**w05-01-MoreFunctions-2024**

0:00  
OK.

0:01  
So this week we are in week five of our little module.

0:05  
Now this module runs for 10 weeks.

0:08  
The 11th week is essentially an opportunity to practise, to do a bit more revision for the programming exam.

0:16  
So in this week we're looking again at functions, and we're going to play with functions a bit more just to get a feel for what they can and can't do.

0:24  
So it's a little bit of the same sort of thing as last week, but a bit more complicated.

0:29  
And in the lab we've got some more complicated problems.

0:33  
Next week we will be jumping to a slightly different topic.

0:37  
So we'll be looking at object orientation, and that will well coincide with the introduction of object orientation in Java.

0:47  
So a lot of the ideas that we are looking at in this particular module with Python do exist in other languages such as Java.

0:56  
Anyway, that's next week.

0:58  
Before we go on, does anybody have any comments or questions about the module so far?

1:03  
General questions, no.

1:06  
OK, good, let's go on then.

1:08  
So what are we doing today?

1:10  
We're basically talking about more functions in terms of let's get rid of that silly little box.

1:19  
Yeah, what we can do with them.

1:20  
So I'm going to come on to a particular concept, which is recursion, but I'm not going to do that straight away.

1:27  
I'm going to look at some other function examples to start with.

1:30  
So let's go to our code editor.

1:32  
So for those keen people, I'd uploaded the Python files to my place.

1:37  
I did actually change one and I'll come to it in a minute, but not not any of these.

1:42  
Now, one, one issue that students sometimes come up with, and I'm thinking, I don't do this, but you know, you need to think carefully about what a function does.

1:54  
So here we've got a simple function and this function, we are not giving it anything where it's got a name which is meaningful in terms of what it's going to do.

2:04  
Remember, your variable names and your function names should be meaningful, suggesting what they will do.

2:11  
So this one I've called return number because that's all it does.

2:14  
It returns a number.

2:15  
We don't give it an input, and you can see here it returns the value 10, an integer.

2:20  
Now that 10 is sent back from the function and then we can assign it to something else if we want, or we can just lose it, right?

2:31  
If we don't assign it, it's gone.

2:33  
So let's have a look at this other function.

2:36  
Now this other function I've called print number because this one, it doesn't return anything.

2:42  
It actually because I've not said return, it will return none, nothing at all that is.

2:50  
Now to illustrate the difference between these two functions, I've got here the two functions.

2:55  
I'm going to call them and then I'm just going to print out what they returned.

3:00  
So the first one, as we've already seen, is returning a value, and once the value has been returned, it's assigned to this variable, result.

3:11  
In the second instance, the second function doesn't return anything and it will then return None because I'm not set to return anything and none will be assigned to result.

3:22  
So why am I doing this?

3:23  
I'm doing this to make it absolutely clear, hopefully, that print and return are not the same thing.

3:29  
Print puts it on the screen, returns, sends it back to the thing that called it.

3:35  
So if we run this, you just run it and we can see that if we Scroll down.

3:43  
So they were the code.

3:44  
What happened here?

3:46  
Well, the first function call returned the number 10.

3:50  
In the second case, the number 10 was printed, but the function itself returned none because we didn't tell it to return anything.

3:59  
So remember, return and print are not the same.

4:04  
So if you are trying to return a value, you have to return it.

4:09  
If you want to print it on the screen, you print it.

4:12  
I don't hopefully want to see this.

4:15  
Somebody think, oh, I've got a return of value.

4:17  
No, you're not.

4:18  
Now some of the confusion arises because when you're using the Python interpreter in interactive mode, if you don't assign it, it will print it.

4:28  
Now, this is a feature of the Python interpreter.

4:31  
It's not should we say, returning equals printing or anything like that.

4:36  
So I'll just show you where maybe the confusion comes from.

4:40  
So if I open Python, so I'm just going to do this here.

4:45  
So this is a Python interpreter window.

4:48  
Let's see if we can make it bigger or not.

4:50  
Maybe not.

4:52  
What I can do here is I can say let's define a function so def def fun.

4:57  
And our function is going to return something.

5:02  
So it's going to return 32 in this case, return 2.

5:05  
Great.

