Chapter10

Welcome to Chapter 10, we're going to talk about tuples.

And in a way, you kind of already know tuples.

Tuples are a limited version of lists.

And the sum total of this is that tuples are sort of a more efficient

version of lists that you can't modify.

So they're really unmodifiable lists.

So here is just the syntax of tuples, but

it exactly looks like lists except we're going to use parentheses.

So here is a three-tuple.

The concept of tuple, the word tuple comes from mathematics.

Two-tuples, three-tuples, a two-tuple is a thing that has two things.

It's a way to call it.

So three-tuples, it's a set of three things, basically.

So this is a three-tuple with three strings in it.

Again, it looks exactly like a list, except using parentheses.

We use the index operator, x sub 2

We can use the constant syntax, we assign these things in. In a printout,

just prints it out like the constant syntax.

And it just has no square braces, but instead parentheses.

Things like max use tuples, it looks through a sequence of things.

List is a sequence, tuples are a sequence, a string is a sequence, and so

max looks through them.

And the for loop basically creates an iteration variable and

then bounces through the things that are in a tuple.

And so in a way, these are the same, right, the same.

Now there are differences.

The place where there are differences are that tuples are not changeable,

not, they're immutable.

And if you recall strings are also not mutable.

And so if we take a look.

If we have a list 9, 8, 7, and we can change x[2] to 6 and

then that changes the 7 to be a 6.

Lists are mutable.

Here we have a three character string 'ABC' and

we want to do character 2 and turn that to a D

with this assignment statement, and Python says no we're not allowed to do that,

that is a traceback.

And the same thing happens with a tuple where we want to do z[2] = 0.

And it says sorry, you cannot change, so these are not mutable.

So the difference is lists are mutable and then strings and tuples are not mutable.

But that's part of the efficiency of tuples is that they're not mutable and

that allows them to be stored more densely than lists.

And so there's a whole bunch of things that can't be done with tuples.

You can't sort a tuple.

Whatever order you put the tuple in when you create it, it stays in that.

You can't append,

you can't extend it, you can't flip it with a reverse, lots of things don't work.

And if you take a look at the directory output of lists and tuples, you see whoa,

it's a subset.

You can do count and index.

Count looks up how many things

match a particular value, index says where is a particular value, but

you can't do append or extend or insert, pop, remove.

Because all those methods inside lists that tuples don't

have are the ones that make changes to it.

So it's prohibited but it also

allows Python to be more efficient knowing that tuples are not going to be changed.

Lists it has to actually allocate extra memory and

stuff like that to let them be changed.

从 :3:2 开始播放视频并学习脚本3:02

So that's the point, it doesn't have to build these structures to be changeable,

and so they're just more efficient and more performant than lists.

从 :3:10 开始播放视频并学习脚本3:10

If you need a list, use a list, but if you can get away with a tuple we tend to

use tuples, especially if we're going to make a variable for

a brief moment, and then throw that variable away.

Now there's a couple of cool interesting things that we can do with tuples.

And one is you can put a tuple on the right-hand side of

an assignment statement.

Now they both have to be variables in this case.

Now that's very different than putting a tuple on the left-hand side.

If I put it on the left-hand side of x and y, it would be like oh, x is 4

and y is 6 and that's a 4, 6 tuple.

But if we put on the right-hand side,

it basically expects a tuple on the left-hand.

I mean if you put it on the left-hand side,

it expects a tuple on the right-hand side, and then does a one-to-one correspondence.

So it's the same as saying x equals 4, and y equals 'fred'.

And so that is sort of a simultaneous assignment statement. You could write this

with x = 4, y = 'fred', right, that'd be okay.

But you can do this and we'll see in a second.

And just whatever the order is.

Now this could be an expression or even a function that returns a tuple.

And you can then assign it into two variables at the same time.

Now if you do something like (a, b) equals 9, it's going to blow up on that.

It's going to be, I'm unhappy about that, because it expects if there's a tuple on

this side, there's got to be a tuple on that side.

And in this case, if I just put 9 over here, there is no tuple on that side.

Now we've played with this actually already without even knowing it.

And so the way this works is if you remember d.items,

d.items gives you a list of tuples.

So this dictionary has two things, csev maps to 2 and cwen maps to 4.

And so if we say d.items we say give me the tuples, right?

And so the tuples come back. So this is a list of tuples.

Now what happens is that is how the two-iteration variable for works.

And this a list of two-tuples.

从 :5:10 开始播放视频并学习脚本5:10

And so what happens is that the first goes into the k and

the second goes into the v.

This is like an assignment statement for this first tuple.

Then it runs the loop and then the second one gets put in the k and

the v and then it runs the loop again.

And so that's kind of how we construct these two-iteration variable

for loops using tuples.

Because that turns out to be a tuple assignment each time through the for loop.

从 :5:36 开始播放视频并学习脚本5:36

Now an interesting thing that tuples can do is we can compare them.

Remember, we can compare strings.

And so the way tuples work is it does the comparison

starting at the left-most one and compares it.

So if it's asking the question, is this less than?

Well, 0 is less than 5 and at that point it does not look beyond the 0 and the 5.

