**w10-01-WorkedExamples-2024**

0:01  
OK, so today we're going to look at some worked examples.

0:04  
So I'm not adding any more in the way of functionality or ideas.

0:09  
We're just using those ideas to build more complicated programmes.

0:13  
So this is essentially reworking things we've already looked at.

0:18  
Next week is somewhat similar.

0:19  
We're going to have another look at similar ideas.

0:24  
Now what you've noticed is because we had the lab last week as the class test this week, we're now a week out of phase with our labs.

0:33  
So in the lab this week we'll be looking at unit tests and modules.

0:37  
And so in the following week's lab, we'll be looking at the the example exam paper and having a go at that and seeing how we do.

0:46  
OK, so today I'm going to go ahead and look at some worked examples.

0:51  
So let's see what we've got here.

0:55  
So I'm going to talk through them, and if things are not clear, I'm going to pause and ask the questions.

1:01  
Please just ask me.

1:04  
Now we've got 3 examples, and these are building up in terms of the complexity.

1:09  
The last one is the most complicated thing and includes a few other ideas.

1:14  
So let's start with the first one, which is relatively easy.

1:18  
So in this example, we have a class called Network Device.

1:22  
Here it is.

1:23  
It has a constructor, and we're going to pass in three variables here, 3 values or parameters, and they're all text strings.

1:31  
So you've got a Mac address, an IP address, and an OS where OS is short for operating systems such as Linux or Mac.

1:40  
You can see that we pass in these variables.

1:42  
There are three of them, and they're assigned to the data members just like that.

1:48  
Now we also have one of these repr functions.

1:51  
And as I've said before, the normal convention for these functions is that they return a text string that when you run it as Python, it will create another object.

2:01  
And we can test that by using eval as we've discussed before.

2:06  
So here we have the.

2:07  
That's the name of the class.

2:09  
And then you can see that the two names of the, well, three names of the parameters and the three corresponding values are given here.

2:17  
Now the values are in quotes because they need to be in quotes so that when that string is run as Python, they are strings.

2:27  
If I remove those quotes, what will happen when you evaluate it is Python will say that it doesn't know what that variable is, basically because it's treating the string as a variable, which it isn't.

2:38  
So if you don't want to use back slash double quote you can use a single a quote that will work fine.

2:45  
In all my examples I'm sticking to double quote just to avoid confusion.

2:51  
OK now I've got 2 little functions here and often when you are thinking about how you deal with classes you want a function to transfer data into the class and another one out.

3:03  
Now my 2 two functions here, 1's called from Jason and the other one's called to Jason.

3:09  
This is building on an idea which you've previously seen where we were reading and writing to Jason format.

3:16  
Now here rather than a file, the idea is that we're going to pass in a text string which contains the Jason.

3:23  
And then rather than using load, which is the function that loads from a file, here I've used loads.

3:30  
So loads you can give a text string and then it will load up the Jason and it appears as a Python entity.

3:42  
So in this case it could be a list or a dictionary or something else.

3:46  
Now to be a bit more careful than I was previously, I'll actually put in here a type error exception.

3:53  
So in case there's something wrong with the input so that loads fails, I'm now catching that exception.

4:02  
And for this function, as is often the case, you think OK, something's gone wrong.

4:06  
This function is going to return false if something has gone wrong.

4:10  
That means it can't read the data from the input.

4:14  
Now I could load Jason from a text string and the outside data structure could be a list.

4:22  
Now in my case, I don't want it to be a list.

4:25  
I want to check is it a dictionary before I start using it as a dictionary.

4:29  
So here what I've done is I've said if it isn't a dictionary then again I want to return false so I'm catching the problem.

4:38  
So when you're programming, you often need to think, oh, they might do this.

4:43  
How will I stop that from causing the programme to crash?

4:47  
And then finally, I use the dictionary.

4:50  
Now again, I'm very paranoid here.

4:52  
So I'm taking what's in the dictionary and I'm actually casting it to a text string because that dictionary could contain an integer.

