**w08-01-InputOutput-2024**

0:04  
Anyway, we're taking a break from classes and we're looking at input and output operations.

0:10  
Now, you could put these inside functions or inside classes.

0:14  
It's totally up to you.

0:15  
So I'm not going to introduce all of the possible input and outputs.

0:20  
I'm only going to introduce some that are common.

0:23  
If you can think of a file type, maybe a Excel file for example, there is a Python library for it, so you can directly read Excel file content straight into Python without a problem.

0:35  
I'm not going to cover that in this module because I don't want to make it too complicated.

0:40  
OK, so the types of file that you tend to come up against, they are really into two groups, Text files.

0:49  
They're files that you can open with a text editor such as Notepad and look inside them and you can read what's in that file or a binary file where the computer has taken some of what was in the memory and written it straight to the file.

1:06  
Now those files, if you open them with a text editor, you'll see all sorts of weird symbols because it can't really show that to you using normal text characters.

1:18  
OK, so those are our two file types.

1:21  
Now, databases are there and available to us.

1:25  
They are servers.

1:26  
So normally I say normally they're servers.

1:29  
SQLite is not a server.

1:30  
It is a database however, and the way we interact with databases is also spread into two.

1:36  
So there are so-called relational databases where we talk to them using SQL.

1:42  
So this is standard query language and in this case our programme sends a SQL command to the database, the database replies with some data and we read it back into the programme.

1:54  
Now we will look at this next semester in 993 with a web service example, but I'm not going to cover it in this module because again I wish to limit the complexity.

2:07  
No SQL databases are so-called not only SQL.

2:11  
So for example Mongo DB is a no SQL document database.

2:15  
Again, I'm not going to cover this in this module or even in 993.

2:20  
If somebody's keen, I can give them a bit of example code.

2:24  
It's something that you could pick up for your project.

2:26  
If you want to store data in a no SQL database, you can and you can read those data back into Python.

2:33  
You can also send data over a network connection.

2:37  
So imagine my laptop here has got Wi-Fi.

2:41  
I could write a little Python programme which will send or receive data over that Internet connection and listen to traffic.

2:48  
So just in the same way we have a web server that listens to requests for web content or Http://requests, we can create that sort of thing in Python And it can listen and react to somebody's request.

3:04  
Now, when they request data, the data themselves can be sent in text or binary, it doesn't matter.

3:10  
You can do either.

3:12  
All right, so that's a general introduction to what you can do with input output.

3:16  
You can also have what I haven't put here, devices, so for example, a sensor that is measuring the temperature and the way that works is similar to a file or something else.

3:29  
You connect to it and you then take or you read values from it.

3:33  
Or if you have a device which is say a motor driver that's going to spin some kind of motor on a robot in the same way you open a connection to it and you write values to it and then it just behaves as you told it to.

3:46  
Maybe, you know, advances 10 metres or whatever you've done.

3:50  
So the outward connections, whether they be files or network, look like I open it, I write some dates to it, or I open it and I read from it.

4:00  
OK, general introduction.

4:02  
Now in this module I'm only going to cover the basics.

4:06  
So we're looking at text files where we don't have any formatting.

4:12  
We also have a quick look at so-called, separated value files.

4:16  
You can write these out from Excel or something else where we store, we're storing the data in tables.

4:23  
So imagine we have some header row with label names and then underneath we've got columns of data and we separate to those columns by a comma.

4:33  
We can also have so-called Java, sorry, JavaScript object notation or Jason for short.

4:40  
Now this is again just text, but it's text formatting in a particular way that we can read it or write it more easily from a programme.

4:50  
You often see Jason used for web service communication like again next semester.

4:55  
If you want, you can have a look at the web service example which does this.

5:00  
XML is often used, so the office files that is for Microsoft Office are actually zipped XML files.

5:09  
So if you unzip them, you can look at the XML inside.

5:12  
And then lastly, something special for Python is a so-called pickle.

5:16  
Now what is a pickle you say?

5:18  
Well if you've got an object, that means you've taken a class, you've instantiated an object, then you can just save the object straight to the disc.

5:26  
Now that's what we call a pickle, and a pickle can then just be read straight back into memory.

5:33  
So it's essentially what The thing is in memory.

5:35  
Save it or read it.

5:37  
Now, Pickles are good if you've done a lot of work before making that object.

5:42  
So imagine your programme has done a lot of things, and then you go, all right, OK, we're now going to save the pickle in case somebody else wants to carry on.

5:50  
They're bad because if you rely on that as an input, you are directly reading it into the computer's memory without any cheques.

5:57  
And so potentially you might have some issues.

6:00  
Now, if you want to read about Pickles, I've left in the Python programme examples, some examples of Pickles, but I'm not going to assess you on Pickles, so you can switch that off for now if you wish to.

6:12  
All right, So what kind of file operations can we do?

6:15  
Well, all sorts really.

6:16  
We can create a file path.

6:19  
Now you say, well, what is a file path?

6:22  
It is a full address to where a file is on your computer's hard disc or some other place.

6:29  
So imagine if you're on Windows, you see C: back slash and then something like users back slash your username back slash some directory back slash file.

