**w03-01-ConditionsAndLoops-2024**

0:01  
All right, so it's 5 minutes past the hour, So we normally start at 5 minutes past the hour.

0:09  
Thanks.

0:11  
So this week we are picking up with the next part of our syllabus.

0:15  
So you remember last week we introduced variables and we looked at basic variables and we also looked at containers.

0:22  
So these two types we looked at were a list and a dictionary.

0:27  
And so this week we're picking up with loops.

0:29  
So we're going to look at looping over things.

0:32  
So we can loop over a text string, or we could loop over one of these container types that we talked about last week, or we can loop over a series of numbers, which is essentially treated a little bit like a list in terms of how the programme works.

0:47  
OK, so today's lecture is going to be a bit of a show and tell in that I'm going to go through some ideas and then I'm going to flick between the slides and some code so we can describe it and pause and discuss.

1:02  
OK, so conditions and loops.

1:06  
Now let's just do that for now.

1:10  
What of it?

1:12  
So with conditions and loops we can start to build a programme that's a bit more meaningful.

1:18  
You would have noticed last time that we could create a variable and add some numbers together and multiply some numbers.

1:24  
But unless you have some sort of conditions, you can't handle any sort of logic.

1:29  
OK, So what are we talking about?

1:31  
Well, with logic conditions we want to test if something is true or false.

1:37  
Now we can combine lots of particular logic conditions, single ones, into a big logic condition.

1:45  
And we do this with logic operators.

1:47  
So for example, we can ask is this thing true or this other thing true?

1:53  
Now with computers and logic in general, not the English language, or means one or the other or both.

2:01  
It does not mean one or the other only.

2:05  
If you want one or the other only, you have XOR.

2:08  
XOR means really just one or the other one and means that you have to have both.

2:15  
So if you say I want to have this and this, the computer will say it's only true when both are true.

2:21  
And then the last thing that you often use is not.

2:24  
So you are saying when this is not true, that's what I what I want to do, whatever I want to do.

2:30  
So you can build up a series of cheques that are combined together with OR and XOR or not as you want.

2:39  
So the examples I'm giving you here are when a programme has reached a specific state.

2:44  
So for example, if you're running a computer programme, it might be running and running and running until somebody presses Q on the keyboard.

2:52  
So you could say is Q pressed?

2:55  
No, carry on is queue press yes right, Stop the programme.

3:00  
It could be, as I've said, when user provides specific input, you could ask a user what to do next other than the queue idea and the computer has to react to that.

3:11  
It could be when you're reading a file.

3:14  
So imagine you've got a data file and in that data file there is some bad data.

3:20  
So you carry on reading and then you find that in one column something is wrong.

3:25  
It isn't a number, it's a text string and you can't make it into a number and you need to tell the user something's wrong.

3:32  
So you have a condition that you check, and if that is true, then you tell the user there's a problem in this particular column and row area.

3:41  
OK, so that's the idea of logic conditions.

3:44  
So I was going to have a quick look at a logic condition.

3:47  
So hopefully we can flick over to the code and I'm going to make my screen a little bigger.

3:54  
Come on.

3:55  
No, I want to go bigger, right?

3:58  
So a little bit bigger, hopefully.

4:00  
Is that readable for the people at the back?

4:02  
Or should I make it bigger still?

4:06  
People at the back, is it all right?

4:10  
OK, So we'll try.

4:11  
We'll see if I can avoid cutting it off the screen.

4:15  
Now I will come back to the end the idea of coding style.

4:19  
All my examples are PEP 8 compliant, which is the standard Python style which we will be checking that you're using in the exam.

4:28  
So what have we got in this?

4:30  
We've got a very simple logic programme.

4:32  
So we have a variable here called X and we've given it the value three.

4:36  
That means that the variable X is now an integer whole number, and it contains the value 3.

4:41  
Now I've done this because I'm then going to play with the variables.

4:45  
You might say, well this is silly, we know it's 3.

4:48  
So why have we got a logic condition?

4:50  
Well, in a normal programme, not a toy example like this, you've taken the value X from somewhere else.

4:56  
It could be coming from a keyboard input or a file or somebody has provided to you in some other manner.

5:02  
So I've just set this equal to three purely so I can write a simple programme.

5:07  
And then we're doing a test here which says if X is less than four and greater than 0, we print out the text small value.

5:17  
Otherwise you print out out of range.

5:20  
So you can see if we run this thing, what's going to happen.

5:22  
It's pretty obvious it's going to print out small value, which is what you expect.

5:29  
It reaches this point now, as we said last time, and also in the lab, it's a good idea to use the debugger to understand what's going on.

5:37  
You can set a breakpoint.

5:39  
So we can set say, for example, a breakpoint here and we can say run and debug, run and debug Python debugger the current file.

5:54  
And then once the debugger started, it will stop where that little red dot is.

5:59  
The red dot is called a breakpoint.

6:01  
All debuggers that I know, I've used this idea of a breakpoint, lots of different languages.

6:06  
So essentially you're telling the debugger stop where that breakpoint is.

6:11  
And then I'll carry on from there.