5:02  
So I don't want my data type to become a different type depending on the input data.

5:06  
So I've decided to cast it to a text string.

5:08  
Now anybody who's thinking will think right dictionary, if he uses a key on the dictionary, that key might not exist in the dictionary.

5:18  
And you're right, it might not.

5:20  
So again, I've caught the key error here.

5:22  
So if any one of these 3 keys don't exist in the dictionary, it will end up here in this key error exception.

5:30  
And then I'm just saying what the exception is.

5:33  
And again, I am returning false.

5:38  
That is, something went wrong in my function.

5:40  
So I've caught all of the obvious errors.

5:44  
Now if there's another error that's not obvious, I'm thinking, well, the programme can crash because something very bad has happened because I've dealt with all of the ones I can think of.

5:56  
And then we've got yeah, accept value error.

5:58  
So this appears again in terms of I can't cast it to a text string.

6:05  
You could appear down here, it's highly unlikely, but I've just left it in in case.

6:10  
OK, so that's our from Jason, and now we've got a little function which is to Jason.

6:15  
This one's a bit easier because I've already got the values in memory, so there's less that can go wrong.

6:22  
I'm just going to write them out as Jason.

6:24  
And I'm using dumps rather than dump.

6:27  
So dumps is the opposite from the loads in that it dumps the Jason into a text string.

6:36  
So here I have inputted a dictionary.

6:39  
You can see there's my dictionary.

6:41  
It's got 3 keys with their three corresponding values which are the data member values.

6:47  
I'm then dumping this into a text string and I'm returning it.

6:50  
And so that's my little class.

6:51  
It's essentially a data class with a few pieces of data in it and it can be converted to or from Jason.

6:59  
That's it.

7:01  
Now, if somebody's written a little class like this, you then probably want to test it.

7:06  
And so often we generate a bunch of unit tests to match what we wanted.

7:11  
So in this case, you can see my unit test file is called Test\_Network.

7:16  
My original module was called Network.

7:18  
So it seems reasonable to have test network because then another developer realises that, yeah, it's associated with the other module.

7:26  
OK, the recipe for our unit tests, we've discussed before we import unit test because we're going to inherit from test case, which is in unit test.

7:37  
And then you can see what I've done here.

7:39  
I've imported the class network device, which is in the module network.

7:45  
So that network dot PY contains this class network device.

7:49  
Now, once I've imported it, I can then just go ahead and use it in this file as if I declared it in this file.

7:55  
All right, So once I've done that, I can then just go ahead and use it as you see here at line 7.

8:00  
So my two tests are one, to check the repr function works correctly, and two is to check that I can read and write Jason.

8:10  
That's really what I want to do.

8:12  
Now I would perhaps want to read and write bad Jason as well, just to check that the exceptions are thrown.

8:20  
So in this first Test here, we've created an object you can see here Network device.

8:25  
That's our little object and then what have we done?

8:28  
We've cast it to a text string and then we've given it to eval and hey presto, we have then put that new object into new device.

8:37  
Now if our repr function was wrong, as in you couldn't do that, what would happen is that this line here at line 12 would fail, it would crash and the test would fail.

8:50  
So it would tell us test failed and which line.

8:53  
So if it has then passed, what I can do is I can just check that the data members of one object are the same as the data members in the other object.

9:03  
Now I could, if I wanted to, put an EQ function in the class definition and then I could just compare here the two objects.

9:11  
But because I don't have that EQ function, I've explicitly used each of the data members to check they're all the same.

9:19  
It's a good idea to check everything because occasionally if you don't check one thing, that's the one thing that isn't working properly.

9:27  
OK, so then our test, Jason passing here.

9:30  
What are we doing again?

9:31  
We've created a little object.

9:34  
So I've just put in some values which seem reasonable.

9:37  
And then what have I done?

9:39  
I have converted that object to Jason.

9:42  
It's actually a Jason text string here by calling our member function.

