Chapter6.1

0:08

Hello and welcome to Chapter 6.

Now we're going to talk about strings.

This is really the last chapter that I'm just going to ask you

to please learn something without exactly knowing how to do it.

We're just sort of chopping food.

It's like you're going to be a chef eventually,

but we're just chopping food.

So Chapter 6 is the last chapter that you just have to learn how to chop food.

We're going to actually make a meal in Chapter 7.

Once we have a file, then all of the things that

we've learn how to do are going to come into play.

So just trust me and listen for one more chapter.

So we literally have been using strings from

the very first moment because the first thing we did is print Hello world, and so,

you know, this is a slide from a couple of lectures ago.

And so, you know, we take two strings,

double quote, single quotes,

we use the plus, remember it looks to the left,

looks to the right, concatenates,

remember that doesn't put a space there.

And here's a string that has digits.

And now we're going to try to add 1 to it, and it blows up.

Yeah. You know, it's hardly even – you're not

really sad when you see traceback by now, hopefully.

You're just like, oh, traceback's a normal thing. I'm trying to learn.

TypeError: cannot concatenate strings and integer.

It's trying to tell you what's going on. And we're all good.

We then take the string,

we pass it to the int function,

and then that comes back with 123,

and we add that and it becomes 124.

So it's all good, right?

It's all good, and we've been doing that for a while.

Another thing we've been doing is reading data from input.

The input function prints out a prompt. That's a prompt.

We type something, and then whatever that is comes back as the result of the function,

and then it gets stuck into name,

and so if we print that – print, of course,

is also a function -- we pass name in, and we get out Chuck.

Even if we enter some numbers, like 100,

right, that doesn't make apple a integer. Apple is a string.

Input gives us back a string,

so we can't subtract 10 from it. Traceback again.

But if we can convert it to an integer and then subtract 10,

then we can get that 100 minus 10 becomes 90.

So we've been manipulating strings and using

internal functions and converting them to floats and doing this,

that, and the other thing as we have gone forward.

But now, we're going to start tearing apart strings.

So, the ultimate thing we're going to do is read through a bunch of data,

tear that data apart,

read line by line, and then look at each line and find things in the line.

So we need to know that a line of characters, many characters,

which turns into, would be a string,

a multi-character string, has indexable data within it.

So, the string banana – and I didn't come up with banana.

Actually, the book that I use is based on a book by two people,

Allen Downey and Jeff Elkner,

and one of those two came up with banana.

I would never have come up with banana because I don't know how to spell banana,

and I'm terrified of having a slide or the book with

a mistyped banana because I just think

somewhere in banana there's supposed to be two n's.

But, and I read this and it looks like a misspelling to me,

but I'm pretty sure that's right.

But that's neither here nor there.

We have this string banana,

which is six characters, and we stick it in fruit.

And if we look in fruit,

you can actually pull each character out.

We call this the index operator,

and the square brackets are the index operator.

And I pronounce this sub,

so that's fruit sub one.

Now, as we look at the index,

the first one is zero.

Now that's counterintuitive – it goes right back

to elevators in Europe that have zero as the first floor.

Right? Zero's the first floor,

so Python was invented in the Netherlands –

that's Europe – and so all the elevators are zero,

so the first thing is zero.

Actually, that's not the reason at all.

The reason has to do with performance in computer science,

where zero is easier to add than subtracting one, but whatever.

Just remember: the first thing, the second thing,

the third thing, fourth thing is sub zero.

So this is a six-character string,

but the last position is position five.

You'll get it; it won't take you long.

It will seem natural pretty soon.

Right now, it seems unnatural.

So fruit sub one,

that means the character in position one.

So, a ends up in letter,

and so we indeed can verify that.

This thing inside of the brackets can be an expression,

it can be a variable, it can be anything you

want. There's a constant, here's an expression.

x is 3, x minus 1, that becomes 2,

so fruit sub two comes down here to x and we see the n that comes out of that.

So, that's the index operator.

We pronounce it as sub, you know,

fruit sub x minus one is how I pronounce that last little bit.

And so, it wouldn't be Python if we didn't have a traceback,

and in this one, I'm making a mistake.

