Module 1: Video Transcripts

Video 1 – Where Do You Start?

So, when it comes to learning how to code, where do you start? If you're anything like me, you've probably heard some of these terms before, C, Java, Python, Ruby, PHP and the sheer volume of all the different things that are out there can be pretty overwhelming. At least, I know for myself that I was actually afraid to dive-in because of the fear that I would start in the wrong place, that I would learn JavaScript, and realize that I actually needed PHP or whatever.

And the truth is, once you actually start learning a little bit about coding, you realize it's not as big of a problem as you would think upfront, but when you don't know anything, you don't even know where to start. So, I wanted to begin by giving you, kind of a high-level overview of what's possible with coding, and to help you start to put together some of these concepts, just in your own mind, and create a mental map. So, I like to start by talking about something called a web application, and a web application is an application that you access over the internet.

So, we're all used to applications; we download apps on our phone, we download things like PowerPoint or even a browser or Microsoft Excel; these are applications and software that you download. But with a web application, you don't download it; you go to a website. So, you open your browser, and you go to facebook.com, or you go to twitter.com, and the application is actually running on someone else's computer; it's called a server, and it's in the cloud somewhere, and by going to the website, you're interacting with it, and the advantage there is you can all access one shared database, and it's possible for them to update the code without you having to keep downloading different versions and so, there's a bunch of reasons why web applications have started to become more and more popular.

Now, every application that you use has a front-end and a back-end, and the front-end is the part that you see. So, there are actually three different languages that come together to create the front-end. When I say the part you see, I mean the web pages themselves, when you go to Twitter, and you read your newsfeed, or you do that on Facebook, those websites are created using these three front-end languages. You have HTML, CSS, and JavaScript, and it's not one or the other, all three of them are working together, but they're doing different things. So, HTML is the things on the page; I have a friend who likes to describe this as the nouns, the verbs, and the adjectives.

So, if you have a sentence such as, the boy kicks the red ball, you've got the boy and the ball, those are the nouns of that sentence, and in the same way, the HTML is the nouns of the web, the CSS is the adjectives, so that's the red, that's what makes different things on the website look the way that they do. So, that would describe the colors of the webpage or the orientation or how things are designed, and then the JavaScript is kind of like the verbs, it controls the behavior or the actions on the website. So, when I see, you know, a little notification or a pop-up or something like that telling me that there's a new tweet for me to reload, that's the JavaScript.

So, these three things are working together, but we're not going to be dealing all that much with the front-end in this class because what I want to talk more about is the back-end, and

the back-end is the part that you don't see. So, the back-end is kind of like this black box. Most of us are at least familiar with those front-end languages, like we've heard of HTML or CSS or JavaScript before, but the back-end is actually, kind of, the, you know, the part of the iceberg that's below the surface of the water, so, you can't really see it or interact with it. So, you've got two parts of that, the rules and the database.

Now, your database is the part where all of the information is being stored, everything about your users, their emails, passwords, photos that they've uploaded, tweets, status updates, events, whatever it is, it's all shared in the database. And then, the rules are actually the part that decides when someone logs in, and they go to, say their, their twitter feed, what tweets are they going to see? You know, what information needs to come out of the database, and then how does it need to be organized to display it? So, in terms of languages, you've got one database language, really, and it's called sequel or SQL, both of those are interchangeable. But then, for the rules portion, that's where all these other languages fit in.

So, some of the more popular ones are PHP, Python, Ruby, Java, you've got C and a lot of other things that fit in there, and that's where you really, you know, get to choose which programming language is used. Now, often people ask me, what languages they should learn or, you know, they say I have this idea for a product that I want to build or this app I want to create, what language should I use?

And it's kind of, a little bit of a funny question once you know a little bit about how programming works because it's sort of like saying, I have a story that I want to write, you know, I came up with this, I had a dream and I came up with this story, it's a love story, you know, between a boy and a girl, they meet and they can't be together, what language should I write this story in? Should I write it in English, or French, or Chinese? I mean, the reality is you can tell that story in any of those languages, it doesn't mean that all the languages are the same, right?

Some languages have different ways of indicating past and present tense, and some languages you have to remember if a noun is male or female. But fundamentally, all of these languages are used for telling stories, and in the same way, all programming languages can do things, they're all the same, just a little different. So, I've created this example, I'm pulling out three programming languages, PHP, Python, and Ruby, and here's what a little snippet of code might look like in each of those three languages. I'll, you know, leave it to you to take a little look and to see if you can figure out some things about how the language works.

And so, you know, as you'll see PHP, you've got 'echo "Hello World";' Python is 'print("Hello World")', Ruby is 'puts "Hello World"' And when you run this code, and we'll talk about what running code actually means pretty shortly. What you get is the same output, 'Hello World'. So, if you're just looking at the output, you actually don't really know what code was used to generate that output. Now, Python itself is just a language for humans to talk to computers; actually, all of these programming languages are. Languages started out being very computer-friendly, but not very human-friendly.

So, if we go way back to the day of when computers were first starting out, and a lot of these programming languages didn't even exist, they've been invented over time. You start with something called binary. Now, in binary, here's how you might 'Print' something like, 'Winter is coming.' It's a lot of numbers and letters that are very hard for us as humans to actually understand what's going on. So, at the lowest, lowest level this is actually what's happening on

a computer. The computer, this is just zeros, and one and, in fact, this is actually of what you're looking at is a representation of those zeros and ones, it would be even longer in zeros and ones, but it's still, you know, more or less, impossible for us to code in.

So, pretty quickly, people realized, well binary's not going to work. So, they invented something called assembly. Now, assembly is one step away from binary in the sense that, you know, I would code this, and this also does the same thing as the previous example, just 'Print "Winter is coming."' And this very quickly just gets converted into binary by the computer, and then run and does the same thing. And you start to, at least, recognize some of this, but it's still pretty illegible. I mean, you've got section, _start:, mov, int but, you know, it's pretty complicated to do something, even relatively simple.

Now, you've got Java. So, sort of, representing the next step in human friendliness and readability. This is how you would 'Print "Winter is coming."' in Java, and there's still a lot going on here. You've got, you know, Public class, static void main, all these things, which I think make Java, not an ideal language to, at least, you know, learn when you're starting out because to do even the most basic thing, there's all this overhead of concepts that you have to learn and become familiar with first.

And then, you've got Python. So, just a simple 'Print ("Winter is coming.").' And the nice thing about this is I can show you this example. You can understand it without having to, you know, really understand the specifics of exactly how this Print works. But at the end of the day, the computer is actually just converting all of this into binary, anyway. So, with that, let's talk a little more about what Python is.

****

Video 2 – What is Python?

So, what is Python, actually? Well, it was created in 1991 by this guy, Guido van Rossum, who's really a very friendly guy. In the Python community, he's kind of known as the benevolent dictator for life; it's the title that they have anointed him with, and interestingly, Guido created Python, and then he ended up getting hired by Google when they first started becoming popular. Google, going back, if you look at their white paper, from when they were just Stanford Ph.D. students, were released this code of how to make this idea happen, and they used Python.

Who knows how they initially found Python, but as they started growing, they made the really smart decision to hire the guy who invented Python, and as a result of that, when all of the smartest CS students were graduating from school, and they wanted to learn Python from the best, they would go work at Google. So, interesting from like a strategic business decision. He now actually works at Dropbox. And just a random fact. Python is actually named after Monty Python, not the snake, as most people think. So, when you think about, you know, what companies actually use Python, the answer is just about all of them.

So, I've listed a few examples here; Dropbox, Google, Pinterest, Washington Post, et cetera. But even if a tech company doesn't use Python for their main product, they're most likely still using Python somewhere because Python has become kind of like the de facto language for data science and data analysis. So, even, you know, they might just be using Python for analyzing data, even if the actual product is built in some other language. So, Python has actually broken

into the mainstream a little bit more than a lot of the other languages for reasons that will become more apparent as we start learning it, but just simplicity.

I talked about human readability in the previous video, and so when you're teaching beginner's language, Python is really a nice one to start with, and it's still very powerful, even though it's simple. And as a result, as I mentioned, it's one of the fastest-growing programming languages, and it's also one of the most wanted programming languages when it comes to, you know, asking companies and asking developers what language do you find the most valuable. It was actually listed as number one in Stack Overflow, which is an online, very popular forum for developers that we'll actually end up using in this class for troubleshooting and questions and answers.

It was voted number one in terms of the most wanted languages. And, you know, I constantly see more and more articles coming out. For example, this one that says that Citi wants all their analysts to add Python to the list of languages on their resume. So, it's no longer something that only engineers are expected to be able to do. It's kind of like what Excel was, say 15 or 20 years ago. Now, if you know Python, even if you're not a developer, you can find uses for it that would make you be able to do some really powerful things even in, you know, a business-related role. So, we're specifically going to be using Python 3.

And I know we just introduced Python, and I'm already sort of throwing a wrench into it, but there's Python 2 and Python 3, and for some reason, even though Python 3 came out in 2008, a lot of people still use Python 2. And actually, the reason for that is because a lot of organizations, when they were building tools to do data analysis, or whatever, built it using Python 2 and when Python 3 came out, there was not a huge amount of consideration put into how do we make sure the previous code written for Python 2 still works in Python 3, so, a lot of things would break and companies and developers thought, oh, I'm not going to worry about that, I'm not going to upgrade it until I really need to.

And even though it's been, you know, more than 10 years at this point, a lot of companies still use Python 2. Although, to be fair, at this point, more of the coding has switched into Python 3, and so it's still a very good reason to learn. But I wanted to point this out because as you're using online resources, such as troubleshooting problems you're running into, there's always the possibility that what you're looking at might be code written for Python 2. So, for example, the very popular online code learning site Codecademy still teaches Python 2.

And there's this website called Python 3 Readiness, which catalogs the top 360 most popular third-party libraries and tools that you can use, kind of like plugins, to make Python more powerful, as we'll talk more about later in this class. And at this point, almost all of them, 359 out of 360, are upgraded for Python 3. So, they're ready to be used. So, we've definitely tipped over into Python 3 becoming the de facto version of Python that's used, but in case you're curious and, you know, honestly, when I'm, you know, explaining this to beginners, I just say the only real thing you need to know about the difference is this.

So, I've shown you Python 2 code on the left and Python 3 code on the right, and you can probably already figure out the difference, which is that the Python 3 code uses parentheses around the text and print and the Python 2 code doesn't. But when you're looking online, and you see code with a print that doesn't have parentheses, it's a really good telltale sign that the code was written for Python 2 and most of the time, if you want to get that code to work, all you have to do is add those parentheses around there.

Of course, there are a few other differences that, you know, could pose a problem, but for the most part, that's, you know, all you need to know. And when there are some examples where it is more relevant, I will point it out to you. And that's all I really have to say about that. I think that right now, we should do some is basic setup and dive into our first Python script.

****

Video 3 – Quick Sanity Check

So, before we get started, let's do a quick sanity check, just to make sure that we've successfully set up our development environment and aren't going to run into any glitches. So, I had you install a text editor, one called Atom, which we'll see shortly. I had you install Python using the Anaconda's downloader, and also check out the command line, which should be on your computer. Now, the command line has a different name, whether you're on a Mac or a Windows, and I pointed that out in the previous one, the setup video. On a Mac, it's called terminal, on a Windows, you should use the Anaconda's Power Shell command line.