9:46  
So I've just created the object, called the member function.

9:49  
Great, got the member function.

9:51  
Now what am I going to do?

9:52  
I'm going to go from that Jason text string back to the object to check if that works.

9:58  
So I create here a new object.

10:00  
You notice I haven't given it any inputs.

10:03  
That means it's going to use the default, which if you remember from the constructor is 3 text strings that are empty.

10:08  
So it's going to have three data members that are just empty text strings.

10:12  
We're then going to read from that Jason, which is going to update our data members.

10:18  
And then again, because I don't have an EQ function, I'm checking all the data members are the same.

10:25  
All right?

10:25  
So one object has got the same data as the other object.

10:29  
Now they're not in the same memory space, they're not the same object, but the data that's in both is the same.

10:36  
OK, now having thought about this, I'm thinking right I've got to try out some of the nasty exceptions that could be thrown.

10:44  
So the first one I am going to try is I'm going to pass in here just a list.

10:51  
Now I've already said a list shouldn't work because it's not the right type.

10:57  
So I'm going to see if that throws an exception.

10:59  
Then I'm going to pass a list inside a text string.

11:02  
Now a text string is the right type for this function to read, but look what's inside.

11:07  
It's a list.

11:08  
So as I said before, what I'm expecting inside this text string is a dictionary.

11:12  
So that one will be passed by the Jason input, but it should be caught by the check.

11:19  
Is it a dictionary?

11:21  
And then lastly I have inputted a dictionary inside a text string which again is going to go all the way in, but notice there are no keys.

11:29  
So here I'm just checking if that the key exception is caught correctly and if that returns false.

11:36  
So if any of these asserts fail, that test as a whole will fail and it'll tell you which line it fails on.

11:44  
And then at the end here, I've got a main, so this is the unit test main to run the tests.

11:50  
And if we run them, they all pass just because the code is currently correct.

11:56  
You can see here it's ran 2 tests.

11:58  
Now it says there are two tests because there are two functions inside this class that are named test\_something.

12:06  
So there are two functions, it says they both pass, and there you go.

12:11  
Does anybody have any comments or questions about this first example?

12:17  
No.

12:18  
All right, then let's go to something slightly more complicated.

12:25  
So this one we are dealing with is going to read and write some text data.

12:33  
So again, we're just playing with the same functionality, but the example itself is a little bit more complicated.

12:40  
So now I've got the idea that I have something in a shopping basket.

12:46  
You know, you've seen this kind of application several times before.

12:49  
You buy something online, you have something in the shopping basket.

12:53  
OK, so to describe what's in the shopping basket, I've created this class called basket Item.

12:58  
And the purpose of basket item is it's going to store a name, a quantity.

13:04  
And when that basket item was last modified, all right, so it's got a modified here is actually an input text string.

13:14  
You can see quantity is by default an integer of 0.

13:18  
So what's happening here is that again, when we call the constructor, we're passing in the name, the quantity.

13:25  
Now I've done a check here, which is that if somebody hasn't given me a modified text string, so that's that means somebody hasn't told me when it's being modified, I'm actually just going to use the current date time.

13:38  
This is quite helpful.

13:39  
Imagine you create an object.

13:41  
You don't necessarily want to be asking what the date time is in the code outside, you just create it.

13:46  
Then inside the object here we find out the current date time and we store that.

13:52  
So this self modified is actually a date time type.

13:57  
It's not a text string type.

13:59  
Now if somebody has given us the text string, we assume that the text string they've given us can be converted and it's an ISO format date time string.

14:09  
Now you could say, well will, if they don't do that, this might throw an exception, you know?

14:14  
Well, yeah, it might do, but maybe I'm OK with it being crashing at that point because they really shouldn't do that.

14:21  
I mean, you could try and catch this a bit more intelligently, but that's what the constructor is doing.

14:28  
OK, so the repr function is the same as before.

14:32  
Notice we have not quoted the quantity here because it is an integer.

