Live-Coding con Python y FoxDot

por **Guillermo Ambrosio**
para el **CGSOL 2020**

Live-Coding

Programación en vivo

https://colectivo-de-livecoders.gitlab.io/

Grupo de Telegram:
https://t.me/clic_livecoding

FoxDot

Librería para hacer música algorítmica. Es una capa de abstracción sobre Supercollider.

https://github.com/Qirky/FoxDot/

https://foxdot.org/docs/

Para instalar:

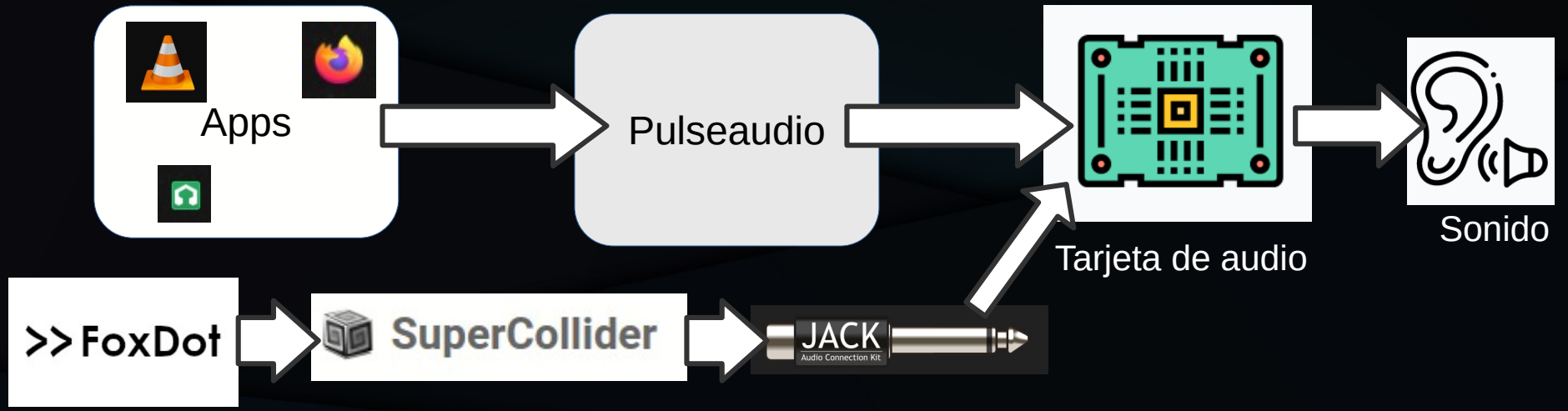
Instalar paquetes:

```
$ sudo apt install qjackctl supercollider-ide python-pip pulseaudio-module-jack
```

Instalar Foxdot: https://foxdot.org/installation/

Audio en Linux

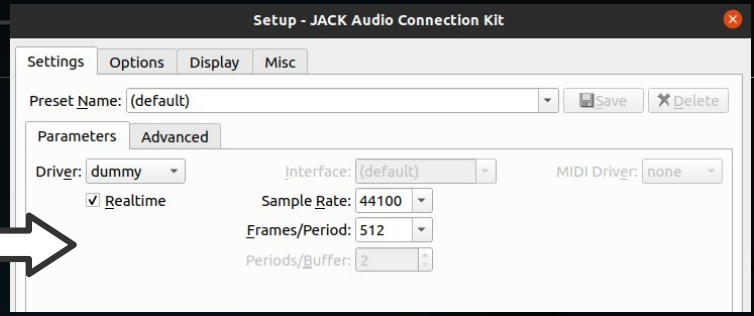
En mi caso, en el sistema Ubuntu, tengo este escenario:



Jack y Pulseaudio son sistemas de audio para Linux que toman control completo de la tarjeta de audio. Para evitar conflicto con Pulseaudio, usualmente este se suspende mientras Jack corre. También se puede usar una tarjeta de audio USB o se puede conectar jack con pulseaudio.