6:42  
OK, so that is a file path.

6:44  
It's an address to where a file is.

6:47  
Now you can have file paths which are relative, meaning you were saying from where I am now to some file.

6:53  
Or you can have absolute file paths where you start from the beginning, which is the example I just gave you with C: back slash.

7:01  
Now if you've been paying attention, you'll realise that the file paths are not the same on Windows or Linux or Mac.

7:10  
Actually Linux and Mac are the same.

7:12  
Windows is the odd one.

7:14  
So that means when we write them we can't just use text strings.

7:18  
And I'll come to that in a minute.

7:20  
We might want to test if the file exists.

7:23  
So imagine our programme is going to read something.

7:26  
We don't want it just to crash.

7:28  
It might want to check is that actually a file?

7:31  
If not, tell the user that we have some problem I am recording on today.

7:39  
Yeah, good.

7:41  
So let's get rid of that.

7:44  
So what we can do is that we can go ahead and check is it there.

7:50  
Now the first thing we do before we check is it there is we want to make the file path.

7:56  
I'm just going to hide my panel again.

7:59  
So in this little bit of Python, I've imported OS PATH and you can see I've got 2 text strings here and I've said to it join so I'm joining this text string to this one.

8:11  
So what happens here is Python, depending on the computer operating system, whether it's Windows or Mac or Linux, we'll put in the right type of directory separator.

8:24  
So you remember on Windows it's a back slash and on a Mac or Linux it's a/ So if I use this and I say OS path join my dear here, my dear is supposedly my directory name, a comma and then a file name depending on window.

8:41  
If it's on Windows, it will add a back slash.

8:44  
If it's on Linux or Mac, it will add a/ And then when I try and open this, it will work if I just use a text string here.

8:51  
So if I imagine I put my dear back slash my file, if I ran that on Mac or Linux, it would crash because it wouldn't find the file anyway.

9:01  
So we have this library called OS path, which is useful for joining paths together and checking if file exists.

9:09  
So anyway, here are the examples.

9:11  
That's the Windows version and that is the Mac version where these are absolute paths.

9:16  
And here I've just used a relative path.

9:18  
So this would be my dear back slash my file.

9:22  
OK, so that's file paths you can check also if files exist, you can ask is this thing a file?

9:31  
You can also ask is it a directory to check if the person's given you what you think they should have given you.

9:38  
So here this is file just returns true or false depending on whether that file actually exists.

9:46  
Now, because I haven't prefixed this with a directory name, it's going to look in our present working directory for this file, myfile dot TXT, and if it's there, it'll return true, and if it isn't, it'll return false.

10:02  
You can also check the size of a file, that's easy as well.

10:06  
You just use get size and the file name or the file path here, and this then returns the size in bytes.

10:14  
So there are all sorts of operators in this path library module, which I'm not going to discuss all of them, I'm just included a few useful ones.

10:25  
You might want to use get size to say check is the file there if it's got any sort of reasonable size or is it close to 0?

10:35  
All right, I'm now going to discuss a few little examples and then run through some code.

10:43  
The first one is writing a text file.

10:47  
Now when you write a text file, you need to open it first.

10:51  
The opening gives you a connection.

10:54  
So this thing here is I've called it output text file.

10:58  
You can call that variable whatever you like, but this variable contains a connection to that open file.

11:05  
Now notice I've done two things here.

11:07  
I've given it a file name and I've also given it W Now W here means write.

11:13  
I wish to have a connection to this file that I can write into.

11:18  
Now if I just have W on its own, what's going to happen is that file is going to be cleared.

11:24  
So when I write, I'm going to write from the beginning.

11:28  
You can actually write with a pend as well, or you can read.

11:32  
Now, if we don't have any specification here as to what we want to do, the default is read.

11:38  
So if I omitted this, it would be a read connection to the file.

11:42  
Anyway, once you've got a write connection to a file, you can then just go ahead and use write the function and write some text in there.

11:50  
So in this case, I've got a text string and I've appended a
To it.

11:57  
That's a
Character.

11:57  
Now for the clever people in the room, the clever people who know programming in other languages, the
Character on Windows and Mac and Linux is not the same.

12:10  
So Windows has a different
Character with respect to Mac and Linux.

12:13  
However, Python is clever enough that when it writes a back slash
It, will actually write it in the correct format for the operating system it's running on so.

12:23  
Unlike the path join you, don't have to worry about this one now.

12:28  
You can then go ahead and close the file now.

12:31  
There is another way of opening and writing to a file so, you can actually use the with operator so.

12:36  
You can say open the file name as sorry with, open file name as and, then something else all.

12:44  
Right now I'm Not teaching that because you end up with more and more nesting depending on what you're doing.

12:51  
It's kind of preferable in some cases if you put it in your assessment and it works fine.

12:57  
But I'm just teaching this way cuz it reduces the amount of nesting and potentially it's less complicated in some cases.

13:04  
OK so that was for people who care about with and open.

13:08  
Otherwise, just ignore me.

13:10  
Once we've written a text file, we can then go ahead and read it.

13:13  
So reading is very similar.

