Nonvisual Python (Nonvisual Data Science Workshop #1)

(0:01 - 0:30)

Hi all, and welcome to the non-visual Python workshop. This is going to be the first workshop of a five-workshop series on non-visual data science in Python. And this workshop series is led by myself, Patrick Smyth, and also Sarah Kane, and it's funded by a grant from NumFocus and Pandas, so thank you very much for supporting that.

(0:32 - 4:27)

And so in this workshop series, we'll be learning how to work with data without vision, sight, or using much of the focus of many introductory data science tutorials, workshops, and so on, which is on visualization. So in this workshop series, we'll be working up to creating a sonification of a data set. We'll be putting together that data set, and on the way, we'll be learning for the fundamentals of the Python programming language, some fundamentals of Pandas, which is a data science library or tool that's very widely used, and also for sonification, we'll be using Astronify, which is a library, a tool for sonification.

And so in this workshop series, we'll be using or we'll be demoing for you NVDA, non-visual desktop access, the screen reader, and as we use Python. You can use other screen readers. I've heard, we've heard good reports that JAWS works well with this tutorial with very similar shortcuts, but what you will be seeing and hearing is NVDA.

So just want to explain a little bit about what is Python, what is, and what are the tools that we're going to be using in this workshop series. So the Python is a, it's what's called a high level programming language. So if you imagine that the computer being kind of a stack from high to low, and then on the high end of that stack, there will be languages that are a little closer to how people think, how people talk, and how people would deal with problems.

And on the low end of that is, so a low level language would be how the computer thinks in terms of numbers, data, and so on. So Python is a high level programming language, meaning it's a little closer to how we as people think. The trade-off there of course is in speed.

The computers are so fast these days that often that trade-off is kind of one we're willing to make. So Python is, it's been around for quite a while now. I think it was created in 1989.

It's a programming language that's used widely in industry and the academy. It's used to create applications. It's used to perform what are called scripting tasks, which are essentially automated tasks that you make on your computer, say automating, moving around files and folders and so on.

But for our purposes, it's also used widely for data science. And the very short, data science is an application of technology and statistics for studying data. And it's sort of a, people who are data scientists, they describe them, they do a wide variety of things, but it kind of grew out of the mathematical discipline of statistics and so on.

And these days it's a very technological discipline where often programming and other technical skills are, if not necessary, then incredibly useful. So what are we, how are we going to be accessing Python today? We'll be switching now to, in a second, to the environment that we'll be using to interact with Python, which is called iPython. But for this workshop, you should have installed two pieces of software.

(4:28 - 5:15)

The first is NVDA, non-visual desktop access. That's the screen reader we're going to be using to interact with our computer. And then the second is Anaconda, which is a, what's called a distribution of Python.

So Python, you can go to the Python website and download sort of the standard Python installation, which is called CPython. And that comes with a whole bunch of useful tools and so on. It's for a general purpose.

What we're asking you to install is, which is called Anaconda. And you can go to some search engine and type Anaconda download, and it should be the first page that comes up. But what Anaconda is, it's a distribution of Python.

(5:16 - 6:51)

So it's basically what you would get if you installed sort of the default Python, plus a bunch of extra tools that are useful for data science specifically. It is a little bit big, so make sure you have a little extra room on your hard drive. I think it's something like a gigabyte, and it can take a little while to install.

But once you have it, you can get a whole bunch of additional functionality, additional tools, specifically related to data science. Unfortunately, not all of the tools that come to Anaconda are accessible to the blind and screen reader users. Notably, Jupyter Notebooks, which are one of the main ways people interact with Python for data science, those are not currently 100% accessible.

There's definitely been improvements over the last couple years in usability for screen reader users, but it's not quite there. It's still a little bit clunky. What we will be using after we install Anaconda is called the Anaconda prompt, and specifically IPython.

So we'll be opening that up in a minute, and you'll see in here what it looks like. But IPython essentially is kind of a fancy program to interact directly with Python, to have a conversation with Python. I'll explain a little bit more about what IPython does over the course of this workshop.

(6:54 - 9:32)

Before we move on to the part of the workshop where we actually start writing some code and working with Python, I just want to thank a few people. This is a re-recording due to some technical issues of the workshop we had on February 6th, where we had a number of helpers to facilitate, and so I just want to thank the helpers that were there. So those are Alex Ogden, Elizabeth Sola, Sarah Kane, who is leading this workshop series with me, Stephen Zweibel, Monica Thew, and Paul Alexander-Bloom.

So thank you to our helpers who helped out on February 6th, 2024. And I also really want to thank Patrick Hofler, who sort of supported this initiative from within Pandas. So Patrick is a core developer at Pandas, so I'd just like to thank Patrick.

And just to say who I am, I'm Patrick Smyth. I am a teacher, a writer, and a programmer. I am the chief learner at IOTA School, so we're a consultancy where we work with clients on accessibility, documentation, and infrastructure coding.

And you can learn more at IOTASchool.com. And I personally, I'm visually impaired. I'm a visually impaired programmer. I have maybe something like 2% vision remaining.

I have no central vision. I have retinitis pigmentosa. So I have a little residual vision that you may see me, you know, use occasionally, but mostly I'm a screen reader user.

And so I will, the focus of this workshop series is definitely on the screen reader use, but I will, you know, I know there are some low vision people and some sighted people who are following along with these workshops, so I will attempt also to explain some of the low vision and the, you know, sighted ways of doing some of these same things. Okay, so without further ado, let's transition to sort of demo mode here, and we will get to writing a little bit of Python. So I'm here on my Windows desktop, and the first thing we're going to do is, you know, if you haven't already, you should start NVDA.

(9:33 - 12:26)

I already have NVDA running. I'm going to turn it off and start it again just for demo purposes. That's the sound of NVDA turning off.

I'm going to press the Windows button and type NVDA and press enter. Excellent. We should have, we're good to go with NVDA.

So the next thing I'm going to do is, I am an eSpeak speech user, so the voice I use is a little scratchy sounding, so I'm going to make it, change the profile to be a little more of a friendly sounding voice, so NVDA menu, NVDA button and N, and I'm going to go to profile. Okay, and change it to a slightly nicer sounding voice, and then you should have

Anaconda installed. If not, you can go to, in any search engine type, Anaconda download, and it should be the first result.

Go ahead and install it. You can use the default settings as you install, and once you have Anaconda installed, what we want to use is the Anaconda prompt. So the Anaconda prompt is, essentially it's a command, it opens a command line application that we'll be using throughout this workshop series.

So let's go ahead and open that up. Okay, so press the Windows button.

Now we have search, so I'm going to type Anaconda. Okay, so I typed a little bit of the word Anaconda.

I typed a little bit of the word Anaconda, and Anaconda prompt was the first result. What you don't want is what is called the Anaconda navigator. So the Anaconda navigator is something that's only partially accessible.

It's just another part of this Anaconda distribution. We do not want that. What we want is the Anaconda prompt.

Okay, so make sure that that's what you're getting. I'm going to press enter to start that. Okay, and then I'm pausing it. So we have, if you listen now, you'll hear what we have in front of us, or what we're going to hear is essentially a prompt. Basically it's a command line and a prompt, meaning prompt is some text suggesting, oh, type something in here.

So listen for what that sounds like. "(base) C:\Users\Patrick>." It said C drive, users, Patrick, greater, and the greater is sort of the prompt.

The greater is kind of a little greater than sign, and that's suggesting, hey, type something in. When they say prompt, it's sort of like encouraging you to type something. So if you hear that, you should be in the right place.

(12:29 - 13:23)

And I want to do one thing before we move on, which is if you're using this video and you're totally blind or you're not using your vision for this, it won't be relevant to you. Or if you're sighted, it may not be relevant to you. But for low vision people, I want to give a little advice on making this environment maximally visible for a low vision usage.

So first of all, on Windows, I personally, I invert the colors on the whole OS. So I'm going to do that now. That's using the magnifier.

When the magnifier is activated, and to start the magnifier, you can hold down the Windows button and press the plus key, which will zoom in. But if you don't have it open, it will open magnifier. Once you have magnifier open, Windows magnifier, you can use control alt I, and that will invert the colors on your OS.

(13:25 - 13:35)

That can be obviously very useful if you're not using it already. I'm going to maximize this application that we have. So I'm going to hold on Windows, the Windows button and press up.

(13:36 - 21:10)

So that kind of maximizes the application automatically using Windows up. And then what I want to do is show you how if you if you would like to make the text extremely large, as I have on my screen. So let's pull down the I think it is control space. Maybe alt space, hold down alt and press space. Yeah, there we go.

And then you can press the P button. So it's alt space, then you press the P button to open properties for this application. Or you can go through this menu, "move unavailable, size unavailable, minimize n, maximize unavailable, close C, edit E, defaults D, properties P" properties P or you can just press the P button. But I'm press enter now, ""Anaconda Prompt (Anaconda3)" Properties dialog true type fonts are recommended for high DPI displays as raster font. I'm going to go ahead and pause that. But that is what we're we have open is the properties for our for our Anaconda prompt application.

And essentially, the first of these properties is the font size. So if you press tab "font grouping, size combo box." So what I did was right there was I pressed tab and then I tabbed backward and then you hear it's a combo box for size and you can adjust the size up and down.

And I just maxed it out. So set at minus like 72 or 76 or something like that. They maxed it out, made it maximally large.

And if you go a little back, so shift tab backward "tab control tab selected", then you you have a bunch of tabs and one of the other tabs is color. So if you move along in the tabs, "layout tab select colors tab selected", there's a colors tab. So you can also if you're low vision and colors matter to you, then you can adjust the colors.

And in my case, I made it light because I now invert the whole OS. But you know, your mileage may vary, you should pick the colors that work best for you. Okay, so we can increase the font and change the colors.

You can also change the font if that is something that matters to you. So I'm going to go ahead and press escape to close this. Okay, and now we are in you know, we're in the place where we are right after we type Anaconda prompt and and we are our command line prompt.