Antes de usar Foxdot:

Iniciar **qjackctl**, configurar con tarjeta de audio “dummy” y rate de 44100, iniciar jackd



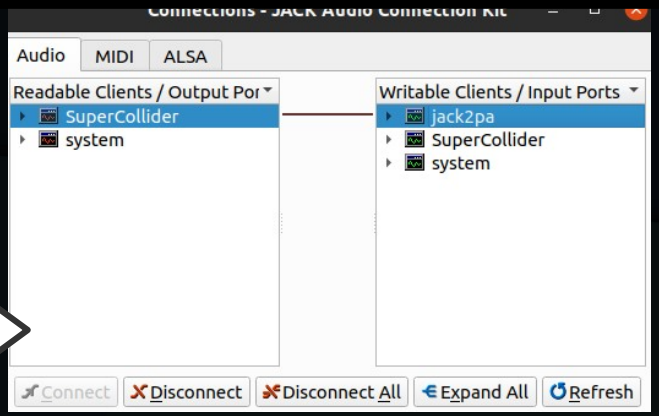
Crear un *source* para conectar jack a pulseaudio y activar en ese *source* el módulo loopback:

```
$ pacmd load-module module-jack-source channels=2 source_name=jack2pa client_name=jack2pa connect=false
$ pactl load-module module-loopback latency_msec=10 source=jack2pa sink=0 source_dont_move=true
```

Iniciar **scide**, configurar rate a 44100 e iniciar Foxdot (se usa Shift+Enter para ejecutar cada línea):

```
Server.local.options.sampleRate=44100
FoxDot.start;
```

Conectar SuperCollider al puerto jack2pa que se creó con **pacmd**



Usando FoxDot

Se puede utilizar **`python -m FoxDot`**, **`ipython`**, **`Spyder`** o incluso **`jupyter lab.`**

En python se debe importar la librería FoxDot:

```
from FoxDot import *
```

Tiempo

FoxDot tiene un objeto **Clock** que lleva cuenta del tiempo. Con este objeto se puede configurar los beats por minuto (la velocidad),

```
>>> Clock.bpm = 120
```

el compás (4/4 por defecto),

```
>>> Clock.meter = (3,4) # Compás de 3/4
```

se puede programar tareas que se ejecutarán en el futuro y otras cosas.

Este código programa la llamada de una función para el próximo inicio de compás. La función cambia la escala y la tonalidad de todos los instrumentos.

```
def change():
    Scale.default = "minor"
    Root.default = "C"

Clock.schedule(change, Clock.next_bar())
```

Escalas y tonalidad

FoxDot tiene la estructura de varias escalas. El objeto Scale permite modificar la escala por defecto:

```
>>> Scale.default = “dorian”
```

El objeto Root permite modificar la tonalidad por defecto:

```
>>> Root.default = “D”
```

Todas las notas que se usarán para los Players seguirán esta tonalidad y escala por defecto. En este caso, se ha puesto la tonalidad de Re y la escala menor.

Al configurar esta escala y tonalidad, los números de las notas irían así:

Número	Nota
0	Re
1	Mi
2	Fa
3	Sol
4	La
5	Si
6	Do

Players

Los objetos Player ejecutan las notas. Todas las combinaciones de 2 símbolos alfanuméricos están reservadas para Players (aa, a1, bd, xy, yz, …). A los players se les pone un sintetizador con el operador >>.

```
>>> aa >> pads([0,2,4,6,7])
```

En este caso, aa estará tocando en un sintetizador “pads” las notas (en Do mayor) Do, Mi, Sol, Si, Do, en negras.

Se puede alternar notas con corchetes [ ], se puede hacer varias notas a la vez con paréntesis ( ).

Ritmos de percusión

Las percusiones se tocan con el sintetizador “play” utilizando cadenas de texto. Cada sonido disponible por defecto está asociado a un carácter.