13:16  
We connect to it.

13:17  
So we say open written text file.

13:19  
So you've noticed that's the file I wrote here.

13:22  
And this time I've explicitly said R.

13:25  
Now I could, as I've already said, omit this, as in don't have R here and it will just read by default.

13:30  
Personally, I like to put R just to remind myself, yeah, it is a read connection.

13:36  
It's up to you.

13:37  
And then you've got a connection here.

13:38  
And once it's a read connection, you can go ahead and call read.

13:41  
Now there are several read functions.

13:44  
You could read the whole file.

13:46  
You could read a line or read all lines.

13:49  
And this read in this case is reading all the file.

13:53  
So the entire text file is then returned as a text string.

13:57  
So you can decide if you want to read a line, read all the lines, or just read the whole file, which is what I've done.

14:04  
Now, once you've got file content, you often want to remove any leading or trailing white spaces.

14:12  
A white space is either a space or a tab or a
So imagine somebody has written some text and they've written into the file, but they've left the trailing
At the end.

14:23  
You don't want that, so you can use strip.

14:27  
Strip will remove any space that is a tab or a space at the beginning and at the end.

14:34  
So then you have a nice clean string.

14:36  
So it's a good idea to do that if you're reading from a text file, or even if you're reading from a field within a table.

14:44  
Better to strip back the white space anyway.

14:48  
In this case, I'm then just printing the text and I'm finally closing the file.

14:51  
Now, yes, I could have closed the file here just after I've read.

14:57  
That's right, I could have done there, or I could close at the end if I was going to do some other operations.

15:01  
Anyway, normally speaking, you only leave the file open as long as you need to.

15:07  
So once you've done whatever you're going to do with the file, close it essentially, because then you are reducing the risk that somebody fiddles with the file or does something else while you've got it open.

15:19  
So it's probably a good idea.

15:20  
Just close it when you're done.

15:22  
OK, so that's reading a text file.

15:24  
Does anybody have any comments or questions so far?

15:30  
Yeah, OK.

15:39  
There are two ways of doing the same thing.

15:42  
In Python as well as other programming languages, there are many ways to do the same thing.

15:47  
So pathlib will allow you to join paths a little bit like this, but with a slightly different syntax.

15:56  
OS path join will also allow you to do it.

15:59  
I'm just using this one to kind of reduce the amount of stuff I'm teaching.

16:03  
If you are going to, sorry, if you want to go and learn to use path Lib and use that, fine, don't mind.

16:08  
But I've just, you know, you have a choice.

16:10  
Like what?

16:11  
Which bits am I going to teach?

16:12  
Am I going to teach this one and it's alternative or?

16:15  
Yeah.

16:15  
So yeah, yeah.

16:21  
Path Lib.

16:22  
Yeah.

16:23  
In some ways, path Lib might be easier to use.

16:26  
It might be more confusing.

16:28  
It's, I mean, there are some things in pathlib which aren't in path.

16:32  
And anyway, so if you want to use pathlib, go for it.

16:35  
It's doing the same job.

16:36  
So in pathlib here you'd have a/ character and I think inside probably pathlib is using operator overloading to say/ character and then string concatenation depending on the operating system.

16:51  
So if you'd like to use that, go for it.

16:53  
I'm just showing you this one.

16:55  
OK Anybody else?

17:02  
All right, let's carry on.

17:04  
So in terms of debugging this thing, let's go ahead and debug.

17:08  
So here I've got a little example, and I already have a file called myfile dot Jason, but I don't have a myfile dot TXT.

17:19  
So we can go ahead and run this thing.

17:23  
And lo and behold, what does it say at the bottom here?

17:27  
It says if I make this bigger.

17:32  
So we've got myfile.

17:36  
So we've printed that out and that's the file path.

17:40  
So all this is doing, let's debug.

17:44  
Debugging's probably easier on this tiny screen.

17:50  
We're on debug, so yeah, you could use pathlib to do this if you so wish.

18:04  
And so remember, I'm on Windows, so this is going to use a back slash.

18:07  
If it was on Linux, it would use or Mac it would use a/.

18:12  
So we'd run a debug and other computer programme languages have or some of them have a similar idea.

18:22  
So you see here I've got a text string which is my dear back slash back slash my file dot TXT.

18:29  
Now you're saying to me, well will why does it contain 2 backslashes?

18:33  
It's because the first one escapes the second one.

18:37  
So when it's written, it's actually one back slash.

18:42  
OK, so that is that one.

18:44  
And we go back to here we check if the file exists.

18:49  
So in this case I'm checking does myfile dot TXT exist.

18:52  
You can see I don't have said file.

18:55  
So what's going to happen is that this is just going to return false.

19:10  
Oops.

19:12  
And here we are.

19:13  
And it says false because that file is not in my present working directory.

19:16  
Now what I've done here is I've said file open folder.

19:21  
And you'll notice if I stop this.

19:24  
So because I said file open folder, if I open a terminal, let's see if I can open a terminal DD screen terminal.

19:34  
So it's a new terminal.

19:37  
I'm currently SAT in the place where the example code is.