That's like the most significant element of the tuple, like the most significant digit of

a number.

If 199 is less than 201.

If there's a 1 there and a 2 there, then the rest of these numbers really don't

matter because that is the most significant digit.

从 :6:12 开始播放视频并学习脚本6:12

And so that's how this works, it doesn't look at these.

On the other hand, if the first one matches, and it's saying is it less than?

Then it has to look at the second one.

It says, oh, 1 is less than 3, it never looks at this one and

it doesn't look at that one, but it gives us back True.

The same is true for strings, it's doing string-by-string comparison.

It compares these two

and they're the same and so it has to look to Sally and Sam.

And then it looks at Sa, well, they're the same, so Sa.

And then at some point goes oh, there's the l and that m, and so

the l is less than m and so that's why this entire thing is True.

Jones, Sally is less than Jones, Sam.

从 :6:52 开始播放视频并学习脚本6:52

If, on the other hand, the first two don't match, Jones,

Adams, then it never looks at the second piece whatsoever.

It doesn't bother because the J is greater than the A, so this thing becomes True.

So it's sort of within strings

it looks at the first pair, and then within that if they're not the same and

then goes to the second pair, third pair, whatever.

And so it's pretty clever.

So that's one of the nice things about tuples.

And we're going to use this in a moment to sort them.

从 :7:18 开始播放视频并学习脚本7:18

And so you can think of tuples as being sortable by their first item.

And when the first item matches, then it looks at the second item,

So if we go back to looking at the items, what comes out.

Remember that dictionaries aren't always the same order.

从 :7:33 开始播放视频并学习脚本7:33

But if we look at the dictionary items, d.items, then we got a tuple a 10, c 22, b 1

Now what we can do is we can say, hey, we would like to make a sorted copy of this.

And so we pass this in to the sorted function.

And we get back a sorted list.

But it sorts it based on the first thing.

Because in dictionaries, you can't have two keys.

You can't have duplicate keys, right?

You can't have two a's or two b's.

And so it sorts it based on that.

If they matched, it would look at the second one.

But in this particular case, because it's coming from a dictionary, these keys

are going to be unique because the very fact of constructing the dictionary,

you can't put the same key in more than once.

You can put the same value in more than once, but

you can't put the same key in more than once.

从 :8:20 开始播放视频并学习脚本8:20

And so sorted basically is this function that takes a sequence, and it sort of works

through the function and gives you back a list that is the sorted version of that.

And so we can write and compose this in a for loop as follows.

So sorted. d.items would give us the items sort of unsorted.

But if we say sorted and pass in d.items then

we have a sequence that is sorted by key order.

So k and v are

going to go through this dictionary in key order, not value order, key order.

So this is a way to say I want to loop through this dictionary in key order.

从 :9:2 开始播放视频并学习脚本9:02

And so that's what's going to come out.

So it's going to print out

a. b, c regardless of the order inside the dictionary.

We sort it before we loop through it and

so that's how we're going to get this thing in key order, but not value order.

Now we're going to quickly get to how do we sort by value?

Okay, so that's how we sort by key.

That's really easy, but it takes a little bit more work to sort by value.

But we're going to be using tuples.

So here's how we do it.

从 :9:28 开始播放视频并学习脚本9:28

So if we think of these, we can get key-value tuples and

all we have to do is make value-key tuples.

And if we make a tuple where the value is first and

the key is second, then we're okay.

So what we're going to do is,

we're going to create a for loop that's going to go through these items

in not particularly any order in this case.

We're not using sorted here, so it's not in any order.

So k and v are going to work their way through these key-value pairs.

And what we're going to do, is we're going to make a new tuple but

we're going to reverse the order of what we would get from items.

In that the value is going to be the first item, or the zeroth item in the tuple, and

k is going to be the second item.

So if we print this out, they're not particularly sorted at this point but

we have the values and the key.

Value, key, value, key, right? Value, key, value, key.

从 :10:17 开始播放视频并学习脚本10:17

Now what we can do is we can say, oh, this now is just a list.

It's a list of tuples and if we tell it to be sorted,

then it's going to compare these things and then sort, and when those match.

Now, the values, there can be duplicates.

So if there's a matching value,

then it will look to the key as the next thing to sort based on.

And so we're going to call sorted again.

And then we're going to pass in tmp,

which is our little list that we just made of two-tuples flipped value-key.

And we're going to give an extra parameter, say go backwards.

So sort from high to low.

reverse=True, that's what that does.

You just add that as a parameter, reverse=True to sorted.

And then we print this out, so it's now 22, 10, 1.

So it's from big value to small, okay?

So that's how you get a sorted by value instead of by key, okay?

从 :11:8 开始播放视频并学习脚本11:08

But we have to make a temporary list.

So tmp is a list because we have to append to it.

But each of the things in the list is a tuple.

So we end up with a list of tuple, tuple, tuple. Okay.

So this again is data structures.

We're trying to create shapes of data that help us solve our problem.

So now let's actually solve a problem.

The ten most common words. In a previous chapter, we did the most common word.

But now we want the ten most common words.

So here's how we do it. So, we've done this before.

