Module 02 - 072: Python - Tuples with Nested Lists

In this guide, we explore an advanced use case for tuples: **tuples containing nested lists**. Tuples are immutable, meaning they cannot be changed after creation. However, if a tuple contains a **mutable object** (like a list), the contents of that object **can** be modified.

Understanding this concept allows for more flexible data structures while maintaining the advantages of tuples.

Python Documentation: Tuples and Sequences

Creating Tuples with Nested Lists

A tuple can contain different data types, including lists. Here's an example:

```
post = ('Python Basics', 'Intro guide to Python', ['Altuve', 'Correa', 'Bregman'])

print(post)
```

The third element is a **list**, while the tuple itself remains immutable.

Why use this?

- We can **protect** the tuple's structure while still allowing changes within the list.
- Useful for **data groupings** where some values need updates while others remain constant.

Modifying Lists Inside Tuples

Although the tuple itself is immutable, we can modify the list inside it:

```
post[2].append('Springer')  # Adds an element to the nested list
print(post)
```

The new item **'Springer'** is added to the list inside the tuple.

Important: The tuple **itself** has not changed; only the mutable list inside it was modified.

Attempting to Modify Tuple Elements

Trying to modify a tuple's elements **directly** results in an error:

```
post[1] = 'Advanced Python Guide'  #  TypeError
```

Tuples do **not** support item assignment.

Instead, we can create a **new tuple**:

```
post = ('Advanced Python Guide', post[1], post[2])
print(post)
```

A new tuple is created, keeping immutability intact.

Replacing Lists in Tuples

Although we cannot modify tuple elements directly, we can **reassign the entire tuple** with an updated list:

```
new_list = post[2] + ['New Player']  # Creates a new list
post = (post[0], post[1], new_list)  # Creates a new tuple
print(post)
```

The list is **replaced** by a new one, keeping tuple immutability.

---

large

Figure 1: large

large

Figure 2: large

Practical Use Cases

Where is this useful?

- **Configurations:** Some values remain fixed, while others (lists) need updates.
- **Data structures:** Maintaining structured, protected data with modifiable sections.
- **Machine learning:** Feature sets where some parts are mutable.

Summary & Best Practices

Tuples are immutable, but **nested lists can be modified**. To modify tuple elements, **create a new tuple**. Tuples with lists **balance structure and flexibility**. Use carefully when mixing immutable and mutable types.

Video lesson Speech

In the same way, that python list can contain nested Data Structures tuples can perform the same task.

So extending our example here of our blog post what I want to do is be able to add a new element at the end of this post that contains a list of tags.

So, what we're going to essentially have is a tuple with a set of nested strings and then a list inside of that so let's see how we can perform this task.

I will give you a little bit of a preview. It is going to essentially be identical to when we simply wanted to add a new string element inside of the tuple so if you want to pause this video and try to do this yourself I would highly recommend that so that you can see exactly how it works and if you cannot figure it out then follow along in the solution.

So, I'm going to start off by creating a list here called tags and I'm going to add some strings inside of that list.

I'm going to say python, coding, and tutorial.

Now in order to update our post tuple because tuples are immutable and I know you may be tired of hearing me say that I say it so many times because it is one of the most key elements of working with tuples and so I want it to become second nature that you always remember the reason why you would use a tuple in the first place and that is because it's immutable and it can't be changed.

So, in order to update this tuple we're going to have to perform reassignment the same way we did with the string so I'm going to say post and then use our assignment operator of plus equals and then pass in a new tuple and this is going to be **tags comma**.

So, this is essentially doing the same thing as when we added our status of publish just like this and because we have tags stored inside this variable it's the same thing as if we performed this kind of addition where we simply put that entire list inside of this tuple and also as a reminder the reason why we have to place a comma after the tag is because if we didn't then Python would assume that we're simply trying to override the order of operations and it wouldn't recognize it as a tuple but now because we have then we're going to be able to concatenate these two together and override our post tuple.

So, now, if I say print we can access this post and you're going to see that it has the list as the last element here.

Now, because this is a tuple, we can access that last element.

So, let's say that we want to access the string coding.

We can start by simply using our bracket syntax and because it's the last element we can give the index of negative 1. If I run this you can see we have our list

large

Figure 3: large

and if we want the code then we can type 1 because this is the second element which means it has an index of 1.

And now if I run this we have access to the string of coding

So, **we were able to traverse the entire post tuple the same way that we would a list with a nested list inside of it**.

And so this is going to be something that you're doing quite a bit if you're working with large data structures in machine learning or if you're building out web or mobile APIs for Python applications and any kind of tasks like that.

You're going to be working with state data structures quite a bit so it's good to practice these so you can be familiar with how that works.

Code

```
# 02-072: Tuples with Nested Lists

post = (
    'Python Basics',
    'Intro-Guide to Python',
    'PostContent'
)

tags = ['python', 'coding', 'tutorial']

post += (tags,)

print(post)
"""
The tags list is "inserted" (reassigned) to the new Tuple:

('Python Basics', 'Intro-Guide to Python', 'PostContent', ['python', 'coding', 'tutorial'])
"""


# Accesing the second value for the list, which is the last value on this tuple

print(post[-1][1])      # coding
```