1
play chord(:C, 'm6*9')
sleep 1
play chord(:C, '7-9')
sleep 1
play chord(:C, 'm7-9')
sleep 1
play chord(:C, '7-10')
sleep 1
play chord(:C, '9+5')
```

```
# Sonic Pi supports a large range of chords
# Notice that the more exotic ones have to be surrounded by ' quotes
# this is just to get through all the chords more quickly
```

```

sleep 1
play chord(:C, 'm9+5')
sleep 1
play chord(:C, '7+5-9')
sleep 1
play chord(:C, 'm7+5-9')
sleep 1
play chord(:C, '11')
sleep 1
play chord(:C, :m11)
sleep 1
play chord(:C, :maj11)
sleep 1
play chord(:C, '11+')
sleep 1
play chord(:C, 'm11+')
sleep 1
play chord(:C, '13')
sleep 1
play chord(:C, :m13)
sleep 1
play chord(:C, :major)
sleep 1
play chord(:C, :M)
sleep 1
play chord(:C, :minor)
sleep 1
play chord(:C, :m)
sleep 1
play chord(:C, :major7)
sleep 1
play chord(:C, :dom7)
sleep 1
play chord(:C, '7')
sleep 1
play chord(:C, :M7)
sleep 1
play chord(:C, :minor7)
sleep 1
play chord(:C, :m7)
sleep 1
play chord(:C, :augmented)
sleep 1
play chord(:C, :a)
sleep 1
play chord(:C, :diminished)
sleep 1
play chord(:C, :dim)
sleep 1
play chord(:C, :i)
sleep 1
play chord(:C, :diminished7)
sleep 1
play chord(:C, :dim7)
sleep 1
play chord(:C, :i7)
sleep 1

```

## Construct chords of stacked thirds, based on scale degrees

`chord_degree degree (symbol_or_number), tonic (symbol), scale (symbol), number_of_notes (number)`

In music we build chords from scales. For example, a C major chord is made by taking the 1st, 3rd and 5th notes of the C major scale (C, E and G). If you do this on a piano you might notice that you play one, skip one, play one, skip one etc. If we use the same spacing and start from the second note in C major (which is a D), we get a D minor chord which is the 2nd, 4th and 6th notes in C major (D, F and A). We can move this pattern all the way up or down the scale to get different types of chords. `chord_degree` is a helper method that returns a ring of midi note numbers when given a degree (starting point in a scale) which is a symbol `:i`, `:ii`, `:iii`, `:iv`, `:v`, `:vi`, `:vii` or a number `1-7`. The second argument is the tonic note of the scale, the third argument is the scale type and finally the fourth argument is number of notes to stack up in the chord. If we choose 4 notes from degree `:i` of the C major scale, we take the 1st, 3rd, 5th and 7th notes of the scale to get a C major 7 chord.

Introduced in v2.1

### Examples

```
# Example 1
```

```
puts chord_degree(:i, :A3, :major) # returns a ring of midi notes - (ring 57, 61, 64, 68) - an A major 7 choi
```

```
# Example 2
```

```
play chord_degree(:i, :A3, :major, 3)
```

```
# Example 3
```

```
play chord_degree(:ii, :A3, :major, 3) # Chord ii in A major is a B minor chord
```

```
# Example 4
```

```
play chord_degree(:iii, :A3, :major, 3) # Chord iii in A major is a C# minor chord
```

```
# Example 5
```

```
play chord_degree(:iv, :A3, :major, 3) # Chord iv in A major is a D major chord
```

```
# Example 6
```

```
play chord_degree(:i, :C4, :major, 4) # Taking four notes is the default. This gives us 7th chords - here it pl
```

```
# Example 7
```

```
play chord_degree(:i, :C4, :major, 5) # Taking five notes gives us 9th chords - here it plays a C major 9 chord
```

## All chord names

`chord_names`

Returns a ring containing all chord names known to Sonic Pi

Introduced in v2.6

### Example

```
# Example 1
```

```
puts chord_names #=> prints a list of all the chords
```

## Block level commenting

`comment`

Does not evaluate any of the code within the block. However, any optional args passed before the block *will* be evaluated although they will be ignored. See `uncomment` for switching commenting off without having to remove the comment form.

Introduced in v2.0

### Example

```
# Example 1
```

```
comment do # starting a block level comment:
```

```

play 50 # not played
sleep 1 # no sleep happens
play 62 # not played
end

```

## Control running synth

`control node` (synth\_node)

Control a running synth node by passing new parameters to it. A synth node represents a running synth and can be obtained by assigning the return value of a call to `play` or `sample` or by specifying a parameter to the `do/end` block of an FX. You may modify any of the parameters you can set when triggering the synth, sample or FX. See documentation for opt details. If the synth to control is a chord, then control will change all the notes of that chord group at once to a new target set of notes - see example.

Introduced in v2.0

## Examples

<code># Example 1</code>	
<pre> my_node = play 50, release: 5, cutoff: 60 sleep 1 control my_node, cutoff: 70 sleep 1 control my_node, cutoff: 90 </pre>	<pre> # Basic control # play note 50 with release of 5 and cutoff # Sleep for a second # Now modify cutoff from 60 to 70, sound i # Sleep for another second # Now modify cutoff from 70 to 90, sound i </pre>
<code># Example 2</code>	
<pre> s = synth :prophet, note: :e1, cutoff: 70, cutoff_slide: 8, release: 8 control s, cutoff: 130 </pre>	<pre> # Combining control with slide opts allows # start synth and specify slide time for c # Change the cutoff value with a control. # Cutoff will now slide over 8 beats from </pre>
<code># Example 3</code>	
<pre> notes = (scale :e3, :minor_pentatonic, num_octaves: 2).shuffle  s = synth :beep, note: :e3, sustain: 8, note_slide: 0.05 64.times do   control s, note: notes.tick   sleep 0.125 end </pre>	<pre> # Use a short slide time and many controls # get a random ordering of a scale # Start our synth running with a long sust # Keep quickly changing the note by tickin </pre>
<code># Example 4</code>	
<pre> with_fx :bitcrusher, sample_rate: 1000, sample_rate_slide: 8 do  bc     sample :loop_garzul, rate: 1   control bc, sample_rate: 5000  end </pre>	<pre> # Controlling FX # Start FX but also use the handy    goalp # to grab a handle on the running FX. We c # our handle anything we want. Here we've # We can use our handle bc now just like w # previous example to modify the FX as it </pre>
<code># Example 5</code>	
<pre> cg = play (chord :e4, :minor), sustain: 2 sleep 1 control cg, notes: (chord :c3, :major) </pre>	<pre> # Controlling chords # start a chord # transition to new chord. </pre>

	<pre># Each note in the original chord is mapped # to the equivalent in the new chord.</pre>
<b># Example 6</b>	
<pre>cg = play (chord :e4, :minor), sustain: 4, note_slide: 3 sleep 1 control cg, notes: (chord :c3, :major)</pre>	<pre># Sliding between chords # start a chord # slide to new chord. # Each note in the original chord is mapped # to the equivalent in the new chord.</pre>
<b># Example 7</b>	
<pre>cg = play (chord :e3, :m13), sustain: 4, note_slide: 3 sleep 1 control cg, notes: (chord :c3, :major)</pre>	<pre># Sliding from a larger to smaller chord # start a chord with 7 notes # slide to new chord with fewer notes (3) # Each note in the original chord is mapped # to the equivalent in the new chord using ri # This means that the 4th note in the original # chord will be mapped onto the 1st note in the second</pre>
<b># Example 8</b>	
<pre>cg = play (chord :c3, :major), sustain: 4, note_slide: 3 sleep 1 control cg, notes: (chord :e3, :m13)</pre>	<pre># Sliding from a smaller to larger chord # start a chord with 3 notes # slide to new chord with more notes (7) # Each note in the original chord is mapped # to the equivalent in the new chord. # This means that the 4th note in the new # chord will not sound as there is no 4th note in # the original chord.</pre>

## Cue other threads

`cue cue_id (symbol)`

Send a heartbeat synchronisation message containing the (virtual) timestamp of the current thread. Useful for syncing up external threads via the `sync` fn. Any opts which are passed are given to the thread which syncs on the `cue_id` as a map. The values of the opts must be immutable. Currently only numbers, symbols and booleans are supported.

Introduced in v2.0

### Options

<code>your_key:</code>	Your value
<code>another_key:</code>	Another value
<code>key:</code>	All these opts are passed through to the thread which syncs

### Examples

<b># Example 1</b>	
<pre>in_thread do   sync :foo   sample :ambi_lunar_land end  sleep 5  cue :foo</pre>	<pre># this parks the current thread waiting for a foo cue message to be received.  # We send a cue message from the main thread. # This then unblocks the thread above and we then hear the sample</pre>

## # Example 2

```
in_thread do
  loop do
    cue :tick
    sleep 0.5
  end
end

loop do
  sync :tick
  sample :drum_heavy_kick
end
```

```
# Start a metronome thread
# Loop forever:
# sending tick heartbeat messages
# and sleeping for 0.5 beats between ticks

# We can now play sounds using the metronome.
# In the main thread, just loop
# waiting for :tick cue messages
# after which play the drum kick sample
```

## # Example 3

```
in_thread do
  loop do
    cue [:foo, :bar, :baz].choose
    sleep 0.5
  end
end

in_thread do
  loop do
    sync :foo
    sample :elec_beep
  end
end

in_thread do
  loop do
    sync :bar
    sample :elec_flip
  end
end

in_thread do
  loop do
    sync :baz
    sample :elec_blup
  end
end
```

```
# Start a metronome thread
# Loop forever:
# sending one of three tick heartbeat messages randomly
# and sleeping for 0.5 beats between ticks

# We can now play sounds using the metronome:

# In the main thread, just loop
# waiting for :foo cue messages
# after which play the elec beep sample

# In the main thread, just loop
# waiting for :bar cue messages
# after which play the elec flip sample

# In the main thread, just loop
# waiting for :baz cue messages
# after which play the elec blup sample
```

## Get current arg checking status

### current\_arg\_checks

Returns the current arg checking setting (**true** or **false**).

Introduced in v2.0

## Example

### # Example 1

```
puts current_arg_checks # Print out the current arg check setting
```

## Duration of current beat

### current\_beat\_duration





## Get current sample pack

`current_sample_pack`

Returns the current sample pack.

Introduced in v2.0

### Example

```
# Example 1
puts current_sample_pack # Print out the current sample pack
```

## Get current sample pack aliases

`current_sample_pack_aliases`

Returns a map containing the current sample pack aliases.

Introduced in v2.0

### Example

```
# Example 1
puts current_sample_pack_aliases # Print out the current sample pack aliases
```

## Get current sched ahead time

`current_sched_ahead_time`

Returns the current schedule ahead time.

Introduced in v2.0

### Example

```
# Example 1
set_sched_ahead_time! 0.5
puts current_sched_ahead_time # Prints 0.5
```

## Get current synth

`current_synth`

Returns the current synth name.

Introduced in v2.0

### Example

```
# Example 1
puts current_synth # Print out the current synth name
```

## Get current synth defaults

### current\_synth\_defaults

Returns the current synth defaults. This is a map of synth arg names to either values or functions.

Introduced in v2.0

### Example

```
# Example 1
use_synth_defaults amp: 0.5, cutoff: 80
play 50
puts current_synth_defaults
```

```
# Plays note 50 with amp 0.5 and cutoff 80
#=> Prints {amp: 0.5, cutoff: 80}
```

## Get current transposition

### current\_transpose

Returns the current transpose value.

Introduced in v2.0

### Example

```
# Example 1
puts current_transpose
```

```
# Print out the current transpose value
```

## Get current volume

### current\_volume

Returns the current volume.

Introduced in v2.0

### Examples

```
# Example 1
puts current_volume
```

```
# Print out the current volume
```

```
# Example 2
set_volume! 2
puts current_volume
```

```
#=> 2
```

## Decrement

### dec n (number)

Decrement a number by 1. Equivalent to  $n - 1$

Introduced in v2.1

### Examples

```
# Example 1
dec 1 # returns 0

# Example 2
dec -1 # returns -2
```

## Define a new function

`define name (symbol)`

Allows you to group a bunch of code and give it your own name for future re-use. Functions are very useful for structuring your code. They are also the gateway into live coding as you may redefine a function whilst a thread is calling it, and the next time the thread calls your function, it will use the latest definition.

Introduced in v2.0

### Example

```
# Example 1

define :foo do
  play 50
  sleep 1
end

foo

# Call foo on its own

# You can use foo anywhere you would use normal code.
# For example, in a block:

3.times do
  foo
end
```

## Define a named value only once

`defonce name (symbol)`

Allows you to assign the result of some code to a name, with the property that the code will only execute once - therefore stopping re-definitions. This is useful for defining values that you use in your compositions but you don't want to reset every time you press run. You may force the block to execute again regardless of whether or not it has executed once already by using the override option (see examples).

Introduced in v2.0

### Options

```
override: If set to true, re-definitions are allowed and this acts like
define
```

### Examples

```
# Example 1

defonce :foo do
  sleep 1

  puts "hello"
  10
end

# Define a new function called foo
# Sleep for a beat in the function definition. Note that this amount
# of time in seconds will depend on the current BPM of the live_loop
# or thread calling this function.
# Print hello
# Return a value of 10
```

```

puts foo
puts foo

defonce :foo do
  puts "you can't redefine me"
  15
end

puts foo

3.times do
  play foo
end

```

```

# Call foo on its own
# The run sleeps for a beat and prints "hello" before returning 10

# Try it again:
# This time the run doesn't sleep or print anything out. However, 10 is still

# Try redefining foo

# We still don't see any printing or sleeping, and the result is still 10

# You can use foo anywhere you would use normal code.
# For example, in a block:

# play 10

```

```

# Example 2

defonce :bar do
  50
end

play bar

defonce :bar do
  70
end

play bar

defonce :bar, override: true do
  80
end

play bar

```

```

# plays 50

# This redefinition doesn't work due to the behaviour of defonce

# Still plays 50

# Force definition to take place with override option

# plays 80

```

## Convert a degree into a note

`degree` `degree` (symbol\_or\_number), `tonic` (symbol), `scale` (symbol)

For a given scale and tonic it takes a symbol `:i`, `:ii`, `:iii`, `:iv`, `:v`, `:vi`, `:vii` or a number `1-7` and resolves it to a midi note.

Introduced in v2.1

### Example

```

# Example 1

play degree(:ii, :D3, :major)
play degree(2, :C3, :minor)

```

## Squash and repeat time

`density` `d` (density)

Runs the block `d` times with the bpm for the block also multiplied by `d`. Great for repeating sections a number of times faster yet keeping within a fixed time. If `d` is less than 1, then time will be stretched accordingly and the block will take longer to complete.

Introduced in v2.3

### Examples

```

# Example 1

```

<pre>use_bpm 60    density 2 do     sample :bd_hause     sleep 0.5   end</pre>	<pre># Set the BPM to 60  # BPM for block is now 120 # block is called 2.times # sample is played twice # sleep is 0.25s</pre>
<b># Example 2</b>	
<pre>density 2 do  idx    puts idx   sleep 0.5 end</pre>	<pre># You may also pass a param to the block similar to n.times # prints out 0, 1 # sleep is 0.25s</pre>
<b># Example 3</b>	
<pre>density 0.5 do    play 80, release: 1   sleep 0.5 end</pre>	<pre># Specifying a density val of &lt; 1 will stretch out time # A density of 0.5 will double the length of the block's # execution time. # plays note 80 with 2s release # sleep is 1s</pre>

## Random dice throw

`dice num_sides` (number)

Throws a dice with the specified `num_sides` (defaults to **6**) and returns the score as a number between **1** and `num_sides`.

Introduced in v2.0

### Examples

```
# Example 1
dice # will return a number between 1 and 6 inclusively
# (with an even probability distribution).
```

```
# Example 2
dice 3 # will return a number between 1 and 3 inclusively
```

## Factor test

`factor? val` (number), `factor` (number)

Test to see if `factor` is indeed a factor of `val`. In other words, can `val` be divided exactly by `factor`.

Introduced in v2.1

### Examples

```
# Example 1
factor?(10, 2) # true - 10 is a multiple of 2 (2 * 5 = 10)
```

```
# Example 2
factor?(11, 2) #false - 11 is not a multiple of 2
```

```
# Example 3
```

```
factor?(2, 0.5) #true - 2 is a multiple of 0.5 (0.5 * 4 = 2)
```

## Hz to MIDI conversion

`hz_to_midi freq` (number)

Convert a frequency in hz to a midi note. Note that the result isn't an integer and there is a potential for some very minor rounding errors.

Introduced in v2.0

### Example

```
# Example 1
```

```
hz_to_midi(261.63) #=> 60.0003
```

## Run code block at the same time

`in_thread`

Execute a given block (between `do ... end`) in a new thread. Use for playing multiple 'parts' at once. Each new thread created inherits all the use/with defaults of the parent thread such as the time, current synth, bpm, default synth args, etc. Despite inheriting defaults from the parent thread, any modifications of the defaults in the new thread will *not* affect the parent thread. Threads may be named with the `name:` optional arg. Named threads will print their name in the logging pane when they print their activity. Finally, if you attempt to create a new named thread with a name that is already in use by another executing thread, no new thread will be created.

Introduced in v2.0

### Options

`name:` Make this thread a named thread with name. If a thread with this name already exists, a new thread will not be created.

`delay:` Initial delay in beats before the thread starts. Default is 0.

### Examples

```
# Example 1
```

```
loop do
  play 50
  sleep 1
end

loop do
  play 55
  sleep 0.5
end
```

```
# If you write two loops one after another like this,
# then only the first loop will execute as the loop acts
# like a trap not letting the flow of control out
```

```
# This code is never executed.
```

```
# Example 2
```

```
# In order to play two loops at the same time, the first loops need to
# be in a thread (note that it's probably more idiomatic to use live_loop
# when performing):
```

```
# By wrapping our loop in an in_thread block, we split the
# control flow into two parts. One flows into the loop (a) and
# the other part flows immediately after the in_thread block (b).
# both parts of the control flow execute at exactly the same time.
```

```

in_thread do
  loop do
    play 50
    sleep 1
  end
end

loop do
  play 55
  sleep 0.5
end

```

# (a)

# (a)

# (b)

# This loop is executed thanks to the thread above

#### # Example 3

```

use_bpm 120
use_synth :dsaw

in_thread do
  play 50
  use_synth :fm
  sleep 1
  play 38
end

play 62
sleep 2
play 67

```

# Set the bpm to be double rate  
# Set the current synth to be :dsaw

# Create a new thread  
# Play note 50 at time 0  
# Switch to fm synth (only affects this thread)  
# sleep for 0.5 seconds (as we're double rate)  
# Play note 38 at time 0.5

# Play note 62 at time 0 (with dsaw synth)  
# sleep 1s  
# Play note 67 at time 1s (also with dsaw synth)

#### # Example 4

```

in_thread(name: :foo) do
  loop do
    sample :drum_bass_hard
    sleep 1
  end
end

in_thread(name: :foo) do
  loop do
    sample :elec_chime
    sleep 0.5
  end
end

```

# Here we've created a named thread

# This thread isn't created as the name is  
# the same as the previous thread which is  
# still executing.

#### # Example 5