And that mistake is I'm going beyond abc,

which is zero, one, and two.

And so, zot sub five.

No, sorry. Python is angry at us.

Python is angry at us,

and so we get an index error. IndexError: string out of range.

Oh, well, I mean string index – well this is the string, that's the index.

That's the word. We're doing an index operator,

a look-up operator, or a sub operator.

So, that's just a thing you're not supposed to do.

After a while, you kind of get used to

the idea that Python is just going to traceback on you from time to time.

There is a number of functions.

We can pass a string into the len function,

and we can get the length.

The length of this is six characters.

It is indeed six characters;

even though it's zero through five,

it's still six characters.

So the len is just another function,

we talked about functions before.

Functions take as input some parameter,

so fruit is assigned into banana, and then we're doing this.

Remember, we evaluate the right-hand side here.

This fruit gets passed into len,

so the string banana is passed into the len function.

The len function does something in the middle of it,

and then the len returns us a 6 with the return statement,

and then that 6 goes – that's an integer 6 – goes to x,

so then we print it out and we get the 6.

Okay, so len is a function, takes an input parameter, and away we go.

And so inside of len,

there is some code that takes this.

It's got a for loop or who knows what's in here,

and then it's got a return statement,

and then it returns the stuff which then

replaces this as the residual value in the expression,

and then the assignment statement finishes,

and then 6 ends up in x,

and away we go.

So, there's lots of things that you can do with strings. Asking how long they

are is one of the things that you can do with a string.

Now, we want to loop through a string.

Well, given that we can have this index operator sub,

we can then generate a sequence of numbers zero, one, two,

three, four, five, and then we can look up all of the things, right?

And so to do that, you know,

we got fruit banana and then index – this is

our iteration variable – and we're going to construct a loop,

where we're going to add 1,

we're going to increment index by 1.

And then we're going to say while index is less than len of fruit –

and that's 6, not 5.

And this will give us the numbers 0 through 5.

So this loop will run with index being 0 through 5.

So the first time through it's 0,

the second time through it's a 1, then a 2,

and then we're going to take the sub zero letter and stick it in the string letter.

Sorry, letter is a bad choice of a variable.

This could be x, as long as this were x, it doesn't matter.

It's just letter is a reasonably

mnemonic variable unless you're trying to give a lecture.

The letter is letter. The letter gets assigned into letter.

So if I just said x, x,

then I would say it looks up the letter at

the position zero and then puts that letter into the variable x,

and then we print out the variable x.

Sometimes mnemonic, but there you go.

So that's going to run six times

zero, one, two, three, four, five, six,

and each time it's going to print out

the index and the letter that happens to be in that string at the index.

So now we've got a loop that goes through each of the letters in a string.

Now, that was the indeterminate loop.

We had to construct it.

We had to make our own iteration variable, etc.

etc. etc. But a much more convenient way,

unless you actually need to know the position,

just if you want to go through all the letters in a loop,

a much more convenient thing to do is just use a for – a determinate loop.

Right? So, we're going to use for and in.

And remember, in is like, you know,

member of, for all the letters in the set fruit, but in this case,

it's for the iteration variable letter

taking on all the successive values of the characters of fruit,

so letter's going to be b,

then a, then n,

then a, then n, then a.

And that means it's going to run this loop six times,

and each time through, letter is going to be something different,

and so it just prints this out.

And we didn't have to construct any of that index stuff or any of the fancy stuff,

we just rock and roll our way right through that.

Here's those two loops that I just showed you.

Right? Here is the determinate loop with the for and the in;

and it's nice and clean and they produce it.

Here we construct index,

have the while loop, use len,

add 1 to index, pull the letter out,

and so this line is the same as that line.

So this is kind of like five lines of code or four lines of code,

and this is like two lines of code.

And this might not seem like much,

but there's so many places that you can make a mistake here,

you know, if this is like index + 2 or something.

Now you do have more flexibility when you're

constructing it this way and sometimes you do have to construct it,

but to do the exact same thing,

these two things are doing the exact same thing,

and so it's always better to use a more succinct and direct way of describing

your code rather than this more

this is like showing off how good you are with the while loop,

but it's sort of unnecessary.