So, go ahead and open that up and you might have to find it, you know, just search your computer. The easiest way on a Mac is the spotlight, and Windows has similar search functionality. So, look for the terminal. Nope, not that one. It's called terminal. And there are different kinds of command lines, so that's why I would say use the ones that I'm recommending because it's going to be the most similar to what I'm showing you. And this is what the command line looks like on a Mac by default.

I'll change the formatting soon to make it easier to read. On a Windows, it's already a little more legible. But the way that you can check to make sure you did this properly is by going into here and just typing ‘python --version’ and then hit Enter. So, if the install worked correctly, what you should see is Python 3.7. something. Now, the actual number doesn't matter so much, as long as you have at least 3.7. So, it can be 3.7.X or maybe even 3.8, depending on, you know, what version of Python is the newest version that came out.

As long as it's not below this number, because we will be using some functionality that's new to 3.7, and if you have 3.6, for example, or if you have something older, like maybe even 2.7 or 2.6, then you're going to want to go back and make sure you have all the same things ready. By default, by the way, Macs actually come with a much older version of Python, so if you see 2. something, it means that something about the Python installation didn't work properly.

And if you're on a Windows, and you see an error message that says the Python command cannot be found, then it also means that something didn't work about your install and you're going to want to go back or talk to the, you know, class facilitator and see if you can get help with this. Also, if you're on a work computer, there might be some sort of, you know, let's say anti-virus setup or some permissions settings that have to be changed. So, you might want to talk to your IT team if that's the case, as well. But hopefully, you are able to get this working, and in the next video, we'll talk more about what all these things that we just installed actually do.

****

Video 4 – The Command Line: Part 1

So, the command line is actually a good place to start, and even though it's not exactly Python, it will be necessary to understand the command line in order to run Python code. So, like I showed in the last video, you'll want to start by opening up your command line. Again, if you're on a Mac, it's called terminal and, on a Windows, the command line you should use is one that comes with Anaconda, called Anaconda's PowerShell. So, I'll go ahead and reopen it now. I'll just search for terminal, and there it is and open it.

Now, by default, just from like a visibility perspective, this is kind of scary, and I will walk through what we're seeing here. But the first thing I want to do is actually change the theme on my computer to make it easier to read, so it's not just small black on white text. If you're on a Mac, you can follow along with what I'm doing, and I've actually included instructions along with this video for you to customize it the same way that I do. If you're on a Windows, the default is actually pretty good. There we go. So, the text is just a little bigger, basically, and it's changed the theme.

So, this is called the command line. And what we're actually seeing here is something called a prompt. So, let's break it down. This first line here, that's actually something that only Mac users see, and it shows them the last time they used the command line, which I've never really found a good reason to, to look at it. So, it's there, but I'm not sure why. But the next part you see, you've got a few different things. This base, this is actually something that was added by Anaconda, and it has to do with the fact that you can, sort of, contain different versions of code on your computer. We won't be using it in this class.

So, you don't really need to worry about base itself. The next part where you're seeing is the name of your computer. So, in my case, it's Mattans-MacBook-Pro. But you'll probably see your computer name if you're on a Mac. If you're on a Windows, all that you'll see is your location on the computer at that given time, which is also what I see here, this little squiggly is called a tilde. It's actually telling you where on my computer I am, which doesn't make all that much sense right now, but I'll explain that in a second.

And then this is the user name, and then you have this dollar sign that is just an indication that after the dollar sign, that's where I can type stuff and run commands. That's what they're called, things you type in, and you hit Enter, you are telling your computer to do things. That's why they're called commands. And then your computer responds with something, depending on what you type in. On a Windows, it's usually a little arrow, and either way, the commands that I tell you should be the same. So, the first command I'm going to tell you is ‘pwd’.

So, just type ‘pwd’. And in all of these cases, make sure you are typing exactly what I'm typing, as well, and that includes the case-sensitivity. So, I'm typing all lower case. Whenever I do that, you should type all lower case, as well. Sometimes, case-sensitivity won't matter. So, this might work with some capitalization, and sometimes it will. So, to avoid any kind of problems later on where you're doing something and it's not working, and you don't understand why it's good to get in the practice of just starting to pay attention to the little things because those will matter. So, if you type ‘pwd’ and you hit Enter, what you should see is a location on your computer for where you are right now.

So, I'm seeing '/users/ mattan'. So, that's actually a folder on my computer, and that's by default when I open up the command line, that's where I am, which, what does that mean, that I'm in a particular location? Well, it doesn't make that much sense when you look at this, and, by the way, on a Windows, you'll see another path, but it should be like a C colon, and then the location and the slashes are going in a different direction. But that just has to do with how Windows and Macs are, are set up differently. It's the same thing; it's a location on your computer.

But when we're working with a command line, think of like an old text-based game, where, you know, there's not a lot of visuals, but it describes to you, like you wake up in a room somewhere and you don't know where you are. And there's three doors in front of you, but they're locked, and you need to find the key. What do you want to do? And then you type in, you know, search by the cabinet or whatever. You're just working with text, and so that's one of the things that's kind of overwhelming about the command line when a lot of beginners are first starting out. It's like how do you even know what to do here. So, 'pwd' is one thing that tells you where you currently are.

The next command that I'll tell you is actually the only command that's different for you depending on if you're on a Mac versus a PC. So, first, I'll show you the Mac version of it and then the PC version, and it's just one word that's different. So, the Mac version is 'open .' So, type this exactly open, all lower case, space period, and then hit Enter. What you should see is something on your computer pop up. And in my case, it's a folder on my computer. And actually, if I look, you'll notice this path for this folder is exactly what I saw before at '/users/mattan'.

You might not see the same kind of layout that I see. It might be a little more like this, but there's different ways of viewing it. On a Windows, the command is 'start .' It's the same space period, but the first word is open instead of start. So, on a Windows, you run 'start .', and that should do the same thing. And if you hit Enter, it'll pop up on your computer. But if you run the wrong one and you hit Enter, you'll get something like a command not found error. And that's fine; then you'll just know to use the other one. So, this is actually where I am on my computer right now when I open up the command line.

****

Video 5 – The Command Line: Part 2

So we're in the command line, and I've already showed you two commands, 'pwd' and 'open .' or 'start .' depending on if you're on a Mac versus a PC. By the way, I didn't mention, but 'pwd' stands for Print Working Directory. So, each of these commands I'm going to show you as a thing that they stand for. The next one is 'ls'. So, you just type 'ls' and hit Enter. Now, see if we can figure out actually what 'ls' is doing. Let's take a quick look, and then look at the folder that was open before, and you should notice that actually what we're seeing here when we type 'ls' and hit Enter is the same thing that we see here with the folder open.

It's actually all of the folders that are on our computer in the folder that we're in, it's because 'ls' stands for list, and it just says, list everything that's in my folder, it will also list files as well, but we don't have any files here. The next command, so we know how to see where we are right now, we know how to see what's in the folder that we're in, and also open it up. But

how do we move into one of those folders or out of one of those folders? The next command let us do that, and it is ‘cd’ stands for change directory, but this is one where we have to put a space and then tell it, where do we want to change directory into.

So, we have the option here to change directory into any of the folders that are in the folder we're currently in. So, if we chose, for example, Desktop, we would type 'cd Desktop'. And again, here's where case sensitivity might matter. So, just if you're going to the Desktop, make sure you type it, the same way that you see it, with a capital D day and then hit Enter, and now, I'm in my Desktop, and you'll actually notice that for me this little '~' here, now says Desktop, and that's actually because this '~' is short for that folder that you first find yourself in, it’s called your home folder. So, basically, it's your username in Users. So, now we're in this other folder, and actually, if I type ‘pwd’, I can confirm that yes, in fact, I'm in ‘/Users/mattan/Desktop’.

And you know, I can do this, and basically, navigate around my computer. Now, if I do 'ls', I won't see anything here because I don't have anything on my Desktop. If you do, you'll see that there and you can go even further. But let's say, what if I wanted to go back a level now, how do I 'cd' back a level? It's not so intuitive. The way you would do that is with ‘cd ..’, so, '..', in this case, is actually, it's a placeholder it's sort of, it's a little bit of a magical symbol that stands for the folder that is one level outside of the folder that you're currently in. It's kind of like clicking the back button in your browser, says, just go back.

So, if I do this and I hit Enter, now I'm back. So, in all of these cases, by the way, when I did 'cd', 'cd' is the command, and then the thing that comes after the space, it's an argument, it's one of the options, it tells the command, you know, gives it more information that it wouldn't know where to go into without that argument. So, '..' is the folder that's like back one level, and by the way, '.' stands for the folder that you're currently in. So, that's why when we did 'open .' or 'start .' it said, open our current. So, why are we learning all this stuff? It's because this command line is where we're actually going to run Python code.

So, we'll write Python code using the text editor, but you can't run Python code in the text editor, you can run it in the command line. So, I'm showing you just enough of the command line for you to know how to actually run this, but we won't dive too deep into all the different possible things you can do here because there are thousands of different commands and different permutations. This is effectively, if you've ever heard of like Linux or Unix, a lot of these commands are what you're learning; you're basically learning those commands.

The language here is actually called Bash. In case you do want to dive into it further. But that's all we really need to know for now. Just to recap, ‘pwd’ tells you where you currently are, 'open .' or 'start .' opens it up, 'ls' lists what's on your computer, and then in that folder, and then 'cd' and a folder name will allow you to move into that folder, and that'll allow us to write Python code somewhere, and then move into that folder and then run that code. So, in the next video, we'll be creating a folder, and then we'll be diving into actually running Python code.

****

Video 6 – The Command Line, Part 3

So, I've gone ahead and closed down the command line, and I'll open it up and start again from the beginning, which I recommend doing because some of this stuff I've just taught you just can take a little bit of practice. So, get a little bit of practice and get comfortable with those commands. So, I'll start out by opening up the terminal, and by the way, what you can do to make it easier, to open up every time is you can actually right-click on it, in the Dock, and then there should be an option for pinning it there or keeping it in the Dock. On a Mac, it's Keep in Dock, and on Windows, it's pin to, you know, the Start bar.

So, that'll allow you to go back there and open it up again quickly. So, we start out, and let's see; we'll start by doing 'ls' and see that I have a folder for my Desktop, and I'm going to move into that folder. And the goal actually right now is going to be, to create a folder on our desktop, we'll call it code, and that's where we'll be storing the code for this class, that way you have a similar setup to me, and that way we don't have different Python files all over our computer, and we lose track of where they are, and we know that they're in a folder called code on our desktop.

So now, I'm in my Desktop, and I'll actually show you another command now, which I didn't talk about before, which is 'mkdir code', and copy this, run it, and let's see if you can guess as to what this actually did. That's right. Magically, I have this folder called code on my desktop, and it just appeared. Now, I could, of course, just have gone into Finder and clicked File, New Folder, that would work the same. So, why did I do that? I don't know; I just thought it was fun, and that's kind of nice to see magical stuff show up there. You won't really need to remember 'mkdir' to be honest.

But now that I've created this folder, I can move into it, or I'll need to move into it because I'm not in this folder yet, in the command line. So, to move into it, what's the command? It's 'cd code', now I'm in the code folder, and of course, I can confirm by typing ‘pwd’, hitting Enter, and I should see that it's telling me, I'm in the code folder. So, for the next video, there's this happy hour Python file that we're going to be interacting with.

