Here's the final part of Module four, where we take deep dive into tuples. We'll work through the coding example using both our interactive modules and Thai, explore some business use case for tuples and discuss common pitfalls to avoid along with best practices. Let's start by learning what tuple is. Tuple is very similar to a list. It's an ordered collection of item, but the big difference is that once you create a tuple, you cannot change it. Think of it like a birth certificate or a set of map coordinates. It holds important information that shouldn't be altered. You create a tuple by using parenthesis separating your item with commas. Here's a fun fact. Python is smart enough to make a tuple even without parenthesis, as long as it sees separating the items. But be careful. If you want a tuple with just one item, you must use a comma after it. That trips up a lot of beginners, so we'll pause and see that in our demo. So I'm on 3.1 for Module four, and we're going to be looking at creating tuples, grouping fixed data, and related data. Right now, what we're looking at is a empty tuple. It's possible, but it's less common because you will be seeing empty lists most of the time and not tuple because remember in list, we can add more items to it, but tuples are immutable, so it's very unlikely that you're going to come across empty tuple. So right now we have a variable called empty tuple, and I'm going to try to run this part. Notice how when I try to, um, work with empty tuple, it's nothing but just an empty, um, variable right over here. See, there's nothing in here, okay? So another example here is representing a fixed geographic coordinate. Let's say you want to explain or you want to, like, work with a data set where you have geographic coordinates and you want to assign these fixed latitude and longitude. So that will not change because if you're trying to change the latitude and longitude, it will not represent that specific geographic coordinate. It would mean it's at a specific location. So let's say for NYC you have this as the latitude, and this is your longitude value, and this is always fixed. Once I print that, tuple is going to store that information. Let's see over here. You want this item to be fixed because it's not like a list. You're not going to go back and change the value, it stays fixed. Again, similarly, product record, you have a specific product record that I'm assuming will stay constant throughout, and the name of the product directly relates to the product code. And also, if you're trying to keep the price fixed, you're also going to be using tuple. So we have our parenthesis and then notice we have double quotation after each item, followed by a comma. We're trying to print it as is product record. And then we have curly braces, and then we put the product record. And what it's going to do is it's going to take this variable, and it's going to be adding all these items inside the tuples. So that would look like this, as we already talked about, tuples, is that similar to a list. Tuple will also have its index position after each item. So this is our product code, which is at position zero or index zero. And then we start counting the next one, which is one or index one for the wireless mouse. And then for the price is at index two. And then we're also going to be talking about why parenthesis optional and is the key. So if we do not assign a comma, what Python see or it recognizes it as a simple value instead of a tuple. So let's say we have a variable point under score two D. Notice how we're missing the parenthesis, but we do have a comma, so this makes it a tuple. And once we have that, Python will automatically save to its memory that it is a tuple. So you see it says class tuple. So let's look at another common mistake that we make as a beginner when coding with tuple is missing the comma, just because it's a single item. If you miss out the comma because it's a single item, tuple will confuse it as a string or an integer, is the key. So let's see what happens if I miss a comma here. If you miss a comma, what Python stores. What Python does is it thinks that it's a string now. Now if you see here, Notice for this specific portion where we missed out on the comma, it just says class is string, which is this one. So that makes it a string. And then similarly for this case, which is it's working with just a number. And notice we're also missing a comma here, and Python is going to say it's an integer. So Python has saved it as a class type integer and Alice here would be an integer versus the one with a comma would be a tuple. Accessing the items in a tuple works just like a list and strings. This keeps the thing simple and consistent. You can use indexing to get individual items, and you can use slicing to get a range of item. For example, if we have a tuple representing a data like your month, and day, we can just easily extract each part using the square bracket syntax. We'll take a quick look at this in the interactive module. No need to use Tony here. So let's look at 3.2 accessing tuple elements just like list and strings. So similarly, how we work with list and strings, tuple also have its index position or indexes starting from zero, one, two, and so on. So we have a variable called event date, and within that we have some stored information. As you can see, we have year, month, and date. And what we can do is we can use the indexing method to get access to each items on the tuple. For say you want to get year only. So what we're going to do is we're going to be calling out this variable just the way it's written exactly here. Event date. And within our square brackets, we're going to put in our index number. So if we put zero, it's going to give us 2025. Similarly, for month, if you're again, putting the variable name here, and then within the square bracket, if you're putting one, you're going to be getting the month, which is four, and then for day, same event date, and then putting two inside the square bracket would give

