EPISODE 907

[INTRODUCTION]

[00:00:00] JM: Python is one of the most popular programming languages in the software world. After working with Python and developing a love for the language, Michael Kennedy started to wonder why there was not a high-quality podcast dedicated to covering the community of Python users and the new technologies of the Python ecosystem. Michael Started Talk Python To Me as a podcast with the goal of telling stories from the world of Python.

Today, Talk Python To Me is the most popular podcast dedicated to Python and related technologies. Subjects on the podcast include web frameworks, compilers career development and cloud services. One of the more recent developments in the world of Python is the prevalence of data science, which Michael covers in great detail on his podcast. Michael has also spun up a second podcast called Python Bytes, which offers timely updates to the Python community.

Michael joins the show to share his thoughts on several topics related to Python, including compilers, data science workflows and web frameworks. Michael also gives his perspective on the world of software podcasting, which is he has been doing for more than four years. As you probably are aware, I'm quite intrigued by the world of software podcasting. So, it was a lot of fun to talk to him, and I hope you enjoy it as well.

[SPONSOR MESSAGE]

[00:01:31] JM: Today's episode of Software Engineering Daily is sponsored by Datadog. With infrastructure monitoring, distributed tracing, and now logging, Datadog provides end-to-end visibility into the health and performance of modern applications. Datadog's distributed tracing and APM generates detailed flame graphs from real requests enabling you to visualize how requests propagate through your distributed infrastructure.

See which services or calls are generating errors or contributing to overall latency so you can troubleshoot faster or identify opportunities for performance optimization. Start monitoring your

applications with a free trial and Datadog will send you a free t-shirt. Go to softwareengineeringdaily.com/datadog and learn more as well as get that free t-shirt. That's softwareengineeringdaily.com/datadog.

[INTERVIEW]

[00:02:30] JM: Michael Kennedy, welcome to Software Engineering Daily.

[00:02:33] MK: It's really great to be here. Thank you for having me. It's an honor to be on your show. So I'm really excited. I'm looking forward to talking to you and everyone in your audience.

[00:02:41] JM: Absolutely. So you host two podcasts related to Python. The first one is Talk Python To Me. What did you like about Python so much that compelled you to start a podcast devoted entirely to the programing language?

[00:02:58] MK: It's interesting. It's not so much that I felt like I was the super expert in Python. When I started the podcast, I had only been doing Python for a couple of years. But I've always been a podcast fan. I suspect you've been a podcast fan for a long time as well.

[00:03:13] JM: Addict.

[00:03:15] MK: Yeah, absolutely. I find the medium is just incredible. I've got a really busy life. I have kids. At the time, I had a job, and I always wanted to squeeze in this extra level of like learning, especially on programming and stuff, and I found podcasts were perfect for that, and I had been listening to them since like 2003, 2004. Probably they won't even call that there. It was just like, well, it's an audio RSS feed or something, right?

What I found when I got into Python was there were no podcasts for Python developers. There had been. There had been three or four and they had all stopped at least six months – For at least a six-month period, some for longer back. I'm like, "I don't understand why this is. Why are there no podcasts? What if no one else is going to do it?" and I want to have a podcast where I can hear these stories in the background and the history and what's going on. I guess I'll just have to make one. So that as the idea, is like I just wanted there to be a podcast for the Python

community for very selfish reasons initially. But also being a podcast fan, I was always on the lookout of like, "When is there an opportunity to make a meaningful contribution? Not be the 10th JavaScript podcast or whatever? Where would people really value this?" So I thought, "I'll give this a try for a few months. If people make fun of me, and it seems like, really, I'm not doing a good job or either they don't appreciate it, I'll stop, but I'll give it a shot." And it's taken off from there and it's been great.

[00:04:40] JM: When you think about the podcast medium, are you a historian? Are you a day-to-day journalist? Are you somebody who's trying to create durable content? Are you somebody who's trying to create daily updates that may have an expiration date? Tell me how you think about the content durability that you're working with.

[00:05:02] MK: Sure. I think of it having sort of two modes. I think of myself primarily as a storyteller. I think so much around programming gets polished down to just the final essence, and the people behind it, and it's original reason and all those things just get polished away and lost if you don't hear the story of, "Well, why did you do this? Or what came before this?" And so on.

So first and foremost, I think of the job to like sort of tell those stories and humanize technology. But I also see it as both durable and evergreen and very temporal. That's why I created two podcasts, because Talk Python To Me, I interview people like scientists working at, say, Large Hadron Collider, or the Linear Collider and using machine learning to like discover particles. That is good for a couple of years, at least. It's really interesting.

But mixed in with that, probably it doesn't make sense to say, "Oh! And guess what, on Tuesday, there's going to be – It's the last day for call for papers for this random conference." Those are just so incompatible in my mind, so I decided to create Talk Python To Me, which tells this evergreen stories, and Python Bytes, which is basically like a weekly newsletter, but in audio form with commentary. So that's where the transient stuff that keeps the community up-to-date is supposed to. Then Talk Python is where these long-term stories go.

