Chapter 8.1

0:08

Welcome to Chapter 8. So in Chapter 8,

we're going to take a little, things in a little bit different direction.

Up to now, we've doing,

we've been talking about and building code

to understand what we in computer science call algorithms.

Algorithms are the paths that we call,

write code to go certain paths.

We go here and we go over there.

We go back up and we come this way.

We check to see if it's this. We do this that and the other.

Algorithms are like GPS routing for code.

Data structures are structured

variables and up to now we've been using very, very simple variables.

We've been using variables that are like x equals or y equals or whatever.

But there are more complex ways to use memory, use the variables.

There are, there are collection variables and so

this is the first time we're going to really start

talking about ways to use variables differently.

And the design of your variables,

the design of your data structures,

how you organize your data while your program is running,

can help you solve your problem.

So you have a set of steps,

and you have an internal shape of data.

And lists and dictionaries and tuples are

the first three data structures we're really going to talk about.

So these are collections,

where we have more than one value in a variable

and so before we talk too much about

collections,let's talk about what's not a collection.

So in this little example, x = 2, that finds a piece of memory.

Python labels it "x" and sticks a 2 in it.

Then later it says x = 4. It finds that same piece of memory,

overwrites the 2 with a 4.

And so when it's all said and done,

what we have is a 4.

Now, a list is a collection,

meaning that in a single variable we can have more than one thing.

And so in this we've actually,

we've actually been using lists all along.

When we see the square bracket syntax we've been using

these in for statements, etc., etc., etc.

This is a list. Square brackets is a list constant.

But what you're noticing here is I can put three things, three different strings,

Joseph, Glenn, and Sally,

into a single variable by enclosing them in this brackets.

And so it's still a single variable.

friends is no longer a string.

It's a list of strings, which is different than a string.

And the same for carryon: socks, shirt, and perfume.

So you can think of it as, sort of

like a piece of luggage with a name, x or y.

But then you can put multiple things in it.

And the interesting thing about these data structures that we'll use for

different purposes is how you put them in and then

how you get them back out will be the distinctions between

these three different kinds of collections that we're going to talk about.

So like I said, list constants are anything in square brackets.

So that's a list of three integers: 1, 24 and 76.

This is a list of three strings: red, yellow, and blue.

You see it can be either kind.

We just use commas and we use square brackets to start them and stop them.

You do not have to have the same type of data in a list.

Here is a list that has a string with red,

an integer of 24, and then 98.6,

which is a floating point number.

And it doesn't even have to be,

we won't do this too much for a while,

but this is a three-element list.

The first element in the list is an integer.

The second element in the list is another list.

It's lists within lists,

it's turtles all the way down its inception.

So you can have lists that include lists.

And when we talk about data structures in a more advanced way,

having a list that includes another list or

a dictionary that includes lists is just part of the design.

Now, we're way, it's way premature to talk about why you would do this but it's possible.

So this is a 1,

a list, and a 7.

And so there are actually three elements in,

in this particular list.

Okay? And you can have an empty list.

And so that is list constants are these square brackets.

Now like I said, we have been using lists already.

We have for loops and we iterate the i through 5, 4, 3, and 2, and 1

to print these five things out which means that for is going to run

this once for each value in the list

and i is going to take on the successive values in the list.

And like I said, we have already been doing that.

Right?

You know, we've been doing that since Chapter 4, or

Chapter 5, or something like that.

So lists and definite loops,

the kinds of things that we've been doing all along,

are best friends. The in.

So you could think of a list as like a set.

It's a set of three strings and in is like for all the members of this set.

And so "for friend in friends" and remember,

the fact that I use plural here and singular here does

not mean anything to Python. It means something to you as you're reading the code.

That's a mnemonic variable.

It can be anything you want.

As a matter of fact, this is exactly the same code.

z equals Joseph, blah, blah, blah,

and for x in z, x

takes on a successive value.

Python sees these two things as completely equivalent.