14:37  
So that when we print that, we don't want quotes around it.

14:40  
We've quoted the modified and we've quoted here the name.

14:45  
Now we've quoted the modified because what's going to happen is we're going to cast the self modified to a text string, which actually turns what's the date time into a text string.

14:56  
And we want to pass the text string back in anyway.

14:58  
So that's why we've quoted those two, not this one.

15:02  
Now we go down to the list file.

15:04  
OK, So what are we doing here?

15:06  
So we've got again two functions.

15:08  
There's one list to file and you can guess what's going to happen.

15:11  
There's another one file to list.

15:14  
And the idea here is that we let's go down to the end and we'll we'll go back up.

15:19  
We've got a little programme to show how we meant to use this code.

15:23  
So you can see that we've protected the main function using this if name equals, equals main.

15:29  
Now I've done that because if I don't do this, if I just put main outside of this, if when I imported this file or this module in the test code, it would run the main programme.

15:40  
I don't want that to happen.

15:41  
So that's why I've got here if name equals equals main.

15:46  
Now if you've run the pro run this file like Python basket IO dot PY, then it will call here the main function.

15:54  
So this main function just shows us how to use the code so we can get the current date time.

16:00  
We can then turn the date time into a text string.

16:04  
So this is just an ISO format text string.

16:07  
It's got the, you know, the year, the month, the day, the time, the hour, minute, seconds.

16:13  
You can have microseconds as well in there.

16:16  
Now we've created a few basket items.

16:19  
So the idea is that you can have several basket items within some list.

16:24  
You can see there and then there are several groups of these basket items.

16:29  
So I've actually just created a list which inside has lists.

16:34  
Now when you're storing data in a file, you can think to yourself, I want to separate this record or this data block from a next one.

16:44  
Now depending on what you're dealing with, there might be a way of doing this.

16:48  
It could be in documentation.

16:50  
Here I've just made something up which mimics roughly what you might see in other places.

16:55  
So the idea is if I go to the top for a minute, we'll come back to where we were in the text file.

17:01  
We're going to print begin record when we're starting a group of Basque items, and we're going to print end record when we finish that group so that we can see when the data that are one block are in the file.

17:14  
We could have then many records in the file rather than just one.

17:18  
That sort of behaviour or approach is sometimes used with binary files, where in a binary file you you don't have text formatting, so you're not quite sure when the next block is.

17:31  
You might use a special character or something to do that.

17:35  
Right, let's go down to where the code was.

17:37  
So we create these Basque items and then what are we going to do here?

17:41  
We are just going to output the items as a text file and then we're going to repeat reading them back in and checking that this works.

17:53  
So I'm going to go and run the code and then we'll have a quick look at what it's doing.

17:58  
OK, it ran it printed some stuff out as well, but let's ignore that for now.

18:05  
You can see there there is the output.

18:07  
So what have we got?

18:08  
We've got begin record, and then inside here we have one list of Basque items, and then we've got begin record and we've got another list.

18:18  
Now for ease of use here, what I've done is I've just used the repr function to dump the data into this file.

18:26  
All right, now you might want to do in a different way.

18:29  
I just this way, just as an illustration.

18:32  
OK, let's go back to the code.

18:35  
So what's going on?

18:36  
List to file?

18:38  
Let's try that one first.

18:44  
So this one is the easier one of the two.

18:47  
What happens here is we are looping over the records where you can see each record is a list of Basque items.

18:58  
And then when we pick up the first one, we start by writing into this text file our start record text string.

19:06  
Now that that I defined at the top of the file, so I'm not redefining it, I could have defined it in this function.

19:11  
I put it outside the function because I'm going to need to use this text string inside the from function as well.

19:20  
So the file to list in this case, right?

19:24  
Then we write out the objects.

19:26  
So here I've just converted the object to a text string and then I've written the end record.

19:31  
I'm using here just a for loop because I've got a series of records in a list and I'm just taking each one in turn and writing them into the record.