[00:06:33] JM: Do you think there is something about the podcast infrastructure that is immature? Because we have this overloaded medium, where I wake up and I look at my podcast player, and I don't know about you, but I subscribe to a lot of podcasts.

[00:06:53] MK: Yeah, too many.

[00:06:54] JM: Well, I would say not enough still.

[00:06:58] MK: Sure. It's a matter of time though, right. I feel bad for not listening to them, but I got to just treat it more like Twitter, I guess.

[00:07:04] JM: See. That's the thing. That's how I view it, is it's like – My podcast feed should feel like YouTube, to some extent. It should feel like an overwhelming amount of content that I will never get to. Actually, even YouTube doesn't quite feel that way yet. I still feel like I reach the bottom of the meaningful content that is at least being served to me. But do you think there is something wrong with the way that the infrastructure is currently presenting podcast to us? Do you think the form factor for podcast consumption will change?

[00:07:36] MK: I don't know if it will change. Probably it will change. I hadn't really thought too much about it, but your analogy with YouTube, and I guess mine with Twitter a little bit, makes me feel like at least my podcast player apps, like for example I use Overcast on the iPhone. It shows me a bunch of shows, then your show, my show, others, and then I can drill into it and say, "Well, here are the recent ones and the ones I've downloaded and so on."

You're right. What I should really see is here's the flow of all the stuff I'm interested in. I think that that would be great. I don't know how well podcast work for very timely stuff, unfortunately. I wish it was better. Because there might be something timely on your show, it came out the next day, but I didn't get around to pulling it out of my list to listen to it for three weeks and then it's too late. That's still tricky, and I don't know. It's like Netflix having news or something. It's a little weird.

[00:08:33] JM: Yeah, exactly. Right. It is both the treasure and the shortcoming of the podcast infrastructure that it's not an algorithmic feed. None of these – There are podcast players that I

think have tried this. But anyway, let's talk about Python for a while. We can get back to podcasting. Much as I love talking about podcast idiosyncrasies.

So the interesting thing about podcasting about software is that the industry changes so quickly. So even in a short span of 4-1/2 years that you've been doing your podcast, the Python community has changed quite rapidly. What are the most market changes that have occurred over the 4-1/2 years since you started your podcast?

[00:09:20] MK: I'd say there's probably two, and there's many of course, but there's two that really stand out to me. One, as I feel like people follow Python closely, there's been this Python 2 versus Python 3 divide. In 2008, the core developer said, "We're going to rewrite in very minor, but breaking away some of the way the language works to sort of clean up all the craft that's built up over the last 20 years or so." The expectation was people would just switch over quickly to that and we just keep carrying on.

But what actually happened was there were ton of older libraries that didn't get updated, so couldn't work on Python 3. If you're dependent on any of those libraries, you were basically stuck on the older version of Python. That's been an ongoing debate and whatnot for years. But I feel like the changes, that was still a solid debate now, and at this point, that's over. Everyone's pretty much either on to Python 3 or freaking out that they need to figure out a way to get there in the next 6 months. Something like that.

So change one is the Python 2 versus 3 debate. It's kind of over. I like to phrase is as legacy Python and Python. But that's taken off a little bit, not entirely. The other one is the rise of data science and data scientists, and just the use of Python in scientific computation. Things like Jupyter notebooks and all the tooling around that, for example. Around 2012, people started moving in and it's just – Last couple of years, it's just really grown in that regard.

[00:11:01] JM: Do you have a perspective as to why Python became the data science language as supposed to Ruby, for example, or Go, or something else?

[00:11:13] MK: It's a little bit hard to compare it on Ruby. I feel like Ruby was super focused on the web. So maybe that's a bit of the challenge. Also, the Ruby language, I don't know that it's

quiet as simple as Python. It's got some really clever conventions to it. I don't know it that well, but that's my impression.

I do think one of the main reasons, especially comparing to, say, against Go, is Python is what I consider a full spectrum language. That's not really a term that's used. So I got to define it.

So Python, you can be super productive with Python with a very partial understanding of the language. So, I might not even know what a class is. I maybe can't even create a function, but I can go and say, "I would like to use these three libraries out of the data science space. Type in these 8 lines of code and have trained up or executed a machine learning model." I don't even know what a function is and here I am like writing AI or whatever. Something crazy like this.

So you can start out incredibly simple. You don't need to know about header files, static linking, compilers, static main void, all these stuff that you typically have in a lot of these languages. You can just get started. But at the same time, it's not like VB6 or scratch or block or some of these very simple learning languages. It scales all the way up to create YouTube and Instagram and all the other amazing stuff that people are doing with it. It's like you can start really easy and then add on these more complex concepts, like classes, generators, whatever, as you need them to build real software.

