Module 3: Video Transcripts

Video 1 – Dictionaries

This week we're going to start by talking about lists and the limitations of lists, and I'm going to introduce a new data type called a dictionary; that's going to be really useful in getting around some of those limitations. So, as we saw in the last weeks' videos, a list is really useful for keeping track of different kinds of things in Python, and they allow you to loop over them, but there's some things that they're not so good for keeping track of or representing, as I'll show you in a moment. We're going to start out again with, you know, blank computer, and open up our command line.

Make sure we've 'cd' into our code folder. And here's a little trick that I like to use sometimes, I can just do 'cd ' and then drag the folder directly onto there, and then directly to my folder. And I'll open up the text editor. That way, I have both these ready here, alongside each other, and that file, and close this out. Okay, so let's start out with a new file, and I'm going to go ahead and name this one 'dictionaries.py'. Based on the new kind of object, I'm going to tell you that in Python called a dictionary.

So, let's say we were talking about a stock, like Microsoft, right? And I wanted to in Python, somehow, keep track of Microsoft's stock ticker, their, the index that they're trading on, the company name, stock price days, things like that. One way they might think based on learning about lists of doing that would be to create a list. So, I could say 'stock =' and then inside of a list put some strings and say okay it's '["Microsoft"," And then the second thing that's going to be the ticker which is '"MSFT"," And then the third it's trading on the '"NASDAQ"]' So, I can keep track of that in there.

Now, this is not necessarily the best thing to use a list for because, for one; I have to remember how to get to one specific thing in there, like, let's say, I know I have stock with this list. I wanted to just pull out the ticker. It have to remember that it's in position number 2, which I would use '[1]' to access. And then if I wanted to add in, let's say the last open and close price, I could add in say, '108.25, 106.03]' but immediately, I've lost all the context for this. So, it's easy for me to lose track of, wait, was '108.25' the opening price or the close price?

Do you see what I'm getting at here? A basic problem with a list is that none of the elements are labeled, which are fine if everything in the list is the same kind of thing. But, as soon as you put similar things like this and different things and there, it becomes easy to lose track. So, this is where something called a dictionary actually comes into play. So, instead of doing this, I would create a dictionary and that looks like follows. Alright, '# Dictionaries let you label values using strings'; they're basically lists where you can label each of the values. So, I'll create the same variable called 'stock =', but instead of using '[]', we're going to use '{}'. we've seen '{}' before inside of 'f' strings.

But in this case, this indicates that we're creating a dictionary. So, it's another example of the same symbol being used for different kinds of things. So, don't get confused. Now, inside of here what I'm going to do is first put in a string, such as '{"name":, and that's going to be my label for something called a value. And the value here is going to be ' "Microsoft"}' Okay, so

this is called a ‘key:value’ pair and a dictionary allows you to store multiple of these. So, let's add another one after this ‘key:value pair’, I'll put a ',' and the next one is going to be another string.

And this one's going to be '"ticker"' another ':' and then another string, '"MSFT"}' So, there we go another ‘key:value’ pair. And then, finally, I'll add a third one, which is '"index": "NASDAQ"}' So, this is one area where the space I put after the ':' doesn't really matter. It's more of a convention to make it easier to read, and in fact, I could put all of these tickers, all these key:value pairs on different lines to make it easier to read. So, this is a very basic dictionary. The nice thing about this is now I know, oh, this is the name, this is the ticker, and this is the index, so I've now essentially labeled it.

So, again, the '{ }', that's how we create a dictionary, and the dictionary consists of multiple of these pairs. And this is, the one on the left is called a key, the one on the right is called a value. Now, the key is essentially almost always going to be a string. It's possible to use a number as well, but we don't see that done very often in Python. It would basically just be a list in that case. But we're going to use a string for the keys, whereas the value in these cases they're all strings, but they could be anything.

The value could be a number, could be a list, could even be another dictionary, which makes a dictionary a powerful way of organizing data, because you can go multiple levels deep in terms of hierarchy, and as it turns out dictionaries are going to be used very often inside of Python, and we'll see that in the lesson on web scraping and on APIs etc. Now, to access something out of your dictionary, the actual syntax is very similar to accessing something out of a list. You use the variable name in this case, any use '[]'. So, even though this is a dictionary, and not a list, we're still using '[]' after the variable name to get something out, but instead of a number in here, can you guess what we're going to use to get something out?

We're going to use a key. So, remember these are ‘key:value’ pairs. What we're looking for is a certain value. So, either we want to get the company name "Microsoft", or the ticker "MSFT", or the index "NASDAQ". So, if I wanted to get the name I would put the actual string of the key in here, 'stock ["name"]' This is going to return "Microsoft" , I could also get the 'stock ["ticker"]' and that's going to return "MSFT". So, if I printed these, I should see "Microsoft" and "MSFT".

Let's go ahead and save this, run my code. And there we go, that's exactly what I was expecting. So, if I wanted to, I could put this inside of an 'f' string and sort of write a longer, more complex statement. So, instead of just 'print (stock ["name"])' and '(stock ["ticker")' I'll 'print(f ', now inside of here, I'm going to have to use '{ }' because that's how strings work. Let's close this out and inside of here the same lookup that I had before. So, '{stock [ ]}' but now I have to be careful what kind of string I use because I'm inside of a string, so I can't actually use double quotation marks.

I'd have to use single quotation marks inside of here, 'name'. But, actually that's fine because this and this are equivalent in Python, even though one uses double quotation marks and the other uses single; you can use either one when creating a dictionary or when getting things out of a dictionary. So, this works just fine. And in fact, as you'll see if you printed out the whole dictionary Python when end up converting all these strings into single quotation marks anyway.

So, here I could say that this will end up being "Microsoft" and I could say "Microsoft's" with an apostrophe, 'stock ticker is' and then another set of '{}', '{stock ['ticker']}', and let's give this one a shot and there we go. So, that's what I was looking for. And you could see that this starts to get a little confusing because there's a lot of symbols in here. So, if you don't understand exactly how this print statement works, I'd recommend pausing the video, sort of looking through it and figuring out, well where's the, where's the dictionary in there, what's the key that we're using, and which set of quotation marks and curly brackets match up with which one.

****

Video 2 – Challenge: Dictionaries

So, let's dive immediately into a challenge to test out our ability to create a new dictionary. I want you to create a dictionary called user and give it the following keys. Name, height, shoe size, hair and eye color, and then I want you to fill in the values of those keys with your own information, there's going to be kind of a personalized dictionary, and then once you've done that, go through and print out the value of each of those keys. Do it however you prefer. Give that one a shot now.

****

Video 3 – Dictionaries Challenge Solution

So, here's the dictionary that I created, as you can see, called it 'user=', and I created the keys of 'name', 'height', 'shoe size', 'hair', and 'eyes'. Now, these are the values, and interestingly, you might have coded the values differently, in the sense that I put my shoe size as '10.5', but I put my height as a string, '5\'10" '. Maybe you used inches, or centimeters, or whatever you preferred, you could have used a number as well.

In this case, I actually use a string, and because the convention of using a apostrophe after the number of feet and double quotes after the number of inches, I actually had to escape this apostrophe quote because I'm using single-quoted strings in this case. If I would have used double quotes, I would have to put this escape before the second set, for this one, right there, but, so I'll just make a little note of that is that, '# You can escape ' in strings using \'.

Okay, and then down here, I am now printing name, height, shoe size, hair, and eyes using an 'f'-string, and inside of this 'f'-string, I have curly brackets, and then I'm doing this dictionary key-value lookup. If you're wondering about the name dictionary or how this works, think of an actual dictionary.

Whereas with a list you're using a number to look something up, in a dictionary if I told you to get the definition of a word, I would give you the word and then you would use that word to find its definition, and that's actually why a dictionary is named, what it's named is that in the same way, you're using a word to get some sort of value, which could be a definition or it could be anything.