5:06  
Now if I call this function by just saying fun with no inputs, then what will happen is you can see it prints 2 on the screen.

5:17  
Now that is a Python interpreter helpful feature.

5:23  
It's being printed because I didn't assign it to anything.

5:26  
So if I assign it to something, let's say X equals fun, it's now not printing it because it's been captured in the variable X.

5:35  
So don't be confused.

5:37  
Return and print are two different things.

5:39  
The interpreter helps you by putting stuff on the screen that hasn't been assigned to a variable.

5:47  
All right, So that's a feature of the interpreter and it's not a feature of the language itself.

5:53  
So hopefully that clears that up.

5:55  
Does anybody have any comments or questions about this before we move on?

5:59  
Yes, something function.

6:04  
Yeah, yeah, quite a lot really.

6:09  
So the return is normally used if you are returning some sort of hint that something worked properly, right?

6:18  
So you're either returning a block of data, which you can do in Python.

6:22  
In other languages, you wouldn't do that.

6:24  
You're just returning a hint that it works like you write.

6:26  
You might return an error code, say 0 is good and anything else is bad.

6:32  
Or you might return a boolean to say true or false, it passed or failed.

6:36  
But you may have some functions that just going to read or write data maybe.

6:40  
And instead of returning something, you've taken the decision that you're just going to throw an exception, right?

6:46  
So this thing, you're going to call it and if it fails, it will throw an exception.

6:52  
Then you'll have to catch that exception.

6:54  
So it's a choice.

6:56  
I've got a few functions in a minute which I'm going to discuss.

6:59  
We don't have returns, but it's about thinking how does the data flow around or how do the data flow around my programme?

7:07  
Should I be returning something to this function or not?

7:10  
It's up to you what you're doing.

7:12  
So there are plenty of standard functions where you don't have anything returned and some way you do.

7:19  
So if you are going to pass back a simple value, maybe you should consider returning it because that makes more sense to somebody rather than putting it in a list and then sort of hiding the fact that you're returning it because the list is mutable, right?

7:35  
So if you're just going to return add number, probably just return it, OK.

7:41  
So it's a design choice really.

7:42  
Anybody else?

7:46  
No.

7:46  
OK, let's have a look at the next one.

7:50  
So in this in this little programme, what I've got here is 2 functions.

7:57  
The first function is calculating the area of a circle and the second function is calculating the area of a square.

8:04  
Now in this case I've passed in a value.

8:08  
So I'm saying in the case of the area of the circle, the value I'm passing in is the radius.

8:14  
I'll show you in a minute.

8:15  
It's the radius in the area of the square case.

8:18  
I'm passing in the length of the side.

8:19  
Now I've just made these functions up again.

8:22  
The name is meaningful as in all right, area of circle.

8:26  
Great.

8:26  
Yeah, we're going to pass it in.

8:28  
It's going to give us the area of the circle back.

8:31  
Let's have a look at them.

8:32  
So it's simple maths.

8:33  
So we pass in here the length of the one of the sides of the square and you can see all that happens here is the area is just the length squared and and we return that.

8:46  
So here we have a simple value and we've just returned it back.

8:50  
We put in a value, it returns it.

8:52  
I've got a comment here.

8:53  
Again, normal process of Python is to have a multi line comment, which is also a documentation string where we say what the function does.

9:05  
So you can see here I've just left a little comment to say what is there for?

9:09  
What does it do?

9:11  
Now obviously you can have many lines inside your function or you can have one.

9:16  
I've just got one here which returns a value.

9:19  
In this case we have the area of the circle.

9:22  
Again, I've got an input variable and I've put it all on one line, the logic and I've returned the value.

9:28  
Now I could put it on 2 lines, right?

9:30  
So I could have a variable called radius\_SQ which I COULD then assign the radius squared to and, then I COULD do another line where I MULTIPLY it with the value of π and then I RETURN it here.

9:43  
I've Just put it all on one because that's simplest math is the math module so.

9:50  
Since I've Imported the math module at the top here, I CAN then say math dot Pi π is a constant which is defined in the math library multiplied by the radius squared, and I RETURN it.

10:04  
So there you go.