us the date. Let's say, what if we want to work with the slicing method? And in this case, we want to get the month and date. So for that instance, we need to understand the indices. So for month we would only need one and two. So how we're going to do that is we're just going to call this a new variable such as month day tuple. And how we're going to do is we're going to be calling it event date, and we want it to start from one and stop at default, which would be from here, and the end would be all the way up till here. And that would give us the result for four and 15. And let's say, what if we want to get year and month? So the year and month falls in zero and one indices, and how would we write that as a code? Similarly, we're going to call this variable year month tuple, and we're going to call event date. And what we're doing is we're saying start from default and make sure you end before it hits two. So sometimes that can get confusing because you would think, okay, if I'm saying two, wouldn't it stop at day? No. If you, um put two, what it's going to do is it's not going to even go up till two, but it's going to stop here, which would give you the year and month and exclude date. Then you can see it says the return tuple result would be our year and month. That's how you access specific elements in a tuple. Once a tuple is created, you cannot change it. You can add items, remove items, or change an existing items. Why is it useful? Well, it helps to protect your data, especially if you're passing information around your program that should never be accidentally changed. It also makes your code much more predictable. If you try to change a tuple, Python will give you an error. We're going to step to this very carefully using Tony so you can really see how it works. So let's look at an example where we try to make some changes to our tuple to prove whether or not a tuple is immutable. So we have a database config, and within that we have our database, which is db.example.com. And we have this specific number. Let's say, we're just going to say that this is the number of, let's say, rows or tables in the database, and we have a username, which we call it read only users right now. And this is what the initial database would look like. So let's try to just run that. So we have all the information stored in. So let's say we're trying to make some changes within that database config, and it's since it's written in tuple format, we're trying to see if that changes can go through. So as we have been doing this throughout our coding practice, we're using the try and accept code block. So what T is doing is that it's saying that, hey, try to make this change. But if it doesn't work, just accept the error and display this message. In that way, the whole code is not crashing and you're not getting a trace back, and it's way more easier. You can just skip to the next line or figure out where you went wrong. So here, what I'm trying to say is for my database config at index one, which is right over here because we start at zero, I'm saying that change my let's say change my the number of table on my database, change it to, let's say, 9,999, and if it doesn't work, just give me the error. So let's try to run that. So it's trying to attempt to make that change. And as you can see that it didn't work, so it gave us the error received message. So it says tuple does not support this item assignment, and you can notice that it didn't work because Python did not even say that in its memory and it's still sticking to its initial value. So let's try to make another change. We're trying to use the dot Aen method. So what we're trying to do here is we're calling our database config, and we're using the dotted pen, and we're seeing within this specific variable we have for this tuple. Can you add another item? Let's say, new password would be the password for the database. So we're saying that, hey, can you add that right next to the read only user? And normally for list, this wouldn't have been a problem as we have seen our previous coding demonstration. So let's see if this works here. So it attempted to do it, but it has failed, and it skipped all the way to the last code. But you can see that it did not support that at all, and if it did, it would have been right next to the read only user. Let's see what it means for the next line. The next line is nothing but just trying to create a new tuple based on the old one, and in order to do so, you will have to create a new variable. For our case, we're calling it updated config. What if you want to update the config, but you want to keep specific information as it is, but you're just trying to make changes to one specific item. So let's say for read only users, I'm trying to make it admin users. So what I'm doing is we're calling it updated config, and we're