Episode Book

EPISODE #5813

How Python Became AI's Default Language

PUBLISHED	9 October 2026 at 13:38 UTC
DURATION	24:06
LISTEN	myweirdprompts.com/episode/python-ai-default-language-history/
HOSTS	Corn, Herman, Hilbert
SCRIPT MODEL	deepseek-flash
VOICES	chatterbox-regular (chatterbox-local)
PIPELINE	V5.2
SOURCES	15 cited by the research (listed at the end)
BOOK GENERATED	19 October 2026 at 17:40 UTC

SYNOPSIS

Python has no scientific data structures at its core — no arrays, no matrices — yet it leads AI development at 58% adoption. This episode traces the unlikely path: a 1995 mailing list post that seeded NumPy, a 2005 MATLAB licensing change that pushed research institutes toward Python, and a NIPS workshop where the scientific community converged on one toolchain. Then the AI boom arrived and changed who was showing up. We dig into the packaging chaos newcomers inherit — distutils, pip, conda, Poetry, uv — and why "there should be one obvious way to do it" never applied to distribution. Plus: why TIOBE shows Python dipping while PYPL shows it surging, and what that gap actually measures.

TRANSCRIPT



Corn

December 1989. Guido van Rossum sits down over Christmas break to build a successor to a language called ABC, and what he ships has no scientific data structures in it at all. No arrays. No matrices. Nothing a physicist would recognize. And yet, thirty-six years later, that language is carrying the entire AI industry on its back.



Herman

And it is worth saying right at the top, that was not the plan. Nobody designed Python as the numerical language. It got drafted.



Corn

Daniel sent us this one, and the framing comes out of his own experience. He came to Python late, through AI. His background was HTML, CSS, JavaScript, SQL, the web stack. And when he finally opened up Python, the first thing that hit him was not the syntax. It was the environment madness. Versions, virtual environments, conda, poetry, pip, and a statistician friend he describes as his sounding board, or victim, for all the questions about which one to use. So what he wants to know is three things, really. What was the Python world before the AI boom, who actually lived in it, and what were they like. How did it feel for a community built around math and data to suddenly become the language every developer felt obliged to learn. And then the sharp one. Has all that popularity finally lifted Python's front-end story, or is it still a back-end and heavy-computation language, AI or no AI?



Herman

The short answer to the last one is no, and it is architectural rather than a matter of effort. But the long answer is worth an episode.



Corn

So let's start before the boom, with a language that was never built for science at all.



Herman

The founding story matters here because of what it says about the community that later formed around it. Van Rossum was at CWI in the Netherlands, and Python was meant to be a general-purpose scripting language, cleaner than ABC, easier to extend. The scientific stuff was bolted on by other people, years later. And the first bolt was a mailing list.



Corn

Nineteen ninety-five.



Herman

Jim Hugunin, a grad student at MIT, posts the opening message to a brand new list called Matrix-SIG. His line is, "There seems to be a fair amount of interest in the Python community concerning the addition of numeric operations to Python." That is the seed of everything. NumPy descends from that thread.



Corn

And the interesting thing is how small that world stayed for another decade. It was a niche inside a niche.



Herman

Right up until a licensing decision at MathWorks changed the trajectory, and this is the part I find funny, because it means Python's scientific dominance was never a design victory. Around 2005, MathWorks changed MATLAB licensing, and it suddenly became prohibitively expensive for publicly funded research institutes that were not universities. Fraunhofer Institute FIRST is the case everyone cites. Mikio Braun was an ML researcher there, and his group abruptly needed something else. The interesting part is what they chose it on.



Corn

Not on the language.



Herman

Braun's conclusion was, "the infrastructure and set of available toolkits is at least as important as the language." He is judging the ecosystem, not the syntax. Python won on the surrounding scaffolding, not on being prettier than the alternatives.



Corn

And then 2005, the NIPS satellite workshop on machine learning tools.



Herman

That is the social tipping point. The community could have fragmented across three or four options and instead converged on Python, and once you get a single meeting where everyone agrees on the tooling, the toolkits start multiplying. NumPy, Matplotlib, the Shogun bindings, then scikit-learn and pandas, and once those exist the argument is over. The R-versus-Python fight in data science is entirely downstream of that one 2005 decision.



Corn

So the people in the room at that point were scientists, statisticians, data wranglers. Not web developers.



