A Mini DAW in the Python REPL

March 2026

8 min read • 1,758 words

So that music theory library I just wrote about — the one that identifies chords and models fretboards and knows about Arabic maqam? It grew. It grew a lot. Somewhere between "what chord am I playing" and "let me just add a simple playback function," PyTheory became something I didn't plan for.

It became a DAW.

Not a full DAW. Not Ableton, not Logic, not even GarageBand. But a thing where you can open a Python REPL at 2am, type a few lines, and have a beat playing through your speakers with bass, pads, effects, and automation. No GUI. No mouse. No timeline to click on. Just code and sound.

Let me show you.

Start with drums

```
from pytheory import Score, Pattern, play_score

score = Score("4/4", bpm=140)
score.drums("bossa nova", repeats=4)
play_score(score)
```

That's a bossa nova beat, synthesized from scratch, playing right now. No samples. No WAV files loaded from disk. Every drum sound — kick, snare, hi-hat, everything — is generated mathematically. Sine waves shaped by envelopes, noise filtered through resonant bands, FM synthesis for metallic textures.

There are 58 drum presets. Fifty-eight. Rock, jazz, funk, salsa, techno, drum and bass, reggae, waltz, samba, breakbeat, afrobeat, shuffle, half-time, trap, lo-fi hip hop. Each one is a pattern of 27 synthesized drum sounds: kick, snare, closed hi-hat, open hi-hat, ride, crash, low tom, mid tom, high tom, rimshot, clap, cowbell, shaker, tambourine, clave, low conga, mid conga, high conga, bongo low, bongo high, agogo, cabasa, guiro, timbale, woodblock, triangle, and cross-stick.

Twenty-seven instruments, all built from waveforms and envelopes. No sample packs. No audio files. Just math pretending to be a drum kit, and doing a surprisingly convincing job.

There are also 21 drum fills — short, long, half-bar, crash fills, breakdowns, build-ups — so your four-bar loop doesn't have to just loop. It can breathe.

```
score = Score("4/4", bpm=140)
score.drums("funk", repeats=3)
score.drums("funk", fill="crash", repeats=1)  # fill on the 4th bar
play_score(score)
```

Already sounds like something. But drums alone are just a rhythm. Let's give it a body.

Add a bass line

```
bass = score.part(
    "bass",
    synth="sine",
    envelope="pluck",
    volume=0.6,
    lowpass=800,
)

bass.add("A2", Duration.HALF).add("D2", Duration.HALF)
bass.add("E2", Duration.HALF).add("A2", Duration.HALF)
```

Now we have a bass playing a simple i-iv-v-i movement underneath the drums. The `synth="sine"` gives us a clean, round tone. The `envelope="pluck"` shapes it — fast attack, quick decay, short sustain, fast release. The `lowpass=800` rolls off the highs so it sits where bass should sit: underneath everything, felt as much as heard.

There are 8 envelope presets: piano, organ, pluck, pad, strings, bell, staccato, and a default. Each one shapes the amplitude over time in a way that suggests its name. Piano has a sharp attack and a long natural decay. Organ has instant attack and sustain for as long as you hold the note. Pad fades in slowly and hangs in the air. Bell rings and shimmers away. Staccato is there and gone. Each preset is an ADSR curve — attack, decay, sustain, release — and they transform the same raw waveform into completely different instruments.

Layer a pad

```
pad = score.part(
    "pad",
    synth="supersaw",
    envelope="pad",
    volume=0.3,
    reverb=0.6,
    reverb_decay=2.0,
    chorus=0.4,
)

pad.add("Am", Duration.WHOLE).add("Dm", Duration.WHOLE)
```

Notice something: I passed a chord name, not a note. `"Am"` is A minor. The pad part knows what to do with that — it plays all the notes of the chord simultaneously. One line, full harmony.

The `supersaw` synth is what makes it lush. It's multiple detuned sawtooth waves stacked together, the classic trance/ambient texture. Combine that with the `pad` envelope (slow attack, full sustain), reverb with a 2-second decay, and a touch of chorus, and you have something that fills the entire stereo field with warmth.