19:38  
The file to list is a bit more complicated because things could go wrong whenever you are reading from a text file or some of their input.

19:49  
There's many ways a user could get in there and edit that file and do something bad, so that's why this function is more complicated.

19:57  
So what am I doing here?

19:58  
I'm starting off by clearing my records.

20:01  
I'm paranoid.

20:02  
I want the function to behave in the same way if I call it twice, so I'm clearing them to start with.

20:09  
And then I've got here while true because I don't know how much data I'm going to read.

20:14  
So I don't want a for loop.

20:16  
I'm just going to keep reading until I reach the end of the file.

20:19  
Hence while loop now I'm going to read a line from the text file.

20:25  
Now this read line will read one line from the file or if we've got to the end of the file, it will actually give us a line back, but that line will be empty.

20:36  
So this Len line is 0 is reached when we reach the end of the file and you can see when we've got to the end of file.

20:43  
I'm just going to break out the while loo, so no more data to be read end.

20:48  
The next thing that I might have a problem with is somebody might added a bit of extra white space at the end of the line, maybe a couple of spaces or tabs or something.

20:57  
So I'm using strip.

20:58  
What strip does is it removes the white space at the front and the back of the texturing.

21:03  
So now we've just got text without any spaces or tabs either side of it.

21:09  
So I've now got this text string.

21:11  
OK, so I want to know if I'm inside a record.

21:15  
So I'm asking is the text string equal to our record string?

21:19  
If it is, I'm saying right, I'm inside a text string now.

21:22  
And what I do once I'm inside a text string is I create a little list.

21:27  
I'm going to use this list to put objects in that I've read from the record, and I'm going to keep those until I finished the record because I could have several lines in the file where I'm going to find objects.

21:40  
And then having found the start of the record, I continue.

21:44  
So I go back around the loop.

21:46  
Now notice this is safe for a while in this case because I've left the read line up here, all right?

21:53  
So in some cases, if I had like I plus plus or something continue wouldn't be safe.

21:57  
But here it's OK.

22:00  
I've then got another check, which is if I've reached the end of the record, I'm going to append the objects to our list of records and I'm going to clear the object list again by creating a new object list.

22:17  
Now I've created new ones here because I don't want them to interfere if I didn't create a new one.

22:23  
If I just cleared it, I'll be clearing the object list, the first one, which wouldn't be what I want.

22:29  
And then lastly, I've got a cheque to say if I'm inside a record, I'm going to run that line as Python And create an object.

22:37  
Now this is not safe.

22:38  
Don't do this yourself.

22:39  
It's just done for this simple example to quickly bring the data into memory.

22:45  
The problem with a vowel like this is somebody could put something naughty in that text, which you then just run.

22:52  
So yeah, it's just to simplify the code.

22:55  
I've lift this here.

22:56  
Now you can see what will happen is if we are in the text file and we haven't reached the record yet, what this programme will do is it will ignore text.

23:06  
So you can put anything you like, you know, like I'm really fed up.

23:10  
I need a coffee or something at the top of the file.

23:12  
It will ignore it until it reaches the first record because it's requiring this in record, right?

23:19  
So you've got to think to yourself, what are we going to do about data that's inside or outside of record definition?

23:25  
All right, so there you go.

23:27  
Now if you want to test this again, you need to think about what am I testing.

23:32  
First of all, I'm going to go and test the repr function here.

23:36  
I've created an object, I've called it ice cream, just because it seems like a nice name.

23:41  
I'm going to have one of those.

23:42  
I'm going to use the ISO format time just now just check eval.

23:47  
Does eval work?

23:47  
Yes, hopefully it does and then check the data members that they are correct.

23:52  
Now the eagle eyed among you will notice I have forgotten to check a data member, right?

23:58  
I should check the data member and I could do that by just checking this time.

24:04  
Now, is the correct text string inside.

24:08  
Here we go, the text passing.

