



A Mini DAW in the Python REPL

MARCH 2026

3 min read • 732 words

[That music theory library I wrote about](#) kept growing. I added playback because I wanted to hear what I was modeling. Then synthesis because I didn't want external dependencies. Then drums, then effects, then automation. Each step was small and made sense at the time.

And now it's... kind of a DAW? A weird one. A tiny one. But it works, and I keep using it, so here's what happened.

Drums

```
from pytheory import Score, Pattern, play_score

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

Every drum sound is synthesized from scratch — no samples, no WAV files. Sine waves shaped by envelopes, noise through resonant filters, FM synthesis for metallic textures. 27 sounds in total (kick, snare, hats, toms, congas, the whole kit) and 58 pattern presets across a bunch of genres. There are also 21 fills so your loops can breathe a little.

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

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)
```

Sine wave, plucked envelope, lowpass at 800 Hz. Sits where bass should sit. There are 8 envelope presets — piano, organ, pluck, pad, strings, bell, staccato, and a flat default. Each one is just an ADSR curve, but they change the character of everything.

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)
```

You can pass chord names directly — "Am" plays all the notes of A minor at once. The supersaw is 7 detuned sawtooth waves stacked together. Add reverb and chorus and it fills the room.

There are 10 synth waveforms total: sine, saw, triangle, square, pulse, FM, noise, supersaw, and two PWM variants. All generated sample by sample.

Effects

Every part gets its own chain: distortion → chorus → lowpass → delay → reverb. You set them as parameters.

```

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)

```

The `delay_time=0.375` syncs echoes to dotted eighths at 140 BPM. When delays lock to tempo, everything sounds better. The lowpass softens the raw sawtooth.

Automation

Parameters can change mid-song:

```

lead.add("E5", Duration.WHOLE)
lead.set(lowpass=3000, reverb=0.4)
lead.add("G5", Duration.WHOLE)
lead.set(lowpass=5000, reverb=0.6)
lead.add("A5", Duration.WHOLE)

```

Or you can use an LFO:

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

That sweeps the filter between 400 and 3000 Hz over 8 bars. Shapes can be sine, triangle, saw, or square. LFO on a filter is the oldest trick in electronic music and it still works every time.

Arpeggiator

```
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 call. C minor, two octaves, sixteenth notes, rippling up and down. The `legato=True` connects notes without retriggering the envelope. The `glide=0.05` adds a 50ms pitch slide between notes so it feels liquid instead of mechanical.

All together

```
from pytheory import Score, Duration, play_score

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

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

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 = 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 = 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)

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)
```

Under 30 lines. Drums, bass, pad, lead with filter automation, two-octave arpeggio with portamento. It's not going to win any production awards, but it plays, and it sounds like something.

Export

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

44.1 kHz, 16-bit WAV. Or MIDI if you want to take it into a real DAW:

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

How it got here

I didn't set out to build this. Each feature was just the obvious next thing — "well, now I need to hear it" → "well, now I need drums" → "well, dry synths sound bad" → "well, static sounds are boring." The architecture absorbed each addition without complaining, which I take as a sign that the abstractions were mostly right.

Whether anyone needs a DAW in their Python REPL is a fair question. I don't have a good answer. I just know that I keep opening a terminal at odd hours and sketching ideas in code instead of reaching for Ableton, and that the ideas come out differently when the instrument is a programming language. More structural, maybe. More surprising.

```
pip install pytheory — docs
```

Generated from kennethreitz.org • 2026