[00:12:56] JM: It is so hard to understand the evolution in retrospect, and I'm certainly not an expert on the evolutionary background of Ruby or Python. And you're right, Ruby did develop this association with Ruby on Rails. Probably the most popular web framework even today in terms of volume of web developers working on it. I could be wrong about that. But I feel like the Python web frameworks have caught up in terms of at least there are very substantial communities around them. I know there was always a substantial community around Django, but Django was never as trendy as Rails. But I feel like Flask, the Flask Python web framework, has achieved some moderate level of trendiness. Am I correct there? I mean, trendiness in an important way in the sense that you actually get the network effects needed to build a substantial community, which leads to the edges being sanded more quickly.

[00:14:01] MK: It's really interesting the difference between Flask and Django. Those are definitely the two most popular ones. Until I think last year, Django was more popular at least in

terms of number of web developers using that as their primary web framework. But if you look at the rate of change, Flask is growing more quickly and it looks like it just eclipsed Django, but will actually continue to do so.

Flask is interesting, because it has all these little plugins and extensions and stuff that you can grab and bring all these pieces together.

I think those two frameworks have a really different philosophy and it kind of depends on what you're drawn to, right? Django comes with a lot of what you need kind of prepackaged and put together. Sa, if you want an admin backend, well, you kind of just turn that on and have a little data table type of management thing for your database. Whereas with Flask, you're creating ORM classes, and if you want an admin section, well, start writing some code, some HTML, and your admin stuff in there and it's all about little tiny building blocks in Flask and more like pre-built, big Legos, I don't know. The big block Legos versus little tiny ones is kind of how I think of the two.

[SPONSOR MESSAGE]

[00:15:25] JM: As a programmer, you think in objects. With MongoDB, so does your database. MongoDB is the most popular document-based database built for modern application developers and the cloud era. Millions of developers use MongoDB to power the world's most innovative products and services, from cryptocurrency, to online gaming, IoT and more.

Try MongoDB today with Atlas, the global cloud database service that runs on AWS, Azure and Google Cloud. Configure, deploy, and connect to your database in just a few minutes. Check it out at mongodb.com/atlas. that's mongodb.com/atlas.

Thank you to MongoDB for being a sponsor of Software Engineering Daily.

[INTERVIEW CONTINUED]

[00:16:20] JM: With regard to data science, one thing I have had a lot of trouble understanding is exactly the way in which data science workflows get productionized. Have you talked to many

people about this process? Like let's say I'm a machine learning person or a data scientist and I join a company. I'm like the first data scientist the joins. I've got some datasets. I'm tinkering with them. I'm trying to find correlations. I'm trying to find ways to do recommendations, perhaps. Then a day comes where I need to productionize what I've been building, which I'm not sure what that means. Does that mean I turn my scripts into some kind of offline batch job, or does that mean that I set up a Flask API and put my machine learning model behind a simple Flask API and then other people in the company can start hitting it, and then all of a sudden I'm a web developer in addition to a data scientist.

[00:17:28] MK: Right. Right, exactly, and these are really different paths that you might take, right? So one of the things in the data science space is a big move towards more software engineering, computer science-y skills, more version control, more factoring of your code possibly into multiple reusable scripts and modules and whatnot rather than just, say, packaged up in some kind of executable notebook, like Jupyter.

So I think it really depends what kind of experience that data scientists person has. But, generally, I guess the most – Probably the most appropriate way most of the time would be to go and create some HTTP endpoint with Flask and then have it interact with your models on the server. Just like a regular web app might call the database or then call the Stripe API. You might run your models. The thing is most of the time, getting the model answer the question is pretty quick. Training the model is super, super intense.

So you might spend hours feeding data into your model to generate it. But then you've just got a small saved model and you just tell it to run and the answers might come out super quick. So it might be very appropriate to run real time, "Here's some input. What does the model think?" whereas the training of it might be some kind of batch job or something done in the background.

[00:19:00] JM: Productionized machine learning, it's an interesting field, because I feel like there's a limited scope of domains where people feel safe productionizing machine learning, and those domains are things like recommendations of movies, or recommendations of podcasts. But machine learning – Then there's also mapping. You kind of have no other way to build good mapping software than to have some machine learning backing to it.

[00:19:34] MK: Yeah, image recognition, I would put as another one that's pretty common.

[00:19:38] JM: Image recognition. Right. What are the common applications where people feel comfortable productionizing machine learning today, and are there applications where people are a little bit afraid because the quality bar is higher and we're kind of not there yet in terms of what you can use?

[00:20:00] MK: Yeah. I'm more of a web developer. So I don't know that I really can reliably answer that question. So, yeah, I don't know. I definitely agree with the ones that you put out there. The recommendation engines seem very common. I had a guy on the show, Ajay, on Talk Python, who's doing exactly what we talked about, putting machine learning models behind Flask endpoints and stuff like that. And it was all about recommendation engines, and that definitely works.

I don't know. It's really interesting where we are with machine learning. We're to the point where – I don't know, how you feel about it, but 20 years ago when I would hear people talk about AI and stuff, it felt like there was kind of toys, there was the Turing test. If you can have a chat with the AI and you believe it's real, then like it somehow become good enough AI or whatever, and that was cute. It's like, "Oh, yeah! But now we're just having cars drive themselves."