6:14  
So let's move this down a bit so you can see that we've stopped here at that line, the sort of yellowish triangle.

6:23  
I know it's hard to see on the screen.

6:24  
This is where the programme execution is.

6:27  
So we can hop forward and yeah, we find that the programme execution comes in here, which isn't unsurprising because this variable X contains 3.

6:37  
So there is a simple logic condition.

6:39  
Does anybody have any questions or comments about the logic condition before we go on?

6:45  
Yes.

6:56  
So, yeah, So what I've done, OK, So what I've done here is I've told the debugger run this file.

7:04  
Now if I didn't have this, this is the breakpoint, it would run the file until it hit an error or it finished.

7:12  
So by putting this breakpoint here, when I tell the debugger to run, it goes to line 10 and then stops.

7:19  
So it reaches that break point.

7:21  
Now if you have a more complicated programme, imagine one that's calling another function or something, you can put the break point anywhere you like inside all the programme files, and it will run until it either reaches an error, finishes, or reaches your break point.

7:35  
And once it's reached the break point, you can then step forward as you wish.

7:39  
So this is the step forward.

7:43  
And in this case, yeah, it's just printed that and it's finished.

7:46  
All right, it's a good idea to get used to the debugger.

7:48  
At the moment, the programme is simple, but once you write a more complicated programme that has a bug, it's far easier to debug with a debugger than printing out lots of things.

7:59  
All right, OK, so let's go back to the slides.

8:05  
Yep, that's that one.

8:09  
All right, so we've got several types of loop, and I'm just going to talk generally in terms of a loop, and then we'll look at the two types that are existing in Python.

8:18  
In other languages you find slightly different loop types as well, which aren't in Python, but the main couple that are in Python are available in other languages.

8:28  
The idea of a loop is very simple.

8:30  
You want to repeat a series of instructions.

8:33  
So you have essentially got series of programme lines.

8:36  
It could be one line that you're going to do, or it could be many lines, but you just want to repeat them for as many times as you need.

8:45  
So I've got a little graphic here to conceptualise what's going on.

8:48  
You've got some actions and then you're going to come back to the beginning and there's a question, do I carry on?

8:55  
So that's our loop condition.

8:57  
So there are two loops, 2 loop types rather 1 is that we're going to step through a list of values.

9:05  
Basically we're going to step through that list and we're going to use each of these values in series.

9:12  
Now this is what a for loop is.

9:14  
So if you can think about a deck of cards where you have some fixed number of playing cards and you're asking the dealer, can I have the next card, can I have the next card, can I have the next card?

9:25  
And then the dealer finally runs out of cards and the for loop stops.

9:29  
That is basically what a for loop is doing.

9:31  
It needs to be given something which contains a fixed number of things.

9:37  
So the for loop is just going to keep giving you them until it doesn't have any more.

9:41  
So what does this look like?

9:43  
We can go back to our code here, go back here to the for loop.

9:50  
So here we have a list, and the list in this case contains 3 elements.

9:56  
So what's going to happen with our for loop is it's going to take each number in turn.

10:01  
It'll start from this first number two, and it'll then take four and five.

10:06  
And it'll put the number into this new variable called number.

10:10  
So what's happening here is the numbers is the container, is the list, and it will go around each time putting one of these values into number.

10:19  
And then at line 9, we're just going to print it out.

10:22  
We could do anything we wanted At line 9, we could have several lines that are indented.

10:29  
You notice that they are indented, and this is Python's way of saying it belongs to the loop.

10:35  
It's inside the loop.

10:36  
The structure here is you've got a colon and then you've got an indented one or more lines of stuff to do in a repeated manner.

10:46  
So if I debug this again, so I'm going to put my break point here, I'm going to run and debug it and debugger, I'm going to rundown.

10:56  
Yeah, get rid of all this gobbledygook and it will stop where we wanted it to stop, which is there.

11:05  
So if we hover over numbers, you can see that the numbers list contains 3 values.

11:11  
Now if you can look at the output on the screen, it's telling us that our position 0 or offset 0, there's the number 2, then 4, then five.

11:21  
So what's going to happen is the lift, the for loop is going to take it from the top.

11:25  
You know, imagine the playing cards again, you're just taking from the top.

11:28  
So hop forward and what can you see?

11:31  
You can see in this variable number, there is now the number two.

11:36  
We go round and now if we hop forward again, we can see we've got the number four and we hop forward once more and what happens?

11:46  
We have the number 5 and then we've now run out of numbers.

11:51  
So we've run out what's in this list and so the programme stops.

11:55  
So next time it's going to try and get a number.

11:58  
There isn't 1, so it finishes.

12:00  
So the exit condition here is there's just nothing left to pick up.

12:05  
Does anybody have a quick question about a for loop before we go on?

12:11  
Nope.

12:11  
OK, so let's go back to our loop time.

12:16  
So we have a for loop, which as I say, is really useful for a fixed number of items, rather like cards in a deck of cards.

12:26  
While loop is the other common type you've got in Python.

12:29  
The idea of a while loop is that we have a condition where we carry on looping.