And essentially what we're in right now is the it's a version of the Windows command line, also known as sometimes as CMD, where but it has a few extra things loaded into it.

That's what Anaconda prompt is. It's a version of the default Windows command line with some extra stuff loaded in that is useful for us, especially for like our purposes with data science and learning Python.

And right now we're using the sort of the general CMD command line. So you know, which is it can be useful. But what we actually want is to be talking directly to Python for the next part of this workshop series.

So let's go ahead and we're going to type the letter I and then the word Python. Okay, so letter I and the word Python, just like a big snake. Okay, so I, P, Y, T, H, O, N. So I just have the word IPython.

I'm going to press space just so you hear it again. I deleted the space. And then I'm going to press enter.

"Python 3.8.8 (default, Apr 13th, 2021. 1:5:08:03." So I'm going to pause that. So there's a bunch of text that gets printed out. But what you want to listen for is the Python version number. Okay.

And the Python version number will sort of tell you, okay, now we're actually talking to Python. Okay. So it's printed out a whole bunch of stuff.

We could resume it and hear the rest, but it's a little long. "[MSC v.1916 64 bit (AMD 64)". It's a whole bunch of information about this specific version of Python that we're running. But what we can do now is I'm going to teach you a shortcut that we'll use a bunch of times in this workshop, and that is to clear the screen.

Okay. Which can be very useful. So, so let's get rid of this extra information that we had printed out when Python starts.

We're going to hold down control and press L as in Lima. So control L "In [1]: . And what we heard was in left bracket one, right bracket colon. Can we have it say that again? "Space left bracket one, right bracket colon" in it's in left bracket one, right bracket colon. Okay. And that is basically waiting for us to type some Python in.

And let's, what we're going to do is we're going to type a little Python and then something will happen. Okay. So we're going to type, type something in, and then something will happen.

The obvious thing will happen, but then something else will happen. So I will explain both of these to you, but let's actually just go ahead and do it. So what we're going to do is we're going to do a little math.

Okay. So what I want you to do is type along with me and then I'll do some explaining. And throughout this workshop series, we'll kind of have that pattern a little bit where I will ask you to do something, go ahead and do it, and then I'll do a little explaining.

And just because I feel that it's easier to, to, to understand the explanation or for the explanation to make sense after you've sort of done it and, and, and heard the output or, or experienced the output. So let's go ahead and type in five, the number five, a space, a plus, a space, and another, a six. So it's a five plus space six.

So we, we heard, what we heard was "out left bracket one right bracket colon 11". So our output is 11. We typed five plus six and our output is 11.

There's a little extra stuff that appears before we get the output that we want. And I'm going to explain that now. But so, so what happened when we did this? Basically you type something in, in this case it was five plus six, you got an out, some, then something happened.

Python did a random, sort of its own little processes of simplifying the, what we gave it. So five plus six got turned into 11. And then it printed out the results to us or returned the result to us.

And then we got a prompt to given another line to Python. So basically we're doing, what we're doing is we're having a conversation with the machine where we give it code, something happens, and then we get something back. Okay.

And there's a fancy programmer word for this that I'm going to explain to you. And then we're going to use that word from now on when we talk about this process. And that is the fancy word, the fancy term is REPL.

So what we're using here is the Python REPL. And essentially it's a four stage process and that's exactly what I explained to you. So it's read, that is we type something into the computer, the computer reads it in.

(21:11 - 21:23)

Evaluate, that's the process of something happens, Python runs a process or it simplifies what we have. That's the evaluation. Then print, that's the computer returning something back to us.

(21:23 - 21:43)

And then loop, that's the process of the prompt happening again or waiting for our input again. Okay. So read, eval, print, loop, that is our four stage process.

It's the REPL. Okay. And another way you use the word REPL is, oh, I'm going to open the Python REPL.

(21:44 - 23:09)

So when we typed IPython a little while ago, we started the Python REPL. So this process where we talk to Python, that is a REPL. Okay.

And now what I want to do is now explore and explain a little bit of the structure of this input and output that we've done. So let's, before we do that, let's type in one more line of math. And then we're going to talk about how we can move around using NVDA and explore our input and output that we did.

So let's do, we did five plus six. Let's do another plus. So let's do two space plus space two. Two.

So I typed two space plus space two.

So let's run it. Two plus two. Out left bracket two right bracket colon four.

So you heard, what you heard was out left bracket two right bracket colon four. So we typed in two plus two and our result is four. But we always hear this OUT before we get our result.

Okay. So now what we've run so far is we had five plus six, it did in one, five plus six, and then out one, 11. So five plus six, we got the output was 11.

(23:09 - 24:21)

And then we did another input output pair. We did in two, so the second input output pair. And then we said two plus two.

And then we got out two, four. Okay. So we have two input output pairs so far.

And there's a kind of a term for this. And we can use the word cell. So a pairing of input and output, we can use the term cell.

So a cell is a combination of an input and an output in this IPython environment. And it's language commonly used in the Jupyter notebook environment. So if you have sighted colleagues who use Jupyter notebooks, and you hear them talking about cells, that's essentially what they're talking about.

It's one of these pairings of input and output. Okay. So what I want to do now is go backward and use the NVDA review functionality to travel backward in what we've done and review the inputs and outputs.

So you can hear what the structure is using NVDA. Okay. So I'm going to hold down NVDA and press up.

(24:22 - 25:37)

"Top. In left bracket, one right bracket, colon, five plus six." So we're actually at the very top.

So we're hearing in one, colon, five plus six. And now we're going to move down. So we're going to do NVDA button and then down.

And we should hear our output from five plus six. "Out left bracket, one right bracket, colon, 11." Okay.

So I'm hitting NVDA down. Okay. And I just want to say I'm using the laptop hotkeys for NVDA, and I will mostly be using the laptop hotkeys throughout this tutorial.

However, if you're using the desktop hotkeys in NVDA, which are actually the default, then what you want to do is use the numpad for navigating backward and forward. And I believe, I'll double check this, but I believe the hotkeys for that are holding down the NVDA button and typing either, I think it's seven for moving backward and nine for moving forward by line. But essentially what the numpad, the numpad, it's divided into nine keys.

(25:38 - 25:49)

And there are three keys for moving, three of the keys are for moving line by line. Three of the keys are moving from word for word. And three of the keys are for moving character by character.

(25:49 - 26:02)

And then each of those three is divided into going backward, reading what you're currently on, and moving forward. Okay. And it's actually, it's fairly intuitive when you get using it.

(26:02 - 27:15)

If it's not working and you know you're on NVDA desktop mode and you have a numpad, then remember the numpad is the numbers that are sort of on the right, if you have a large keyboard on the right, not the numbers that are on the top of your alphabetical keys, but the numbers that are on the right of your keyboard, then you make sure, I think you have to make sure, sometimes your numpad, your numlock being on and off can matter in that. So you might want to experiment with that. Okay.

So if you're using the desktop and you, or you have a numpad, you might want to experiment with your holding on the NVDA button and using the numpad to move around, to review. But I'm using the laptop, so I am, I'm holding on NVDA, pressing up and down to move line by line. Okay.

And then let's keep reviewing. So we just heard out, one, and 11. And I'm going to move down one more.

"Blank". And I heard blank. So now there's a blank line. There's a blank line separating each cell input output pairing has a blank line to separate it. So it's in out, and then you have a blank line. Then we have in out and a blank line and so on.

(27:15 - 28:42)

Okay. That's the general structure. So let's quickly go through the next cell.

"In left bracket to right bracket, colon two plus two." Okay. I'm navigating by line still, NVDA down.

"Out left bracket to right bracket, colon four blank. And there's our blank." Okay.

So there's, there should be a input output blank, input output blank. That's the structure that we're working with here. Each cell input output pairs is separated by a blank line.

Okay. And now we should be, have one more. In left bracket three, right bracket, colon.

And then that should be the end. "Blank. Blank.

Bottom." Yeah. So there's actually some blank lines and then the bottom.

But basically the end is here. In left bracket three, right bracket, colon. And it's in left bracket three, right bracket, colon.

It's waiting for us. This is the third cell. It's waiting for the input for the third cell.

Okay. It's waiting for our third piece of input before it gives us some output. Okay.

So that's the structure. Now I recommend, you know, at this point in the workshop, I give a little time for reviewing. You know, so I would say, you know, pause this video, take a few minutes to now practice your reviewing, moving around in this command line environment before things get too complicated.

And a few other useful hockey. So you will probably also want to move character by character. So you can kind of really get a sense of what the lines are.

(28:42 - 31:54)

So for example, if we go to this output line, "blank out, left bracket, two, right bracket, four." That's our output from the second cell. And we can move using the NVDA right.

We can move character by character. So that beep was capital O. So it's out, two, and then that four is the output from two plus two. Okay.

So you can move character by character. So go ahead and practice reviewing in this so you get a hang of it. And then, you know, pause the video and then come back when you're ready.

So let's do, you know, hopefully you're ready. You've practiced your reviewing. Let's go ahead and do a little more math.

Okay. So let's do, we only have learned the plus. So let's learn the other four.

And then I'll talk a little bit about, you know, we put some spaces in. I want to talk about spacing and so on in Python after that. So let's go ahead and do, let's do ten space, minus, I'm sorry, minus, dash.

And that's the same as a hyphen. And then I'm going to do ten minus three.

"left bracket, three, right bracket, colon, seven." So ten minus three is seven. And, you know, remember you're once you type your line of Python, you hit enter and you'll get some output.

Okay. So the hyphen or minus is subtraction. Now let's do multiplication.

Let's do three times four. So we're going to do three, space. And then we'll do, it's an asterisk or star, which is you have to hold down shift and press eight.

So asterisk or star. And then do a space.

And let's do four.

So three times four. And it's star or asterisk is multiplication. "Out, left bracket, four, right bracket, colon, 12."