19:41  
So because I'm in the present working directory of that code or that file, I can say open that file and it will look in that directory.

19:51  
If you don't use file, open folder.

19:54  
You will be running in your home directory and you'll be wondering why you can't open a file in another directory.

19:59  
OK, so for all my examples, I'm not asking you to have absolute paths.

20:03  
Just use file open folder so that you're sat in the present working directory.

20:08  
OK, so in terms of writing a file, so there's the code we talked about earlier.

20:15  
And all that does is it just writes out a little text file here a text string, and there you go.

20:22  
If we do the read text, read text file, we can do that and we'll just get it back.

20:30  
Well, can I just ask, Yeah, first one to file part.

20:36  
Yeah.

20:38  
See if you just sort of open it to write, is it automatically created?

20:42  
You don't have to do that.

20:43  
Yes, if you just if you open to write and the directory is valid.

20:48  
I mean, imagine you haven't got a directory, right?

20:50  
If it's just a file and you say open to right, it will create it.

20:54  
If it's already there and you've just used W, it will open it and it will remove the content.

21:00  
You there's you can open it to append.

21:03  
I've forgotten the command.

21:05  
I think it's W plus somebody could correct me later.

21:08  
So you can choose do I want to create or clear which is W?

21:13  
Do I want to append or do you want to read?

21:16  
Now if it doesn't exist, it's completely fine.

21:19  
You can write to it and it will make the file.

21:21  
If it doesn't exist and you try to read from it, then you'll have an error when you try and open it.

21:29  
Yes, exactly.

21:31  
Yeah.

21:31  
So it's a specific folder.

21:32  
Do I want it to be in like the users directory, the home directory where the Python file is?

21:37  
There are lots of ways of doing this.

21:39  
OK?

21:39  
So in this case, we've written, we've said file, open, read, oops, sorry, and there's our text.

21:45  
We've got it back our text string.

21:49  
So let's go back to the code, OK?

21:52  
So we're going to go on to CSV files.

21:55  
So let's go on to CSV files.

22:01  
OK, so CSV files, the idea is very simple.

22:06  
You've got a value, you have a comma, and then you have another value.

22:10  
Now you can have more than one row, maybe several rows.

22:15  
It's up to you.

22:17  
Here we go.

22:17  
We're going to open a file connection.

22:19  
So we say my file CSV.

22:22  
So this is I've just used my file CSV.

22:25  
It can be called whatever you like dot CSV again W because I'm writing the file.

22:30  
So it's the same thing to start with.

22:32  
We're opening an output file connection.

22:35  
Now the one thing you'll notice, which is a bit odd, is I've set the
To be null actually, empty now.

22:43  
Why you do this if, you look at the documentation for the CSV WRITER is it needs to have control over this
Itself all.

22:51  
Right so?

22:52  
That's why you set this
To null or to an empty string rather and.

22:57  
Now we have an open file connection in the same way we did when we were writing a normal text file and.

23:02  
Then we can open the CSV WRITER so.

23:04  
You see here I HAVE to import the module CSV AND I CAN say CSV dot writer, I give it the output connection.

23:11  
So that's where I want to write to.

23:14  
I can tell it what kind of delimiter, although this module is called CSV writer, instead of having commas, you can pick another character.

23:21  
You can have a pipe, a back slash, whatever you want.

23:24  
You just tell it what do you want to separate the columns with.

23:28  
So normally it's a comma.

23:31  
When you have a text or piece of text which actually might contain a comma, then you want to quote it, meaning you put a double quotes around it.

23:40  
This is just the CSV standard.

23:42  
So normally the quote character is a double quote.

23:46  
Again, if you want to have a different quote character you can.

23:49  
You can tell the CSV writer you want something else like an exclamation mark or something weird.

23:54  
Now the quoting style the CSV writer will give you is that it will quote non numeric, which means text strings basically.

24:02  
So if it's not a number, it will put quotes around it.

24:06  
You can have quote non numeric, quote minimal, quote all.

24:09  
Now mostly excel.

24:12  
When you export from it will quote minimal.

24:15  
But when you're writing, you can decide, do you want to quote just the text strings or whatever?

24:19  
Anyway, once you've got the writer, so we've now got a writer, you can then write a row to it with this command or this function here, write row.

24:29  
So write row is a function.

24:31  
You pass it a list.

24:33  
All right, So it's just a list.

24:35  
I know it's inside these parentheses, but it is just a list.

24:39  
So I've got host IP and then I can write another row here and I've got local host and this IP address.

24:47  
And then I've closed the file.

24:49  
And what this looks like when you run it is you've got the 1st row and then the 2nd row.

24:55  
So the CSV writer has helpfully put in the comma here for us.

25:01  
And because we've said quote non numeric, it's put the quotes around it.

25:06  
OK, so that's it.

25:07  
So writing a CSV file, all right, So to read it back is a similar idea.

25:15  
You open the file and you say R, or if you want to, you can emit R and it will be R But I prefer just to use R to say, yeah, it really is R There's really a read only connection.

25:27  
Again, I set the
To be empty string because the CSP reader needs to control this.