I do know. The ranges is so insane these days. I feel like I couldn't really take a good guess, because it seems so high. I said earlier about the folks doing the particle accelerators, and they're actually using that to help discover corks and other subatomic particles by analyzing the pictures. There're astronomers figuring out where the planets are over in the stars using machine learning and stuff like that. There's really crazy stuff.

But a pretty common theme around all those has to do with image processing. You process the image of the particle collision and then you get an answer. You process the image of the stars and you get an answer. It feels like something around images. The machine learning is really good at pulling out the nuance details that humans aren't. We had this thing where a whole bunch of images for breast cancer were analyzed, and the computers actually did better than the radiologists in detecting like early stage cancer. So maybe that's a guide, but I don't really know for sure the whole spectrum. But that feels like that's a pretty big area.

[00:22:17] JM: So that kind of application. The application where you've got an image set. You're training a model to do some stuff with the image set, and you could build your entire system in some data science framework, or you could use the assistance of Amazon Web Services APIs, Azure APIs, Google APIs. Do you have a sense of the breakdown for how does a data scientist leverage the cloud APIs, like the image recognition APIs? Or are people still just kind of building everything themselves?

[00:22:59] MK: I feel like talking to some of the folks at, say, Azure and stuff like that, that some of those sort of pre-canned AI things, some of the cognitive services and things like that, those are typically used more by people who are developers or technical in some way, but they're not machine learning AI specialists.

If you're a real specialist, you're probably using something like TensorFlow or Keras and just writing the code directly against that. But I do see even in that realm of sort of the professional data scientists, machine learning folks, they might put that into a Docker container and then run that on the GPU's and the high-end processors and servers on AWS, on EC2, or on Azure or something like that. That's pretty common.

[00:23:49] JM: Changing the topic. When I was early on in my career, I did some internships it Ruby-based companies, and both of the companies that I worked at had made this transition to JRuby, which is like the Java runtime version of Ruby that allowed – I think the main thrust was to allow for higher perform garbage collection. Because if you translate your Ruby into Java, then you get to use the Java byte code and all the associated breakthroughs in garbage collection. I think the same thing has happened to some extent in the Python ecosystem. You have Gython. And then you also – You've had some episodes on the subject. You also have people who build custom compilers for Python, and I guess that takes Python essentially from an interpreted language to a precompiled language which allows for higher performance. Give me any thoughts or the state of the compilation ecosystem, the compilation execution ecosystem of Python and why people take these alternative paths to compiling their code.

[00:25:04] MK: Sure. I guess the simple, quick answer before I give you the detailed one is it's complicated. So when you think about Python speed, I mean, the reason you would make these

changes almost all the time, not all the time, but most of the time, has to do with performance. So it's just really, really interesting. So let me just talk about the plane Python often referred to as C Python, because the interpreter is implemented in C. Then I can explain like the stuff around it.

So C Python, when you run your Python code, it's gets converted to byte code much like Java byte code or .NETIL. But instead of being further transformed with a JIT compiler, it's just fed to this gigantic, wild hoop with a huge switch statement. It's like at 3,000 lines long the switch statement. It's crazy. It just says if the byte code is this, do that. If the byte code is that, do that. And that makes it obviously much slower than compiling to machine instructions.

But where it gets complicated is Python has a very rich interoperability and foundation in C and other compiled languages itself. So, as part of a Python package, I can say compile some of the code into C code. Let's just s tick with C. It could be other compiled stuff as well. And then that part of the code I can then just call through an interop layer in my byte code. So maybe I have this really tight loop. If it was pure Python, it'd be slow, but we rewrote it in C in a multithreaded high-performance way with good compilers. Now it's going superfast.

What speed does that run at? Does that run at Python speed or C speed? And it depends how much time you spend in each. So people have tried to either make alternate versions of the Python runtime. One for improved speed, or two, for improved interoperability and other things.

So you mentioned Gython. That does let it run on the Java JVM, but I believe it also gives you direct interoperability with the Java base classes library, also with like your code. There was something called Iron Python, which did the same thing for .NET, and that gave you a deep interoperability between the .NET base class library.

So if you'd already had a bunch of .NET code and you wanted to have a little Python program that could use that, then you could maybe run it on Iron Python and then it can have access to all this .NET work that you've done, which it normally wouldn't have.

So one is this interoperability, and I don't really know if the iron Python and the Gython version was more focused on interoperability are more focused on performance. I would guess – I don't know. It could go either way.

On the other hand, you have things like PyPy, why which is a jit compiled version of C Python, basically. But it's not as straightforward as, say, Java or .NET. The thing there is it will run in the traditional way with this giant loop that I talked about until it finds a hotspot and then it will actually jit compile that bit of code. And that makes that much faster. So far it sounds like, "Well, why wouldn't you just always do that?" That sounds great. The problem is you give up the C interoperability to a large degree when you switch to these other frameworks.

So when you think about speed you're like, "Okay. Well, it's really great for pure Python. Almost all these are probably faster. Certainly Pypy is almost certainly faster." There're been some stats like five times faster. So who doesn't want that?