And the answer is 12. That's our output. And then we have our current line in five.

So it's waiting for our input. "Out, left blank. In left bracket, five, right bracket, colon." In five. It's waiting for our input. So now let's finally do division.

So let's do, why don't we go ahead and do 24 divided by four.

And then let's use forward slash.

So it's the slash on the bottom right of your keyboard. Not backslash, forward slash.

24 divided by four.

"Out, left bracket, five, right bracket, colon, 6.0. And our output was 6.0." And I'll explain a little bit about why that output is a little bit different in a minute. Okay. So why do we, now we did addition, multiplication, subtraction, and addition.

(31:54 - 33:18)

And there's one thing I said a little earlier. I said, oh, when we type in, something will happen. So we typed, we've typed a few lines of code and we've, and something has obviously happened.

We get some output back. Okay. Some, some, when we run, we run code, some process happens and we get output back.

That's the obvious thing that happened. But when you ran your first line of code there, when you ran, when you write six, five plus six, something else happened. Okay.

And that is that you became a programmer. Okay. So you can't take it back now.

You ran some code, you know, so you, you, I now have the right to call yourself, you don't have to call yourself a programmer, but you have the right to call yourself a programmer. Okay. A programmer is, is someone who writes code, you know, and, and you've written some code.

So you're now a programmer. Okay. And you can't take it back.

Okay. So I kind of tricked you guys. When you've written code, now you're a programmer.

So you, you know, don't let anyone say you're not a programmer. Okay. So, you know, we've written our first few lines of code.

I want to explain just a couple of things about the ins and outs of that. So one is that we put some spaces in when we wrote, wrote this code. And it, now the question is, are the spaces necessary? And Python is very heavy on spaces and annotation and making new lines and stuff like that.

(33:20 - 34:00)

And a lot of it is designed to make things more readable for sighted people. Some of it is necessary for the, even the computer to parse what you've written. And some of it is actually just to make things a little easier to read for sighted and some low vision people.

It, it generally will work if you don't put spaces in. So let's try two plus two without spaces. I just typed in, sorry, it's a little too fast. And I didn't put any spaces in. Out left bracket six right bracket colon four. So we have four.

(34:00 - 38:49)

Okay. And it worked just as well. So why do we bother putting the spaces in? The spaces make things more readable.

Now you would basically have to decide readable for sighted people. So you basically have to decide if you, how much you care about that. So if you think you'd be working, you know, in a professional capacity with sighted people, if you think you'll be putting your code online and looking for people to contribute to it or maybe you'd be looking for a job or something like that, you want your code to look professional, then I would get in the habit of you putting in those spaces now because it will make your code look more professional.

It will, you know, it's the quote unquote correct way to code is to put in those spaces. However, if you feel like you're going to be working on your own project, you're not going to be collaborating or you're going to be collaborating mainly with other blind developers, you may not care as much about putting in spaces. So your mileage may totally vary.

I do know blind programmers who don't bother with the spaces because they kind of find them, you know, not useful or annoying. If you want the best of both worlds, you can not put the spaces in and you can run programs that are called linters after the fact, which basically will make your code like all fancy and nice, but you don't have to write it that way. It will kind of go through your code and clean everything up and put the spaces in where Python thinks there should be spaces and so on.

So that's if you really want the best of both worlds, you can look into linters. One I like is called Black, and that is a linter that will take Python code and tidy it up, make it look kind of fancy or whatever. So that spaces, they're more a style thing than something that's actually super necessary, okay? All right, so we have done our little bit of math here.

Now let's talk about what are called data types. So these are, we're going to learn five data types in this introductory Python workshop, and these data types will also, a lot of this becomes very useful when we move on in the next session to pandas and to, you know, to the data science context. These are data types that are all going to be quite critical in that data science context.

So let's go ahead and learn the five data types, and let's do that thing, like I said before, we're going to run some code, I'm going to tell you what to type, you're going to get some output, and then after we're done, I'll explain it, okay? So I'm going to clear my screen, control L, "In [7]:" we're all ready to, we're on our seventh cell, we're ready to type some new code in, and what I want you to enter is, follow along after me, and we're going to enter the word type, t-y-p-e, and then open parenthesis or a left parenthesis. Well, I might say those words interchangeably, but I want you to use left paren, type, and then I'm going to have you just enter the number five, and then a right paren. So it's type, left paren, five, right paren, okay? Type, left paren, five, right paren.

Int is the output that we got from that. So let's do a couple more. So let's do type, left parenthesis, left paren, and then let's, now let's do 5.0, right paren, space, "out left bracket, eight right bracket, colon, float."

So I put a space on the end there, but it wasn't necessary, so, but what we got back was float. "Blank, out left bracket, eight right bracket, colon, float." There we go.

"Blank, in left bracket, nine right bracket, colon." So we got float, which is short for floating point number. We'll explain what that is in a minute, but let's do our other ones.

So we did int, five was int, 5.0 was float. Now let's do type, t-y-p-e, left parenthesis, double quotes, hello, double quote, right paren. Okay, so it's type, left parenthesis, double quotes, h-e-l-l-o, hello, double quote, right parenthesis.

(38:50 - 43:56)

Okay, and let's hit enter. "Out left bracket, nine right bracket, colon, s-t-r, in left bracket, ten right bracket, colon." We got out, our output was s-t-r, short for string.

Okay, we're going to do two more, so that's three. We did integer, float, string. We're going to do two more.

Let's do type, t-y-p-e, left paren, left parenthesis, and let's do capital T, true, and that beep was me putting in the capital. "True, right paren." And there's a right paren, enter.

"Out left bracket, ten right bracket, colon, bool, in left bracket." Boolean or bool was what we got for that. So we did type, left paren, capital T, true, right paren, and that result was boolean.

Okay, and then let's do one more, and then I'm going to explain what all of these are. Okay, so let's do last one, type. This is the most complicated one, so I'll go slow.

"Type, left parenthesis, and then let's do right square bracket, or sorry, left square bracket, left bracket. Okay, and that is, it's a row down from your number row on your keyboard, number row down, and it's over toward the right. Okay, so from my keyboard, it's the third from the right on the second row from the top.

It's near the backspace key, and then let's type in one, comma, two, comma, three, and then I'm going to do right square bracket, right bracket, right parenthesis. So that's type, left square bracket, I'm sorry, start again, type, left parenthesis, left square bracket, one, comma, two, comma, three, right square bracket, and then right parenthesis. Let's run that.

And that's a list, so . Our type there, or the object we gave it, we'll talk about objects in a minute, but was a left square bracket, and then a bunch of other objects, so in this case, one, two, and three, separated by commas. Okay, so a list, that's how a list, you create a list in Python.

So what are all these data types? Let's go over them, and we won't really be using any other data types in this, with one exception, which we'll get to at the end. So let's talk about integers and floats. So integers and floats, that's what we started with when we did our math, we used integers, and integers are numbers without decimals, they're whole numbers.

Floats are floating point numbers, they're numbers with decimals. And I guess an obvious question here is, why does Python keep track of these differently, or why can't we just have decimals on all our numbers? The short answer is that decimal points sometimes make things a little bit tricky. Numbers with decimals can be computationally intensive, and then you have to make some decisions about where to cut off, like if you do certain kinds of division and stuff, where do you cut off decimals, say if you have three, three, three, three, three, if you do, you know, certain kinds of division, and then maybe you don't know exactly where to cut off.

So if you do, say, for example, 10 divided by three, three, "out left bracket, 12 right bracket, colon 3.3333333333333335." So you have to make, that could have been different, we could have, they could have decided to add a couple extra threes, and so on. So because decimals are always sort of, by definition, sort of imprecise, and because they can be kind of computationally intensive, programmers often keep track of them separately. However, you often won't have to think, unless you're working with extremely large numbers, or unless precision matters a lot, you won't have to think about this too much, okay? Because Python is pretty sensible in how it handles math, but what you just need to know is that there are two kinds of numbers that you're going to encounter frequently.

There's actually other kinds of number data types, but we won't talk about those in this workshop. But you just need to know that there are different number data types, and that the behavior is a little bit different between them, okay? So integers and floats are both numbers. Then we had one which is a little bit close to my heart, because I have a humanities background, I have a PhD in English, and you know, I'm very into, you know, books and text, and this data type is basically text, and it's called a string.

(43:57 - 48:57)

So a string is a sequence of characters, okay? And people often say an arbitrary sequence of characters, arbitrary from the computer's perspective, because the computer generally doesn't care what is in a string. It's humans who care what's in the string, but it's arbitrary from the computer's perspective, and so it can really be any kind of text characters. It can be text in the form of alphanumeric characters, so it could be letters, numbers, or also special symbols, okay? Or also white space.

These are all valid things to have in a string, okay? And a string could be, you know, a whole novel can be contained in a string. So strings are very humanities data type. And then let's talk about Boolean.

So that's a much more philosophical data type. So it's either true or false. So Booleans are, there's only two Booleans.

So there's capital T, true, and capital F, false. And true represents truthiness, or statements that are true, and false represents falsiness, statements that are false, okay? And we'll get into this a little bit more later in the workshop, but, you know, for now, you know, true represents true, false represents false. We'll get a little more into why that's useful later.

And the last one, which is going to be very important in this workshop series, is the list, because it's going to be how we store a lot of our data. We're going to store our data in lists like objects, not necessarily lists, but things similar to lists. And the list is a sequence of other data types.

So in a list, you can have numbers, you can have strings, you can have, you know, in numbers in the form of integers and floats, you can have Booleans, or you can even have other lists. And in fact, that is a frequent technique is to have a list full of lists, which sounds confusing, but it's actually quite useful. So those are our five data types.

And we'll be working with them all more, and we're going to learn more about each of them as we move on in this workshop. So and we're going to learn one more data type at the end, but it's kind of a special case. So okay, so those are our data types.