25:32  
What the CSP reader is doing is it's actually reading and writing the correct
But, it the manual says I SHOULD set it to be an empty string fine.

25:41  
So, I've Got the reader here I.

25:43  
THEN have the input file connection you.

25:45  
Can see there's the input file connection I've.

25:47  
Set the delimiter again explicitly if.

25:49  
You don't set it by, the way by, default it will be a comma and it will be a double quote I've.

25:53  
Just put it in so you can see that you can change it and.

25:56  
Then once you've got the reader connection, this reader behaves like a list or some other object you can take values from.

26:05  
Imagine the playing card analogy again.

26:08  
So we can basically say I'd like a row, I'd like another row, I'd like another row, and it will keep giving you rows until there's no more rows in the file, in which case the fall it will stop.

26:18  
So here you can just say I want a row from this file, and what appears here in this variable row is the list from that row.

26:28  
So you end up with a list, so the first column and the second column separated inside the list.

26:37  
OK, so you can write a list to a CSV file and read it back again.

26:42  
And then, yeah, when I'm done reading, I need to close the file and there we go.

26:46  
Anybody got any questions about that?

26:51  
Yeah, Why am I closing it?

26:56  
So if you leave file connections open, you you are using some kind of resources.

27:04  
You, it's, it's generally a good process to close them if you use with, so with open as whatever, then when you've exited the sort of with part, it will automatically close it.

27:21  
If you don't do that, you have to close the file on a small scale.

27:26  
It won't hurt you because if you leave a few files open, the computer will sort of be all right.

27:31  
If you do this a lot, the computer will slow down because it will have more and more open file connections on its file system, and the actual file system will slow down.

27:41  
So it just sort of there is a kind of memory of what files are opened right now.

27:49  
The worst case, the worst thing you can do is you open a bit like this, lots of connections to a database and then you're not closing them.

27:58  
And so the database has all these connections open, open, open, open, and eventually the database runs out of available connections.

28:04  
So it's a good discipline to get into just if you're not using with to explicitly make sure that you close the file because otherwise if you run a lot of open connections, that computer will just slow down and eventually stop because it's keeping track of all the open connections.

28:21  
OK, yeah, there is there an advantage using one or the other?

28:30  
Well, with it's fine if you, if you've got a simple thing.

28:36  
So you're just saying with this file as whatever, do some stuff and then when you come out with it automatic closes it.

28:44  
The problem with with is when you have to do lots of things with the open file.

28:49  
So you've you've got a indent for with and then you've got a whole bag of mess.

28:55  
And then that's all indented as well.

28:57  
And then maybe you don't want to close the file there, you want to close it in another function somewhere else.

29:04  
But with won't let you do that.

29:06  
So with is good.

29:07  
If you just want to do something simple like here with would be fine.

29:11  
I'm just showing you this way of working because in the generic, sorry, in the most general case, with is not helpful.

29:19  
So it's, again, it's a bit like path Lib.

29:21  
It's another way of working, which is good for some things, but not great for others.

29:26  
All right, go.

29:27  
You can go and have a look at widths and use it.

29:29  
I don't mind.

29:30  
I'm only teaching one of them because this is fine for the general.

29:34  
So as I said, you could close the file in another function.

29:38  
So you could open the file, pass that file connection to a bunch of functions, do some stuff, and then close it somewhere else.

29:46  
Right now, obviously that's going to be a bit more work, a bit more complicated, but it's not unheard if you want to do that, because sometimes the file is too big and you can't just read it all into the computer's memory.

30:00  
But you can read it and process it as you go.

30:03  
And therefore you want the file to be opened during whatever you're doing, and then when you're finished, you close it.

30:08  
A lot of Python examples assume that you can read the whole thing into memory, which is not always true.

30:18  
Yeah.

30:19  
Purpose of
This one here.

30:21  
Yeah, right.

30:23  
So normally when we write into a text file, it will put a
Character in, right?

30:30  
And as I've said already, the
Character, the way what's actually written depends on the operating system.

30:36  
Is it a Mac or is it a Windows or whatever.

30:39  
So here we are basically setting this to be empty so that the CSV reader is going to do that work for us.

30:46  
OK, so it's just that the this if you read about CSV reader in the documentation, they say you should do this.

30:54  
You know, it's like cooking.

30:56  
If you're making a cake, you end up with so many ounces of flour and sugar and butter and add two eggs or whatever and you mix it up.

31:04  
And it's the same with a lot of these functions is you just follow the recipe.

31:08  
So they're telling you to set it to an empty string because inside the reader it will handle it.

31:15  
Now, if you don't do this as you don't set it to an empty string.

31:19  
If you read and write to, say, Windows, just Windows will be fine.

31:23  
But if you read on, sorry, if you wrote on Windows and read on Linux and you didn't do this, the CSV reader can fail in some cases.

31:32  
All right, So it's just to hand over that job to the CSV library.

31:37  
OK, great, fine.

31:41  
So I'm going to go on to Jason, and then I'm going to run through a few other examples.

31:46  
So what is Jason?

31:47  
Jason, as I've already said, is just a text file which is formatted in a particular way.