Herman

Not remotely. And Brett Cannon's line from PyCon 2014 covers the culture better than any statistic I could give you. "I came for the language, but I stayed for the community." That is a scientific community talking about itself. The loyalty was to the people, not the feature set.



Corn

Here is what I want to push on, Herman, because it is the heart of Daniel's question. That community was small and it was self-contained. Then AI arrives, and suddenly every developer on earth is being told to learn this thing that was built by and for people who think in matrices, and there was no vote on that. What does it actually feel like to be a niche community that gets drafted into being the center of the industry?



Herman

I think the honest version is that it was less a jolt to them than you would expect, because they had already won. By the time the AI wave arrived, the libraries were settled, the conventions were settled. What changed was who was showing up and asking questions, not what the core community was doing. The jolt landed mostly on the newcomers, which is Daniel's experience exactly.



Corn

That tracks with the packaging story, which is where I want to spend some real time, because it is the thing that confused him and it is the thing that tells you who was in charge.



Herman

Or who was not in charge. Python shipped in 1991. distutils, the first packaging tool that came with the language, shipped with Python 1.6 in 2000. That is nine years with nothing.



Corn

Nine years.



Herman

Then the metadata gets standardized, PEP 241 in 2001, PEP 314 in 2003, PyPI launches in 2003. Now, to give you a comparison that stings, Perl had CPAN in 1995. It predates PyPI by eight years. Perl was doing centralized package distribution almost a decade before Python had a package index.



Corn

And then the explosion, which is the part that explains what Daniel walked into.



Herman

setuptools in 2004, Phillip Eby, with easy_install and the Egg format. virtualenv in 2007, Ian Bicking. pip in 2008, also Bicking, and it was originally going to be called pyinstall. requirements dot txt shows up around 2010 and it is a community convention, not a PEP. wheel, PEP 427, in 2012 and 2013, from Daniel Holth. pipenv in January 2017 from Kenneth Reitz, which was actually recommended by Python dot org in November of 2017. Poetry in April 2018 from Sébastien Eustace, using a resolver called PubGrub. pip does not get proper backtracking resolution until pip 20.3, late 2020.



Corn

Which is a real date to sit with. The default tool does not get a competent resolver until 2020.



Herman

pyproject dot toml gets standardized across two PEPs, 518 in 2016 and 621 in 2020. And then uv in February 2024, out of Astral, Charlie Marsh's outfit, written in Rust, ten to a hundred times faster than pip.



Corn

So from distutils to uv, that is a twenty-four year run, and if you count the nine years before distutils, twenty-five years of fragmentation in a language whose stated value is that there should be one, and preferably only one, obvious way to do it.



Herman

And a commenter on Hacker News named endgame put the irony better than I can. "How does a language ecosystem that bakes there should be one, and preferably only one, obvious way to do it into the interpreter end up with such a convoluted packaging story?"



Corn

Now conda, because that is the one that seems to trip people up worst. It is not a newer pip. It was built in 2012 by Anaconda, then calling itself Continuum Analytics, for a specific reason.



Herman

Compiled dependencies. NumPy and SciPy lean heavily on compiled C and Fortran, and early pip handled that badly. So conda was "not trying to be a better version of pip. It was trying to solve a different problem." That is their own framing. And the practical consequence is that mixing conda and pip causes subtle breakage, because neither fully understands what the other did.



Corn

Which is exactly the class of problem Daniel was asking his statistician friend about.



Herman

And the friend is not being difficult. There is no single right answer because there are two ecosystems sitting on top of each other, and which one you should be in depends on whether you have compiled dependencies, which depends on your field, which is why a data scientist says conda and a web developer says pip and they are both correct.



Corn

I like that, actually. The confusion is not a knowledge gap. It is a genuine fork in the road that nobody has removed.



Herman

That is the whole packaging story in one line. The Zen of Python describes an aspiration, not a description. Nobody was ever appointed to own it. There is a comment that Guido never cared about packaging, and a reply to it that I like a lot, which is, if you think Guido never cared about packaging, try calling it the Cheese Shop within earshot of him. So he cared. He just did not run it, and a language where no one is in charge of distribution is going to end up with eight competing answers.



Corn

Twenty-five years of it.



Herman

And the current landing point is uv. The recommendation floating around for 2026 is that for most Python projects, uv is the right default. Which is worth noting, because uv does not solve the problem by being tidier. It solves it by collapsing pip, virtualenv, pipx, poetry and pyenv into a single Rust binary. The answer to fragmentation was to swallow all the fragments.



