Tidal Cycles Reference Card

1 Mini Notation

Symbol	Description	Example	Equivalent
~	Rest	`d1 $ s "~hh"`	-
[ ]	Grouping	`d1 $ s "[bd sd] hh"`	`d1 $ fastcat [s "bd sd", s "hh"]`
[a, b]	Grouping	`d1 $ s "[bd sd, hh hh]"`	`d1 $ cat [s "bd sd", s "hh hh"]`
.	Shorthand grouping	`d1 $ s "bd sd . hh hh hh"`	`d1 $ s "[bd sd] [hh hh hh]`
,	Polyrhythm	`d1 $ s "[bd sd, hh hh hh]"`	`d1 $ stack [s "bd sd", s "hh hh hh"]`
*	Repeat (as group)	`d1 $ s "bd*2 sd"`	`d1 $ s "[bd bd] sd"`
!	Replicate	`d1 $ s "bd!3 sd"`	`d1 $ s "bd bd bd sd"`
/	Slow down	`d1 $ s "bd/2"`	`d1 $ s (slow 2 $ "bd")`
|	Random choice	`d1 $ s "[bd|cp|hh]"`	-
< >	Alternate	`d1 $ s "bd <sd hh cp>"`	`d1 $ slow 3 $ s "bd sd bd hh bd cp"`
_	Elongate	`d1 $ s "bd _ _ ~sd _"`	Result: `(0>1/2)|s: "bd" (4/6>1)|s: "sd"`
@	Elongate	`d1 $ s "superpiano@3`	`d1 $ s "superpiano _ _"`
?	Random removal	`d1 $ s "bd? sd"`	`d1 $ fastcat [degradeBy 0.5 $ s "bd", s "sd"]`
:	Sample selection	`d1 $ s "bd:3"`	`d1 $ s "bd" # n 3`
( )	Euclidean sequence	`d1 $ s "bd(3,8)"`	`d1 $ euclid 3 8 $ s "bd"`
{ }	Warp around	`d1 $ s "{bd bd bd, cp hh}"`	`d1 $ stack [s "bd*3", s "cp hh cp"]`
%	Warp around	`d1 $ s "{bd cp hh}%8""`	`d1 $ s "bd cp hh bd cp hh bd cp"`

2 Pattern Structure

Function	Both	Structure from Left	Structure from Right
Addition	|+| or +	|+	+|
Subtraction	|-| or -	|-	-|
Multiplication	|*| or *	|*	*|
Division	|/| or /	|/	/|
Modulo	|%| or %	|%	%|
Left values	|<| or <	|<	<|
Right values	|>| or >	|> or #	>|

2.1 Examples

Combining (both) structures is similar to slicing a cycle at intersection of both patterns, e. g. $3 = 3 \cdot 1/3$ and $4 = 4 \cdot 1/4$ results in $1/4, 1/12, 2/12, 2/12, 1/12, 1/4$

Code	Description	Equivalent
`n "1 2 2 2" |> n "1 1 3"`	Structure left, values right	`n "1 1 1 3"`
`n "1 2 2 2 2 2" # n "1 5 3"`	Structure left, values right	`n "1 1 5 5 3 3"`
`n "1 2 2 2" |< n "1 1 3"`	Structure left, values left	`n "1 2 2 2"`
`n "1 2 2 2" <| n "1 1 2"`	Structure right, values left	`n "1 2 2"`
`n "1 2 2 2" >| n "1 1 2"`	Structure right, values right	`n "1 1 2"`
`n "1 2 2 2" |>| n "1 1 2"`	Structure both, values right	`n "1 [1 1 _] [1 _ 2] 2"`
`n "1 2 2 2" |<| n "1 1 2"`	Structure both, values right	`n "1 [2 2 _] [2 _ 2] 2"`

3 Oscillators

Oscillators are continuous patterns, which means they don't have any structure, and must be used with a pattern that does. They usually spit out values between 0 and 1.

OSC	Example	Description
sine	`d1 $ s "bd*8" # pan` **sine**	Sine wave
cosine	`d1 $ s "bd*8" # pan` **cosine** `# speed (sine + 0.5)`	Cosine wave
square	`d1 $ s "bd*8" # pan (cat [`**square**`, sine])`	Square wave
tri	`d1 $ s "bd*16" # speed (slow 2 $ range 0.5 2` **tri**`)`	Triangle wave
saw	`d1 $ s "bd*8" # pan (slow 2` **saw**`)`	Sawtooth wave
isaw	`d1 $ s "bd*8" # pan (slow 2` **isaw**`)`	Inverted Sawtooth wave
smooth p	`d1 $ s "bd*4" # pan (slow 4 $` **smooth** `"0 0.5 1")`	Linear Interpolation
rand	`d1 $ s "bd*8" # pan` **rand**	random numbers in $[0; 1]$
irand n	`d1 $ s "drum*8" # n (`**irand 8**`)`	random integers in $[0; n]$