24:11  
In this case, what I've done is I've introduced an idea which I mentioned or hinted at last time, which is that you can actually use a text string as if it's a file.

24:20  
And this is very helpful when testing.

24:23  
So here, what are we doing?

24:26  
This one is read, write.

24:27  
So what we've done is we've created this thing IO, string IO.

24:32  
So string IO can be given a text string, right?

24:36  
So it's already given a value here, I'm just creating one that's empty.

24:40  
So this string IO has associated with a text value and we're reading and writing from that text value in in memory.

24:49  
Why is this good, good idea?

24:50  
Because often when we're running unit tests, we don't want to read and write files.

24:54  
It's going to slow down the unit testing.

24:56  
It'll just make a mess.

24:57  
We have all these files for all these different reasons.

25:00  
So let's just keep it in memory.

25:02  
OK, so I've then defined some records.

25:06  
You can see I've used my ice cream washing loved liquid and potato.

25:10  
I've then called our function here list to file.

25:15  
And then what's happening is that we are seeking 0.

25:20  
Now, if you remember our previous discussion about files, when you're dealing with a file, you have a cursor and as you read, the cursor just advances down the file to get to the end.

25:32  
And likewise, as you write it advances.

25:35  
So what am I doing here?

25:36  
I'm saying go back to the beginning of our file, which here is a text string.

25:42  
It's the string IO.

25:43  
So I am writing to the inside the text string of this string IO.

25:48  
I'm then going back to the beginning and then I'm reading back from it and I'm checking if my records are the same.

25:57  
So there's no file actually being written.

25:59  
It's just using this string IO, which is purely in memory, which means then we're not messing up the hard drive and it's slightly faster because it's just in memory.

26:09  
Does anybody have any comments or questions about this one?

26:19  
No, OK, should have maybe checked in and had an extra shot of espresso this morning or something.

26:27  
OK, so let's go and have a look at the next one.

26:33  
How you know, run, run these programmes and sort of see what's going on.

26:37  
I'm just playing with the same functionality as we did before.

26:40  
Now this last one is the most complicated and includes a useful idea when you're testing code, which is of having a mock version.

26:50  
So sometimes what happens when you're testing is that you want to just test one class or two classes or a couple of functions.

26:59  
You don't want to test all of your code.

27:02  
The problem with testing all of your code at once is that you're not sure where the problem is.

27:07  
So often what we want to do is we might want to separate the code from the database.

27:12  
So we forget about the database.

27:13  
We're just going to test the code in the middle.

27:15  
We might want to separate the code from the user interface.

27:18  
So we're going to send in some some values and check does it behave as we expect.

27:23  
So in this particular example, I've got the idea of a configuration service.

27:29  
And this little configuration service, it's going to load up the settings that we want to use, and then we're going to use it.

27:36  
But in the unit test, I don't want to deal with files rather like it didn't in the last example.

27:41  
So I'm going to create a mock configuration service, which I'm then going to put data in.

27:47  
And so then I can call the function, avoiding testing the whole application.

27:52  
I'm just testing part of it.

27:53  
All right, let's have a look at the code quickly.

27:58  
Great.

27:59  
So here's our dice configuration.

28:03  
So the dice config here has inside the dice that we want to roll.

28:08  
So you can see here we've got for each die that we're going to roll, we've got here a dictionary and inside there's the number of sides.

28:15  
So this one's got 6 sided die, another six sided die.

28:17  
So I could change, I could have like A10 sided and a six sided or 210 or whatever.

28:23  
I've also put here the number of roles I want the programme to run for.

28:27  
The programme is very simple.

28:29  
All it's going to do is calculate the average value you'll have with the configuration.

28:35  
So you could change this file, right?

28:37  
You can have six or seven die with different numbers of sides.

28:40  
Make the number here and rolls quite big and it'll tell you the most likely value to be rolled on that combination of dice.

28:48  
OK, let's have a look at the programme.

28:53  
So down here, like I say, this is the most complicated.