Corn

Daniel would have killed for that in year one. He was navigating this on vibes and a statistician.



Herman

Everyone was. That is the correct way to do it.



Corn

So that is the pre-AI world. A scientific community, a weak distribution story, and twenty-five years of tools layered on top of each other. Then AI shows up and the numbers move.



Herman

And the numbers move in two directions at once, which is the most interesting thing on the research sheet. In September 2026, the TIOBE index has Python at seventeen point seven six percent, still number one, but down eight point two two points year on year. Meanwhile PYPL, which measures tutorial search share, has Python at fifty-two point zero eight percent, up twenty-one point eight four points year on year.



Corn

So one lens says it is weakening and the other says it is eating everything. How do you square those?



Herman

The lenses are measuring different populations. TIOBE is trying to estimate engineer footprint, who is writing it professionally. PYPL is measuring who is going online to learn it. And a twenty-one point jump in people searching for Python tutorials, while the professional footprint dips, is consistent with one story. A huge wave of people are learning Python specifically because of AI, and not all of them are becoming Python engineers. They are learning it to drive a product, then going back to whatever they were doing.



Corn

Which is Daniel exactly. He did not become a Python developer. He learned enough Python to do AI work.



Herman

And that population is enormous now. Python leads AI development at fifty-eight percent adoption, and there are over a hundred and fifty-two thousand AI-related job listings in the September 2026 count. On the Axis Intelligence consensus index, which aggregates popularity lenses, Python scores ninety-seven out of a hundred. The runner-up, Java, is at seventy-five point eight. That is not a lead, that is a different category.



Corn

But the interesting number is the library growth, because that gets at whether people are actually writing Python or just installing it.



Herman

Probabl did a deep dive in July of 2026, tracking forty popular data science libraries on PyPI, and they split the timeline across the release of Claude Code in February 2025. In the sixteen months before that, the compound monthly growth rate across those libraries was about two point four percent. In the sixteen months after, it is over five point four percent. scikit-learn's growth rate roughly doubled, from three point seven to six percent. Deep learning libraries grew the most, by about four point eight percentage points.



Corn

So the boom shows up in the download data.



Herman

It shows up, and then you have to be careful, because there is a caveat that Probabl themselves flag. PyPI downloads are installation events, not users. Every CI pipeline, every Docker rebuild, every automated job that reinstalls the same package inflates that number. So two billion downloads of scikit-learn in a year, sixty-three downloads a second, ninety-three percent year on year, is real, and it is partly machines installing it for machines.



Corn

Sixty-three a second. How much of that is a person?



Herman

Unknown, and I do not think anyone can tell you. What they can tell you is that the correlation between those library downloads and the Python SDKs of five AI vendors shifted. It was zero point one eight in the LLM era and zero point three seven in the agentic era. scikit-learn correlates with all five SDKs at point three eight, p less than point zero five. And Probabl is explicit that correlation is not causation.



Corn

They would have to be. Point three eight is a relationship, not a proof.



Herman

It is a real relationship. It is not a claim that Claude causes scikit-learn downloads.



Corn

So the shape of it is, the scientific libraries are accelerating right alongside the AI tooling, and they are not the same population, but they move together.



Herman

Which is why the scope question is the right one to ask. Python is not the AI language. Python is the language that happened to already have the numerical stack that AI needed, and AI pulled the whole stack up with it. scikit-learn is not an AI tool. It is a statistics library that was sitting there in 2005 when the community needed one, and now it is getting sixty-three downloads a second because everything downstream of it needs it.



Corn

And the scope of what it is the default for keeps expanding outward from that. Dropbox is the case that convinced me, because Drew Houston says Python "ended up being a big force multiplier on our effort and no other language that we considered had anything close to that kind of capability." That is not a research lab. That is a file-sync company.



Herman

They migrated over a million lines of front-end code from Python 2 to 3 in 2019, which is a sentence with a strange punchline in it, because front-end at Dropbox meant the desktop client, not the browser. Instagram, Google, Dropbox, all built substantial infrastructure on it. And the big shift in who is doing the asking is visible in what the community is now designing for. Probabl's open question for 2026 is what it means to make libraries agent-ready. They are not designing for human ergonomics anymore. They are designing for a model that will call the function.



