Chapter 8.2

So now we're going to talk about some of the operations that you can do to lists.

We've been talking about their internal structure but

now we are going to start working with them.

So the first thing that we see here is that we can add them together and

it's just like concatenating strings.

We concatenate lists together.

It pretty much works the same way.

从 ::24 开始播放视频并学习脚本0:24

Adding a and b to make c does not hurt either a or b.

Slicing operator, again think of strings.

Think of slicing.

Think of beginning position up to but not including so

sub 1 is that position right there up to but not including sub 3 which is 41, 12.

So that's what we get there.

Just like with lists you can omit the first one so that's up to but

not including 4, which is this one right here so this is the 0, 1, 2, 3, 4.

You'll mentally have to count that up to but not including 4.

That's that one.

3 to the end so 0, 1, 2, 3 that's 3 to the end so that's 3, 74, and 15.

And then, just like with strings, you can have the beginning and the end.

从 :1:15 开始播放视频并学习脚本1:15

So again it's very much like strings.

Slicing of lists is like slicing of strings except

that the list is slicing entire cells within the string.

Just like in strings we have lots of library methods,

there are lots of methods in the list object.

从 :1:31 开始播放视频并学习脚本1:31

append which adds a thing to the end of the list.

count which actually, len tells you how many things are in a list.

count tells you how many things that match a particular thing are in a list.

extend, index, insert, pop, remove, there's a whole bunch of things and

they're all very well documented in the Python documentation.

从 :1:50 开始播放视频并学习脚本1:50

So we'll use a couple of these things.

There's a couple of ways to build a list from scratch.

We're calling this, this is constructor and so list is a predefined type.

This is the type of list.

So we say, make a brand new list, it's the same as saying, give me an empty list

and then assign it into stuff. So that is a list with nothing in it, so

the length of that is zero.

And then we say let's append the string book and then let's append the string 99

and we print it and now we have a book with two items, and

so you can sort of do this over and over and over and append things if you like.

And you can keep on going.

We can append the string cookie.

So at the end of cookie, you've got three things in there.

book, 99, and cookie.

So append is a method.

Now, recall that strings are not mutable, so

all these methods would return variables.

But lists are mutable, and so appending actually changes this variable.

So note that stuff is different from here to here because we appended it, and

that's because lists are mutable, but not strings.

And so they kind of function a little differently, and

just when you learn strings, and you learn that the return value.

从 :2:57 开始播放视频并学习脚本2:57

So, for example, you don't want to do something like x = x.append.

Because the return value that you get back here, the residual return value, actually

messes x up.

So you do not want to do that. You just say x.append and that works a lot better.

in. Again, one of my favorite Python operators is in.

In strings you're asking if a character or a substring was in another string.

Here now you're looking at the members of the list and

you're saying is 9 somewhere in the list?

Yes, it is.

Is 15 somewhere in the list?

No, it is not.

And then not in is also an operator.

It's the opposite of in, so is 20 not in the list?

And the answer is 20 is not in the list so we get back True.

从 :3:43 开始播放视频并学习脚本3:43

That's just a question that you can ask in any old if statement that you want.

从 :3:48 开始播放视频并学习脚本3:48

Lists maintain order.

The order that you put things in is the order that you get them out.

So append, append, append.

Whatever you've appended, it comes out in exactly the same order.

List maintain that.

They're kind of dense and compacted, and

you can only add to the end, although you can say, push all those ones down and

put something in the middle using insert, if you like.

But it also means that we can sort them.

And so, here we have three things.

Joseph, Glenn, and Sally.

And we say, hey friends, sort yourself.

This is kind of more object oriented.

We call the sort method within the friends object.

And then now, it has been changed.

Right? So it actually changes it.

And so it's now in alphabetical order, Glenn, Joseph, and Sally.

And friend sub 1 is the new sub 1 and so

this actually changes it because lists are mutable.

从 :4:37 开始播放视频并学习脚本4:37

We have lots of functions.

We've already played with len and talked about len and

how many things are in the list.