So, you can actually go ahead and download that file now, I've got it here in my Downloads folder, as you can see 'happy_hour.py', you don't need to open this file yet, at all or worry about what's inside of it, but I do want you to take this file, and move it into your code folder. So, I can just drag it and drop it into here. And now, I can confirm and just make sure that it's in there, and how would I confirm in the command line whether or not that file is actually there? That's right; I can type 'ls' and hit Enter. And I should see 'happy_hour.py'. So, make sure you move the code into that folder, and you see it when you type 'ls', and then we can move on to the next video, and actually see what this file does.

****

Video 7 – It's Happy Hour!

So again, to recap, you should have downloaded the 'happy_hour.py' file, put that into this code folder on your Desktop. And again, I've closed down the command line, just to get us a little bit of practice. Opening the command line, moving it into this folder, and then, I'll show you what to do. So, go ahead and do that now. And I can do it here as well, 'cd Desktop', 'cd code'. And now, just as a quick sanity check 'ls', and I should see this file here. Now, I've given you a

file with Python code inside of it, but we're not going to open it up, and actually see what the code does, we'll do that shortly in the next video.

But first, I want us to run this code and try to figure out if we can, you know, guess this to what it does. And the way that you run Python code is you type ‘python ‘ and then the name of the file. So, here we have python happy_hour.py, and it needs to be typed exactly like this, case sensitive, and then hit Enter. So, take a look at what I have, and then take a look at what your code shows. Are they same, are they different, what are you noticing? Well, first of all, if you get some sort of error, that means there's a problem, there's a few different things that might be going wrong here if you don't see any output.

The first is if it says that file cannot be found, then it means that the file isn't actually there, maybe you didn't move it into the right folder. So, just double-check, or maybe you made a little bit of a typo, and you didn't put the underscore in 'happy_hour.py' or whatever, just double-check and make sure the code was exactly like what I saw. If you see three arrows, you might have accidentally just typed python and hit Enter before you typed the name of a file, and that's totally normal, nothing's broken, but it does mean you entered into something called Python interactive mode, which we won't go into now.

So, the easiest way to exit that is to just type ‘exit’ in parentheses, and then hit Enter or you could do ‘Control D’ on a Mac, or ‘Control Z’ and Enter on a Windows, and that should get out of there as well, or you can just close down the command line and open it up again if you're freaking out, you can always just do that. And it usually brings you back to where you started, which is also fine. But hopefully, everything worked well, and you got some output, and you probably found that it's not the same as mine. Let's actually try running this line again. So, 'python happy_hour.py', and there we go, I'm getting a different result.

Now, actually, run this a few times and see if you can figure out the general structure of like what's actually happening here? And here's a little bit of a shortcut, instead of typing out the whole thing, something you could do is hit the up-arrow on your keyboard, and it should pop-up the last command that you just ran. Actually, you can keep hitting up and down, and go through kind of a history of all of your previous commands. So, you don't have to keep retyping it every time. You just click up, hit up, and then hit Enter, and go on.

So, have you figured out how this is working? Well, it looks like what's happening, and I haven't taken a look at the code yet, so we'll dive into whether or not this is true? But it looks like what's happening is we have this sentence like, how about you go to someplace, a bar and there's kind of this list of them that are randomly populated, and then a width, and then a person, and there's this list of people that are randomly populated in there. And every time you run it, a different name and a different bar pops-up. So, let's actually, in the next video, take a look at this code and see what we can figure out about how it works.

****

Video 8 – Happy Hour: Reading Code

So, we've got this happy hour file, and we've run it a few times just to see what the output is or what happens when I run it. And now, it's time to take a look at the actual code. So, how do we do that? Well, look into our code folder, and I see this file, and by default, if I double click this or right-click and select open, I don't really know what application it's going to open it. It

could be a text editor; it could be whatever your computer is set up to open it with by default. But what we want to do is open it with the text editor that we downloaded called Atom or whatever text editor you're using or whatever you prefer.

But I have this downloaded one called Atom. So, probably the easiest way to do that is to either right-click and select Open With and then click Atom. You might want to make sure that it opens with that by default. So, I can do it this way. Or I can open Atom on its own and then go into File and then Open File and find this file itself. So, you can go at it from either way. But consider it, it's sort of like Microsoft Word. Right? You can open a Word file in that application. So, I'm going to open it with Atom. And what I should see is something like this.

A little bit of code, and I'll, I'll maximize this a little bit to make it easier for us to read. And this is actually the code that generates what we saw before in the command line when we run it. And before I talk about Python at all or introduce any of these concepts, I want you to actually read through this file. So, I want you to pause this video, read through this file. Top to bottom, left to right, and see if you can come up with a guess as to how this is working.

See if there are any questions that pop up in your mind about this might be working. Things you understand, things you don't understand. And then, I'll walk through it, how I might read it and then we can go from there, and maybe it make a little bit of changes. So, hopefully, you've read through this file so far. Now when I'm reading a file, I typically like to think about it in terms of the top, the middle, and the bottom.

And with code, files can get really long so that can kind of break down at a certain point. But what you have going on here is the top I think of as kind of like the setup for the file. It's, it introduces some of the data, some of the libraries, the things like that which we'll talk about later. But we read the top here, and we've got line number one, 'import random'. Now that's actually probably the most confusing line in this entire file, and if you don't know anything about Python, you don't really know what this does. So, okay, 'import random', whatever, we'll come back to that.

The next part on line three, we've got bars equals, and then it looks there's a list of 'bars'. We've got 'The Hamilton', '1020 Bar', 'The Heights', 'The Dead Poet'. And that seems somewhat straight forward. Now you might not understand exactly how this is working. Like what are these square brackets doing there? Quotation marks, commas, like, how, how, what is this thing? But you can understand at the very least that it looks like there's a list of 'bars' and here in line eight, you have another list, a list of people. And then, so that's sort of the head, that's the set up of this information, the data of this file.

Then we move to what I think of as the middle of the file, and that's lines 13 and 14. This is where some, a little bit of work is being done. So, on line 13, 'random_bar = random.choice(bars)' it looks like what might be happening here is we're just choosing a bar randomly, and then, you know, calling it random bar. Line 14, we're doing something similar, we're choosing a random person. Now, what's happening here is we're creating variables, but I haven't talked about creating variables yet. We'll do that in a later video.

So, you know, it, it's somewhat intuitive; we're creating, pulling out a bar and a person, putting them in this thing, and then in the last line, line 16, we're doing a 'print (f"How about you go to {random bar} with {random person}?")' So, again, a lot of the specifics of this probably aren't going to make sense, and you might have some questions such as, you know, random, is this

random the same as this random we're seeing down here? Or what is little f doing here? And we'll, don't worry; we'll get to all that in due time. But before we get there, I actually want to give you a little bit of a challenge and see if you can figure out how to solve this.

****

Video 9 – Happy Hour Challenge Setup

So, I've got actually three challenges for you, increasing in difficulty. So, I'll see if you can figure out all of these. The first is, you might have noticed, I misspelled Samuel L. Jackson's name, which is kind of a mess up on my part. So, can you go into the code and fix that for me? Number two, can you add another person to the list, maybe pick a friend or pick your name, add them into the list and then run the code, and actually confirm that the person shows up there? And then number three can you have this print out a second person as well, two random people instead of just one random person? So, take five minutes or so, give this a shot now. Don't take too much time if you're, you know, running into trouble, that's fine. In the next video, we'll walk through how I might solve this, and you know, show you some interesting things about this. It would take five minutes or so and see if you can figure this out.

****

Video 10 – Happy Hour Challenge Solution

Alright, so hopefully, you figured out how to do some of this if not all of it. That would be great. But let's walk through the solutions to this challenge. So, the first one was quite simple. Can you change Samul L. Jackson's name to fix it? And to do that, we would look into the code, and you, you would probably find Samul L. Jackson here on line 11. And it looks like that 'e' is just in the wrong spot. So, I can delete the 'e' and put that in there. And let's run the file and make sure we got it working. So, to make this easy to read, I've got this program called Moom that'll actually just move this around a little bit. So, I'll put the text editor code on the right and then the command line on the left, and that way, we can sort of jump back-and-forth.

So, let's go ahead and rerun python happy hour. Remember, it's 'python happy_hour.py' or just the up-arrow and Enter, and keep running this until I get Samuel L. Jackson's name. Now, this can take a while, even though it should be random. It looks like I actually still have his name is showing up misspelled. So, what do you think is going on here? Well, if you guessed that I forgot to save the file, that's right. I made the change, but I didn't save the file, so, actually my change has not been made permanent in the same way that if you make an edit to like a Microsoft Word file or Excel file or PowerPoint, but you don't save it, and you send someone the file, they'll end up getting the old version of it.

So, you always want to, every time you make a change, make sure you save the file, and that's as simple as just going to File, Save or hitting Command or Ctrl+ S, and that'll make the change. And there's actually a little bit of a hint, by the way. If you make a change in Atom, and you haven't saved it yet, up here in the tab, what you should see is a little dot, a little blue dot, and then once you save it, the dot disappears. So, that's one way of knowing. Oh, did I make a change there? I forgot to save. So, now I've saved it, and I should be able to run this. Eventually, see that my change was reflected.

There we go. By the way, if you ever get a little bit overwhelmed by all the different texts that's in the command line, you can actually type 'clear' and hit Enter, and that'll get rid of everything. There is also a shortcut that is ‘Command+K’ that works only on Macs that I'll use sometimes, and I won't have to type clear. But if you want to just get rid of everything before, that's an easy way to clear it up. Alright. So, the second thing is we wanted to add a second person to the list.

So, if you took a look at how the people list works and wanted to add another one, you'd probably would figure out that you would put a comma after the last one, hit Enter, you should see this square bracket go down and then I'll put in ‘Daniel Guetta’, my co-teacher here and save this file. And I'll go here and just run it and make sure that his name pops up there. There we go. Now, it's possible that you didn't notice that there are commas between each one of these bits of text. So, let's see what happens without the comma. I've gotten rid of it.

I run this code, and everything looks fine until, now I'm seeing here, 'How about you go to The Heights with Samuel L. JacksonDaniel Guetta?' It's sort of smashed together like that, right? So, I've actually introduced a little bit of a bug here by forgetting to put in a comma. And this is pretty common, a little thing like a comma can make a difference, and it's not creating an error. Sometimes when you have a bug, you'll get an error message, and it'll tell you something's not working, and that's how you'll know.

But sometimes, you won't get an error, and you'll just realize at some point, oh, actually, this isn't working how I expected it to work. So, in this case, we kind of learn something about Python, which is when you have two bits of text together, they often just get smashed together like we have here. So, the correct way to do this in a list is with a comma. And we'll talk more about lists later on in another video, but just know that, you know, that's something that can go wrong. So, part of the coding process, it's a lot of running code, checking to make sure that what you're getting back is what you were expecting, and then iterating from there.

And as a, sort of a side note on that, I highly recommend running code often as you're writing it so that at every stage along the way, you can see whether it worked right or not as opposed to writing an entire thing and then running it at the end. And if it doesn't work at that point, you don't really know what might've gone wrong. Okay, now the last one is the most challenging. Did you figure out how to add a second person to this output so that it says, how about you go to random bar with random person one, random person two and so forth? So, there's a few ways to do this, actually.