12:34  
Now this condition could be anything we like, which means that a while loop is more useful for when we don't know exactly when we're going to exit.

12:44  
Imagine you've got a programme and it's running all the time and you think the user might want.

12:49  
Maybe it's going to stop today, tomorrow, next week.

12:53  
You don't know.

12:54  
So the programme is just going to keep doing its thing until the user presses Q.

12:58  
So there's no fixed number of things you're iterating over.

13:02  
And this is where a while loop is really useful.

13:05  
You can use a while loop for iterating over a fixed number of things, such as a deck of playing cards, but a for loop is better for that.

13:12  
So here we've got a couple of examples.

13:15  
While a user input has not been provided, rather like my pressing Q.

13:20  
So the user hasn't done the thing yet.

13:22  
You don't know when they're going to do it.

13:24  
Or you basically are just going to keep reading some data from some input stream until something happens and then you're going to do something else and you don't know exactly when that's going to happen.

13:36  
So while loops.

13:37  
So let's have a quick look at while loops.

13:40  
The structure here is a little bit different, so here's another while loop.

13:47  
So in this case we've got a condition.

13:49  
So we're saying while I is less than three, we're just going to keep looping.

13:56  
Now you have here a condition that you're testing.

14:00  
So if you wanted something to loop forever until the programme shut down completely, you could put here while true and it will continue to loop forever.

14:11  
Obviously in this case what's going to happen is I is 0, so I will be less than three.

14:16  
We come in here, I becomes bigger so it's I = I + 1 and then once I is equal to three we exit the loop.

14:25  
OK, so we can do again a debug here just to convince ourself we have got something working great.

14:49  
Get rid of the gobbledygook.

14:50  
I think it's because I've left the debugger running before, right?

15:12  
So yeah, here you can see it's printed out 01 and 2:00.

15:16  
So I need a break point, and we'll put the break point there, and here we are.

15:32  
So we've got the value of 0 in I.

15:34  
And then we're gonna hop forward and print, increment the number, increment the number.

15:41  
Now you can see I is equal to 2, which is fine.

15:45  
And then what is I?

15:46  
I is 3.

15:48  
Then this point, we exit the loop.

15:50  
OK, so that's a while loop.

15:54  
Great.

15:56  
So let's go back to some other concepts and ideas.

15:59  
Now we've got a loop and we can repeat things either a fixed number of times, the for loop, or we can keep repeating until some condition, which is the while loop.

16:09  
Now if during that execution of the loop, we don't want to carry on processing.

16:16  
So imagine here you've got a series of steps when you're doing step one and two and three, and then you get to Step 3 and you say we've found what we need, let's just go back and loop again.

16:27  
So that's what the word continue does.

16:29  
So you're going through the loop, you reach some condition inside the loop, and then you want to continue.

16:36  
And then that means you go back to the start of the loop.

16:39  
Now this is really useful for where your loop has got a series of steps and you don't want to process all the steps.

16:46  
You know, if you've already made the decision, I know what's going to happen next, you can skip the rest of the things inside the loop.

16:54  
It's quite helpful as a programming strategy so that you don't need to have long if, else requirements.

17:01  
So you could have if this, if else, this, if else, if, else, if, else.

17:05  
Else.

17:06  
Now instead of that, you can have if something continue and then you're going to skip everything else.

17:11  
So you're only going to go past the if into the next part if the condition isn't true.

17:19  
So it can be used to reduce nesting.

17:23  
Now nesting here has nothing to do with birds and trees.

17:26  
Nesting is where we have an indented thing.

17:30  
So that means we're inside a loop, so we're indented.

17:33  
Now you can imagine that you've got a condition inside that loop.

17:37  
So then you have one indent because you're inside the loop.

17:39  
You have another indent because you're now inside an if.

17:43  
And then you might have another indent because you're inside another if.

17:46  
Now if you start having large numbers of indents nesting, then it becomes hard to read like where am I?

17:55  
Imagine you're in a busy St.

17:56  
You're like, excuse me, where am I?

17:58  
I've no idea.

18:00  
And that's what happens to developers.

18:02  
If, if you've got a large amount of nesting, it's just hard to read, hard to understand.

18:06  
So it's better that you try to think about using continue where possible because one, it reduces the execution time and two, it reduces the amount of nesting, the amount of indents potential in your code, making it easier to read and easier to debug.

18:23  
Now you do have to be careful when you're using continue with a while loop.

18:28  
And that's because if we have told it to continue, it will just go back to the loop condition at the top of the while loop.

18:35  
And if we haven't incremented the state, like for example, if we're counting, as you saw previously, we're counting with I and that's after I continue, well, I is never going to get any bigger.

18:46  
So we'll just go around in an infinite loop.

18:48  
So you've got to be careful when you're using continue in while, whereas in for it's harmless in the sense of the top of the for loop can takes care of getting the next thing.

18:59  
So you're going to go back to that.

19:01  
So it's harmless, right?

19:02  
So I'm going to show you an example of using continue.

19:06  
So let's go back here.

19:08  
We'll go back to our code.

19:10  
We'll get rid of this.

19:12  
So here is a for loop again.

