Tidal

1 Tidal

Welcome to this workshop on tidalcycles, known as *tidal* for short.

I'm Alex - alex@slab.org + http://slab.org/

Rough schedule for the two days:

Day one:

14:00 - 17:00

- Background
- Introduction to patterns - repetition, symmetry, interference and glitch
- Basics of (polyrhythmic) sequencing
- Haskell syntax
- Pattern transformation

Day two:

10.30-12.00

- More complex patternings
- Strategies for live coding performance
- Community

1.1 What is a cycle?

- Cyclic notion of time from Indian Classical music (Bol processor)
- The end is also the beginning (the *sam*)
- Time in Tidal is based on cycles, rather than beats
- Cycles are ticking over all the time
- Cycles have fixed duration (which you can change with the *cps* command)

1.2 Pattern

Types of pattern:

- Repetition
- Symmetry
- Interference
- Randomness/glitch

2 Basics of (polyrhythmic) sequencing with Tidal

2.1 Sub-sequences

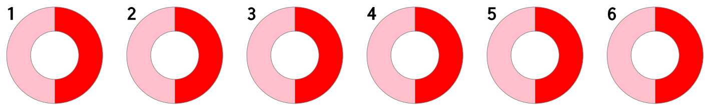
You can break down an step put it inside `[` and `]` with a comma.

```
d1 $ sound "[bd]"
```

Before we get hands on, lets look at some visual renderings of tidal patterns.

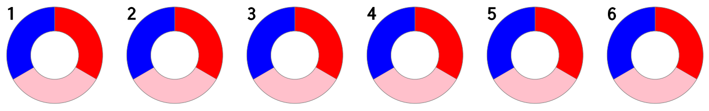
Sequences in tidal are generally denoted with double quotes:

```
"red pink"
```



You can 'read' the above diagram clockwise, from the top. You can see that the pattern repeats once per cycle. Six cycles are shown, but it will go on for ever. Here's what happens if we add another step to the sequence:

```
"red pink blue"
```



You can see that the steps have got shorter, so that they fit into the space. This already demonstrates how tidal is about cycles, not beats.

2.2 Make a sound

Make sure supercollider is running, and SuperDirt is running inside that (via the `SuperDirt.start` command, which you can put in the supercollider startup file so it automatically runs).

Run the following in atom, via ctrl-Enter (or on a mac, cmd-Enter):

```
d1 $ sound "bd"
```

Stop making a sound!

```
d1 silence
```

Tip: Make sure your tidal patterns have an empty line above and below, as tidal can't currently run two commands at once (there are workarounds for this).

Lets break that down..

- `d1` is a connection to the sound synthesiser, SuperDirt. There is also `d2`, up to `d9` or so.
- `$` passes what's on the right (`sound "bd"`) to what's on the left (`d1`). Without this, Tidal would only read as far as `sound` and get confused!
- `sound` says that a pattern of sounds (samples or synths) is coming up. That is, it turns a pattern of words into a pattern of sounds.

- `"bd"` is a pattern of words, in this case the single word `bd`, refering to a bass drum sound sample somewhere on your computer.

2.3 Where do the sounds come from?

Some are softsynths, e.g. supermandolin, superpiano, supergong etc

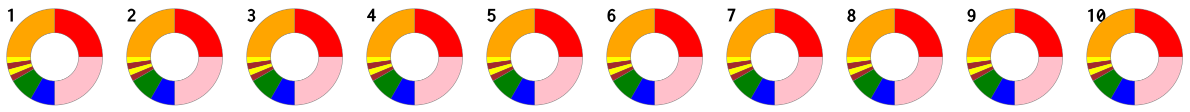
Most are sound samples, you can find them (and add to them) via SuperCollider.. File > Open User Support Directory -> downloaded-quarks > Dirt Samples

3 More visual patterns

Some patterns to mull over as we go through Tidal..