```

define :foo do
  play 50
  sleep 1
end

in_thread(name: :main) do
  loop do
    foo
  end
end

```

# Named threads work well with functions for live coding:  
# Create a function foo  
# which does something simple  
# and sleeps for some time

# Create a named thread  
# which loops forever  
# calling our function

# We are now free to modify the contents of :foo and re-run the entire buffer.  
# We'll hear the effect immediately without having to stop and re-start the code.  
# This is because our fn has been redefined, (which our thread will pick up) and  
# due to the thread being named, the second re-run will not create a new similarly  
# named thread. This is a nice pattern for live coding and is the basis of live\_loop

#### # Example 6

```

in_thread delay: 1 do
  sample :ambi_lunar_land
end

```

#Delaying the start of a thread

# this sample is not triggered at time 0 but after 1 beat

```
end
play 80 # Note 80 is played at time 0
```

## Increment

`inc n` (number)

Increment a number by **1**. Equivalent to `n + 1`

Introduced in v2.1

### Examples

```
# Example 1
```

```
inc 1 # returns 2
```

```
# Example 2
```

```
inc -1 # returns 0
```

## Invert a chord

`invert_chord notes` (list), `shift` (number)

Given a set of notes, apply a number of inversions indicated by Shift. Inversions being an increase to notes if Shift is positive or decreasing the notes if Shift is negative.

Introduced in v2.6

### Example

```
# Example 1
```

```
play invert_chord(chord(:A3, "M"), 0) #No inversion
sleep 1
play invert_chord(chord(:A3, "M"), 1) #First chord inversion
sleep 1
play invert_chord(chord(:A3, "M"), 2) #Second chord inversion
```

## Kill synth

`kill node` (synth\_node)

Kill a running synth sound or sample. In order to kill a sound, you need to have stored a reference to it in a variable.

Introduced in v2.0

### Examples

```
# Example 1
```

```
foo = play 50, release: 4
sleep 1
kill foo

# store a reference to a running synth in a variable called foo:
# foo is still playing, but we can kill it early:
```



### # Example 2

```
bar = sample :loop_amen
sleep 0.5
kill bar
```

## Knit a sequence of repeated values

`knit value (anything), count (number)`

Knits a series of value, count pairs to create a ring buffer where each value is repeated count times.

Introduced in v2.2

### Examples

#### # Example 1

```
(knit 1, 5) #=> (ring 1, 1, 1, 1, 1)
```

#### # Example 2

```
(knit :e2, 2, :c2, 3) #=> (ring :e2, :e2, :c2, :c2, :c2)
```

## Create a ring buffer representing a straight line

`line start (number), finish (number)`

Create a ring buffer representing a straight line between start and finish of `num_slices` elements. Num slices defaults to **8**. Indexes wrap around positively and negatively. Similar to `range`.

Introduced in v2.5

### Options

<code>steps:</code>	number of slices or segments along the line
<code>inclusive:</code>	boolean value representing whether or not to include finish value in line

### Examples

#### # Example 1

```
(line 0, 4, steps: 4) #=> (ring 0.0, 1.0, 2.0, 3.0)
```

#### # Example 2

```
(line 5, 0, steps: 5) #=> (ring 5.0, 4.0, 3.0, 2.0, 1.0)
```

#### # Example 3

```
(line 0, 3, inclusive: true) #=> (ring 0.0, 1.0, 2.0, 3.0)
```

# A loop for live coding

`live_loop name` (symbol)

Run the block in a new thread with the given name, and loop it forever. Also sends a `cue` with the same name each time the block runs. If the block is given a parameter, this is given the result of the last run of the loop (with initial value either being `0` or an `init` arg).

Introduced in v2.1

## Options

<code>init:</code>	initial value for optional block arg
<code>auto_cue:</code>	enable or disable automatic cue (default is true)
<code>delay:</code>	Initial delay in beats before the <code>live_loop</code> starts. Default is 0.
<code>seed:</code>	override initial random generator seed before starting loop.

## Examples

# Example 1	
<pre>live_loop :ping do   sample :elec_ping   sleep 1 end</pre>	
# Example 2	
<pre>live_loop :foo do  a    puts a   sleep 1   a += 1 end</pre>	<pre># pass a param (a) to the block (inits to 0) # prints out all the integers  # increment a by 1 (last value is passed back into the loop)</pre>

# Pre-load sample

`load_sample path` (string)

Given a path to a `.wav`, `.wave`, `.aif` or `.aiff` file, this loads the file and makes it available as a sample. See `load_samples` for loading multiple samples in one go.

Introduced in v2.0

## Example

# Example 1	
<pre>load_sample :elec_blip sample :elec_blip</pre>	<pre># :elec_blip is now loaded and ready to play as a sample # No delay takes place when attempting to trigger it</pre>

# Pre-load samples

`load_samples paths` (list)

Given an array of paths to `.wav`, `.wave`, `.aif` or `.aiff` files, loads them all into memory so that they may be played with via `sample` with no delay. See `load_sample`.

Introduced in v2.0

## Examples

#### # Example 1

```
sample :ambi_choir # This has to first load the sample before it can play it which may
                    # cause unwanted delay.

load_samples [:elec_plip, :elec_blip] # Let's load some samples in advance of using them
sample :elec_plip # When we play :elec_plip, there is no extra delay
                  # as it has already been loaded.
```

#### # Example 2

```
load_samples :elec_plip, :elec_blip # You may omit the square brackets, and
sample :elec_blip # simply list all samples you wish to load
                  # Before playing them.
```

#### # Example 3

```
load_samples ["/home/pi/samples/foo.wav"] # You may also load full paths to samples.
sample "/home/pi/sample/foo.wav" # And then trigger them with no more loading.
```

## Load external synthdefs

`load_synthdefs path (string)`

Load all pre-compiled synth designs in the specified directory. The binary files containing synth designs need to have the extension `.scsyndef`. This is useful if you wish to use your own SuperCollider synthesiser designs within Sonic Pi.

### Important note

If you wish your synth to work with Sonic Pi's automatic stereo sound infrastructure *you need to ensure your synth outputs a stereo signal* to an audio bus with an index specified by a synth arg named `out_bus`. For example, the following synth would work nicely:

```
(
  SynthDef(piTest,
    {|freq = 200, amp = 1, out_bus = 0 |
      Out.ar(out_bus,
        SinOsc.ar([freq,freq],0,0.5)* Line.kr(1, 0, 5, amp, doneAction: 2))}
  ).store;
)
```

Introduced in v2.0

### Example

#### # Example 1

```
load_synthdefs "~/Desktop/my_noises" # Load all synthdefs in my_noises folder
```

## Obtain value of a tick

`look`

Read and return value of default tick. If a `key` is specified, read the value of that specific tick. Ticks are `in_thread` and `live_loop` local, so the tick read will be the tick of the current thread calling `look`.

Introduced in v2.6

### Options

`offset:` Offset to add to index returned. Useful when calling look on lists, rings and vectors to offset the returned value

## Examples

# Example 1

```
puts look
  puts look
  puts look
```

```
#=> 0
#=> 0
#=> 0 # look doesn't advance the tick, it just returns the current
```

# Example 2

```
puts look
  tick
  puts look
  tick
  puts look
  puts look
  tick
  puts look
```

```
#=> 0 # A look is always 0 before the first tick
# advance the tick
#=> 0 # Note: a look is still 0 after the first tick.

#=> 1
#=> 1 # making multiple calls to look doesn't affect tick value

#=> 2
```

# Example 3

```
tick(:foo)
  tick(:foo)
  puts look(:foo)
  puts look
  puts look(:bar)
```

```
#=> 1 (keyed look :foo has been advanced)
#=> 0 (default look hasn't been advanced)
#=> 0 (other keyed looks haven't been advanced either)
```

# Example 4

```
live_loop :foo do
  tick
  use_synth :beep
  play (scale :e3, :minor_pentatonic).look
  sleep 0.5
  use_synth :square
  play (ring :e1, :e2, :e3).look, release: 0.25
  sleep 0.25
end
```

```
# You can call look on lists and rings
# advance the default tick
# look into the default tick to play all notes in sequence
# use the same look on another ring
```

## Minecraft Pi - normalise block code

`mc_block_id name (symbol_or_number)`

Given a block name or id will return a number representing the id of the block or throw an exception if the name or id isn't valid

Introduced in v2.5

## Examples

# Example 1

```
puts mc_block_id :air
```

```
#=> 0
```

# Example 2

```
puts mc_block_id 0
```

```
#=> 0
```

# Example 3

```
puts mc_block_id 19
```

```
#=> Throws an invalid block id exception
```

```
# Example 4
```

```
puts mc_block_id :foo #=> Throws an invalid block name exception
```

## Minecraft Pi - list all block ids

`mc_block_ids`

Returns a list of all the valid block ids as numbers. Note not all numbers are valid block ids. For example, 19 is not a valid block id.

Introduced in v2.5

### Example

```
# Example 1
```

```
puts mc_block_ids #=> [0, 1, 2, 3, 4, 5...
```

## Minecraft Pi - normalise block name

`mc_block_name id` (number\_or\_symbol)

Given a block id or a block name will return a symbol representing the block name or throw an exception if the id or name isn't valid.

Introduced in v2.5

### Examples

```
# Example 1
```

```
puts mc_block_name :air #=> :air
```

```
# Example 2
```

```
puts mc_block_name 0 #=> :air
```

```
# Example 3
```

```
puts mc_block_name 19 #=> Throws an invalid block id exception
```

```
# Example 4
```

```
puts mc_block_name :foo #=> Throws an invalid block name exception
```

## Minecraft Pi - list all block names

`mc_block_names`

Returns a list of all the valid block names as symbols

Introduced in v2.5

### Example

```
# Example 1
```

```
puts mc_block_names #=> [:air, :stone, :grass, :dirt, :cobblestone...
```

## Minecraft Pi - fixed camera mode

`mc_camera_fixed`

Set the camera mode to fixed.

Introduced in v2.5

### Example

```
# Example  
1
```

## Minecraft Pi - normal camera mode

`mc_camera_normal`

Set the camera mode to normal.

Introduced in v2.5

### Example

```
# Example  
1
```

## Minecraft Pi - move camera

`mc_camera_set_location`

Move the camera to a new location.

Introduced in v2.5

### Example

```
# Example  
1
```

## Minecraft Pi - third person camera mode

`mc_camera_third_person`

Set the camera mode to third person

Introduced in v2.5

### Example

```
# Example
1
```

## Minecraft Pi - synonym for mc\_message

`mc_chat_post`

See `mc_message`

Introduced in v2.5

## Minecraft Pi - restore checkpoint

`mc_checkpoint_restore`

Restore the world to the last snapshot taken with `mc_checkpoint_save`.

Introduced in v2.5

### Example

```
# Example
1
```

## Minecraft Pi - save checkpoint

`mc_checkpoint_save`

Take a snapshot of the world and save it. Restore back with `mc_checkpoint_restore`

Introduced in v2.5

### Example

```
# Example
1
```

## Minecraft Pi - get type of block at coords

`mc_get_block` `x` (number), `y` (number), `z` (number)

Returns the type of the block at the coords `x`, `y`, `z` as a symbol.

Introduced in v2.5

### Example

```
# Example 1
puts mc_get_block 40, 50, 60 #=> :air
```

## Minecraft Pi - synonym for mc\_ground\_height

---

`mc_get_height`

See `mc_ground_height`

Introduced in v2.5

---

## Minecraft Pi - synonym for `mc_location`

`mc_get_pos`

See `mc_location`

Introduced in v2.5

---

## Minecraft Pi - get location of current tile/block

`mc_get_tile`

Returns the coordinates of the nearest block that the player is next to. This is more coarse grained than `mc_location` as it only returns whole number coordinates.

Introduced in v2.5

---

### Example

```
# Example 1
puts mc_get_tile #=> [10, 20, 101]
```

---

## Minecraft Pi - get ground height at x, z coords

`mc_ground_height` `x` (number), `z` (number)

Returns the height of the ground at the specified `x` and `z` coords.

Introduced in v2.5

---

### Example

```
# Example 1
puts mc_ground_height 40, 50 #=> 43 (height of world at x=40, z=50)
```

---

## Minecraft Pi - get current location

`mc_location`

Returns a list of floats `[x, y, z]` coords of the current location for Steve. The coordinates are finer grained than raw block coordinates but may be used anywhere you might use block coords.

Introduced in v2.5

---

### Examples

```
# Example 1
puts mc_location #=> [10.1, 20.67, 101.34]
```

```
# Example 2
x, y, z = mc_location #=> Find the current location and store in x, y and z variables.
```



---

## Minecraft Pi - post a chat message

`mc_message` `msg` (string)

Post contents of `msg` on the Minecraft chat display

Introduced in v2.5

### Example

```
# Example 1
```

```
mc_message "Hello from Sonic Pi" #=> Displays "Hello from Sonic Pi" on Minecraft's chat display
```

---

## Minecraft Pi - set area of blocks

`mc_set_area` `block_name` (symbol\_or\_number), `x` (number), `y` (number), `z` (number), `x2` (number), `y2` (number)

Set an area/box of blocks of type `block_name` defined by two distinct sets of coordinates.

Introduced in v2.5

---

## Minecraft Pi - set block at specific coord

`mc_set_block` `x` (number), `y` (number), `z` (number), `block_name` (symbol\_or\_number)

Change the block type of the block at coords `x`, `y`, `z` to `block_type`. The block type may be specified either as a symbol such as `:air` or a number. See `mc_block_ids` and `mc_block_types` for lists of valid symbols and numbers.

Introduced in v2.5

### Example

```
# Example 1
```

```
mc_set_block :glass, 40, 50, 60 #=> set block at coords 40, 50, 60 to type glass
```

---

## Minecraft Pi - synonym for mc\_teleport

`mc_set_pos`

See `mc_teleport`

Introduced in v2.5

---

## Minecraft Pi - set location to coords of specified tile/block

`mc_set_tile` `x` (number), `y` (number), `z` (number)

Introduced in v2.5

### Example

```
# Example  
1
```

## Minecraft Pi - teleport to world surface at x and z coords

`mc_surface_teleport` `x` (number), `z` (number)

Teleports you to the specified x and z coordinates with the y automatically set to place you on the surface of the world. For example, if the x and z coords target a mountain, you'll be placed on top of the mountain, not in the air or under the ground. See `mc_ground_height` for discovering the height of the ground at a given x, z point.

Introduced in v2.5

### Example

```
# Example 1
mc_surface_teleport 40, 50 #=> Teleport user to coords x = 40, y = height of surface, z = 50
```

## Minecraft Pi - teleport to a new location

`mc_teleport` `x` (number), `y` (number), `z` (number)

Magically teleport the player to the location specified by the x, y, z coordinates. Use this for automatically moving the player either small or large distances around the world.

Introduced in v2.5

### Example

```
# Example 1
mc_teleport 40, 50, 60 # The player will be moved to the position with coords:
# x: 40, y: 50, z: 60
```

## Create a ring buffer of midi note numbers

`midi_notes` `list` (array)

Create a new immutable ring buffer of notes from args. Indexes wrap around positively and negatively. Final ring consists only of MIDI numbers and nil.

Introduced in v2.7

### Examples

```
# Example 1
(midi_notes :d3, :d4, :d5) #=> (ring 50, 62, 74)

# Example 2
(midi_notes :d3, 62, nil) #=> (ring 50, 62, nil)
```

## MIDI to Hz conversion

`midi_to_hz` `note` (symbol\_or\_number)

Convert a midi note to hz

Introduced in v2.0

## Example

```
# Example 1
midi_to_hz(60) #=> 261.6256
```

## Define a new function

`ndefine name (symbol)`

Does nothing. Use to stop a define from actually defining. Simpler than wrapping whole define in a comment block or commenting each individual line out.

Introduced in v2.1

## Describe note

`note note (symbol_or_number)`

Takes a midi note, a symbol (e.g. `:C`) or a string (e.g. `"C"`) and resolves it to a midi note. You can also pass an optional `octave:` parameter to get the midi note for a given octave. Please note - `octave:` param overrides any octave specified in a symbol i.e. `:c3`. If the note is `nil`, `:r` or `:rest`, then `nil` is returned (`nil` represents a rest)

Introduced in v2.0

## Options

<code>octave:</code>	The octave of the note. Overrides any octave declaration in the note symbol such as <code>:c2</code> . Default is 4
----------------------	---

## Examples

```
# Example 1
puts note(60)
puts note(:C)
puts note(:C4)
puts note('C')
# These all return 60 which is the midi number for middle C (octave 4)

# Example 2
puts note(60, octave: 2)
# returns 60 - octave param has no effect if we pass in a number

puts note(:C, octave: 2)
puts note(:C4, octave: 2)
puts note('C', octave: 2)
# These all return 36 which is the midi number for C2 (two octaves below middle C)
# note the octave param overrides any octaves specified in a symbol
```

## Get note info

`note_info note (symbol_or_number)`

Returns an instance of `SonicPi::Note`. Please note - `octave:` param overrides any octave specified in a symbol i.e. `:c3`

Introduced in v2.0

## Options

**octave:** The octave of the note. Overrides any octave declaration in the note symbol such as :c2. Default is 4

## Example

```
# Example 1
puts note_info(:C, octave: 2) # returns #<SonicPi::Note:0x0000010206bf78 @pitch_class="C", @octave=2, @interval=0,
```

## Get a range of notes

`note_range low_note (note), high_note (note)`

Produces a ring of all the notes between a low note and a high note. By default this is chromatic (all the notes) but can be filtered with a `:pitches` argument. This opens the door to arpeggiator style sequences and other useful patterns. If you try to specify only pitches which aren't in the range it will raise an error - you have been warned!

Introduced in v2.6

## Options

**pitches:** An array of notes (symbols or ints) to filter on. Octave information is ignored.

## Examples

```
# Example 1
(note_range :c4, :c5) # => (ring 60,61,62,63,64,65,66,67,68,69,70,71,72)
```

```
# Example 2
(note_range :c4, :c5, pitches: (chord :c, :major)) # => (ring 60,64,67,72)
```

```
# Example 3
(note_range :c4, :c6, pitches: (chord :c, :major)) # => (ring 60,64,67,72,76,79,84)
```

```
# Example 4
(note_range :c4, :c5, pitches: (scale :c, :major)) # => (ring 60,62,64,65,67,69,71,72)
```

```
# Example 5
(note_range :c4, :c5, pitches: [:c4, :g2]) # => (ring 60,67,72)
```

```
# Example 6
live_loop :arpeggiator do
  play (note_range :c4, :c5, pitches: (chord :c, :major)).tick
  sleep 0.125
end # try changing the chord
```

## Random true value with specified probability

`one_in num` (number)

Returns `true` or `false` with a specified probability - it will return true every one in num times where num is the param you specify

Introduced in v2.0

### Examples

```
# Example 1
one_in 2 # will return true with a probability of 1/2, false with probability 1/2

# Example 2
one_in 3 # will return true with a probability of 1/3, false with a probability of 2/3

# Example 3
one_in 100 # will return true with a probability of 1/100, false with a probability of 99/100
```

## Relative MIDI pitch to frequency ratio

`pitch_to_ratio pitch` (midi\_number)

Convert a midi note to a ratio which when applied to a frequency will scale the frequency by the number of semitones. Useful for changing the pitch of a sample by using it as a way of generating the rate.

Introduced in v2.5

### Examples

