Welcome to Module four, Part one and two. In Module four, we're going to learn how to organize data, introduction to list and tuples. In this module, we'll learn how to group related data using two essential Python data types, lists and tuples. Instead of creating dozens of separate variables, you'll learn how to explore and how to store and manage collection of values more efficiently. You'll also learn when to use list versus tuples based on their flexibility and mutability. Mastering these tools will help you structure your code like real world business systems. So far, we have learned how to work with individual pieces of data, like a customer name as a string, a price as a float, or yes, no flag as a Boolean. But what if you need to manage a wholesale item like list of customer or all sales from each month? That's where collections come in. Python gives us a way to group data together. Perfect for business scenarios like tracking monthly sales, customer list, or records. Instead of creating dozens of separate variable, collection lets us store and manage data efficiently. Lister like editable list, such as written in a pencil, you can add, remove, or change item, which is great for dynamic data. Whereas tuples are like permanent record, written in a pen. Once created, they don't change. It's useful for fixed data like dates or coordinates. Since lists are mutable, let's run a code to see if we can modify an element in the list for G. Here, we're starting with a list named G, and it contains a sequence of number. We first print the original list to show what it looks like before any changes. Next, we modify the element at index three. Remember in Python, index starts at zero, so this changes the fourth item here, which is four, and we want to replace it with the number 77. Finally, we modify and print the updated output and see what it looks like. So as I run my code in debug mode, you can also fix your Tony IDE as you like. I like to go on View and hit Variable, make sure the checkmark is there, and it's going to show me this little section where it gives you the name and all the values. So let's see what that will look like. So what I'm going to do is hit step I, and this will give you a lot more detailed analysis of the element. It's a lot more helpful when you're working with a lot more complex code, but due to best practices, I'm going to be running all our codes in debug mode. As you can see, I have all my list that appears on this site. It has all the values and it also has the name. Let's see what it would look like as I step out again. Notice that I was able to run my code and I get the original list as long as the modified list here. As you can see, the difference here is that it's replacing the number, which also changes the memories for Python. Tuples are immutable. Tuples cannot be modified after creation. Any attempt to change any elements will result in an error. The explanation behind tuples is that it does not support item assignment, making them immutable, this prevents unintended modification. Here's an example for a tuple as we run our code. We have a tuple named H. We store values like zero, 15, 25, and 35. So in the original version, we tried to change the value using H for index zero with four. So what we're expecting is to replace zero with four, and let's see if Python is able to do that. So I'm going to be running each line. And here we have our name and the values saved in Python's memory. And then let's ask for Python to print this result for us. And notice how we have a trace back error. And then also on the assistance side of Tony, it's telling you why it's not supported. So it says tuple object does not support item assignment, and this is a great way to furthermore debug your code. So this proves that tuples are indeed not mutable. So I'm currently on our module four interactive lecture, and I'm going to go to part two. And then we're going to be at exactly 2.1 creating list. And you can also run these code on Tony. But for now, I'm going to be running it on our interactive lecture module, and I will explain everything step by step. As I've said earlier, this is where Python saves all our memories, and sometimes you can notice that even when you run the code, it might not show up, but when you move to the second line, it may show the previous one. So don't worry about that. So let's run the first example. So a list can look different ways. It doesn't only have to be numbers. It can have numerical data, it can have text data. It can also hold different data types. Let's see what a basic empty list looks like. We can create an empty list and all we're doing is we're just saying Python that, hey, I'm creating an empty list, but don't worry, I'll come back and add the items later. Let's run that. So notice how we have a pending task, and this is a empty list, so there is no index or any position assigned to it just yet. And then I'm going to be running the next line, which basically focuses on numerical data like units sold per quarter. And then this is where it's positioned at index zero. This specific units sold per quarter, which is 18 50 would be position one or index one, and so on. And then As you can see, this is what our list is broken down into at index zero, one, two, three, or so on. And notice how Python was just able to save the memory. And then we're going to run this part which deals with text data, and we have something called product categories on our website. So let's go down and run that. Okay. And notice how Python also gives certain words a index position. For instance, electronics for our list starts at zero, clothing is at one, groceries at two, home good is at three. So notice how it hasn't saved the memory yet, but as I run to the next part, it will show up here. So we're also dealing with mixed data. Like here, it says list can hold different data types. It's useful for unstructured collection. For structured records like employee data, tuples are often preferred. So let's look at this