One way that people often think, and I'll expand this, so it's a little easier to read, is you just go into here, and you actually add and random person So, you probably figure out that you need these curly brackets, {random_person} You want to type it correctly and save that. Now you probably would have already figured out that this isn't going to do exactly what we were expecting it to do because if I run this now, I'll see that I always get the same person every time and that's because I'm using the same random person in both cases. So, okay. It doesn't pick a second random person for the second one.

So, maybe you'll realize, at this point, now I need to create another random person. Maybe I take random_person2 is what we'll call it, and then, you know, just sort of copy the code from above. You can literally copy that in, and then at this point, I would actually have to replace this second random person here. Save that, and move this over. Now run my code. Let's go

ahead and 'clear' this just so that it's easier to see what code is produced by what. It's now 'python happy_hour.py' Okay, works pretty good, right? We've got Mattan and Costis Maglaras.

But what you'll notice at some point is that this, you know, unintended consequences, sometimes the first random person is the same as the second random person. And I saw that pretty quickly here with, 'How about you go to the Hamilton with Daniel Guetta and Daniel Guetta?' and why is this happening? Well, it's because even though both random people are being picked randomly, one out of every five times actually, the second person will be the same as the first person. So, I've introduced this little bit of an edge case, which means the code is working fine most of the time, but every once in a while, something is going wrong. Now, I won't show you how to solve this problem at this point.

We will eventually know enough about how to solve this. And you might, at this point, already start thinking about some ideas. You know, maybe some of you are thinking oh, let's pick two random people but check first to make sure that they're different and if they are the same, you know, pick another random person to replace the second one. Or another way to do it might be after you pick the first random person, take them out of the list before you pick a second random person. There is a number of different ways of solving this, and this is one of the cool things about Python is that there is many different tools that you have available that I'm going to be teaching you about in the next few videos, but it is kind of a creative process to think, okay, I have this problem, what tools can I use to solve that problem?

****

Video 11 – Challenge: Create Your Own Randomizer Setup

So, I've got a challenge for you again. I want you to create your own randomizer script based on the 'happy_hour.py' file that we've been working with, and when you're done, I want you to post your Python code and check out what some of the other students came up with. So, I'll give you a little bit of inspiration first. Now, you might be thinking, sure that's, you know, this file is kind of fun, but doesn't seem all that useful. But I want to show you three examples of how this was actually used in the wild, that each became relatively popular. So, the first is something called, what does your startup do? itsthisforthat.com. And it comes up with startup ideas by randomly matching two different kind of startup buzzwords together.

So, we've got, wait, so what does your startup do? Well, basically it's like a meme generator for parking tickets, or I can click refresh here and say, it's basically like a Kickstarter for gay marriages or Groupon for cougars, just two random ideas and it's kind of fun, but this website kind of blew up. The second one is, well, pardon my French; WTF Is my social media strategy. And what this one does, which I love, is it generates a social media strategy by just kind of throwing a bunch of buzzwords together into one sentence, and you'll find that it's surprisingly on point. We've got, maximize buzz by driving word of mouth from relevant influencers.

I think I've actually seen an agency use that one, or activate audiences by giving them compelling social experiences encouraging advocacy; I just think it's great. The final one is WTF Should I make for dinner. So, this one's actually kind of useful, you know, it generates a random recipe that you can make for dinner, I can refresh it if I don't like that, or there's even a vegetarian option, you can click this link over here. And then, each one of these, actually, if I'd say, Oh! southern buttermilk fried chicken looks good, I click on that, and then it pops-up

the actual recipe for me to use. So, that's kind of useful. So, go ahead now and use that as some inspiration to come up with your own randomizer script, and then post the code into the forum and see what the other students come up with as well.

****

Video 12 – Two Ways of Running Python Code

So, I want to introduce two different ways that you can run Python code, and we've already seen the first one. When you have a file and it contains Python code, I can run that file by opening up the command line, typing Python, space and then the name of the file, and it'll run that code. A second way is I can open up something called the Python Interactive Mode. And there's some, you might have already seen this actually by accident. If you tried to run the file, but you kind of hit Enter early, and you just typed Python and then hit Enter, what you'll see is this mode where it changes.

It'll say the version of Python you have, and then you'll see three different arrows here. This mode is kind of for like debugging or playing around with Python code. It allows you to run Python directly in the command line. So, you'll notice that in here I can actually do some Python things such as, you know, let's say '1+1' it's '2', it does a little bit of math. I can do a 'print' and, you know, put '("Winter is coming.")' hit Enter, and I get the result. Or I can even do something silly like take my name and multiply it by '1000' and just see my name printed out 1000 times. And there was the command for that in case you missed it.

So, again, this is used more for debugging, troubleshooting, or if you don't know exactly the code that you want to be writing, you can sort of play around in here and then actually write the code into your text editor, and save it into a file so that it's easier to run regularly. Or just so you know, you might have accidentally opened this up, so I wanted to introduce people to this. But you should know that in Python interactive mode, we're no longer really in the command line, so the other command line commands won't work anymore.

So, you can't type ‘pwd’ here. You can't do ls. You can't do those things. So, if you accidentally open this and you wanted to go back, you'd have to exit. And there's a few ways of doing this, but if you type 'exit' and hit enter, it'll actually tell you on a Mac, 'exit() or Ctrl-D to exit,' or on a Windows it's 'Ctrl-Z', and then hit Enter. So, I can do any of those options. I can type 'exit()' and now I'm back in the command prompt. So, just so you know that that's an option, but for the most part what we're going to be doing in this class is going to be writing Python code in a file and then running that file directly in the command line, which I'll show you how to do in the next video.

****

Video 13 – Printing in Python

So, now we're going to start out creating our own Python files. So, if you haven't done so already, let's just close out of Atom, and you can even close out of the command line as well. And we'll start out by opening up Atom again, find it. And this is another one, where it might be useful to pin it to your Dock so that it's easy for you to access as well. But when you open Atom, but you don't open a particular file, what you'll see is this, you have empty mode here, where we

can start typing, and I can start typing Python code directly into here if I want. So, we'll go ahead, and I'll introduce the print command.

The first thing that we're going to do in Python, and 'print ( )', it'll automatically put the other parentheses around it, which is kind of convenient. And inside of print, let's type quotation marks and write '(" Winter is coming.")' Now, the first thing you'll notice is that we don't see any colors here, it's just all grayed out text, and why isn't there or why don't we see colors, it's because we actually haven't saved the file yet. So, we haven't saved the file, and so Atom doesn't know what this code is. It doesn't know if it's Python, it doesn't know if it's some other programming language, so it doesn't know how to do the color highlighting.

When we save the file, what we'll do is we'll add the '.py' extension at the end of the file name, and that's what we'll tell Atom, that this is Python code. So, if I go ahead and do File, Save As, and make sure you move to the code folder, so you're saving this file inside the code folder, and call this 'print.py'. You'll actually have to add this '.py' extension at the end; it won't do it for you automatically because it doesn't know what kind of code we're writing. So, if I Save this.

Now, what you should see is that the color highlighting pop-up, pops-pop, you might also see this bar along the left-hand side open up, and that actually shows you the folder that we're currently in, and it actually, it makes it easy for you to jump back and forth between different code files inside of this folder, but we don't need that for now, and you can get rid of it, you can shrink it by clicking this hover-arrow there, if you hover over this bar, click the arrow and it disappears. So now, you should see that the text is highlighted, that the print is blue, and the text is green, and the parentheses are white.

The actual colors don't really matter so much because their different depending on what theme you're using or what text editor you're using. So, don't worry too much about, why is it blue versus green? Although different corresponding parts of code will often be similarly colored, just to make it easier for you to read, and you'll also notice that down at the bottom here, where it says Python, it switched to Python, and that's because the file has the '.py' extension. I could actually click on that, and I could override what code is in here, and it will change the highlighting, you'll notice that by default it's auto detect.

So, it's using that extension to figure out what's inside of here. So, now that you've saved this as 'print.py', can you open the command line and run this file? Pause the video now, give it a shot, see if you can jog your memory as to how to move into here and run this file. So, I'll move this over, and I'll open up the command line, put that over here. And of course, I can't just run 'python print.py', it'll tell me that I can't open the file because there's no such file or directory, and that's because I'm not in the right place with this file, right? Always remember when you open the command line, you have to move into your folder where the code is.

So, I'll first start out by doing 'cd Desktop', 'cd code' as I've remembered at this point where it is. And now, 'ls' I should see I have both of these files here, and now I can run 'python print.py' 'Winter is coming.' Great! Let's make another change. Go back into this file, and now we're going to type print("You know nothing", "Mattan Griffel") put my name, but feel free to replace that with your name if you want to personalize it. So, I have saved this file. Now, go back into the command line and run 'python print.py' or up-arrow and just hit Enter, to redo the last

command, and there you go. Now it says, 'Winter is coming.' and 'You know nothing Mattan Griffel.'

Then, noticed here that we've learned a little bit about how Python works because we've put these two different bits of text inside of this print with a comma in between, but out here, we don't see a comma, the comma actually got replaced by a space. And that's how this print is called a function, the print function works, and we'll talk more about functions in a later class. But, with print, you can put as many things as you want to, after and with commas, and it'll just put spaces in between. So, that's the default of how it works; it is possible that change that all later.

And just as a final note, in Python 2, you could have done 'print' and then gotten rid of these parentheses, and just run it this way, and this would have worked. But I'll run this code now, and show you that, I'm getting an error message, and it says here, we'll talk more about error messages, shortly. It looks a little bit crazy and complicated, but it's actually, you know, pretty easy to understand. But this last part is the actual message, and it says, Missing parentheses in call to 'print'. Did you mean 'print("Winter is coming.")?' and it actually makes a little bit of suggestion because it's such a common problem, that it sort of knows what you might want to do. So, we'll put those back in. And just run this and make sure that you can get it working.

****

Video 14 – Challenge: Lyrics

I've got a little bit of a challenge here just so we can get some practice doing printing. I want you to add prints to “print.py” so that it prints out the lyrics of your favorite song or poem. You don't have to pick the whole thing, maybe just an excerpt or something like that. So, give that a shot now. And when you finish, post your code in the forum as well so we can all get to learn a little bit about each other.

****

Video 15 – Lyrics Challenge Solution, Debugging Errors and Googling, Stack Overflow and Asking a Good Question

So, I've gone ahead and taken my favorite poem, which is E.E. Cummings' "Since Feeling is First," and put it into my print.py file. And I'll go ahead and run it in the command line and see what we get. And it looks like it's not working. I get this error message here. And so, it's a good opportunity to actually take a look at this. Well, we will see error messages pop up a lot as we continue to write Python code, so we need to be able to understand it and know how to parse it and also maybe know how to troubleshoot it. So, the typical error message looks kind of like this, although this is a relatively simple one.

It starts out with File, and then ‘print.py’ So, it just starts by telling you what file this was running when it got the error message. Now, that may be kind of obvious because we just ran this file. But sometimes as code grows, you might have multiple files that are interacting with each other, where you're loading code from one file into another file, and so this can be really helpful for you to know where the error might actually be taking place. And then, in fact, it tells you line 8. So, it tells you where it thinks the error message is. It actually pulls out that

line specifically. So, I have print ("my blood approves") and then there's this little caret here, and that's essentially where it thinks the error is.