```
# Example 1
pitch_to_ratio 12 #=> 2.0

# Example 2
pitch_to_ratio 1 #=> 1.05946

# Example 3
pitch_to_ratio -12 #=> 0.5

# Example 4
sample :ambi_choir, rate: pitch_to_ratio(3) # Plays :ambi_choir 3 semitones above default.

# Example 5
(range 0, 16).each do |n|
  sample :ambi_choir, rate: pitch_to_ratio(n)
  sleep 0.5
end
# Play a chromatic scale of semitones
# For each note in the range 0->16
# play :ambi_choir at the relative pitch
# and wait between notes
```

## Play current synth

`play note` (symbol\_or\_number)

Play note with current synth. Accepts a set of standard options which include control of an amplitude envelope with `attack:`, `decay:`, `sustain:` and `release:` phases. These phases are triggered in order, so the duration of the sound is attack + decay + sustain + release times. The duration of the sound does not affect any other notes. Code continues executing whilst the sound is playing through its envelope phases.

Accepts optional args for modification of the synth being played. See each synth's documentation for synth-specific opts. See `use_synth` and `with_synth` for changing the current synth.

If note is `nil`, `:r` or `:rest`, play is ignored and treated as a rest.

Introduced in v2.0

### Options

<code>amp:</code>	The amplitude of the note
<code>amp_slide:</code>	The duration in beats for amplitude changes to take place
<code>pan:</code>	The stereo position of the sound. -1 is left, 0 is in the middle and 1 is on the right. You may use a value in between -1 and 1 such as 0.25
<code>pan_slide:</code>	The duration in beats for the pan value to change
<code>attack:</code>	Amount of time (in beats) for sound to reach full amplitude ( <code>attack_level</code> ). A short attack (i.e. 0.01) makes the initial part of the sound very percussive like a sharp tap. A longer attack (i.e 1) fades the sound in gently.
<code>decay:</code>	Amount of time (in beats) for the sound to move from full amplitude ( <code>attack_level</code> ) to the sustain amplitude ( <code>sustain_level</code> ).
<code>sustain:</code>	Amount of time (in beats) for sound to remain at sustain level amplitude. Longer sustain values result in longer sounds. Full length of sound is attack + decay + sustain + release.
<code>release:</code>	Amount of time (in beats) for sound to move from sustain level amplitude to silent. A short release (i.e. 0.01) makes the final part of the sound very percussive (potentially resulting in a click). A longer release (i.e 1) fades the sound out gently.
<code>attack_level:</code>	Amplitude level reached after attack phase and immediately before decay phase
<code>decay_level:</code>	Amplitude level reached after decay phase and immediately before sustain phase. Defaults to <code>sustain_level</code> unless explicitly set
<code>sustain_level:</code>	Amplitude level reached after decay phase and immediately before release phase.
<code>env_curve:</code>	Select the shape of the curve between levels in the envelope. 1=linear, 2=exponential, 3=sine, 4=welch, 6=squared, 7=cubed
<code>slide:</code>	Default slide time in beats for all slide opts. Individually specified slide opts will override this value

### Examples

# Example 1	
<code>play 50</code>	<code># Plays note 50 on the current synth</code>
# Example 2	
<code>play 50, attack: 1</code>	<code># Plays note 50 with a fade-in time of 1s</code>
# Example 3	
<code>play 62, pan: -1, release: 3</code>	<code># Play note 62 in the left ear with a fade-out time of 3s.</code>

## Play notes simultaneously

## play\_chord notes (list)

Play a list of notes at the same time.

Accepts optional args for modification of the synth being played. See each synth's documentation for synth-specific opts. See [use\\_synth](#) and [with\\_synth](#) for changing the current synth.

Introduced in v2.0

## Options

amp:	The amplitude of the note
amp_slide:	The duration in beats for amplitude changes to take place
pan:	The stereo position of the sound. -1 is left, 0 is in the middle and 1 is on the right. You may use a value in between -1 and 1 such as 0.25
pan_slide:	The duration in beats for the pan value to change
attack:	Amount of time (in beats) for sound to reach full amplitude (attack_level). A short attack (i.e. 0.01) makes the initial part of the sound very percussive like a sharp tap. A longer attack (i.e 1) fades the sound in gently.
decay:	Amount of time (in beats) for the sound to move from full amplitude (attack_level) to the sustain amplitude (sustain_level).
sustain:	Amount of time (in beats) for sound to remain at sustain level amplitude. Longer sustain values result in longer sounds. Full length of sound is attack + decay + sustain + release.
release:	Amount of time (in beats) for sound to move from sustain level amplitude to silent. A short release (i.e. 0.01) makes the final part of the sound very percussive (potentially resulting in a click). A longer release (i.e 1) fades the sound out gently.
attack_level:	Amplitude level reached after attack phase and immediately before decay phase
decay_level:	Amplitude level reached after decay phase and immediately before sustain phase. Defaults to sustain_level unless explicitly set
sustain_level:	Amplitude level reached after decay phase and immediately before release phase.
env_curve:	Select the shape of the curve between levels in the envelope. 1=linear, 2=exponential, 3=sine, 4=welch, 6=squared, 7=cubed
slide:	Default slide time in beats for all slide opts. Individually specified slide opts will override this value

## Examples

# Example 1	
<pre>play_chord [40, 45, 47]</pre>	# same as:
<pre>play 40 play 45 play 47</pre>	
# Example 2	
<pre>play_chord [40, 45, 47], amp: 0.5</pre>	# same as:
<pre>play 40, amp: 0.5 play 45, amp: 0.5 play 47, amp: 0.5</pre>	
# Example 3	
<pre>play_chord chord(:e3, :minor)</pre>	

## Play pattern of notes

## play\_pattern notes (list)

Play list of notes with the current synth one after another with a sleep of 1

Accepts optional args for modification of the synth being played. See each synth's documentation for synth-specific opts. See `use_synth` and `with_synth` for changing the current synth.

Introduced in v2.0

## Examples

# Example 1	
<code>play_pattern [40, 41, 42]</code>	<code># Same as: # play 40 # sleep 1 # play 41 # sleep 1 # play 42</code>
# Example 2	
<code>play_pattern [:d3, :c1, :Eb5]</code>	<code># You can use keyword notes</code>
# Example 3	
<code>play_pattern [:d3, :c1, :Eb5], amp: 0.5, cutoff: 90</code>	<code># Supports the same arguments as play:</code>

## Play pattern of notes with specific times

### play\_pattern\_timed notes (list), times (list\_or\_number)

Play each note in a list of notes one after another with specified times between them. The notes should be a list of MIDI numbers, symbols such as `:E4` or chords such as `chord(:A3, :major)` - identical to the first parameter of the play function. The times should be a list of times between the notes in beats.

If the list of times is smaller than the number of gaps between notes, the list is repeated again. If the list of times is longer than the number of gaps between notes, then some of the times are ignored. See examples for more detail.

Accepts optional args for modification of the synth being played. See each synth's documentation for synth-specific opts. See `use_synth` and `with_synth` for changing the current synth.

Introduced in v2.0

## Options

<code>amp:</code>	The amplitude of the note
<code>amp_slide:</code>	The duration in beats for amplitude changes to take place
<code>pan:</code>	The stereo position of the sound. -1 is left, 0 is in the middle and 1 is on the right. You may use a value in between -1 and 1 such as 0.25
<code>pan_slide:</code>	The duration in beats for the pan value to change
<code>attack:</code>	Amount of time (in beats) for sound to reach full amplitude ( <code>attack_level</code> ). A short attack (i.e. 0.01) makes the initial part of the sound very percussive like a sharp tap. A longer attack (i.e 1) fades the sound in gently.
<code>decay:</code>	Amount of time (in beats) for the sound to move from full amplitude ( <code>attack_level</code> ) to the sustain amplitude ( <code>sustain_level</code> ).
<code>sustain:</code>	Amount of time (in beats) for sound to remain at sustain level amplitude. Longer sustain values result in longer sounds. Full length of sound is <code>attack + decay + sustain + release</code> .
<code>release:</code>	Amount of time (in beats) for sound to move from sustain level amplitude to silent. A short release (i.e. 0.01) makes the final part of the sound very percussive (potentially resulting in a click). A longer release (i.e 1) fades the sound out gently.
<code>attack_level:</code>	Amplitude level reached after attack phase and immediately before decay phase
<code>decay_level:</code>	Amplitude level reached after decay phase and immediately before sustain phase. Defaults to <code>sustain_level</code> unless explicitly set
	Amplitude level reached after decay phase and immediately before release phase



<code>sustain_level:</code>	Amplitude level reached after decay phase and immediately before release phase.
<code>env_curve:</code>	Select the shape of the curve between levels in the envelope. 1=linear, 2=exponential, 3=sine, 4=welch, 6=squared, 7=cubed
<code>slide:</code>	Default slide time in beats for all slide opts. Individually specified slide opts will override this value

## Examples

### # Example 1

```

play_pattern_timed [40, 42, 44, 46], [1, 2, 3]
# same as:

play 40
sleep 1
play 42
sleep 2
play 44
sleep 3
play 46

```

### # Example 2

```

play_pattern_timed [40, 42, 44, 46, 49], [1, 0.5]
# same as:

play 40
sleep 1
play 42
sleep 0.5
play 44
sleep 1
play 46
sleep 0.5
play 49

```

### # Example 3

```

play_pattern_timed [40, 42, 44, 46], [0.5]
# same as:

play 40
sleep 0.5
play 42
sleep 0.5
play 44
sleep 0.5
play 46

```

### # Example 4

```

play_pattern_timed [40, 42, 44], [1, 2, 3, 4, 5]
#same as:

play 40
sleep 1
play 42
sleep 2
play 44

```

## Display a message in the output pane

```
print output (string)
```

Displays the information you specify as a string inside the output pane. This can be a number, symbol, or a string itself. Useful for

debugging. Synonym for `puts`.

Introduced in v2.0

## Examples

# Example 1	
<code>print "hello there"</code>	<code>#=&gt; will print the string "hello there" to the output pane</code>
# Example 2	
<code>print 5</code>	<code>#=&gt; will print the number 5 to the output pane</code>
# Example 3	
<code>print foo</code>	<code>#=&gt; will print the contents of foo to the output pane</code>

## Display a message in the output pane

`puts output` (string)

Displays the information you specify as a string inside the output pane. This can be a number, symbol, or a string itself. Useful for debugging. Synonym for `print`.

Introduced in v2.0

## Examples

# Example 1	
<code>print "hello there"</code>	<code>#=&gt; will print the string "hello there" to the output pane</code>
# Example 2	
<code>print 5</code>	<code>#=&gt; will print the number 5 to the output pane</code>
# Example 3	
<code>print foo</code>	<code>#=&gt; will print the contents of foo to the output pane</code>

## Quantise a value to resolution

`quantise n` (number), `step` (positive\_number)

Round value to the nearest multiple of step resolution.

Introduced in v2.1

## Examples

# Example 1	
<code>quantise(10, 1)</code>	<code># 10 is already a multiple of 1, so returns 10</code>
# Example 2	

<code>quantise(10, 1.1)</code>	<code># Returns 9.9 which is 1.1 * 9</code>
<code># Example 3</code>	
<code>quantise(13.3212, 0.1)</code>	<code># 13.3</code>
<code># Example 4</code>	
<code>quantise(13.3212, 0.2)</code>	<code># 13.4</code>
<code># Example 5</code>	
<code>quantise(13.3212, 0.3)</code>	<code># 13.2</code>
<code># Example 6</code>	
<code>quantise(13.3212, 0.5)</code>	<code># 13.5</code>

## Create a ramp vector

`ramp list` (array)

Create a new immutable ramp vector from args. Indexes always return first or last value if out of bounds.

Introduced in v2.6

### Examples

<code># Example 1</code>	
<code>(ramp 1, 2, 3)[0]</code>	<code>#=&gt; 1</code>
<code># Example 2</code>	
<code>(ramp 1, 2, 3)[1]</code>	<code>#=&gt; 2</code>
<code># Example 3</code>	
<code>(ramp 1, 2, 3)[2]</code>	<code>#=&gt; 3</code>
<code># Example 4</code>	
<code>(ramp 1, 2, 3)[3]</code>	<code>#=&gt; 3</code>
<code># Example 5</code>	
<code>(ramp 1, 2, 3)[1000]</code>	<code>#=&gt; 3</code>
<code># Example 6</code>	
<code>(ramp 1, 2, 3)[-1]</code>	<code>#=&gt; 1</code>

```
# Example 7
```

```
(ramp 1, 2, 3)[-1000] #=> 1
```

## Generate a random float below a value

`rand max` (number\_or\_range)

Given a max number, produces a float between 0 and the supplied max value. If max is a range, produces a float within the range. With no args returns a random value between 0 and 1.

Introduced in v2.0

### Example

```
# Example 1
```

```
print rand(0.5) #=> will print a number like 0.375030517578125 to the output pane
```

## Roll back random generator

`rand_back amount` (number)

Roll the random generator back essentially 'undoing' the last call to `rand`. You may specify an amount to roll back allowing you to skip back n calls to `rand`.

Introduced in v2.7

### Examples

```
# Example 1
```

```
puts rand # Basic rand stream rollback
# prints 0.75006103515625

rand_back # roll random stream back one
# the result of the next call to rand will be
# exactly the same as the previous call

puts rand # prints 0.75006103515625 again!
puts rand # prints 0.733917236328125
```

```
# Example 2
```

```
# Jumping back multiple places in the rand stream

puts rand # prints 0.75006103515625
puts rand # prints 0.733917236328125
puts rand # prints 0.464202880859375
puts rand # prints 0.24249267578125

rand_back(3) # roll random stream back three places
# the result of the next call to rand will be
# exactly the same as the result 3 calls to
# rand ago.

puts rand # prints 0.733917236328125 again!
puts rand # prints 0.464202880859375
```

## Generate a random whole number below a value (exclusive)

`rand_i max (number_or_range)`

Given a max number, produces a whole number between 0 and the supplied max value exclusively. If max is a range produces an int within the range. With no args returns either 0 or 1

Introduced in v2.0

### Example

```
# Example 1
print rand_i(5) #=> will print a either 0, 1, 2, 3, or 4 to the output pane
```

## Reset rand generator to last seed

`rand_reset ()`

Resets the random stream to the last specified seed. See `use_random_seed` for changing the seed.

Introduced in v2.7

### Example

```
# Example 1
puts rand      # prints 0.75006103515625
puts rand      # prints 0.733917236328125
puts rand      # prints 0.464202880859375
puts rand      # prints 0.24249267578125
rand_reset     # reset the random stream
puts rand      # prints 0.75006103515625
```

## Jump forward random generator

`rand_skip amount (number)`

Jump the random generator forward essentially skipping the next call to `rand`. You may specify an amount to jump allowing you to skip n calls to `rand`.

Introduced in v2.7

### Examples

```
# Example 1
puts rand      # Basic rand stream skip
               # prints 0.75006103515625
rand_skip      # jump random stream forward one
               # typically the next rand is 0.733917236328125
puts rand      # prints 0.464202880859375
```

```
# Example 2
puts rand      # Jumping forward multiple places in the rand stream
               # prints 0.75006103515625
puts rand      # prints 0.733917236328125
puts rand      # prints 0.464202880859375
puts rand      # prints 0.24249267578125
```

```

rand_reset      # reset the random stream
puts rand      # prints 0.75006103515625
rand_skip(2)   # jump random stream forward three places
               # the result of the next call to rand will be
               # exactly the same as if rand had been called
               # three times

puts rand 0.24249267578125

```

## Create a ring buffer with the specified start, finish and step size

`range start (number), finish (number), step_size (number)`

Create a new ring buffer from the range arguments (start, finish and step size). Step size defaults to **1**. Indexes wrap around positively and negatively

Introduced in v2.2

### Options

<code>step:</code>	Size of increment between steps; step size.
<code>inclusive:</code>	If set to true, range is inclusive of finish value

### Examples

# Example 1	
<code>(range 1, 5)</code>	<code>==&gt; (ring 1, 2, 3, 4)</code>
# Example 2	
<code>(range 1, 5, inclusive: true)</code>	<code>==&gt; (ring 1, 2, 3, 4, 5)</code>
# Example 3	
<code>(range 1, 5, step: 2)</code>	<code>==&gt; (ring 1, 3)</code>
# Example 4	
<code>(range 1, -5, step: 2)</code>	<code>==&gt; (ring 1, -1, -3)</code>
# Example 5	
<code>(range 1, -5, step: 2)[-1]</code>	<code>==&gt; -3</code>

## Relative frequency ratio to MIDI pitch

`ratio_to_pitch ratio (number)`

Convert a frequency ratio to a midi note which when added to a note will transpose the note to match the frequency ratio.

Introduced in v2.7

### Examples

```
# Example 1
```

```
ratio_to_pitch 2 #=> 12.0
```

```
# Example 2
```

```
ratio_to_pitch 0.5 #=> -12.0
```

## Random number in centred distribution

`rdist width (number), centre (number)`

Returns a random number within the range with width around centre. If optional arg `step:` is used, the result is quantised by step.

Introduced in v2.3

### Options

`step:` Step size of value to quantise to.

### Examples

```
# Example 1
```

```
print rdist(1, 0) #=> will print a number between -1 and 1
```

```
# Example 2
```

```
print rdist(1) #=> centre defaults to 0 so this is the same as rdist(1, 0)
```

```
# Example 3
```

```
loop do
  play :c3, pan: rdist(1) #=> Will play :c3 with random L/R panning
  sleep 0.125
end
```

## Determine if note or args is a rest

`rest? note_or_args (number_symbol_or_map)`

Given a note or an args map, returns true if it represents a rest and false if otherwise

Introduced in v2.1

### Examples

```
# Example 1
```

```
puts rest? nil # true
```

```
# Example 2
```

```
puts rest? :r # true
```

# Example 3	
puts rest? :rest	# true
# Example 4	
puts rest? 60	# false
# Example 5	
puts rest? {}	# false
# Example 6	
puts rest? {note: :rest}	# true
# Example 7	
puts rest? {note: nil}	# true
# Example 8	
puts rest? {note: 50}	# false

## Create a ring buffer

`ring list` (array)

Create a new immutable ring buffer from args. Indexes wrap around positively and negatively

Introduced in v2.2

### Examples

# Example 1	
(ring 1, 2, 3)[0]	#=> 1
# Example 2	
(ring 1, 2, 3)[1]	#=> 2
# Example 3	
(ring 1, 2, 3)[3]	#=> 1
# Example 4	
(ring 1, 2, 3)[-1]	#=> 3



## Generate a random float between two numbers

`rrand min (number), max (number)`

Given two numbers, this produces a float between the supplied min and max values exclusively. Both min and max need to be supplied. For random integers, see `rrand_i`. If optional arg `step`: is used, the result is quantised by step.

Introduced in v2.0

### Options

<code>step:</code>	Step size of value to quantise to.
--------------------	------------------------------------

### Examples

# Example 1

<pre>print rrand(0, 10)</pre>	#=> will print a number like 8.917730007820797 to the output pane
-------------------------------	---

# Example 2

<pre>loop do   play rrand(60, 72)   sleep 0.125 end</pre>	#=> Will play a random non-integer midi note between C4 (60) and C5 (72) such as 67.3453 or
---	---

## Generate a random whole number between two points inclusively

`rrand_i min (number), max (number)`

Given two numbers, this produces a whole number between the min and max you supplied inclusively. Both min and max need to be supplied. For random floats, see `rrand`

Introduced in v2.0

### Examples

# Example 1

<pre>print rrand_i(0, 10)</pre>	#=> will print a random number between 0 and 10 (e.g. 4, 0 or 10) to the output pane
---------------------------------	--

# Example 2

<pre>loop do   play rrand_i(60, 72)   sleep 0.125 end</pre>	#=> Will play a random midi note between C4 (60) and C5 (72)
---	--

## Real time conversion

`rt seconds (number)`

Real time representation. Returns the amount of beats for the value in real-time seconds. Useful for bypassing any bpm scaling

Introduced in v2.0

### Example

# Example 1