But you as a human being read them as something different.

And so that's the choice of,

reminding us of the choice of variable names.

So these collections, there's putting stuff into the collection

and putting multiple things in the collection but there's also getting them out.

And so, we use the index operator,

the same operator we use to look inside strings,

to look inside lists.

So I, I read this in my mind as sub.

So this is friends sub 1.

Now remember that these,

using the elevator scheme in Europe technique,

the first one is 0,

the second one is 1. And so friends sub 1 is actually the second item in

the list and that's how Glenn ends up getting printed out here.

So you can look inside lists.

We can change lists and the word that's used in

Python speak is called mutable, means changeable.

It can be changed.

And so you'll notice that some things are not mutable.

So like strings.

So we put in banana in fruit.

This seems like it would change that first letter to lowercase b.

We've got sub 0, fruit sub 0 equals b.

And unless you saw this traceback, you know,

you might think, ah, this is no problem.

Syntactically Python could actually support it but it doesn't

because we are not allowed to change this string.

And that's part of the reason that when we do things like dot

lower on strings we always say it makes a copy of

the string and gives us back a lowercase copy without changing

the original string. And that's because strings are not mutable.

Item assignment is another word that Python uses to talk about this.

Item assignment, so strings are not mutable.

Okay, but on the other hand,

if we have a list, 2, 14, 26,

five integers, and we print it out and then we say lotto sub 2 equals 28.

Well, that is this thing,

so that puts 28 into that position.

And at this point the list is truly changed, right?

So we print it out. The original 26 is long gone.

It's just like any assignment statement except that

we can assign into a piece of the list.

We can put something into a piece of the list.

Lists have length, just like strings We use the exact same function.

Python knows the type of its parameter and the length of the thing that we're looking at.

And it says, oh, I know what that is.

In this case it's a string and it's nine characters.

And if we say, oh, here's this list and this is another list that

not all the types are the same it's one, two, three, four.

How many things are in there? Tell me about that.

Four things are in there. So len is,

you can pass in various sequences

and we'll use them for dictionaries and tuples as well when we get to that point.

The range function is kind of a weird

special function that we use mostly when we're constructing

loops but the range function returns a list, returns a list.

And so it returns a list 0 up to but not including.

Sound familiar? 0 but up to but not including the number.

So range 4 gives us four numbers.

But they start from 0 to 3.

And the reason for this is that they can then correspond to

the index position either in lists or in strings because zero's,

you know, a three-element list is

0, 1, 2, up to but not including 3.

And so, if we have a three-item list

here and we say how many things are in there, it's 3.

And if we say print range of len friends so this len friends is 3.

And so then this range is 0 through 2.

And so we can use these to construct lists

if we want to have a counted variable. So that the best way to,

sort of, go through a list is this one here.

It's a nice simple determinant loop that basically has

an interation variable and it's very clear if all we want to do is print them out.

But what if we somehow want to know where they're at relative to the list.

We want some i variable.

You could say i equals 0,

use a while loop, i equals plus 1.

You can do all that stuff. We've seen that,

those loops in previous things.

Or you could use a for loop.

But instead of having the iteration variable go through the actual elements of the list,

we have the iteration variable go through

the numbers 0 through 3 and so that's what this is.

So range here is 0, 1, 2.

And so then i is going to go through 0, 1, and 2.

And then, when we want to look at the particular item of the list we

use just use the index operator sub 0, sub 1, sub 2.

So these two lists produce the exact same output but in this one we know,

during each iteration of the loop,

we know what i is. And there may be

reasons that you want to know that. You might want to print out

oh, I found something at the third position.

Which you wouldn't be able to see here unless you added like i = 0,

i = i + 1. So this is a way to create what we call a counted loop.

So instead of it being sort of a loop that just goes through a set it is counted loop so,

because it's going through a range of numbers.

Okay. So up next,

we'll continue talking about some of the things that you can do with lists.