tidalcycles

Little book of patterns

Mini-notation (in double quotes)

```
-- Four sounds in a cycle:
d1 $ sound "bd hh sd cp"

-- Six sounds in a cycle:
d1 $ s "bd hh sd cp arpy kurt"

-- Silence/rest with '~'
d1 $ s "bd hh ~ ~ mt ~ kurt:2 ~"

-- Sub-sequence with []
d1 $ s "bd ~ [bd sd mt] mt"-

--- More than one at the same time:
d1 $ s "[ht mt lt, arpy kurt]"

-- Curlies for stepwise polyrhythm:
d1 $ s "{sd mt, arpy arpy:1 arpy:2}"
```

```
--  speed up a step with *
d1 $ s "bd sd*2 ~ [cp arpy]*2"

--  slow down with /
d1 $ s "bd sd/2 ~ [cp arpy ht lt]/2"

-- pick one per cycle
d1 $ s "bd <arpy arpy:1 arpy:2>"

--  repeat with !
d1 $ s "bd!3 sd"

-- drop out randomly
d1 $ s "bd sd? mt? lt"

-- Distribute 3 events over 8 steps
d1 $ s "bd(3,8)"
-- 'Euclidian rhythms', same as:
d1 $ s "bd ~ ~ bd ~ ~ bd ~"
```

Pattern all the things

sound (or just s) patterns sample set or synth. There is much more to pattern!

***ctrl-enter** to run multiline patterns*

```
-- sample number
d1 $ n "[0 1 [~ 0] 1,2*4 2*8]"
    # s "drum"

-- vowel filter
d1 $ vowel "a o e*2 i" # sound
"drum"

-- Combine multiple controls
d1 $ note "1 [2 7] 4 5"
  # sound "jungbass:6"
  # legato 1 # crush "3 2"
```

```
-- Howabout a sine wave on gain:
d1 $ sound "bd*32" # gain sine

-- Or randomised panning:
d1 $ sound "bd*4" # pan rand

-- '#' takes structure from the left
-- Two sounds:
d1 $ sound "drum kurt" # n "0 1 2"
-- Three sounds
d1 $ n "0 1 2" # sound "drum kurt"

-- You can add patterns together
d1 $ n ("0 5*2 7" + "<0 3 5>")
  # s "supermandolin"

-- Or even add control patterns:
d1 $ n "0 5*2 7" # s "supermandolin"
  + n "<0 3 5>"
```

Functions for manipulating time, sound and space

```
-- reverse
d1 $ rev $ n "0 1 2 3" # s "arpy"

-- successively shift time
d1 $ iter 4 $ n "0 1 2 3" # s "arpy"

-- make faster
d1 $ fast 4 $ n "0 1 2 3" # s "arpy"

-- make faster and pitch up
d1 $ hurry 4 $ n "0 1 2" # s "arpy"

-- lets pattern that
d1 $ fast "2 3" $ n "0 1 2 3"
  # s "arpy"

-- chop up samples, reverse the bits
d1 $ rev $ chop 8 $ n "0 [2 1] 4 3"
  # s "cp speakspell"
```

Higher order functions

Passing functions to functions

```
-- apply function in one speaker/ear
d1 $ jux rev $ chop 8 $
  n "0 [2 1] 4*2 3" # s "speakspell"

-- apply function every 'n' cycles
d1 $ every 3 (fast 2) $
  n "0 2 [~ 1] 5" # s "speakspell"

-- apply offset, on top of original
d1 $ off 0.125 (+ note 7) $
  note "0 3 7 12" # s "gtr"

-- join functions together with .
d1 $ every 2 (rev . chop 8) $
  s "bd sd"

see tidalcycles.org for many more
functions!
```

Operator re-cap:

join two control patterns (taking triggers from the left)

$ resolve what's on the right, and give it as a value to the function on the left. These are the same:

```
d1 $ rev $ s "bd sn"
d1 $ rev (s "bd sn")
```

Links
Home: tidalcycles.org
Chat channels: chat.toplap.org
Forum: forum.toplap.org

See also
* toplap.org
* algorave.com
* iclc.livecodenetwork.org