```

use_bpm 120      # modifies all time to be half
play 50         # actually sleeps for half of a second
sleep 1
play 62         # bypasses bpm scaling and sleeps for a second
sleep rt(1)
play 72

```

## Trigger sample

`sample name_or_path (symbol_or_string)`

This is the main method for playing back recorded sound files (samples). Sonic Pi comes with lots of great samples included (see the section under help) but you can also load and play `.wav`, `.wave`, `.aif` or `.aiff` files from anywhere on your computer too. The `rate:` parameter affects both the speed and the pitch of the playback. See the examples for details. Check out the `use_sample_pack` and `use_sample_pack_as` fns for details on making it easy to work with a whole folder of your own sample files. Note, that on the first trigger of a sample, Sonic Pi has to load the sample which takes some time and may cause timing issues. To preload the samples you wish to work with consider using `load_sample` or `load_samples`.

Introduced in v2.0

## Options

<code>rate:</code>	Rate with which to play back the sample. Higher rates mean an increase in pitch and a decrease in duration. Default is 1.
<code>beat_stretch:</code>	Stretch (or shrink) the sample to last for exactly the specified number of beats. Please note - this does <i>not</i> keep the pitch constant and is essentially the same as modifying the rate directly.
<code>pitch_stretch:</code>	Stretch (or shrink) the sample to last for exactly the specified number of beats. This attempts to keep the pitch constant using the <code>pitch:</code> opt. Note, it's very likely you'll need to experiment with the <code>window_size:</code> <code>pitch_dis:</code> and <code>time_dis:</code> opts depending on the sample and the amount you'd like to stretch/shrink from original size.
<code>attack:</code>	Time to reach full volume. Default is 0
<code>sustain:</code>	Time to stay at full volume. Default is to stretch to length of sample (minus attack and release times).
<code>release:</code>	Time (from the end of the sample) to go from full amplitude to 0. Default is 0
<code>start:</code>	Position in sample as a fraction between 0 and 1 to start playback. Default is 0.
<code>finish:</code>	Position in sample as a fraction between 0 and 1 to end playback. Default is 1.
<code>pan:</code>	Stereo position of audio. -1 is left ear only, 1 is right ear only, and values in between position the sound accordingly. Default is 0
<code>amp:</code>	Amplitude of playback
<code>norm:</code>	Normalise the audio (make quieter parts of the sample louder and louder parts quieter) - this is similar to the normaliser FX. This may emphasise any clicks caused by clipping.
<code>cutoff:</code>	Cutoff value of the built-in low pass filter (lpf) in MIDI notes. Unless specified, the lpf is <i>not</i> added to the signal chain.
<code>res:</code>	Cutoff-specific opt. Only honoured if <code>cutoff:</code> is specified. Filter resonance as a value between 0 and 1. Large amounts of resonance (a <code>res:</code> near 1) can create a whistling sound around the cutoff frequency. Smaller values produce less resonance.
<code>rpitch:</code>	Rate modified pitch. Multiplies the rate by the appropriate ratio to shift up or down the specified amount in MIDI notes. Please note - this does <i>not</i> keep the duration and rhythmical rate constant and is essentially the same as modifying the rate directly.
<code>pitch:</code>	Pitch adjustment in semitones. 1 is up a semitone, 12 is up an octave, -12 is down an octave etc. Maximum upper limit of 24 (up 2 octaves). Lower limit of -72 (down 6 octaves). Decimal numbers can be used for fine tuning.
<code>window_size:</code>	Pitch shift-specific opt - only honoured if the <code>pitch:</code> opt is used. Pitch shift works by chopping the input into tiny slices, then playing these slices at a higher or lower rate. If we make the slices small enough and overlap them, it sounds like the original sound with the pitch changed. The <code>window_size</code> is the length of the slices and is measured in seconds. It needs to be around 0.2 (200ms) or greater for pitched sounds like guitar or bass, and needs to be around 0.02 (20ms) or lower for percussive sounds like drum loops. You can experiment with this to get the best sound for your input.
<code>pitch_dis:</code>	Pitch shift-specific opt - only honoured if the <code>pitch:</code> opt is used. Pitch dispersion - how much random variation in pitch to add. Using a low value like 0.001 can help to "soften up" the metallic sounds, especially on drum loops. To be really technical, <code>pitch_dispersions</code> is the maximum random deviation of the pitch from the pitch ratio (which is set by the <code>pitch</code> param)
<code>time_dis:</code>	Pitch shift-specific opt - only honoured if the <code>pitch:</code> opt is used. Time dispersion - how much random delay before playing each grain (measured in seconds). Again, low values here like 0.001 can help to soften up metallic sounds introduced by the effect. Large values are also fun as they can make soundscapes and textures from the input, although you will most likely lose the rhythm of the original. NB - This won't have an effect if it's larger than <code>window_size</code> .
<code>slide:</code>	Default slide time in beats for all slide opts. Individually specified slide opts will override this value

## Examples

### # Example 1

```
sample :perc_bell # plays one of Sonic Pi's built in samples
```

### # Example 2

```
sample '/home/yourname/path/to/a/sample.wav' # plays a wav|wave|aif|aiff file from your local filesystem
```

### # Example 3

```
sample :loop_amen # Let's play with the rate parameter
sleep sample_duration(:loop_amen) # play one of the included samples

# this sleeps for exactly the length of the sample

# Setting a rate of 0.5 will cause the sample to
# a) play half as fast
# b) play an octave down in pitch
#
# Listen:

sample :loop_amen, rate: 0.5
sleep sample_duration(:loop_amen, rate: 0.5)

# Setting a really low number means the sample takes
# a very long time to finish! Also it sounds very
# different to the original sound

sample :loop_amen, rate: 0.05
sleep sample_duration(:loop_amen, rate: 0.05)
```

### # Example 4

```
sample :loop_amen, rate: -1 # Setting a really negative number can be lots of fun
sleep sample_duration(:loop_amen, rate: 1) # It plays the sample backwards!

# there's no need to give sample_duration a negative number to

sample :loop_amen, rate: -0.5 # Using a rate of -0.5 is just like using the positive 0.5
sleep sample_duration(:loop_amen, rate: 0.5) # (lower in pitch and slower) except backwards

# there's no need to give sample_duration a negative number to
```

### # Example 5

```
puts sample_duration(:loop_amen, rate: 0) # BE CAREFUL
# Don't set the rate to 0 though because it will get stuck
# and won't make any sound at all!
# We can see that the following would take Infinity seconds to
```

### # Example 6

```
s = sample :loop_amen_full, rate: 0.05 # Just like the play method, we can assign our sample player
sleep 1 # to a variable and control the rate parameter whilst it's pl
control(s, rate: 0.2) #
sleep 1 #
control(s, rate: 0.4) # The following example sounds a bit like a vinyl speeding up
sleep 1 # Note, this technique only works when you don't use envelope
control(s, rate: 0.6)
sleep 1
control(s, rate: 0.8)
sleep 1
control(s, rate: 1)
```

```
# Example 7
```

```
sample :loop_amen, start: 0.5, finish: 1
```

```
# Using the :start and :finish parameters you can play a section of a sample  
# The default start is 0 and the default finish is 1  
# play the last half of a sample
```

```
# Example 8
```

```
sample :loop_amen, start: 1, finish: 0.5
```

```
# You can also play part of any sample backwards by using a start  
# higher than the finish  
# play the last half backwards
```

```
# Example 9
```

```
sample {sample_name: :loop_amen, rate: 2}
```

```
# You can also specify the sample using a Hash with a `:sample_name`
```

```
# Example 10
```

```
sample lambda { [:loop_amen, :loop_garzul].choose }
```

```
# You can also specify the sample using a lambda that yields a sample  
# although you probably don't need a lambda for this in most cases
```

## Get sample data

`sample_buffer` `path` (string)

Alias for the `load_sample` method. Loads sample if necessary and returns buffer information.

Introduced in v2.0

### Example

```
# Example 1
```

```
see load_sample
```

## Get duration of sample in beats

`sample_duration` `path` (string)

Given the name of a loaded sample, or a path to a `.wav`, `.wave`, `.aif` or `.aiff` file returns the length of time in beats that the sample would play for. `sample_duration` understands and accounts for all the opts you can pass to `sample` which have an effect on the playback duration such as `rate:`. The time returned is scaled to the current bpm.

Introduced in v2.0

### Options

<code>rate:</code>	Rate modifier. For example, doubling the rate will halve the duration.
--------------------	--

<code>start:</code>	Start position of sample playback as a value from 0 to 1
---------------------	--

<code>finish:</code>	Finish position of sample playback as a value from 0 to 1
----------------------	---

<code>attack:</code>	Duration of the attack phase of the envelope.
----------------------	---

<code>decay:</code>	Duration of the decay phase of the envelope.
---------------------	--

<b>sustain:</b>	Duration of the sustain phase of the envelope.
<b>release:</b>	Duration of the release phase of the envelope.
<b>beat_stretch:</b>	Change the rate of the sample so that its new duration matches the specified number of beats.
<b>pitch_stretch:</b>	Change the rate of the sample so that its new duration matches the specified number of beats but attempt to preserve pitch.
<b>rpitch:</b>	Change the rate to shift the pitch up or down the specified number of MIDI notes.

## Examples

<b># Example 1</b>	
<pre>puts sample_duration(:loop_garzul)</pre>	<pre># Simple use # returns 8.0 because this sample is 8 second</pre>
<b># Example 2</b>	
<pre>use_bpm 120 puts sample_duration(:loop_garzul) use_bpm 90 puts sample_duration(:loop_garzul) use_bpm 21 puts sample_duration(:loop_garzul)</pre>	<pre># The result is scaled to the current BPM # =&gt; 16.0 # =&gt; 12.0 # =&gt; 2.8</pre>
<b># Example 3</b>	
<pre>live_loop :avoid_this do   with_fx :slicer do     sample :loop_amen     sleep sample_duration(:loop_amen)   end end  live_loop :prefer_this do   use_sample_bpm :loop_amen   with_fx :slicer do     sample :loop_amen     sleep 1   end end  live_loop :or_this do   with_fx :slicer do     sample :loop_amen, beat_stretch: 1     sleep 1   end end</pre>	<pre># Avoid using sample_duration to set the sleep time # It is possible to use sample_duration to do this # However, if you're using a rhythmical sample # in the same BPM as the current BPM, then the sleep time # badly out of sync. This is because the slicer # this live_loop is looping at a different BPM  # Instead prefer to set the BPM of the live_loop # two benefits. Now our sleep is a nice and simple # Also, our slicer now works with the beat and the sleep time  # Alternatively we can beat_stretch the sample # side effect of changing the rate of the sample # FX works nicely in time and the sleep time</pre>
<b># Example 4</b>	
<pre>sample_duration :loop_garzul, rate: 1  sample_duration :loop_garzul, rate: 0.5  sample_duration :loop_garzul, rate: 2  sample_duration :loop_garzul, rate: -2  sample_duration :loop_garzul, attack: 1 sample_duration :loop_garzul, attack: 100 sample_duration :loop_garzul, attack: 0</pre>	<pre># The standard sample opts are also honoured # Playing a sample at standard speed will return the sample duration # =&gt; 8.0  # Playing a sample at half speed will double the duration # =&gt; 16.0  # Playing a sample at double speed will halve the duration # =&gt; 4.0  # Playing a sample backwards at double speed will halve the duration # =&gt; 4.0  # Without an explicit sustain: opt attack: just returns the sample duration # =&gt; 8.0 # =&gt; 8.0 # =&gt; 8.0</pre>

```

sample_duration :loop_garzul, release: 1
sample_duration :loop_garzul, release: 100
sample_duration :loop_garzul, release: 0

sample_duration :loop_garzul, decay: 1
sample_duration :loop_garzul, decay: 100
sample_duration :loop_garzul, decay: 0

sample_duration :loop_garzul, sustain: 0, attack: 0.5
sample_duration :loop_garzul, sustain: 0, decay: 0.1
sample_duration :loop_garzul, sustain: 0, release: 1
sample_duration :loop_garzul, sustain: 2, attack: 0.5, release: 1

sample_duration :loop_garzul, sustain: 0, attack: 8, release: 3

sample_duration :loop_garzul, rate: 10
sample_duration :loop_garzul, sustain: 0, attack: 0.9, rate: 10

sample_duration :loop_garzul, rpitch: 12
sample_duration :loop_garzul, rpitch: -12

sample_duration :loop_garzul, rpitch: 12, rate: 2

sample_duration :loop_garzul, beat_stretch: 3
sample_duration :loop_garzul, beat_stretch: 3, rate: 0.5

sample_duration :loop_garzul, pitch_stretch: 3
sample_duration :loop_garzul, pitch_stretch: 3, rate: 0.5

sample_duration :loop_garzul, start: 0.5
sample_duration :loop_garzul, start: 0.5, finish: 0.75
sample_duration :loop_garzul, finish: 0.5, start: 0.75
sample_duration :loop_garzul, rate: 2, finish: 0.5, start: 0.75

```

```

# Without an explicit sustain: opt release: j
# => 8.0
# => 8.0
# => 8.0

# Without an explicit sustain: opt decay: jus
# => 8.0
# => 8.0
# => 8.0

# With an explicit sustain: opt, if the attac
# duration is less than the sample duration t
# sample time.
# => 0.5
# => 0.1
# => 1.0
# => 3.5

# If the envelope duration is longer than the
# sample duration
# => 8

# All other opts are taken into account before
# => 0.8
# => 0.8 (The duration of the sample is less

# The rpitch: opt will modify the rate to shi
# and therefore affects duration.
# => 4.0
# => 16

# The rpitch: and rate: opts combine together
# => 2.0

# The beat_stretch: opt stretches the sample
# It also combines with rate:
# => 3.0
# => 6.0

# The pitch_stretch: opt acts identically to
# duration.
# => 3.0
# => 6.0

# The start: and finish: opts can also shorte
# with other opts such as rate:
# => 4.0
# => 2.0
# => 2.0
# => 1.0

```

#### # Example 5

```

sample :loop_amen
sleep sample_duration(:loop_amen)
sample :loop_amen

# Triggering samples one after another
# start the :loop_amen sample
# wait for the duration of :loop_amen before
# starting it again

```

## Get all sample groups

### sample\_groups

Return a list of all the sample groups available

Introduced in v2.0

## Get sample information

sample\_info path (string)

Alias for the `load_sample` method. Loads sample if necessary and returns sample information.

Introduced in v2.0

## Example

```
# Example 1
see load_sample
```

## Test if sample was pre-loaded

`sample_loaded?` `path` (string)

Given a path to a `.wav`, `.wave`, `.aif` or `.aiff` file, returns `true` if the sample has already been loaded.

Introduced in v2.2

## Example

```
# Example 1
load_sample :elec_blip # :elec_blip is now loaded and ready to play as a sample
puts sample_loaded? :elec_blip # prints true because it has been pre-loaded
puts sample_loaded? :misc_burp # prints false because it has not been loaded
```

## Get sample names

`sample_names` `group` (symbol)

Return a list of sample names for the specified group

Introduced in v2.0

## Create scale

`scale` `tonic` (symbol), `name` (symbol)

Creates a ring of MIDI note numbers when given a tonic note and a scale type. Also takes an optional `num_octaves` parameter (octave 1 is the default)

Introduced in v2.0

## Options

`num_octaves`: The number of octaves you'd like the scale to consist of. More octaves means a larger scale. Default is 1.

## Examples

```
# Example 1
puts scale(:C, :major) # returns the list [60, 62, 64, 65, 67, 69, 71, 72]

# Example 2
play_pattern scale(:C, :major) # anywhere you can use a list of notes, you can also use scale

# Example 3
```

```
play_pattern(:C, :major, num_octaves: 2)
```

```
# you can use the :num_octaves parameter to get more notes
```

#### # Example 4

```
use bpm 300
play_pattern scale(:C, :diatonic)
play_pattern scale(:C, :ionian)
play_pattern scale(:C, :major)
play_pattern scale(:C, :dorian)
play_pattern scale(:C, :phrygian)
play_pattern scale(:C, :lydian)
play_pattern scale(:C, :mixolydian)
play_pattern scale(:C, :aeolian)
play_pattern scale(:C, :minor)
play_pattern scale(:C, :locrian)
play_pattern scale(:C, :hex_major6)
play_pattern scale(:C, :hex_dorian)
play_pattern scale(:C, :hex_phrygian)
play_pattern scale(:C, :hex_major7)
play_pattern scale(:C, :hex_sus)
play_pattern scale(:C, :hex_aeolian)
play_pattern scale(:C, :minor_pentatonic)
play_pattern scale(:C, :yu)
play_pattern scale(:C, :major_pentatonic)
play_pattern scale(:C, :gong)
play_pattern scale(:C, :egyptian)
play_pattern scale(:C, :shang)
play_pattern scale(:C, :jiao)
play_pattern scale(:C, :zhi)
play_pattern scale(:C, :ritusen)
play_pattern scale(:C, :whole_tone)
play_pattern scale(:C, :whole)
play_pattern scale(:C, :chromatic)
play_pattern scale(:C, :harmonic_minor)
play_pattern scale(:C, :melodic_minor_asc)
play_pattern scale(:C, :hungarian_minor)
play_pattern scale(:C, :octatonic)
play_pattern scale(:C, :messiaen1)
play_pattern scale(:C, :messiaen2)
play_pattern scale(:C, :messiaen3)
play_pattern scale(:C, :messiaen4)
play_pattern scale(:C, :messiaen5)
play_pattern scale(:C, :messiaen6)
play_pattern scale(:C, :messiaen7)
play_pattern scale(:C, :super_locrian)
play_pattern scale(:C, :hirajoshi)
play_pattern scale(:C, :kumoi)
play_pattern scale(:C, :neapolitan_major)
play_pattern scale(:C, :bartok)
play_pattern scale(:C, :bhairav)
play_pattern scale(:C, :locrian_major)
play_pattern scale(:C, :ahirbhairav)
play_pattern scale(:C, :enigmatic)
play_pattern scale(:C, :neapolitan_minor)
play_pattern scale(:C, :pelog)
play_pattern scale(:C, :augmented2)
play_pattern scale(:C, :scriabin)
play_pattern scale(:C, :harmonic_major)
play_pattern scale(:C, :melodic_minor_desc)
play_pattern scale(:C, :romanian_minor)
play_pattern scale(:C, :hindu)
play_pattern scale(:C, :iwato)
play_pattern scale(:C, :melodic_minor)
play_pattern scale(:C, :diminished2)
play_pattern scale(:C, :marva)
play_pattern scale(:C, :melodic_major)
play_pattern scale(:C, :indian)
play_pattern scale(:C, :spanish)
play_pattern scale(:C, :prometheus)
play_pattern scale(:C, :diminished)
play_pattern scale(:C, :todi)
play_pattern scale(:C, :leading_whole)
play_pattern scale(:C, :augmented)
play_pattern scale(:C, :purvi)
play_pattern scale(:C, :chinese)
play_pattern scale(:C, :lydian_minor)
```

