The introduction to Live Coding via the TidalCycles Syntax

NOTE: The syntax here described is mostly going to be based on MiniTidal, the language present in Estuary. For an excellent and thorough tutorial on TidalCycles go here

Let us begin.

Functions functions functions!!!

The simplest program in MiniTidal could be:

```
sound "bd"
```

The word `sound` here is the way of naming a function. A function can be understood as a process and/or algorithm that will take an input and return an output. For example:

```
add1 x = x + 1
```

Here the function name is `add1`, input is x, the output is x+1. This function might be out of sight but, if we know it exists and what input values it requires we can invoke it. So, when I invoke my function add1 and I input a number I can expect always the input plus one to be the output.

The function `sound` will be given a value and will return... a sound!

So, what value we need to provide as input to the `sound` function?

That would be a Pattern. Here is where Tidal might be distinguished from other ways of computar-mediated music-making.

A pattern can be identified in Tidal because it has quotation marks around it always. It is a notation that describes an arrengements of elements. In the case of `sound` (or `s` for short from now on) the name of samples (or the name of sounds that exists in a bank of samples somewhere out of sight).

So in the example: `sound "bd"`

We have a pattern of 1 sample named bd to be played as a sound. Remember that TidalCycles generates cycles, so the sound we just invoked will be played cyclically.

Pattern notation

A sound pattern that will distribute 4 sounds in a cycle:

```
s "bd feel cp bd"
```

A `s` pattern with two claps instead of one in the third beat:

```
s "bd feel cp*2 bd"
```

A `s` pattern that will distribute 4 sounds in a cycle but will also distribute two sounds in the last segment of that cycle (notice the notation for rest!):

```
s "drum glitch:1 cp*2 [~ bd]"
```

A pattern that will have nested patterns (notice the `808:5` and `808:3`, which means that in the sample bank 808 we want to play the sixth sample (since we start counting samples at 0) and the fourth):

```
s "[808:5 house:2 [jungle sn] [~ [cp ~ 808:1]]]"
```

Euclidean rhythms:

A cuban tresillo (notice the comment after the -- that does not compute anything but stands as a note so other humans can know something you want them to know):

```
s "bottle(3,8)" -- this is a Cuban tresillo!
```

A Cuban quintillo:

```
s "blip(5,8)"
```

Poly-meters:

This is a tough one to explain but listen and try to understand with the ear:

```
s "{bd ~ ~ ~ ~, cp ~ ~}%8"
```

Attaching functions together

Let's try to make a melody:

```
note "0 4 7 12 9 5"
```

This is a valid program but we are not telling Tidal what instruments to use to make this notes! So we need an instrument for our notes:

```
note "0 4 7 12 9 5" # s "ukulele"
```

Back to sound (instrument) as priority:

```
s "ukulele*8" # note "0 4 7 12 9 5"
```

Sound, then notes, and then low pass filter:

```
s "ukulele*8" # note "0 4 7 12 9 5" # cutoff "1200 650 300 200 150"
```

Sound, notes, low pass filter, and volume:

```
s "ukulele*8" # note "0 4 7 12 9 5" # cutoff "1200 650 300 200 150" # gain "0.4 0.7 1.0"
```

Functions that take more than patterns:

Slowing the pattern down:

```
slow 2 $ s "cp*4"
```

Notice how our `s "cp*4"` is bound to the `slow 2` by a very important operator: `$` !!!

Now also try `fast`.

Sometimes slower:

```
sometimes (slow 2) $...
```

Every third cycle do something:

```
every 3 (slow 2) $...
```

Play the orginal pattern and another modified pattern:

```
jux (slow 2) $ ...
```

Try `rev` within the `jux` function.