Or, more specifically, that's where the error initially popped up. Now, I'm looking at this, and it doesn't look like there's actually a problem with that specific line of code. So, we'll get into that and what might be going on and how we can solve it. The last part of that error message is, of the error is the error message. So, here we have SyntaxError. That's actually the category of error, and there's multiple different kinds of errors, such as, you know, dividing by zero. You can't do that, so there's actually a zero division error. But this is a syntax error and it just says invalid syntax.

So, actually, this error message honestly is not all that helpful, and if I were to look at this line, I would say it doesn't look like anything is actually wrong on this line. So, what might be going on? Now, you might be able to look at this code and figure out why this is happening, but let's say I looked at the code and I couldn't figure it out, how might I then go solve this problem? Well, the first thing that I'm, that I always do and I highly recommend that you do is I just take this entire last part of the error message where it says SyntaxError: invalid syntax, copy that, and just honestly open up Google and search for it.

And it's a good idea to search for that error message and just the word python because sometimes error messages pop up in different programming languages. And one of the resources I want to direct you to is the second one here is stackoverflow.com. So, let's go there because a lot of times when you search for something that's going wrong, you'll end up on the site Stack Overflow. Now, Stack Overflow is this question and answer site where people go and they post code and they ask questions and then other people go and answer those questions just for free, out of the goodness of their heart. So, in this case, what I'm seeing is the actual error message, SyntaxError: invalid syntax for no apparent reason.

And then here is the question. This person posts, I've been trying to get a fix and I can't figure out why this error keeps appearing. And they actually include the code that they have that is producing the error. And down here, they've actually included all of the different code inside of this file that's producing it. Now, the code is obviously different that we're running versus what they're running, but we're getting a similar error. So, maybe there's something that we could extract from this page that helps us figure it out. So, the way this site works is different questions can get upvoted or downvoted and if you scroll down, you'll see different answers, as well as some comments directly on the question.

So, in this case, there's three answers to this question or this problem, and the answers can get upvoted or downvoted. So, it looks like, you know, this is the top answer with 59 upvotes. And in fact, it even has this green checkmark, which means that it's the approved answer by the person who asked the question, that they've indicated that yes, this is what solved it for them. And this is actually an interesting one to kind of read through. If you read through the answer you'll see the problem where it seems to be an error on a line you think is correct, you can often remove or comment the line where the error appears to be and if the error moves to the next line there's two possibilities.

Though we actually haven't even tried this, but you'll find that in this case if I, if I did that, if I removed line 8 it would just jump to the next line which seems somewhat mysterious. So, it gives me an example. So, either both lines have the problem, or it's a problem on the previous

line. So, the example they're giving is here where I have some crazy variable = (1+ and then I don't complete it. And then here I'm creating another variable and setting it equal to something. And this, if you ran this code, it would produce the same error message. And the problem here is that I didn't complete this line here. I said (1 + and then I kind of left it open. And the way that Python works when it's running code it's going from the top of the file to the bottom of the file, from left to right.

So, it's actually fine to leave a line sort of open ended like this, but if this happens, Python's going to get to that, end of that line and then go to the next line and try to find, you know, the part that was missing and see well maybe it's on the next line. And then it'll get to a certain point and realize oh, the thing I'm looking for isn't there. And that's when the error message pops up. So, let's see if we can take a look at our file and see if we can figure out what's going on. So, here we go, on line 8, this is fine, but actually, if we look at the last line of code here on line 6, what you'll notice is there's a parenthesis missing at the end of this line. I was kind of in a hurry, I was sloppy, and I forgot to close the parenthesis.

So, I can do that here, and I'll save the file and run the code, and, you know, make sure that we got it working. And it looks like I have a, you know, an error message still so, you know, what's going on? Did I fix it or not? Well, if you get really specific, if you look at this error message, you'll see that now it says I have an error on line 13. So, I actually did fix the last error message, but now I've produced a new error. And if I look here at line 13, in fact, line 12 had the same problem, by forgetting the parenthesis at the end of line 12. So, I want to go in there and save that and run it. And now it looks like I got the code to run.

And this is actually a very common thing because if you're writing code, if you introduce one, maybe two problems, you might do all this work to fix one of the problems only to get a second error message. And that can be a really frustrating experience. Trust me, I've been through it before, had an error message I didn't know how to solve, and it can feel like I'm not making progress. So, I just wanted you to know that, yeah, sometimes debugging can be a really, sort of, tricky process, but, you know, progress is slow and steady, and it can take a little while.

I have a friend who says 95% of coding is trying to fix broken code because if the code is working, you're not spending a lot of time on it. But it's when you try to do something, and it's not working, that's where you're spending a lot of your time. And Googling is one of those things that a lot of developers are doing. So, just because you find yourself Googling a lot doesn't mean that, you know, you're a bad coder or that you don't know what you're doing. It's very, very common even as an expert because even expert developers are trying to do things that they've never tried to do before or working with different libraries they've never worked with before.

So, just know that it's fine. Another thing you'll notice though that we've got this working is that our empty lines here, lines 7, 14, and 18, didn't end up actually showing up in our code. They're kind of ignored, and that's because Python does ignore empty lines. So, if I did want to actually put the empty lines there, there's a few different ways of doing it. One way might be to just write print and not put anything inside of it, and save that and run it. And now they show up, but again, it might not be the best way, in fact, I'll even end up showing cleaner ways rather than having to write a bunch of different prints over time.

****

Video 16 – Commenting

The next thing I want to introduce is comments and how to write comments inside of code. So, the way that you can add a comment into code that you already have is by putting a pound sign, or a hashtag, or octothorp is one of the technical names for it, in front of something, and then I can write in this case, for example, the name of the poem. This is' # "Since feeling is first" by E.E. Cummings.' So, you'll notice it's grayed out, and that indicates that actually, Python is not going to run this code. It's just for humans to read.

So, a lot of times, you might find at the top of a file some attribution. A person name's the file; they give their name, they give their contact info if you want to reach them. You'll use, you'll see comments being used often to explain what the code below is doing in case it's hard to understand, or even as notes to yourself as, you know, come back to edit this later or to add some functionality. Another way you can write a comment is actually adding it into a line at the end of the line. So, I can go here to line 12, go to the end, and put a hashtag, and say, '# I love this line.' It's one of my favorite in this poem, and that will work. What I can't do is I can't add a comment in the middle of a line.

So, inside of here, it actually won't work if it's inside of text. I'll just see that printed out. So, that doesn't make it a comment. Or, in the beginning of a line, I could add this comment, but it would make the entire line of code a comment. So, actually, there's multiple uses of comments for making notes to other people or to yourself. But comments are also often used for debugging and troubleshooting. You can take either one line of code and comment it out, which just means, you know, put the little comment sign in the beginning of it. And then, when you run the code that line will get ignored.

So it's like, it's like a way if you want to see how it works without that code, rather than just cutting it out, saving it somewhere else if you plan to put it back in later, you're sort of keeping it around for a little bit. You could actually comment out entire sections of your code really easily. So, rather than going, if I want to see what this would be like without, say, this entire section, instead of putting this in front of each one, I can actually just select this entire section of code and then hit, the key is command and then the '?' or '/', or 'Ctrl-/' on a Windows, and just turn the entire thing into a comment instantly.

Then do that again and uncomment it. So, this is actually a really useful shortcut for doing it. If you ever forget, you can go into your Selection, and here we go, under Edit, Toggle Comments. Here's a little shortcut, and it reminds you of how do you do it and that's what it's doing. So, I can now save this, run my code, see what it would do without that line. And then go back here and then uncomment that line, and then run it with the code, and now it's back in.

So, to go back to our earlier troubleshooting problem that we had where we had one line that wasn't working, I could have commented out that line and then see if that got the code running or see if actually, maybe the error was in some other line. So, you can, sort of, use this to drill further down. I highly recommend as just a best practice when you're writing code, write comments in there because it will make it a lot easier for someone else to understand. And also, I promise you that as you start to write more code, even though the code makes sense to you when you're writing it, if you go back to it a week later, a month later, maybe a year later,

you might have no idea what you actually did. So, putting a nice, helpful comment in there can be really useful, even as a reminder to yourself, here's what this line of code does.

****

Video 17 – Variables and Printing Undefined Variables

Next up, I want to introduce something called a variable. So, we'll go ahead and close out 'print.py' for now. We don't need that anymore, or just go to 'File', 'New file' and it'll actually open up both as a tab, so you can switch back and forth easily which, either way, whatever you prefer. And let's call this one, we'll just start by saving it and we'll call this 'variables.py'. I'll introduce the concept of variables, and every new concept I'll introduce I'll save it as a new file, so that way you can go back to that file to kind of refresh how do variables work.

So, we'll start out with a comment here explaining what a variable is. And a variable, '# Variables are just nicknames.' '# they're plain, lowercase words'. So, I like to think of a variable as a box. And actually you're probably familiar with variables already if you've ever done algebra. So, if I told you a=1, b=2, c=a+b, what is c? Well, it's 3 because you're adding those two variables together. And variables in Python work very similarly. It's kind of a box that you can throw stuff into, and then you put a label on the outside of the box, and you can use that label to refer to whatever is in the box later.

So, let's create a few variables and actually play around with this idea. I'll start by creating the name variable, and I'll say 'name = "Mattan Griffel"' put my name in there. You can put your name in there as well. Alright, there's one variable. Let's create another variable, 'orphan_fee = 200'. Okay? So, here's a variable but this one has a number inside of it. Let's create a third variable and we'll call this one 'teddy_bear_fee = 121.80'. So, this is a number as well, just like the last one, but this has some decimals added to the end of it.

Following so far? Let's create a fourth variable and this one is going to be called total. And 'total = orphan_fee + teddy_bear_fee'. And you'll actually notice one of the features of Atom and many text editors is that if you've created a variable and then you start typing something that looks like that variable, it'll show you that, oh this variable already exists, and I could actually go, only type half of it and then hit the Tab key, and it'll automatically complete the whole variable. So, that's definitely a useful feature to have when you're writing code. So, here's a variable and it's just the other two variables added together. And now that we've created these variables, let's print out some of our variables.

So, let's 'print(name, "the total will be", total)'. Once you've done this, save the file and then go into the command line, and run the file and see what you get. So, I'm in the command line. Again, make sure that you're in the right folder where this file actually is before you run it. But once you've done that, you can run 'python', and then the name of the file 'variables.py', and what you should get is something like this. So, I have 'Mattan Griffel the total will be 321.8'.

So, you'll notice already that the numbers got added together, 200 and 121 got saved into the total variable. And when I do printing I can print out text alongside variables and, again, the spaces automatically get added in there whenever I put a comma. So, a variable can contain any number of different kinds of things. Here we have text. We have numbers. And when I say that they're plain lowercase words, what I mean is that the convention in Python is when you're creating a variable to use all lowercase, and then to use, you can't put spaces inside of variable

names directly so instead of a space what you typically find is an underscore. So, 'orphan_fee', 'teddy_underscore_bear_fee ', and so on.

Now this is a convention, and it's not a requirement, meaning if you didn't put underscores, it would still work. If you put capitalization in there it would still work. But it's a convention because a few years ago the Python community got together and they said there's so many different ways that different people are doing things; you've got people not using underscores, people using capitalization, people doing something called snake case or camel case where there's capitalization within underscores. It just got really messy. So, they did this thing where they came up with a style guide.