```
# Sonic Pi supports a large range of scales.
# otherwise playing all these will take ages...
```



## All scale names

### scale\_names

Returns a ring containing all scale names known to Sonic Pi

Introduced in v2.6

### Example

```
# Example 1
puts scale_names #=> prints a list of all the scales
```

## Set control delta globally

### set\_control\_delta! time (number)

Specify how many seconds between successive modifications (i.e. trigger then controls) of a specific node on a specific thread. Set larger if you are missing control messages sent extremely close together in time.

Introduced in v2.1

### Example

```
# Example 1
set_control_delta! 0.1
# Set control delta to 0.1

s = play 70, release: 8, note_slide: 8
control s, note: 82
# Play a note and set the slide time
# immediately start sliding note.
# This control message might not be
# correctly handled as it is sent at the
# same virtual time as the trigger.
# If you don't hear a slide, try increasing the
# control delta until you do.
```

## Control master mixer

### set\_mixer\_control!

The master mixer is the final mixer that all sound passes through. This fn gives you control over the master mixer allowing you to manipulate all the sound playing through Sonic Pi at once. For example, you can sweep a lpf or hpf over the entire sound.

Introduced in v2.7

### Options

pre_amp:	Controls the amplitude of the signal prior to the FX stage of the mixer (prior to lpf/hpf stages). Has slide opts. Default 1.
amp:	Controls the amplitude of the signal after the FX tage. Has slide opts. Default 1.
hpf:	Global hpf FX. Has slide opts. Default 0.
lpf:	Global lpf FX. Has slide opts. Default 135.5.
hpf_bypass:	Bypass the global hpf. 0=no bypass, 1=bypass. Default 0.
lpf_bypass:	Bypass the global lpf. 0=no bypass, 1=bypass. Default 0.
limiter_bypass:	Bypass the final limiter. 0=no bypass, 1=bypass. Default 0.
leak_dc_bypass:	Bypass the final DC leak correction FX. 0=no bypass, 1=bypass. Default 0.

## Example

```
# Example 1
set_mixer_control! lpf: 30, lpf_slide: 16 # slide the global lpf to 30 over 16 beats.
```

## Set sched ahead time globally

`set_sched_ahead_time!` `time` (number)

Specify how many seconds ahead of time the synths should be triggered. This represents the amount of time between pressing 'Run' and hearing audio. A larger time gives the system more room to work with and can reduce performance issues in playing fast sections on slower platforms. However, a larger time also increases latency between modifying code and hearing the result whilst live coding.

Introduced in v2.0

## Example

```
# Example 1
set_sched_ahead_time! 1 # Code will now run approximately 1 second ahead of audio.
```

## Set Volume globally

`set_volume!` `vol` (number)

Set the main system volume to `vol`. Accepts a value between 0 and 5 inclusive. Vols greater or smaller than the allowed values are trimmed to keep them within range. Default is 1.

Introduced in v2.0

## Examples

```
# Example 1
set_volume! 2 # Set the main system volume to 2

# Example 2
set_volume! -1 # Out of range, so sets main system volume to 0

# Example 3
set_volume! 7 # Out of range, so sets main system volume to 5
```

## Randomise order of a list

`shuffle` `list` (array)

Returns a new list with the same elements as the original but with their order shuffled. Also works for strings

Introduced in v2.1

## Examples

```
# Example 1
```

```
shuffle [1, 2, 3, 4] #=> Would return something like: [3, 4, 2, 1]
```

```
# Example 2
```

```
shuffle "foobar" #=> Would return something like: "roobfa"
```

## Wait for duration

`sleep beats` (number)

Wait for a number of beats before triggering the next command. Beats are converted to seconds by scaling to the current bpm setting.

Introduced in v2.0

### Examples

```
# Example 1
```

```
play 50
play 55
play 62

sleep 1

play 50
sleep 0.5
play 55
sleep 0.5
play 62
```

# Without calls to sleep, all sounds would happen at once:  
# This is actually a chord with all notes played simultaneously  
  
# Create a gap, to allow a moment's pause for reflection...  
  
# Let's try the chord again, but this time with sleeps:  
# With the sleeps, we turn a chord into an arpeggio

```
# Example 2
```

```
use_bpm 120
play 50
sleep 1
play 55
sleep 1
play 62

use_bpm 30
play 50
sleep 1
play 55
sleep 1
play 62
```

# The amount of time sleep pauses for is scaled to match the current bpm. The default bpm is 60. Let  
# This actually sleeps for 0.5 seconds as we're now at double speed  
  
# Let's go down to half speed:  
# This now sleeps for 2 seconds as we're now at half speed.



## Print a string representing a list of numeric values as a spark graph/bar chart

`spark`

Given a list of numeric values, this method turns them into a string of bar heights and prints them out. Useful for quickly graphing the shape of an array.

Introduced in v2.5

### Examples

# Example 1	
spark (range 1, 5))	#=> 
# Example 2	
spark (range 1, 5).shuffle)	#=> 



## Returns a string representing a list of numeric values as a spark graph/bar chart

### spark\_graph

Given a list of numeric values, this method turns them into a string of bar heights. Useful for quickly graphing the shape of an array. Remember to use puts so you can see the output. See [spark](#) for a simple way of printing a spark graph.

Introduced in v2.5

## Examples

# Example 1	
puts (spark_graph (range 1, 5))	#=> 
# Example 2	
puts (spark_graph (range 1, 5).shuffle)	#=> 

## Euclidean distribution for beats

spread num\_accents (number), size (number)

Creates a new ring of boolean values which space a given number of accents as evenly as possible throughout a bar. This is an implementation of the process described in 'The Euclidean Algorithm Generates Traditional Musical Rhythms' (Toussaint 2005).

Introduced in v2.4

## Options

rotate:	rotate to the next strong beat allowing for easy permutations of the original rhythmic grouping (see example)
---------	---

## Examples

# Example 1	
(spread 3, 8)	#=> (ring true, false, false, true, false, false, true, fa
# Example 2	
(spread 3, 8, rotate: 1)	#=> (ring true, false, false, true, false, true, false, fa
# Example 3	
	# Easily create interesting polyrhythmic beats

```
live_loop :euclid_beat do
  sample :elec_bong, amp: 1.5 if (spread 3, 8).tick
  sample :perc_snap, amp: 0.8 if (spread 7, 11).look
  sample :bd_haus, amp: 2 if (spread 1, 4).look
  sleep 0.125
end
```

```
# Spread 3 bongs over 8
# Spread 7 snaps over 11
# Spread 1 bd over 4
```

#### # Example 4

```
(spread 2, 5) # Spread descriptions from
              # 'The Euclidean Algorithm Generates Traditional Musical Rhythms'
              # A thirteenth century Persian rhythm called Khafif-e-ramal

(spread 3, 4) # The archetypal pattern of the Cumbria from Columbia, as well as
              # as a Calypso rhythm from Trinidad

(spread 3, 5) # When started on the second onset, is another thirteenth
              # century Persian rhythm by the name of Khafif-e-ramal, as well as
              # as a Romanian folk-dance rhythm.

(spread 3, 7) # A ruchenitza rhythm used in a Bulgarian folk-dance.

(spread 3, 8) # The Cuban tresillo pattern

(spread 4, 7) # Another Ruchenitza Bulgarian folk-dance rhythm

(spread 4, 9) # The Aksak rhythm of Turkey.

(spread 4, 11) # The metric pattern used by Frank Zappa in his piece Outsider

(spread 5, 6) # Yields the York-Samai pattern, a popular Arab rhythm, which
              # started on the second onset.

(spread 5, 7) # The Nawakhat pattern, another popular Arab rhythm.

(spread 5, 8) # The Cuban cinquillo pattern.

(spread 5, 9) # A popular Arab rhythm called Agsag-Samai.

(spread 5, 11) # The metric pattern used by Moussorgsky in Pictures at an
               # Exhibition

(spread 5, 12) # The Venda clapping pattern of a South African children's
               # song.

(spread 5, 16) # The Bossa-Nova rhythm necklace of Brazil.

(spread 7, 8) # A typical rhythm played on the Bendir (frame drum)

(spread 7, 12) # A common West African bell pattern.

(spread 7, 16) # A Samba rhythm necklace from Brazil.

(spread 9, 16) # A rhythm necklace used in the Central African Republic.

(spread 11, 24) # A rhythm necklace of the Aka Pygmies of Central Africa.

(spread 13, 24) # Another rhythm necklace of the Aka Pygmies of the upper
                # Sangha.
```

## Get server status

### status

This returns a Hash of information about the synthesis environment. Mostly used for debugging purposes.

Introduced in v2.0

### Example

#### # Example 1

```
puts status # Returns something similar to:
           # {
           #   :ugens=>10,
           #   :synths=>1,
           #   :groups=>7,
```

```
# :sdefs=>61,
# :avg_cpu=>0.20156468451023102,
# :peak_cpu=>0.36655542254447937,
# :nom_samp_rate=>44100.0,
# :act_samp_rate=>44099.9998411752,
# :audio_busses=>2,
# :control_busses=>0
# }
```

## Stop current thread or run

### stop

Stops the current thread or if not in a thread, stops the current run. Does not stop any running synths triggered previously in the run/thread or kill any existing sub-threads.

Introduced in v2.5

### Examples

#### # Example 1

```
sample :loop_amen      #=> this sample is played until completion
  sleep 0.5
  stop                 #=> signal to stop executing this run
  sample :loop_garzul  #=> this never executes
```

#### # Example 2

```
in_thread do
  play 60           #=> this note plays
  stop
  sleep 0.5        #=> this sleep never happens
  play 72          #=> this play never happens
end

play 80           #=> this plays as the stop only affected the above thread
```

#### # Example 3

```
live_loop :foo
  sample :bd_haus
  sleep 1
  stop
end

live_loop :bar
  sample :elec_blip
  sleep 0.25
end

# Stopping live loops
# live loop :foo will now stop and no longer loop
# live loop :bar will continue looping
```

## Stretch a sequence of values

`stretch list (anything), count (number)`

Stretches a list of values each value repeated count times. Always returns a ring regardless of the type of the list that is stretched. To preserve type, consider using `.stretch` i.e. `(ramp 1, 2, 3).stretch(2) #=> (ramp 1, 1, 2, 2, 3, 3)`

Introduced in v2.6

### Examples

#### # Example 1

```
(stretch [1,2], 3) #=> (ring 1, 1, 1, 2, 2, 2)
```

## # Example 2

```
(stretch [:e2, :c3], 1, [:c2, :d3], 2) #=> (ring :e2, :c3, :c2, :c2, :d3, :d3)
```

# Sync with other threads

`sync cue_id` (symbol)

Pause/block the current thread until a `cue` heartbeat with a matching `cue_id` is received. When a matching `cue` message is received, unblock the current thread, and continue execution with the virtual time set to match the thread that sent the `cue` heartbeat. The current thread is therefore synced to the `cue` thread. If multiple cue ids are passed as arguments, it will `sync` on the first matching `cue_id`. By default the BPM of the cueing thread is inherited. This can be disabled using the `bpm_sync`: opt.

Introduced in v2.0

## Options

`bpm_sync`: Inherit the BPM of the cueing thread. Default is false

## Examples

### # Example 1

```
in_thread do
  sync :foo
  sample :ambi_lunar_land
end

sleep 5

cue :foo
```

# this parks the current thread waiting for a foo sync message to be received

# We send a sync message from the main thread.  
# This then unblocks the thread above and we then hear the sample

### # Example 2

```
in_thread do
  loop do
    cue :tick
    sleep 0.5
  end
end

loop do
  sync :tick
  sample :drum_heavy_kick
end
```

# Start a metronome thread  
# Loop forever:  
# sending tick heartbeat messages  
# and sleeping for 0.5 beats between ticks

# We can now play sounds using the metronome.  
# In the main thread, just loop  
# waiting for :tick sync messages  
# after which play the drum kick sample

### # Example 3

```
sync :foo, :bar
```

# Wait for either a :foo or :bar cue

### # Example 4

```
in_thread do
  loop do
    cue [:foo, :bar, :baz].choose
    sleep 0.5
  end
end
```

# Start a metronome thread  
# Loop forever:  
# sending one of three tick heartbeat messages randomly  
# and sleeping for 0.5 beats between ticks

```

# We can now play sounds using the metronome:

in_thread do
  loop do
    sync :foo
    sample :elec_beep
  end
end

in_thread do
  loop do
    sync :bar
    sample :elec_flip
  end
end

in_thread do
  loop do
    sync :baz
    sample :elec_blup
  end
end

```

## Trigger specific synth

`synth synth_name (symbol)`

Trigger specified synth with given arguments. Bypasses current synth value, yet still honours synth defaults.

Introduced in v2.0

### Options

**slide:** Default slide time in beats for all slide opts. Individually specified slide opts will override this value

### Examples

# Example 1

```
synth :fm, note: 60, amp: 0.5 # Play note 60 of the :fm synth with an amplitude of 0.5
```

# Example 2

```
use_synth_defaults release: 5
synth :dsaw, note: 50 # Play note 50 of the :dsaw synth with a release of 5
```

## Increment a tick and return value

`tick value (number)`

Increment the default tick by 1 and return value. Successive calls to `tick` will continue to increment the default tick. If a `key` is specified, increment that specific tick. If an increment `value` is specified, increment key by that value rather than 1. Ticks are `in_thread` and `live_loop` local, so incrementing a tick only affects the current thread's version of that tick. See `tick_reset` and `tick_set` for directly manipulating the tick vals.

Introduced in v2.6

### Options

**step:** The amount to tick up by. Default is 1.

**offset:** Offset to add to index returned. Useful when calling tick on lists, rings and vectors to offset the returned value. Default is 0.

### Examples



#### # Example 1

```
puts tick
puts tick
puts tick
puts tick
```

```
#=> 0
#=> 1
#=> 2
#=> 3
```

#### # Example 2

```
puts tick(:foo)
puts tick(:foo)
puts tick(:foo)
puts tick(:bar)
```

```
#=> 0 # named ticks have their own counts
#=> 1
#=> 2
#=> 0 # tick :bar is independent of tick :foo
```

#### # Example 3

```
live_loop :fast_tick do
  puts tick
  sleep 2
end

live_loop :slow_tick do
  puts tick
  sleep 4
end

end
```

```
# Each_live loop has its own separate ticks

# the fast_tick live_loop's tick will
# be updated every 2 seconds

# the slow_tick live_loop's tick is
# totally independent from the fast_tick
# live loop and will be updated every 4
# seconds
```

#### # Example 4

```
live_loop :regular_tick do
  puts tick
  sleep 1
end

live_loop :random_reset_tick do
  if one_in 3
    tick_reset
    puts "reset tick!"
  end
  puts tick
  sleep 1
end

end
```

```
# the regular_tick live_loop's tick will
# be updated every second

# randomly reset tick

# this live_loop's tick is totally
# independent and the reset only affects
# this tick.
```

#### # Example 5

```
live_loop :scale do
  play [:c, :d, :e, :f, :g].tick
  sleep 1
end
```

```
# Ticks work directly on lists, and will tick through each element
# However, once they get to the end, they'll return nil

# play all notes just once, then rests
```

#### # Example 6

```
live_loop :odd_scale do
  tick
  play [:c, :d, :e, :f, :g, :a].tick

  sleep 1
end
```

```
# Normal ticks interact directly with list ticks

# Increment the default tick
# this now play every *other* note just once,
# then rests
```

#### # Example 7

```
live_loop :looped_scale do
  play (ring :c, :d, :e, :f, :g).tick
  sleep 1
end
```

```
# Ticks work wonderfully with rings
# as the ring ensures the tick wraps
# round internally always returning a
# value

# play all notes just once, then repeats
```

# Example 8

```
live_loop :looped_scale do
  play (scale :e3, :minor_pentatonic).tick
  sleep 0.25
end
```

```
# Ticks work wonderfully with scales
# which are also rings

# play all notes just once, then repeats
```

## Reset tick to 0

### tick\_reset

Reset default tick to 0. If a **key** is referenced, set that tick to 0 instead. Same as calling tick\_set(0)

Introduced in v2.6

### Examples

# Example 1

```
tick
tick
tick
puts look
tick_set 0
puts look

# increment default tick a few times

#=> 2 (default tick is now 2)
# default tick is now 0
#=> 0 (default tick is now 0)
```

# Example 2

```
tick :foo
tick :foo
tick :foo
puts look(:foo)
tick_set 0
puts look(:foo)
tick_set :foo, 0
puts look(:foo)

# increment tick :foo a few times

#=> 2 (tick :foo is now 2)
# default tick is now 0
#=> 2 (tick :foo is still 2)
# reset tick :foo
#=> 0 (tick :foo is now 0)
```

## Reset all ticks

### tick\_reset\_all value (number)

Reset all ticks - default and keyed

Introduced in v2.6

### Example

# Example 1

```
tick # increment default tick and tick :foo
```

```

tick
tick :foo
tick :foo
tick :foo
puts look           #=> 1
puts look(:foo)    #=> 2
tick_reset_all
puts look           #=> 0
puts look(:foo)    #=> 0

```

## Set tick to a specific value

`tick_set value` (number)

Set the default tick to the specified `value`. If a `key` is referenced, set that tick to `value` instead. Next call to `look` will return `value`.

Introduced in v2.6

### Examples

# Example 1

```

tick_set 40
puts look           # set default tick to 40
                    #=> 40

```

# Example 2

```

tick_set :foo, 40
puts look(:foo)    # set tick :foo to 40
                    #=> 40 (tick :foo is now 40)
puts look           #=> 0 (default tick is unaffected)

```

## Block level comment ignoring

`uncomment`

Evaluates all of the code within the block. Use to reverse the effect of the comment without having to explicitly remove it.

Introduced in v2.0

### Example

# Example 1