10:06  
The function name suggests what it does, the input variable name suggests what it contains, and then because we are dealing with a simple value, we are just returning it.

10:19  
Does that seem reasonable?

10:21  
Yeah, that could, yeah, Yeah, this will, this will use maybe slightly less memory.

10:36  
Normally what you do is if let's imagine you have a line 17, a much longer more complicated equation, right.

10:44  
So it's got lots of different pieces in it.

10:47  
Yeah, you could put it all on one line.

10:49  
But what happens is you may have a problem somewhere in your programme, and then you'll like, oh, why is this returning -1 I have no idea.

10:58  
So sometimes you want to break the equation down into a few steps so that you can go ahead and debug what's this value?

11:06  
What's that value?

11:07  
And so it becomes much easier to spot why the return value is what it is.

11:12  
In this case, the equation is pretty simple, and in this case, also pretty simple.

11:16  
But occasionally we might want to break it down.

11:18  
I've got actually another example, or I do break it down because I just want to check a value before I put it into something else.

11:26  
So yeah, it's a choice.

11:27  
Again, I would suggest that don't make whatever you're doing in one line too complicated, because then debugging an issue becomes more of a problem because, you know, it's just less obvious what's going on.

11:40  
Anybody else?

11:42  
No.

11:43  
OK, let's go and have a look at another one.

11:47  
So in this case, what have we got now?

11:50  
It would be a little bit more complicated, and we're now dealing with something that is mutable, right?

11:55  
And we have a list.

11:59  
And in this list we have some text.

12:01  
So I've got the word apple.

12:03  
Now the word pear, I've got apple and apple again.

12:06  
So the idea here is I've got a little function which is going to tell me how many times does a word exist in the input list, and it's going to return it with the unique word as the key and the value as the number of times the unique word is in the input list.

12:26  
So this kind of this little function could be useful for text indexing where you're trying to understand a word frequency.

12:33  
Anyway, so we've got here a list, and you can see there's our empty dictionary.

12:39  
Now in this case, I'm not using the return because I've decided that all right, I'm going to put my results, oops, sorry, keep clicking on the right mouse button.

12:51  
I'm going to put my results into this.

12:52  
Dictionary counts.

12:54  
And you remember, a list and a dictionary are both mutable.

12:58  
So when I pass counts into the function, the function can update it.

13:02  
And when we finished with the function, it will remain updated.

13:06  
Now, we'll just go ahead and run this so we can see what it does, and then we'll look at the actual code.

13:11  
So you can see it says the basket contains 3 apples and one pair.

13:17  
Great, let's have a look.

13:19  
Yeah, it does 3 apples and one pair.

13:23  
So that's what the function does.

13:25  
And you can see there's the print out here.

13:27  
I've got the counts dictionary in there.

13:30  
So because it's in those curly braces, Python knows to turn the Python, the dictionary into a text string and then it's all printed on the screen.

13:40  
Let's go and have a look at the count names function.

13:42  
What is it doing?

13:45  
So again, I've name the function in a somewhat sensible manner.

13:50  
OK, it's called count names.

13:51  
You could call it count unique or something else meaningful.

13:56  
I've got an input list.

13:57  
Yeah, that suggests it is a list.

14:00  
And then I've got this variable counts.

14:02  
Now you could call it something else.

14:03  
You could call it unique counts or something that's meaningful.

14:08  
Anyway, the first step I've done here is I've cleared the dictionary.

14:13  
Now, why have I done this?

14:15  
Well, I'm paranoid and I'm thinking if somebody else calls my function and then calls it again, what will happen?

14:24  
So if they call it once, it will fill up the dictionary because of course the dictionary is to start with empty.

14:30  
But then if they call it again, it's not going to work in the same way because the dictionary has already got something in it.

14:36  
So I'm not going to let that happen.

14:38  
I want my function to behave in the same way no matter how many times it's been called.

14:43  
So I'm just going to clear the dictionary at the start.

14:46  
And again, this is a design choice.

14:47  
You might want your function to append or add to a dictionary, just keep counting up stuff.

14:54  
But in this case, I want the counts to be matching the input list.

15:00  
I don't want it to be accumulating counts.

15:03  
All right, so that's the design choice.