There are 10 synth waveforms: sine, saw, triangle, square, pulse, FM, noise, supersaw, PWM slow, and PWM fast. Each one has its own harmonic character. Sine is pure and clean — one frequency, nothing else. Saw is bright and buzzy, full of harmonics. Square is hollow and woody. FM synthesis gives you those glassy, bell-like tones that defined 80s pop. Noise is useful for percussion and texture. Supersaw is instant atmosphere. PWM (pulse width modulation) gives you that slow, shifting, slightly queasy analog quality.

All of them are generated sample by sample. No wavetables, no pre-rendered buffers. Raw math, in Python, producing audio.

The effects chain

Every part in a score has its own effects chain: distortion, chorus, lowpass filter, delay, and reverb. They process in that order. You set them as parameters when you create the part, or change them later.

```
lead = score.part(
    "lead",
    synth="saw",
    envelope="pluck",
    reverb=0.3,
    delay=0.25,
    delay_time=0.375,
    lowpass=2000,
)

lead.add("E4", Duration.QUARTER)
lead.add("G4", Duration.QUARTER)
lead.add("A4", Duration.QUARTER)
lead.add("E5", Duration.QUARTER)
```

That `delay_time=0.375` syncs the echoes to dotted eighths at 140 BPM. Which is one of those small things that makes a massive difference — when delays lock to the tempo, everything suddenly sounds professional instead of amateur. The lowpass at 2000 Hz takes the harshness off the raw sawtooth and pushes the lead slightly behind the mix, like it's playing from the next room.

You can stack as many effects as you want on a single part, and each part has its own independent chain. The bass can be dry and clean while the pad drowns in reverb. The lead can have delay while the arpeggiator stays tight and present. Mix engineering, from the REPL.

Automation

Static mixes are boring. Real music moves. Parameters change over time — filters open, reverb swells, things breathe. PyTheory handles this with mid-song parameter changes:

```
lead.add("E5", Duration.WHOLE)
lead.set(lowpass=3000, reverb=0.4)  # filter opens up
lead.add("G5", Duration.WHOLE)
lead.set(lowpass=5000, reverb=0.6)  # opens further
lead.add("A5", Duration.WHOLE)
```

Each call to `.set()` changes the parameters from that point forward. So the filter opens progressively as the melody climbs. It's simple, it's linear, and it's effective.

But if you want something more organic, there's LFO automation:

```
lead.lfo(
    "lowpass",
    rate=0.125,
    min=400,
    max=3000,
    bars=8,
    shape="sine",
)
```

That sweeps the lowpass filter between 400 Hz and 3000 Hz over 8 bars, following a sine wave curve. The rate controls how many cycles per bar. The shape can be sine, triangle, saw, or square — different curves for different vibes. Sine is smooth and hypnotic. Saw ramps up and snaps back. Square toggles abruptly between min and max, which sounds more dramatic than you'd expect.

LFO on a filter is the oldest trick in electronic music and it still works every time. Something about the cyclical opening and closing of harmonics — it's like the sound is breathing.

The arpeggiator

This is where it starts to feel like a real instrument.

```
arp = score.part(
    "arp",
    synth="square",
    envelope="pluck",
    legato=True,
    glide=0.05,
)

arp.arpeggio(
    "Cm",
    bars=2,
    pattern="updown",
    division=Duration.SIXTEENTH,
    octaves=2,
)
```

One line. That generates a C minor arpeggio rippling up and down over two octaves in sixteenth notes for two bars. The `legato=True` means notes connect without retriggering the envelope. The `glide=0.05` adds a 50-millisecond portamento between notes — a subtle pitch slide that makes the whole thing feel liquid instead of mechanical.

The `pattern` parameter controls the direction: up, down, updown, downup, random. The `division` controls note length. The `octaves` parameter controls how wide the arpeggio spreads. Combine a two-octave updown arpeggio at sixteenth notes with a synced delay and some reverb, and you have the backbone of half the electronic music ever made.

```
arp = score.part(
    "arp",
    synth="square",
    envelope="pluck",
    legato=True,
    glide=0.05,
    delay=0.3,
    delay_time=0.375,
    reverb=0.4,
    lowpass=3500,
)

arp.arpeggio("Cm", bars=2, pattern="updown",
             division=Duration.SIXTEENTH, octaves=2)
arp.arpeggio("Fm", bars=2, pattern="updown",
             division=Duration.SIXTEENTH, octaves=2)
```

Four bars. Two chords. Sounds like a track.