Corn

That is the community's identity moving in real time. They spent thirty years making libraries comfortable for a scientist at a terminal, and now the primary consumer is an agent that does not care about ergonomics at all.



Herman

It might be the best thing that ever happened to the packaging problem, honestly. An agent does not care how ugly the interface is.



Corn

Let's get to front-end, because that is Daniel's closing question and it deserves a straight answer. He is right that Python has never been the web developer's language. The question is whether AI changed that.



Herman

It did not, and the reason is not effort or popularity. It is that browsers do not run Python. They run JavaScript. HTML, CSS, JavaScript. That is the whole menu. If you want Python in the browser you have to transpile it, through something like Brython or Transcrypt, or run a full interpreter in the browser through Pyodide, and both of those add overhead and complexity that the native path does not have.



Corn

So every Python-in-the-browser project starts by paying a tax.



Herman

The argument I find most convincing comes from Meredydd at Anvil. His point is that JavaScript was never the actual problem. Look at what a web request really does. It goes from SQL to Python on the server, then out as JSON over HTTP, then into JavaScript, then into the DOM, and then finally onto pixels. Five translations. Replace the JavaScript with Python and you have not removed a single one of them. In his words, you still have all the same translations, all the same frameworks, and all the same pain. That is why Brython and Transcrypt never took over the world. They move a piece without simplifying the board.



Corn

He is right, and it is a slightly sad conclusion. The Python people keep assuming the barrier is the language, and the barrier was never the language.



Herman

The language is the visible layer. The scaffolding underneath it is the whole problem.



Corn

What actually exists on the Python front-end side, because there is a real long tail there.



Herman

Anvil itself is the most serious attempt, full-stack Python, built on a Python-to-JavaScript compiler, so you write Python and it becomes a browser app. Reflex has been pushing comparisons as recently as April 2026. Taipy, Streamlit, a handful of Python-native CMS projects. They exist, they work, people build real things on them.



Corn

But they are choices people make because they want to stay in Python, not choices the front-end community makes.



Herman

If you are a Python purist who resents touching JavaScript, Streamlit is a gift. If you are a front-end developer, you are not looking at Streamlit. You are looking at the JavaScript ecosystem, because that is where the hiring is, the components are, and the browser lives.



Corn

That is the answer to Daniel's question. The AI boom has not uplifted the front-end story at all, because it was never a popularity problem. You could triple the number of Python developers tomorrow and the browser would still not execute Python.



Herman

Right. And the AI-driven web apps are the cleanest proof, because even those are split the same way. The model runs on the server, in Python. The interface to the model runs in the browser, in JavaScript. AI did not merge the two halves. It reinforced the split by making the server half more valuable and leaving the browser half exactly as it was.



Corn

Front-end stays JavaScript. Back-end and the numerical work stay Python. The line is in the same place it was in 2010.



Herman

The line is in the same place. What moved is how much traffic is on the Python side of it.



Corn

Before we hand things over, there is a piece of this I could not fit into the main thread and I want to land it, because it is the endgame of the packaging story. The reason twenty-five years of fragmentation might actually get solved is not that the community agreed on a standard. It is that the consumer changed. An agent does not care that there are four ways to declare a dependency. It will pick whichever one resolves. So the thing that humans could never agree on, because every tool had a constituency, gets resolved by removing the humans from the room.



Herman

Which is a strange outcome, because it means the Zen of Python might finally come true for a reason nobody in 1995 could have wanted. There will be one obvious way to do it, and it will be the way the machine prefers.



Hilbert

They agree with you.



Hilbert

The scientists. They agree with Daniel. The tools never mattered to them. I put money into a Python company, back in the early nineties, and it died of a dependency.



Corn

A startup, before the AI wave.



Hilbert

A company. Nineteen ninety-three. There were four of them and a statistician. The statistician was the one who told me about it. He said, it is just like MATLAB but free. I gave him eleven thousand dollars. I thought that was the price of the whole thing. It was



Hilbert (continued)

not the price of the lot.



Corn

It failed on a dependency.



Hilbert

It failed on a Tuesday. A library they relied on published a new version, and nothing would build on Wednesday. Nothing was changed in their code. The library changed, and the machines would not reproduce what they had on Monday. Their whole product was four hundred lines. It was the environment that killed them. Nobody could tell me which of the eleven files in the folder was the one that mattered.



Herman

You never got the money back.