Imagine how annoying it would be to use a dictionary if you had to memorize what page number, every word, and definitions are. So, that sort of gives you an idea. Now in the next

video, I'm going to show you how you can add a new key-value pair to a dictionary that you already have.

****

Video 4 – Adding Key/Value Pair to an Existing Dictionary

So, adding a new key and a value to a dictionary is kind of like looking one up except you're setting a value. So, I'll show you. '# Adding a new key/value pair'. Now, you always add them with both; there you can't add a key without a value. You can't add a value without a key, so you're always putting both in at the same time. So, let's say, I take my 'stock' dictionary, which already exists, and I wanted to also keep track and say the open price and the closed price for a particular date. So, if this already had that 'key:value' pair, then I would do stock, open price like that, but it doesn't, so I can actually create it like this, say, 'stock['open price'] = 108.25'.

So, this is effectively creating a new 'key:value' pair. It's sort of like creating a new variable except we're just already using a variable we already have, which is a dictionary. I'll add another one, and then I'll print it out and see what the resulting dictionary looks like. We'll put the 'stock ['close price'] =' as '106.03' and then let's print out the whole thing, 'stock'. So, let's run, 'dictionaries.py' and there we go. We have these other prints from before, but now we're seeing the dictionary itself fully printed out. So, now we have 'name', 'ticker', 'index', and we've added 'open price' and 'close price'.

Something to note, which I mentioned before, was that even though we created the dictionary with strings that had double quotes, they all just got converted into single quotes anyway, which is the default in Python for strings and how they're displayed. Again, either way, you can do it, doesn't really matter. So, as you've noticed here, of course, you can use spaces inside of key names because they're really just strings, so anything that works inside of a string will work as a key, but have you ever wondered what would happen if you tried to use a key that didn't exist or even are these keys case sensitive? Well, let's give it a shot.

If you try to print out, let's say, the '(stock ['Open Price'])', but with capital O and capital P, and then we ran this, we're going to get an error, and now this is a pretty common error to get once you start working with dictionaries, so it's worth being familiar with. This last part is essentially the full error that we're getting, and it's saying that the error is on line 25, obviously, and that the error is a key error, and that open price doesn't exist. So, as you could tell, keys are case-sensitive.

So, in this case, you know, we would either have to lowercase the open price, close price, or you know, create this, but it would be a little bit confusing to have two keys where one is capitalized and one was lowercase. Do you think you can use the same key more than once in a dictionary? Give it shot. Maybe you have an idea of it; maybe you don't. Does it override it, or would it create a new piece? I'm going to comment out this last line here because obviously, it won't work. Make sure this is running again, and in the next video, I'll have a brief challenge for you about adding keys to dictionaries.

****

Video 5 – Challenge Setup: Adding a Key

Next up, I want you to add a favorite movies key into your user dictionary that you created before and fill it in with your favorite movies. Let's say, pick two or three. I want you to think about, should this be a list, or should it be a string? Probably, a list, and then when you print it out, how do you want it to be displayed? Do you want it to be displayed as a list, or, as sort of a bonus challenge, can you convert it when you're printing it out into a string so that it looks like something you or I would read?

****

Video 6 – Challenge Solution: Adding a Key

So, here I've added a new key to my user dictionary by essentially doing as if I would look it up, but then setting it equal to, and in this case, I put in a list of three movies, where the movies are strings. Now, I prefer to do a list because it's the Python data type that is used for lists. If you put in a string with just commas in between, there'd be some things that would be harder to do later on, such as adding a new movie to the list, or things like that, or looping over the list. And then, in this case, I'm just using an 'f-string' to print this out directly, and let's see what the result looks like. 'dictionaries.py'.

Now, as I mentioned before, this is not necessarily the most intuitive or appealing way to print out the list, so how could I make this a little bit easier to read? Well, I could convert this list into a string, and do you remember the way to do that? I mentioned it inside of the video on lists, so pause now, and see if you can give it a shot. And hopefully, you remember that you first put a string for what you want to join them together using.

Now, because this is inside of a string already, I need to be careful about what kind of quotation marks I use to indicate the string. I can't use double quotes because I already have double quotes on the outside, so I'll use ', ' and then I'll do '.join', and then pass in this list as the argument for 'join'.

So, the whole thing looks like this, '{', '.join(user['favorite movies'])}")' and you can pause and copy it down, make sure you got it correctly as well, and then I'll run it to show you what the difference is. See, so now, we converted this list into a string where we're putting a comma, and a space between each element of the string, so we can keep the actual value as a list and just display it as a string, which is pretty convenient.

****

Video 7 – Why Are Dictionaries Used?

So, let's step back for a second and talk about what are some scenarios in which you might use a dictionary versus a list? Now, a dictionary is really useful for representing rows of, let's say, a table, wherein this example I have Excel file with a table, where the column names are name, height, shoe size, hair, favorite movies, and I only have one row here, but you can imagine, I might have lots of different rows for different users.

Now, the best way to represent the data of one particular row would be to use a dictionary, where the key is the column name, and the value is the value of that column in that specific

row. And then, if we wanted to store multiple rows, somehow inside of Python, where you'll often end up seeing is a list of dictionaries.

So, you might have, if I had three rows here, I might see all three of these individually represented as dictionaries, but within a list. So, the first dictionary is row 1, the second is row 2, the third is row 3, and they would all have the same keys, but they would have different values. The keys can actually be replicated in other dictionaries, so it won't be a problem if we use the same keys as long as they're used in different dictionaries. So, as a little bit of a thought experiment, let's talk about another kind of data, a table that you might find in a web application, which is blog posts.

So, if you're running a blog, and you have multiple blog posts and they're somewhere inside of your database, think for a second about what the column names of that table might be, what kind of information do you need to store about a blog post, and if you were to pull one of those blog posts out, and store it as a dictionary in Python, what would the keys of that dictionary be?

So, maybe pause this video, try to think of a few, maybe even write them down? So, the ones that come to mind for me would be a title. A blog post has a title. The text of the blog post, which is actually often called the body of the blog post. The author, the date and time that it's created, and the date and time that it's published, as well as maybe tags, or other things like that. Those might all be contained inside of the table for blog posts, and they would also essentially be the keys of the dictionary representing a single blog post.

****

Video 8 – Group Dictionary Challenge Setup

So, I've gone ahead and provided you with two additional user dictionaries representing different users that you might have in your database or in your table. And, what I want you to do is combine those plus the one you created about yourself and the previous challenge into a list. You're going to have a list of dictionaries, things getting too complicated yet? Once you have this list, I want you to then write a 'for loop' that loops over the list and prints out some information about each user.

So, here's the structure of what I wanted it to look like. It should say '[Name]'s shoe size is', and then their '[shoe size]'. If you'd like, you can also print out other information about that user as well. But, it starts to get complicated as you can tell to think about, will lists are simple on their own, and dictionaries are relatively simple on their own, but when you combine them, how does that start to work? You might have to play around with this one a little bit before you figure it out.

****

Video 9 – Group Dictionary Challenge Solution

So, let's actually tackle this one step-by-step. What I've done is I've gone ahead and put these three user dictionaries inside of my file, and I've given them each individual variable names that way, it's easy to refer to each one of I want to. What this does is it makes it easier for me to put them into a list. So, presumably, these are my users, so I'll call this new list 'users',

and what would be the easiest way to get these three dictionaries into a list? Well, pretty easily, I could just create a list, and then make the values the dictionaries, which I've already saved as variables.

First one's 'mattan', second one's 'chris', and the third one's 'lisa'. Now, what's the actual value of 'users' going to be? Let's print it out, and you'll see that this list now has these three dictionaries inside of it. Essentially, I just use these variables as boxes to move a dictionary over from one place and to another, but I could have just actually pasted the dictionaries directly into the list if I wanted to; it just would have been a little harder to read. Okay, so now I have a list 'users'. So, how do I loop over a list? Well, remember from the looping videos that you do 'for' something in users, probably a singular user would do.