specific example. So we have miscellaneous info. Let's say we have someone named Joe Doe, and this could be their employee ID or sales ID. And have they made a sale true or, you know, what are they specifying on or what they're specialized on would be sales department. So let's try to run that and So you can see that it's a mix of different types of list. So it has John Doe, which is a number, which is also a string. And we also have our integers. We have float, we have Boolean. We also have integers here or sorry, we have strings here again, and they all have different index for each specific list. So how do we check the type? If you use a function like this, type with a closed bracket, this will show us whether it's a list or not. So what we're doing is we're using the print and F string, and then we're saying the type of units sold is, and then we're using the type, and then we're adding type sold within this bracket, and then we're closing it. So this is where we're going to get whether or not if it's a list or tuples. So let's try to run this line. And I'm going to go up and show you guys what that looks like. So I ran this line, and it says the types of units sold is. It's a class which belongs to list. So that's how you can check for type using the type function. So this is what our learning outcome looks like for the next few parts for Module four, and we're going to be basically covering Um, identifying why collections are needed instead of individual variable. Define and create Python list using square brackets, access list item using index and slicing for both positive and negative indices. Demonstrate that lists are mutable by updating items directly. Use list method like dot append, dot insert dot pop and dot remove to modify the content. We're also going to be applying list operation like n function with concatenation or multiplication in and not in. We're also going to be representing and accessing data using nested lists for table like structures and also understand when and why to choose list over other data types. Let's do a brief introduction to sequence and why order matters. A sequence is a data of structure that maintains the order of item, just like drawers in a filing cabinet. Each item is placed in a numbering spot starting from zero. That's why you can retrieve data using an index, just like opening drawer number two to grab February's report. All right. Let's look at the common beginner mistake in programming. Let's say we're trying to store monthly sales data, but instead of using one structure, we have created separate variable for each month like AN sales, Feb, or March sales, and so on, all the way to the end. At first, this might seem simple enough, but here's the problem. It's repetitive, it's hard to manage. What if we want to calculate total sales or average monthly sales? That means we would have to manually add each one, and that's not scalable or efficient. So imagine what happens when your new data grows, say weekly or daily sales. This approach would get messy real fast. Instead, we should store this data in a list, a single organized container where we can loop, calculate, and update easily. We'll see how to fix this using list in the next example. Now, let's further continue why we think it's a problem. Let's say you want scalability. That is not possible because here, if you're tracking sales every week, you'd need 52 different variables such as week sales one, week sales two, and so on, and that's just not practical. Next, the code becomes repetitive and hard to maintain. Writing out each variable manually takes time and increases the chances of error and also searching to forget it. There's no easy way to check if a value like, let's say, 4,800 exists across those variables. And that's when it becomes difficult when adding more data. So you'll need to create a whole new variable. That's not efficient. Lastly, there's no way to build a easy way to count, such as how many sale entries you have. You would literally have to count the variables yourself. All of this points to one thing. Individual variables aren't the right tool for the job. When working with sets or related data, list or other data structures solves these problems, and we'll see how. Now, let's look at a better way to store repetitive data such as using list also called a collection. First, a list acts as a one container that holds all your sales data in need organized group, and no need for 12 different variable. List allows dynamic growth. You can use the dotted pen method to easily add new data whenever needed. No extra variables are required. If you need to check if a certain value like 4,800 is in your data, just use the value in the list. It's a built in search feature that saves you time. What if you want to process all the sales? List lets you loop through each time without writing repetitive code, even if you need to count the numbers of entries. Just call Len list function, a simple and automatic way to track how many items you have. This is why lists are the go to tool for handling related sets of data in Python. Lists are incredible flexible container. You can store any kind of data and update it anytime. Think of it like a shopping list, you can change while walking through the store. You can store anything in a list, starting from number, name, or a mix of data. We can use type function to confirm whether or not the following item is a list. Accessing list elements, getting items by position using the index and slicing. We use the index to grab a single item, and we use slice to grab a chunk. Let's talk about an important feature of list. As we demonstrated earlier, strings are mutable. They can change their contents after creating them. That's different from strings which are immutable. You cannot change characters directly in a string. But with list, you can update any value using index assignment. For example, if you want to change the second item in a list, you can simply write sales Inside the bracket, you can put one equals to 5,500 and just like that, the value is updated. This ability to modify data in place makes lists super flexible for real world use, like correcting a sales figure or