So the other thing I want to explain here is what did we do when we did type, we typed the word type, or we entered the word type, and then we did a left paren, and then we put something in it, in the parenthesis, and then we close the parenthesis. So we did type, left paren, say five, and then a right paren. And then what we got back was integer.

And what were we doing with when we did that whole type thing? What we're doing there is using a function. And a function is a really critical part of, of using Python. And there's three ways to think about functions, or I'll give you three definitions of functions.

And you for now just pick the one that makes the most sense to you and stick with that. And I'll tell them they're kind of each a little more accurate, maybe, or a little more complete part of the puzzle. So the first way of thinking about a function is that it's a way of doing something in Python.

You can kind of think about it as a verb. And, you know, so we did type, and then we gave it a five, and then we said, and then it's, it came back with integer. So type is a way of saying, hey, tell me the type of this object.

Okay, every, all of these things that we're doing using in Python are objects. So it's a very general word that I'm using. We'll talk about objects more later as well.

And so it's a verb. And then inside the parenthesis is sort of the thing that the verb acts on. So you can think about it if you're, you know, into grammar, it's the direct object or whatever.

But if you're not into grammar, don't worry about it. But it's the thing the verb acts on. Okay, that's one way of thinking about functions.

Another way of thinking about functions is, it's a way of, that you can save code to run later. Okay, so you give some, a name to some code, and they can run it again later. Okay, and another word for that type of function or thinking about functions that way is a, it's a routine, or a subroutine.

Those are kind of old fashioned words, because they're not used as often anymore. They're, because they apply more readily to programming languages that were programming languages that were a lot more limited than Python, or modern programming languages, but it's still accurate. So a function is a routine.

So it means you can store some code and use it later. And then the last way, which is maybe the most accurate way of thinking about a function is that it is a, it is an, it takes, it's something that takes an input, it runs a process, and then it gives you an output. Okay, so it takes an input, it runs a process, and it gives an output.

(48:57 - 50:57)

And I often think about it in my mind, I think about it as a box. Okay, so a function is a box with a hole in the top and a hole in the bottom. So you put things in the hole in the top, and then you turn the box on, it's a machine, you know, machine type box, you turn it on, you press the on button, and it goes beep, beep, boop, boop.

And then something comes out the bottom. Okay, that's one way of thinking about a function. So for example, one function could be like, if you imagine in real life, a pinkifier.

So imagine you're a big machine, say 10, you know, a couple feet tall, and there's a hole in the top and a slot in the bottom. And you can put things in the top. And you say I take my favorite mug, which is green, my favorite color is green, and I put it in the pinkifier.

And then I turn it on, it goes beep, beep, boop, boop. And then out the bottom comes my mug, but now it's pink. Okay, the pinkifier.

Okay, so that's the kind of things that functions do. It takes something, often it transforms it, but an accurate way to think about it is it does something, because it doesn't always transform it. And then it returns something back output, as an output.

Sometimes the thing it returns is it says nothing, but that's also an output in Python. So those are the three ways of thinking about functions. Okay, and we'll be learning a few more functions as we go on in this workshop, at least, you know, three, four, five more functions in this workshop.

All right, so, you know, if this is, if you're following along with me at home, this might be a good time to, you know, let your brain cool, go get a cup of coffee, go get a tea. I'm going to be pushing on here. But our next topic is we're going to talk about variables.

And variables, you know, since we're moving on to another topic, I'm going to clear the screen. So I'm going to hold down control and press L. Okay, we're going to have a nice clear screen. It's waiting for our input.

(50:57 - 51:12)

And we're going to do, we're going to talk about variables. So let's do the thing where I tell you something, and then we do it, and then I'll explain it afterward. So I'm going to type, I'm going to tell, I'm going to type the word greeting.

(51:14 - 51:33)

"g-r-e-e-t-i-n-g, greeting". I did a space, sorry. I did a space.

And then I pressed equals, And then I do a space. And then I'm going to do a quote, double quote, and then I'm going to type, hello.

(51:38 - 55:14)

So I did greeting equals double quote, hello, double quote. Okay, greeting equals double quote, hello, double quote. And I'm pressing enter.

Now notice something funny there. You notice something missing? We didn't get a line with output. I just got in again.

So "blank, blank, blank, blank, blank, blank, blank, in left bracket 14, right bracket blank, in left bracket 13, right bracket colon, greeting equals quote, hello quote." Okay, and then a blank line. And then the next one is input again. "In left bracket 14, right bracket colon." There's no output. And the way to think about this is, what we did was, we typed greeting equals quote, hello quote.

And basically what that did is, we take a piece of data, which is our string, hello, and we assign it to a variable. So basically, the best way to think about it is, you give it a name. So we take that piece of data and we gave it a name, which in our case is greeting.

So hello gets the name greeting. And we get no output because instead the iPython likes to return sort of the last piece of data that we give it. And in this case, the data sort of goes, instead of coming back to us in the REPL and us getting the output, the data is going into the variable.

You can think about it as being stored in the variable. So when you save a variable, or when you assign a variable, that's the correct technical or programming term in Python, when you assign the variable, you won't get output. And you can think about it as that data going into the variable or being stored in the variable and not coming back to us.

So let's do one, assign more and more variables. So let's call it say parting. Space. Equals. Space. Double quote. And then goodbye. And then a double quote. So it's parting. Space.

Equals. Space. Double quote.

Goodbye. Double quote. And I'm going to press enter.

Again, no output. We just heard our in waiting for a line again. So there's no output on these cells that assign a variable.

So now we have two variables, greeting and parting. So let's try putting greeting on a line by itself and running that. "G-R-E."

Now I started typing this and suddenly I'm like, man, I typed G-R-E and it's so much work to type the rest. It's just really tiring me out. And something I always say is programmers are lazy.

Programmers don't like to do things more than once. So I already typed in greeting once and I really don't want to do it again. I want to make the computer do the work.

So programmers, whenever possible, hate to repeat themselves or ourselves, because like I said before, you're now a programmer. And they hate to repeat themselves. You hate to repeat yourself.

(55:15 - 1:00:27)

And they also are really lazy. Okay. And they like to make the computer do the work.

So I typed G-R-E.

And I want the computer to fill in the rest. So go ahead and hit the tab button. "Edding."

And you heard it say edding. And that was it. It said we already had G-R-E and the edding was it filling in the rest of greeting.

So it's greeting now. So if I press space, you'll hear. "Space."

Oh, you should have heard greeting, but it didn't do it. But we have greeting on this line. And I'm going to hit enter. "Out left bracket 15 right bracket colon.

Hello." Okay. "In left bracket 16 right bracket colon."

So I typed in greeting and I let the computer fill in the rest, because I'm lazy. And when greeting was on a line by itself, the input line, I pressed enter. And then what I got back was quote hello quote.

And in fact it blank "in left bracket 13 right bracket colon. Greeting equals quote hello quote." Yeah.

Quote hello quote. "Blank in left bracket 14 right bracket colon equals quote goodbye quote. Blank in left bracket out left bracket 15 blank in left blank bottom." Sorry, I'm going all the way to the bottom again. And so we have our output, which is hello.

So when you type a variable by itself, you put it in a line by itself, what you get back is what the fancy programmer term is a representation of that variable. But basically what you're getting back is it tells you what that variable, what data is in that variable or what data is assigned to that variable. So we typed in greeting and we got hello, because that's the data we say we, you know, quote unquote saved, we assigned to the variable.

Okay. And let's do parting. "P-A-R-T."

And then I'm going to press tab to finish the rest. P-A-R-T in. Parting.

You know, get used to using that tab button, because it comes in very useful and it'll prevent you from making typos. Because if you let the computer do the work, you know, we as humans tend to do typos because, you know, we have fingers and all this complicated stuff that we have going on. The computer tends to get things like this a little more right where it just fills things in.

So go ahead and use that tab a lot. Okay. "Out left bracket 16 right bracket colon goodbye."

Goodbye. So parting is assigned to or the goodbye string is assigned to parting. Okay.

All right. So let's also do something. Let's do greeting space parting.

Sorry, greeting space plus space parting. So it's greeting plus parting. "G-R-E-E-T-I-N-G space plus space P-A-R-T-I-N-G."

So you're hearing only a part of it because I'm filling it in with tab. Greeting plus parting. Out left bracket 17 right bracket colon hello goodbye.

Hello goodbye. So when you add strings together, it actually combines them together, which is cool. The fancy word is concatenates them.

It combines them together. So greeting plus parting. And then the output was hello goodbye.

And there's no space between them. So if you wanted a space, you could, you know, add one to the end of hello or the beginning of goodbye. But we didn't do that.

So so variables. And then, of course, they don't need to be full words. So you can totally say X X equals five "in left bracket 19 right bracket colon."

And there's no output. Right. Remember when we assign a variable, there's no output. And then we could say Y equals 10 "in left bracket 20 right bracket colon." And then we can do X plus Y"out left bracket 20 right bracket colon 15." So the output was 15.

And or we could do X times Y "X star space Y out left bracket 21 right bracket colon 50". So, you know, five five times 10 is 50. You know, X times Y five times 10 is 50. Or you could do Y times Y. So be Y. Remember, Y is 10. So we could do Y times Y "out left bracket 22 right bracket colon 100. So 100."

I cut it off there a little early, but the answer with the output was 100. So variables can be long or short. What you the only rules with variables is you can't you have to start with a letter.

You can't start with a number or space or something. And well, you can't use spaces at all. And there's only a few special symbols you're allowed to use.

You're allowed to use, for example, underscores. But you so variables in general should be lowercase. And if you have more than one word, you should use underscores.

(1:00:27 - 1:00:53)

There are exceptions to that, but we're not going to get into them in this workshop. Now, there's certain conditions where you use all uppercase or starting with an uppercase or some other types of variable names. But we won't get into them in this workshop.