So, 'for user in users:'. If it helps you think about it, you can use the letter 'i', or you can use something else, but I like to use the singular and plural. So, I have three users, and now within this 'for loop', I have the user variable. Remember that each user is a dictionary, so if I wanted to print out a particular value, and I know the key, and the key is the same for all three dictionaries. Well, then I can just print out. Let's say, let's first grab the user's name, so '(user['name'])'. So, what this should do is work correctly is just print out 'Mattan', 'Chris', 'Lisa', which it does right here.

So, you see how this works. I'm looping over a list, and then each user is a dictionary, and I'm using the key to get the value of that particular dictionary. But of course, I don't have a exact dictionary named user up here; it's just the placeholder for whatever element of the list I'm looping over. So, now, if I wanted to get each user's shoe size as well and put it inside of a string. I can use an 'f-string'; I could first just wrap this in quotation marks and curly brackets and then put an 'f' in front of it.

Now, I have 'users', and I'll say no, I don't need that second apostrophe. It's one annoying thing about some text editors is when you put one apostrophe, it always puts the matching one; in this case, we don't want that. So, I'll say that their 'shoe size is' then a second set of curly brackets, 'user', and then we'll do the 'shoe size' lookup. So, let's save that and get this running, and there we go. I see 'Mattan's shoe size is 10.5' 'Chris's shoe size is 11' 'Lisa's shoe size is 6.5'

Now, if this is confusing, pause the video now, look through the solution and see if you can figure out exactly why it's working and another thing that's sort of interesting to note here is that there's not necessarily consistency between how these values are represented, but even though my 'Mattan user' has height that's a string, 'Chris' and 'Lisa' both have their heights coded as integers, and that's actually, relatively common when you're working with datasets.

This is sort of foreshadowing a problem; we're going to run into later when we're doing data analysis is that you don't necessarily get consistency. However, in this case, the keys are consistent, and that's the reason why this 'for loop' works. The key names were different for all the dictionaries; we wouldn't be able to loop over them and use the same key name to get the value out of that dictionary.

****

Video 10 – Get a list of users demo and VLOOKUP demo

Can you think of how we might be able to use 'for loops' and also an 'if statement' to get a subset of users that we currently have? So, for example we currently have three users. What if I wanted to get all of the users who have brown eyes? So, think back to the video on looping, and how we converted one list to another. I'll give you a shot and show how I would do this. So, '# Selecting a subset of users'. So, we'll start out by creating a empty list that we can append users into. That way we have two lists; one with our original users and one with just our 'brown_eyed_users = []', so this is equal to an empty list.

Now, let's loop over our users list says, 'for users in users:' Now, what do we want to do here? Well, basically we're going down the line for each one and we want to check, does that one have brown eyes? If so, append them into the list that we just created. So, we can say, 'if user['eyes']' so that will get the value of the eyes key for that particular dictionary and how do we check if it is equal to a certain value? Or, we can just say '== 'Brown' ', and that should give us True or False.

And if it is True, we can just append that user into this 'brown_eyed_users' dictionary or list. So, 'brown_eyed_users. append(users)' because that will be that particular user in that case. In that way, now if we wanted to we could just print out that list of 'print(brown_eyed_users)' and we'll test it out and see if it worked. So, that's this last thing that was printed out right here and it looks like we got 'Mattan' and 'Lisa' both have brown eyes, but we don't have Chris in there because Chris has blue eyes.

So, it looked like that worked. Now this seems, you know, there's a lot that's going on; we're doing a 'for loop', we're doing 'if', we're appending. It is possible to do this in one line using a list comprehension, which is something that I mentioned earlier in the lesson on lists as well. But this is sort of the basic and maybe one of the easier ways to understand this. Similarly, you could identify just one particular user. So, if we wanted to do like an exact search for a user with a specific name, it might go very similarly.

So, you know, 'for user in users:' ' If user ['name'] == 'Chris': ' then, well then, what do I want to do? Do I want to print the entire dictionary? So, I could say, 'print (user)' and that would then just print out basically any user whose name is "Chris", in this case, there's just one. Or I could print out Chris's favorite movies or all the information about him. Or if I wanted to just use him for something, I could actually set him equal to some variable and that variable would come out of there.

So, you know, I would, say 'chris =user' for example, even though I all ready have Chris up here. So, maybe this will be something like 'target_user' or 'searched_user', I don't know, but it's up to you to decide at that point. And this is sort of similar to like how a VLOOKUP function in Excel might work, if you're familiar with that. Searching through one particular column, finding a value and then you can pull out that entire row and do whatever you want to with it.

****

Video 11 – Functions

We're now ready to talk about one of the most powerful concepts that exists in all of programming, which is a function. Now, you're probably already familiar with the idea of a function from Excel, for example. Excel has plenty of built-in functions that you would use all the time. And, in fact, we've already been using many functions in Python, such as Print, Length, lower case, upper case, like those sorts of things, they're all functions. But I'm going to show you a little more about how functions work, and then I'm also going to show you how you can create your own functions that you can then re-use, just like any other function we've been using inside of Python.

So, in order to do this, let's create a new file inside of Atom. Let's save it and call this one 'functions.py,' and we'll demonstrate creating a few different functions. But first, I want to show you sort of the use case for why we create functions. So, let's say we have a string, and let's call the string, let's put it into the 'address =' variable then, the string is ' "3022 Broadway, New York, NY 10027, USA" ' So this is the address of our particular location. This happens to be Columbia Business School.

And, you know, maybe this is something we've gotten from a user when they signed up and they saved their address, or maybe we're shipping something to someone, and they give us their address. So we have this full string, now it would be useful to be able to extract certain things out of the string, because it contains a lot of information, and we can look at this, and we know what the city is, and we know what the state is. But how can we have Python pull out the city, for example, here, I'll show you how I might do that.

I would take first say 'city = address.' and then take the address, and one of the functions that I mentioned earlier, that's a string function, or a function you can run directly on a string, such as address here, is the split function. So, what split will do, it will take a string, and it will split it by some character or set of characters, and turn it into a list. So, by default, split will split on spaces, which is not exactly what we want in this case, because then we would end up with '3022' and then the second element 'Broadway' with a ',' 'New York', etc.

If want to get specifically this part, the city out of the list, it makes sense for me to split on a ', ' and a space. And that way we'll end up with a list with one, two, three, four different elements, because there's three sets of comments and then what? Well, then what I can do, I can grab the second element of that list because that would be the city, and that should end up, you know, I can save that into the city variable. So, do you remember how I would grab the second element of a list? Inside of '[]', what number do I put in there? It's the number '1', because lists actually start with the number 0. So, if we did this all correctly, and then I print out this 'city', what I should see is 'New York'.

So let's give that a shot. There we go. So, that worked properly. Now, what I did here was I combined at least two steps into one step. I split a string, and then I took out an element of that resulting list. And presumably, depending on what kind of addresses I was given that I would imagine that the same series of steps would work on many different kinds of addresses, as long as there's, you know, the right commas in the right places. So, let's say I actually have a bunch of addresses like this, and I don't want to have to manually go through and do this same step to each one of them.

Of course, I could use a 'for loop,' that's one way to do it. But if I'm doing this all the time, it might make sense for me to just create a function I could use, where I can pass the function string, such as an address, and it would get the city out for me. So, let me show you how I would create this as a function. So, let's do this below. The way that you create a function in Python is you start with the three letters, 'def ' and then a space. The 'def ' stands for define, and what we're doing is we're defining a new function.

Now, I define, and I put the function name. So, we can call this one, let's say 'get_city'. That will be the name of the function, so when we actually run the function, we'll do that by running 'get_city'. Then I need ' () ' So, ' () ' almost always go with functions, when you actually run the function, you have to put ' () ' afterwards, and when you define a function, you have to put ' () ' after the function name. What goes inside of ' () ' is called a function argument. So, in this case, I'm going to call it address.