Well, if you're main bit of code was actually relying on SQL Alchemy, which had the C compiled layer from the deserialization, or it's working with data science in NumPy. All of a sudden you have to now give up that C code, which maybe was – I don't know, 50 times faster. So now you're slow again.

There's been a lot of sort of false starts and back-and-forth. There're some folks working on trying to do a Rust reimplementation of C Python, which maybe has some promise. There're a lot of attempts at these things, and yet most people day-to-day, I would say the vast majority people day-to-day still use just standard, old C Python. That's where the core developers are working. That's where the new features show up first. And most importantly, that's where all of the libraries. There're almost 200,000 different packages in the package manager, right? PiPY. That's where those all work. Some of them don't work in the other things. Many of them don't work in the other things. So it just gets really, really wonky and it always turns out to be some kind of tradeoff. You can get speed here, but then you'd give it up there, or you get speed here, but you can't have those libraries that you'd really want or something like that.

[00:29:57] JM: Well, I have a question about that. So if I have some of my Python that's running in the –Python has like a virtual machine. The Python virtual machine, because it gets interpreted and whatever.

[00:30:11] MK: Yeah. I mean, it's not as much of a virtual machine as Java or .NET. The memory management story is much simpler and things like that. But, yeah, there is an interpreter that's processing it and orchestrating all the execution and stuff like that.

[00:30:27] JM: Okay. So let's say I have some of my Python executing just in a normal Python virtual machine. Some of my Python is getting translated into Gython code and then running on the JVM. Maybe some of it is getting translated into C code so that it runs in super-high performance. What is the interoperability penalty there for interacting between those different lower-level translated Python code segment? Is it like serialization deserialization?

[00:31:04] MK: It's usually – There's a specific API that the C Python interpreter, the runtime, knows how to – It's like it's baked into it. How to work with C and how to exchange data? There's a whole C API, like Py object pointers and stuff you exchange. There're guarantees around like thread safety, and who owns what object, and all that kind of stuff. Then all the other worlds with jit compilers and garbage collectors, the sort of memory safety promises and like now whose job is it to do what is broken. And it's not that they couldn't be agreed upon, but a lot of the libraries that are out there, they were written with the C Python's promises around memory ownership and threading, and the global interpreter lock all these all these conventions that it has.

So assuming that those things are there, and if you were to throw it into, say, Java, then maybe you would end up with race conditions or somebody deleting something before it didn't – Just the guarantees that the runtime makes for the C layer to depend upon no longer are there. And I think that that's where mostly things get broken is the C layers are making assumptions that may no longer be true in the other runtimes. I haven't done a lot of work at that layer, but that's my understanding.

[00:32:27] JM: Who actually needs to do this stuff? Is this like finance, like trading companies that need super-high performance or like who are the applications that actually need this like alternative execution path?

[00:32:39] MK: It's not usually the end developer that cares. It's usually the person making the library, and that library that – All of these different applications, depending on maybe it has some part that is too slow. I did leave an – And that person builds it. But if your application is built on top of those libraries, you all of a sudden have the limitations of those libraries.

[00:33:04] JM: Can you give an example? What would be a library example?

[00:33:07] MK: So one of the libraries that's really popular that is much, much faster for computation and stuff that is I think the most depended upon library and the data science third of Python or whatever. You think of the various places it's used, is NumPy. So it does like array transformations and matrix type of math and operations of large datasets all at once. Take all of the Xs and make them two times as big as they were, stuff like that. All that happens down and in C, because it's much, much faster to do. If you don't have the C layer working right, then you can't use that.

For example, the PyPy team tried to solve the problem by entirely re-implementing NumPy in Python so that they could jit compile it and still have it compatible. But, of course, that was sort of a bad option because they're always behind a little bit on the real one and all of a sudden they've become this parallel maintainer of another implementation that it doesn't really have much to do with what they want. But so many people needed it to be there. That was I think an attempt that had been done there.

A more simple one is SQL Alchemy I believe uses C, like a little bit of C code for the serialization, deserialization. SQL Alchemy is an ORM, object relational mapper, and it has to do transformations to and from what comes off the database back into Python and vice versa. And I believe there's some C code that does that in like the very, very, very tight loops way, way down.

So you want to be able to have that code to run that can actually run without those being in place, but it just runs more slowly. So there're just these foundational bits that people build on

top of. But if people do want to make their code faster – You talked about the compilation story. There's one other class that's really interesting, and that's called these types of things, either Numbug or Cython. C Python, but drop the P. Cython.

And what that does is it lets you put type annotations on your Python code and then it literally compiles it to C machine instructions. And then it's superfast and it has other – It allows you to escape other limitations of the way the Python is normally executed. So if you wrote a program, maybe you don't actually write it in C. You maybe write it a little bit in Cython, which is almost Python. But the effect of consuming it is still you're doing the C interaction, because not this machine layer interaction, because the way it actually compiles.

