>> SPEAKER: Hi, everyone. Welcome back. My name is Ethan, and I'm here to meet the PyMarc in 30 minutes. So, thank you for joining me today. My goal for today is to convince you that you can run a Python script that will allow to you to manipulate data in customizable ways. I want to tell you about who this presentation is for, because if you're into libraries and Python, you probably already know about PyMarc. So, maybe this is you, you've said this at some time; I'm interested in tech, but I haven't tried Python. I'm interested in automation. I'm intimidated by technology, but I'm interested and willing to give it a go. I like it, but wish it was more flexible. I have several steps as part of my workflows and would like to string them together. I'm tired of Excel bogging down. So, if any of that's you, I encourage you to stay with us, and I want to emphasize us, because, look, all these people who are here today joining us for this session are also interested in using PyMarc and Python and learning about these things, so we have a community. The forum for this talk has been pretty quiet so far, but I imagine afterwards, we can ratchet it up and really get the conversation going. We're going to go through the installation process really, really fast. We're going to go through the installation process really, really fast, because so many of you are actually at your work stations right now, and we'll discuss a few of the things that you can do with Python and PyMarc, and when we're done, you won't be a master or expert, but you'll see there's a shallow learning curve and an incredible value to learning Python. So, who am I, and why am I the person you should listen to on this? Well, hi. I'm Ethan Fenichel. I'm a librarian in a cataloging unit who likes to automate stuff while respecting the need for well-formed meta data. I was lucky enough to have the chance in school to learn Python and later PyMarc, and I've been working with it for a couple of years. I found it really helpful and thought I should share, and fortunately, the ALCTS Exchange organizers thought so too. So, some projects I've done with Python and PyMarc include URL access checks, e-book error checks, geographic reconciliation, comparison of local records in my ILS to OCLC records.

So, it's worth noting that I am self-taught at coding, and the amount I don't know and don't understand which would be a far less interesting but much longer talk. As such, there are likely folks joining in today who know as much as I do about Python and PyMarc, and I want to hear from you. I encourage you to share all your thoughts in the chat as we go, and in particular, if you have examples of Python or PyMarc that you've written, share those links. I want to mention that today, we're going to focus not on learning about a problem or a solution, but we're going to learn on a tool, we're going to learn about a tool that can help with that as we're thinking about the future tools that we need. It's a powerful tool, and this tool can help us get to solutions, and those solutions that we create are what gives our community this power. We're going to quickly run through the download steps, and I promise, by the time we're done, we will have written a script with PyMarc. Instructions have been previously provided and are available in the forum. So, since most of you aren't at your work stations, I expect that you'll be rewatching this at this time, so those will come in handy. I want to make a quick note about those who need permission to install software on your work stations, like I do. You should work with your system administrators for that. Hopefully, they'll be encouraging, but it could vary your installation process just slightly, so don't worry, my system administrators have been fantastically supportive and helpful. Um, why are we going to learn Python? It's a scripting language, and in many important ways, it's similar to other scripting languages, such as ruby pearl, Java Script. If you have some familiarity with those things, you will a big leg up. If you have familiarity with Excel formulas,

you're not that far off either. Is Python superior to the other options? Not necessarily, but it is named after monte Python, so if you're familiar with that, it should be good for a chuckle. It's definitely the funniest named scripting language.

Much of what you might learn with Python would carry over to the other languages, so the big differences will be syntax. A quick note, if you're using an Apple computer, congrats, you already have Python, it comes on most Apple computers. This presentation is going to focus on the windows installation. If you're using another type of operating system, you should also have Python too on that machine. If you're using a Chrome Book, you might want to look at some cloud-based solutions, and I suggested Chrome Books, although I've never actually used it, I hear it's great. So, to download Python, if you just go to Python.org, the installation is super easy. If you do a web search for it, you'll get some things about snakes, but mostly about this scripting language. Python is currently available in two flavors, Python three and Python two. I'm going to focus on Python three. It works fine with everything I've done with PyMarc, and again, there's some small syntactical differences. The latest version of Python is 3.6.1, and you just click on that big yellow button. As I mentioned, there's not, we're going to go with the Python three version, not the Python two version. Once you click on that download button, you will be able to, an executable file will download, and once that finishes pretty quickly, you'll just want to run it as administrator, if applicable. If you can, go ahead and install for all users, and then you're going to want to select this little add Python 3.6.1. Next, you'll select the customize installation, and you don't have to change any of these options, you'll just click next. On the advanced options, you want to click install for all users, and you might want to change your install location to just Python 3.6 instead of Python 3.6-two. Once that finishes, you'll click install, it'll finish, and then you're done, you've got Python on your machine. You can test this by just going into a command line or command prompt or into the power shell, which is just start, and then it'll open up in this, and you can just type Python-M, and it will recognize that you're trying to do a Python thing. Don't worry, we won't be in the command line too much more though.