And actually, if you search for pep8 'p e p' and the number 8, you'll find the style guide for Python code. And so, you actually see here this was introduced in 2001. Then the style guide is this long manual for how Python code should be written. None of it's a requirement but it's just stylistically speaking trying to create a set of standards. So, actually if you search in here for variables, you would end up finding, essentially, something very similar to what I just told you. A little more about underscores, if necessary, all lowercase, and so on and so forth. Just know that this is available here.

I wouldn't necessarily say to go read it all the way through because it's, you know, it can be pretty dry or pretty boring. But if you're wondering where this convention came from, it's from there. That being said, I'm going to introduce throughout this class some of the best practices in terms of the convention. So, I'd definitely recommend following along with what I'm doing because it tends to be the standard here. A little final few notes on creating variables is if you try to use a variable that you haven't created yet, you'll get an error.

When you create a variable it's called variable assignment. You're assigning a variable some value, but you don't really need to worry about that terminology. But here we've, you know, assigned four different variables. But what happens if I'd try to print out the '(subtotal)' which is a variable that I haven't actually created? And if I ran this code I get this error message, and the last line of the error is 'NameError: name 'subtotal' is not defined'. So, you'll see that if I tried to use a variable or maybe there was a typo in here and it's telling me this name 'subtotal' is not defined.

That actually means it's not defined. So, sometimes, there's any number of reasons why this might happen, but if you see this, you'd want to make sure and go back and see did I actually assign this variable? So, I will comment this out for now. Remember, the shortcut is command or 'Ctrl-/', the key with the question mark, and now this should work. But just know, I'll put a little note above this, '# You can't use variables that haven't been created yet' Alright? So, yeah, that's a little bit about variables. And next up we're going to talk about math.

****

Video 18 – Math Symbols

So, let's introduce a little bit of math, which is something you definitely do in Python, and we'll do this by creating a new file. And we'll save this one as 'math.py'. File, Save As, 'math.py'. Okay, so I'll start out by writing a little bit of a comment about what we do. '# In Python you can do math using the common symbols or the following symbols:' and you know, you can do the basic one such as '# + addition' '# - subtraction' '# * multiplication', of course, '# - division'.

So, those ones we know. If you want to do an exponent, that's actually a little different from what you might be used to if you've been doing Excel, it's '# ** exponent (not ^)' like it is in some other languages. You can also do '# > greater than', can do '# >= greater than or equal to', which is just the greater than symbol with an equal sign at the end, you can do '# < less than', '# <= less than or equal to', and et cetera. And so forth, there's many more kinds of things you can do, you can do modulo, and other sort of things in there. Those are some of the basics. So, now let's try it out a little bit. So, let's 'print("What is the answer to life, the universe, and everything?")'; let's make this text a little smaller so that we can start to see.

Might have to just expand this, then we'll put a comma and here I'm going to do ', 40 + 30 - 7 * 2 / 3)'. So, directly in here, and I'm actually only adding spaces to make it easier to read, it's a convention, but it's not a requirement. This would work without the spaces after and before the plus sign. So, let's go ahead and save this actually, let's save this and then let's add another two lines of code. 'print("Is it true that 5 * 2 > 3 * 4?")'. And then let's actually print out, inside of 'print(5 * 2 > 3 * 4)'. Okay, now, we can print this out, so let's move this over.

And run 'python math.py'. Okay, well the first thing is I get, 'What is the answer to life, the universe, and everything?' and it's saying '65.33333' repeating, which if you've ever read, "The Hitchhiker's Guide to the Galaxy", you know, it's not right, the answer should be 42. So, we'll have to go in there and see what's going on, in a bit. You also see, 'Is it true that 5 * 2 > 3 * 4?' and then 'False'. And what's happening is actually, Python is doing the math inside of this print, and then it's printing out the result in both of these cases.

Although you'll notice that on line 15, where we have the math inside of the quotation marks inside of the text, it's not doing the math for us automatically, it's just printing out the full result. But on line 16, it is doing the math, and it's just printing out 'False'. So, what's going on over here? Well, the problem here actually is that I forgot my order of operations. So, Python follows standard order of operations, you know, PEMDAS. But, what I really wanted to do is put parentheses around this part here, so '40 + 30 - 7', easiest way to put parentheses around something in Python is to just select the whole thing.

Start typing the first parentheses, and it'll actually automatically put the second set at the end. So, it won't remove it; it'll just wrap it in parentheses. So, let's try saving this and running the code, and there we go. So, now I got the right answer. 'What is the answer to life the universe, and everything?', it's '42'. You notice something a little bit weird here, which is I'm getting '42.0' instead of just '42', it's a round number. So, what's going on? Well, in order to explain that, I need to dive a little bit deeper, peel back a layer of the onion, and talk about something called integers and floats, which I'll do in the next video.

****

Video 19 – Math Symbols: Integers and Floats - Part 1

So, in the last video, what we saw was that when we ran this Python code here, especially this last part, what we got, as a result, was the number '42.0', and the question was why is it '42.0', instead of just 42? To the answer that, I need to talk about the fact that there're in Python at least two different kinds of numbers. There are whole numbers, and there are decimals. In Python language, you've got integers, which are whole numbers, and then you have floats, which are decimals. So, in order to sort of dive deeper into this a little bit, I'm actually going

to have us open the Python interactive shell. Intentionally, we'll go into the command line and type just 'python' and hit Enter.

And what this will let us do is just play around with Python code in here, and see what the result is without having to print it out and save it to a file, and to keep running the file. So, just sort of shortens down our cycle to see the result. So, in here you can do, you know, some pretty basic stuff such as '1+1', for example, which we already saw, and then you get the answer. If I put in '42' and hit Enter, I get back '42'. So, it's kind of like call and response; it just gives you back whatever you get it for the most part. But if I put '42.0', what I get back is '42.0'.

And, in fact, if I put '42.00000" and keep putting zeros, I just get back '42.0', and this is a little bit of a hint to us that, there's two kinds of numbers in Python, 42 and 42.0. As I mentioned 42 is called an integer, whereas 42.0 is called a float, and the reason why they're different is because even though you and I might look at these two versions of the same number, and realize that they're the same, to a computer they're different because it has to save a certain amount of space to store a number, and if you have a number with decimal points afterwards, it kind of has to save more space because every single digit takes up additional space.

So, it's kind of storing a few extra little bits of memory, in case there is something that changes after that point, maybe it's 0.1 or 0.2, so it's setting that aside early. In other programming languages, you have to be really specific when you create a variable, what kind of number is going to go inside of it, is that an integer, is it a float? In Python, you can kind of be a little looser with it, but it will often kind of make an assumption for you about what you actually want, so that's why there's a difference. Whenever you are doing division in Python, the results of what you get back will always be a float, which is why when we do, in this case, if I gave you, for example, a '126 / 3', I'm getting back '42.0', and that's essentially what's going on over here.

This is actually interestingly different from what used to happen in Python 2. In Python 2, when you do division of two whole numbers, what you always get back was a whole number, an integer. But then you would end up with some weird situations such as if you did one divided by two in Python 2, the answer is actually '0', whereas in Python three it's '0.5', which is, you know, the actual answer. Of course, you can imagine how it might be confusing to people, and lead to some sorts of different errors. In Python 2, the thinking was if you divide two integers by each other, and you don't want an integer back, you better specify that. You better be really explicit about it. Whereas they ended up deciding, we kind of want to change the way the language works, so to avoid possible problems later on.

So now, division always produces a float whenever you do it inside of Python. Now, you can convert from one to the other pretty easily. If I take a whole number like 42, and I want to turn it into a float, I can just actually use the float function. So, the word float with parentheses, and then the number 42 inside of it, will give you back 42.0, if you want to convert a float into an integer, what would you do? Well, it's not integer; it's actually just 'int'. So, if I did 'int(42.0)' and, you know, parentheses around the float, what I get back is just '42'.

Now, you should be aware that the int function doesn't do rounding exactly, it doesn't round to the nearest integer, it'll just cut off anything after the decimal point. So, if I did 'int(10.58)' and hit Enter, what I get I back is '10' not 11. There is a different function for doing rounding, kind of works similarly, but what the int function does is it just basically says, only give me

back whatever's before the decimal point. So, we've learned a little bit using the interactive Python mode here, about integers versus floats, and also how you can convert from one to the other.

So, let's exit out of here, and then apply what we've learned to our file, to get it to run the way that we want it to run. Do you remember how to exit out of Python interactive mode? Well, in case you forget, you can always just type 'exit' and hit Enter, and you'll get a little bit of a reminder. So, this time I'll use, for me it's Control-D, and now I'm back in the command line, and I need to do that. So, in the next video, let's apply what we've learned about integers versus floats to get our file to do what we want it to do.

****

Video 20 – Math Symbols: Integers and Floats - Part 2

How can we take what we just learned about integers and floats, and converting one to the other and vice versa, to getting our file to display the number the way we want it. Now, before when we ran it, you're getting '42.0' as the answer 'to life, the universe, and everything'. What if just wanted 42? Well, we can use the int function that we had before. So, going back to this file, first of all, this one line is getting quite long. So, one way that you might deal with a really long line of code as you see, I keep having to scroll back and forth, all the way to the right, is to actually, take it out, and put it into a variable, and then use that variable. You'll actually notice, interestingly in this file, there's this really subtle line in Atom going down this side.

This happens at the 30 character mark. So, once you have 30 characters on a line of code, you see this line going down. The convention in Python and in a lot of different coding language generally is that, no line of code should be more than 80 characters long, and that is to avoid having to scroll all the way to the right, or you know, trying to do way too much in one code. So, an easy way to deal with this is to take this out, just copy it, cut it, put it up here, and let's put this into a little box, which is a variable, and we'll call it 'answer'. And then, you can go here and just plug, insert directly, and that makes our lines a little shorter, but it does add more lines to the code, which is not always bad.

So, now we've got answer here, and the question is, how do we make answer an integer? There's actually two ways of doing this. One is I can just put int around this part right here, so I can do it directly in here, just 'int(answer)', and just make sure at this point that you have the other set of matching parentheses because it might look like it's matching now, but I also, you know, need this one set of parentheses from print. So, you actually need to have two. It can get very confusing once you start writing a lot of code that's doing different things at once because we have multiple sets of parentheses doing different things. For example, up here we had parentheses around the math equation. That was to say do this first, but also whenever you write a function like print or int, it also need a set of parentheses, and sometimes it can get hard to keep track of how many parentheses. So, we can try it this way, save that, and now let's run our code, 'python math.py', and here we go. Now, I'm seeing that 'the answer to life, the universe, and everything' is '42'. Another place I could've done this was up here before actually saving the answer into the variable.

And then answer would've actually contained an integer as opposed to right now, the way the code is written, answer has a float, and then we're converting it to an integer, inside of this

print. At the end of the day, the result of what we see is the same, but depending on what your code is doing, you might want to do one way or the other way. If you plan to reuse 'answer' multiple times inside of our code, and you want it to be an integer, it might make sense to do this conversion before saving it to that integer. But just posing, sort of, both of this options to you, just so that you know both of them.