I started out by saying it's complicated. It kind of is. But it seems to be the real challenges have to do with so many of the libraries out there that need to go fast, have gone to this, "We're going to have a little bit of C code that runs to make it fast, much, much faster than maybe even, say, Java or C# or some of these jit compiled languages, because it's truly native C. But then if you go to the other implementations, often those are not valid anymore and it's complicated.

[SPONSOR MESSAGE]

[00:36:15] JM: Looking for a job is painful, and if you were in software and you have the skillset needed to get a job in technology, it can sometimes seem very strange that it takes so long to find a job that is a good fit for you.

Vettery is an online hiring marketplace to connects highly-qualified workers with top companies. Vettery keeps the quality of workers and companies on the platform high, because Vettery vets both workers and companies. Access is exclusive, and you can apply to find a job through Vttery by going to vettery.com/sedaily. That's V-E-T-T-E-R-Y.com/sedaily.

Once you're accepted to Vettery, you have access to a modern hiring process. You can set preferences for location, experience level, salary requirements and other parameters so that you only get job opportunities that appeal to you. No more of those recruiters sending you blind messages that say they are looking for a Java rock star with 35 years of experience who's willing to relocate to Antarctica. We all know that there is a better way to find a job.

So check out vettery.com/sedaily and get a $300 sign-up bonus if you accept a job through Vettery. Vettery is changing the way people get hired and the way that people hire. So check out vettery.com/sedaily and get a $300 sign-up bonus if you accept a job through Vettery. That's V-E-T-T-E-R-Y.com/sedaily. Thank you to Vettery for being a sponsor of Software Engineering Daily.

[INTERVIEW CONTINUED]

[00:38:05] JM: I find this compilation path stuff pretty interesting. And I don't know if you've looked into WebAssembly much, but my sense is that there's going to be some interesting stories coming out of there. I don't know how much that will overlap with the Python world. I could imagine people wanting to write machine learning models in Python and then have those execute in the browser, for example, on WebAssembly. That could be interesting.

[00:38:30] MK: I think WebAssembly is super interesting, and I think a lot of – I think there's going to be a lot of interesting stuff coming out of it. We have this project called Py Oxidizer. No. Sorry. Pyodide. These are very similar names, but not actually at all the same. So Py Oxidizer is something else.

[00:38:47] JM: The scientists have really taken over the Python community it seems.

[00:38:52] MK: Yeah, they have. So Pyodide is this project by some folks at Mozilla where they've taken some of the major libraries that we spoke about that compiles a C, like NumPy and some of the other ones, and they've actually compiled them down along with the C Python runtime itself into WebAssembly.

So you can go to a webpage. It just downloads Python and some of the libraries as WebAssembly, and then you can just execute native Python right there over WebAssembly, which is pretty interesting. The challenge is it doesn't make – Usually it doesn't make a ton of sense to like, say, "Hey, we're going to take this 20 meg WebAssembly or whatever it turns out to be of compiling up all the Python into this WebAssembly version and just shipping it. Nobody

in their phone is going to have a good time going and getting 50 megs of data and then trying to execute it and all that of WebAssembly bits.

However, there's definitely some talk and some thinking around of how might there be like some essential Python or maybe a WebAssembly native version of Python that makes more sense that maybe gives up a lot of the other stuff. We have TKInter as built into the C Python distribution. You don't need to ship that. That's a GUI framework. Who cares? You don't do that. So how could you maybe have like an essential Python and just a few of the libraries that come along and things like that?

So there's some thinking about how it might be involved, and there's some place that might make sense today. I don't know if you build in an electron-based app, for example, right? It doesn't matter that you ship a 20 meg binary, because you're already shipping Chrome. Who cares?

[00:40:40] JM: Right. If you have a bunch of electron apps open, they're each running Chrome, right? You can't share Chrome between them.

[00:40:49] MK: It's so bad. I know.

[00:40:51] JM: Pretty funny.

[00:40:51] MK: Slack. I got my other chat window. Yeah, all of them. Yeah.

[00:40:56] JM: Yeah. So you spent several years working on a PhD in math before getting into programming. I'm sure this is something that a lot of people who listen to your Python podcasts can identify with. I don't know the extent to which you've talked about it on your podcast, but so many of the people who are learning Python and podcasts factor into the learning process. I think they play quite a big role in the learning process.

[00:41:27] MK: Yeah, tons of beginners actually listen to it. Almost like language immersion. It really surprised me, but they're like –

[00:41:33] JM: Right. That's a great analogy.

[00:41:35] MK: Yeah, I'm sure for you as well, people do listen in that fashion.

[00:41:38] JM: Well, I've had this conversation with a number of different podcasters, like the question of why people listen to podcasts. Are you actually actively learning in the same way that you might be in like sitting in front of a lecturer, absorbing the information like a sponge, and it's hard to know, or are you like kind of listening to – Yeah, I think language immersion is a really good analogy, because it's like you're not exactly retaining it on like a factual fact-to-fact basis where if somebody handed you a bubbling quiz, you would answer the questions correctly. But we know, empirically, at least I know empirically from my own experience, I do retain something. There's something going on here. I am just picking up these bits and pieces now and then. So, I don't know. It's an interesting question. I think is multifaceted.