15:05  
And then what we're doing here is we've just got a full loop, normal full loop that we'd have outside of function.

15:11  
We can have inside of function and we're looping over what is in the list.

15:15  
So remember the illustration of taking cards from a deck?

15:19  
We're taking each one of these words and we're just asking the question, is this word in the list of dictionary keys?

15:26  
If it isn't so if it isn't in there, then we're going to create a new dictionary key called the word, and we're going to have no counts there.

15:37  
Otherwise, we're just going to keep counting up.

15:40  
And what happens is as we loop around here, it's going to increment that counter.

15:45  
And so when we finish the function, this thing counts, contains the word and the associated frequency of that word in the input list.

15:54  
Does that seem reasonable?

15:57  
Yeah.

16:00  
Do you, shall I debug it or is that all right?

16:02  
You're going to debug yourselves.

16:05  
OK, I'll, I'll leave it if you are feeling happy.

16:10  
Yeah, yes.

16:16  
Counts as a dictionary.

16:17  
So you can see down here, we, we set counts to be an empty dictionary.

16:23  
So at this point, we've created a dictionary and we've passed the reference to it to this thing counts.

16:29  
And then the reference to this dictionary that's empty is being passed in here.

16:33  
So if we, well, let's debug it since we've got a question, run a debug, but put breakpoint in, that would help.

16:41  
1 breakpoint and we'll step in here debug Python file.

16:52  
So we'll start off with the empty dictionary counts.

17:05  
And then we'll step in.

17:06  
So I'm going to just close the debug window.

17:09  
So once the cursor arrives at the right place, yeah, so counts is a empty dictionary.

17:18  
You can see it's length 0.

17:20  
We can then hop into the function.

17:23  
What do we have here?

17:24  
The input list is containing these 4 words apple, pear, apple, apple counts.

17:33  
You can see is referring to that empty dictionary that we passed the reference in for.

17:39  
We are saying clear, but this is harmless.

17:41  
It's already empty so saying clear it won't do anything.

17:44  
I've left that there as I said before because I worried about people calling my function more than once by accident.

17:52  
Then what happens is we loop around this is a list.

17:56  
So the first time what's going to be in here is apple, and yet apple is not in the dictionary.

18:02  
So as a key.

18:04  
So we end up with a new key, which has now got zero associated with it.

18:09  
So you can see there's oops, we've got apple, zero is associated with it as the value.

18:16  
And then we're going to count the apple.

18:17  
So now we've counted 1 apple.

18:19  
You can see that we've got Apple one and now we loop around and the second time it's a pair.

18:25  
So Yep, pair is not in the dictionary.

18:27  
So we are going to add that key with 0 and we loop around again, and this time we've got an apple.

18:34  
Hop around.

18:35  
There you go, 1 apple.

18:37  
And this time it is in the dictionary.

18:40  
So we hop here, we increment it.

18:42  
And so you can see that as we loop around, we've now got apple two and pair one.

18:49  
OK, we haven't looked at the last element of the list, but you can see how the function is working now if I keep going.

18:55  
Let's go around one more, and now we've run out of values.

18:59  
We've come back out here and you can see our dictionary, which was empty.

19:03  
Now it contains Apple three pair one.

19:06  
Now it contains that because we passed a reference into it because it's mutable.

19:11  
All right, yes, that 'd be the other way around.

19:18  
So instead of saying value not N, you could just say value N and then just add an else.

19:26  
You could have an else.

19:28  
The reason I've not used an else here is I don't need one.

19:33  
So line 11 has to be done every time.

19:36  
Every time I want to count the word.

19:39  
Line 10 only has to be done if it's not already in the dictionary.

19:45  
So you could say if it's in the the keys pass else this.

19:53  
Yeah, So that that would work.

19:55  
It's just a bit more.

19:56  
Effort in terms of code and yeah, extra complexity doesn't need to be there.

20:02  
So when you're writing ifs and elses, you can think to yourself, do I need an else?

20:06  
No.

20:07  
OK, fine.

20:08  
I won't have one.

20:09  
You know, if you don't need the complexity, don't have it.

20:11  
Yeah.

20:13  
Anybody else?

20:16  
No.

20:16  
All right, great, let's stop that.

20:19  
We'll go and look at the other one.