You can look those up if you if you're interested. But you shouldn't start your variables with numbers. And I would also say, in general, you should make your variables as descriptive as possible.

(1:00:54 - 1:05:00)

So we will be using some short variable names in the next workshop. They're sort of also a little bit exceptional. But broadly speaking, you should make your variables descriptive and easy to understand rather than making them all x or z or whatever.

Okay, that's fine in some very specific circumstances, but mostly try to err on the side of making them more understandable. Okay. All right, so we've done variables.

So we did variables. And now we're going to do the we're going to talk a little bit about first, I'm going to I'm answer a common question that I get at this point, which is about double quotes, and we have to use double quotes. And then we'll get a little into errors and how we deal with errors in Python.

So, so okay, so first of all, double quotes. So in this workshop, I'm almost exclusively going to use double quotes, because it's less confusing. However, you can totally use single quotes to when you're creating strings.

Okay. So for example, you can do we did greeting, parting. I'll make one called exclamation, "e, x, c, l, a, m, a, t, i, o, n, exclamation equals equals space quote, quote, tick."

So you heard the word you heard the character tick there, I used a single quote, tick. And then I'll say yee haw, "y, e, e, a, a, w." So it's "yee haw, tick." I put another tick.

So that is exclamation, space equals space, tick, which is a single quote, yee haw, tick. Okay, and this will work just as well "in left bracket 24, right bracket colon." Okay, and so we didn't get any output there.

But now we can do e, x, c, I typed in e, x, c, I'm going to let it fill in the rest, lamation, lamation, exclamation. Let's press enter. "Out left bracket 24, right bracket colon, yee haw."

There's yee haw. And, and that worked fine. But what you can do is mix double quotes and single quotes.

Okay. So stick to one or the other, you can have use single quotes, you can use double quotes, but you can't use both. Okay.

One thing. So now, I'm going to have you go ahead and mix and match a single and a double quote, because I want to get an error. And then I want to talk about how you can deal with errors in Python.

Okay. So let's go ahead. And this is going to be our first of two main kinds of errors.

Okay. So we're going to learn about two kinds of errors that are very different. And, and we'll get an example of each.

So this is going to be the first kind of error. I'll explain it. But let's go ahead and write a line of code that will give us an error.

So let's type in, we did exclamation. Let's just go ahead and do exclamation again. "E, x, c, l, a, m, a, t, i, o, n. Exclamation."

Exclamation. Equals. Space. And I'm going to do double quote.

Yeehaw. And then let's do single quote. "Yeehaw.

Tick." There's the tick. Okay.

So exclamation equals double quote yeehaw tick. "File quote lesson Python dash input dash 25 dash c 0 1 d 42 39 0 5 6 greater quote line 1 exclamation equals quote yeehaw carrot syntax error colon eol while scanning strin g literal in left bracket 26 right bracket colon." Okay.

So that's a lot of output. Not as bad as some other programming languages, but basically what this is, it's our error. And let's go up.

(1:05:00 - 1:05:38)

And the last thing that gets printed out is kind of usually the most descriptive. And the rest of it is trying to tell you where the error happened. So let's review and try to find our description of the error.

Bottom. So we're at the bottom. "Bracket 26 right bracket colon."

Let's move up. "Blank blank g literal syntax error colon eol while scanning string literal." So and it's cut off there because I made the text so big that it cuts off after a pretty short line.

(1:05:39 - 1:05:58)

If you didn't make the text huge like I did, you shouldn't have this problem the same way. But it said "syntax error colon eol while scanning string literal." So it's syntax error eol, which is short for end of line while scanning string literal, which sounds complicated.

(1:05:58 - 1:07:04)

But the most important part here is syntax error. And the syntax error, it's one of the main types of errors in Python. It's one of the two big categories of error.

And basically it means a syntax error happens even before your code gets run. So you type in your code and then there's a process that happens before it gets run, which is a checker. And it checks that your code is valid Python, that it's correct Python.

And if it breaks a fundamental rule of Python, then it gives you a syntax error. So and basically it's saying, hey, the code they wrote, it's not even really Python because it broke a rule. So even before your code gets run, if there's okay, I had someone calling me very loudly there had to pause for a second.

But so the syntax error, it happens before your code gets run. And if it doesn't pass that check, you get a syntax error. And typically the syntax error is with some piece of grammar.

(1:07:05 - 1:15:09)

So it's often with quotes, for example, this quote mismatch with maybe you left off a parenthesis, that's a very common one. Something is messed up with the spacing, or something that's wrong with say like a comma, a square bracket, something like that. It's most common source of a syntax error.

So those are what you should check first in your line, when you get a syntax error. Now, the stuff before this, where it says syntax error is trying to tell us where the syntax error originates. And so let's go back and review that really quick.

But unfortunately, it's not, a lot of Python is actually pretty accessible. This part is a little bit annoying, especially the syntax error, because it tries to point out where our error is. And it does it in a somewhat visual way, which is annoying.

But let's review backward "syntax error colon caret." So okay, you heard caret. And that's where the problematic thing is that caret is trying to tell us in the line above where the syntax error happened.

So if we move up one line, "exclamation equals quote yeehaw." Yeehaw. So there's the caret is on the line below, and it's trying to point up to where the error happened.

But the only way to get that as a blind person is so if you go to that caret line, and then you have to count over, so you have to go "space, caret." That was 26 characters over, I believe. Okay.

So and then you could go up to the line above and count 26 over, which is pretty annoying. But let's just do it just for to show you the technique. "Exclamation equals quote yeehaw."

Okay. So I'm going to quickly try to type in NVDA write, write 26 times to try to move all the way over. And that's, it was, I moved one a little too far there, but it was under, it's under the, it's trying to tell us in this annoying somewhat sighted way, that by putting a caret underneath it visually, that, that the tick is the problem, that the, the quotes are mismatched.

So that, so this is definitely something you can do. You can count where the caret is by moving manually and then going up. But really, when you get a little bit more used to using Python, you often really won't need to do this only in kind of like a bad situation when you have to resort to something like this, because when you become more experienced in Python, the syntax errors, they often become a little more obvious to you when you review the line.

Okay. Because there's only a limited number of ways to trigger a syntax error. So, and especially as, if you're a beginning programmer, you'll encounter syntax errors a lot.

As you become more experienced, they will become less frequent, and also you'll become pretty handy at identifying them. And you won't necessarily need to do all this counting. But it's good to know that that's a technique you can do, which is to count where the caret is, how many characters the caret is, and then go up one line, count the same number of characters over, and that will be where it's trying to tell you where the syntax error is.

Another useful thing is, and I'm going to teach you this, this is mainly useful for us screen reader users. I haven't seen that many sighted people do this, but IPython provides a special command to make the output of errors a little more minimal. So I'm going to teach you how to do that now, and then we'll switch it back to the way it is now.

Okay. So let's go ahead and do that together. So it's going to be percent, the percent symbol, which is above the five on your keyboard, and then we type xmode, x-m-o-d-e.

And then that's percent xmode, and then we type minimal. I typed and deleted the space just so you'd hear the word. Okay.

So it's percent xmod minimal. And it said it changed the "exception reporting mode to minimal." So if we run our line again to get the exception, and this is, I'm going to teach you one more useful thing, is that you can press, just press the up key by itself, no NVDA or anything, just up key by itself, and you can go back through lines that you entered before.

So let's enter the line. We got the error again. That's, it's doing, it's not actually doing the minimal thing, so let's try one more time.

Minimal and then, and then let's do quote. Okay. So actually I misapprehended there.

This xmode only works on this second time type of error that I'm going to teach you now. It doesn't work on syntax errors, so that's actually something I'm learning. So let's go ahead and set the xmode back to verbose.

So I typed percent xmod verbose, and then I will, we'll use it again when I show you the second kind of error. Okay, so okay, so we've learned a little about syntax errors. I want to show you the second kind of error, which is the error, eventually it's the error that you're going to see the most often, which is kind of why I was even not even thinking about it with xmode, because this is the error that I tend to see the most often, which is, it's called a traceback error.

Basically what happens with the traceback error is Python, you know, it passes the syntax check, and your Python, your code runs, but then sometime during your code running some kind of logical inconsistency or problem occurs. Okay, so let's go ahead and try to create a one of these traceback errors. So let's do one which is to divide by zero.

So that's a we could try to do 10 divided by zero. And then forward slash. So I did 10 space forward slash space zero.

10 divided by zero. So what you heard there was it said what the error was at the beginning, the type of error, division by zero error. Then what it tried to do was tell us where the error happened.

(1:15:10 - 1:19:50)

Now it's pretty clear where the error happened because we only typed in one line, but often you're running Python code that's more than one line, so it tries to explain where the error happened. So it said, okay, you know, it's this 10 divided by zero. And then at the end, it says the full version of the error, which is.

I'm reviewing down. Zero division error colon, division by zero. That's the name of the error.

So the structure here is the beginning part is mostly about telling you where the error happened. And then at the very end, you hear what type of error it is, which is actually usually the most useful information. And this is where the X mode that I just wanted to show you before that I didn't work with syntax error comes in.

So if we turn it on now, again, "percent sign X mode, percent, X, M, O, D, E, space, mode, space, minimal, M, I, N, I, M, A, L, minimal, space." I typed in minimal just so you'd hear the word. So percent, X mode, space, minimal, "exception reporting mode, colon, minimal."

So it turned on the minimal exception reporting. And then now let's run our line that got us an error again, the division by zero, 10 divided by zero. One, zero, 10, slash, space, zero.

10, space, slash, space, zero. 10 divided by zero. "Zero division error colon, division by zero." Okay. So when minimal is on, we just hear the error, which is actually really nice. Okay.

So you can play around with that a little bit. And often I'll turn it on and off pretty frequently. And then let's go ahead and turn it back to verbose.

Okay. So we could do "percent, X, M, O, D, E, mode, V, E, R, V, O, S, E, verbose, exception reporting mode, colon, verbose." So, okay.