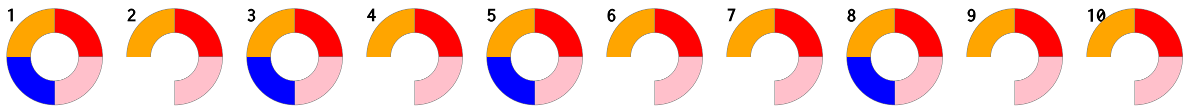
Embedded substeps

```
"red pink [blue green [brown yellow]*2] orange"
```



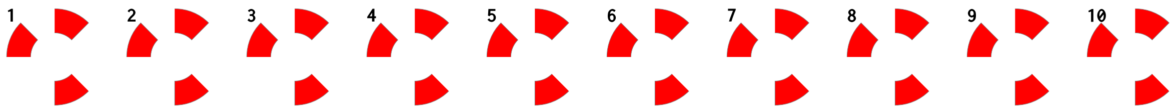
Sometimes take one out

```
"red pink blue? orange"
```

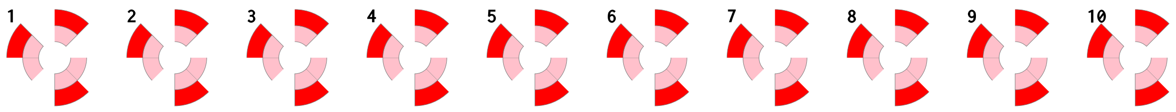


Euclidean rhythms

```
"red(3,8)"
```

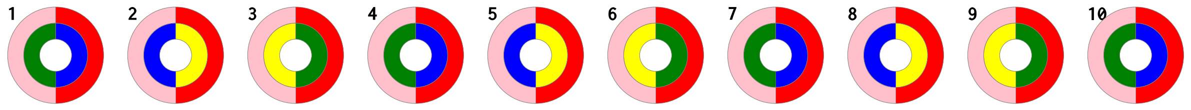


```
"[red(3,8), pink(5,8)]"
```



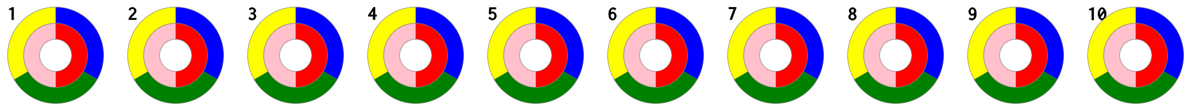
Polyrhythm

```
"{red pink, blue green yellow}"
```



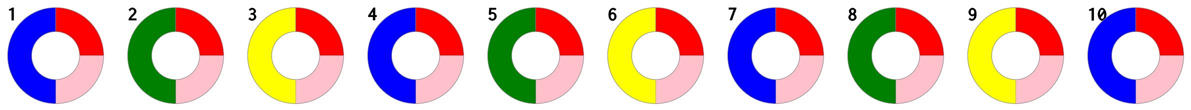
Polymeter

```
"[red pink, blue green yellow]"
```



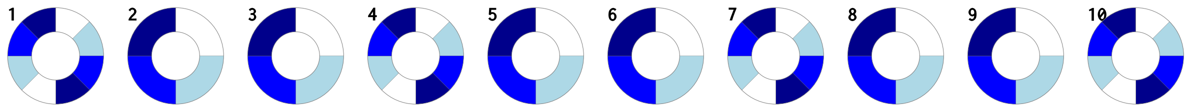
Take one per cycle with `<>`

```
"[red pink] <blue green yellow>"
```



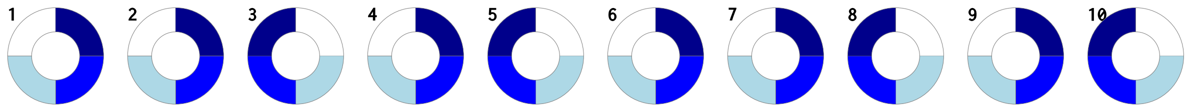
Twice as fast every third cycle

```
every 3 (fast 2) "white lightblue blue darkblue"
```