4 Sampling

Function	Notation	Description
chop n_p p	`chop "16 4" $ s "xmas xmas"`	Cut into n_p parts
striate n_p p	`striate 16 $ n "1 2 3" # s "cpu"`	& interleave
striateBy n_p d_p p	`striateBy 16 "<0.1 0.01>" $ s "cpu:1 cpu:2"`	& d_p controls grain length
slice n_p m_p p	`slice 8 "0 2 4" $ s "breaks125"`	Cut & arrange via m_p
splice n_p m_p p	`splice 8 "0 2 4" $ s "breaks125"`	& auto pitch
randslice n_p p	`randslice 8 $ s "breaks125"`	Cut & arrange randomly
bite n m_p p	`bite 5 "1 [1 4]" $ s "breaks125" # cut 1`	Cut **cycle** & mini notation m_p
chew n m_p p	`chew 5 "1 [1 4]" $ s "breaks125" # cut 1`	& auto pitch
loopAt t_p p	`loopAt "<1 0.5>" $ s "breaks125"`	auto pitch to fit cycle number
smash n $[t_p]$ p	`smash 3 [2,3,4] $ s "ho ho:2 ho:3 hc"`	spread & striate
smash' n $[t_p]$ p	`smash' 3 [2,3,4] $ s "ho ho:2 ho:3 hc"`	spread & chop
segment t_p s	`lpf (segment 16 $ range 200 400 $ sine)`	discretize **signal** (t_p per cycle)

5 Alteration

Function	Notation	Description
range n_p m_p s	`lpf (range 200 5000 $ sine)`	Scaling in $[n; m]$
xrange n_p m_p s	`lpf (xrange 200 5000 $ sine)`	Exp. scaling in $[n; m]$
quantise r $[q]$	`quantise 5 [0, 1.3 ,2.6,3.2,4.7,5]`	Quantise all q to multiple of $1/r$
degradeBy q p	`degradeBy 0.9 $ s "bd*5"`	Removes with a prob. q
degrade p	`degrade $ s "bd*5"`	Removes with a prob. 0.5
unDegradeBy q p	`unDegradeBy 0.1 $ s "bd*5"`	Removes with a prob. $(1 - q)$
ply n_p p	`ply 3 $ s "bd ~ sn cp"`	Repeat n_p times within a cycle
stutter n d p	`stutter 4 (1/16) $ s "bd cp"`	Repeat n times & seperate by d cycles
echo n d r p	`echo 4 0.2 0.5 $ s "bd sn"`	& make each r times quieter/loude
slowstripe n p	`slowstripe 3 $ s "bd sd [mt ht]"`	& avg. pattern length is one cycle
palindrome p	`palindrome $ n "1 2 3" # s "cpu"`	Reverse every other cycle
trunc t_p p	`trunc "<0.75 0.25>" $ s "bd sn*5"`	Truncates a patttern (rests at the end)
linger t_p p	`linger 0.25 $ n "0 2 1" # s "arpy"`	& but fill cycle by repeating
chunk n f p	`chunk 4 (# speed 2) $ s "bd hh sn cp"`	Chunk p, apply f to one chunk
chunk' n f p	-	& reverse cycling
loopFirst p	-	Loop only the first cycle of the p
bite n m_p p	`bite 5 "1 [1 4]" $ s "breaks125" # cut 1`	Cut **cycle** & mini notation m_p
shuffle n p	-	Random permutation n parts of p
scramble n p	-	random selection of n parts of p
rot n_p o_p	-	rotates n_p times