We turned it back to what it is originally, which is verbose. Okay. So that's a useful technique for screen reader users because often we don't want to hear all of that output about where the error was.

We just want to hear what the error is. Okay. And then, you know, you can often, you can switch things.

If you want more information, you can switch it back to verbose. Okay. So it's percent X mode.

And what that percent symbol is, it's a special, it's not regular Python. So it might not work in a, if you're writing Python in another program, but it's a special command related to IPython, you know, this special REPL that comes with Anaconda. Okay.

And it's called a magic. It's a IPython magic command. Okay.

That gives us a little bit of control over the environment that we're programming in right here. All right. So that is a little bit on error.

So there's two kinds of errors. The first one happens before you run the, before the program runs. And it's basically, oh, something didn't pass muster in terms of like the rules about what makes a valid Python.

And what you want to do in those situations is check for error problems with the syntax. So things like quotes, parentheses, square brackets, and so on. Okay.

Then the other kind of error is that your program is running, but it gets into some kind of state where there's a logical inconsistency. And in those, you want to often, what you want to look at is what is the error that it gave us. So that last thing in the error output.

And in our case, it said division by zero. And that's a pretty straightforward error. That means we divided by zero.

That's not allowed. It's a logical inconsistency. And a lot of errors are like that.

A very common error you might get is a name error. So if you type in a variable that isn't assigned, you haven't created yet. Like if I just type in my name, Patrick, without, and we haven't assigned anything to it.

And press enter. I'm just going to use reviewing to get to the error here. Name error.

Name Patrick is not defined. And that's straightforward enough. It's a name error, and the variable Patrick doesn't exist.

We tried to access a variable. It isn't assigned. So those are the two kinds of errors.

They're syntax errors and traceback errors. And as you get more advanced in programming, you're going to see the traceback error type more often. So those are errors.

And now what we're going to do is we're going to clear our screen. We're starting a new section. And then we're all ready to start a new section here.

(1:19:50 - 1:21:01)

What we're going to do is we're going to learn a little more about lists. So lists are going to be really critical for what we do in the next section, in the section on pandas. So let's learn to work with them a little bit more.

And let's create a list that we'll be working with in this section. And so we're going to write a variable name. So it's going to be flowers.

And let's do a space. Equals. And now let's start our list.

So it's going to be left square bracket, double quote. And I'm going to go slow here because it can get confusing. And let's say rose, like the flower rose.

Double quote. And then let's do a comma. Space.

Now let's do another one. Double quote. That's a violet.

(1:21:01 - 1:27:15)

Violet. Comma. Space.

Double quote. Buttercup. B-U-T-T-E-R-C-U-P.

Buttercup. And then a double quote. And then now we have our rose, violet, buttercup.

Let's do a right parenthesis. And now what we did was we wrote the word flowers.

That's our variable name. Flowers. And then we did equals.

And then we do left parenthesis. I mean, sorry. Left square bracket.

Then double quote. Then the word rose. Then a double quote.

Then a comma. Then a double quote. Then the word violet.

V-I-O-L-E-T. Then a double quote. Then a comma.

Then we did a double quote. We did the word buttercup. B-U-T-T-E-R-C-U-P.

Then we did a double quote. Then we do a right bracket. And so another way of thinking about this is there's three strings.

Rose, violet, buttercup. You know, they're words surrounded by double quotes. And then they're separated by commas.

So it's rose, comma, violet, comma, buttercup. And then we surround that with square brackets. So left square bracket, right square bracket.

And then we assign that to flowers. So it's flowers equals all of that list. That's another way of thinking about it.

So let's go ahead and press enter. And remember, when we save a variable, we don't actually get any output. So it was flowers equals left square bracket, rose, violet, buttercup.

Each is a string. And then right square bracket. So we should have our variable.

Let's type flowers by itself. So I'm going to type flow. And then let it finish it by pressing tab.

It's pronouncing buttercup a little because the line is cut off. So there's a hard return in the middle of buttercup because I've made the text so big. That's why it's pronouncing buttercup a little bit funny.

But when we type in flowers, we get to hear our list. Our rose, violet, buttercup. So let's try to do a few things with the list.

So if we type in flowers, the variable name. And now no space. Don't make a space.

We want a left square bracket. And then type the number zero. So it's flowers, left bracket, zero, right bracket.

And no spaces. So our output was rose. So we did flowers, left bracket, zero, right bracket.

And our output was rose. And so what we're doing here is a technique in Python called slicing. And what slicing does is it allows you to pull out parts of a list.

So if you want the first item in a list, and this is a little confusing in programming, then you start counting from zero. So it's flowers, left bracket, zero, right bracket. That's the first item in our flowers list.

So in programming, counting always starts from zero. So in this case, the first item in the list is rose. So it's flowers.

The zeroth item is rose. So our output is rose. Now what if we want the second item? Well, if the first item is zero, the second item is one.

I know that's confusing. Programmers are weird. They start counting from zero.

They say things like the zeroth item. But you do get used to it a little bit. So let's do flowers, and then a left bracket for our slicing syntax.

And then we do type the number one, and then we do a right bracket. So it's flowers, "F-L-O-W," and I'll do tab, "E-R-S, right bracket, I meant left bracket, one, right bracket." So it's flowers, left bracket, one, right bracket.

"Out, left bracket, 39, right bracket, colon, violet." And then we had violet. So that's the second item in the list, or the first if we want to do it the way programmers.

The zeroth is rose. The first is violet. The second is buttercup.

Or as a normal person would say it, the first is rose, second violet, third buttercup. But in programming, counting starts from zero. And you can also put in a range.

So let's try that now. So let's do flowers, F-L-O-W-E-R-S, left bracket. And let's do zero, zero, colon, two, two, right bracket. So we did flowers, left bracket, zero, colon, and then a two, and then a right bracket. And let's see how many we get. "Out, left bracket, 40, right bracket, colon.

Out, left bracket, 40, right bracket, colon, left bracket, rose, violet, right bracket." So it's rose and violet. And this does get a little confusing.

So we asked for this from the zeroth to the second. And it's a little confusing, but basically the second number has to be one higher than you in some case you think it should be, which is a little hard to get used to. But basically, we wanted the first two items.

So I flowers, left bracket, zero, colon, two, right bracket. And that sliced the first two items in the list. So using colon, you put two numbers, the first number, then a colon, then a second number.

(1:27:16 - 1:28:25)

And then it gives you the range of those two numbers. But you do have to make the second number one higher than maybe you think it should be, which is also confusing. And this is, believe me, even very experienced programmers get confused by the counting from zero and things like the second numbers being higher.

There are good reasons why things are the way they are. But suffice to say, it is confusing and it's unintuitive. And so there's a very common type of error in programming called an off by one error, maybe the most common, one of the most common errors.

And it basically just means you tried to pull out some data and you're off by one. You missed one. So this is something to keep an eye out for.

It's very common to mess that little part up. But you can always go back and fix things just as long as you look out for that type of error. So we learned how to slice from a list.

And I want to show you one other thing that's going to come in useful when we start working with pandas. And it's going to be our second function that we learned. We only learned one function so far that's type.

We're going to learn another function. It's going to be called len. So let's go ahead and use it.

(1:28:29 - 1:28:46)

So we're typing len, left parenthesis. And now let's give it our flowers variable. So it's len, l-e-n, left parenthesis, flowers, right parenthesis.

(1:28:53 - 1:29:40)

So the output was three. So what len does, it's short for a length. And we get the length of a list or whatever else we pass into that function.

And that's the programmer term when you kind of give something to a function by putting it in the parenthesis. We say we passed it into the function. And then there's also another fancy programmer term, which I'm sorry, I apologize for using these terms.

But it is useful to learn some of the vocabulary because then it helps you to look things up or to sound smart when you're talking to people. That's one of the main reasons. But the word, when you pass something into a function, when you pass an object into a function, like for example, we gave len our flowers variable.

(1:29:41 - 1:31:12)

When you pass that object into the function, we call that object an argument. So the thing you give to a function is an argument. So that's a fancy programmer word for something you pass into a function.

So our flowers is the argument to len. And then we get back three, because there's three items in our list. And that does work on other types of objects.

So the only other type of object that we've learned so far or type data type that we've learned so far that len will work on is a string. So we can do len and pass it our greeting variable that we defined earlier. And then the argument will be greeting.

So we did len, left parenthesis, and let's give it greeting. So it's len, left paren, greeting, right paren. It didn't, blank, let's review to get it, "out left bracket 42 right bracket colon five."

Five is the output. For some reason it didn't, it skipped over our output there. Five is the output because, and what it does with strings is it counts how many characters are in the string.

So greeting is hello, h-e-l-l-o, that's five characters. So it told us how many characters is in the string, which can be pretty useful in some contexts. It tells you how long the string is.

(1:31:14 - 1:32:16)

There's one other thing I want to teach you about lists, but we're going to learn it in a minute. I want to kind of come back around and we're going to learn about, we said we talk about objects. So let's go ahead, we've learned the basics of lists.

We'll do a little more with them in a minute, but I want to teach you one little thing. We'll take a little break from lists, come right back to it. And then, so we're going to learn this next thing, which is about true and false and called conditionals.

Then we're going to learn a little more about objects and how to look inside them and finish off learning about lists. And then we're going to take 10 minutes and make a little application just to pull together all the things that we've learned. And then we'll be, that'll be the end of, more or less the end of our lesson.

And that last bit, when we make the application, I'm going to teach you the most important, or maybe the most powerful, let's say, little bit of functionality in Python. And kind of the coolest in a way, just to give you a little roadmap of what's coming up. So let's clear our screen.

(1:32:19 - 1:35:06)

We're up to our 40, 43rd cell. So we've have entered 42 lines of Python so far. So very auspicious.