It's, I make it up. I actually would create, it's kind of like creating a new variable. I'm creating a placeholder for what's going to be passed into the function when I later run it. So, one of the things to think about when you're creating a function is you don't really know exactly what kind of data is going to be passed into it, but you should have some idea of is it a string, is it a number, what kind of thing is it? In this case it's a string, but I'm going to expect it to have a particular format, kind of like the address string that we saw above.

But what's also important to note is that this address does not necessarily refer to the variable we created before, called 'address'. It will refer to whatever we give the function when we actually call it later, as I'll show you. The final thing we need when we define a function is a ':' at the end. Kind of like when we were using 'if' or when we were doing a 'for loop'. When you create a function, use a ':', and that allows you to then put the code inside of the function. All the code in the function is going to be tabbed in.

So, inside this function here now, presumably we're going to get an address when we run the function, so what do we need to do to that address? Well, we do the same steps that we did above. So, we take the 'address' and we do '.split ', and we're going to split it on a (', ') and then we're going to grab out the second element. The final thing you have to do here when you are defining a function is you have to define what the output of the function is.

We'll talk more about function inputs and outputs in a following video and sort of clarify some of this stuff, but a function might have multiple lines of code and so, how do you define what someone gets back when they use the function. You put the word 'return' in front of that line of code. So, just 'return ' and a space. Return is not a function; it is just how you indicate which line of code is the output of the function. And in this case, there's only one line of code, but you can create functions with as many lines as possible.

So, actually, just to sort of prove that this works, let's get rid of the code above, we can, we can take this string out of here, paste it below, and then we'll run 'get_city' and put that string into there, but we will get rid of all this code above. So, there's actually, I got rid of the address and variable that I created above. And you'll see this will still work. Because all I'm doing is creating a function, calling the first argument 'address.'

This becomes whatever I pass into it. Got it? So within the function, I can use that name as like the placeholder for whatever I'll pass later, but this could be any kind of address. I can

put in any street number here, or any other city. The thing is, if I run this code, 'python functions.py' I'm not seeing anything.

And can you think of why that might be? It's not that the code isn't running, it's that I haven't printed anything out. So, even though this does run, and it does actually return 'New York', I need to print it out. So, I can do that either by setting that equal to a variable, or just wrapping the entire thing with print. Now, I run this, and there we go. I got New York. So, we just created our first function.

****

Video 12 – Function Challenge Setup: get_state

So, as a challenge, I want you to create a second function, and this time we're going to call it 'get_state()' Can you guess what it's going to do? Well, it's going to take a string of the same form that we saw before, so address, state, or city, state, zip code, and country, and I want it to return just the state. In this case, it would be New York. Also, give it a shot with different kinds of addresses, and make sure that it works.

****

Video 13 – Function Challenge Solution

So, let's walk through this challenge step by step. We're going to start by defining a new function. Do you remember how to do that? It always starts with the letters 'def' then the function 'name' in this case, we'll do 'get_state'. Then, an ' (' which in Atom will automatically give you ') ' as well. and then whatever the arguments are that it accepts. We'll have the same argument as before, though it is possible to create a function with multiple arguments, putting ' , ' here, then putting another one, another one, and so on.

We can take as many arguments as you want or need. Then, a ' : ' at the end. So, don't forget the colon at the end of the function definition. So, now, we're going to start. Don't forget the return, and I'll show you what happens if we do forget the return in a second, but we're going to have something called a variable that's going to get input as a function argument when this function is called or run. So, we are starting with our address, you know, what's going to come in.

And now, what do we need to do with it? Well, our goal is to take this state part out. So, we can first of all split it on a ','. So, if we do 'split (,)' what we'll end up with is the following elements, the street, the city, the state and the zip, and then the country. So, the part that contains the information we want, namely the state, is going to be the third element of the list. That is produced when you split it. However, it is going to also include the zip code. So, there is going to be a little bit of extra work that we need to do.

But we know that we need to get the third element of the list. And how do we do that, if you have a list? '[ ]' and then the number 2, because that's the third element. The first one is at 0. The problem is, right now, this just returns the state and the zip, but we only want the state. So, how do we now go a step further to extract only the first part? And if this is hard for you to keep track of, without doing it step by step, and actually printing out the results, that's what I would recommend doing, is either, you know, go through this and print out the results

so you can really see tangibly, but it's useful to start to stretch yourself by thinking about, you know, what are we going to get back from this right now?

And being able to keep that just in your mind, without necessarily having to see it. Although if you do have to see it at this point, of course, you can just 'print(get_state( ))', and then pass in the same address before, and you'll see that right now we're getting back 'NY10027'. So, how would we go from that to just the state? Well, if you're thinking you can do another split, that's one way to do it. So, if you add '.split( )' at the end, what we're now doing is we're splitting this string, and if you don't pass in an argument, it will automatically split on a space by default, which is good, because that's what's there.

You can also add in the space, and make it explicit, and just put in an empty strain with the space, and that will also work. But now, we basically split this subsection into a list, and we want to grab the first element of that, which will be at '[0]'. So, if we're following, we're basically doing a split, grabbing the third thing, and then splitting that again, and then grabbing the first thing. And if I run this, I should see 'New York'. So, this should work. And as I mentioned before, I'll tell you what happens if you forget to put the return at the end of this function.

When you run this, you're going to see the word 'none' printed out, and that's because if you forget to put a return at a function, if you forget to specifically define what the output of a function is, well, then the output of a function will always be what Python calls none, which is basically nothing, it's empty. Even though the work is still being done inside the function, it's still splitting the address, and splitting it again, and getting that New York part, nothing happens to it.

It essentially gets lost, and vanishes into ether. It would be like doing some work in Python, but then forgetting the save the result to a variable so that you can do something with it later. It just disappears. And the same is true for a function. Which is why you always want to remember to be explicit about putting a return on the line of the function where you actually want the output to be. It's usually the last line, and in fact, any code that you have after a return in a function won't get run anyway. So, but unfortunately, it's something you do have to make explicit, and remember.

****

Video 14 – Functions as Process Machines

I want to take a minute to just talk about functions generally because they're a big topic, and I want to make sure we understand when to create them and how they can be used, and really what they do. I like to think about function as kind of a process machine. So, you can imagine the idea of a process generally, like in a factory; it takes in different kinds of inputs, and it does things to them, it combines them together in different ways, and then you get an output, it can be a car, it can be whatever. So, if a function sort of encapsulates that process, we've got the function as a box, that takes multiple inputs.

It can be one input; it can be as many as you want and gives you one output. In Python, a function can only give you one output. However, that output could be a list, or it could be a dictionary. So, it could have multiple different values inside of it, but it does take multiple inputs. So, it makes sense to create a function that encapsulates a process for a number of

reasons. If you plan to do that process multiple times, then it makes sense to create a function. Another thing, another reason why creating a function might be useful is if you want someone else to be able to do that process, without them really having to worry much about exactly how it works.

So, if you were to write a function that other people were going to use, that basically means they could use the function and not worry about how you defined it, for example, we've been using print all the time in Python. We have no idea how print was defined in the first place, and it sort of doesn't really matter, because all we know is that it takes strings and that it prints the output directly into the command line. So, that's very useful, and it provides a level of abstraction that increases productivity a lot. Now, a function and a process generally can take one kind of input and even give you a different kind of input, or the same.

So, I'll give you a few examples, and I'll give you an example, and then I'd recommend pausing the video and actually thinking about it a bit. So, let's say I have a function that takes a number. One number, as an input, and then gives you back a number as an output, and I haven't told you anything about what the function does. Can you think of an example of a function that takes a number as an input, and then gives you back another number as an output?

So, two examples that I can think of would be either getting the square of a number, theinput would be 2, and then output would be 4 or 3, and then the output would be 9 or the square root, that's another function that takes one number as an input, and gives you one number back as an output. How about a different function? This one takes a list of numbers as an input and gives you back a number as an output. Can you think of some examples of functions that do that? So, one example of this might be calculating the average of a list of numbers.