```

uncomment do
  play 50
  sleep 1
  play 62
end
# starting a block level comment:
# played
# sleep happens
# played

```

## Enable and disable BPM scaling

`use_arg_bpm_scaling bool` (boolean)

Turn synth argument bpm scaling on or off for the current thread. This is on by default. Note, using `rt` for args will result in incorrect times when used after turning arg bpm scaling off.

Introduced in v2.0

### Examples

# Example 1

<pre>use_bpm 120 play 50, release: 2 sleep 2 use_arg_bpm_scaling false play 50, release: 2 sleep 2</pre>	<pre># release is actually 1 due to bpm scaling # actually sleeps for 1 second  # release is now 2 # still sleeps for 1 second</pre>
--	--

#### # Example 2

<pre>use_bpm 120 play 50, release: rt(2) sleep rt(2) use_arg_bpm_scaling false play 50, release: rt(2) sleep rt(2)</pre>	<pre># Interaction with rt # release is 2 seconds # sleeps for 2 seconds  # ** Warning: release is NOT 2 seconds! ** # still sleeps for 2 seconds</pre>
--	---

## Enable and disable arg checks

`use_arg_checks true_or_false` (boolean)

When triggering synths, each argument is checked to see if it is sensible. When argument checking is enabled and an argument isn't sensible, you'll see an error in the debug pane. This setting allows you to explicitly enable and disable the checking mechanism. See `with_arg_checks` for enabling/disabling argument checking only for a specific `do/end` block.

Introduced in v2.0

### Example

#### # Example 1

<pre>play 50, release: 5 use_arg_checks false play 50, release: 5</pre>	<pre># Args are checked # Args are not checked</pre>
---	--

## Set the tempo

`use_bpm bpm` (number)

Sets the tempo in bpm (beats per minute) for everything afterwards. Affects all subsequent calls to `sleep` and all temporal synth arguments which will be scaled to match the new bpm. If you wish to bypass scaling in calls to `sleep`, see the fn `rt`. Also, if you wish to bypass time scaling in synth args see `use_arg_bpm_scaling`. See also `with_bpm` for a block scoped version of `use_bpm`.

For dance music here's a rough guide for which BPM to aim for depending on your genre:

- Dub: 60-90 bpm
- Hip-hop: 60-100 bpm
- Downtempo: 90-120 bpm
- House: 115-130 bpm
- Techno/trance: 120-140 bpm
- Dubstep: 135-145 bpm
- Drum and bass: 160-180 bpm

Introduced in v2.0

### Example

#### # Example 1

<pre>4.times do   play 50, attack: 0.5, release: 0.25   sleep 1 end  sleep 2</pre>	<pre># default tempo is 60 bpm # attack is 0.5s and release is 0.25s # sleep for 1 second  # sleep for 2 seconds</pre>
--	--

```

use_bpm 120
4.times do
  play 62, attack: 0.5, release: 0.25
  sleep 1
end

sleep 2

use_bpm 240
8.times do
  play 62, attack: 0.5, release: 0.25
  sleep 1
end

```

```

# Let's make it go faster...
# double the bpm

# attack is scaled to 0.25s and release is now 0.125s
# actually sleeps for 0.5 seconds

# sleep for 1 second

# Let's make it go even faster...
# bpm is 4x original speed!

# attack is scaled to 0.125s and release is now 0.0625s
# actually sleeps for 0.25 seconds

```

## Set new tempo as a multiple of current tempo

`use_bpm_mul mul (number)`

Sets the tempo in bpm (beats per minute) as a multiplication of the current tempo. Affects all containing calls to `sleep` and all temporal synth arguments which will be scaled to match the new bpm. See also `use_bpm`

Introduced in v2.3

### Example

```

# Example 1

use_bpm 60
play 50
sleep 1
play 62
sleep 2
use_bpm_mul 0.5
play 50
sleep 1
play 62

# Set the BPM to 60
# Sleeps for 1 seconds
# Sleeps for 2 seconds
# BPM is now (60 * 0.5) == 30
# Sleeps for 2 seconds

```

## Enable and disable cue logging

`use_cue_logging true_or_false (boolean)`

Enable or disable log messages created on cues. This does not disable the cues themselves, it just stops them from being printed to the log

Introduced in v2.6

### Examples

```

# Example 1

use_cue_logging true # Turn on cue messages

# Example 2

use_cue_logging false # Disable cue messages

```

## Enable and disable debug

`use_debug true_or_false (boolean)`

Enable or disable messages created on synth triggers. If this is set to false, the synths will be silent until debug is turned back on. Silencing debug messages can reduce output noise and also increase performance on slower platforms. See `with_debug` for setting the debug value only for a specific `do/end` block.

Introduced in v2.0

## Examples

```
# Example 1
use_debug true # Turn on debug messages

# Example 2
use_debug false # Disable debug messages
```

## Merge synth defaults

### `use_merged_synth_defaults`

Specify synth arg values to be used by any following call to play. Merges the specified values with any previous defaults, rather than replacing them.

Introduced in v2.0

## Examples

```
# Example 1
play 50 #=> Plays note 50
use_merged_synth_defaults amp: 0.5
play 50 #=> Plays note 50 with amp 0.5
use_merged_synth_defaults cutoff: 80
play 50 #=> Plays note 50 with amp 0.5 and cutoff 80
use_merged_synth_defaults amp: 0.7
play 50 #=> Plays note 50 with amp 0.7 and cutoff 80

# Example 2
use_synth_defaults amp: 0.5, cutoff: 80, pan: -1
use_merged_synth_defaults amp: 0.7
play 50 #=> Plays note 50 with amp 0.7, cutoff 80 and pan -1
```

## Set random seed generator to known seed

### `use_random_seed seed` (number)

Resets the random number generator to the specified seed. All subsequently generated random numbers and randomisation functions such as `shuffle` and `choose` will use this new generator and the current generator is discarded. Use this to change the sequence of random numbers in your piece in a way that can be reproduced. Especially useful if combined with iteration. See examples.

Introduced in v2.0

## Examples

```
# Example 1
use_random_seed 1 # Basic usage
                  # reset random seed to 1
```

```
puts rand
use_random_seed 1
puts rand
```

```
# => 0.417022004702574
# reset random seed back to 1
#=> 0.417022004702574
```

## # Example 2

```
notes = (scale :eb3, :minor_pentatonic)
# Generating melodies
# Create a set of notes to choose from.
# Scales work well for this

with_fx :reverb do
  live_loop :repeating_melody do
    use_random_seed 300
    # Create a live loop
    # Set the random seed to a known value every
    # time around the loop. This seed is the key
    # to our melody. Try changing the number to
    # something else. Different numbers produce
    # different melodies

    8.times do
      # Now iterate a number of times. The size of
      # the iteration will be the length of the
      # repeating melody.

      play notes.choose, release: 0.1
      # 'Randomly' choose a note from our ring of
      # notes. See how this isn't actually random
      # but uses a reproducible method! These notes
      # are therefore repeated over and over...

      sleep 0.125
    end
  end
end
```

## Sample-duration-based bpm modification

`use_sample_bpm` *string\_or\_number* (sample\_name\_or\_duration)

Modify bpm so that sleeping for 1 will sleep for the duration of the sample.

Introduced in v2.1

### Options

`num_beats`: The number of beats within the sample. By default this is 1.

### Examples

#### # Example 1

```
use_sample_bpm :loop_amen
#Set bpm based on :loop_amen duration

live_loop :dnb do
  sample :bass_dnb_f
  sample :loop_amen
  sleep 1
end
#`sleep`ing for 1 actually sleeps for duration of :loop_amen
```

#### # Example 2

```
use_sample_bpm :loop_amen, num_beats: 4
# Set bpm based on :loop_amen duration
# but also specify that the sample duration
# is actually 4 beats long.

live_loop :dnb do
  sample :bass_dnb_f
  sample :loop_amen
  sleep 4
end
#`sleep`ing for 4 actually sleeps for duration of :loop_amen
# as we specified that the sample consisted of
# 4 beats
```

## Use new sample defaults

### use\_sample\_defaults

Specify new default values to be used by all subsequent calls to `sample`. Will remove and override any previous defaults.

Introduced in v2.5

### Example

```
# Example 1
sample :loop_amen # plays amen break with default arguments
use_sample_defaults amp: 0.5, cutoff: 70
sample :loop_amen # plays amen break with an amp of 0.5, cutoff of 70 and defaults for rest
use_sample_defaults cutoff: 90
sample :loop_amen # plays amen break with a cutoff of 90 and defaults for rest of args - no
```

## Use sample pack

### use\_sample\_pack pack\_path (string)

Given a path to a folder of samples on your filesystem, this method makes any `.wav`, `.wave`, `.aif` or `.aiff` files in that folder available as samples. Consider using `use_sample_pack_as` when using multiple sample packs. Use `use_sample_pack :default` To revert back to the default built-in samples.

Introduced in v2.0

### Example

```
# Example 1
use_sample_pack '/home/yourname/path/to/sample/dir'
sample :foo #=> plays /home/yourname/path/to/sample/dir/foo.{wav|wave|aif|aiff}
# where {wav|wave|aif|aiff} means one of wav, wave aif or aiff
sample :bd_haus #=> will not work unless there's a sample in '/home/yourname/path/to/sample/dir'
# called bd_haus.{wav|wave|aif|aiff}
use_sample_pack :default
sample :bd_haus #=> will play the built-in bd_haus.wav sample
```

## Use sample pack alias

### use\_sample\_pack\_as path (string), alias (string)

Similar to `use_sample_pack` except you can assign prefix aliases for samples. This lets you 'namespace' your sounds so that they don't clash, even if they have the same filename.

Introduced in v2.0

### Example

```
# Example 1
use_sample_pack_as '/home/yourname/my/cool/samples/guitar', :my_guitars
use_sample_pack_as '/home/yourname/my/cool/samples/drums', :my_drums
sample :my_guitars__bass # let's say you have two folders of your c
# and they both contain a file named 'bass.wav'
# You can now play both the 'bass.wav' sample
#=> plays '/home/yourname/my/cool/samples/guitar/bass.wav'
```



```
sample :my_drums__bass
```

```
#=> plays '/home/yourname/my/cool/samples/
```

## Switch current synth

`use_synth synth_name` (symbol)

Switch the current synth to `synth_name`. Affects all further calls to `play`. See `with_synth` for changing the current synth only for a specific `do/end` block.

Introduced in v2.0

### Example

```
# Example 1
```

```
play 50           # Plays with default synth
use_synth :mod_sine
play 50           # Plays with mod_sine synth
```

## Use new synth defaults

`use_synth_defaults`

Specify new default values to be used by all subsequent calls to `play`. Will remove and override any previous defaults.

Introduced in v2.0

### Example

```
# Example 1
```

```
play 50           # plays note 50 with default arguments
use_synth_defaults amp: 0.5, cutoff: 70
play 50           # plays note 50 with an amp of 0.5, cutoff of 70 and defaults for rest of
use_synth_defaults cutoff: 90
play 50           # plays note 50 with a cutoff of 90 and defaults for rest of args - note 1
```

## Note transposition

`use_transpose note_shift` (number)

Transposes your music by shifting all notes played by the specified amount. To shift up by a semitone use a transpose of 1. To shift down use negative numbers. See `with_transpose` for setting the transpose value only for a specific `do/end` block.

Introduced in v2.0

### Examples

```
# Example 1
```

```
play 50           # Plays note 50
use_transpose 1
play 50           # Plays note 51
```

```
# Example 2
```

```
# You may change the transposition multiple times:
```

```
play 62      # Plays note 62
use_transpose -12
play 62      # Plays note 50
use_transpose 3
play 62      # Plays note 65
```

## Use alternative tuning systems

`use_tuning tuning (symbol), fundamental_note (symbol_or_number)`

In most music we make semitones by dividing the octave into 12 equal parts, which is known as equal temperament. However there are lots of other ways to tune the 12 notes. This method adjusts each midi note into the specified tuning system. Because the ratios between notes aren't always equal, be careful to pick a centre note that is in the key of the music you're making for the best sound. Currently available tunings are :just, :pythagorean, :meantone and the default of :equal

Introduced in v2.6

### Examples

```
# Example 1
play :e4      # Plays note 64
use_tuning :just, :c
play :e4      # Plays note 63.8631
               # transparently changes midi notes too
play 64       # Plays note 63.8631
```

```
# Example 2
play 64       # You may change the tuning multiple times:
               # Plays note 64
use_tuning :just
play 64       # Plays note 63.8631
use_tuning :equal
play 64       # Plays note 64
```

## Create a vector

`vector list (array)`

Create a new immutable vector from args. Out of range indexes return nil.

Introduced in v2.6

### Examples

```
# Example 1
(vector 1, 2, 3)[0]  #=> 1
```

```
# Example 2
(vector 1, 2, 3)[1]  #=> 2
```

```
# Example 3
(vector 1, 2, 3)[2]  #=> 3
```

```
# Example 4
(vector 1, 2, 3)[3]  #=> nil
```

```
# Example 5
```

```
(vector 1, 2, 3)[1000] #=> nil
```

```
# Example 6
```

```
(vector 1, 2, 3)[-1] #=> nil
```

```
# Example 7
```

```
(vector 1, 2, 3)[-1000] #=> nil
```

## Get current version information

`version`

Return information representing the current version of Sonic Pi. This information may be further inspected with `version.major`, `version.minor`, `version.patch` and `version.dev`

Introduced in v2.0

### Examples

```
# Example 1
```

```
puts version # => Prints out the current version such as v2.0.1
```

```
# Example 2
```

```
puts version.major # => Prints out the major version number such as 2
```

```
# Example 3
```

```
puts version.minor # => Prints out the minor version number such as 0
```

```
# Example 4
```

```
puts version.patch # => Prints out the patch level for this version such as 0
```

## Get virtual time

`vt`

Get the virtual time of the current thread.

Introduced in v2.1

### Example

```
# Example 1
```

```
puts vt # prints 0
```

```
sleep 1
puts vt # prints 1
```

## Wait for duration

`wait beats` (number)

Synonym for `sleep` - see `sleep`

Introduced in v2.0

## Block-level enable and disable BPM scaling

`with_arg_bpm_scaling`

Turn synth argument bpm scaling on or off for the supplied block. Note, using `rt` for args will result in incorrect times when used within this block.

Introduced in v2.0

## Examples

# Example 1

```
use_bpm 120
play 50, release: 2 # release is actually 1 due to bpm scaling
with_arg_bpm_scaling false do # release is now 2
  play 50, release: 2
end
```

# Example 2

```
use_bpm 120
play 50, release: rt(2) # release is 2 seconds
sleep rt(2) # sleeps for 2 seconds
with_arg_bpm_scaling false do # ** Warning: release is NOT 2 seconds! **
  play 50, release: rt(2) # still sleeps for 2 seconds
  sleep rt(2)
end
```

## Block-level enable and disable arg checks

`with_arg_checks true_or_false` (boolean)

Similar to `use_arg_checks` except only applies to code within supplied `do/end` block. Previous arg check value is restored after block.

Introduced in v2.0

## Example

# Example 1

```
use_arg_checks true # Turn on arg checking:
play 80, cutoff: 100 # Args are checked
with_arg_checks false do #Arg checking is now disabled
  play 50, release: 3 # Args are not checked
  sleep 1
  play 72 # Arg is not checked
end
```

```
play 90 # Arg checking is re-enabled
# Args are checked
```

## Set the tempo for the code block

`with_bpm bpm` (number)

Sets the tempo in bpm (beats per minute) for everything in the given block. Affects all containing calls to `sleep` and all temporal synth arguments which will be scaled to match the new bpm. See also `use_bpm`

For dance music here's a rough guide for which BPM to aim for depending on your genre:

- Dub: 60-90 bpm
- Hip-hop: 60-100 bpm
- Downtempo: 90-120 bpm
- House: 115-130 bpm
- Techno/trance: 120-140 bpm
- Dubstep: 135-145 bpm
- Drum and bass: 160-180 bpm

Introduced in v2.0

### Example

```
# Example 1

4.times do
  sample :drum_bass_hard
  sleep 1
end

sleep 5

with_bpm 120 do
  4.times do
    sample :drum_bass_hard
    sleep 1
  end
end

sleep 5

4.times do
  sample :drum_bass_hard
  sleep 1
end

# default tempo is 60 bpm
# sleeps for 1 second
# sleeps for 5 seconds
# with_bpm sets a tempo for everything between do ... end (a block)
# Hear how it gets faster?
# set bpm to be twice as fast
# now sleeps for 0.5 seconds
# bpm goes back to normal
# sleeps for 1 second
```

## Set new tempo as a multiple of current tempo for block

`with_bpm_mul mul` (number)

Sets the tempo in bpm (beats per minute) for everything in the given block as a multiplication of the current tempo. Affects all containing calls to `sleep` and all temporal synth arguments which will be scaled to match the new bpm. See also `with_bpm`

Introduced in v2.3

### Example

```
# Example 1

use_bpm 60
play 50
sleep 1
play 62
sleep 2
with_bpm_mul 0.5 do
  play 50
end

# Set the BPM to 60
# Sleeps for 1 second
# Sleeps for 2 seconds
# BPM is now (60 * 0.5) == 30
```

```

sleep 1      # Sleeps for 2 seconds
play 62
end
sleep 1      # BPM is now back to 60, therefore sleep is 1 second

```

## Block-level enable and disable cue logging

`with_cue_logging true_or_false` (boolean)

Similar to `use_cue_logging` except only applies to code within supplied `do/end` block. Previous cue log value is restored after block.

Introduced in v2.6

### Example

```

# Example 1

use_cue_logging true      # Turn on debugging:

cue :foo                  # cue message is printed to log

with_cue_logging false do #Cue logging is now disabled
  cue :bar                # cue *is* sent but not displayed in log
end
sleep 1

cue :quux                 # Debug is re-enabled
                          # cue is displayed in log

```

## Block-level enable and disable debug

`with_debug true_or_false` (boolean)

Similar to `use_debug` except only applies to code within supplied `do/end` block. Previous debug value is restored after block.

Introduced in v2.0

### Example

```

# Example 1

use_debug true           # Turn on debugging:

play 80                  # Debug message is sent

with_debug false do     #Debug is now disabled
  play 50                # Debug message is not sent
  sleep 1
  play 72                # Debug message is not sent
end

play 90                  # Debug is re-enabled
                          # Debug message is sent

```

## Use Studio FX

`with_fx fx_name` (symbol)

This applies the named effect (FX) to everything within a given `do/end` block. Effects may take extra parameters to modify their behaviour. See FX help for parameter details.

For advanced control, it is also possible to modify the parameters of an effect within the body of the block. If you define the block with a

single argument, the argument becomes a reference to the current effect and can be used to control its parameters (see examples).

Introduced in v2.0

## Options

<code>reps:</code>	Number of times to repeat the block in an iteration.
<code>kill_delay:</code>	Amount of time to wait after all synths triggered by the block have completed before stopping and freeing the effect synthesiser.

## Examples

<pre># Example 1  with_fx :distortion do   play 50   sleep 1   sample :loop_amen end</pre>	<pre># Basic usage # Use the distortion effect with default parameters # =&gt; plays note 50 with distortion  # =&gt; plays the loop_amen sample with distortion too</pre>
<pre># Example 2  with_fx :level, amp: 0.3 do   play 50   sleep 1   sample :loop_amen end</pre>	<pre># Specify effect parameters # Use the level effect with the amp parameter set to 0.3</pre>
<pre># Example 3  with_fx :reverb, mix: 0.1 do  fx     play 60   sleep 2    control fx, mix: 0.5   play 60   sleep 2    control fx, mix: 1   play 60   sleep 2 end</pre>	<pre># Controlling the effect parameters within the block  # here we set the reverb level quite low to start with (0.1 # and we can change it later by using the 'fx' reference we  # plays note 60 with a little bit of reverb  # change the parameters of the effect to add more reverb # again note 60 but with more reverb  # change the parameters of the effect to add more reverb # plays note 60 with loads of reverb</pre>
<pre># Example 4  with_fx :reverb, reps: 16 do   play (scale :e3, :minor_pentatonic), release: 0.1   sleep 0.125 end  with_fx :reverb do   16.times do     play (scale :e3, :minor_pentatonic), release: 0.1     sleep 0.125   end end</pre>	<pre># Repeat the block 16 times internally  # The above is a shorthand for this:</pre>

## Block-level merge synth defaults

## with\_merged\_synth\_defaults

Specify synth arg values to be used by any following call to play within the specified **do/end** block. Merges the specified values with any previous defaults, rather than replacing them. After the **do/end** block has completed, previous defaults (if any) are restored.

Introduced in v2.0

### Examples

```
# Example 1
with_merged_synth_defaults amp: 0.5, pan: 1 do
  play 50
end # => plays note 50 with amp 0.5 and pan 1
```

```
# Example 2
play 50 #=> plays note 50
with_merged_synth_defaults amp: 0.5 do
  play 50 #=> plays note 50 with amp 0.5
  with_merged_synth_defaults pan: -1 do
    with_merged_synth_defaults amp: 0.7 do
      play 50 #=> plays note 50 with amp 0.7 and pan -1
    end
  end
end #=> plays note 50 with amp 0.5
play 50
end
```

## Specify random seed for code block

**with\_random\_seed** seed (number)

Resets the random number generator to the specified seed for the specified code block. All generated random numbers and randomisation functions such as **shuffle** and **choose** within the code block will use this new generator. Once the code block has completed, the original generator is restored and the code block generator is discarded. Use this to change the sequence of random numbers in your piece in a way that can be reproduced. Especially useful if combined with iteration. See examples.

Introduced in v2.0

### Examples

```
# Example 1
use_random_seed 1
puts rand
puts rand
use_random_seed 1
puts rand
with_random_seed 1 do
  puts rand
  puts rand
end
puts rand
# reset random seed to 1
# => 0.417022004702574
#=> 0.7203244934421581
# reset it back to 1
# => 0.417022004702574
# reset seed back to 1 just for this block
# => 0.417022004702574
#=> 0.7203244934421581
# => 0.7203244934421581
# notice how the original generator is restored
```

```
# Example 2
notes = (scale :eb3, :minor_pentatonic, num_octaves: 2)
# Generating melodies
# Create a set of notes to choose from.
# Scales work well for this

with_fx :reverb do
  live_loop :repeating_melody do
    with_random_seed 300 do
      # Create a live loop
      # Set the random seed to a known value every
      # time around the loop. This seed is the key
      # to our melody. Try changing the number to
      # something else. Different numbers produce
      # different melodies
```



```

8.times do

  play notes.choose, release: 0.1

  sleep 0.125
end

play notes.choose, amp: 1.5, release: 0.5

end
end

```

```

# Now iterate a number of times. The size of
# the iteration will be the length of the
# repeating melody.

# 'Randomly' choose a note from our ring of
# notes. See how this isn't actually random
# but uses a reproducible method! These notes
# are therefore repeated over and over...