updating product inventory. Let's run a quick code demo based on what we have discussed earlier. Let's start off with accessing list elements and getting items by position using the indexing and slicing methods. So here I have a very basic and simple example where we have our quarterly sales and our assigned indexes. We have both positive and negative. So let's say we want to see the first quarter sales, which would be at index zero, and we're naming it Q one sales, and then we're putting zero because index zero belongs to the first element or the first number, which is 4,500. And let's run this code. So So notice how for quarter one sales index at zero, we got 4,500. Similarly, for third quarter at index two, we got 4,800 for fourth quarter, which you can also refer with the negative indexing, which is at negative one, and it will start counting from backwards. It's also equivalent to the positive index for three, and then we get 5,500. And then we're going to be looking at more example when it comes to working with slicing our list. So as we already know how slicing method works, um, a quick recap is that when it comes to slicing, always remember what these following notation really mean. For instance, within the square brackets, I want my slicing to grab at index zero and immediately stop where index two ends, right? So we're going to be using the print and Esring function to see how that looks like. Just a quick recap, if we were to do that, it would start exactly here. And stop before it gets to two, which is 4,800. But rather, the results we're going to see would consist from zero all the way till one. So let's take a quick look at how that result looks when we run it. So notice how the first half would give us result like this and it shows up exactly how we want to show up using our print and F string functions. So that's how our slicing method is working. Similarly, you can also do a lot more such as slicing 1-3. You can also play around with your own list and see how that works. Let's look at this example. So the second half slice would mean that, you know, start at two and stop all the way in the end. So let's see what that would mean. Okay. So we're skipping all these. We're starting here, and notice how where it's ending. Our list ends here, so we're using this colon to demonstrate that, okay, just stop wherever the list ends or at default. And then If you want to slice something in the middle, you can do something like this where you can say start at one and at three and based off our list, one and all the way to three should look like this because we do not want this portion to get included. We want Python to stop exactly when it reaches at three, so this is what our result would look like. We're at 2.3, the power of mutability changing list continent place. So we have a code here, and basically, we have a list. As you can see, we have three team members here. We have Alice, Bob, and Charlie, and let's say that is our initial list for the project team. So what if Bob was moving to a different team and we're replacing him with David? How do we do that? So how we do that is, of course, try to remember how index work. So here we know that, um, If we want to replace Bob, so Alice is at placeholder zero or index zero. Bob is at one. So we would have to put project team, and then we have our square bracket, and then we put the index here. So the index for Bob. And then we basically use double quotation and put whatever we want to add in our list. Here, we're adding David. And then, similarly, let's say, what if, um, due to certain reasons, the system, let's say, only accepts all capitalized letter, and how would you do that? It's very simple. All you do is very similar to what we did for David. All you do is, um, find out what index Alice is at. So she's at index zero. And it's nothing crazy. All you have to do is just, um, put all uppercase, and if you want to show it in a more clean matter, you can also use the F string to just compare between the two. Let's just run that in our interactive lecture for the code platform. So here it's just breaking it down what we already talked about and how it's doing it. It stored the initial team name and then moving forward. Notice how it was able to replace David with Bob and everyone else is still remain unchained. Similarly, Alice has all capital letter in her name. Common list method for modifying our list. This is a very common method you'll be using throughout when working with list on Python. Pay close attention to each list methods when used as a tool for modifying our list. You can pause this video to understand what each method means in terms of the use of index item and the syntax. List also supports the standard sequence of operation we saw with strings, providing convenient ways to get information or create new lists based on existing one. Common list method includes dot a pen, followed by a closed bracket, dot insert, dot pop, and dot remove. Think of these list method like a tool in your organizer kit. Dot pen is like adding a new item to the bottom of your grocery list. It's quick and always goes at the end. Dot insert is like writing a forgotten item between two other items on your list. You choose exactly where it goes and the rest shifts down. Dot pop is like pulling a paper from the middle of the stack. You take it out and you actually keep it. It doesn't disappear and you get the item back. Dot remove is like crossing off the first item, let's say, soda on your shopping list, but be careful if the item is not there, Python throws it error. So I have copied the code from 2.4, and I'm going to be working with two specific list method. One is when it comes to adding item using dotted pen and dot insert, and later on, we're going to be working with removing item. So we have a list called daily task, and inside that list, we have a list of task, let's say, check email, prepare report. And we want Python to display that as morning task. And then it's going to be taking whatever items we have on the list from daily task. Now, let's say, what if we want to add some more item, and let's