19:15  
And in this case, again, we have a list with three values, and we're going to go looping over these values.

19:23  
Now, what would typically happen is we've got to some particular state, right?

19:28  
So somebody has given us some object and we're looping over those things in the list, and we've reached this state.

19:34  
And so we're going to do something about it.

19:36  
So I've just put here as a comment some other logic, meaning something else is probably going to be here.

19:41  
And then once you've done the something else, you're going to hop back to the top of the list.

19:47  
So let's have a go at this one, see exactly what happens.

19:50  
So we're going to run in debug and like this item file, get rid of the gobbledygook.

20:05  
So we're here, you can see that we've got a list.

20:08  
It's got three values in it.

20:10  
So we're going to hop forward, right?

20:13  
So the number at the moment contains 2.

20:16  
So we're not that's not going to be true.

20:18  
So we skip over that.

20:19  
We print out the number.

20:21  
Now this time you can see the number is going to be 4.

20:24  
Look, it's four in the variable when it finally shows me go on 4.

20:31  
So we're now going to go inside there and the continue implies that we don't reach line 12.

20:38  
So we're going to go straight back to line 8.

20:41  
So Hop and there we are with line 8.

20:44  
So this can be used.

20:46  
Imagine at line 12, instead of just a print statement, you had a lot of other things you were going to do and you reached the state that was interesting to you at line 9.

20:55  
That means you're not going to process 12 and beyond.

20:58  
So effectively, that's how you save time.

21:02  
And there we go.

21:04  
So I'm now going to show you one with a while.

21:12  
So in this case, we have a while statement and we're going to debug.

21:16  
Sorry, that's our other programme.

21:19  
This one is what we want right now.

21:37  
What I'm doing here is I've done something which I should not do.

21:41  
And the purpose of this is to show you why it's dangerous to use continue potentially with while.

21:48  
So what happens here is I is equal to 0.

21:52  
So we go into this loop as before, we print it out.

21:54  
Great.

21:55  
Yes.

21:56  
It's not equal to 2:00.

21:57  
That's fine.

21:58  
So we're going to skip over.

21:59  
We're going to make I-1 bigger.

22:01  
We're back here.

22:02  
So now.

22:03  
Right.

22:04  
What's in the value of I?

22:05  
OK, it's one.

22:06  
Let's go one more.

22:08  
And now what's in the value of I?

22:11  
This time I contains 2.

22:13  
Now what's going to happen?

22:16  
Oh dear.

22:16  
Any any bright sparks?

22:20  
Everybody's needing an extra coffee, so what's going to happen is we reach I = 2.

22:27  
Continue.

22:28  
Look, we're missing outline 12.

22:31  
What's the value of I?

22:34  
Yeah, it's 2.

22:35  
Interesting.

22:37  
What's the value of I?

22:39  
It's still 2.

22:41  
So we've created an infinite loop by not putting the I + 1 here.

22:47  
So normally you'd want the condition that's going to update the variable you're checking to be before the continue.

22:55  
So you might have to duplicate it.

22:56  
You might have to I + = 1 here and then some stuff and I + = 1.

23:02  
Otherwise you make an infinite loop and we can run this and it will be an infinite loop.

23:06  
But I'm just going to stop it for now.

23:08  
If you have an infinite loop in Python, you can just press control C and stop the programme.

23:15  
OK, You might find that in the lab.

23:17  
Control C, stop the programme.

23:19  
Great.

23:20  
So let's go back to where we were.

23:23  
So does anybody have any questions or comments about continue and loops before we go on to the next bit?

23:31  
No.

23:31  
OK, let's go on to the next bit.

23:37  
So that's continue.

23:38  
The other thing we can do is break.

23:40  
Now break doesn't mean that we're going to go back to the top of the loop.

23:45  
It basically just is get out of here right now.

23:47  
We're going to break out of the loop.

23:49  
So we've got a series of instructions.

23:51  
We've been going through them.

23:52  
We don't know which loop we're at counter we're at.

23:55  
Potentially something bad has happened, like potentially somebody says stop now and then you can exit.

24:02  
Now you can use this effectively with a for loop.

24:05  
Imagine you are going through the fixed number of items.

24:10  
So the for loop itself doesn't have a concept of the user's told me stop.

24:14  
It's just going to go through the items.

24:16  
And then you could check what the user said.

24:19  
And if the user said stop, you could break out of the for loop.

24:23  
So the idea here is we avoid running the loop when it's not needed.

24:26  
We're just going to stop it.

24:28  
We can use this with while true.

24:31  
So while true on its own means that it will continue to run forever.

24:37  
But we might not want to make the decision to continue the loop at the top.

24:41  
So when we enter the loop we don't know what the state is.

24:44  
So we just say while true and then inside the loop somewhere something happens and then we say break and then get out.

24:53  
So this construct of using while true and break can allow you to create a, a do while loop essentially, which is something you see in another language that you may or may not be familiar with anyway.

25:09  
So that is break.

25:10  
So we can have a look at break.

25:11  
I've got a couple of examples of break here.

25:15  
So in this case, we've got a full loop again with our list of numbers.

25:19  
And yes, we break out here when number is greater than 4.

