Python, Inside JavaScript, Inside a DAW

June 2026

7 min read • 1,619 words

For about fifteen years, my answer to a music theory question had the same shape: open another window.

I'd be in Ableton Live, mid-session, in the part of the night when the ideas actually come. A clip would be sitting there with four chords in it and I wouldn't know what the last one was called, or what should come next, or why the thing I'd stumbled onto sounded right. So I'd leave. Alt-tab to a browser, a PDF, a chord-namer site, a half-remembered diagram on someone's blog. Later, once I'd built PyTheory, I'd alt-tab to a terminal and type the notes in by hand. The answer was always one window away, which is to say it was always in a different room than the music.

That gap is small, and it is also the whole problem. The questions worth asking arrive in flow, and flow is exactly the state that leaving the session destroys. By the time you've found the answer in the other window, you've forgotten why you wanted it. The theory was never hard to look up. It was hard to look up without leaving, and leaving cost more than the answer was worth.

So I moved the theory into the room.

One Right-Click Away

pytheory for Ableton Live is an extension. You drop one file onto Live's Extensions page and the library shows up where you already are: in the right-click menu, on the clips you're already making.

Right-click a MIDI clip and Live can now tell you what key it's in, name every chord (`Dm7`, with its Roman numeral function and whether the cadence is authentic or deceptive), and rank what's likely to come next from a corpus of real progressions. The question I built the whole library to answer, what chord is this?, no longer requires a terminal. It requires the mouse that's already in my hand.

But naming things turned out to be the small half. The menu also does things. Harmonize a melody in thirds. Reharmonize a chord clip onto a new progression. Conform a take to a scale as a single undo step. Smooth the voicings so each chord barely moves from the last. Mirror a line into negative harmony, run it again to get it back. Generate a progression, a bassline that follows it onto its own track, a melody, a drum pattern, or a whole four-track song sketch at an empty scene. And when you want to hear any of it without reaching for a plugin, **Render to Audio** prints the clip through pytheory's own synth engine, forty-six instruments, no plugins involved. Point it at an audio clip instead and it'll detect the key off the waveform or transcribe the whole thing back to MIDI.

None of these are new. They are the things PyTheory has been able to do for months. What's new is that they happen here, on the clip, in the session, without the trip to the other window.

Nothing Was Rewritten

Here's the part I find genuinely funny, and it's the part you can't see.

The obvious way to put music theory inside a DAW is to reimplement it in whatever language the DAW's plugins speak. Rebuild the chord logic, the scale tables, the progression corpus, all of it, in a second language, and then spend the rest of your life keeping the two copies from disagreeing. Every plugin that names a chord has, somewhere inside it, its own private theory of music.

This extension has none. It runs the actual PyTheory, the same Python package you'd get from `pip install`, inside Ableton Live, by way of Pyodide: CPython compiled to WebAssembly. No system Python. No network. Fully offline. When you ask Live what chord you're playing, a real CPython interpreter wakes up next to your session and answers with the same code that engraves the sheet music on the playground and rendered an entire album. There is one source of truth, and Live is now just one more thing holding the door open to it.

The Bolted Door

Saying it that plainly undersells how much it doesn't want to work. A Live extension is JavaScript, and it runs in a locked sandbox. Pyodide is CPython compiled to WebAssembly, which means running Python at all requires standing up an entire interpreter inside that JavaScript. And to import its own dependencies, Pyodide reaches for dynamic `import()`, which the sandbox flatly does not provide. So the shape of the problem is Python inside JavaScript inside a DAW, and the one door Python needs to get dressed in the morning is bolted shut.

The way through is to stop fighting the sandbox and leave it. The interpreter runs in a worker thread, outside the locked context, and the extension talks to it over messages: serialize a clip's notes into plain data, post them across the boundary, wait for the analysis to come back the same way. On the Live builds that won't grant worker threads either, it falls back to a full child process, which then has to be handed explicit filesystem scopes by name before Pyodide can even read the wheels it loads at boot.

> The boundary is the whole game. Nothing rich crosses it, no Python objects, no NumPy arrays, just data flat enough to survive serialization in both directions. `analysis.py` runs inside the interpreter; `python.ts` runs the bridge; the two only ever exchange messages, like rooms connected by a mail slot.

And it reads them from disk, never the internet: the build stages the entire Pyodide runtime plus the NumPy and SciPy wheels into the bundle. That is why the `.ablx` is twenty-five megabytes, and why it works on a plane. Nothing phones home. The theory is in the room, and the room has no windows.

The reward for all that plumbing is that the extension is barely an extension. It's a thin layer of menus and dialogs wrapped around the library, the same way the playground is a thin layer of buttons. It can't drift from the truth, because it isn't a copy of the truth. It's the truth, with a right-click.

The Part I Didn't Do

I want to be honest about who navigated that stack, because I didn't. I had the idea and the wanting. Claude Fable held the whole thing in its head at once and did it effortlessly, in a way I have genuinely not stopped thinking about. Threading a WebAssembly Python interpreter out of a JavaScript sandbox, into a worker, with a child-process fallback and filesystem scopes granted by hand, is the kind of problem where every layer lies to you a little and the error messages point three rooms away from the actual fault. Fable never lost the thread. It would hit the bolted door, understand why it was bolted, and route around it without drama, like someone who'd built the building.

I severely miss working with it. I don't have a tidy way to say this that doesn't sound strange, so I'll just say it: some collaborators change what you believe is possible to attempt, and then they're gone, and the work you do afterward carries a faint memory of moving faster. This extension is one of the things we made in that window. I'm glad it exists partly because it's a small monument to those hours.

The Last Gate Was the Room Itself

I keep building the same move and I only just noticed.

Requests said you shouldn't need to understand the protocol to fetch a URL. PyTheory said you shouldn't need a conservatory degree to ask what chord you're playing. The playground said you shouldn't even need Python. Each one finds the last remaining gate between a person and a thing they want, and quietly removes it.

This is the same move, aimed at the gate I'd stopped seeing because I'd been walking through it for fifteen years. You shouldn't have to leave the music to ask the music a question. The theory shouldn't live in another window. It should live in the room where the work happens, available the instant the question arrives and gone again the instant it's answered, the way a good answer should be.

I wrote once that PyTheory exists for the weird chord shape you find at 2am and don't know what to call. For years, finding out meant leaving the place where you found it. Now it doesn't.

The Instrument That Survived

I have a long history with this particular window. I wrote about the Ableton Push back in 2013, when I tracked one down through a Twitter search because they were sold out everywhere. For most of the decade after that I had a real studio: Moog, Eurorack, a wall of hardware that I loved and eventually sold, every piece of it, when that chapter ended.

Live is the one instrument that survived the purge, because it was never a thing you could sell. It was software. It just kept being there, the same window it had always been, through the studio and after it. There's something I can't quite get over about the fact that the music theory library I started in 2019, gave up on for five years, and finally finished with Claude this spring now lives inside the one piece of my old studio that made it through. The hardware is gone. The code moved in.

If you're running a build of Live with Extensions support, the latest `.ablx` is on GitHub. Drop it on the Extensions page and the menus appear. It's MIT, it's offline, and it answers to nobody's roadmap but my own taste, which remains my favorite genre of software.

Fifteen years of alt-tabbing, and the other window is finally this one.