We can say what's the largest number of a list?

What's the smallest number of a list?

Add them all up.

What's the sum of this list?

Sum of the list is 154 and you can calculate the average by the sum.

Now, some of you who will go back to Chapter 5 and

say, well why did you give us all those for

loops to do the same thing instead of just telling us this function?

So the answer is, well there are reasons for

knowing how to construct those for loops.

And they were good, easy-to-understand for loops.

If you have a list and

it's a list of numbers, these are far better than constructing a for loop.

So all those examples from that earlier chapter,

they're not the way you would do that particular problem.

But sometimes you're reading through lines in a file and

you have to do it a little bit differently.

So here's a couple of loops that sort of demonstrate this notion

of how you might use these functions to calculate an average versus

how we calculated an average in the earlier chapter.

Although I kind of redo this in a different way because now I'm pretending

I'm reading from a file but instead I'm going to read from input.

I'm going to read the numbers dot dot dot until I get a done.

And I'm going to calculate the average of those numbers.

And so it's just a standard loop where we set total to 0 and count to 0.

I'm going to make it a while loop, if I was reading a file it would be a for

loop but this is, I'm going to ask for some input, prompt it.

If it's done I'm going to break out of the loop so that handles this case,

so I can get out of the loop somehow.

And then I calculate the floating point value of that input, and

I add it to my running total, and I have my running count, count = count + 1.

And then this runs you know in this case three times, and then it comes down here,

and computes the average and prints it out, and away we go, okay?

And so that's kind of the manually constructed loop with a manually

constructed counter and a manually constructed sum and away you go.

Now, if I were try to use max and min to accomplish the exact same thing.

从 :6:36 开始播放视频并学习脚本6:36

So what I would do then is I would accumulate these values in a list.

So I'm using a data structure now rather than just logic.

And so this is kind of an algorithm that's carefully constructed to produce the count

and the sum when it's all said and done.

Now what we're going to do is we're going to use the data structure to solve

that same problem.

And so we'll say hey,

let's make an empty list because we're going to accumulate some numbers.

And then we have the same while loop, same two lines, we look for

actually the same three lines.

We get a value, we check to see if we're done, we get out, and

we convert it to a float.

But now instead of doing the summing as we're doing here,

we say let's just remember that.

And we've got this list, we're going to fill it up, we're going to put numbers in it.

So we put a 3 in, we put a 9 in, we put a 5 in.

And then, put a 3 in, put a 9 in, put a 5 in.

So we don't, we're not actually calculating any calculation.

And then when we hit done, we come out, and we say, oh, let's calculate the average.

Well it's, you've got these numbers in a list now.

We have a list that looks like [3, 9, 5], that's what's in here.

So we take the sum of that, and then divide it by the len, and

we have the average.

And this same program produces exactly the same output, okay?

So that both these things produce the same output.

And you can decide for yourself which you think is more elegant.

从 :7:55 开始播放视频并学习脚本7:55

In general, there is one tiny difference between these two.

从 :8:1 开始播放视频并学习脚本8:01

And that is this one actually has to have

all of the numbers in memory before it can actually do the calculation.

This only keeps one number in memory.

And that probably doesn't matter if there's like a hundred thousand numbers or

less, But if there's like a hundred million numbers, this one doesn't

use a hundred million spaces in memory and this one does use a hundred million.

So there are sizes of data that are large enough that it matters,

the memory usage between these two things matter.

because this requires two variables for any number.

This one here requires two variables for

any numbers of things we're going to do the average of.

And this requires an extra bit of memory for every single number.

So it scales based on the numbers.

For the small stuff that you're doing it hardly matters.

And if you've already somehow have these numbers in a list

this is a great way to do a sum.

You wouldn't write the for

loop to do the sum. If you've already got numbers in a list then add them up with

a sum or figure out the maximum using a function, that's just perfectly fine.

So that's kind of manipulating lists and writing some interesting loops.

And up next we're going to play with connecting lists and strings together.