25:25  
So we'll just do it quickly, run debug.

25:29  
Yeah, thank you very much.

25:32  
And it will come down to the line, come down to line 8 once it gets there.

25:40  
Great.

25:40  
So what can you see?

25:41  
You can see it's got the number two.

25:44  
We're going to run over again and it has now it's got the number 4.

25:50  
We're going to run over again.

25:51  
Now it has the number 5.

25:54  
And so you can see, yes, 5 is greater than 4.

25:57  
So we break out.

25:58  
Now what looks what happens, we go from line 9 to line 11.

26:03  
So we never hit line 10.

26:04  
So break just means exit the loop.

26:07  
Right now breakout break exists in all sorts of programming languages.

26:11  
So does continue.

26:12  
It's not just a concept of Python.

26:14  
You'll find it in CC Plus plus Java, many other languages.

26:20  
OK, so let's stop the programme.

26:24  
Then we've got another for loop.

26:27  
So this one's a bit more complicated where we're actually doing some maths and then we're going to break out from one particular loop and not the other.

26:38  
So here you've got a loop within a loop.

26:41  
Now you can have as many loops within loops as within loops as loops as you wish.

26:46  
You might want to ask yourself, is this understandable to a human?

26:51  
And so sometimes you want to put some of the insides in a function so that you can see which each part's doing and test it.

26:58  
In this case, I've just got one loop within the loop.

27:00  
So let's see what happens.

27:02  
Let's go to here.

27:06  
So you can have a loop which is a while loop inside a for loop inside another while loop, and so on and so forth.

27:11  
Rather like you can have a dict inside a list inside a list inside a dict and so on and so forth, right?

27:18  
So what's going to happen is we're here and what can you see?

27:22  
Well, at the moment we've got the number four in I now I've introduced something here very quickly and I've not really told you what it does.

27:31  
So this word range, what it does is it creates essentially something like a list of numbers, and it starts in this case, from the number four and stops at the number six.

27:43  
So this range 4/6 is going to give us 2 numbers.

27:46  
When we ask for numbers, the number four and five, it never gets to number 66 is the stop condition.

27:52  
So what you've got here is 4 and yeah, OK, we go into this inside loop, we do whatever you need to do in there, and then we come back here and now you can see the number is 5.

28:07  
So we're going to hop forward, Yep, 5 and watch what happens.

28:11  
We break and we don't enter the 2nd loop.

28:15  
So a break will break you out of the loop that you're in if we put the break inside.

28:20  
So if we put it in line 10 and we had some of the condition there, it would break us out of the inner loop, but not out of the outer loop.

28:27  
So it only breaks you out of one level of the loop.

28:31  
OK, so let's have a look.

28:36  
Anybody got any questions about that?

28:40  
Yes, by the way, continue after else, else continue break.

28:52  
Yes, a break is only used inside a loop.

28:55  
So the if else is gonna take the decision straight away.

28:59  
So it's just if it's this, do that.

29:02  
If it's not, do this.

29:04  
You can think of if else as being a bit like in a post office.

29:07  
So you got a letter.

29:08  
You've given the letter, all right.

29:10  
Is it addressed to this person?

29:11  
Great.

29:12  
Chuck it in that box.

29:13  
So the if else just takes the decision, where's this thing going to go?

29:18  
It's there is no loop there.

29:21  
The loop is created by the the for or the while.

29:25  
So you're just saying for until some condition or while until, Well, until it's not true anymore.

29:33  
Yeah.

29:33  
So if else you don't need a break.

29:36  
We're using a break here basically inside the if, because we will want to say if this is true, break out.

29:44  
But the thing that's making it go round and round is this 4 at the top.

29:48  
All right.

29:50  
Yeah.

29:51  
Anybody else?

29:54  
All right.

29:55  
Good son.

29:56  
Just going to go through some other for loop examples.

30:00  
So we've seen this one already.

30:01  
Very common.

30:02  
We've got a list of something and we can iterate over it using a for loop.

30:07  
In other languages this would be called a for each loop.

30:11  
In this case.

30:15  
Here we're using range.

30:16  
Now, as I said before, what Range does is it creates something that behaves a bit like a list that you can get values from.

30:24  
It has up to three input variables that you can put into it.

30:28  
So this first one implies start from the number one.

30:33  
In this case I could put 10 or whatever, start from the number.

30:37  
The second one is the end condition.

30:40  
So in this case, I'm going to stop when I get to six.

30:43  
And the third one is the step size.

30:46  
So instead of counting up in one, which is the default, I could count up in two or three or whatever I want.

30:54  
Range won't allow you to use floating point, so you can't add +5, you know, .5.

31:00  
Rather, you're going to have to use something else to do that, which I'm not covering in this module, and I'm not expecting you to know.

31:06  
You can read about it if you like yourself here.

31:10  
We've got another kind of loop where we've combined the list idea with the range idea.

31:17  
Now, if you're just going to loop over a list, you aren't going to do this, but you might want to know the position in the list, which is what I've done here.

31:26  
So here I'm printing the value of the that's in the list, and I'm printing the position in the list.