Hilbert

I got a shell script. I wrote it myself, trying to fix it, because I thought the problem was that nobody had ever written down the correct order to do things in. It is four thousand lines long. I still run it. I will not show it to anyone.



Corn

Four thousand lines.



Hilbert

It is held together with duct tape and prayers. It has one bug I have not found since the early nineties and I have decided to live with it. Herman, you would hate it. It starts by copying everything to a folder called yesterday and then it works from there.



Herman

Why yesterday?



Hilbert

That was the name of the folder. I did not name it, the script did.



Corn

Does it still work?



Hilbert

It works on the machine I wrote it on. That is where it runs. I talk to the statistician every few years. He still says the same thing about the pronunciation. I say pipe and conda, the way a person would say pipe and conda at a hardware store. He gave up correcting me sometime around the second company.



Herman

The second company.



Hilbert

Two thousand four. Also the tools. Also a statistician. It is in the script.



Corn

Let's leave it there, because the honest open question is whether the scientific community that built all of this feels like the host of the AI era or a bystander in it.



Herman

The number that stays with me is the two directions the same month moved in. TIOBE down eight points, PYPL up twenty-one. That is a community whose work is being used by an enormous number of people who will never think of themselves as part of it.



Corn

The second open question is whether the agent era finally settles the packaging problem, or makes a new one, because if the consumer is a machine, the tools will get optimized for machines and humans get whatever is left.



Herman

Which might be better. We will find out.



Corn

If you enjoyed this deep dive into Python's pre-AI soul and its AI-era identity crisis, leave us a review. It helps other listeners find the show.



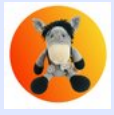
Herman

Thanks to our producer, Hilbert Flumingtop, who has a folder called yesterday somewhere and will not tell us where.



Corn

If you enjoyed this one, try episode ten twenty-one, The Python Paradox; episode thirty-five, The Privacy Gap; and episode twenty-five, The Open-Source Battle for AI's Hardware Soul. This has been My Weird Prompts. Send us your own prompt on Telegram at t dot me slash MWP listener bot.



Herman

We will be back soon.

SOURCES

What the research for this episode read before the script was written. Primary sources first.

1. **conda.org, Conda and pip are two ecosystems, 2026-05-07 . [primary]**
   https://conda.org/blog/2026-05-07-conda-and-pip-ecosystems/

2. **Ryan Orban, summarizing Mikio Braun on the 2005 MATLABPython inflection.**
   https://ryanorban.com/notes/python-language-choice-data-science/

3. **Wikipedia, History of Python.**
   https://en.wikipedia.org/wiki/History_of_Python

4. **Martijn Faassen, A history of Python packaging, 2009-11-09.**
   https://blog.startifact.com/posts/a-history-of-python-packaging/

5. **xpe, Python Package Management: Three Histories, 2025-10-27.**
   https://gist.github.com/xpe/9c35a88aa6df6dc53653f5f170fa40ec

6. **Probabl, Scikit-learn was downloaded 2B times, 2026-07-21.**
   https://blog.probabl.ai/data-deep-dive-1

7. **Inference by Sequoia Capital, 2025-09-18.**
   https://inferencebysequoia.substack.com/p/why-python-became-the-language-of

8. **Anvil (Meredydd), Why Python in the Browser Isn't Enough.**
   https://anvil.works/blog/why-python-in-the-browser-isnt-enough

9. **Hacker News discussion, Uv: Python packaging in Rust, 2024-02-15.**
   https://news.ycombinator.com/item?id=39387641

10. **TechRepublic, TIOBE Sept 2026, 2026-09-07.**
    https://www.techrepublic.com/article/news-tiobe-september-2026-julia-nears-top-20/

11. **Axis Intelligence, Most Popular Programming Languages 2026.**
    https://axis-intelligence.com/programming-language-statistics/

12. **SecondTalent, AI language stats, Sep 2026.**
    https://www.secondtalent.com/resources/top-ai-programming-languages-by-usage-stats/

13. **pydevtools handbook, 2026 recommendation.**
    https://pydevtools.com/handbook/explanation/which-python-package-manager-should-i-use/

14. **Reflex, full-stack Python framework comparison, April 2026.**
    https://reflex.dev/blog/django-vs-flask-vs-reflex-comparison/

15. **Simbainfotech, on why browsers can't run Python.**
    https://simbainfotech.co.in/why-python-isn-t-used-for-front-end-real-reasons-and-surprising-facts