31:53  
Now, you can send Jason over a network connection as well, but in this case we've just got a file.

32:01  
So I start off by importing the Jason module.

32:06  
I made here an empty dictionary.

32:07  
It's just normal empty dictionary and I've given it a key here of local host which has an associated value of a text string.

32:15  
All right, nothing complicated, this is just standard Python.

32:19  
I've then gone ahead and opened the text file, which happens to be a Jason file, but it's just open again.

32:26  
I've used W because I want to write to it.

32:28  
So it's a new file.

32:30  
Now I've specified the encoding.

32:32  
Now you say to me, what is encoding?

32:35  
Now the way text is written, it depends on the character set.

32:39  
So you can have, for example, if you speak another language, you will have text with special characters, like in French you have E acute or where you have the accent on the top, or you can have the circumflex.

32:55  
Now that's not on a standard UK keyboard, that's in an extended character set.

33:00  
If you write in Mandarin or something else, you have a different character set.

33:04  
So when you're writing, you can basically specify what kind of encoding of text do you want.

33:10  
Now, UTFA is fairly standard in some cases.

33:14  
For example, when you're reading a text with extra like, well, extra characters that are non standard, you need to change this all right.

33:25  
And if you don't, when you read it back, you'll end up with some weird character in computer's memory.

33:32  
Anyhow, so I'm just introducing this.

33:34  
Yeah, here is something that you might need to worry about.

33:37  
It's nothing to do with the fact that we're writing Jason.

33:39  
It's just I've dropped in here as something else to think about.

33:43  
OK, so we can then say Jason.

33:45  
So you see, that's the module, Jason.

33:47  
Dump is the function.

33:49  
And the first thing is the thing we want to write.

33:53  
Now when we're writing with Jason dump, we need here either a dictionary or a list normally.

34:00  
So we're going to give it the dictionary or list.

34:03  
And the way this library Jason dump works is that it has to be some sort of library or dictionary which has inside maybe other libraries or dictionaries.

34:15  
You can't dump objects.

34:18  
There are other libraries or other things you can do to dump objects.

34:21  
I'm not going to go into that because that's extra complexity.

34:24  
But basically here we can just give it a dictionary and it will write again.

34:29  
We give it the output file connection.

34:31  
Here I've I'm using another feature which you don't actually have to use.

34:36  
So here I'm allowing the possibility of writing some text which isn't humanly readable.

34:42  
So ASCII is the normal character set you see on your keyboard.

34:46  
It's the American standard character set, so I've set that to false to allow me to write some other text, which is is perhaps in a, you know, the rest of the character encoding set.

34:59  
Now if you want to write Jason, that's humanly readable, meaning some users going to read it, You can set the indent to four and then it will format this nicely.

35:10  
If you leave this alone, it will just write it all on one line, which is fine for computer, but it doesn't look very nice for humans to read.

35:19  
And then finally we close it and you can see this is the output here.

35:23  
It's written the curly braces, which looked like a Python dictionary, the text string, this is just a colon and then another text string.

35:33  
OK, so it's written the dictionary straight to the file.

35:36  
Now you can see already.

35:38  
This is really handy.

35:39  
If you've got dictionaries and lists inside, you can just write them to the file.

35:43  
You don't need to worry about Pickles.

35:45  
Just chase and dump into the file and you can do the same thing.

35:50  
You can read it back, you can open the file, read connection.

35:54  
Notice my encoding here matches the encoding I use to write.

35:58  
If you use one encoding to read and another to write or whatever you mix them up, then you run into trouble.

36:04  
So same encoding, I'm loading the content by.

36:09  
I've got the input file connection.

36:10  
I give it to this function.

36:11  
Load and load just reads whatever is in that file, and we've now got it in this variable called Jason data.

36:20  
Now, if the Jason file is badly formatted, like there's a missing curly brace or whatever, this will actually throw an exception.

36:29  
You can play around with that and see what happens, if you so wish.

36:33  
Now, once we've read, we can then close and you can then see we can print.

36:37  
And what we have in the computer's memory here is we've got the dictionary back again, which has inside the key and value pair that we wrote.

36:45  
And there we go.

36:48  
Now I've got any more.

36:49  
No, I don't.

36:50  
So before I go and look at some more examples, does anybody have any questions or comments about this one?

36:56  
Yeah.

37:01  
Default value for what?

37:02  
Sorry, yes there is a default value.

37:07  
I personally prefer often just to say encoding UTF 8 because I think the if you do like a coding style check with pylint, it will pick you up on assuming the default because you're sort of being a bit vague.

37:21  
The same way if you don't use R you're like OK it is really a read file so it's probably best just to put your coding UTF 8IN anyway.

37:31  
But yeah, all right, let's go and have a look at some code.

37:38  
So I'm going to go to a bit more complicated thing.

37:43  
You can play with the other examples as you wish, right?

37:48  
So this is a different example of using the CSV reader.

37:52  
Now what the CSV reader can do is instead of reading each row as a list, you remember that you can also read the table as a dictionary.

38:04  
OK, how does this work?

38:06  
It means that now each row has the key as the column name and the value for that cell in the in the input data.