# Note that this line is outside of
# the with_random_seed block and therefore
# the randomisation never gets reset and this
# part of the melody never repeats.

```

## Block-scoped sample-duration-based bpm modification

`with_sample_bpm string_or_number (sample_name_or_duration)`

Block-scoped modification of bpm so that sleeping for 1 will sleep for the duration of the sample.

Introduced in v2.1

### Options

`num_beats:` The number of beats within the sample. By default this is 1.

### Examples

# Example 1

```

live_loop :dnb do
  with_sample_bpm :loop_amen do
    sample :bass_dnb_f
    sample :loop_amen
    sleep 1
  end
end

```

#Set bpm based on :loop\_amen duration

#`sleep`ing for 1 sleeps for duration of :loop\_amen

# Example 2

```

live_loop :dnb do
  with_sample_bpm :loop_amen, num_beats: 4 do

    sample :bass_dnb_f
    sample :loop_amen
    sleep 4

  end
end

```

# Set bpm based on :loop\_amen duration  
# but also specify that the sample duration  
# is actually 4 beats long.

#`sleep`ing for 4 sleeps for duration of :loop\_amen  
# as we specified that the sample consisted of  
# 4 beats

## Block-level use new sample defaults

`with_sample_defaults`

Specify new default values to be used by all subsequent calls to `sample` within the `do/end` block. After the `do/end` block has completed, the previous sampled defaults (if any) are restored. For the contents of the block, will remove and override any previous defaults.

Introduced in v2.5

## Example

```
# Example 1
sample :loop_amen # plays amen break with default arguments
use_sample_defaults amp: 0.5, cutoff: 70
sample :loop_amen # plays amen break with an amp of 0.5, cutoff of 70 and defaults for rest
with_sample_defaults cutoff: 90 do
  sample :loop_amen # plays amen break with a cutoff of 90 and defaults for rest of args - no
end
sample :loop_amen # plays amen break with a cutoff of 70 and amp is 0.5 again as the previous
```

## Block-level use sample pack

`with_sample_pack` `pack_path` (string)

Given a path to a folder of samples on your filesystem, this method makes any `.wav`, `.wave`, `.aif`, or `.aiff` files in that folder available as samples inside the given block. Consider using `with_sample_pack_as` when using multiple sample packs.

Introduced in v2.0

## Example

```
# Example 1
with_sample_pack '/path/to/sample/dir' do
  sample :foo #=> plays /path/to/sample/dir/foo.{wav|wave|aif|aiff}
end
```

## Block-level use sample pack alias

`with_sample_pack_as` `pack_path` (string)

Similar to `with_sample_pack` except you can assign prefix aliases for samples. This lets you 'namespace' your sounds so that they don't clash, even if they have the same filename.

Introduced in v2.0

## Example

```
# Example 1
with_sample_pack_as '/home/yourname/path/to/sample/dir', :my_samples do
  sample :my_samples__foo # The foo sample is now available, with a
end #=> plays /home/yourname/path/to/sample/dir/my_samples__foo
```

## Block-level synth switching

`with_synth` `synth_name` (symbol)

Switch the current synth to `synth_name` but only for the duration of the `do/end` block. After the `do/end` block has completed, the previous synth is restored.

Introduced in v2.0

## Example

```

# Example 1
play 50
sleep 2
use_synth :supersaw
play 50
sleep 2
with_synth :saw_beep do
  play 50
end
sleep 2
play 50

```

# Plays with default synth  
# Plays with supersaw synth  
# Plays with saw\_beep synth  
# Previous synth is restored  
# Plays with supersaw synth

## Block-level use new synth defaults

### with\_synth\_defaults

Specify new default values to be used by all calls to `play` within the `do/end` block. After the `do/end` block has completed the previous synth defaults (if any) are restored.

Introduced in v2.0

### Example

```

# Example 1
play 50
use_synth_defaults amp: 0.5, pan: -1
play 50
with_synth_defaults amp: 0.6, cutoff: 80 do
  play 50
end
play 60

```

# plays note 50 with default arguments  
# plays note 50 with an amp of 0.5, pan of -1 and defaults for rest of  
# plays note 50 with an amp of 0.6, cutoff of 80 and defaults for rest  
# plays note 60 with an amp of 0.5, pan of -1 and defaults for rest of

## Block-level note transposition

### with\_transpose note\_shift (number)

Similar to `use_transpose` except only applies to code within supplied `do/end` block. Previous transpose value is restored after block.

Introduced in v2.0

### Example

```

# Example 1
use_transpose 3
play 62
with_transpose 12 do
  play 50
  sleep 1
  play 72
end
play 80

```

# Plays note 65  
# Plays note 62  
# Plays note 84  
# Original transpose value is restored  
# Plays note 83

## Block-level tuning modification

`with_tuning tuning (symbol), fundamental_note (symbol_or_number)`

Similar to `use_tuning` except only applies to code within supplied `do/end` block. Previous tuning value is restored after block.

Introduced in v2.6

## Example

```
# Example 1
```

```
use_tuning :equal, :c
play :e4
with_tuning :just, :c do
  play :e4
  sleep 1
  play :c4
end
play :e4
```

# Plays note 64  
# Plays note 63.8631  
# Plays note 60  
# Original tuning value is restored  
# Plays note 64

---

[Apprentice] Haunted  
[Illusionist] Fm Noise  
[Illusionist] Jungle  
[Illusionist] Reich Phase  
[Illusionist] Filtered Dnb  
[Illusionist] Ocean  
[Magician] Tron Bike  
[Magician] Ambient  
[Magician] Idm Breakbeat  
[Magician] Compus Beats  
[Magician] Echo Drama  
[Magician] Wob Rhyth  
[Magician] Acid  
[Sorcerer] Driving Pulse  
[Sorcerer] Square Skit  
[Sorcerer] Rerezzed  
[Sorcerer] Bach  
[Sorcerer] Monday Blues  
[Wizard] Tilburg 2  
[Wizard] Shufflit  
[Wizard] Blip Rhythm  
[Wizard] Blimp Zones  
[Wizard] Time Machine  
[Algomancer] Sonic Dreams

## # Haunted Play Example

# Coded by Sam Aaron

```
live_loop :haunted do
  sample :perc_bell, rate: rrand(-1.5, 1.5)
  sleep rrand(0.1, 2)
end
```

---

## # Fm Noise Play Example

```
# Coded by Sam Aaron
```

```
use_synth :fm
```

```
live_loop :sci_fi do  
  p = play (chord :Eb3, :minor).choose - [0, 12, -12].choose, divisor: 0.01, div_slide: rrand(0, 10), d  
  control p, divisor: rrand(0.001, 50)  
  sleep [0.5, 1, 2].choose  
end
```

---

## # Jungle Play Example

```
# Coded by Sam Aaron
use_bpm 50

with_fx :lpf, cutoff: 90 do
  with_fx :reverb, mix: 0.5 do
    with_fx :compressor, pre_amp: 40 do
      with_fx :distortion, distort: 0.4 do
        live_loop :jungle do
          use_random_seed 667
          4.times do
            sample :loop_amen, beat_stretch: 1, rate: [1, 1, 1, -1].choose / 2.0, finish: 0.5, amp: 0.5
            sample :loop_amen, beat_stretch: 1
            sleep 1
          end
        end
      end
    end
  end
end
end
end
end
```

---



## # Reich Phase Play Example

```
# Steve Reich's Piano Phase
# See: https://en.wikipedia.org/wiki/Piano\_Phase

# use_synth :piano
notes = (ring :E4, :Fs4, :B4, :Cs5, :D5, :Fs4, :E4, :Cs5, :B4, :Fs4, :D5, :Cs5)

live_loop :slow do
  play notes.tick, release: 0.1
  sleep 0.3
end

live_loop :faster do
  play notes.tick, release: 0.1
  sleep 0.295
end
```

---

## # Filtered Dnb Play Example

```
# Coded by Sam Aaron
```

```
use_sample_bpm :loop_amen
```

```
with_fx :rlpf, cutoff: 10, cutoff_slide: 4 do |c|  
  live_loop :dnb do  
    sample :bass_dnb_f, amp: 5  
    sample :loop_amen, amp: 5  
    sleep 1  
    control c, cutoff: rrand(40, 120), cutoff_slide: rrand(1, 4)  
  end  
end
```

---

## # Ocean Play Example

# Coded by Sam Aaron

```
with_fx :reverb, mix: 0.5 do
  live_loop :oceans do
    s = synth [:bnoise, :cnoise, :gnoise].choose, amp: rrand(0.5, 1.5), attack: rrand(0, 4), sustain: r
    control s, pan: rrand(-1, 1), cutoff: rrand(60, 110)
    sleep rrand(2, 4)
  end
end
```

---

## # Tron Bike Play Example

```
# Coded by Sam Aaron
```

```
use_random_seed 10
```

```
notes = (ring :b1, :b2, :e1, :e2, :b3, :e3)
```

```
live_loop :tron do
```

```
  with_synth :dsaw do
```

```
    with_fx(:slicer, phase: [0.25,0.125].choose) do
```

```
      with_fx(:reverb, room: 0.5, mix: 0.3) do
```

```
        n1 = (chord notes.choose, :minor).choose
```

```
        n2 = (chord notes.choose, :minor).choose
```

```
        p = play n1, amp: 2, release: 8, note_slide: 4, cutoff: 30, cutoff_slide: 4, detune: rrand(0, 6
```

```
        control p, note: n2, cutoff: rrand(80, 120)
```

```
      end
```

```
    end
```

```
  end
```

```
  sleep 8
```

```
end
```

---

## # Ambient Play Example

# Coded by Sam Aaron

```
load_samples(sample_names :ambi)
sleep 2
```

```
with_fx :reverb, mix: 0.8 do
  live_loop :foo do
    # try changing the sp_vars..
    sp_name = choose sample_names :ambi
    # sp_name = choose sample_names :drum
    sp_time = [1, 2].choose
    #sp_time = 0.5
    sp_rate = 1
    #sp_rate = 4

    s = sample sp_name, cutoff: rrand(70, 130), rate: sp_rate * choose([0.5, 1]), pan: rrand(-1, 1), pa
    control s, pan: rrand(-1, 1)
    sleep sp_time
  end
end
```

---

## # Idm Breakbeat Play Example

# Coded by Sam Aaron

```
live_loop :idm_bb do
  n = [1,2,4,8,16].choose
  sample :drum_heavy_kick, amp: 2
  sample :ambi_drone, rate: [0.25, 0.5, 0.125, 1].choose, amp: 0.25 if one_in(8)
  sample :ambi_lunar_land, rate: [0.5, 0.125, 1, -1, -0.5].choose, amp: 0.25 if one_in(8)
  sample :loop_amen, attack: 0, release: 0.05, start: 1 - (1.0 / n), rate: [1,1,1,1,1,1,-1].choose
  sleep sample_duration(:loop_amen) / n
end
```

---

## # Compus Beats Play Example

```
# Coded by Sam Aaron
```

```
use_sample_bpm :loop_compus, num_beats: 4
```

```
live_loop :loopr do
  sample :loop_compus, rate: [0.5, 1, 1, 1, 1, 2].choose unless one_in(10)
  sleep 4
end
```

```
live_loop :bass do
  sample :bass_voxy_c, amp: rrand(0.1, 0.2), rate: [0.5, 0.5, 1, 1,2,4].choose if one_in(4)
  use_synth :mod_pulse
  use_synth_defaults mod_invert_wave: 1
  play :C1, mod_range: 12, amp: rrand(0.5, 1), mod_phase: [0.25, 0.5, 1].choose, release: 1, cutoff: rr
  play :C2, mod_range: [24, 36, 34].choose, amp: 0.35, mod_phase: 0.25, release: 2, cutoff: 60, pulse_w
  sleep 1
end
```

---

## # Echo Drama Play Example

```
# Coded by Sam Aaron
```

```
use_synth :tb303
use_bpm 45
use_random_seed 3
use_debug false

with_fx :reverb do
  with_fx(:echo, delay: 0.5, decay: 4) do
    live_loop :echoes do
      play chord([:b1, :b2, :e1, :e2, :b3, :e3].choose, :minor).choose, cutoff: rrand(40, 100), amp: 0.
      sleep [0.25, 0.5, 0.5, 0.5, 1, 1].choose
    end
  end
end
```

---



## # Wob Rhyth Play Example

```
# Coded by Sam Aaron

use_debug false

with_fx :reverb do
  live_loop :choral do
    r = (ring 0.5, 1.0/3, 3.0/5).choose
    cue :choir, rate: r
    8.times do
      sample :ambi_choir, rate: r, pan: rrand(-1, 1)
      sleep 0.5
    end
  end
end

live_loop :wub_wub do
  with_fx :wobble, phase: 2, reps: 16 do |w|
    with_fx :echo, mix: 0.6 do
      sample :drum_heavy_kick
      sample :bass_hit_c, rate: 0.8, amp: 0.4
      sleep 1
      ## try changing the wobble's phase duration:
      # control w, phase: (ring 0.5, 1, 2).choose
    end
  end
end
```

---

## # Acid Play Example

```
# Coded by Sam Aaron

use_debug false
load_sample :bd_fat

8.times do
  sample :bd_fat, amp: (line 0, 5, steps: 8).tick
  sleep 0.5
end

live_loop :drums do
  sample :bd_fat, amp: 5
  sleep 0.5
end

live_loop :acid do
  cue :foo
  4.times do |i|
    use_random_seed 667
    16.times do
      use_synth :tb303
      play_chord(:e3, :minor).choose, attack: 0, release: 0.1, cutoff: rrand_i(50, 90) + i * 10
      sleep 0.125
    end
  end
end

cue :bar
32.times do |i|
  use_synth :tb303
  play_chord(:a3, :minor).choose, attack: 0, release: 0.05, cutoff: rrand_i(70, 98) + i, res: rrand(0
  sleep 0.125
end

cue :baz
with_fx :reverb, mix: 0.3 do |r|
  32.times do |m|
    control r, mix: 0.3 + (0.5 * (m.to_f / 32.0)) unless m == 0 if m % 8 == 0
    use_synth :prophet
    play_chord(:e3, :minor).choose, attack: 0, release: 0.08, cutoff: rrand_i(110, 130)
    sleep 0.125
  end
end

cue :quux
in_thread do
  use_random_seed 668
  with_fx :echo, phase: 0.125 do
    16.times do
      use_synth :tb303
      play_chord(:e3, :minor).choose, attack: 0, release: 0.1, cutoff: rrand(50, 100)
      sleep 0.25
    end
  end
end

sleep 4
end
```

---

## # Driving Pulse Play Example

# Coded by Sam Aaron

```
load_sample :drum_heavy_kick
```

```
live_loop :drums do
  sample :drum_heavy_kick, rate: 0.75
  sleep 0.5
  sample :drum_heavy_kick
  sleep 0.5
end
```

```
live_loop :synths do
  use_synth :mod_pulse
  use_synth_defaults amp: 1, mod_range: 15, cutoff: 80, pulse_width: 0.2, attack: 0.03, release: 0.6,
  play 30
  sleep 0.25
  play 38
  sleep 0.25
end
```

---

## # Square Skit Play Example

```
# Coded by Sam Aaron
```

```
use_debug false
```

```
live_loop :skit do
  with_fx :slicer, phase: 1, invert_wave: 1, wave: 0 do
    with_fx :slicer, wave: 0, phase: 0.25 do
      sample :loop_mika, rate: 1, amp: 2
    end
    sleep 8
  end
end
```

```
live_loop :foo, auto_cue: false do
  tick(:note) if factor? tick, 4
  use_synth :square
  density 2 do
    play (knit :c2, 2, :e1, 1, :f3, 1).look(:note), release: 0, attack: 0.25, amp: 1, cutoff: rrand_i(7)
    sleep 0.5
  end
end
```

```
live_loop :kik, auto_cue: false do
  density 1 do
    sample :bd_haus, amp: 2
    sleep 0.5
  end
end
```

```
live_loop :piano, auto_cue: false do
  sleep 4
  with_fx :slicer, phase: 0.25, wave: 1 do
    sleep 4
    sample :ambi_piano, amp: 2
  end
end
```

---

## # Rerezzed Play Example

```
# Coded by Sam Aaron

use_debug false
notes = (scale :e1, :minor_pentatonic, num_octaves: 2).shuffle

live_loop :rerezzed do
  tick_reset
  t = 0.04
  sleep -t
  with_fx :bitcrusher do
    s = synth :dsaw, note: :e3, sustain: 8, note_slide: t, release: 0
    64.times do
      sleep 0.125
      control s, note: notes.tick
    end
  end
  sleep t
end

live_loop :industry do
  sample :loop_industrial, beat_stretch: 1
  sleep 1
end

live_loop :drive do
  sample :bd_haus, amp: 3
  sleep 0.5
end
```

---

## # Bach Play Example