say we want to add team meetings. And then maybe, you know, what if there's more items being added in the future? Let's say urgent call needs to be done the first thing, and what you want to do is like replace it with check email? So let's see how that would look like once we run our code. So we have the initial daily task, which includes check email and prepare our report. And then we're using dot appen to add meaning to our initial list. And we want that to be added to our daily task. So notice once we add the meeting, it has been added on this list. We have check email, prepare report, team meeting. So as I mentioned, what if I want to add urgent task? So what we do is we're placing urgent task at index three. So what happens is we have zero, which is check email. One would be prepare report, and then three would be the team meeting. But what it's doing is it's replacing team meeting with what we just added, which was review meeting notes, which comes before the team meeting, and that update has been successful. So let's take a quick look at this example. We have an inventory, which is our list, and within that inventory, we have laptop, mouse, keyboard monitor, webcam, and we also have mouse again. So, let's say, what if we want to remove something from our inventory? So Yeah. What I did is I ran this code and Python's memory has saved it with the initial list. Let's say I want to remove mouse. But we're not specifying which specific mouse we want. Would it be this one at index one or the last one? Since we're using dot pop and we haven't mentioned anything right within our close bracket, what Python is going to do is it's going to assume by default that you want the last item to get removed. If webcam was here instead of mouse, webcam would have gotten replaced. Let's run that So once we removed the item, Python stored that in its memory, and it said the last item would be mouse. And as you can see, once I run that, So it went from being laptop mouse keyboard monitor, it stopped at webcam. So let's run another one. So let's take a quick look at this example. We have an inventory, which is our list, and within that inventory, we have laptop mouse keyboard monitor, webcam, and we also have mouse again. So, let's say, what if we want to remove something from our inventory? So What I did is I ran this code and Python's memory has saved it with the initial list. Let's say I want to remove mouse. But we're not specifying which specific mouse we want. Would it be this one at index one or the last one? Since we're using dot pop and we haven't mentioned anything right within our close bracket, what Python is going to do is it's going to assume by default that you want the last item to get removed. If webcam was here instead of mouse, webcam would have gotten replaced. Let's run that So once we removed the item, Python stored that in its memory, and it said the last item would be mouse. And as you can see, once I run that, So it went from being laptop mouse keyboard monitor, it stopped at webcam something, but we call or we put the item name in the argument. Let's say we're using dot remove, but we're not specifying the index. We're just saying that I want to remove mouse. So how would Python do that? So let's try to run that code and see what kind of result we get. So once we do that, Python is not sure which specific mouse you want to remove. So what it does is it removes the first item. Now, what if we want to remove something that's not a part of our list? Let's say we want to remove an item called microphone. Notice how our initial list does not contain any item called microphone. So if we try to run that. So once we do that, we're going to get a trace bag and we tried the try and accept method. So if it cannot find the item, all it's going to say is could not remove microphone because it was not found. And you can see that in our shell, it says could not remove microphone because it was not found. So that means you cannot remove an item that's not part of your list. It will give you error. On Python. We know how to add and remove item. Let's talk about organizing them. Python gives us two handy tools to change the order of the item in a list using dot sort and dot reverse method. These don't create a new list, they update one you already have. As we go through the demo, watch how the list changes directly. So let's demonstrate this code where it uses the sort and the reverse method. So we have a list called product code, and we also have another list called sales figure. So these are the item that's inside my product code and sales figure. Pay close attention to this part as I run each code and see how it changes. So for the first part, all I'm doing is I'm using the sort code just to arrange them alphabetically, and notice how it just changes them directly. It just changes the position directly. And what I'm going to do is I'll try to rearrange this alphabetically from A to Z. So let's demonstrate this code where it uses the sort and the reverse method. So we have a list called product code, and we also have another list called sales figure. So these are the items that's inside my product code and sales figure. Pay close attention to this part as I run each code and see how it changes. So for the first part, all I'm doing is I'm using the sort code just to arrange them alphabetically and notice how it just changes them directly. It just changes the position directly. And what I'm going to do is I'll try to rearrange this alphabetically from A to Z. So let's run that and see how. So I'm going to go one at a time. And as you can see, when I'm hitting or when I'm running the first few lines, Python already stored that in its memory, and the shell will show a lot more detailed based off what we write in our F string. So I highly recommend to just follow along for this part, and then I'm going to run the first part. And I hope you notice how the product changed it alphabetically from A and then B, then M and comes in the end. But the initial one looked like this. Also notice how it changed the

arrangement within the list. It didn't create a new list, such as new product code or anything like that. Now, let's try to sort our numerical figure, which is the sales figure. And what we're doing is we're sorting it by ascending order and pay close attention to what the initial numbers look like versus once we try to change it using dot sort method. And notice now what we have is it's going from small figure all the way to the bigger figure. And then let's also try to print them descending where it goes from highest to lowest. Notice now we have 8,000.0 500.5 and so on and 150.2 comes all the way in the end. Let's see how reverse would work. We have this as A, B, M, and Z currently and adding reverse Basically, it does the opposite of what we had. So here's some common beginner mistake using dot sort. Someone might try to sort the result of dot sort in a new variable, expecting it to hold the sorted list. But remember, dot sort doesn't return anything. It just changes the list directly and returns none. So if you're writing something