20:22  
So we've now got a little bit more complicated function.

20:26  
Now in this case, what happens is we're dealing with numbers again and we've got an input list and we've got an output list.

20:33  
Again, the list is mutable.

20:35  
So we're going to fill up that list inside the function.

20:38  
I have a list here of three numbers, so 1-2 and three.

20:42  
And all this function does is it calculates the cumulative total, meaning that it starts off with one and then 1 + 2 is 3 and 1 + 2 + 3 is 6.

20:54  
So it is the triangular number series.

20:56  
But anyway, it, it's just a little function to illustrate a point.

20:59  
So it's calculating the cumulative total.

21:01  
Let's, let's run it right.

21:05  
What's, what's the output?

21:06  
1-3 and six, which is what I told you would be SO11 plus two, 1 + 2 + 3, that's six.

21:15  
OK, that's all it's doing.

21:17  
So let's have a look at it.

21:20  
Go up to the top here.

21:22  
What have we got?

21:23  
Our function is called cumulative total, again suggesting what it does.

21:27  
The input variable here is called input list because again, it is a list and I've got here output total.

21:35  
Now I could call this output list, but it's, yeah, it's the cumulative total.

21:39  
So I didn't want to make it output total list or something.

21:43  
It starts to become a bit long.

21:45  
So you're going to have a choice as to you want your variable to be meaningful, but you don't really want it to be very, very long if you can help it Anyway.

21:53  
So what do we do here?

21:55  
Again, I'm paranoid.

21:56  
The output list might contain something, somebody might have done something naughty and put something in there.

22:03  
And I don't want my function to so-called misbehave, right?

22:07  
So I could come along with the output list before this function call, and I could put some values in, call the function, and then say, oh, the function's wrong or actually it's not.

22:16  
It's the fact that you put something in the list before.

22:18  
So again, here I'm clearing the list because I want my function to do what it should do and nothing strange happened.

22:26  
Then what I'm doing here is I'm checking the length of the input list because I want the output list to have the same size, meaning the same number of elements.

22:35  
So I'm asking how long is the input list?

22:39  
Now this is a quite a nice shortcut if you have a mutable, sorry, an immutable value.

22:45  
So an immutable value like an integer or a string you can use the shortcut.

22:50  
Don't use the shortcut for a mutable 1 because it will just copy the reference anyway.

22:55  
So here I've got a I'm saying a list of one element which contains 0 and the multiplication operator for a list causes it to create several elements.

23:07  
So what this what this actually does is it says I want to list these three elements long.

23:16  
All three elements contain zero.

23:19  
That's what line 10 does.

23:21  
And it then appends that list to this total input.

23:25  
Notice if I had equals here, what would happen is I would have broken the connection between the input list reference and it being returned.

23:37  
What am I on about if I have here like this?

23:41  
What's happened is I've said ah new list.

23:45  
I want to assign a reference to the new list to output total.

23:49  
Which then means I'm just talking to this new list that's inside the function and when I return it's gone.

23:55  
But if I put in the plus, I'm saying, all right, take my new list and stick it on the end of this other one, which I have a reference to which came to my function.

24:05  
So you've got to be a bit careful with append or assignment when dealing with a list.

24:10  
So I'm pending here.

24:12  
And then I'm going around the loop and I'm using this range.

24:15  
So I'm going around some number of times and I'm using the I, the variable I as the position in the list.

24:23  
So as we go around that loop, we're assigning values.

24:28  
So we've got 2 lists that have the same number of entries, which in this case is 3, and we're assigning values into the respective boxes.

24:37  
Now, the other thing I've done here is I'm adding up.

24:41  
So I'm adding the value that I had from the last total up into the next box.

24:48  
And this is where you get the cumulative effect from.

24:51  
Now, I've written in a particular way to, you know, use some of the ideas that you have in the language.

24:56  
You could probably write this in a different way.

24:58  
As we said earlier, you can have a variable which is the total inside the loop, and then you could add to that and assign it to the list, whatever, it's a choice.

25:08  
So let's have a go at debugging this one very quickly.

25:14  
Does anybody have any questions or comments while this debugger is starting up?

25:19  
Actually, I'll put it up here.