```
# Bach Minuet in G
#
# Coded by Robin Newman

use_bpm 60
use_synth_defaults release: 0.5, amp: 0.7, cutoff: 90
use_synth :beep

## Each section of the minuet is repeated
2.times do

  ## First start a thread for the first 8 bars of the bass left hand part
  in_thread do
    play_chord [55,59]#b1
    sleep 1
    play_pattern_timed [57],[0.5]
    play_pattern_timed [59],[1.5] #b2
    play_pattern_timed [60],[1.5] #b3
    play_pattern_timed [59],[1.5] #b4
    play_pattern_timed [57],[1.5] #b5
    play_pattern_timed [55],[1.5] #b6
    play_pattern_timed [62,59,55],[0.5] #b7
    play_pattern_timed [62],[0.5] #b8
    play_pattern_timed [50,60,59,57],[0.25]
  end

  ## Play concurrently the first 8 bars of the right hand part
  play_pattern_timed [74],[0.5]#b1
  play_pattern_timed [67,69,71,72],[0.25]
  play_pattern_timed [74,67,67],[0.5]#b2
  play_pattern_timed [76],[0.5]#b3
  play_pattern_timed [72,74,76,78],[0.25]
  play_pattern_timed [79,67,67],[0.5]#b4
  play_pattern_timed [72],[0.5] #b5
  play_pattern_timed [74,72,71,69],[0.25]
  play_pattern_timed [71],[0.5] #b6
  play_pattern_timed [72,71,69,67],[0.25]
  play_pattern_timed [66],[0.5] #b7
  play_pattern_timed [67,69,71,67],[0.25]
  play_pattern_timed [71,69],[0.5,1] #b8

  ## Start a new thread for bars 9-16 of the left hand part
  in_thread do
    play_chord [55,59]#b9=b1
    sleep 1
    play 57
    sleep 0.5
    play_pattern_timed [55,59,55],[0.5] #b10
    play_pattern_timed [60],[1.5] #b11=b3
    play_pattern_timed [59,60,59,57,5],[0.5,0.25,0.25,0.25,0.25] #b12=b4
    play_pattern_timed [57,54],[1,0.5] #b13
    play_pattern_timed [55,59],[1,0.5] #b14
    play_pattern_timed [60,62,50],[0.5] #b15
    play_pattern_timed [55,43],[1,0.5] #b16
  end

  ## Play concurrently bars 9-16 of the right hand part the first six
  ## bars repeat bars 1-6
  play_pattern_timed [74],[0.5]#b9 = b1
  play_pattern_timed [67,69,71,72],[0.25]
  play_pattern_timed [74,67,67],[0.5]#b10=b2
  play_pattern_timed [76],[0.5]#b11=b3
  play_pattern_timed [72,74,76,78],[0.25]
  play_pattern_timed [79,67,67],[0.5]#b12=b4
  play_pattern_timed [72],[0.5] #b13=b5
  play_pattern_timed [74,72,71,69],[0.25]
  play_pattern_timed [71],[0.5] #b14=b6
  play_pattern_timed [72,71,69,67],[0.25]
  play_pattern_timed [69],[0.5] #b15
  play_pattern_timed [71,69,67,66],[0.25]
```

```

    play_pattern_timed [67],[1.5] #b16
end

## =====second section starts here=====
## The second section is also repeated
2.times do

  ## Start a thread for bars 17-24 of the left hand part
  in_thread do
    play_pattern_timed [55],[1.5] #b17
    play_pattern_timed [54],[1.5] #b18
    play_pattern_timed [52,54,52],[0.5] #b19
    play_pattern_timed [57,45],[1,0.5] #b20
    play_pattern_timed [57],[1.5] #b21
    play_pattern_timed [59,62,61],[0.5] #b22
    play_pattern_timed [62,54,57],[0.5] #b23
    play_pattern_timed [62,50,60],[0.5] #b24
  end

  ## Play bars 17 to 24 of the right hand concurrently with the left
  ## hand thread
  play_pattern_timed [83],[0.5] #b17
  play_pattern_timed [79,81,83,79],[0.25]
  play_pattern_timed [81],[0.5] #b18
  play_pattern_timed [74,76,78,74],[0.25]
  play_pattern_timed [79],[0.5] #b19
  play_pattern_timed [76,78,79,74],[0.25]
  play_pattern_timed [73,71,73,69],[0.5,0.25,0.25,0.5] #b20
  play_pattern_timed [69,71,73,74,76,78],[0.25] #b21
  play_pattern_timed [79,78,76],[0.5] #b22
  play_pattern_timed [78,69,73],[0.5] #b23
  play 74 #b24
  sleep 1.5

  ## Start a new thread for bars 25-32 of the left hand part
  in_thread do
    play_pattern_timed [59,62,59],[0.5] #b25
    play_pattern_timed [60,64,60],[0.5] #b26
    play_pattern_timed [59,57,55],[0.5] #b27
    play 62 #b28
    sleep 1.5 #includes a rest
    play_pattern_timed [50,54],[1,0.5] #b29
    play_pattern_timed [52,55,54],[0.5] #b30
    play_pattern_timed [55,47,50],[0.5] #b31
    play_pattern_timed [55,50,43],[0.5] #b32
  end

  ## Play bars 25-32 of the right hand part concurrently with the left
  ## hand thread
  play_pattern_timed [74,67,66,67],[0.5,0.25,0.25,0.5] #b25
  play_pattern_timed [76,67,66,67],[0.5,0.25,0.25,0.5] #b26
  play_pattern_timed [74,72,71],[0.5] #b27
  play_pattern_timed [69,67,66,67,69],[0.25,0.25,0.25,0.25,0.5] #b28
  play_pattern_timed [62,64,66,67,69,71],[0.25] #b29
  play_pattern_timed [72,71,69],[0.5] #b30
  play_pattern_timed [71,74,67,66],[0.25,0.25,0.5,0.5] #b31
  play_chord [67,59] #b32
  sleep 1.5
end

```

---

# # Monday Blues Play Example

```
# Coded by Sam Aaron
```

```
use_debug false
load_samples [:drum_heavy_kick, :drum_snare_soft]
```

```
live_loop :drums do
  puts "slow drums"
  6.times do
    sample :drum_heavy_kick, rate: 0.8
    sleep 0.5
  end
end
```

```
  puts "fast drums"
  8.times do
    sample :drum_heavy_kick, rate: 0.8
    sleep 0.125
  end
end
```

```
live_loop :synths, delay: 6 do
  puts "how does it feel?"
  use_synth :mod_saw
  use_synth_defaults amp: 0.5, attack: 0, sustain: 1, release: 0.25, mod_range: 12, mod_phase: 0.5, mod
  notes = (ring :F, :C, :D, :D, :G, :C, :D, :D)
  notes.each do |n|
    tick
    play note(n, octave: 1), cutoff: (line 90, 130, steps: 16).look
    play note(n, octave: 2), cutoff: (line 90, 130, steps: 32).look
    sleep 1
  end
end
```

```
live_loop :snare, delay: 12.5 do
  sample :drum_snare_soft
  sleep 1
end
```

---



## # Tilburg 2 Play Example

```
# Coded by Sam Aaron

use_debug false
load_samples :guit_em9, :bd_haus

live_loop :low do
  tick
  synth :zawa, wave: 1, phase: 0.25, release: 5, note: (knit :e1, 12, :c1, 4).look, cutoff: (range 60,
  sleep 4
end

live_loop :lands, auto_cue: false do
  with_fx :reverb, room: 1, reps: 4 do
    use_synth :dsaw
    use_random_seed 310003
    ns = (scale :e2, :minor_pentatonic, num_octaves: 4).take(4)
    16.times do
      play ns.choose, detune: 12, release: 0.1, amp: 2, amp: rand + 0.5, cutoff: rrand(70, 120), amp: 2
      sleep 0.125
    end
  end
end

live_loop :fietsen do
  sleep 0.25
  sample :guit_em9, rate: -1
  sleep 7.75
end

live_loop :tijd, auto_cue: true do
  sample :bd_haus, amp: 2.5, cutoff: 100
  sleep 0.5
end

live_loop :ind do
  # sync :tijd
  sample :loop_industrial, beat_stretch: 1
  sleep 1
end
```

---

## # Shufflit Play Example

```
# Coded by Sam Aaron

use_debug false
use_random_seed 667
load_sample :ambi_lunar_land
sleep 1

live_loop :travelling do
  use_synth :beep
  notes = scale(:e3, :minor_pentatonic, num_octaves: 1)
  use_random_seed 679
  tick_reset_all
  with_fx :echo, phase: 0.125, mix: 0.4, reps: 16 do
    sleep 0.25
    play notes.choose, attack: 0, release: 0.1, pan: (range -1, 1, step: 0.125).tick, amp: rrand(2, 2.5)
  end
end

live_loop :comet, auto_cue: false do
  if one_in 4
    sample :ambi_lunar_land
    puts :comet_landing
  end
  sleep 8
end

live_loop :shuff, auto_cue: false do
  with_fx :hpf, cutoff: 10, reps: 8 do
    tick
    sleep 0.25
    sample :bd_tek, amp: factor?(look, 8) ? 6 : 4
    sleep 0.25
    use_synth :tb303
    use_synth_defaults cutoff_attack: 1, cutoff_release: 0, env_curve: 2
    play (knit :e2, 24, :c2, 8).look, release: 1.5, cutoff: (range 70, 90).look, depth: 10, amp: 2 if
    sample :sn_dub, rate: -1, sustain: 0, release: (knit 0.05, 3, 0.5, 1).look
  end
end
```

---

## # Blip Rhythm Play Example

```
# Coded by Sam Aaron

load_samples [:drum_heavy_kick, :elec_plip, :elec_blip]
use_bpm 100
use_random_seed 100

with_fx :reverb, mix: 0.6, room: 0.8 do
  with_fx :echo, room: 0.8, decay: 8, phase: 1, mix: 0.4 do
    live_loop :blip do
      n = [:e2, :e2, :a3].choose

      with_synth :dsaw do
        with_transpose -12 do
          in_thread do
            2.times do
              play n, attack: 0.6, release: 0.8, detune: rrand(0, 0.1), cutoff: rrand(80, 120)
              sleep 3
            end
          end
        end
      end
    end
  end

  sleep 4

  with_synth :tri do
    play chord(n, :m7), amp: 5, release: 0.8
  end

  sleep 2
end
end
end

with_fx :echo, room: 0.8, decay: 8, phase: 0.25, mix: 0.4 do
  live_loop :rhythm do
    sample :drum_heavy_kick, amp: 0.5
    sample :elec_plip, rate: [0.5, 2, 1, 4].choose * [1, 2, 3, 10].choose, amp: 0.6
    sleep 2
  end
end
```

---

## # Blimp Zones Play Example

```
# Coded by Sam Aaron

use_debug false
use_random_seed 667
load_sample :ambi_lunar_land
sleep 1

live_loop :foo do
  with_fx :reverb, kill_delay: 0.2, room: 0.3 do
    4.times do
      use_random_seed 4000
      8.times do
        sleep 0.25
        play chord(:e3, :m7).choose, release: 0.1, pan: rrand(-1, 1, res: 0.9), amp: 1
      end
    end
  end
end

live_loop :bar, auto_cue: false do
  if rand < 0.25
    sample :ambi_lunar_land
    puts :comet_landing
  end
  sleep 8
end

live_loop :baz, auto_cue: false do
  tick
  sleep 0.25
  cue :beat, count: look
  sample :bd_haus, amp: factor?(look, 8) ? 3 : 2
  sleep 0.25
  use_synth :fm
  play :e2, release: 1, amp: 1 if factor?(look, 4)
  synth :noise, release: 0.051, amp: 0.5
end
```

---

## # Time Machine Play Example

```
# Coded by Sam Aaron
```

```
use_debug false
```

```
live_loop :time do
  synth :tb303, release: 8, note: :e1, cutoff: (range 90, 60, -10).tick
  sleep 8
end
```

```
live_loop :machine do
  sample :loop_garzul, rate: (knit 1, 3, -1, 1).tick
  sleep 8
end
```

```
live_loop :vortex, auto_cue: false do
  use_synth [:pulse, :beep].choose
  sleep 0.125 / 2
  play scale(:e1, :minor_pentatonic).tick, attack: 0.125, release: 0, amp: 2, cutoff: (ring 70, 90, 100)
  sleep 0.125 / 2
end
```

```
live_loop :moon_bass, auto_cue: false do
  sample :bd_haus, amp: 1.5
  sleep 0.5
end
```

---

## # Sonic Dreams Play Example

```
# rand-seed-ver 32
#
# Coded by Sam Aaron
#
# Video: https://vimeo.com/110416910

use_debug false
load_samples [:bd_haus, :elec_blip, :ambi_lunar_land]

define :ocean do |num, amp_mul=1|
  num.times do
    s = synth [:bnoise, :cnoise, :gnoise].choose, amp: rrand(0.5, 1.5) * amp_mul, attack: rrand(0, 1),
      control s, pan: rrand(-1, 1), cutoff: rrand(60, 110)
    sleep rrand(0.5, 4)
  end
end

define :echoes do |num, tonics, co=100, res=0.9, amp=1|
  num.times do
    play chord(tonics.choose, :minor).choose, res: res, cutoff: rrand(co - 20, co + 20), amp: 0.5 * amp
    sleep [0.25, 0.5, 0.5, 0.5, 1, 1].choose
  end
end

define :bd do
  cue :in_relentless_cycles
  16.times do
    sample :bd_haus, amp: 4, cutoff: 120
    sleep 0.5
  end
  cue :winding_everywhichway
  2.times do
    2.times do
      sample :bd_haus, amp: 4, cutoff: 120
      sleep 0.25
    end
    sample :ambi_lunar_land
    sleep 0.25
  end
end

define :drums do |level, b_level=1, rand_cf=false|
  synth :fm, note: :e2, release: 0.1, amp: b_level * 3, cutoff: 130
  co = rand_cf ? rrand(110, 130) : 130
  a = rand_cf ? rrand(0.3, 0.5) : 0.6
  n = rand_cf ? :bnoise : :noise
  synth :noise, release: 0.05, cutoff: co, res: 0.95, amp: a if level > 0
  sample :elec_blip, amp: 2, rate: 2, pan: rrand(-0.8, 0.8) if level > 1
  sleep 1
end

define :synths do |s_name, co, n=:e2|
  use_synth s_name
  use_transpose 0
  use_synth_defaults detune: [12,24].choose, amp: 1, pan: lambda{rrand(-1, 1)}, cutoff: co, pulse_width

  play :e1, mod_range: [7, 12].choose
  sleep 0.125

  play :e3, mod_range: [7, 12].choose
  sleep [0.25, 0.5].choose

  play n, mod_range: 12
  sleep 0.5

  play chord(:e2, :minor).choose, mod_range: 12
  sleep 0.25
end

define :play_synths do
```

```

with_fx :reverb do |r|
  with_fx :echo, phase: 0.25 do |e|
    synths = [:mod_pulse, :mod_saw, :mod_dsaw, :mod_dsaw, :mod_dsaw, :mod_dsaw]
    cutoffs = [108, 78, 88, 98]
    synth = synths.rotate!.first
    4.times do |t|
      puts shuffle("0" * (30 - t) + ("1" * t)) unless t == 0
      co = cutoffs.rotate!.first + (t * 2)
      7.times do
        n = chord([:e2, :e3, :e4, :e5][t], :minor).choose
        synths(synth, co, n)
      end
      sleep 2
    end
    sleep 1
    cue :within
  end
end
end

define :binary_celebration do |n=1, st=1|
  in_thread do
    n.times do
      puts (0..30).map{|_| ["0", "1"].choose}.join
      sleep st
    end
  end
end

puts 'Introduction'
puts 'The Curved Ebb of Carpentry'
sleep 2

cue :oceans
at [7, 12], [:crash, :within_oceans] do |m|
  cue m
end

uncomment do
  use_random_seed 1000
  with_bpm 45 do
    with_fx :reverb do
      with_fx(:echo, delay: 0.5, decay: 4) do
        in_thread do
          use_random_seed 2
          ocean 5
          ocean 1, 0.5
          ocean 1, 0.25
        end
        sleep 10
        use_random_seed 1200
        echoes(5, [:b1, :b2, :e1, :e2, :b3, :e3])
        cue :a_distant_object
        echoes(5, [:b1, :e1, :e2, :e3])
        cue :breathes_time
        in_thread do
          echoes(5, [:e1, :e2, :e3])
        end
        use_synth :tb303
        echoes(1, [:e1, :e2, :e3], 60, 0.9, 0.5)
        echoes(1, [:e1, :e2, :e3], 62)
        echoes(1, [:e1, :e2, :e3], 64, 0.97)
        echoes(1, [:e1, :e2, :e3], 66)
        echoes(1, [:e1, :e2, :e3], 68)
        cue :liminality_holds_fast
        echoes(4, [:b1, :e1, :e2, :b3, :e3], 80)
        echoes(1, [:b1, :b2, :e1, :e2, :b3, :e3], 85, 0.98)
        cue :within_reach
        echoes(5, [:e1, :b2], 90)
        cue :as_it_unfolds
        in_thread do
          echoes(5, [:e1], 90)
        end
      end
    end
  end
end

```

```

    end
  end
end

in_thread(name: :bassdrums) do
  use_random_seed 0
  sleep 22
  3.times do
    bd
  end
  sleep 28
  live_loop :bd do
    bd
  end
end

in_thread(name: :drums) do
  use_random_seed 0
  level = -1
  with_fx :echo do |e|
    sleep 2
    drums -1, 0.1
    drums -1, 0.2
    drums -1, 0.4
    drums -1, 0.7
    puts "Part 2"
    puts "Inside the Machine"
    3.times do
      8.times do
        drums level, 0.8
      end
      6.times do
        drums(level)
      end

      sleep 1
      level += 1
    end
    sleep 4
    cue :dreams
    8.times do
      drums 1, 1, true
    end

    10.times do
      m = choose [shuffle(:within_dreams), :within_dreams, :dreams_within]
      cue m
      drums 2, 1, true
    end

    6.times do
      m = choose [shuffle("within") + "_dreams", :within_dreams.shuffle, "dreams_" + shuffle("within")]
      cue m
      drums 2
    end

    live_loop :drums do
      8.times do |i|
        drums 1
      end

      16.times do |i|
        cue " " * rand_i(32)
        at 1 do
          cue " " * i
        end
        drums 2
      end
    end
  end
end

in_thread name: :synths do

```



```
use_random_seed 0
sleep 12
cue :the_flow_of_logic
play_synths
end

in_thread do
  use_random_seed 0
  sync :within
  puts "Part 3"
  puts "Reality A"
  sleep 12
  use_synth_defaults phase: 0.5, res: 0.5, cutoff: 80, release: 3.3, wave: 1

  2.times do
    [80, 90, 100, 110].each do |cf|
      use_merged_synth_defaults cutoff: cf
      puts "1" * 30
      synth :zawa, note: :e2, phase: 0.25
      synth :zawa, note: :a1
      sleep 3
    end
    4.times do |t|
      binary_celebration(6, 0.5)
      synth :zawa, note: :e2, phase: 0.25, res: rrand(0.8, 0.9), cutoff: [100, 105, 110, 115][t]
      sleep 3
    end
  end

  puts 'Part n'
  puts 'The Observer becomes the Observed'
  # Your turn...
end
```

---