28:58  
What we've got here is a little test programme.

29:01  
We have created an object of this config service type, and we've given that constructor the file name, all right?

29:10  
And then once we have this config object, we're then passing it to this function average dice total, which is then going to give us the average value for rolling those dice.

29:22  
And we're then printing out the config as a text string.

29:26  
And we are printing out the average.

29:28  
All right, So we can run it and we'll see it works.

29:33  
Oh, yeah.

29:33  
OK, so here's our config.

29:35  
And it tells us that for those two, it's 7 pretty much, which is what you expect from maths.

29:42  
All right, so let's now dive inside a little bit further.

29:48  
So I'm going to go quickly because I want to point out the mocking as being kind of useful.

29:53  
Now, what I've done here is I've again been paranoid, all right, So what could be wrong in my input file?

30:03  
Let's go to the config.

30:06  
So the config we're passing in here, the file name, and I've made a private data member.

30:12  
I know I said normally don't use private data members, but there are some cases even in Python where they're a good idea.

30:19  
So in this case, I don't want somebody else fiddling around with the configuration data.

30:24  
So I'm going to keep that private because I want to be able to load them when they're needed.

30:28  
And then if somebody asks for them again, I'm just going to give them the stuff that's loaded.

30:33  
So if somebody calls the config function, what happens is if this private data member is None, it goes ahead and tries to load the data.

30:43  
And if that isn't successful, it sets the data member as None.

30:47  
But if it is successful, it just returns the data member.

30:50  
So this sort of functionality is what you have when you've got some kind of caching, you read the data in, you're going to offer the data back.

30:59  
If you, if you don't have it, you load it.

31:03  
So here's the load code.

31:05  
OK, what we're doing here, we're checking if the file exists.

31:09  
Somebody could perhaps give us a file name that isn't valid.

31:13  
So if that is the case, we tell them and we return false.

31:16  
Again, thinking about what the user could do, that's not going to work.

31:21  
Then we go ahead and try and open the file.

31:24  
We read it now what you see down here is I've actually I'm just reading it here, so there could be a read error perhaps.

31:32  
That's unlikely, so maybe I'll just let my programme crash if that happens.

31:36  
I'm checking the Jason, so I'm loading it with loads again.

31:40  
Remember, loads, loads from a text string rather than a file.

31:43  
And I'm catching the exceptions that could be thrown by loads and if they occur, I'm going to return false.

31:51  
So this is the same idea as we had in the first example when dealing with Jason.

31:55  
We're just catching the exceptions.

31:58  
Great.

31:58  
So that's config.

32:01  
Now we can, as before, we can test the config service so we can have a little unit test file just for the config service.

32:09  
And what I've done here, rather than use our string IO, I've decided, ah, I'm going to go back to files again.

32:14  
So I've got 3 files where I've come up with combinations that are not good.

32:20  
So the first one is just a text string, it's not Jason at all.

32:26  
The second one is, well, almost Jason, but this part is bad.

32:31  
There should be a key with a value.

32:34  
There's only a key.

32:35  
And then we've got our actual working piece.

32:38  
Now you can see that I haven't covered all the potential errors here.

32:42  
I've covered some of them, right?

32:45  
I then go ahead and call the constructor with these different files I load and the load has to operate as we expect, meaning if the file is bad, it returns false and if the file is good, it returns true.

32:58  
Now I could check beyond this, I could check, well, what are the values here?

33:02  
I've just made a simple unit test rather than checking all the values, probably a good idea to check the values are loaded as well.

33:10  
OK, go back to our little programme.

33:13  
So back down to the end.

33:16  
So in our programme this is the thing that uses everything else we can rated this config service and then we're going to call this average dicetotal where we're passing in the config.

33:30  
Now the average dicetotal haul it does here is it cheques to start with, is the config non or not?

33:36  
Now you remember if it fails to load, then that will be non.

33:41  
So if it is non, we want to exit and then what are we doing here?