38:16  
So anyway, here is the here are the input data.

38:19  
So you can see I've got a header row with a name, a score a day, and I've got some names and some numbers and yeah, I've got some more text.

38:29  
Notice the number here is not quoted.

38:32  
OK, so let's go ahead and read it.

38:37  
Not this one.

38:38  
That's the normal reader.

38:40  
We want this one, the dict reader.

38:44  
Let's run debug and see what happens.

38:47  
So this is particularly helpful where you've got a header name when it loads.

38:55  
Well, you see, is it the header?

39:07  
Is it the new keys?

39:08  
Is it?

39:09  
Yes, that's correct.

39:10  
You'll see in a minute.

39:12  
So we open the file, we open a Dictreader.

39:16  
So again, this dict reader here behaves as if it's a list.

39:19  
So we can just take, keep taking values from it one after another after another until we run out of rows in the file, and then the fall loop will stop.

39:27  
So let's loop once.

39:29  
So now you can see that this row contains, you can see the keys there, the name, the score and the date there, the column names and the values are just the values for this row.

39:42  
So we can loop around again, and this time key, if I clear it again, this time you can see the keys are the same.

39:53  
They're the column names and the value names are the values in that row.

39:56  
So we've got Sally 17 Tuesday.

39:59  
Now if we hop forward one more, you can see then the for loop will stop because we've run out of rows.

40:06  
And there we go.

40:07  
We close the file.

40:08  
OK, so the dict reader is sometimes really handy.

40:11  
Because you can then pick it up as a dictionary and access directly without having to work out which column am I in, which is what you'd have to do if you're using a list.

40:20  
OK, so I'm going to throw in one more interesting examples for the people who like interesting things, and then we'll stop.

40:28  
So in this case, I've got a nasty problem.

40:31  
I've got a bad CSV file which I cooked up earlier, so I'll show you the file.

40:37  
So here we've got a problem because the first 3 lines are not actually formatted in the same way as the rest.

40:45  
Uh oh.

40:47  
So you can see we've got a text string.

40:49  
We've got, you know, a text string here, which looks like an empty dictionary.

40:53  
We've got 2 exclamation marks.

40:54  
And then we have the content that we really want to read.

40:58  
So how do we deal with that?

40:59  
OK, so we're going to use a mixture of reading text and CSV file reader.

41:05  
So let's have a look how we do this.

41:10  
First of all, we want to open the file.

41:13  
All right, same as before, we've just got a file open connection.

41:17  
Notice I've set the
To be an empty string because I'm going to use the CSP reader in a minute.

41:22  
Now I've written a little function called seek.

41:25  
So the idea here is I want to move down the file to where the first is.

41:31  
Now when you're reading files, what actually happens is you've opened a connection to the file and then when you read, it's got a cursor and it moves down the file all right.

41:41  
And each time you read, you skip down the file.

41:44  
So if you say I want more text, you'll end up at the bottom of the file and you'll, you'll hit the end of file essentially.

41:50  
So you can say move the cursor back up the file, go back.

41:55  
So that's what I'm going to do here.

41:57  
Now once I've found my, I can then just hand the file to the CSV reader and read the rest in the same way we read before the rest of the file.

42:06  
Let's go and have a look at this C, function.

42:10  
So what am I doing here?

42:12  
Now I've passed in here the file connection.

42:14  
Now I've called it text IO because this might not be a file, it might just be a text string opened as a file connection.

42:21  
So I've just called it text IO and down here we've got a while loop.

42:26  
Why have we got a while loop?

42:27  
Because we don't know how long we're going to have to loop through the file before we reach the common, which is why we're using a while.

42:36  
You can see how I've got while true.

42:38  
Now tell tells us the position of the cursor.

42:42  
So it just tells us where are we in the text file.

42:45  
A bit like, you know, if you're wandering around with your phone, you go where am I Thanks, Google Maps.

42:50  
This is where I am.

42:51  
So it's the same idea.

42:52  
It's just telling you where you are in the file and then I can read a line.

42:57  
Now remember reading a line will read the whole line and it will advance me one line down the file.

43:03  
And then I can ask the is the line an empty text string?

43:07  
Now that if it's an empty text string, I've reached the end of the file.

43:11  
That's what happens when you've run out of text.

43:13  
You've just gotten on the left if you're using readline.

43:16  
So in this case, yeah, I've now got an empty text string.

43:19  
So I'm going to exit and I'm going to say false.

43:21  
I've not found a comma.

43:22  
So I've decided in this function I just want to return false to tell the user I've not found a comma.

43:27  
However, if I do have some text which is not of 0 length, then what I'm doing here is I'm just saying is there a comma in the text?

43:36  
So I can use if in and this is a piece of text, is the comma in the text if it is, what I want to do is I want to rewind some seek here.

43:47  
So I'm going back to where I was all right.

43:50  
So I'm using tell to say, where am I?

43:52  
I'm reading a line if I find a comma and then I'm going back to the beginning of that line again so that I'm now ready to give it to the CSV reader.

44:01  
And you can see then what I do is I say return true because I'm returning inside a loop.