What I want to ask you ways this fact that there are so many people who are going from academia, like advanced academia, like as in I bet my entire life on academia and then figured out there's something deeply wrong here. I've made a mistake, or there's just something deeply wrong in the academic industry and I don't even want to be a part of it. What's your take on the state of academia and just the idea of the PhD. Does that even mean anything? Is that still construct that we should even have in our society?

[00:43:15] MK: Yeah, it's really interesting, right. I studied math. I loved math, but what I found out as I got further and further along was that I felt like at some point I was solving these theorems and problems and puzzles, like deep topological stuff or complex analysis or –

[00:43:38] JM: Hey, it's basic science. You're doing basic science. That's what they tell you.

[00:43:41] MK: That is what they tell you.

[00:43:43] JM: Basic science.

[00:43:43] MK: But I felt like – At least in math, it was pushed so far that like you get to the point where there's five other people in the world that even care about the question. He may not even

care about the answer, but they may care about the question. It's so incredibly small, and I also found that I basically gotten into programming because of my math work. We are doing a bunch of research and working on like Silicon graphics mainframe mainframes and just doing really cool C++ and OpenGL work.

After a couple years I thought, "Well, this is so much more fun and rewarding, and I could you make a difference for many, many more people." And at the same time, my wife is also – She got her PhD, and she's a professor. So I have that side of it as well. I think the answer really comes down to like there are certain things. If you deeply want to work with them, you basically have to have a PhD, maybe necessarily so. If you want to be a particle physicists, if you want to be an astronomer, if you want to be – I don't know, something in that realm. Pretty much PhD is what you have to do, or you can't work in that space.

Maybe if you had a bachelors and really good programming degree and a super passion to be part of it, you could find your way into that world as well. But there's still a lot of gatekeepers in that space. In the computer programming space, not so much. I mean, it's kind of merit-based to some degree.

I mean, there're obviously different opportunities for different people. But if you can code really well and you can demonstrate that, like you pretty much can get a job. It doesn't matter even if you have a bachelor's degree much of the time. So I think it really depends on what you want to do. But I do think there's something fairly broken about academics and just the job flexibility. It's such a weird contrast, because my wife, she has a ton of flexibility. She can work from home. She can go to her office. She can work from a coffee shop. She gets to travel, but only if she wants to. There's a lot of flexibility in what she does and what she researches, but if she wants to, say, to change jobs, there's like one time a year where jobs are offered, and you've got to like start preparing for that like nine months before so that you could be ready to participate in that job offering. And there might be like 20 positions in the United States for exactly her role. And that sort of lock-in to like a few places, a few jobs, it's incredibly weird how small academics is in that regard. So I do know. It's a funny contrast. But I got to say given the possibility of being both, I am super glad I'm not in it.

[00:46:34] JM: I mean, this artifice around publisher perish and just the scarcity mentality that – I don't know. And then you have like the defenders of it. The defenders of it or the people who are like defending it from the basic science point of view, there is like loss aversion there. Are you defending this really from a place of rationality or is it more loss aversion? You're just like consoling yourself with these arguments. Because from the outside looking in, I'm like, "Ooh! What are you doing to yourself? You're in a world of madness. You might as well be in North Korea." Compared to – I mean, maybe that's take it to the extreme, but it just seems like there is so much perversion and madness.

Look. I got an undergraduate degree, and I was kicking and screaming through the end of it. Like I almost quit near the end of it. Just the undergraduate side of things seemed like – It seemed slightly insane to me. And as I've gone through the software industry and taking a look at the boot camp side of things, you see what the other side of rationality looks like, and you see, "Wow! Okay. There's a really much better way to do this." I don't know.

[00:47:53] MK: It's funny. Yeah, I don't know. So I have some online training courses and stuff people can take, and I look at what, say, like a semester-long programming course might cover. And then I look at, say, like an eight-hour course that I've created covers. And they're kind of the same. I'm like, "Oh! Yeah, you could either spend like a dedicated week or tons of money in half a year almost and kind of be in the same place. It just seems – I don't know. It seems inefficient in that regard.

[00:48:25] JM: Exactly. Then people will say, "Oh! But you get to be in the same room as a great professor." Does that matter? I mean, first of all, if the professor just took an eye to trying to do mass YouTube accessibility, and then like if you need to have the one-on-one calls, schedule a Zoom call. Like telemedicine works. Telemedicine works really well for cognitive behavioral therapy, and like diagnosis of it of actual illnesses and stuff. I think it works equally well for what you need out of a professor. Anyway, this is really getting off on a tangent.

[00:49:05] MK: Yeah. I have one final thought and we can wrap it up. I mean, I loved the college experience. I loved being on college campuses. I am really grateful that I studied, say, the history of Western philosophy, and that I studied quantum mechanics, and that I studied calculus. But what I think is interesting is I don't know that that necessarily should be tied to

career. I think it was a great sort of maturing rounding out of me as a human being, but I don't know that that needs to say, “And now, that was career prep.”