25:23  
No, with all these programmes, if something isn't clear, just sit down and run it yourself and go.

25:30  
What's going on here?

25:31  
All right then.

25:32  
Yeah, I can see what's happening.

25:37  
So you see that we're going to run to that breakpoint.

25:40  
The input list contains 3 values, 1-2 and three.

25:47  
And you can see it's got a length of three.

25:50  
And so this thing here, this is a list of 0 times by three elements.

25:57  
So this now contains.

25:59  
You can see three elements which are all zero.

26:02  
And now the first time we go around, you can see we're going to assign the value of 1 into here.

26:09  
So this now contains A1, the zeroth position.

26:14  
And then the next time we go around, we are going to hop in here, we're going to add the previous value.

26:21  
And so this output list now contains one and three because 2 + 1 is 3 and so on and so forth.

26:29  
Anyway, OK, let's have a look at the next one.

26:36  
So a little bit more complicated, again, different idea.

26:41  
So in this case I've made a little function, I've called it triangle side.

26:46  
The idea here is I'm just doing a bit of Pythagoras.

26:49  
So the idea with Pythagoras, if you can't remember your maths, is that the you've got a right angle triangle and the longest side you refer to as the hypotenuse and the square of the two other sides added together is equal to the square of the hypotenuse.

27:04  
That's just Pythagoras theorem.

27:05  
So the idea here is I've got a triangle side function and I can give it AB or H where H is the hypotenuse.

27:15  
Now I'm actually using variable assignment here, which is a slightly different concept.

27:21  
So in Python we can say put this value into this variable.

27:26  
Now the nice thing with this is if we we're explicitly saying we're not using the position, we're explicitly saying put it into this.

27:33  
I can miss, I can miss out assignments.

27:36  
So you can see I'm not assigning a value to B.

27:39  
And for this function, the defaults for all of the three variables are none.

27:45  
So if I don't give it anything, it'll be None.

27:48  
So by doing this, B is going to be None.

27:51  
So I'm telling the function I have no idea what B is.

27:55  
And then you'll see inside the function, depending on which one is none, it will calculate the missing 1 and return it that.

28:01  
That's idea anyway, So let's have a look what we're doing.

28:06  
So in this function you can see triangle side the default value is None.

28:12  
That means if I don't give it, it will be None.

28:15  
And then the logic is if a is not None, meaning somebody gave it a value and B is not None and H is none, then we're going to go ahead and calculate the hypotenuse, all right?

28:29  
So it's just a ^2 + b ^2 square root of that and return the value.

28:34  
Now I'm again, I'm paranoid, so I'm thinking to myself, right, if somebody's really silly and passes in three values, the function's got no idea what it's going to do with itself, right?

28:47  
Shouldn't be giving you the hypotenuse or one of the other lengths.

28:51  
So here I have.

28:53  
I've thought about that and I've decided I'm just going to tell the user basically one of the values input values must be None, and then in this case I'm going to return None.

29:03  
The other problem I potentially have, it depends on how square root is implemented.

29:08  
Square root by default, if you give it -1 or it will return, sorry.

29:15  
If you give it zero, it'll, it won't like that.

29:18  
It's minus Infinity.

29:19  
Yeah, zero is minus Infinity.

29:21  
Anything negative, will it be an imaginary number?

29:23  
So it won't like that anyway.

29:25  
So I've decided I'm going to wrap with the square root function and do something.

29:29  
So if the, if the value is zero, I I'm going to return 0 instead of minus Infinity because that's kind of nicer.

29:39  
So I've got here safe square root and up here what happens with safe square root is that again we pass in the value and if the square root is less than 0, I'm going to throw a value error.

29:53  
So I'm going to say no, can't have that length is negative.

29:56  
That doesn't make any sense.

29:59  
Or if the value is 0, so I'm just going to return zero.

30:02  
I'm not going to return minus Infinity.

30:04  
So I've decided what I want square root to do for these edge cases.

30:08  
So sometimes when you're writing functions, you need to think what would the user do?

30:12  
What kind of stupid thing will they do?

30:14  
All right, I'm going to catch that.

30:15  
I know what I'm going to do with it.

30:17  
So here you can see I've got 2 functions, this one here and this one here.