We open a file, we make a dictionary.

We loop through all the lines of the file.

We split the words in the file.

So we got file looks kind of like this, right.

We loop through the lines of the file.

And then for each line, we loop through the words.

That's the outer loop and the inner loop.

And then we do this counts.get(word, 0 ) + 1.

And that creates our little set of our histograms based on each word and

histograms get taller.

And then now what we've got is at the end here,

right here, we've got the completed histogram in no particular order.

So now what we're going to do is we're going to extract the data out of that dictionary,

that histogram data out of the dictionary, and we are going to make a reverse tuple

of the value-key word tuple instead of a key-value.

So we do what we just did on the previous slide.

We write a for loop and I'm doing it a bit differently here.

We're going through key-value but

then I'm going to make a tuple that's value-key, put it in a variable newtup.

This is a tuple assignment that's going in there and

then I'm going to append that to the list.

And so we end up with this list just like on the previous page,

a list of tuples in value-key order.

Now, I'm like, okay let's sort that backwards.

从 :12:57 开始播放视频并学习脚本12:57

So this is the value-key tuples, go backwards, and

then we reassign this result as sorted back into the lst variable.

And so when this is done, it will be sorted from the highest

value down to the lowest value. And now we have to do two things.

Look for the top ten, and we have to flip them out when we're printing.

So this list is now backwards.

It's a v comma k, value comma k.

And we're going to use list slicing.

So we're saying, start it at the first one and go up to but not including 10.

So that's 0 through 9 which is the first ten.

And you'll notice that this val and key are flipped.

And that's because in it, value is the first piece of each tuple and

key is the second piece of each tuple.

And so it's going to go through successively, value, key, value, key,

value, key, value, key, value, key.

But it's only going to go through the first ten because we have limited

this with list slicing.

And then we print it out in the order we wanted to see it in in the first place.

So we see kind of a flipping of the order between how we

iterate through the data structure and then how we print this stuff out.

And that gets us back to the key-value, which is what we want.

从 :14:8 开始播放视频并学习脚本14:08

And so that's quite a sophisticated thing.

We're using dictionaries, we're using lists, and we're using tuples and

sorting and that's a nice little piece of code.

Again, I encourage you to understand every single line of this code.

So this next slide is not something I'm insisting that you understand.

But some of you might find this kind of fascinating.

从 :14:35 开始播放视频并学习脚本14:35

So I can do effectively this bottom part here, not the calculation,

but this bottom part right there.

I can do this all in one line of code

从 :14:46 开始播放视频并学习脚本14:46

that takes the dictionary and

then ends up with the sorted tuples sorted by value order.

And this is it, let you look at it for a second.

It uses a part of Python called list comprehension.

And so the tricky bit here is this syntax in the middle.

And this looks like a list, and it is a list.

The difference is, instead of expressing the list with appends or

as a constant with commas, we're expressing the list as an expression.

And I like to read this as, right here,

just to say the word "for all", create me tuples that are v, k.

And you'll notice this order is I got value-key, v, k tuples for.

So this is like there's a little loop that happens here.

It's going to stamp out a bunch of these little tuples.

Stamp, stamp, stamp, stamp, stamp, stamp, stamp.

And it's going to stamp enough to go all the way through for

each k, v that's in c.items.

So that's the key-value in this dictionary, 'a' :10, 'b' :1, 'c' :22.

This is going to run three times, it's going to make three tuples.

But this list is going to have them flipped.

So this little bit right here,

this little bit in the brackets, is the same as this bit right here.

从 :16:3 开始播放视频并学习脚本16:03

It is basically the same as doing that except it's expressing it in a more

direct but slightly more complex way.

And so, this is a list that's computed on the fly.

It's a three-tuple list flipped with values and keys.

And then I pass this in, right into sorted, to the sorted function.

So that's sorted function parentheses.

And then I take what comes out of sorted, oops.

从 :16:31 开始播放视频并学习脚本16:31

I take what comes out of sorted, which is this, and

I pass that into print. And I forgot to put reverse equals True on here.

So it comes out but not reversed.

It's a value-key but not reversed, so you get the idea.

从 :16:46 开始播放视频并学习脚本16:46

And so if you're really,

really interested in this, go read up a bit on list comprehension.

It's kind of a interesting and powerful way to make even more succinct Python.

But I don't spend too much time talking about it.

Because if you see this version of it and

you feel like that version of it makes a lot more sense to you, stick with it.

If this version makes a lot more sense to you, then use it.

I mean once you get good at it, both of them sort of equally speak to you.

But it's probably easier if you're certainly beginning to use more

simpler primitives.

And here,

once you're more sophisticated, using sort of these more powerful primitives, okay?

So that's a quick run through tuples.

We talked about how they're pretty much like lists.

You can't change them. You can compare them, they're sortable.

And you can have the two values that get into an assignment statement.

And we can now sort dictionaries by key and value by constructing a data

structure, i.e., a list of flipped tuples, and then sorting that.

And we use that to sort of, sorted is there for us, and if we can set the data

up in a way that sorted does what we want, then we get the data that we want.

And we're really taking advantage of data structures.