If I had three numbers, and then I got back one number, that was the average of those three. Two other examples might be getting the minimum or the maximum. Finally, let's say I got text as the input, like a string for a function, but I got back a number as an output. What's an example of a function that might do that? Well, one of those functions might be calculating the length of that string. The number of characters in that string, for example, could be a function or even the number of words in that string. That's one way of getting a number back.

Another way to think about it is let's say I have a function, and I could tell you what the input and the output is, and you guess what that function is actually doing. Some of these will be pretty simple functions; some of them can be more complex functions. So, if my function took 5 as the input and gave you back 6, if it took 0 and gave you back 1, or if it took negative 3, and gave you back negative 2, can you think of what the function is probably doing inside? Yeah, it's just adding one to the number, and a lot of times, the function name would give you a clue as to what the function is doing.

So, this one might be called add one. The thing about functions though, as we'll see later on, is that it's possible to pass them something it's not expecting, such as what happens if you take the add one function and pass it a string? Is it going to give you an error message, or is it going to return something like that's not a number, please try again? There's all sorts of different ways that you can deal with different kinds of inputs, and that would all be defined inside of the function. So, now that we've talked a little bit about functions conceptually let's dive in further and create a few more functions.

****

Video 15 – Using Functions to Reduce Redundancy and Refactoring

So, remember back to the Fizzbuzz challenge, where we printed the list of numbers from 1 to 100, but instead of numbers divisible by 3, we printed Fizz, and Buzz, and et cetera. We did this check over and over again, where we saw if one number was divisible by another, and it looked like this, 'number', that's the modulo sign, another number, '== 0'. And the problem with this is that it's not really clear if someone was reading the code exactly what's happening.

Maybe they're not familiar with the modulo symbol, or they don't know what this check is doing. And the fact that we're doing the same thing, the same kind of check over and over, this is a good indicator that there's probably a function in here somewhere we could create to make this simpler. So, let's try to create this divisible by function. So, let me go back to my functions file, and the goal is to create a function that takes two numbers, and checks if the first number is divisible by the second number or not. But we'll start out by defining the function and thinking about what we might want to call it.

I'll go with 'def is_divisible()' because I try to name these things as clearly as possible, so when we run it, it makes sense, and then, what do we want to call the arguments here? Would you call it if the first number that you were trying to check was divisible by the second number? I might go with something like 'number,' and then let's say, 'divisor', seems good enough, and then a ':'. So, the first time I thought of writing this function, I did the following, and I'll show you how it might work. So I'll do an 'if' check, 'if number % divisor == 0: return True' 'else: return False'.

So, here is actually an example of a function that has two different lines that have a return on them, but because they're inside of an 'if' and an 'else', only one of them will ever get called, so that will still work. Now, I can run this function, I could do 'is_divisible()', spelled that wrong, but it autocorrects, and let's check to see is '15' divisible by '3'? And then also is '20' divisible by '3'? So, I should end up getting True and False. Now, in both of these cases, this is how you call the function, you run the function, and you pass in arguments, but even though I'm getting it back, I'm not seeing the result, I have to print it out, in order to actually see True or False.

So, let's 'print(is_divisible)', and what I should see is True, False, and in fact, that's the output that I get. So, this is a perfectly reasonable function. Now, the first time I made this function, I wrote it this way, and then I realized; actually, there is a way simpler way to write this function and I want to see if you also see it, maybe you saw it when I first created it. I have

this function with four lines, but I could do it with one line because this check actually returns True or False anyway.

So, I don't even need this whole 'if', I can just 'return number % divisor == 0', and the result of this will be either True or False, if it's divisible or not. So, I missed that the first time, now I've changed it, and I run it, and I get the same result. This is actually a really powerful concept called refactoring. When it comes to coding, refactoring is the practice of changing your code so that it's better in some way, without changing the functionality, without changing what it actually does. So, it's better in this case, because it's clearer, it's shorter, and when it doesn't need to be longer, shorter is not always better, but as long as it's clearer then generally less code is better.

And what's cool about a function is that it allows you to refactor your code a lot more easily. Because if you want to change the function in some way, all you have to do is go into the function, make your changes, but anything else, any other part of the code that uses the function doesn't have to change, it doesn't really care about how the function does what it does, it just cares that it works. So, it's a powerful concept, and there's a similar concept called technical debt around coding.

As code tends to grow, people tend to write code in some suboptimal way, you know, maybe they either take shortcuts, or they write it the first way they think of, but it's not necessarily the cleanest, or it's not necessarily the most flexible or easy for someone to change or understand. So, the idea is that, you add up technical debt, to these little shortcuts that accumulate over time, and refactoring is one of those practices of going back to your code, making sure it's clean, making sure that other people can understand it, because if your technical debt accumulates to be too high, then it becomes hard to make changes and add things to your code.

So, you can imagine a company that has been writing code for a long time, where the code is thousands and thousands of lines, and they've never gone back to refactor, you tell an engineer to make even a relatively simple change, it might be a hard thing to do, because you know, it involves changing multiple parts of the file, and it makes it error-prone, and easy to make a mistake, which is something you want to avoid. So, refactoring, always a good thing, reducing technical debt, always a good thing.

****

Video 16 – Challenge: Functions II

So, for your second function challenge, I want you to create a function called 'uppercase_and_reverse()', and even though I'm showing parentheses here, it doesn't mean you don't have any arguments to this function, it will take an argument named text. That's just a convention for when you're referring to a function in Python, so you know, it's not a variable.

So, uppercase and reverse is going to take some text, and give it back to you uppercased and reversed. So, below is an example of how this might work, and I'm showing this to you within the context of imagining that we're opening up the Python interactive shell, that's why you see the '>>>'. The idea would be if you run 'uppercase_and_reverse' and put '(‘banana’)', what

you get back is 'ANANAB'. So, think about how you might do this, and then we'll go over the solution together in the next video.

****

Video 17 – Functions II Challenge Solution

So, I have here my solution to the previous challenge, and it's about as concise as I was able to get it. I've got my 'uppercase_and_reverse(text):' And here in one line, I'm returning both the 'text.upper( )', which upper cases it, and then I'm doing this '[: :-1]'. So, maybe you figured this out on your own as well or maybe you didn't, but it's not clear just looking at this, what does this do? This actually reverses text, and how did I figure that out? Well, I searched Google for reversing a string in Python, and one of the things I found was this Stack Overflow page for how do you reverse a string in Python?

And it's a very simple answer, it shows you that if you take text and you add this at the end, what you get back is the same text backwards. And it actually, then goes on to explain this is part of what's called extended slice syntax. It works by doing '[beginning : end : step]', which we've already seen slice syntax or slice notation when we were pulling out subsets of lists, but it turns out that this also works with strings as well.

So, if you would like, you can explore that further to really see how it works. But, you know, as I kind of say, if it does work, it's not all that important for you to understand why unless it's something you're interested in exploring further. So, if I has to say, this was my solution, and then when I run this file, you can see the result I get back is 'ANANAB'.

****

Video 18 – More About Functions (Part 1): What Happens With Too Many/Too Few Arguments? And Optional Arguments

Let's talk about what happens when you run a function with too many or too few arguments. Maybe you've thought about this already and you were curious. And if so, I hope you actually tried it out, to see what would happen. But if not, we'll do it together. So, let's take one of the functions we've already created, such as the 'get_city' function. And what happens if I try to run 'get_city' without any arguments, just like that?

So, running a function just means putting the name of the function in '()' afterwards, but usually, when you run a function it expects the number of function arguments, when you run it, to match the number that we defined when we created the function. So, even without printing it, this will still run the function. And so, if I try to run this now, what will happen was it will get to that function and it will give me an error. And the specific error is a TypeError that says ' get_city() is missing one required positional argument: 'address.''