30:23  
And the first one is calling the second one.

30:26  
Does anybody have any questions about this?

30:29  
Does it seem reasonable?

30:31  
Yes.

30:36  
Yes, at line 13.

30:39  
So yeah, I'm not.

30:41  
I haven't implemented square root itself.

30:42  
I'm just calling here math dot scroot square root.

30:47  
Basically, give it a value and off we go.

30:52  
So sometimes with computer programming libraries, this thing will return 0 if you give an input 0.

30:59  
Now mathematically that's incorrect and it's a choice.

31:02  
So I've just decided I'm going to make the choice explicit and wrapper it like that.

31:08  
OK, so you may, you may end up with cases where you want to wrap the things.

31:13  
So for example, if you're dealing with a function tan, it goes to Infinity in some cases.

31:18  
So you might, I don't really want Infinity.

31:20  
I'll tell you what, I'll just return 0 or some throw a value error.

31:23  
Or you know, you've got to think to yourself should it do this or not?

31:30  
It can return Infinity.

31:31  
There is a special number if.

31:35  
But then if you try to do something with INF later on, you'll get a value error probably or problem will crash.

31:41  
Divide by zero error can also be raised.

31:45  
Anybody else?

31:49  
No.

31:49  
All right, fine.

31:52  
So positional is quite useful, but also explicit assignment is also useful.

31:59  
So here I'm missing out B.

32:00  
So I'm allowed to do that because I'm explicitly writing it.

32:04  
If I had this, so if I said put 3 like that.

32:11  
Now what's happening here is it's using the position.

32:14  
So a is going to be 3B is going to be 5, and H is undefined.

32:21  
So I didn't want to do that.

32:22  
I actually wanted to assign something to the third variable, which I can do by using the equals.

32:28  
All right, So that just highlighting this is a good thing sometimes.

32:32  
All right, so now we're going to go on to something a little bit more weird, which is recursion.

32:41  
Now, what is recursion?

32:42  
Recursion is when something calls itself.

32:45  
It's a recursive loop.

32:47  
You might have experimented as a child with a mirror, and you hold a mirror in front of your face with a mirror behind you, and you can see an infinite number of reflections, right?

32:57  
This is recursion if you're just referring back to the same mirror all the time that you can do with a computer programme.

33:05  
Depends on the language.

33:06  
Some languages like old style Fortran don't allow this, but you can get away with it by creating another function which calls the original 1 so you can have some kind of recursion, which isn't really recursion, but it works.

33:18  
Why do you ever want this?

33:20  
You want this, typically speaking, when you're navigating a hierarchy, meaning you start off from some point and you just want to keep doing the same thing to different nodes in the hierarchy.

33:32  
For example, if you're dealing with a family tree, you might want to ask, OK, how old is this person?

33:39  
What's their name?

33:40  
Oh, who's their, who's their dad?

33:43  
All right, fine, let's go to their dad.

33:45  
How old is this person?

33:46  
What's their name?

33:47  
What's their dad?

33:48  
And you keep going up and up and up the stack.

33:50  
The function does the same at each of the levels, but it it essentially hops up to the next one by passing the parent to the function again.

34:03  
All right, So what is the issue here?

34:06  
Rather like my mirror illustration, if we don't have some kind of safety net, we can have infinite recursion and therefore we keep declaring more and more variables and we run out of memory.

34:16  
So all languages have some sort of recursion limit that I'm aware of.

34:21  
Otherwise we would just, you know, run out of memory.

34:25  
Now, instead of hitting that hard limit, you want to have some kind of logic that prevents you from hitting the hard, hard limit if you can.

34:33  
Great.

34:34  
So that's recursion.

34:35  
So we're going to have a very quick look at some recursion and then I'm going to stop for extra questions.

34:41  
So here we have a very simple recursive idea.

34:46  
We've got a counter and we are going to make the counter smaller and we're going to keep asking to make the counter smaller until the counter has gone to 0.

34:57  
So decrement counter is our function.

34:59  
We've given it 5 and the value 5 is passed in here.

35:04  
We print out the value 5.

35:06  
We take one away from the value of counter, and then we say if the counter value is greater than 0, I'm going to call decrement counter.