Um, next, we're going to install PyCharm, which is a fancy interface for working with Python. It's called an integrated development environment. It's sort of like working with Microsoft Word for Word processing instead of Note Pad. It provides support, hence formatting, and allows you to manage everything about your scripts in one place. There are other IDEs, and I'm not for one or the other, but I can tell you that I've had more success with learning Python once I switched to using PyCharm, and there are others out there as well. It's open source and very easy to install, and again, if you do a web search for PyCharm, you'll definitely find it. You'll click on the big download button, and it'll take you to a page that gives you the choice between the professional and community version. The community version is the one we're going to download. You're welcome to use the professional one, if you want to pay for it. Once it downloads, go ahead and run the executable file. For this installation, I'm going to do all the defaults, so it'll just look like this. Again, you'll have these slides to refer back to. When you click finish, you've got PyCharm. We're going to configure it now real quick once we open it. To do that, you'll just, um, go to this configure button, and you will go to, this will take you to the settings, and you'll click on the project interpreter, because on this little drop-down, again, project interpreter, click show all. From there, it's going to pop up here, you won't have anything yet because we haven't

selected it, so you'll click on the little green plus sign and click add local. From there, you will find the Python executable that we just installed, and now you are configured.

Next, we're going to create a project, and on that front landing page, it's very prominent how to create project. I'm going to name mine PyMarc M30 and leave it in the default location. Once that opens, we're going to right-click on the project directory and click new, and we're going to create our first Python file. It's going to be this type. So, let's briefly look at the parts of PyCharm, and, you know, I'm going to just leave this here, and we'll come back to it, because I think that's probably going to be too small for most of you to see. I apologize for that, but we'll look at PyCharm live in a moment. Instead, we're going to install PyMarc. So, to install PyMarc, you can do it two ways. If you do something in a Python-y way, it's called Pythonic. The Pythonic way of installing PyMarc is to use something called, or something called pip, and to do that, you're going to go to command line, the last thing we do in the command line, and we'll do it so you don't have to do it in the command line, and from wherever you are, you'll do Python-M pip install PyMarc. Pip is a built-in program that's built into Python, and it's going to go out and find the right installation, and then it's going to install it, and once that says successfully installed PyMarc, you're all done. If you want to do it through PyCharm, you can, so that way, you don't have to worry about the command line, and to do it, you can also press control alt S. From there, it'll take you back to the project's interpreter, and you can click on this little plus sign. This little box here is showing you all the different libraries that you have installed, and I had a few more, but you'll probably not have too many here. Click on this little plus sign, and it'll give you a place to search for available packages. You'll type in PyMarc in the top and then click install package, and then it will install. It's just that easy. If you do run into any problems, I'll be keeping an eye on the forum, and please share, we'll get them worked out. So, that's pretty much it.

So now I'm going to go live to my Python screen, and we're going to actually look at how this works. So, here is PyCharm, and I think that you are seeing it. Um, so, the various parts of it, over here in the top left, you've got the directory for the project, and there's a few files that we'll look at in a moment. On the bottom, when you first open it, it'll look like this. Over here is your script. Now, this will show as empty for you, and I've populated it, so it'll look more like that, and you won't have these here, but, um, they're just other open files. Don't here on the bottom, you're going to have your console and your terminal. The terminal works just like the command line. If I open that up, you'll see, and your console is, you can always recognize a console by the greater than signs. This is where you can really play around with Python. You can check your code and work on things here. Um, I don't use this, so I'm not really going to cover it. If you want to test your installation of PyMarc, you can always go import PyMarc, and that will show you everything that has been installed, or that will show you that, if you don't get an error, that shows you it was installed correctly. Before we can get into using PyMarc, I need to show you some basic Python stuff. We're going to go over that, and we're going to go over how to read a file, how to write a file, and we're going to talk about the components that are within it. So, this function here, um, and this is my function, you can always tell a function because it says DES to define the function, and then you're going to name the function, you're going to use these open and closed parentheses to define any parameters. This function is called read file, and you're done with this line by adding a colon. You'll notice there's a lot of tabs and spaces, and PyCharm is really good for

helping you keep track of them. We're going to use the library which is built into Python, and, so, we're going to need to just import it, just like we did PyMarc. You don't have to install it though.