So, it actually tells me it's missing the address argument. What if we try to do too many? So, let's comment this out for now, and just say '# Too many or too few function arguments.' Just to make a note. So, let's say I try to run 'get_city', but I passed in, you know, '("3022 Broadway"," but then as the second argument I passed in ' 'New York, NY', ' and then, yeah, as a third, ' '10027, USA')' Well, what's going to happen is something very similar to before. It's

going to say 'get_city() takes 1 positional argument but 3 were given' So, it's making it pretty clear that this function was only expecting one argument.

So, neither of these will work. What about print? So, you've noticed before that print would accept either many function arguments or zero function arguments as indicated by the fact that I can 'print()', like that, and that just puts an empty line, or I can 'print("This", "will", "work", "as", "well")' Not that it makes that much sense to do this kind of prints, but prints seems kind of like a special function in this case. Well, we can actually go in and read the print documentation or some, basically descriptions within Python, that tells you how 'print' works.

So, if I look up the python print function, if you search for a function in Python, what you'll often find is the documentation as part of Python, so that's actually, I know an easier way to do this, is to just look up python functions. Python Print documentation. There we go. So, I had to specifically add documentation which is part the coding word for like the manual related to that specific function. Now, Python also has two forms of documentation that they keep live. There's the 3.7.4, in this case, the most recent, and they also keep the documentation for 2.7 around, because as I mentioned before, people still use Python version 2.

So, if you do look up the documentation, make sure you're getting the right one, and it's easy once you're on a page to actually switch up here at the top and click on this drop-down, and actually switch between the different versions. So, this gives you a list of some of the built-in Python functions, in case you were curious about what the different ones are. We have used print, and so that's in here, and a few of the other ones we've also checked out. Int, for example, float, you know, you can use list, and length to get the length of something.

But let's go down to the print one. So, you've got a little bit of description here of how print works. You also see the different function arguments that print accepts. So this, this star objects, that's kind of special. That allows them to define a function that takes as many arguments as it wants for this kind of thing. So, that's how they actually get around that. But this also kind of gives you an interesting trick, that there are these things called optional arguments. So, it is possible, when you define a function, to give it an optional argument.

So, this is something that doesn't have to be passed through, but it can be if you want to. Specifically here, we've got the 'sep' optional argument, and 'file', and 'flush'. So, I'll give you an example of how an optional argument might be defined. Let's say we defined a new function and we call it 'greet'. And the idea is I want to greet someone by name. So, I'm taking a '(name)'. I can give this a default value.

So, I can say as part of the function definition the '(name = ‘You’)' in capital Y, just like when you meet someone you forget their name, you're like ‘Hey you!’ So, let's put a ' : ' and now, I'm going to return a string. And it's going to be an 'f-string', so it's actually going to say ' f’Hey {name}!’ ' and then close the string. We don't need that extra one. So here, it's a very simple function, and I can run it, I can say 'print(greet)', someone by name, so '(greet(‘Mattan’))', and I can run this.

And here it'll say 'Hey Mattan!' but you also see that if I try to greet and not pass in a name, that will just say 'Hey You!' So, this is a really convenient thing, and it does indicate, for example, that the separator, the separator optional argument for print, which by default is a

space, is something you can override. So theoretically, you know, even though it's putting spaces in between words by default, you can replace that with any other character that you'd want to replace it with. So, for example, just copy this, put that below, but I replace the separator with, for some reason, let's say a new line break, ' \n ', that actually means that each of these will now print out on a new line. It's pretty handy.

****

Video 19 – More About Functions (Part 2): Function Gotchas (Sneaking in Variables)

I want to talk about some gotchas when it comes to functions, and I want to start out by calling out a tweet, which is one that I love, and it goes, a QA Engineer walks into a bar. Orders a beer, orders zero beers, orders 99999 beers, orders a lizard, -1 beers, orders a smashing on the keyboard. The idea is a QA engineer is an engineer or a software developer that's responsible for quality assurance of your code and what they're doing is they're going through, and they're trying to see if the code breaks when it gets things that it doesn't expect.

So, when you create functions, such as the ones that we've created, something to keep in mind is, what happens if the function gets an argument that it doesn't expect? Does it still work? Does it not work? And it's something you need to be thinking of in the back of your mind, as well as what kind of error might actually come up? Like, if I try to run 'get_city' with a zip code, is it going to work, or am I going to get a problem?

So, there's that point, but I specifically want to talk about one gotcha, which is a very common one, especially when beginners first learn about functions, which is that they accidentally sneak in variables into their functions. So, I'll create a new function here as a pretty simple example, and this is the reverse function. So, what it does is it takes text, and it returns the text reversed, kind of like what we did above, where we uppercased and reversed it, but this is just reversing.

So, let's say we have a variable up here, and we call it 'word', and the idea is that the 'word' is the word 'jelly', and we're going to pass in, and reverse it. But let's say we made a mistake here, right? We got our variables confused, and we took in the text argument, but down here, we accidentally referred to 'word', right? Which is this thing up here. What happens when I try to 'print(reverse(word))'? Well, it's going to look like it works, I'm going to end up with 'yllej', so jelly backwards. The problem here is that, in fact, when I'm calling this function, the argument I'm passing in no longer matters; it doesn't do anything anymore.

So, you know, if I passed in Python into here, I'm still going to get jelly backwards, and the reason for this is, I've accidentally let this variable sneak into my function. So, the way that functions work is that variables defined outside of the function are technically available inside the function, but you shouldn't use them unless you specifically set them as function arguments, and in that case, they need to be passed through when you run the function. It's the same way as if you have like a process; you need to be really clear about what inputs are going into the process, you don't want other external things sneaking into that process, without you realizing.

So, as a really good best practice, anything that your function is working with as an input should be defined as a function argument, and you should make sure you don't accidentally

refer to any variables that were created outside of it by just using your function arguments. That being said, any variables that you create inside your function will not be available outside of it anymore. So, it's pretty much impossible for things to sneak out of your function unless you explicitly 'return' them at the end of the function argument. So, something to be aware of, and it's something you might run into at some point in the future, but I'll just make this note here, ' # Don't let variables sneak into your function', got it.

****

Video 20 – Import

Another cool thing about defining functions is that they can actually be easily imported into another file. And if you remember back to our 'happy_hour' file, you'll see that actually at the top we have this line, 'import random'. And it's finally time for us to address what that actually did. Down below, what we were using was 'random.choice' to randomly choose one thing out of a list. So, this 'random' is actually a file that includes a bunch of functions that are useful, that we might be able to use, in the same way, we can create our own file and then 'import functions' from that.

So, I'll show you how to do that, and then we'll talk more about what you can do with importing. So, remember in our FizzBuzz challenge, we did this division check a lot. And we've now created a function 'is_divisible' that will check if one number is divisible by another. Now, it's possible to add that into this file and use it directly or we can import it to this file. So, the way that you would do that would be well, there's multiple ways of importing, but let's say at the top of this file, we could 'import functions'.

And I'm saying this, because the file name is 'functions', and when I run 'import functions', it's going to import, it's going to look for a file with that name, even though there's no 'py' at the end, and it's going to import that. And I'll show you what happens here. If I run this 'fizzbuzz' file now, well, I get 'FizzBuzz', but I also get at the top, I get all of the output from before, so, from our 'functions' file. One thing that import does; it literally just runs the entire file up here that we've imported.

So, the first place it looks when you try to run an import is in the same folder that you're currently in, which is this folder, and it's going to find this 'function' file. So, maybe that's not exactly what we want. Maybe the idea is we want to put this 'function' in a separate file, and then just import that. So, one thing to be careful about is, you know, this will run all of the code in another file when you do an import. So, let's actually just create a new file, and maybe call this, paste the, what is it, the 'is_divisible_by' function, directly into there. Save this, and let's call this one 'divisible.py'.