6 Time

Function	Notation	Description
fast t_p p	`(fast "2 4" "bd sn hh hh")`	Speed up p
fastGap t_p p	`(fastGap "2 4" "bd sn hh hh")`	& but insert rest
hurry t_p p	`(fast "2 4" "bd sn hh hh")`	& Speed up control
fastSqueeze t_p p	`fastSqueeze "2 4" $ s "bd*8"`	speed up p & Squeeze into one cycle
slow t_p p	`(slow "2 4" "bd sn hh hh")`	Slow down p
slowSqueeze t_p p	`slowSqueeze "2 4" $ s "bd*8"`	slow down p & Squeeze into one cycle
compress (t_1, t_2) p	`compress (1/4, 3/4) $ s "[bd sn]!"`	Squeeze (by speeding up) p into $[t_1; t_2]$
zoom (t_1, t_2) p	`zoom (1/4, 3/4) $ s "[bd sn]!"`	Squeeze (by cutting) p into $[t_1; t_2]$
within (t_1, t_2) f p	`within (1/4, 3/4) (fast 2) $ s "bd*8"`	Apply f within $[t_1; t_2]$ of p
stretch p	`stretch "  0 1 5 8*4  "`	trim rests from p
off t_p f p	`off 0.125 (# speed 2) $ "bd*4 sn"`	apply f to p & layer it on top
pressBy t_p p	`pressBy 0.3 $ "bd sn hh"`	Add a rest of length t_p before each part
press p	`press $ "bd sn hh"`	press 0.5
rotL t p	`rotL 3 $ s "bd sn hh"`	Shift pattern back in time
$(d$ <~$)$ p	`(0.125 <~) $ s "bd sn hh"`	rotL d p
rotR t p	`rotR 3 $ s "bd sn hh"`	Shift pattern forward in time
$(d$ ~>$)$ p	`(0.125 ~>) $ s "bd sn hh"`	rotR d p
spin d	`spin 3 $ "bd sn hh"`	Copy & successive offset by $(1/d)$ & pan
weave c p_{c_1} $[p_{c_2}]$	`weave 3 (pan sine) $ [s "bd sn", s "hh"]`	Apply p_{c_1} to p_{c_2} with successive offset
weaveWith c p_c $[f]$	`weaveWith 3 (s "bd hh") [fast 2, chop 16]`	Apply f's to p_c with successive offset
rev p	`rev "1 [~ 2] ~ 3"`	Reverse p
swingBy d n p	`swingBy (1/3) 4 $ "hh*8"`	Cut into n slices & swing them
swing n p	`swing 4 $ "hh*8"`	swingBy 0.5 n p
ghost p	`ghost $ s "  sn"`	Add ghost note (drum)
ghost' d p	`ghost' $ s "  sn"`	& define delay d

ghostWith d f p	ghost' $ s " sn"	& modify ghosts with f
inside t_p f p	inside 2 (rev) "0 1 2 3"	Apply f inside a cycle split in t_p
outside t_p f p	outside 2 (rev) $ cat [s "0 1", s "2 3"]"	Apply f over t_p cycles
echo n_p r_p d_p p_c	echo 4 0.2 0.5 $ s "bd sn"	Echo with depth n_p, delaytime r_p, feedback d_p
echoWith n_p r_p f p_c	echoWith 4 0.2 (|* speed 1.5) $ s "bd sn"	Like echo but instead of vol decrease apply f

7 Panning

Function	Notation	Description
jux f p	jux (rev) $ s "bd sn hh hh"	Apply f to p but in right channel
juxBy d f p	juxBy 0.3 (rev) $ s "bd sn hh hh"	Like jux but d controls panning

8 Concatenation

Function	Notation	Description
cat [p]	cat [s "bd*2 sn", s "arpy jvbass*2"]	Concat and keep duration
fastcat [p]	fastcat [s "bd*2 sn", s "arpy jvbass*2"]	Concat but squeeze into one cycle
timeCat [(t, p)]	timeCat [(1, s "bd*2 sn"), (0.5, s "arpy jvbass*2")]	& but squeeze proportional
randcat [p]	randcat ["bd*2 sn", "arpy jvbass*2"]	Like cat but order randomly
wrandcat [(p, q)]	wrandcat [("bd*2 sn",0.9), ("arpy jvbass*2",0.1)]	Like randcat but weighted by q
overlay p_1 p_2	overlay "bd*2 sn" "arpy*2"	"[bd sn:2, cp*3]"
p_1 <> p_2	"bd*2 sn" <> "arpy*2"	"[bd sn:2, cp*3]"
append p_1 p_2	append (s "bd*2 sn") (s "arpy jvbass*2")	Like cat
fastAppend p_1 p_2	fastAppend (s "bd*2 sn") (s "arpy jvbass*2")	Like fastcat
wedge d p_1 p_2	wedge 0.3 (s "bd*2 sn") (s "arpy jvbass*2")	Like fastAppend but d controls the fill-ratio
brak p	brak $ s "[feel feel:3, hc:3 hc:2 hc:4 ho:1]"	Pattern into breakbeat
listToPat [a]	listToPat [0, 1, 2, 3]	Transforms a list into a pattern
fromList [a]	fromList [0, 1, 2, 3]	& each item represents one cycle
fromMaybes [a]	-	-
flatpat [a]	-	-
run n_p	n (run 8)	generates a pattern of one cycle 0 to $n_p - 1$
scan n_p	n (scan 8)	generates a pattern of n_p cycles 1 to n_p

9 Accumulation

Function	Notation	Description
stack [p]	stack [s "bd*2 sn", s "arpy*2"] # speed 2	Polyrhythm
superimpose f p	superimpose (fast 2) $ s "bd sn"	Add modified version
layer [f] p	layer [fast 2, rev] $ s "bd sn"	Add multiple modified versions
steps [str_1, str_2]	-	-
iter n_p p	iter 4 $ s "bd hh sn cp"	divide pattern into n_p parts and shift over n_p cycles
iter' n_p p	iter' 4 $ s "bd hh sn cp"	Like iter but other direction

10 Conditions
11 Harmony & Melody
12 Performance
13 Transitions
14 Samplers
15 Randomness
16 Composition
17 mi-UGens
18 Control Busses