Next, we're going to create a variable called file name. So, this is how you assign a variable, you just do file name equals, and then I'm going to pass in the location of a file called list of journals. This file name is in the same directory as my script, and so I don't have to specify anything else. Let's take a closer look at assigning variables. I can assign a variable by just saying A is equal to Ethan, now A is equal to Ethan. You go also do this with a number, so B is equal to 10, B is 10. I can test that they're equivalent by saying A equals equals B. Of course, that's false, but if I do A equals equals Ethan, it's true. I can also test the relativeness of fields, so if B is greater than 5, it's going to show true, and so that's just very simply how you assign and compare variables. File name is just a variable as well So, we're looking at this one, um, and then we're going to have another one called checklist. Checklist is another variable. In this case, checklist is the variable for an empty list. This open and closed bracket is a list type. We can always check the type of variable by just typing type A, variable, and it's going to tell me it's the type string. Checklist equals open and closed list, if I do that, you can see it's going to tell me that's a list. This is a good spot when I talk about this construction for with open that, um, the open, when you write scripts, much like if you do in cataloging or really any problem-solving, you don't generally do it from scratch. So, this construction of the with open, you're going to see it all the time, and, um, it's just a good construction to use. When you're doing a CSV file, you'll just use CSV.reader here, and this parameter here, this R just means we're going to read it.

So, we're going to read this file name, which will be here, and then we're going to open that, and then while we're in it, we're going to use this construction called a for-loop, and we'll talk about this in a moment. This means for each instance of things in a thing, I'll explain more, um, it's going to do something. So, in this case, reader is an object, it's got a whole bunch of things in it. This is the file we're going to read, and you can see it's got a bunch of lines in it. Each line represents, um, in the CSV file, it's an identifier from my ILS, a journal title name, the vendor, if known, and then an ISSN. So, this for-loop is going to have all this data in this object called reader, and so it's going to go through it and say for each row in that object, do something. In this case, what we're going to do is say, remember checklist that was here? We're going to append that instance, that row, into our empty list checklist so that it's no longer empty, and then when we're done with our function, we're just going to send back this checklist, and I'll show you what I mean by that So, whenever you run a function in Python, in the console, the first thing you have to do is get it into memory. That just means hit enter, and now it's in memory. What's nice is I can use typing, so if I type this re, it'll show me my options, and that includes this function read file. So, um, checklist, we know is an empty nest, but I can reassign it to be the result of this function, read file. So, it's going to run this file, and at the end, it's going to return checklist, which was this empty nest that keeps getting rows jammed into it. So now, checklist has all this data in it. It's got a little formatting weirdness here, which we would work out, if need be, but it's got each one of these rows. Each row is a list, so my list is a list of lists, and that is something that, for Python, is very normal. I want to show you a little bit more about this for-loop construction. I can use this again, so for each, for the journal checklist, this is just a variable, it doesn't matter what you call it, I can call it row, or I can call it journal.

For journal and checklist, print journal, and now it's going to go through and print each one. So, that's what that for-loop does. It's going to do this for each, and I can do lots of things with it. I want to show you something else real quick, because, um, there's really only two more things we need from basic Python before we jump to, um, PyMarc, and that's, the first thing is going to be called slicing. Slicing allows me to get parts back from my, um, my data. In this case, remember my variable A, which was Ethan? If I just want the first letter of Ethan, I can slice it by doing this open and closed bracket construction. A 0 to 1 means get me from this first bit to the 1 bit, which is the second bit, and return that. Likewise, if I leave this open, it'll default to zero, so if I say show me the first three, it'll slice that string into the first three bits. I can also leave the second one, and that one will get me to the end, so from the last two. The last way you can do this is with negative one, and that'll actually give you the last one. So, that's slicing, slicing a string. I can also slice my list. So, remember, sorry, checklist was this whole big, long list. Well, what if I just want the first variable? I can get that back, and I can slice to get the first two, and onward and onward. So, that's how slicing works, and you can think about how this might be useful in your marked fields, if you just want to see part of your marked field, and we'll look at fixed fields in a moment. The last thing I want to show you with basic Python is how to write your data to a file. So, in this function, we're going to use a very similar read file. We're going to call it write results. This time, it's going to have a parameter, which is the results string. Again, the naming of it is arbitrary. The variable here, results log, is going to tell it where to write it. You can see right now, it doesn't exist. Python is really great, it will create the file if it doesn't exist already.