31:34  
So what I've done is I've said, well, how long is the list Len numbers?

31:39  
That's going to go into N, So N will be equal to three.

31:42  
And then because I've got range N, I've I've omitted the other variables.

31:47  
If you've only got 1 variable with range, it's the end condition.

31:50  
So what this implies is range is going to give us the number 01 and two, which are the positions in the list.

31:59  
And then we're going to use that inside printing out here.

32:03  
So we can run this just to make the point of it run and debug.

32:10  
I need a break point.

32:11  
Hopefully I'm not too slow and it will hopefully reach the point.

32:17  
Line 9.

32:22  
Great, get rid of the gobbledygook.

32:30  
There we are.

32:30  
We're at line 9.

32:31  
You can see that the numbers contains a list.

32:34  
It's got 2, four and five.

32:37  
The length of this is obviously 3, length three.

32:40  
Now what's going to happen is as we go around here, the value is coming from range.

32:46  
So the first value it's given us is 0 and then the second value it's going to give us is one, and then it's going to give us 2.

32:57  
And then it stops because there's nothing left for range to give and we are finished.

33:04  
OK, so that's all I've got really in terms of the loops.

33:09  
And then I was going to go on to looking at coding style.

33:13  
Does anybody have any questions or comments before we go on to coding style?

33:19  
Yes, No, it doesn't at the moment.

33:24  
So if you want something like switch, you're going to have to use if else, if, sorry, if LL, if L, if L, if else.

33:33  
Yeah.

33:34  
So his questions about switch statements.

33:36  
So in other languages such as C, there is a switch statement.

33:39  
A switch statement is simply where you're asking what is something.

33:45  
And then you go down to the appropriate option in the list of these switch statements.

33:50  
So you could say A equals some character, could be QA or whatever, and then it goes down to the case and then executes that line.

33:59  
Python doesn't have that at the moment.

34:01  
I say at the moment because it's still a developing language.

34:03  
Somebody might put it in at some point.

34:06  
Anybody else?

34:10  
OK, fine, good.

34:13  
So I'll minimise that.

34:16  
So coding standards.

34:19  
Now compilers will run all sorts of things.

34:23  
They will run code that looks nice, code that looks bad.

34:27  
In fact, there is a competition to produce the most obfuscated code.

34:32  
Obfuscated meaning horrible, hard to read.

34:34  
And so you can create AC programme.

34:37  
That's probably the best language to do it in, or maybe C Or even if you're potentially like this sort of masochistic programming, maybe you want to use white space.

34:47  
Anyway, White space is a programming language.

34:49  
You can look it up.

34:51  
So you could generate these programmes, which are stupidly hard to understand.

34:55  
And I remember some colleagues of mine who loved writing in C++ in such a manner that they looked really clever.

35:01  
IAKA, nobody else can understand my code but me, and I'm the cool guy.

35:06  
Now if you do that in industry, you'll lose your job because what will happen is you built some code, they use it, and then they bring in another developer who's got no idea how to use your code.

35:18  
And that is bad because then the company loses money.

35:22  
So as a developer, you have to think about your other developer friends, meaning that you need to make your code easier to read.

35:31  
And one of the ways of making it easier to read is that you follow certain coding standards.

35:36  
Now the coding standards let's another developer know your intention without you having to spell it out for them.

35:43  
So we're going to follow some of the coding standards, but generally speaking, most of these coding standards are around can I structure my code in a standardised manner such that somebody else can read it?

35:55  
Now there are standards in Python, there are standards which I'm going to talk about Pepe.

36:00  
There are standards in all sorts of other languages like CC Plus.

36:03  
Plus, within a project within a company, you fix yourself normally on one standard and you follow it.

36:09  
In our module, we're going to follow the PEP 8 standard.

36:13  
You can read about it here.

36:15  
So I'm now going to show you some code.

36:18  
So let's have a look at the PEP 8 code, hopefully.

36:22  
Yeah.

36:22  
So here we go, style.

36:25  
So here are some standards of PEP 8 in terms of commenting.

36:30  
What we should have at the top of a Python file is a comment to say what is the intention of this file, like what is inside it?

36:38  
And for a documentation string in Python, we need to use three double quotes and then whatever we want to say and another 3 double quotes in some some write up somewhere.

36:51  
Sometimes people use three single quotes.

36:54  
It's it.

36:55  
Yeah, OK, it does actually function as a comment, but it isn't the standard.

36:59  
The standard is 3 double quotes.

37:00  
So please just use 3 double quotes and then you're texting here.

37:04  
The idea of a comment, as I've mentioned two weeks ago now, is that you're trying to say what the intention is of the code.

37:12  
So if you've got a long module, you might want to say that the module comprises some functions for accessing tables or something like that.

37:20  
So a short meaningful comment.

37:24  
Now, variables in Python should be in snake case.

37:28  
You think, oh great, we've got Python.

37:30  
It's a snake.

37:31  
So of course you're going to have snake case.

37:34  
Snake case implies that everything is in lower case.

37:37  
And then you separate the words with an\_If you think about a snake Slytherin on its belly, everything is low to the ground, and so you have this\_That's where you get the idea of snake case from.