like, let's say, sorted underscore variable equals to sales underscore figure and you're using the dot sort, you're actually saving it as none into the variable. The sorting basically works, but it only updates the original list. So don't assign it to a new name, just sort it and then use the same list. Let's try to demonstrate that. I'm assigning dot sort to this variable, calling it sorted variable from our sales dot figure dot sort, and let's see what happened. Since we already ran the result and you can see instead of saving it as a new list, Python saves it as none. So always be careful when it comes to these common mistakes. Let's look at some common list operation such as N combining, repeating, and checking membership. LN returns the number of items in the list. It's useful for checking the size of your data or validating input. If you're trying to add two lists such as L one plus L two, it combines two lists into one. This is a quick way to merge data from different sources. List with the asterisk, and the number you put basically repeats the entire list and times. It can be handy for generating patterns or creating sample data for testing. Item in a list checks if a specific value exists in a list, returning true if it's found. Item not on the list does the opposite. It checks if a value is not present in the list. These operation make it easier to manage, validate, and combine data efficiently in Python. Python can store other list as elements, creating a structure similar to a Ty table. This is especially useful when working with grids, spreadsheet, or tabular dataset where data is organized in rows and columns. Each inner list acts like a row and the item inside represents a cell in that row. Can access any individual cell using two indices, the row first and then the column. In a business setting, imagine tracking monthly sale across multiple store location. Each row could represent one store, and each column could represent that store's monthly sales figure. Using nested list, it becomes easy to calculate things like total sales per store or company performance across different location, all with simple index and loops. This structure makes your data easier to manage, process, and analyze in real world business application. So let's look at a simple three by two table. We have our student name and grade. So how this works with Python is if you have a more complicated list which basically assembles a small spreadsheet or like a small table, what it's doing is it has its own row and column, and the rows and columns are also represented using index. So if you were to understand row, this is our first row, which would be for Alice, and for Bob is our second row and Charlie, our first row, sorry, third row. But in terms of Python, we're using the index method. So we start with zero, and then Bob would be one, and Charlie would be two. And how do we work with column? It's very simple. Similarly, what we did with row. Alice would be our column zero, and the grade that's assigned to Alice would be column one. Similarly for Bob, it's again, column zero, and then column one, zero and one, but the rows are like row two, row one and row zero. Reason why it's very important so that you do not get confused and you're making sure that you're using the right syntax, or it can give you error or it can be very confusing. Let's talk about accessing nested lists. So when it comes to working with nested lists, always remember we use a square bracket, so the row comes first and then the column index comes after. So if you get them switched over, you're definitely going to be getting a wrong result, and it can also make it very confusing. A remember that it's universal, so row and then column. Let's say we want to just get Bob's grade, okay. So what you're going to do is you're just going to say, let's call it a variable Bob's grade because we want to store it in our memory, and then we're going to call the list, which is student grades, and then we have our square brackets for row and column. So Bob belongs to row one and grade belongs to column one. And once we run that, notice how it gave me row one, column one, and his grade, which is 92. And what if we want to get Charlie's name? It's very similar. We just have to know which row Charlie belongs to. Here it's row two, and his name is at index zero for the column. So it's going to be zero, which gets us the name Charlie, and the row two is his grade. So see how that works two and zero. And once we run that, we get, we get his name. What if I want to get Alice's data? And over here, I have not specified any column. So don't worry. If you do that, there's a reason why we type it just with a row. So if you're just mentioning row and not a specific column, what Python does is it goes and gives you all the column that it belongs. So let's say if the column had more than grades, let's say had her attendance or the classes she was taking or, you know, how many times she attended specific seminar or so and so. If there were more information on her, it's going to pull up all those informations,

okay? So once I run that and notice how I got her name and her grade. And what if we want to update Charlie's grade? We can also do that. So how that works is, we're just going to mention the row, which is two, and his grade is at column one. So notice how we're putting it in the beginning, and then we're assigning the value 81. So once we run that, we have his new grade, and it got updated on Python's memory. So that's how you can work with nested elements. You can access different rows and columns. You can update or remove certain values as well. Let's do a quick summary of what we learned so far in Module four, part two. Throughout this module, we explore the part of list in Python. List lets you group related items into one container. So instead of handling dozens of separate variables, you can manage one organized structure. You learn how to access, change, and remove element using index and list method like dot pen and dot pop. You also use operator like Len, concatenation, and I to measure combine and search list. Finally, we covered nested lists, which are especially useful for representing structured business data like tables of students or inventory data. Mastering list will help your program to look cleaner, smarter, and much more scalable.