We're going to use this same with open construction, but this time, instead of an R, we'll use an A, which stands for append, and we're going to say as X, and X is just an arbitrary object, it's a thing. So, for each time X is opened, we're going to write the parameter, and then we're going to add. This dash N is the Python way, and that just means add a blank line. So, I'm going to put it into my memory, and now I can call it, and this is where we're going to see the power of calling a function. So, let's test it first. Write results, remember, my variable A was Ethan, I can say write that. I can say write B, which was, remember, B was 10. I can say write X, and that's going to write into my results log too. Let's take a look at the results log real quick, and one little trick here is I can always, oh, there it is. I was waiting for it to pop up. Now it's there. It'll show up here, and you can see those three things wrote to the X. Um, next, what we'll do is we are going to, we can call this from, um, within our for-loop. So, for row and checklist, we're going to write results, we're going to write that to our file, and what's cool is now you can see all those wrote to the file, so it's really powerful. All right, so, PyMarc, really quickly. PyMarc works on three different levels. It works on our, um, the construction of the file, it works on the construction of a record, and also on a field. So, how do you create a new record in PyMarc? We're going to create a new record. So, we'll say PyMarc.new, PyMarc.record, and I'm using my type ahead, and now, new record is a PyMarc record, and I can print it, and it's just going to show up as the leader.

Let's say I want to add a field to it, and I'm going to add an 856 URL. Let's look at this function real quick. It's going to take a parameter of a URL, which I'll type in, and it's going to use PyMarc.field, and then these parameters here to, um, add an 856 URL. So, um, I'll create this, I'm going to call it U equals create URL, and we'll say www.google.com, and now what I can do is I can append this to my new record. Now if I print new record, you can see it's now got my 856, and if I want to write this to a file,

I've got a little script here for doing that as well, and you'll see, it looks very similar to the construction we just had. All right, so, we'll test that out real quick. We'll just do write mark, new record, and that will then print out to, um, to our explorer, and you can see, I've got my mark file here, and it looks like a mark file, if I show you in mark edit. So, I wanted to show you one more thing as well, but we're running really low on time, so at this point, I want to just show you quickly, there's a way to read in a file as well, and we're going to do that, and, um, R equals mark read, now I've got a record in R, and I can start calling things from it, and this construction will work. It's got a bunch of different other features you can use with that, and then if I want to, I can also, um, look at, say, a fixed field, like that, and I can slice it as well, so instead of assigning it, I can do this, and let's say I want to see 18 to 19, well, you might know that this is how you would tell if it's electronic or not. So, if I want to see all of R, I just want to show you one more thing real quick, if you want to get multiple fields, you would get this, um, you would just use this construction contributors for the 700, R.get-fields, and now I've got all my contributors in this object.

All right, um, so, I'm going to jump, that's really it, that's the basics for PyMarc, and, um, I want to jump back in now to my presentation, and, so, I'm really interested in your questions. There's a bunch of additional resources here for learning, including a slack channel for librarians who want to share Python stuff. I'm interested in, so, if you want to start asking any questions in chat, I'll be interested in hearing about them, and also the forum. Some additional resources here, in particular this, um, augmenting the cataloger's bag of tricks, and then feel free to check out my GitHub for other examples of code that I referred to before. There's a question about whether technical services librarians should learn to code. I think for me, it's been a privilege to do so. I think it's a good question. We can definitely talk about it. I don't have an answer. Then there's, um, my sources are also available in the forum. So, I know we just have a moment left, but I'm interested, if you have any questions or if you want to see anything else, but that's all the powerful stuff you can do with PyMarc. Again, this is just the tool, and it's up to you now to start coming up with how you apply this tool to your problems to create amazing solutions.

>> SPEAKER: Thank you, Ethan.

>> SPEAKER: And I think, actually, if you can send your questions to the forum, I would be, I think that would be the best place for me to answer them at this point. So, thank you, everyone.

>> SPEAKER: Thank you, Ethan.