Put it all together

Here's the whole thing, start to finish:

```
from pytheory import Score, Duration, play_score

score = Score("4/4", bpm=140)

# Drums
score.drums("bossa nova", repeats=3)
score.drums("bossa nova", fill="crash", repeats=1)

# Bass
bass = score.part("bass", synth="sine", envelope="pluck",
                  volume=0.6, lowpass=800)
bass.add("A2", Duration.HALF).add("D2", Duration.HALF)
bass.add("E2", Duration.HALF).add("A2", Duration.HALF)

# Pad
pad = score.part("pad", synth="supersaw", envelope="pad",
                 volume=0.3, reverb=0.6, reverb_decay=2.0, chorus=0.4)
pad.add("Am", Duration.WHOLE).add("Dm", Duration.WHOLE)

# Lead
lead = score.part("lead", synth="saw", envelope="pluck",
                  reverb=0.3, delay=0.25, delay_time=0.375, lowpass=2000)
lead.add("E4", Duration.QUARTER).add("G4", Duration.QUARTER)
lead.add("A4", Duration.QUARTER).add("E5", Duration.QUARTER)
lead.set(lowpass=3000, reverb=0.4)
lead.add("G5", Duration.WHOLE)

# Arpeggiator
arp = score.part("arp", synth="square", envelope="pluck",
                 legato=True, glide=0.05, delay=0.3, reverb=0.4)
arp.arpeggio("Am", bars=2, pattern="updown",
             division=Duration.SIXTEENTH, octaves=2)
arp.arpeggio("Dm", bars=2, pattern="updown",
             division=Duration.SIXTEENTH, octaves=2)

play_score(score)
```

That's a track. Drums, bass, pad, lead melody with filter automation, and a two-octave arpeggio with portamento. Under 30 lines. Playing through your speakers right now.

Export

When you have something worth keeping:

```
save("track.wav")
```

That's a lossless WAV file on your disk. 44.1 kHz, 16-bit, stereo. Real audio. You can drop it into an actual DAW for further processing, upload it somewhere, or just listen to it.

Or if you want to take your ideas into another tool:

```
score.save_midi("track.mid")
```

MIDI export preserves the notes, durations, velocities, and channels. Open it in Ableton, Logic, FL Studio, whatever you use — your REPL sketch becomes the starting point for a full production.

The REPL is the instrument

This started as a music theory library. A thing for identifying chords and exploring scales. I added playback because I wanted to hear what I was modeling. Then I added synthesis because I didn't want to depend on external audio. Then I added drums because rhythm is fundamental. Then effects, because dry synths sound lifeless. Then automation, because static sounds are boring. Then arpeggiation, because patterns are beautiful. Then export, because ideas deserve to be saved.

Each step was small. Each step made sense. And then I looked up and realized I'd built a thing where you can type 15 lines of Python and have music playing through your speakers. Not a beep. Not a sine wave. A song — with drums and bass and harmony and effects and movement.

No DAW. No GUI. No mouse. No timeline. No piano roll. Just a blinking cursor and the entire expressive range of a synthesizer, a drum machine, and a mixing board, all accessible through the same interface I use to write web servers and parse JSON.

The REPL is the instrument.

I keep finding that the most interesting things I build are the ones I didn't plan. PyTheory was supposed to answer "what chord is this?" and now it's a place where I sketch musical ideas in code at 2am, export them, and wake up surprised by what I made. The library grew because each new capability felt natural — like it was already implied by the architecture and just needed to be written down.

That's the feeling I chase in software. Not "what feature should I add next" but "what does this thing already want to become." PyTheory wanted to make sound. I just followed where it led.

`pip install pytheory`. Open a REPL. Start with drums. See where it takes you.