So let's talk just very briefly about true and false and how those are useful, because we're going to be using those a bunch in the pandas workshop. So the two variables that we're going to be using here are true and false. But we're going to be using them not so much to type them in directly, but we'll be seeing them as output from certain kinds of lines of Python that we're writing in, certain kinds of Python statements that we're writing in.

So let's do the thing where we try it, and then I'll explain it. So let's try this. 10 is greater than 5. 10 is greater than 5. I use the greater than symbol. That's near the bottom right of your keyboard. So I said 10 greater than 5, and the output was true. Okay, now let's try this.

11 greater than 40. 11 is greater than 40. False.

Okay, so I wrote 11 is greater than 40. False. Okay, let's try a less than.

Let's do 0 is less than 3. Now think about that. Is 0 less than 3? Let's get our output. True.

So 0 is less than 3. Okay, so let's do one last one. Let's do 10. Equals equals.

Equals equals. That's two equal signs. Space.

10. So 10 equals equals 10. "Out left bracket 46 right bracket colon.

True. In left bracket 47 right bracket colon." So 10 is equal to 10.

So we got true, and let's do one last one. "1 0 10, space equals equals space 1 1 11. Space."

I did a space just so you can hear the 11. Out left bracket 47 right bracket colon. False.

False. 10 is not equal to 11. Okay, so what are we using here? These are statements.

(1:35:06 - 1:36:46)

So a statement is a word, and it's a like a little phrase in Python, or a line of Python is a statement. Okay, it's actually smaller than a line. You can have multiple statements in a line, but a state Python statement is a little piece of Python.

Okay, so that Python can evaluate, so or simplify. So what we've written here are a couple of little Python statements. So they are, you know, for example 10 is greater than 11.

False. So Python is evaluating those. Remember it's doing that process that Python does of simplification.

It's evaluating them, and then it says, is this true or is this false? And if it's true, it returns the Boolean data type true. If it's false, it returns the Boolean data type false. Okay, and then finally also the one that's probably the most used is equals equals.

That checks if something is equal to something else. So 10 equals equals 10. Is 10 equal to 10? And the result of that you could probably get is true.

10 is equal to 10. Or if you do 10 equals equals 11, then you get false because 10 is not equal to 11. And why is it equals equals and not just equals? Why can't it just be 10

equals 10? The answer is because we already used equals for something.

We used it for assigning variables. Remember we said greeting equals hello, or we did flowers equals blah blah blah, rose, violet, buttercup. That is assignment, okay? It assigns a variable, so it gives something a name.

(1:36:47 - 1:40:33)

This is not giving a name, it's checking if something is equal to something else, okay? So the reason we use two equals for this is because the one equals is already taken and it does something else. These are called conditionals, and basically they are a way of checking for truth and false. Truth or falsehood in Python.

In some kinds of programming, you can build up, hold elaborate programs using true and false. So you can check if a whole bunch of things are true or false, and then go through and be like, okay, is this true? Then do this. If this is true, then do this.

If this is true, then do this. We're not going to go into that in this workshop, mainly because that's more something you would do if you're developing applications. It definitely is something you could do in advanced data science, but we're not really getting to that point in this workshop series.

We will, however, be using these equality or comparison, these conditional statements, these true and false type statements, in our data science to check when we work with our data sets, okay? So we will be using this extensively. But if you're like, well, why is this useful? Well, you'll see in the next workshop, and it's also very useful when you're building applications and you need the application to do different things based on different kinds of behavior, okay? So those are conditionals. We're not going to use them anymore in this workshop, but we will be using them pretty soon after starting Pandas in the next workshop, okay? Now, we're going to now talk about objects and how to look inside them.

Once we're finished with objects and working with them, we're going to make a little application, pull everything together, and that application is going to motivate you for the next workshop series, okay? So we're kind of coming up close to the end here. Let's talk about objects. So we're going to clear the screen.

This is a great time to talk about objects. So far in Python, everything we've used, all the integers, floats, booleans, lists, and strings, those have all been objects. And actually, even our functions that we've used, type and so on, those are objects.

So everything in Python is an object. And an object is really, it's a container, okay? It's a container, and it has a bunch of stuff inside of it. Now, I'm going to tell you about the two types of things that can be inside objects very briefly, and then we're going to use some stuff.

We're going to reach into an object and use some stuff that's in it, and maybe that'll, and then I'll do a few more complicated explanations, but okay. An object, you can think of it as a box that contains other data and functions, okay? So it's a box, every object, and that's everything in Python, every integer, every float, every string, is a box that has a bunch of other stuff inside of it, okay? So who knew, right? So the two types of things you're going to find in inside an object are, one, functions, okay? And then two, other things that aren't functions, but you can kind of think of those as like variables, they're other data, okay? So functions and other data. And there's special words, programmery words we use when we talk about functions and other data that are inside objects.

(1:40:33 - 1:45:10)

So, and those, let me try to teach you those special words and then I'll use them as much as possible so you get a hang, an ear for those words as we move forward. So in an object, if you have a function in the object, we use a special word, that's a method, okay? So a method is a function inside an object. And then if you have other data in an object, then we use a special word, attribute.

So methods and attributes, those are the two types of things that you find inside an object, okay? And they're really just fancy words, a method is a function inside an object, an attribute is other data inside an object, okay? And you can kind of think of it as a variable inside an object, but that is not a correct way to say it, but you can think of it that way. Alright, so what we're going to do is we're going to use a method on an object and we're going to add a flower to our flowers list, okay? So every list has a bunch of methods inside of it, okay? And we're going to learn just one of those methods, okay? So let's type the flowers variable, "f-l-o-w-e-r-s", and you could use tab for that, you probably should. And now we wrote flowers, f-l-o-w-e-r-s, and now don't put a space, just do a dot, period, full stop, however you say that, wherever you are, you know, a period or full stop.

"Flowers, dot." Dot, okay? And then now type append, a-p-p-e-n-d. "A-p-p-e-n-d." And now a left parenthesis, because this is a method, it's like a function inside of our flowers list. "Append, left paren." And now let's put, give it another quote quote.

So let's type in a double quote. And when I, during the workshop, when I asked for a suggestion for flowers, people suggested some flowers, but, but I heard someone said Lily. And I have a newborn baby here at home, and her name is Lily, so I was very excited about that.

So let's go with Lily for our last flower. "L-I-L-Y, Lily, quote." And then let's do a right parenthesis.

"Right paren." So that's flowers, dot, append, left parenthesis, quote, Lily, quote, right parenthesis. "In left bracket, 49 right bracket, colon."

And there was no output, but if we look at our flowers variable, "F-L-O-W-E-R-S, out left bracket, 49 right bracket, colon, left bracket, rose, violet, butter cu, p, lily, right bracket." Rose, violet, buttercup, and lily. Okay, it was on another line, that's why there was a little pause there.

So, so we appended to the end of the list, or we added to the end of the list, the new, a new, a new object, which is a string, lily. So now we have rose, violet, buttercup. And if we did len on the list now, L-E-N, and we gave it the list.

"L-E-N, len, left paren, F-L-O, float, W-E-R-S, right paren. Len flowers, out left bracket, 50 right bracket, colon, four." We have, we get four.

We have now four items in the list, instead of three, because we appended something to the end of it. So we, we, if we want to use a method in an object, we use that dot syntax, okay. So we do flowers dot whatever, flowers dot append, we'll add something to the end of the flowers list, okay.

So, and methods often work on the object that they're contained in. So the lists and lists have a number of methods that are quite useful. And so let's, I'm going to teach you one more function, and it, it will tell you what, what attributes and methods are inside a list.

So what are inside an object. So it will look inside an object, and it will tell you, give you a big long list of what is inside it. And I will say this, the output from this function is very long.

So you may get a little overwhelmed. So I would say, use the review functionality to tell what's inside it. Don't listen to the whole output.

(1:45:11 - 1:46:09)

So the new function, we, this is the third function we learned. We learned type, we learned len. Now let's learn dir, dir, d, for directory, dir, left parenthesis, and let's just use our flowers again, f-l-o-w-e-r-s, flowers.

I press tab to fill it in. So it's d-i-r, left parenthesis, flowers, right parenthesis. Did I type the right parenthesis? No I didn't. "Copy. Count.

Extend. Index." So we're hearing some of the, the, the methods inside the list.

So the things you can do with the list. Copy. Index.

Extend. "Insert. Pop.

Remove. Reverse" So I heard some interesting ones there.

Pop is an interesting one. We won't go into that one, but it pops something off the end of the list. But let's try reverse.

(1:46:09 - 1:48:33)

That's an interesting one. So I just heard it. It sounds interesting.

Let's try it. Let's do flowers.reverse, "f-l-o-w," and let's use filling it in, the tab, press tab to fill in, "e-r-s," flowers, dot, reverse, "r-e-v," and you can actually fill this in too. I typed in r-e-v. Let's see if we can fill it in. Reverse.

So it's flowers.reverse, left parenthesis, and then it doesn't take any arguments, okay? There's nothing that goes inside the parenthesis. So it's just going to be left paren, right paren, in left bracket 53, right bracket colon, and there's no output. But if we check our list, "f-l-o-w-e-r-s, flowers, out left bracket 53, right bracket colon, left bracket lily, buttercup, violet, t, rose, right bracket."

So it's lily, buttercup, violet, rose. It reversed our list. So that's kind of cool.

So methods are functions that live inside of objects and allow you to do something with that object. Usually they will work upon that object, not always, but often they will work upon that object. Every object has methods and attributes in it, or let's say almost every object.

But in practice, every object has methods and attributes inside it. Methods and attributes are different for different data types. So integers will have different methods and attributes than lists will have different methods and attributes from strings.

And strings, for example, will have methods to allow you to find strings inside of strings, or to uppercase the string, or to do things like that, to do things that are useful for strings. So you can kind of think of methods as useful tools in the toolbox that is that object. They allow you to do things with the object.