****

Video 21 – Strings

Next up I want to introduce text and something called a string in Python which is basically just Python speak for text. So we can create a new file, and if you want and get rid of these other tabs kind of clean up our working environment, and we'll save this file as strings dot p-y. Okay so we'll start with a comment here that strings are text surrounded by quotes. And I'll indicate here that both single and double quotes can be used. So far we've been seeing mostly double-quotes, but I'll show you a reason why it would make sense to work with just single quotes and vice versa.

We'll start out by creating a new variable with some text. So, the variable is going to be ‘kanye_quote’. And we'll say, just for some practice, we'll use single quotes. So, I can show you that either way it works. So, kanye_quote should contain the following text, ‘kanye_quote = ‘My greatest in in life is that I will never be able to see myself perform live’, something that Kanye West actually said. And now I can print out kanye_quote, save the file, and run it, and make sure that it's getting what we want. ‘python strings.py’. Okay, so what we're seeing here is the text obviously printed out.

Now, if I wanted to, I could break this out onto multiple lines. The problem with that is I would end up, you know, you have this area that a quotation mark or a quote can't span onto multiple lines because it expects you to close the quotations on the same line because. There is a way of dealing with this and you can do it by putting the slash at the end. And that's one way of telling Python, I actually plan to continue the quote down to the next line. So, it actually might be better to do it earlier up here. That way we can see everything without having to scroll to the right.

So, now I put this slash which you can do this slash by hitting the key above the return key on most computers. If you have an international keyboard you might have to go around searching for it. And now if I run this you'll see that I get the same result even though it's on multiple lines. So that's one way of doing text across multiple lines. So, this is called a string in Python. Strings and texts are basically interchangeable in terms of the language that we use.

The reason why it's called a string has actually because if you go back to older programming languages text was just a lot of characters strung together into like a list. And so, the idea of a string is that it's just stringing together multiple characters into text. Again, single quotes or double quotes, either one can be used. It doesn't matter as long as you have the same one on both sides. So, if you start a string with a single quote, you have to end it with a single quote and vice versa. And so, why would it matter which one you use? Why are there multiple options? I’ll give you one example.

Let's create another quote, and we'll call this one Hamilton quote. And this quote is going to say “Well, the word got around, they said” and then we will put this on another line. Actually,

put “they said” part on an airline. “, this kid is insane man”, and another set of quotes, and we'll print this out. Now obviously it looks like we're going to have a problem. The first hint about this as the color coding of this part. Here. We see this is red, and that's because this first set of double-quotes is matching this second set of double-quotes which we wanted to be inside of our text, not to say, end the first string. And it's the way it's parsing this or the way Python is reading this is that this is a string, this is Python code outside of a string, and then this is another string, which is clearly wrong.

And if I tried to run this code of course I would get an error, the syntax error. So in this case if I have a string and I want to use quotation marks inside of it. And that's one pretty obvious. use case for using a different set of quotation marks. The easiest way to solve this is to just use single quotes on the outside Just make sure that we have matching single quotes. Sometimes, when you enter single or double quotes in the text editor it'll automatically put a second set. So, you want to be careful to make sure you have the right number.

So, this will work save this file and now I run it. There we go. So, if I use single quotes I can put double quotes inside my string and vice versa. But occasionally you run into a situation where you do need to use both single and double quotes inside of a string. And so there's another way of getting around this. And that is by doing something called escaping the quotes and escaping means just basically putting the same character that we had before -- this backslash. It's a kind of a magical character inside of Python, inside of strings. I can do a few different things. And one of them is that if I put this slash in front of any set of quotation marks inside of a string, and I try to print it, you'll see it doesn't actually show up.It gets ignored. But what it does is it tells Python don't worry about that quotation mark it it's not there to end the string. It actually should be printed out as a quotation mark. So, if I was escaping this quote by putting those slashes in front of it, then I could use double quotes on the outside here. And it would print just fine just like this. So, you see how that works. You can do the same thing with single quotes as well.

Now, this magical character also lets you do a few other pretty special things. So I'll show you an example of this. Let's say we wanted to print something but we wanted it to take up multiple lines.

And I'll show you that if you do print and inside of here I’ll write, “This is how you put a.” And then, after this a space we're going to put the escape character but with the letter N. And then, we'll actually just continue as normal. So, no space here. But this is going to be a line break, and we'll do another set after the space backslash n in a string. And what's going to happen when I print this out, you might guess at this point that it's actually going to replace that \ n with what's called a new line New line break so at that point exactly it's actually going to jump down a line.

This \n is actually sands for a new line break. And even though you see it inside of this text inside of the string when you print it out, what the print function does is it just converts that into a command that basically says move the next text onto the next line. There's one other thing you can do, which is you can use tabs as well. So, instead of \n you can do a \t, and that would then be displayed as a tab. So, ‘ print(“ This is how you \tinsert a tab”)’.

And the reason I'm not putting a space after either one of these is that the space would actually end up being printed in there. So, I would have to spaces in a row which is not exactly what I

want. So, even though it's a little harder to read it produces the exact output that I'm looking for. So, now I run this and you'll see, in this case, this is how you and then you see a tab inserted into there. So it's just something to be aware of. There's also a few other things you can do with strings, different functions you can use to manipulate strings that I'll introduce in the next video.

****

Video 22 – String Functions

So, I've shown you a few different things about strings and how you can create them. But I want to show you things you can do to strings; specifically, you can run functions on them to change them or manipulate them in different kinds of ways. What's at the bottom of this file 'print("Some string functions:")', to just sort of indicate. So, let's take the string '("Stop yelling")', for example. So, right now, this would just print out the text 'Stop yelling', but at any point after a string, I can actually put a '.' and then follow it up with something like 'upper' and then put '()' afterwards.

So, this is a kind of function, we've already seen a few different functions, and we'll get deeper into exactly how functions work and how to create them in a later video, but I will start to introduce different functions along the way. We had 'print, int, float', this is 'upper', but when you're using a string function, you are basically attaching it to the end of a string, and putting a '.' in between. So, this '.upper' should actually print out stop yelling, but in all caps. So, let's give this a shot, and there we see the last line here, 'Some string functions: STOP YELLING'.

Similarly, I can do 'print' I did this in all caps, '("STOP YELLING".lower())', it will just lowercase it all. So, there we go, "upper" and "lower", those are two different very simple kind of functions, and they have all sorts of different use cases that we'll see later on, and there are other functions as well. There's capitalization functions such as capitalize, title, which does title case, you've got the bizarre swap case, which just converts capital to lowercase and vice versa. And then you have some other really useful ones. There's '.replace', which actually allows you to replace something in a string with something else, strip, and a bunch of other string functions that I'll let you explore on your own, and that will start to introduce more of when we need to as this class evolves.

****

Video 23 – Writing Variables Within Strings: Using Squiggly Brackets f{}

Now that we've talked a little more about strings, I kind of want to introduce to you a different way to put a variable directly inside of a string. So, let's leave this file aside for now, and let's actually go back to the previous ‘variables.py’ file that we created. So, one way to do that would be, you can click this little arrow here to open up the sidebar, and then just go back to ‘variables.py’ and then close it, or you can just go to 'File', 'Open', and then open the file directly. So, now that we're back here, you'll notice the way that we printed out line nine was we actually printed a variable, a string, and then another variable.

But there's a different way to do this. We can also print out a string, but put an 'f' in front of it. So, here I have an 'print(f"")', and I'll show you that I can put a variable directly into a string to get a certain result. So, here I'm going to put '{}'. Now, curly brackets are like square

brackets, but, you know, with a little point, and you use them by doing Shift and then the same key as the square brackets. You might have to find this on your keyboard, it's not something that we use all that often. But, inside of these curly brackets, I can put a variable such as '{name}', which already I have created in this file. And then, here I'll write 'the total will be', and then I put another set of '{total}'.

So, let's save this, and run it and see what the difference is between this print on line nine and this print here on line 14. 'python variables.py', and you see that actually, the result here is exactly the same, it's two different ways of doing the same thing. How it actually works is slightly different, on line nine we're actually printing three different things, and the print function by default is putting spaces in between them, whereas on line 14 we're printing one thing, and it's a string, but we are putting the variable directly into the string. Now, why is it useful to be able to do it this way as opposed to the other way? Well, the second method actually allows for a little bit more customization.

For one thing, now I'm actually putting the spaces directly in here, and that means that if I wanted say, a '${total}', I can just put it right in there and run this, and I would see the '$'. Whereas, if I tried to put the '$' in the first version of how we did print, it would actually put in a space after it, which it's possible to override that, but it gets a little bit more complicated. So, the second way is actually a little bit better for customizing exactly how you want it to be displayed. So, I got rid of that '$' there, as before. But another thing that you can do is really customize.

So, for example, you see how this number here, '321.8', it's only giving me one decimal place, and that's because the number only has one decimal. But when you're working with currencies. Typically, you want to show two decimal points out because that's how cents work. So, how can you force Python to display two decimal points? Well, there's actually an easy way to do this using this second method of printing it out. So, after this 'total', the way you do it is you put a ':.2f'. So, it's easy, it's not necessarily, not cryptic, but I'll walk you through this in a second. But I'll will show you that if I run this, now you'll see that I'm getting '321.80'.

What this is basically doing is after a variable you put a colon, and then, after that, you can specify some customizations as to exactly how you want this to look. In the case of a number '.2f' means, display this as a float with two decimal points. If I change this to '.3', I'll get three decimal points, '.4', so on and so forth. So, even though there's different customizations, this is one of the common ones, because if you're dealing with currencies, you want to do this often. So, this is one that's sort of worth memorizing and learning how to do.

The other thing is if you're wondering what the 'f' does? Well, I'd actually recommend, trying getting rid of it running the code, and seeing what happens. A lot of times, you might be following along, and you have a question in your mind, what happens if you make this change? I always recommend trying it and seeing what happens. Because it's something I might cover or maybe I won't cover it, and Python is so vast, and there's so much complexity that by experimenting with it, it's probably the best way for you to learn what happens.

So, in this case, let's just get rid of the 'f', try running it, and notice that you won't actually see the variables get plugged in there, you'll actually see literally, '{name}', and then so on and so forth. So, the point of the 'f' is what it lets you do is put '{}' anywhere inside of a string, and basically, Python will run that code inside of there. So, even though I'm putting a variable name

in here, I could actually do some sort of calculation inside of here, you know, some math such as, you know, 1+1 or even add together orphan fee and teddy bear fee, directly into here.

And Python will always run the code inside of the '{}' first before putting it into the string and then printing it out. So, that's essentially what the 'f' does, and this was actually a new feature introduced into Python 3.7. Before that there was a different way to do this, but it was a little more complicated. So, at this point, a lot of people, a lot of students that I work with will end up putting 'f's' in front of all of the strings that they do. It's not really necessary unless you are putting a variable into it or you're using '{}' to do some Python stuff inside of it. But again, it won't hurt. I mean it'll still work to have an 'f' in front of every string, just know that it's not really necessary, I call it 'f' string fever, you don't really need to know how to do it. So, that's how you put a variable directly into a string, and there's actually another reason why we need to know how to do it this way, and that's if we wanted to save this string, the result of this directly into another variable, the only way to do it would be, to do it by putting an 'f' in front of the string, and that'll be useful later on.