Okay, so now, I have just a Python file with a specific function. I no longer need this whole functions file. So, I go back here, and just import 'divisible'. And that will prevent all of this other unintentional code from running, but this function will still run when I import it. So now, how do I use that function inside of here? Well, one way of doing this is now that I have this file, I can do 'divisible.' and then what is the name of the function here? 'is_divisible', so 'is_divisible' and now, I can run that function from that other file.

So, what were we checking before? To see if '(number, 3)'. We can do the same thing here as well. 'divisible.is_divisible(number, 5)', and then going all the way down. Now, this is one way

to do it, and actually, I'll clear this code and run this, and you'll see that it does, in fact, work. I'm still seeing 'Fizz', 'Buzz', and I can go through and I can replace the other ones as well. But again, this is kind of long.

This is how actually we did it in the 'happy_hour.py' file, but there's an even shorter way to do it. I could just import that specific function, which means that I wouldn't have to add the file name, dot, the function. So, instead of importing 'divisible' in its entirety, I can do import, 'from divisible'. So, that will mean go to this file named 'divisible.py', 'import is_divisible'. So that's saying, specifically 'import the function' is_divisible. Which, in this case, it's the only function that's there.

If I do it that way, then I no longer have to preface it with the name of the file and then put dot. So, that allows me to just run the function directly as if it had just been defined in this file. So, here I can just do 'is_divisible' and that would be '(number, 3):' and then finally, 'is_divisible(number, 5):' So now, I've basically replaced all those number checks, to see if one number was divisible by another, with the function 'is_divisible' that I've defined in this file, and I've imported into my 'fizzbuzz' file.

So now, I can run 'fizzbuzz', and you see that it still works, and essentially, I'm using a function that I created somewhere else. Now, why this is useful is because if you've written a lot of functions that you plan to re-use across multiple files, you can easily just save them in a Python file, and then import them into any other file. You can either import the entire file, as we did up here in the first way, but that means you'll have to, when you use any of the functions in that file, put that file name, dot, and then the function, or you can import specific functions like we did here.

And if I wanted to grab another one, like there were multiple functions there, I could put commas, and then list each one, one after the other, and that would make them all available. The reason why they force you to do it this way if you want to grab a specific function, and you know, not have to preface it with something is that it's very easy to accidentally override one function with another function when you're doing an import, like if I had a 'print function' in here that I was defining, I could accidentally import the whole file, not realize that I've just overridden the print function, because I don't know what's there.

So, this is called name spacing, meaning if all of the functions in that file are accessible, then you have to put the file name first, so 'divisible.print'. That would separate it from the other 'print function', which we have, which we don't put a dot in front of. But if we want to access it without having to put the file name dot, then we have to be really explicit about which functions we're pulling in. So, there's a lot that you can do with importing, but let's test it out a little bit and I'm going to give you a challenge in the next video.

****

Video 21 – Import Challenge Setup

So, for this importing challenge, I'm going to have you create a new file called 'finance_functions.py'. And in it, I want you to define a function that calculates the future value of some amount of money based on the following formula. So, there's this concept of future value, which says if you take a certain present value like say a $100, and a rate of return like at 10% interest rate, and a number of periods, five years, for example.

You can calculate what that $100 will be worth in five years using this formula. So, I want you to define a function that takes those inputs and outputs the future value. Put it inside of finance functions. Then what I want you to do is open up the Python interactive shell. So, rather than importing it into another file, you can also import directly into the interactive shell. It works the same way. And I want you to practice importing this function and then using it to calculate the future value of a $1000 in five years at a 10% interest rate. It's a bit of a tricky challenge, but I believe that you can do it.

****

Video 22 – Import Challenge Solution

Okay, so what I've done is created a file called 'finance_functions.py', I've saved it into my code folder, and it works like the other functions it takes a 'present_value' argument, a 'rate', and a number of 'periods', and then it does the formula, so 'present_value * (1 + rate) ** periods'. Remember that exponents are done in Python using '**'. So, now I'm going to experiment with importing this into the Python interactive shell. Now, the trick about this is you have to be inside the folder that this file is in when you open the Python interactive shell.

So, Python and now where in here, and what I can do is either 'import finance_functions', I don't need to put the '.py' at the end, so that's another thing about the import. Even though the filename actually has it when I import, I don't have to put it. And so, either I can do this, and then I would have to do 'finance_functions.future_value', and I can put in, let's say a '(1000', and the rate here would be '0.05' to '10)', that's number of periods, and that's what I'm getting. '$1628.89', and it goes on a lot, or what I could've done was just imported this specific function from the file.

So, 'from finance_functions import future_value', and now I can just run the 'future_value' function directly without having to preface it with a filename, which can be more convenient. Now, I can calculate '(1000, .05, 10)', and I get the same result back, which is kinda cool. Now, there's a few changes I might want to make at this point. So, one nice thing about defining this function somewhere is well, let's say I don't actually need all those cents. Let's say I want to round this to 2 cents, it'd be pretty easy for me to just go in here, use the round function, which I mentioned briefly, but the round function by default, automatically rounds the nearest whole number, but it takes an optional second argument for a number of digits.

So, if I'm rounding this whole thing, and I put a ',' at the end and I put '2'. Well, now I would have to re-import this file to update it, and I can use the up arrow here, by the way, to go back through previous Python commands as well, and save this. Let's try to import it again; I might have to exit out of here. So, let's exit out, and open up the Python interactive shell again, there we go. Now, it's rounded. I believe what happened before it was it had already loaded the function, and so, when I tried to load it again, it said it's already loaded, so it didn't update it, unfortunately.

So, now I have a rounded function, rounded down to the nearest 2 cents. Additional things that you might want to do here is maybe create some default arguments, for example, if you wanted to assign a default interest rate of 10%, or a default number periods of 5, that means that you could run this function without having to pass those things and explicitly. Another thing I'll briefly mention is that you can move around the order of your arguments. So, by

default, we've just been passing them in, and then they get used in the order in which they given, but it is possible to change them around if you explicitly name each one.

So, in this case, I could run 'future_value', and let's say, I wanted to say that the '(rate=.01', I can put that in as the first argument, as long as I tell it that that's the rate. And then I say that the 'present_value=100,' and that the 'periods=5)'. So, now I'm getting really specific, I have to use the same names here as the actual function arguments, that's how it'll know what order to put them in. But this is another way of kind of shifting things around if I want to be explicit about which argument I'm passing in when I run the function.

****

Video 23 – The Python Standard Library

Now, I want to show you one of my favorite online webcomics, and it's related to what we've been talking about, I promise. It's from the site, XKCD, which is a pretty nerdy webcomic that a lot of people like to read, and this one is called Python. So, I'll give you a second to read it; you can pause the video if you'd like. If it's not all that funny, if you didn't get it, don't worry. It's not the funniest joke in the world, but the basic idea here, of what makes Python so powerful, is that it is so easy for you to just import a library and expend the functionality and do something like flying by importing antigravity.

You know this is a joke, but in reality it's one of the things that sets Python apart from other languages, is that it's easy to extend, and not only that, you can extend two libraries that other people have written as well, and we're going to talk a little more about that. So, one of the things that we have already done inside of 'happy_hour.py' is we imported this random library, then the random library allowed us to randomly choose from a set of '(bars)', and to select one of them. So, what is that? Where did this random library come from? We don't have a file name random.py inside this folder, which is how I showed you how you would import a file before.

Well, turns out that 'random' is part of something called the Python standard library, and if you Google python standard library, you'll see what I mean and what the standard library is. The standard library, it's a set of different Python files, essentially, that you can import into any one of your files. So, searching through this page for the word 'random' will, to send you exactly to the random.py file, which is part of the standard library, and if you look through here, you'll see a whole list of functions that are included as part of this random library.

A library in Python is really just a collection of functions that have already been written for you, that you can import all together, and they're related in some way, such as in this case, the random library has a bunch of functions that somehow relate to randomness. For example, the 'choice' function isn't here, and if you read about it, you'll see that it just returns a random element from a list. But there's also a 'choices' function, which you can use to get multiple things from a list, or you can 'shuffle', is in here.