37:49  
OK, so in this case I've got a total cost and I don't want to write it as one word because that would be a little bit harder for somebody to read.

37:59  
Total cost is that one word?

38:00  
Is that 2?

38:00  
I don't know.

38:01  
So I put total\_cost.

38:03  
I don't have AB as a variable.

38:08  
What is ABI?

38:08  
Have no idea.

38:10  
So you want to make the variables meaningful and you also want to have the\_to separate words so that somebody else can read it.

38:18  
Pascal case, which is what this is, is used in all programming languages I can think of, including Python for class names.

38:28  
So don't ever use Pascal case for bog standard variables.

38:32  
Here I've done something that you should not do at line 12 because another developer will come along and say, Oh yeah, we've got total cost.

38:40  
It's in Pascal case.

38:41  
Must be a class.

38:42  
I'll try and use it.

38:43  
It's not a class because I've just been bad and made a variable called total cost.

38:51  
In other languages such as Java, it's normal that you use camel case where you have a variable.

38:59  
So in this case I've got total cost.

39:02  
The C is capitalised to indicate there are actually 2 words stuck together.

39:07  
We don't use camel case in Python, we use snake case.

39:11  
OK, so if you are writing Java later on in the semester, please use camel case for variables, whereas in Python please use snake case.

39:20  
It's just one of those things.

39:21  
C programmers, yeah, they tend to write in snake case as well.

39:25  
Often C++ programmers maybe use camel case once they start creating objects.

39:31  
C# programmers, they use camel case.

39:34  
You get the idea.

39:35  
There's a standard or a way of writing in a particular language, and then people just agree on that and use it.

39:43  
The other thing that's in the PEP 8 standard is putting spaces in.

39:46  
The idea here is to try to make it a bit more readable.

39:50  
So they require a space after a comma.

39:53  
So you can see here we've got a space after the comma.

39:57  
The other thing they require is this space before and after an operator.

40:01  
Now this equals is actually an operator.

40:03  
It's an assignment operator.

40:05  
So we're putting something into something, we're assigning it.

40:08  
So the PEP 8 standard says we must have one space in front, sorry, after and before the operator.

40:15  
The only exception to that is where we're assigning values to to function variables.

40:21  
You you'll see that later on probably, but other than that, Please remember the spaces.

40:26  
The idea here is readability.

40:28  
The last thing that you'll see with PEP 8 standard is to have a line that's not too long.

40:34  
In fact, I think it's 78 characters.

40:36  
You can check it later if I'm right or wrong.

40:38  
78 characters is not very long.

40:41  
Why do you want to do this?

40:42  
Again, it's because of accessibility.

40:45  
Occasionally you might have somebody whose eyes aren't brilliant, so they have to have the screen zoomed in.

40:52  
And if your line is maybe 200 characters long, they can't see the end of that line.

40:59  
And so they're trying to understand what happens at this line in the programme.

41:02  
They can't reach the end of the line.

41:04  
You may be one of these people with extremely good eyesight.

41:07  
You know, you set your resolution, your monitor to very, very small and then somebody normal next to you has no idea what the end of the line means.

41:15  
So it's a kind thing to make it more readable.

41:20  
If we continue our line on the next line, and if you look at some of my examples, you'll see how I've done this, you can actually split the list.

41:29  
So here we could have continue the list like that on the next line.

41:33  
If our list is very long, there are various strategies to try to reduce the line length, but the general idea is we want to have lines that aren't stupidly long, making it hard to read.

41:45  
OK, so that is all I was going to discuss.

41:51  
Let's see, how much time have we got left?

41:55  
Will it go away?

41:57  
Great.

41:58  
So I've got time for a couple of quick questions.

42:01  
I can write a bit of code.

42:03  
Anybody got any ideas, questions, thoughts, ruminations?

42:12  
Yeah, 1 then one here.

42:14  
Yes, yourself.

42:14  
First the brake stuff.

42:20  
Yeah, if you've got lots and lots of conditions, yeah, you, you might want to create a variable which is going to contain the state if something's gone wrong, and then evaluate that at some point and say, ah, something's gone wrong, let's get out.

42:49  
So you can have a variable which is a boolean.

42:53  
So I've, I was doing this recently where we're reading a data file and it, it, we're looping through that and we're checking lots of different things.

43:03  
And then we're saying at the beginning of the loop, we've got a variable called success maybe.

43:08  
And we set it to true.

43:09  
And then inside the loop we'd say, all right, has this happened?

43:13  
Oh dear, it has.

43:13  
Let's set success to false and then we could try something else.

43:20  
Success becomes false at some point during the loop.

43:22  
And then we break out the This might be useful if you want to say, check all the possible errors and then exit the loop.

43:31  
So you're still deciding something bad has happened and then exit without having loads of break points.

43:38  
Sorry, loads of break statements.

43:40  
It's it's really up to you and it depends on what you want to do in terms of the programme.

43:47  
You could have several breaks.

43:49  
I I think often if your loop becomes very long and you, you, it's becoming more and more complicated, you tend to want to put bits of it into functions to say, all right, this is doing something.