33:47  
We're getting the dice config.

33:51  
Yeah, sorry, I'm skipping over check config.

33:53  
Check config is actually defined up here and it's another layer of paranoia.

33:58  
So the first layer is is it Jason?

34:01  
The second layer of paranoia is, are the inputs reasonable?

34:05  
So for example, somebody could give us the wrong data type.

34:11  
It needs to be a list of dice.

34:13  
They could give us something which isn't an integer.

34:17  
We need to have an integer for the number of sides, otherwise our code won't work.

34:20  
They could give us something which isn't an integer for N rolls and they could just not give us the keys that we want.

34:27  
So again, what I've done in this code is I've checked all of the data inside the Jason is OK, and if it's OK, I return true.

34:35  
Fine.

34:36  
So here then I'm getting the config and then getting the number of sides.

34:41  
And once I'm finished, I then have a for loop where I'm throwing the dice many times.

34:47  
You can see there we go.

34:49  
We're throwing the random numbers and we're forming the average.

34:53  
Great.

34:55  
Let's have a look at the unit test.

34:57  
I'm skipping over that quickly because I want to point out the mocking.

35:01  
Now there are mocking frameworks available to you.

35:04  
Some are more or less complicated to use.

35:06  
I'm not expecting you to learn them for this module.

35:08  
I'm showing you this, this particular example to demonstrate.

35:12  
Yes, you can create mocked classes yourself and in some pieces of code it's easier to do this than to use a mocking framework.

35:22  
So the the class that I'm mocking is called config service.

35:27  
So I've called my mock config service, mock config service, very obvious, everybody knows what it's doing.

35:34  
Now the the key thing with mocking is your class needs to have the same outside as the one you're pretending to be.

35:44  
All right, So you can see the constructor looks the similar, right?

35:48  
It's the same idea.

35:49  
And importantly, our config here does the same thing as the other one.

35:56  
Config here is the function that's being called by the piece of code which we want to test.

36:03  
Now why do this?

36:05  
OK, well why?

36:06  
Because I want to separate from the code that's reading the file and just test the bit that's going to run the simulation.

36:15  
So then having done this, I can create an object of mock config.

36:20  
I can set the values.

36:22  
Notice here I'm just setting them.

36:24  
I don't have a text file.

36:25  
I've set them in the mock version.

36:28  
And then I'm passing the mock version into our function check config.

36:33  
OK, can check the config from the mock version and I'm passing it also down here into our.

36:41  
Where has it gone down here into our average dice total and just running that?

36:47  
So I've separated 1 block of functionality from another using mocking.

36:54  
I advise you to have a bit of a look at the code yourself and run it.

36:58  
So we've got a few minutes.

36:59  
Does anybody want to ask many questions?

37:02  
Any comments?

37:14  
All right, I can hang around if somebody wants to ask me afterwards.

37:19  
Anything.

37:21  
Run the code yourself if you haven't done so already.

37:23  
Download it, run it, have a look at it.

37:26  
The important thing we're doing here is we're, we are creating classes, we're practising that skill, We are practising thinking about what could go wrong, especially with inputs.

37:37  
If you have any sort of input from a user, text file, typing on the keyboard, network input, whatever, there's probably going to be an error in there.

37:47  
If you're writing into it like you've already seen, you aren't going to have the same issue because you are writing the data you've defined into the output so that you have control over.

37:58  
And yeah, when you're testing, you need to think about the things that can go wrong in your code.

38:04  
You want to catch the exceptions that you expect, but if there's something that really isn't expected, don't catch it.

38:10  
So don't have Python code.

38:13  
I know you can do it, but don't have Python code that catches every exception, because you do actually want programmes to crash when something very odd happens if they silently continue.

38:25  
That's actually not a good idea.

38:28  
All right, So I think it's absolutely no comments or questions.

38:32  
I'm just going to stay here for anybody shy who wants to ask me anything.

38:37  
Otherwise, I look forward to seeing you in the lab.