print(SynthDefs)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| loop | growl | gong | creep | pluck |
| gsynth | bass | soprano | orient | spark |
| play1 | dirt | dub | zap | fuzz |
| play2 | crunch | viola | marimba | bug |
| audioin | rave | scratch | feel | pulse |
| noise | scatter | klank | glass | saw |
| dab | charm | blip | soft | snick |
| varsaw | bell | ripple | quin | twang |
| lazer |  |  |  |  |

print(Samples)

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| '!' | Yeah! | '=' | Hi hat open | 'M' | Acoustic toms | '\\' | Lazer | 'm' | 808 toms |
| '#' | Crash | '@' | Gameboy noise | 'N' | Gameboy SFX | '^' | 'Donk' | 'n' | Noise |
| '$' | Beatbox | 'A' | Gameboy kick drum | 'O' | Heavy snare | 'a' | Gameboy hihat | 'o' | Snare drum |
| '%' | Noise bursts | 'B' | Short saw | 'P' | Tabla long | 'b' | Noisy beep | 'p' | Tabla |
| '&' | Chime | 'C' | Choral | 'Q' | Electronic stabs | 'c' | Voice/string | 'q' | Ambient stabs |
| '\*' | Clap | 'D' | Dirty snare | 'R' | Metallic | 'd' | Woodblock | 'r' | Metal |
| '+' | Clicks | 'E' | Ringing percussion | 'S' | Tamborine | 'e' | Electronic Cowbell | 's' | Shaker |
| '-' | Hi hat closed | 'F' | Trumpet stabs | 'T' | Cowbell | 'f' | Pops | 't' | Rimshot |
| '/' | Reverse sounds | 'G' | Ambient stabs | 'U' | Misc. Fx | 'g' | Ominous | 'u' | Soft snare |
| '1' | Vocals (One) | 'H' | Clap | 'V' | Hard kick | 'h' | Finger snaps | 'v' | Soft kick |
| '2' | Vocals (Two) | 'I' | Rock snare | 'W' | Distorted | 'i' | Jungle snare | 'w' | Dub hits |
| '3' | Vocals (Three) | 'J' | Ambient stabs | 'X' | Heavy kick | 'j' | Whines | 'x' | Bass drum |
| '4' | Vocals (Four) | 'K' | Percussive hits | 'Y' | High buzz | 'k' | Wood shaker | 'y' | Percussive hits |
| '-' | Hi-hats | 'L' | Noisy percussive hits | 'Z' | Loud stabs | 'l' | Robot noise | 'z' | Scratch |
|  |  |  |  |  |  |  |  | '|' | Hangdrum |
|  |  |  |  |  |  |  |  | '~' | Ride cymbal |

print(Player.get\_attributes())

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| degree | sample | sus | rate | chop |
| oct | env | slide | pshift | tremolo |
| freq | sus | slidedelay | hpf | beat\_dur |
| dur | fmod | slidefrom | hpr | echo |
| delay | pan | bend | lpf | echotime |
| buf | rate | benddelay | lpr | spin |
| blur | amp | coarse | swell | cut |
| amplify | midinote | striate | bpf | room |
| scale | channel | buf | bpr | mix |
| bpm | vib |  | bpnoise | formant |
|  | vibdepth |  |  | shape |

every(n, method\_name, \*args, \*\*kwargs)

Use to call a method every n number of beats. Specify the method name as a string and then the arguments and keyword arguments to supply the method. See Algorithmic Manipulation for more in-depth information on every.

after(n, method\_name, \*args, \*\*kwargs)

Similar to every but only calls the method once after n number of beats. This might be useful for only looping a sequence for a certain amount of time.

# Stop a drum loop after 8 beats

d1 >> play("x-o-").after(8, "stop")

stutter(n, \*args, \*\*kwargs)

Repeat the last musical event n times. You can specify keyword arguments as you would normally update a Player to control the stutter. The dur keyword changes the duration over which to spread the stuttered events (defaults to the Player’s current duration value).

jump(ahead=1, \*\*kwargs)

Plays the event that’s a number of steps ahead of the player’s current position as specified by the ahead argument. This is best used in conjunction with the every method

# Plays the snare drum half a beat early after 6.5 beats

d1 >> play("x-o-").every(6.5, "jump", cycle=8)

degrade(amount=0.5)

Sets the amplitude to be chosen from 1 and 0 at random. The amount argument is the likelihood of the amplitude being 0 as a fraction i.e. a degrade of 0.5 (default) means each event is 50% likely to be a 0 and at 0.1 it is 10% likely to be a 0.

# Start a player

p1 >> pluck()

# Start to decrease the number of events

p1.degrade()

# Can be called repeatedly to degrade by 25% every 8 beats

p1.every(8, "degrade", 0.25)

reverse()

Reverses the order of all the attributes at the current time. This method does not reverse the lists of values but reverses the order in which they are used.

ollow(player)

Follows the pitch of player. It is equivalent to setting the pitch of one player, e.g. p1, to another’s pitch using p2.pitch.

# Start a player

p1 >> pluck([0, 1, 2, 3], dur=2)

# Follow the pitch in p2

p2 >> blip().follow(p1)

# Add values to change the pitch

p2 >> blip().follow(p1) + [0, 2]

accompany(player)

Similar to follow, this method forces the player to play notes that are derived from the pitch of another Player. The pitch will be the closest neighbour to the last pitch used that is the new pitch of the source player plus 0, 2, or 4.

# Start a player

p1 >> pluck([0, 1, 2, 3], dur=2)

# Play accompanying pitches

p2 >> blip().accompany(p1)

# Add values to change the pitch

p2 >> blip().accompany(p1) + [0, 2]