Cada caracter sería una figura (por defecto negras). Se pueden hacer subdivisiones con corchetes, por ejemplo esto “[xxx]” divide una negra en tres partes iguales (tresillo de corchea). Se puede alternar así: “X(H:H:)” (esto haría esta secuencia: Kick, Clap, Kick, Hihat, Kick, Clap, Kick, Hihat)

```
>>> dd >> play(“guille”)
```

```
>>> print(Samples)
```

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

Patrones

Los patrones son objetos que contienen secuencias de números. Esos números pueden usarse como notas, para melodía y armonía, o pueden usarse para cualquier otro parámetro que cambie en el tiempo. Con P[ ] se pueden declarar patrones. Estos objetos permiten hacer patrones complejos como por ejemplo:

```
>>> P[0,1,2,3].stutter(2)
Out: P[0, 0, 1, 1, 2, 2,
3, 3]
```

```
>>> P[0,1,2,3].zip([0,5])
Out: P[P(0, 0), P(1, 5), P(2, 0),
P(3, 5)]
```

```
>>> P[0,1,2].arp([0,3,6])
Out: P[0, 3, 6, 1, 4, 7, 2,
5, 8]
```

```
>>> bb >> jbass(P[0,1,2].arp([0,3,4]),
dur = P[1/4,[1/2, 1/4]])
```

Variables dependientes del tiempo

FoxDot tiene variables que varían conforme el tiempo pasa. La música es un flujo de sonidos que cambian conforme pasa el tiempo. Por ejemplo, una progresión de acordes cambia conforme pasa el tiempo.

Para repetir secuencias de números se puede utilizar la función “var”:

```
>>> var([0, 4, 3, 1], dur = [4,1])
```

Esto repetiría una secuencia de 4 números, cada número tendría una duración de 4 figuras (por defecto negras).

Hay funciones continuas también, como linvar y expvar.

Por ejemplo, para hacer esta progresión de acordes en negras se puede hacer esto:

```
>>> progresion = var([0,3,3,4], dur=4)
>>> a1 >> pads(progresion + (0,2,4), dur=1/2)
```

Sintetizadores y Efectos

Los sintetizadores son los objetos que producen los sonidos. Para ver los sintetizadores disponibles hacer:

```
>>> print(SynthDefs)
```

Para evitar el trabajo largo de desarrollar un sintetizador de sonido en tiempo real, FoxDot trae ya implementados varios sintetizadores. Estos están hechos en supercollider. Python sólo se encarga de enviar las notas a supercollider para que éste las ejecute.

Los sintetizadores pueden recibir parámetros para configurar efectos. Estos pueden verse haciendo:

```
>>> print(FxList)
```

```
<'bandPassFilter': keyword='bpf', other args=['bpr', 'bpnoise', 'sus']>
<'bitcrush': keyword='crush', other args=['bits', 'sus', 'amp']>
<'chop': keyword='chop', other args=['sus']>
<'coarse': keyword='coarse', other args=['sus']>
<'combDelay': keyword='echo', other args=['beat_dur', 'echotime']>
<'distortion': keyword='dist', other args=['tmp']>
<'filterSwell': keyword='swell', other args=['sus', 'hpr']>
<'formantFilter': keyword='formant'>
<'glissando': keyword='glide', other args=['glidedelay', 'sus']>
<'highPassFilter': keyword='hpf', other args=['hpr']>
<'lowPassFilter': keyword='lpf', other args=['lpr']>
<'overdriveDistortion': keyword='drive'>
<'pitchBend': keyword='bend', other args=['sus', 'benddelay']>
<'pitchShift': keyword='pshift'>
<'reverb': keyword='room', other args=['mix']>
<'slideFrom': keyword='slidefrom', other args=['sus', 'slidedelay']>
<'slideTo': keyword='slide', other args=['sus', 'slidedelay']>
<'spinPan': keyword='spin', other args=['sus']>
<'striate': keyword='striate', other args=['sus', 'buf', 'rate']>
<'tremolo': keyword='tremolo', other args=['beat_dur']>
<'trimLength': keyword='cut', other args=['sus']>
<'vibrato': keyword='vib', other args=['vibdepth']>
<'wavesShapeDistortion': keyword='shape'>
```