44:07  
It has the function of just exiting the loop immediately and I then appear with the return value true.

44:16  
Now let's have a go and see how this works.

44:20  
So this is an example of passing a file connection to a function, which is what I was talking about earlier.

44:26  
Let's run this thing and you'll see what happens.

44:31  
Now, occasionally you end up with this kind of nonsense.

44:34  
If you export files as a tutor from Pegasus, it does have some gobbledygook at the top, which is not same as the rest of the file.

44:42  
And you have to do this.

44:44  
So let's close this right?

44:50  
So we've got the open connection.

44:52  
Now we're gonna go into this function seek.

44:57  
So let's hop in, right?

44:59  
So we've got while true, now it's going to tell us the position.

45:02  
So you can see where we are in the file.

45:04  
We're at position 0, where our little cursor is at the top of the file.

45:08  
And we've not moved yet.

45:10  
Now, if we read a line, what will happen is that we've now got some text.

45:15  
So you can see we've got that first piece of text, aha.

45:18  
And now you can see we're on Windows.

45:20  
So we've got a funky
Character we had where Has it gone we?

45:25  
Have this back slash R back, slash N on Mac And Linux It's.

45:29  
Just back slash N so That that's what I MEAN about different U LINE character anyway.

45:33  
It's, read the text from our input text file and it of course is actually text so.

45:39  
Yeah we're, going to jump over that length 0 now.

45:43  
We're going to ask is, there a comma in this text well?

45:46  
The, answer is no there, isn't fine so.

45:48  
We go and loop around.

45:51  
So now we can see our position because we read the line we've advanced by 44 characters down the file.

45:58  
So our little cursor is going to 44.

46:02  
We're going to read a line again, and we're going to check, well, what's in this line?

46:06  
Yeah, OK, this is just these two curly braces and a
Great.

46:11  
Which is, of course not empty.

46:14  
So here's our, check.

46:16  
No, there's no in there.

46:19  
This time.

46:19  
We can see that we're now having read 2 lines.

46:22  
We're now at position 48 in the file, and we can then read the line.

46:28  
There's our line.

46:29  
Now this line has got 2 exclamation marks, so that's not going to help either.

46:32  
So we're going to skip around this time.

46:36  
We've got text, which does have a comma in.

46:40  
Great.

46:40  
We've found our CSP content.

46:42  
Let's hop forward, see what happens.

46:45  
We find the comma in the text, we're going to set the cursor position back to the beginning of the line again, and now we exit with return true.

46:55  
We've come out the loop.

46:57  
What you can see next is we're handing this thing over to the CSV reader, and then we're going to go ahead and read a row.

47:06  
And there you can see I've read the first row just as a list again in this case, which is the name, score and day.

47:13  
All right.

47:14  
And and then once you get to the point of valid CSV, the CSV reader is just exactly the same.

47:19  
OK Anybody got any questions or comments?

47:27  
Yes, yes.

47:42  
So the CSV reader continues reading the file from where the cursor is.

47:47  
So imagine you just opened a file and you're typing in Word or something, right?

47:52  
You type, type, type.

47:53  
Your cursor is at a position in the file.

47:56  
And then you say, OK, carry on typing.

47:57  
You're going to carry on typing from there.

47:59  
So the CSV reader just picks up from exactly where the file cursor is.

48:04  
Now this whole idea of a cursor in a position is part of the file.

48:08  
All right?

48:09  
So when you connected to it as a simple reader, you're just taking and the file will just give you from that cursor onwards.

48:17  
OK yeah.

48:19  
So that's the only works if there's text like that at the beginning.

48:23  
So say if you've got like a line that common and whatnot, and then underneath that there's yes, exactly.

48:34  
So if you wanted to do what you're, let's say you mixed up, you mixed up some mess and then you had CSV after whatever.

48:42  
So if you had a mixture of rubbish, you could read through the file.

48:47  
So you open the file, you'd read all the lines into just normal text, but you'd only read the lines where they had like a comma in them.

48:56  
So then you've got it in text and memory.

48:58  
And then you could give this using the stream IO, which is actually in a future example to the CSV reader.

49:05  
So you'd end up reading the whole thing as text to start with.

49:09  
And then when you're sure you've got valid CSV, then you give it to the CSV reader.

49:13  
The other way you can do it is you could essentially you, well, you can skip over the exceptions.

49:20  
So you can, if there's something bad, you can basically say, all right, ignore that, carry on.

49:25  
So it's probably better that just process it all first and then carry on reading it.

49:31  
Yeah, but you've got choices here.

49:33  
I'm just showing you this nice thing because you can then see the idea of the cursor and, you know, moving up and down.

49:39  
And it's not uncommon that you have this problem.

49:43  
All right.

49:43  
Anybody else?

49:47  
No.

49:48  
All right.

49:48  
OK, well, that's it for today.

49:51  
Play with example code.

49:52  
And I hope to see in the lab, we'll carry on using these file operations in the next few weeks so you get used to them.

50:01  
And we'll use them with classes as well.

50:03  
So you can see that.

50:04  
Yeah, we can use them with whatever we like.