So use the simplest bit of code that you can use to accomplish what you want.

It's easier for you to write,

it's easier for you to debug,

and it's easier for someone else to understand as they're reading your code.

So we can go back to the iteration chapter and think of all the things that we did,

whether it was look for the largest,

look for the smallest,

see if something's there.

What this is going to do is this is a simple loop that's

going to go through and see how many a's are in a word.

Now we happen to know by looking at it,

but it gives you the sense of of iteration.

So we take, you know, letter's going to to take on b a n a n a.

It's going to run this code six times.

And if the letter is an a, we do count = count + 1.

We set it to 0 at the beginning.

Remember how these loops do something at the beginning, they do something in the middle,

and then they have kind of like the payoff at the very end.

And so this just means every time the letter's a,

we're going to add 1 to count,

so this effectively is counting the number of a's in the word banana,

and out comes 3 because there are 3 a's.

Now if I misspelled it, there would be more n's and more a's,

but luckily on this slide,

I think it's spelled correctly.

Now, I love this in,

and we're going to use this to do

a lot of things when we deal with files, when we deal with lists.

This idea that in is kind of like

this membership notion in algebra. Not that you have to know algebra,

but if you do know algebra,

it's like for x such that it's a member of this set.

That's the concept of in.

It's a very clean abstraction.

Maybe you'll actually learn Python,

and then you go back and learn algebra and you go'll like, Oh, yeah!

This little member guy,

that's kind of like an in statement, in statement in for.

It's a very abstract concept that really says this is

how we're supposed to just run this loop six times,

you know, one, two, three,

four, five, six, do it.

Take care of all of the small details for me.

Right? And this is again for me is the magic of the for loop,

it's the Python for loop is – the for itself,

the for loop does a couple of things.

It decides how long the loop's going to run,

when the loop starts, when the loop stops,

and it advances the iteration variable automatically,

so it decides, am I done?

Go get the next letter, run it. Am I done?

No, go get the next letter,

go get the next letter, go get the next letter, go get the next letter.

Oh, now I'm done and I'm going to quit.

Right? And so, the for takes all of this,

all that logic is in one statement.

And like I said,

the less code that you have to write,

the better off that you are.

Now that I showed you how to loop through strings.

I want to show you ways that you don't have to loop through strings.

And so, one of the things you do in strings is

you basically want to grab a piece of the string.

And so this is what we call slicing.

And we're going to use the same square bracket to do slicing,

except that we're going to put an expression in that tells us how far to go.

So here we have a string, 0 through 11.

Remember they start at 0.

And so, in here,

instead of saying s sub 0,

which would be the first character,

we say s sub 0 : 4.

And so this gives us a range.

Now, the key thing here is the end is up to but not including.

OK, up to but not including.

So when we say 0,

start at 0 and go up to but don't include 4,

that says up to but don't include 4.

So we don't include 4.

Now that, again, may seem counterintuitive,

kind of like zero starting is counterintuitive,

but I'll bet you'll see that there are times when it

sort of makes sense to do up to but not including.

So for now, just remember up to but not including.

So if we go 6 to 7,

well, 6 starts here,

and then up to but not including doesn't include the 7,

so that's why we get a capital P. And then,

if we do 6 through 20,

6, starting at 6 going up to – you'd think this would be a traceback,

but it's not a traceback.

It is okay.

After a while,

you're like, "I'm a little disappointed in you, Python.

You're supposed to traceback every time I make a mistake."

Well, somebody decided it was okay to reference beyond the end of a string.

And we'll forgive you,

it's not going to get anything, it's actually going to stop there,

and that's why we get Python as the answer here.

Now, given that the beginning and the end of the string are

a very common thing you want, you want a prefix or a suffix off of this string,

it's really common to either eliminate the first character,

which means beginning of string,

or eliminate the second part of the range,

which means the end of the string.

So this basically says 0 up to but not including 2, so that's Mo.

And this one says 8 through the end, which means thon.

And then you can eliminate them both and so it means the whole thing.

Why do you want to do this? I don't know.

I say it's syntactically there just for completeness.

So up next, we're going to continue learning how we can manipulate strings.