So we're kind of coming up to the end of this workshop. We're just going to do two more things. We're going to make a little very small application that will inspire you to come to the next workshop, or check out the next workshop, and to inspire you or motivate you to do more Python, the little motivational quotes.

(1:48:33 - 1:48:45)

And then I'll show you how to save your IPython session. And then that will be the end of our non-visual Python, the first workshop in this series. So let's create our little application.

(1:48:45 - 1:49:31)

And to do that, I'm going to teach you maybe the coolest feature of Python, or maybe the most powerful feature of Python. So let's clear our screen to kind of clear our heads. And we're in a new mode here.

We're going to be creating our little application. "In left bracket 54, right bracket colon." OK, we're all ready.

I hit control L to clear the screen. And now let's type this. Type this, and I'll explain what it is.

Type import. "I-M-P-O-R-T. Import.

Space. R-A-N-D-O-M. Random.

Space." So it was import random. And I did space there just so you'd hear the word random. Import random. And again, there's no output. But something happened behind the scenes.

(1:49:32 - 1:49:44)

And that is that there's a new object that we have now, random. It's an object called random. And let's just use our type function on it just to see what it is now.

(1:49:46 - 1:52:33)

I filled it in with tab, but you didn't hear it. So I'm going to try it again. "Left paren.

R-A-N-D-O-M. Random. Right paren."

So type, left paren, random, right paren. Let's see what data type this is. "Out left bracket 55, right bracket colon, module."

It's a module, OK? So what we did when we did import random is we imported what Python calls a module. And a more general word for it that programmers use is a library. And when we do that, basically we pull in a whole bunch of code that somebody else, someone usually pretty on the ball and who knows what they're doing, wrote to allow you to do something specific.

In this case, when we import random, we imported a whole toolbox of tools that is dedicated to working with randomness and to create random numbers and to choose random things and so on, which is a pretty cool library or module that comes with regular vanilla CPython, not even an Anaconda module. But we have it available to us here. And it's just an object like any other.

And you use the dot syntax that we learned the same way we did flowers.append to use that method. We use random dot whatever to access functions inside of this module object. And a module is really just an object to contain a whole bunch of functions and other objects and stuff that we can use in our own code to pull stuff into our own code.

So you can kind of think about it as a toolbox object. So really the module data type, it's kind of the sixth data type that I mentioned early in the lesson. So it's kind of that special data type that I was talking about.

But you can kind of think about it as a grab bag toolbox data type. And we can reach into it. And we don't call them methods.

When we reach into a module and we do things, we do still call them functions and so on, which is confusing. But in this case, if we use a function that's in a module, we do still call it a function. So make of that what you will.

But let's go ahead and write our little application. So what we're going to do is we'll create a blank list, an empty list. And then we're going to use the append method to add three motivational sayings to the list.

And then we're going to use the random module to pull a random motivational saying out. So let's go ahead and do that now. So we already imported our random module.

Let's now create an empty list. So I cleared, which I didn't necessarily need to do. But let's do, let's type, I'm going to make the variable name motivational quotes, which I know is a long variable name.

(1:52:33 - 1:52:47)

But there's nothing wrong with a long variable name, motivational line. And where you heard line there, that was me doing an underscore. So it's going to be motivational underscore quotes.

(1:52:49 - 2:05:12)

And if you want to have a space in your variable name, then you want to use an underscore. A space is not going to work. You can't have a space in your variable name.

You can have an underscore. Okay, motivational quotes equals. And then just do left square bracket, right square bracket.

So a list with nothing in it. So it's motivational underscore quotes, space, equals, space, left bracket, right bracket. And press enter.

And now we have a variable called motivational quotes. If we type it in by itself on a line, "M-O-T-I," and I'll let it fill in. "In left bracket 57 right bracket colon motivational line quotes. Out left bracket 57 right bracket colon left bracket right bracket. Left bracket right bracket." That was the output.

So it's an empty list. That's a left bracket right bracket with nothing in it. Okay, and so we now have a motivational, an empty list assigned to the motivational underscore quotes variable.

So now let's add three items to the list using our append method. So we're going to do motivational underscore quotes dot append. And then we're going to pass it, you know, we're going to do our parentheses.

We'll pass it a string with our motivational quote. So let's "M-O-T-I" motivational "in left bracket 58 right bracket colon motivational line quotes" motivational quotes dot append left parenthesis. And now let's do a double quote.

Sorry I talked over that. So it's motivational underscore quotes left parenthesis double quote. And now let's type a quote.

So let I'll do Hey I am getting the hang of this Python skill! getting the hang of hey I am getting the hang of this python stuff exclamation mark stuff and then do a double quote and then do a left sorry a right parenthesis. So it should be motivational underscore quotes dot append left parenthesis double quote type your quote whatever you want then another double quote then the right parenthesis. Okay so we're appending a string to the end of our empty list. I got a I got an error I mistyped something.

Let's type it I'm going to just type it again motivational quotes dot append just typing it fast. Okay so uh I and now we can check the list. "M-O-T in left bracket 60 right bracket colon motivational line quotes."

We're just checking what's in the list by putting motivational quotes the variable name on a line by itself "left bracket 60 right bracket colon out left bracket 60 right bracket colon left bracket hey I am getting the hang of this python stuff right bracket." Okay so we have one item in let's add two more really quick so let's do motivational quotes dot append quote and I'll say um uh I eat bugs for breakfast but not those kinds of bugs so I did motivational quotes dot append open parenthesis quote I eat bugs for breakfast but not those kinds of bugs close quote close parenthesis and let's add one more motivational quotes dot append space um and let's say um when does the data science stuff start I don't know how motivational that is but the answer is next week "in left bracket 63 right bracket colon" okay so now we have three um strings appended to our list three strings in the motivational quotes list so we have our you know three phrases or whatever now let's pick a random phrase from the list okay so let's do we're going to use random the library random dot choose it's choice sorry random dot choice so it's random dot choice and then we'll do it sounded like I spelled something wrong so let's start again random dot choice left parenthesis and then let's give it our motivational quotes list so it's random dot choice left parent motivational quotes variable right parenthesis "out left bracket 63 right bracket I eat bugs for breakfast b-u-t not those kinds of bugs" I eat bugs for breakfast but not those kinds of bugs so every time we run that line random dot choice left parent motivational quotes right parent it should pick out a random quote from our list so we've got a little almost like a little app so every time we you can now run that line over and over but you can press the up button "colon quotes right parent in left bracket 64 right bracket colon random dot choice left paren motivational line" so when you press up it fills in the line that we previously ran and then you press enter "out left bracket 64 right bracket colon out left bracket 64 right bracket colon when does the data science stuff start when does the data science stuff start" so every time we run this line it'll pick out a new one probably it'll be the same one "out left bracket 65 right bracket colon when does the data science stuff" so every time we run the line it will print out a new random motivational quote from our list so hopefully you know you can put something a little motivational for yourself in there and it's a little application they've written for yourself and you might be say to yourself uh Patrick is this really an application and I would say the answer is you know most applications are what programmers call crud applications which stands for create read update delete which basically means they just stick stuff in a database and then they pull it out and basically what we did here was we created a database with three items in it and now we're randomly pulling them out and so really what we've done here is we've created a crud application which is what 95 percent of applications are so yes you have created an application here it's it's not the most complicated application in the world but it is an application so so you know you can kind of pat yourself on the back a little bit there all right so this really is the end of the workshop so I hope that you have you know enjoyed spending a little time in the beginning of you know stages of python remember you are a programmer now and I just want to say you know we talked about motivational quotes you've created your motivational quotes I just want to very briefly give you a few words of motivation myself which is to say you know my background is 10 years ago I started my journey of learning python and you know I'm visually impaired at that time I had a little more vision but I you know still you know very limited vision and I had always my entire life felt like I was like struggling with computers because computers are really they felt like they were made for other people they weren't made for me they didn't allow me to do they weren't built for people with a visual impairment and they didn't make things easy they you know like they didn't make things the you know like with lots of hotkeys or however I or with uh like large text or with screen reader support or whatever they just weren't built for people like me that's how it felt for a long time and when I started learning programming I found that I could make functionality the way I want I needed it or I wanted it and that was very empowering and and really allowed me to completely change my my relationship with the computer and you know I was I for example the first one of the first real quote-unquote things I did with python after I started learning it was that I used a website that had audio books on it audio files and I and it was very difficult to access very inaccessible website and I found that when I I could write a piece of a little program in python that would access the website download all the data go through the data from the website and then pull out what I needed and I could run it every day and pull out the new information and it was very empowering to have it be able to parse that data and I didn't need to access the website in the sighted way at all I could just just access it the way I wanted to access it and so I just want to show you one other thing which you know and so basically that's the essentially end of our workshop but I will say you know do think about seriously about becoming a programmer because I think there should be more programmers in the world we it helps us make applications more accessible it makes us more visible and it empowers us to to do new cool things okay so I just want to show you one last thing which is how to save data in the in your it's to save this session that we've done together okay and we have a crying baby in the background so so this will definitely be the last thing so we'll do percent save and we have to check what our last line was so you need to know what the last line was that we entered "in left bracket 66" so it was 66 so we subtract one from that you wanted to get 65 so you'll need to know that for this so we do percent percent save save and I'll call this um non-visual underscore session so it's percent safe then a file name um percent save space non-visual underscore session session and then space and then you have to enter the number of lines you want to save so we're going to do one one hyphen 65 and I'm going to press enter motivational line quotes and now that should have saved in your home folder so users forward slash your name your entire session that you've had today so it's percent save space a file name space and then one to the last line okay so we had 66 we were on 66 there so we had entered 65 lines so I did one to 65 okay um that's the end of our session um thank you and I'm looking forward to seeing you all in the Python session um next week or you know whenever if you're watching the recording so have a great um have a great time playing around with Python and remember I'm Patrick Smyth um at iotaschool, iotaschool.com you can send an email to me at patrick@iotaschool.com if you have any questions thank you very much