Shuffle allows you to reorganize a list in place, 'sample' allows you to; actually, sample is probably a better one for choosing a set number of things from a list, and 'uniform' as well is another one in here. And 'uniform' allows you to grab a random number between any two other numbers. So, if anything you want to do that revolves around randomness in Python, there's a good case that it's in here in this 'random' Python library, which is included as part of

the standard library, which basically means that when you install Python, you get all of these things for free.

So, why aren't they actually included by default? Well, there's a lot of them, and it turns out if you imported all of them, automatically every time you're in Python, it would take a few seconds, and that just slows it down. If you don't need all of this functionality, they don't include all of it by default. And then, in a second, I'll show you that you can even go beyond this and work with other libraries that other people have written, and I'll show you how you can download that.

There are two really interesting libraries, that if you go into here, you'll find them, but I'll show you what they do, actually, by opening up the Python interactive shell. So, there is one library called 'this', and actually, if you run 'import this' in your Python interactive shell, what you'll get is a nice, beautiful poem, that's actually just all that this library does. It prints out this poem, but this poem gives you a little bit of a sense of the philosophy of Python, and it starts with, 'Beautiful is better than ugly.

Explicit is better than implicit', and so on and so forth. There is another really fun one, so if you type 'import antigravity', no joke, it will actually open up the XKCD comic that I showed you before. It has become so famous that it's one of those inside jokes that is now included in Python as part of the standard library by default, and that's all it does, it just imports the webcomic, which I think is kind of cool.

****

Video 24 – Third-Party Libraries

Now, beyond just the Python standard library that comes with Python, it's also possible to download and use third-party libraries, meaning not ones that you've created, or that the Python developers, you know, themselves created, but other developers and they just want to make it open to the Python community. And these third-party libraries are all available and indexed on this website called the Python Package Index, which is at 'pypi.org', or 'pypi.org'. So, if you go here, you can actually search for all sorts of different projects and different libraries related to Python.

So, why might you need to extend the functionality of Python? Well, I'll give you an example here. So far, we've worked with all these different kinds of data types, lists, dictionaries, numbers, strings. But there's not really an easy way or a built-in way in Python to work with tables that are data. I sort of mentioned that you can do, represent rows, using dictionaries and lists, but it is still not ideal. So, there is a package called 'pandas', and it's one of the most popular data science libraries built for Python, but it's not part of the Python standard library.

So, if you search for pandas on this website, you'll actually see information about this pandas library, a lot, the description of it, the statistics along the left-hand side, and so on and so forth. We'll talk more about using third-party libraries because we'll be doing that a lot throughout the rest of this course. But this is an example of a library. And now, how would I actually install this library? Well, it is possible to download it and then put it in a right place on your computer, but an easier way is this command up here. So, you see how it says, pip or pip install pandas, I can copy that, or I can just click on this, and it copies it to my clipboard

automatically, and I would run this in the command line, and it will install this on my computer.

Now, interestingly, it won't necessarily just install it in this code folder that I'm currently in. It will install it into a special place on my computer, which will make it accessible into any Python file that I'm writing, it's is kind of a universal location, which is similar to how we've been importing other libraries like such as 'random'. It's on a place on our computer that's not in the code folder, specifically. So, there's actually an order when you're importing a file inside of Python, for where it looks for these files to import.

Now, the thing is, if I try to run here, 'pip install pandas' in the command line, it'll actually tell me that it's already installed. So, why is that? See, here it says 'Requirement already satisfied: pandas', and it actually tells me the location of where it's installed on my computer, interestingly, even though you're wondering where this was getting installed, and there's a few different packages here. And that's actually because this one library also depends on some other libraries.

So, there's you know, one might require another, and 'pip', which is this command that Python comes with, handles all of the stuff for you, the different version that you would have to install or whatever, so you don't have to worry about it. In this case, pandas is already installed, and that's because when we downloaded Python, we used the Anaconda's installer. So, the website Anaconda's, it allows you to download and install Python, but it also comes with some of the most popular data science libraries automatically included.

So, that's why pandas, for example, which is the most popular is automatically included in there, as well as some of the other ones that we'll be using throughout the rest of this class. But there will be others, such as when we're working with third-party APIs, where we will have to install, and that's why it's useful to know that this website exists, and the command that you can use to install a command. So, I would highly recommend still running the 'pip install pandas', seeing that it's already installed on your computer, and make sure the 'pip' command works, because it's possible that for a few people the installation didn't quite work properly, and they weren't able to get 'pip' working. As long as you could get that working, in the next video, I'm going to show you what pandas library does.

****

Video 25 – Importing the Pandas Library

So, I've included with this video, this file called census.csv, and this is a dump of data from the U.S. Census Bureau, for 2017, and it includes a lot of different geographies and information such as the population in 2017, the median age, total households, unemployment, just a bunch of really interesting data that it would be useful to be able to analyze. Now, of course, if you're familiar with Excel, you could probably use Excel to manipulate and analyze the data, but there are a lot of things that we can do in Python with this data, that would actually be very hard to do in Excel, and that will become apparent over the next few weeks.

So, how do I work with CSV files in Python? Well, it's hard to do directly with Python, at least with what Python comes available with, right off the bat, but as I mentioned in the last video, the pandas library allows you to work with CSVs in Python very easily, and I'll show you how that works. So, go ahead and download this file, and just move it into our code folder. Just so

that it's in here so that we can then write a Python script that works with the data. So, now that's in here.

I'll open up Atom, and I'll create a new file, and I'll call this, let's say 'reading_csvs.py', the exact name doesn't matter here. 'reading_csvs.py'. So, how do I read this? Well, I need to use the pandas library, which already comes installed, as I showed in the last video. So, I can just 'import pandas', right? That's just importing the pandas library. And so, what does pandas let you do? Well, it has a bunch of different functions, but one of them is the 'read CSV function'. So, I'm going to do 'pandas.read_csv'.

That's a function defined in the pandas library, and specifically, inside of here, this takes an argument, and that argument is going to be the final name. So, remember this was called '('census.csv')' that's the name of our file. Now, this file name has to match the file that we have, and it also has to be in the same location as this file that we're going to run for it to work. It's not going to be able to find the file if it's somewhere else on your computer. So, if I'm reading this, let's put this into a variable, such as 'data'.

So, 'data = pandas.read_csv', and let's just 'print(data)', and see what exactly we're getting. Are you ready? Let's run 'python reading_csvs.py'. Okay, so what are we seeing here? Well, it's a little hard to see because the command line is small, but if we expand it, and you scroll up, what we actually have here is a table; it's actually using Python and the pandas library to get the data in and format it as a table. So, we got our first row here, this is the row numbers, got the first column, there's '...' here because it's a big table, and it's kind of condensing it, and then it's giving us the last column, and then the values for it.

It's also telling us there are 917 rows and 13 columns. So, what the pandas library lets you do, now that we have the data imported directly into Python, is it basically introduces this new data type, this is called a dataframe, I think of it as a table. And over the next few weeks, we're going to talk about how we can interact with dataframes, really do some hardcore analysis on this data, using the pandas library as well as some other plotting libraries and things like that. So, essentially, we've just leveled-up the functionality of Python, and we're going to be able to do some pretty cool stuff with it. All thanks to this one third-party library.

****

Video 26 – Week 3 Recap

So, this week, we've covered a lot of advanced Python topics such as dictionaries, functions, and importing, from one file onto another file. So, we're going to be testing that out in this week's assignment, which reviews a lot of that, and it does get more complex over time. So, you might have to go back to the videos to re-watch certain things, in case you missed something.

Next week, I'm going to introduce Daniel Guetta, the co-teacher of this course, who's going to start talking about applying what we've learned in Python to data analysis, and we'll be coming back later in the course to talk about APIs and web scraping as well. So, look forward to that, and I think you're going to find this to be quite useful over the next few weeks.

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