44:02  
So I'm going to put in a function and then maybe the function itself will return a state to say something's gone wrong and then I might break out.

44:10  
So it, it in terms of code cleanliness, you try to factorise things so that you don't end up with too big a block anywhere.

44:19  
So yeah, you could use this Boolean variable idea, or you could break things out into functions.

44:24  
Is that all right?

44:25  
Yeah, yourself, yeah, yeah.

44:35  
See, you've got little exercises you can.

44:39  
So there are a few things you can do in terms of practise on the My Place page.

44:45  
Let's go back here.

44:47  
My Place page.

44:49  
We do have some references here.

44:52  
So the Eric Matthews Dish and textbook has got lots of stuff in there we can try out.

44:58  
You've got my code as well.

45:00  
When I'm learning a new language, I just tend to set myself little exercises.

45:04  
So at the top of this myplace page, you have these examples which I've given you.

45:09  
So you've got both the documentation and the code.

45:12  
So all the code works.

45:14  
So you could download it and just say, I wonder if I could build this.

45:17  
And then off you go.

45:19  
If you want to, you can look in.

45:21  
Beyond this, you can look in the textbooks down here.

45:26  
Where has it gone?

45:28  
Find the right place in a minute.

45:31  
A reading list.

45:31  
Yeah.

45:32  
So there is another textbook in here which has got some problems in it, which a bit more technical to try out if you like.

45:39  
And then finally one of the other where are we going down here?

45:45  
One of the other things I've given you the reading list is I've often given you these W3 schools and tutorials point things.

45:52  
So in there there's an example of how to do something rather like my examples, and then it says try, try and do something else.

45:59  
So you want to be confident with the language.

46:02  
So if you build up from the very basics, put some stuff together and see if you can build a more complicated programme and and have a look at these other things.

46:11  
If you have an idea how do I do this?

46:12  
You can ask me in the lab or you know, if something comes back to me.

46:16  
All right, anybody else?

46:19  
Yes, Yourself, yes.

46:29  
Well, you say how you can continue, you can use instead of using like FLM.

46:36  
Yes, it can.

46:37  
Good question.

46:38  
So I tend in the actual assessment, I tend to be relatively generous.

46:42  
So if it looks reasonable, I might put in the comment, well, you could have done this with a continue, but I'd still give you the mark.

46:49  
So it tends to be a little bit generous because it's an exam condition in the sense of you got, you know, you've got a fixed time, you're in the lab, you access all the stuff that's on the My Place page, it's all on the computer as well.

47:02  
So yeah, I tend to be generous with the assessment.

47:05  
But when you're coding, always think to yourself, is this the best way of doing it?

47:11  
Now, you can't be perfect all the time.

47:13  
And sometimes you just need to finish the programme, right?

47:15  
So if that's the way you can think of doing it, just put it like that and then maybe you come back and look at it later and go, well, I can make that probably better.

47:23  
Yeah.

47:25  
So in the short answer, probably unlikely to use marks.

47:28  
You might just get a comment saying you could have used this.

47:31  
If it looks like you've got lots and lots of levels of nesting, I might come back to you and say, oh, that's not very nice.

47:37  
You know, what have you done that for?

47:39  
Most of the problems are relatively simple, though, so you probably won't end up with a complicated programme.

47:45  
Anybody else?

47:46  
Yes, that is open book.

47:52  
Yes, you've got access to Myplace.

47:55  
Yes.

47:56  
Is that just gonna be the material from from collectors and labs and such that we've given that access?

48:01  
Good question.

48:02  
So in terms of the sorry, in terms of the exam, you can access a limited amount of content, right.

48:10  
So you've got everything on the Myplace page will be accessible.

48:13  
The Myplace page will be accessible.

48:15  
It'll all be downloaded onto the computer as well.

48:17  
So if you just want to like pick up a Python file, edit it, it'll actually be on the computer when you start.

48:23  
You can get at all these reference pages like W3 schools, tutorial point.

48:27  
They're all will be accessible as well.

48:30  
You can't use Stack Overflow and you can't use generative AI.

48:36  
Basically because I I'm trying to see if you can build a programme yourself.

48:41  
You can bring in your notes so you can bring your textbook in.

48:46  
You can through my place, put some files into your personal area and download them in the exam as well.

48:52  
That's fine, but the reason for this sort of controlled environment is because some of these problems are relatively easy.

48:59  
I'm setting them to see if you can think through the problem.

49:02  
I could, if I was really just sort of like, I don't really care, find a very good generative AI, which probably gave me enough marks to pass the exam, which is why it's no longer an assignment.

49:14  
All right, generative AI is a good thing.

49:18  
We should know how to use it.

49:19  
But the point of doing it this way is we need to know how to programme in order to use generative AI because otherwise it would just generate code which might work today, but it's got horrible bugging.

49:29  
And if we don't have the ability to critique what it's using, then we're in trouble.

49:36  
Anybody else?

49:40  
OK.

49:41  
All right, I'm gonna stop there.

49:42  
If anybody's shy, I'm gonna hang around for the next 5 minutes or so.