I think that we tie college and career prep a little bit too much for a lot of things, and we need something more – Like a better apprenticeship stuff, like, say, Germany has, for example. We don't necessarily have to go down that path, or it doesn't have to be the path that leads to the job.

[00:49:53] JM: All right. Let’s come back to podcasting a little bit, and I guess entrepreneurship. So you are an entrepreneur. You do a variety of things. Podcasting fits in somewhere. Can you describe to me what is like the structure of your business and how does podcasting fit into it?

[00:50:09] MK: Sure. So what I do for my full-time job. I don't do really any consulting these days, very little in-person training or anything like that. What I do is I have my podcasts. I have these two podcasts. We sell ads on the podcasts, much like you do on yours. The podcasts are free for everyone. We have lots of listeners. And then companies sponsor those shows and pay me money directly into the business to reach that audience. And that sets what I considered like a baseline of sort of stability for me to be more risky and trying other things.

So the podcasts serves two important roles. Obviously, it has some of the money from the ad revenue, but also it's very, very hard when you're running a business to reach people. I would say the hardest part is marketing and getting the word out these days even more than creating that the thing itself. So the other part of my business is I have an online training business for software developers around Python.

So people can go take beginner courses on Python. They can take deep dives into asynchronous programming. We recently went on this kick of 100 days of codes. We have two courses that do like 100 days of Python. One in standard Python and one for web apps, and those were huge projects. They took like nine months to create. Me and a couple other folks put those together. That was fun.

So we sell those classes to just the general public, sometimes to companies. Have agreements there, but a lot of times people just come and they’ll buy one class or they’ll buy a bundle of

classes or something like that. So that sort of rounds out the whole thing. I have the podcasts around Python. I have the courses. And the way that I see it is that podcasts can make you curious. They can inspire you, but kind of like you said before, like if you're given a bubbling test or you're said to write this code with this framework, like there's no hope.

So the courses are there to take people who are inspired or interested about a subject and take them from interested to can do programming things with that topic, and that's kind of how I see the spectrum, is like get people interested and excited and then give them something actionable to learn about it. That's the business.

[00:52:23] JM: How does podcasting fit into your life?

[00:52:25] MK: Into my life? Oh! It's really interesting. It was just this thing I started out doing over my lunch hour or I'd get up an hour early before I go to my regular job, and it's just – It's grown to be really, really important. Like I said, it was basically the –

[00:52:41] JM: By the way, I'm asking about consumption of podcasts as well as podcasting.

[00:52:45] MK: Sure. Okay. So on my side, like it was the thing that let me quit my full-time job and start off on my own. So it's really important the stuff that I create on that side and just it's got a special place in my heart, for sure. But, I don't know, podcasting just opens up a world as a consumer to what I felt like they're used to exist, but it doesn't seem to exist very much before.

I mean, remember the Discovery Channel? You could go to the Discovery Channel and you could turn it on and you could see stuff about like science, and engineering and math, and planets. And now it's just reality shows of lumberjacks. What happened there? That used to exist, but that definitely exists even better in the podcasting world. What are you into?

There is probably like a community that you can be part of and you can hear about and interact with on a weekly or daily basis that is really into that thing as well. I bet there's an awesome podcast where people that grow competition pumpkins, like the thousand pound pumpkins. There's got to be one, right? I don't care about it. But if I were into pumpkins, that is the one place where I can have an audiovisual experience around that. Maybe there's like a forearm or

something, but there's probably not a TV channel or something like that or a radio show. But there's just – It doesn't matter what you're into. There's probably something that's like really well done and meaningful in the podcasting space.

Also, I said I was a busy person. I have kids and jobs and all that. But podcasting, there's always a place for podcasting in your day right. You cook, you do dishes, you take care the lawn, you paint the house, you drive possibly to and from work. Podcasting fits in there. Video doesn't, like reading doesn't. It just fills the gaps and lets you make idle time insanely powerful exploration of ideas. That's what I think.

[00:54:56] JM: Michael Kennedy, thank you for coming on the show. It's been really fun talking to you.

[00:55:00] MK: Thank you. It's been great to be here. Thanks for inviting me.

[END OF INTERVIEW]

[00:55:11] JM: Software Engineering Daily reaches 30,000 engineers every week day, and 250,000 engineers every month. If you'd like to sponsor Software Engineering Daily, send us an email, sponsor@softwareengineeringdaily.com. Reaching developers and technical audiences is not easy, and we've spent the last four years developing a trusted relationship with our audience. We don't accept every advertiser, because we work closely with our advertisers and we make sure that the product is something that can be useful to our listeners. Developers are always looking to save time and money, and developers are happy to purchase products that fulfill this goal.

You can send us an email at sponsor@softwareengineering.com even if you're just curious about sponsorships. You can feel free to send us an email with a variety of sponsorship packages and options.

Thanks for listening.

[END]