using the same format for a tuple, which is the parenthesis and within that we have our DB config, which is the first database configuration, and we're using the index number using our square bracket and we're saying each time that, hey, bring me the specific item at index one and two, keep those as it is. But since I'm making this change, I'm not using the same method here. I'm just calling the item that I want to add. So the reason why this is a better way of doing it rather than manually typing each item because when you're running like a let's say you're doing this for a long, long code, it can get very, um, problematic because you can make typing error. Let's say, instead of, um, db dot example, you may have forgotten the dot or you may have made some typo error. It can seriously cause a problem with the configuration overall, and it can cause more errors with what you're trying to do, and you may not get the result that you wanted without even noticing. So try to, um, stick to this way when it comes to just adding a new item and keeping everything else constant. So you can notice that that update has been successful. So we have everything as it is, but just a new item has been replaced and added, which is the admin user, and it replaces

the read only user here. So that's all for this part. Even though tuples are immutable, there are still several useful operation we can perform on them. First, you can check how many items are in a tuple using the LN function, just like what we did with list. You can also combine two tuples using the plus operator. This creates a new tuple containing the item from both. If you want to repeat the item in a tuple multiple times, you can use the asterisk operator again. This creates a new tuple. Finally, you can check whether a value is present in a tuple using the in operator or check the value is not present using not in. These are simple but useful tools that allow you to work with tuples effectively, even though tuples themselves cannot be changed. We'll practice these operation in the interactive module. Let's look at a quick example. I had to copy the code and paste it on Tony because there was no IDE for the interactive lecture for this specific code. Let's begin running this. So we have a bunch of tuple here. The first one is tuple A, and it has the value ten, 20. Then we have tuple B, we have X and Y, and then we have something called point, we have 100 200, and we have something called status with double code, followed by comma. So remember that if we do have a comma, that's what makes it a tuple and not a string. So let's try to run this. And Python is saving everything in the memory. So what if I want to check the length for tuple A? So So the length for tuple A is two because we have two items here, ten and 20. Notice it's not one, two, three, four, it's not four because there's four characters here, but it's two because of the two item. And let's look at the length for status. So we have status, and let's see what the length is. So the length here is one. Notice active is one item. So for one item, the length is one. We're not counting each alphabet that's in there. We're just counting the item here. So let's try concatenation, which basically means adding two tuples together. So for concatenation, we have to call it a new tuple. Let's say we want to call it tuple C, and we want to add tuple A, which is ten and 20 with tuple B, X and Y. So let's see how that would look like. So once we do that, Python has stored that information on its memory. And this is what the detailed output looks like, ten, 20, X and Y. And then let's look at repetation. So let's say we want to repeat status. So for status, we have the word active, and it should repeat active three times because we have the asterisk followed by the number three. So let's see what that will look like. So notice how it repeated active three times. And let's see if we want to say that is 100 in point. So we do have 100 for point. We have 102 hundred. So since it's true, Python should return the value true. Is 100 point there. So it says it's true. And then we want to see is active in status. So for status, do we have something called active? That's also true. And then let's say we want to see is Z not in tuple B. So we're using in, right? So we have tuple B. Notice how we do not have Z. We only have X and Y. So once we run that, So is Z not in tuple B? That's true because Z is not in tuple B. The most convenient features of tuple is tuple unpacking. Tuple unpacking lets you assign each item in a tuple to its own variable all in one line. This is very useful when you're working with tuples that represent records, such as product information or coordinates. But be very careful. The number of variable must be exactly a match with the number of items in the tuple. If it doesn't, Python will give you an error. We'll look at correct examples and also an unpacking error together in Tawny, so you can clearly see both cases. This is an example of unpacking a tuple. So we have our variable called location, and we have two items here. We have the latitude and longitude. So what if we want to unpack it and give it its value to its specific variable. So for latitude, we have 40.71, and for longitude, we have negative 74.06. So how it looks like when we run. So we have our initial location values, and unpacking it would make it look like this. Similarly, we have a variable called employee