Some sounds you can explore:

In order to see available sounds in Estuary. Open a new tab with estuary on it, go to solo mode and input this in the terminal: `!localview audiomap`

808 (6 samples), 808bd (25 samples), 808cy (25 samples), 808hc (5 samples), 808ht (5 samples), 808lc (5 samples), 808lt (5 samples), 808mc (5 samples), 808mt (5 samples), 808oh (5 samples), 808sd (25 samples), 909 (1 samples), ab (12 samples),

acordeon (1 samples), ade (10 samples), ades2 (9 samples), ades3 (7 samples), ades4 (6 samples), alex (2 samples),alphabet (26 samples), altavoz (2 samples), amencutup (32 samples), armora (7 samples), arp (2 samples), arpy (11 samples), auto (11 samples), baa (7 samples), baa2 (7 samples), bajo (2 samples), bass (4 samples), bass0 (3 samples), bass1 (30 samples), bass2 (5 samples), bass3 (11 samples), bassdm (24 samples), bassfoo (3 samples), battles (2 samples), bd (24 samples), bend (4 samples), bev (2 samples), bin (2 samples), birds (10 samples), birds3 (19 samples), bleep (13 samples), blip (2 samples), blue (2 samples), bottle (13 samples), can (14 samples), casio (3 samples), cb (1 samples), cbow (18 samples), cc (6 samples), clak (2 samples), click (4 samples), clubkick (5 samples), co (4 samples), coffee (39 samples), coins (1 samples), contratiempos (3 samples), control (2 samples), cosmicg (15 samples), fm (17 samples), hand (17 samples), hardcore (12 samples), hardkick (6 samples), haw (6 samples), metal (10 samples), miniyeah (4 samples), monsterb (6 samples), moog (7 samples), mouth (15 samples), mp3 (4 samples), msg (9 samples), mt (16 samples), mute (28 samples), newnotes (15 samples), noise (1 samples), noise2 (8 samples), numbers (9 samples), oc (4 samples), odx (15 samples), off (1 samples), quinto (3 samples), rave (8 samples), reverbkick (1 samples), rm (2 samples), rs (1 samples), sax (22 samples), sd (2 samples), seawolf (3 samples), tech (13 samples), techno (7 samples), teclado (6 samples), tink (5 samples), tok (4 samples), toys (13 samples), trump (11 samples), tumba (2 samples), ukulele (14 samples), ul (10 samples), ulgab (5 samples), uxay (3 samples), v (6 samples), vgut (16 samples), voodoo (5 samples), wind (10 samples), wobble (1 samples), world (3 samples), xmas (1 samples), yeah (31 samples)...

Some functions that recieve a pattern as input similar to sound we can explore are:

`note "0 4 7"` , `speed "1 2 -1"`, `vowel "a e i o u"` , `n "0 1 2 3 4 3 5 2 1 0"` , `begin "0 0.5 0.9"`, `end "0.1 0.51 1"`, `pan "0 0.5 1"` , `gain "0.25 0.5 0.75 1"` , `cutoff "100 200 300 1000 5000 12000"` , `hcutoff "12000 5000 1000 300 200 100"`, etc...

Some fun programs to explore:

```
s "bd bd cp bd"
```

```
s "bd ~ cp [808:0 808:1]"
```

```
s "alphabet*4" # n "1 5 0 4"
```

```
s "drum(10,16)" # n (irand 20) # end 0.5
```

```
sometimes (slow 2) $ s "ukulele*8" # note "0 4 7 12 9 5" # cutoff "1200 650 300 200 150" # gain "0.4 0.7 1.0"
```

```
every 3 (#cutoff "1000 2000 1000") $ every 2 (slow 2) $ s "ukulele [ukulele? ukulele*2] ~ ukulele" # note "0 3 5 7 12"
```

```
every 3 (# vowel "a e o") $ every 2 (jux (stut 3 0.5 (1/16))) $ fast 2 $ s "glitch:3*2 808:1 industrial:3 cp*2" # pan "0 1 0.5 1 0"
```

```
ghost $ slow 4 $ struct "t(50,60)" $ chop 60 $ s "birds" # speed "1 1.5 0.5 -1 -2"
```