****

Video 24 – Getting User Input

So, the last Python concept I want to introduce you to this week is how to get input from the user to use inside of your script. So, we'll create a new file here with File, New. You can close out the other ones to just clear up some space. And let's call this one ‘input.py’ So, input is a function in Python, and I can call it by 'input( )', and then in here let's write a question such as, '("What's your name?")' and question mark. We'll save this file and I'll just start out by running this and building this file slowly over time, 'python input.py'.

What you'll notice is instead of running something and then going back to the command line prompt before where we can type other commands is, it puts whatever string we put into that function, writes it here, and then kind of waits. And what it's doing is it's waiting for us to give it a response. So, I could, say, put in my name as 'Mattan', and you'll notice there's not really a space here so, sometimes I like to actually put a space directly after the string here just to make it a little easier. But I put in 'Mattan', I hit Enter, and now I'm back at the command line again.

So, I gave it my input but nothing happened to the input. It just disappeared. So, this is where variables come in. So, I can create a variable here and call this name and set that equal to input. Now, again, the concept of a variable as a box I like to use here. I'm getting an input from a user, I don't know what it's going to be, but I'm putting it into this box called name. It's like moving and, you know, you write on the side of the box where, you know, these are your kitchen things and that way you know what's inside of the box based on what you wrote on the outside. A variable kind of works the same way.

So, it can be helpful to name something based on what people can expect to find in there. And so, now that I have this input, I can just 'print' out, you know, let's practice our f-strings. So, I can put an '(f "Hi ")', do you remember how I would actually put 'name' directly into there, given that this is a string with an f in front of it? That's right. I want to use curly brackets. So, now I can have '{name}' and I can put an exclamation point at the end, and I can try this out. My

name. And now I'm getting back, 'Hi Mattan!' So, this is cool because now we're kind of expanding the functionality of the things that our files can do like getting different inputs.

Let's add in a second input here. Let's also ask for age. So, I can ask 'input("How old are you? ")', and put a space. And so, now I should be able to also put this variable into here and it will say, 'Hi Mattan!' 'You're {age} years old.'. So, I can run this here, say my name is Mattan, and I am 31 years old, and it just prints it back for me. So, pretty simple so far, right? But, what if we want this to do a little bit of work for us, essentially. Rather than printing out just someone's age, what if we want this to tell them how old they are in dog years? Well, now that we have our age variable, rather than printing it out directly, I can actually do a little bit of math.

I can take age, multiply it by seven to calculate someone's age in dog years. So, let's create a new variable, call it 'age_in_dog_years = age*7', and now, I'm going to put a little space here. Sometimes, I use empty lines just to separate parts of the code to make it more readable. And I typically like to do that before I print a statement. So, here we'll say, '("Hi {name}! You're ' and then instead of printing out age, we'll do '{age_in_dog_years}' and then we'll say that we are that 'in dog years. Woof!")' at the end. So, I'll run this here. So, my name is Mattan, and I'm 31, and , Well, it says, 'Hi Mattan! You're 31313131313131 in dog years. Woof!'

So, what's going on here? See if you can take a look and figure out what you think might be happening. Well, what's happening is that all user input, when you use the input function, comes in as a string automatically, by default. And you can't do math directly on a string in Python. If you take a string and you multiply it by a number, it just repeats that string that number of times. So, even though I was giving it the number 31, it was treating it like a string, and then multiplying it by seven, which is why I have the number 31 printed seven times.

So, can you recall back to that lesson we did about converting strings or converting floats into integers and integers into floats and see if you can come up with a creative way to solve this problem.

Well, what you'd want to do is use either one of those functions to convert the string into either a float or an integer. In this case, I'm getting a string and I'll convert it into an integer. So, I can do that in a number of different ways. Probably, where I'd want to do that is right here, around this input.

So, I'll actually put 'int( )' outside of the input. So, actually what's going to happen here is that I'm going to ask a user for how old they are. It's going to get a number with a string inside of it. And then it's going to convert that string into a number before it saves it into the age variable. So, age is going to now contain a whole number or an integer. So, let's give this a shot. And there we go. Looks like it's working.

Alright, now I could have done it here as well. I could have converted this age into an integer first before multiplying it. There's any number of ways you can solve a lot of problems. But if I had done it here, then age still would have had a string inside of it, and if I plan to use that variable later on, I'd have to do the conversion again.

So, it might take a little bit of thinking as to what's the right place to do a conversion from one type, like an integer, to a float, and vice versa. And also, in this case I'm using an integer because I'm expecting the user to give me a whole number. But if I was expecting them to give me a float, like, for example, I was asking how much does your bill come out to so that I can

calculate a tip for you, then I might need to convert it into a float instead. So, just something to be aware of. Now that we're able to get input from people, we can actually start writing much more complex scripts that can ask you for different things and do all sorts of calculations for you that you might find useful.

****

Video 25 – Happy Hour Redux

Okay, so I want to take a second to talk about a really interesting and kind of weird bug that I accidentally introduced the first time I was making this course. Going back, way back to our 'happy_hour.py' file. At this point, not everyone, but some people, some of you taking this class when you run 'happy_hour' at this point, you'll get an error. So, 'python happy_hour.py', and your error might look something like what I see right here. It's a very mysterious error because we haven't changed this file at all, right?

Since we ran it at the last time at the beginning of this module, so why is it not working right now? And then, to make it even more mysterious, take a look at what is actually going on in this error message. At the top, do you see how it says, 'What is the answer to life, the universe, and everything?' 'Is it true that 5 * 2 > 3 * 4?' than 3 times 4? And 'False'. Does that seem familiar? It's actually what we have inside of our 'math.py file', and then, we get this cryptic error, 'Import error: cannot import name 'log' from 'math' '.

Now, not everyone will get this error, it kind of looks like is based on what computer you're using, but this is a really good situation if you do have this error message for you to try to act like a detective and see if you can figure it out, and follow the troubleshooting advice that I gave before. So, the way that you would do this, you could try to pause this video now and solve it yourself, or you can see how I would solve it, which is again just literally copying and pasting the error message and searching it on Google.

Now, this last part is specific to me, it's got my 'Users'; my 'username', so I would probably just copy this part of the error message, 'Import error: cannot import name 'log' from 'math' '. I don't know what that means, but I'll go into Chrome and just you know search for that, and sure enough the first result is a Stack Overflow page where someone posted they got this error, and then the answer says is it possible that you have a file named 'math.py' in the same directory as the program you're running? Which turns out, we do.

Because we created the file called 'math.py', and what it explains is that Python tries to import this before the math module. So, this is kind of confusing to understand what that means, but the sort of the basic overview is that Python actually consists of a bunch of different files and modules that make up the actual programming language Python. It turns out that one of them is called math, and by creating this file in our code folder named 'math.py', we have accidentally overridden this core part of Python, it's meaning, it's basically loading that instead of this other part of Python which is based on math.

This will make more sense when we get into module three and talk about importing external libraries and the Python standard library, but the takeaway is the way to fix this would be to just rename this file or something. So, you know, 'math2.py' should be good enough, and just doing that, and rerunning this code, you'll see that now the file works again, so that was definitely the culprit, and you should just sort of be aware that there are some things that you

shouldn't name files; 'math' is one of them, 'numbers.py' is another one, these other ones seem to work just fine, but every once in a while, you kind of accidentally step on the part of the Python internals, and so if you've created a file, and then things start going wrong, then maybe try renaming that file, you know, or just searching Google like I did and you probably will find the solution.

****

Video 26 – Tip Calculator Challenge Setup

I've got a challenge here that incorporates a lot of the different things we learned, to sort of test and push your limits. It's called the Tip calculator challenge, and what I want you to do is write a script called 'Tip.py', that asks you, how much your bill came out to at a restaurant, and then recommend three standard tip amounts. So, like at the bottom of a receipt, you know, how sometimes it says, if you wanted to type 18%, here's what that would be. 20%, and 25%. And I want you to include comments in your code. So, there's many ways of solving this, see what you come up with, and then in the next video, I'll walk you through how I might solve this, and the different intricacies of doing this.

****

Video 27 – Tip Calculator Challenge Solution

So, here's my first solution to this problem. I've created 'tip.py', and what I'm doing is asking for 'How much was your bill?', and calling that 'total', and then converting it to an integer. We'll talk about that when we actually run this, and I've created three different variables here, 'tip_18', 'tip_20', and 'tip_25', where I'm calculating what that total amount would be for the tip amount, and then I'm printing them all out below. So, 'print("18%: ")' and then putting the actual variable there could, I have could have used an 'f' string as well. And again, there's many different ways of solving this, so here's just one sort of first pass.

So, I'll run this say, 'python tip.dy' say, 'How much my bill?', '10' dollars, and there we go. Now, it's still not perfect, right? I mean, there's a bunch of problems with us, for one, you see this number '1.799999'. That's actually a really sort of fascinating thing around floats, and when you do math with floats is floats actually, can't be represented perfectly in any computer programming language, or by computers in general because it actually, how they store numbers, interestingly enough. So, you sometimes get these weird-looking long digits, and one solution is to just round them, and then it'll show it right. But remember, I showed you a way of putting a variable into a string so that you always got to decimal points, so that might be one way to solve this.

So, I could just take, you know, instead of just using the print function, where you could put ',' between things, I can instead plug this directly in by putting an f in front using "{}", putting the variable in there, and then doing ':.2f' to round it out to two decimal points, and I can do the same with all the others as well. So, put that in there, '.2f', and then '{tip_25:.2f}'. We'll run this now and show that. Okay, that looks like it's working, and you probably also would want to put a '$' in front of this as well.

But a second thing about this is you might have noticed that if you tried to put in like a full amount like '10.58', for example, it won't work. I'll get an error. And if I google this error, what

I would figure out is that actually, Python has trouble converting a string that has a float inside of it directly into an integer, for some reason, I'm actually not sure why. So, one way to solve this is to convert it into a float instead of trying to convert it directly into an integer. And, in fact, given the fact that we are asking someone for their bill amount, and were probably expecting that they might give us cents, it makes sense to use float instead.

So, now this will actually work with, you know, a float like '10.58' or it'll just work if I put in, you know, integer, also it'll just convert it into a float, so there won't be a problem there. Now, there's additional ways that we might be able to customize this like, what if we want the user to be able to give us a bill amount with a '$' in it. Well, there's a function called strip that we can use to actually strip out characters from text, such as "$", if it's there or if it's not there it doesn't do anything, different more advanced functionality, but that's where I'll leave this for now. And we'll talk about the homework, and kind of wrap up the week.

****

Video 28 – Module Debrief and Introduction to Home Study

So, we've covered a lot in this module, we've gone through the command line basics, we've talked about different ways of running Python code, print, comments, variables, math, strings, getting inputs. So, in order to, kind of, refresh ourselves and get a little bit of practice, I've given you an assignment for this week. And the assignment is a file called 'Assignment1.py', and in it, I've listed a set of about eight prompts and challenges as comments. So, the challenges are as simple as, you know, print out the following text you've reached, and then you put in your name, "I'm not available right now," et cetera.

So, the way that you solve this is first, fill out line to where you put in your name, completed by, and then below each prompt write a line of Python code that sort of solves that prompt, and going all the way down in increasing complexity.

If you do find yourself having trouble at any particular point, just know that, oh! Maybe I need to go back and rewatch that set of videos, just to, kind of, clarify something that might be going wrong, and then submit this file with your name, and with the code that's working, and that'll be it for this week.

----------**END**----------