data. So what we have here is our employee ID, full name, Alice Wonderland, department HR, salary is 70 grand. And then this is our employee data because we're putting in equals to sign. And we have four variable which matches the four items in our tuple. So let's print that out. So notice how we have all those items unpacked here. We have department, employee ID, employee data, full name. So what if we're trying to unpack an item here? Let's say we call it RGB colors, and let's say we have the specific item here. This could be like, let's say, the color palette code. And we're trying to assign the variables to these specific, let's say, color palette code for red, it's 255, for green, it's zero, but then we also have another zero here, and we're missing a variable here. So how Python sees that as so when we do that, notice how we got an error here because we have three items, and it's two variables, which means it's too many values to unpack. So therefore, there has to be another value here. Let's say I'm going to put blue here. And let's change that to ten, and let's try to rerun that part again. So you can see that it worked this time. Now, let's compare list and tuples. You can choose the right one for your data. Lists are mutable. You can hold, add, remove, or change their items. Think of lists like writing a, writing on a whiteboard. They're flexible. Whereas tuples are immutable. You cannot change them after creation. Think of tuples like something written in a pen or set in a stone. If your data needs to change, use list. If your data should stay consistent, use tuple. Keep these guidelines in your mind when deciding which collection to use. Let's apply this to some simple business scenarios. If I want to store fixed details about a report version, tuple is perfect. Those details should not change. If I'm collecting customer feedback scores and I expect to add more scores, a list is a better choice. For a transaction, I might use tuple to store

the transaction ID and timestamp, but use a list for the items purchased since the list can grow. We'll practice these examples together in the interactive module. All right. To wrap up this course, let's take a quick look at the final example. Here we have three different scenarios, and for them, we have three blocks of code. And you can see that the first scenario basically deals with fixed details. So if we're working with fixed details, we would need to work with something that's immutable. So in our case, that would be working with tuple. So let's say we have a report and the report consists of our metadata, which has, let's say, V 2.1 could be our version date. And then this is a specific set of data, and that's the username, and we do not want to make any changes to this item because this stays constant throughout. And once we run that, You can also see that we also use the unpacking method where we broke down each version. Let's say we have version here, we have data data generated, and then also the author. So that's something that we just learned about working with tuple. And let's say what if you were to collect customer feedback? So that means you need to work with something that's mutable. And in that case, list is a better option. So let's say we have something called feedback scores. And what if you want to add more scores or make any changes? So What if I want to add a new score? What I'm going to do is use the dotted pen method, and in that way, I can show you what that would look like. So we have our new score, and that gets added right over here. So that's how we're working with list for the second scenario. And what if you want to mix of both list and tuple? That's also possible. So let's say we have our transaction info. These will stay constant throughout. These are our permanent information. We do not want to make any changes. And there are certain things you might want to make changes. Let's say, what if you're adding more items to your inventory? Let's say you have laptop and mouse. What if you want to add a keyboard here? You can use the dot a pen working with list, and you can also add those items here. And you can see certain stuff, like I said, stays the same, like timestamp, those information you do not want to change. So let's run all that. So you see how I was able to add another item using list. So this is how we can wrap up this entire course working with lists and tuples. Here are some best practices and common mistakes to remember when working with tuples. First, choose carefully, use list for dynamic data and tuples for fixed data. Second, always use clear descriptive variable name. This makes your code much more easier to understand. Remember that tuples are immutable, so trying to change them will raise an error and be careful with tuple unpacking. The number of variable must match exactly. Also, watch out for the single item tuple mistake. You must include a comma after the item. Let's summarize. Both lists and tuples are ordered collection. Lists are mutable and flexible, great for changing data, whereas tuples are immutable, great for constant and records that should not be changed or altered. Understanding when to use each type will help you write better and a lot more predictable code. As you continue learning Python, you'll use both lists and tuples often, especially in business application and data processing. So practice both and get comfortable choosing the right tool for your job.