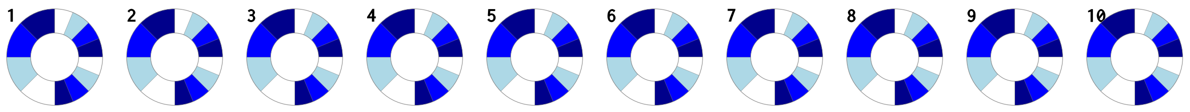
Some cycles, reverse the cycle

```
someCycles rev "white lightblue blue darkblue"
```

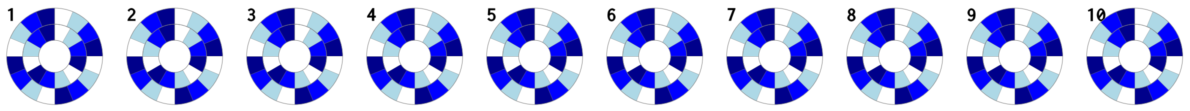


Pattern the speed

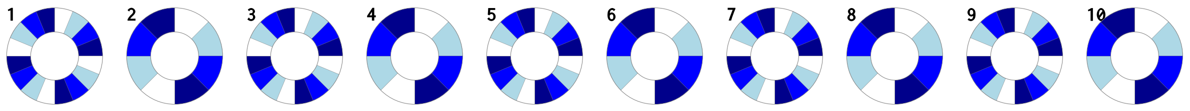
```
fast "4 2" "white lightblue blue darkblue"
```



```
fast "[4, 3]" "white lightblue blue darkblue"
```

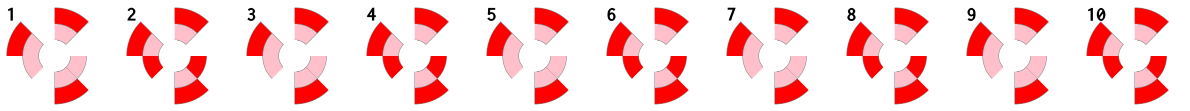


```
fast "<4 2>" "white lightblue blue darkblue"
```



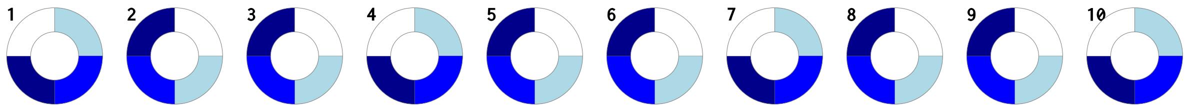
Pattern the euclidean values

```
"[red(<3 5>,8), pink(<5 3>,8)]"
```



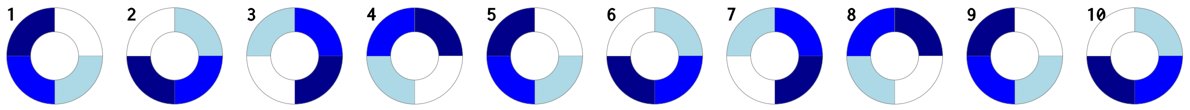
Shift time

```
every 3 (0.25 <~) "white lightblue blue darkblue"
```



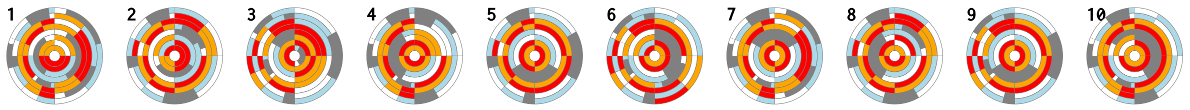
Shift time, successively

```
iter 4 "white lightblue blue darkblue"
```



Combine transformations to get something unexpected

```
superimpose (fast "2 4") $ iter 4 $ superimpose rev
    $ every 3 (0.25 <~) $ "[white grey lightblue, orange red]"
```


1
2
3
4
5
6
7
8
9
10