35:14  
So I'm calling itself.

35:16  
Now this line 9 is wire recursion stops.

35:20  
If I didn't have line 9:00, we'd hit this safety net of the Python programming language, which would say something like recursion depth exceeded.

35:30  
OK, oops.

35:32  
So that that's all that's happening, it's calling itself.

35:35  
So let's see this in action.

35:37  
So we're going to go ahead and run a debug.

35:40  
Nothing mysterious at all, just a bit odd.

35:45  
And I, like I say, it is useful for navigating hierarchies, which is why I'm mentioning it.

35:52  
It will hopefully get to the right point.

35:54  
And then we will hop into the function.

35:59  
Great, let's hop into the function.

36:05  
Here we are at the function.

36:06  
Great.

36:07  
So we're going to hop in.

36:08  
So we're inside the function and you can see, yes, counter contains 5.

36:13  
So we're going to print it out and we're going to reduce the counter by 1.

36:16  
So now you can see counter contains 4.

36:19  
Ah, great.

36:20  
So that's greater than 0.

36:21  
Yep.

36:22  
So we're going to call the function again.

36:24  
Call itself.

36:25  
Here we go.

36:25  
We're back in the function.

36:27  
This, remember, is now the sort of second instance of the function.

36:31  
So in the computer's memory, the first instance hasn't finished yet, so we're calling it again.

36:35  
So there's a second instance running and we're going to hop in again.

36:40  
Yeah.

36:41  
All right.

36:41  
Oh, yeah.

36:42  
So we're going to hop in.

36:43  
So now effectively got.

36:44  
We did have before it finished.

36:46  
We had these different levels that are all running now because I pressed this button.

36:52  
I've basically finished it that I've exited.

36:56  
Anyway, you can play and see the layers running.

37:02  
Now we go back to our other example.

37:06  
This is a little bit more complicated.

37:08  
This is sort of illustrating a bit of the hierarchy idea.

37:11  
I didn't want to create a hierarchy per SE, because it's a bit more complicated.

37:15  
It's the sort of thing you might have with objects, which we haven't got to in the module.

37:18  
But anyway.

37:19  
So here we have a list and it contains a list.

37:23  
And So what I want to do is I want to unpack this list.

37:27  
So I want to go from the outside to the inside in a systematic way.

37:33  
So what's going to happen in our little function here is we're using is instance.

37:38  
So you can see we've got a list that contains a list which contains integers.

37:42  
Great.

37:43  
So what's going to happen here is the print list is giving this variable list and then we are taking values from it.

37:51  
And we're saying if it is a list, then I'm going to call the same function again.

37:56  
I'm going to call print list.

37:59  
If it isn't a list, I'm going to stop printing list and I'm just going to print the value.

38:04  
So what happens is effectively we're navigating inside the list until we reach an integer, print that value out, and then we come back up to the next instance of the function and we ask about the next list and so on and so forth.

38:16  
So if I run this thing here, it prints out 2345.

38:22  
Now let's have a look at why it does that.

38:27  
Inside we've got 2345.

38:30  
So what it's doing is it calls print list.

38:33  
Print list says, Oh yeah, it's a list.

38:35  
Great.

38:35  
So I'm going to, I'm going to try to print this and then it finds this variable is not a list, so it prints it out.

38:43  
This variable is not a list.

38:44  
Then we go back to the other function call and it evaluates the other list.

38:48  
Let's go ahead and do it.

38:49  
So I'm illustrate what I'm on about.

38:53  
We'll do this and then we'll stop for any questions.

39:12  
Right, so here we are.

39:20  
We've got a list that contains lists, and we're gonna hop into the function and you can see that our list that we've received is a list of lists.

39:29  
This is the first time the function's been called.

39:32  
Now we're hopping forward and we've taken the first list.

39:36  
So you can see that now we're dealing with this first list that's inside the lists.

39:42  
And yeah, we're asked the question, is it a list?

39:44  
Oh, yes, it is a list.

39:46  
So now we're going to call the same function.

39:48  
This is the recursion.

39:50  
So hop in and you can see we're now inside here and this.

39:55  
So we've got one instance that's already running.

39:57  
We've now got another instance that's received the inner list.

40:00  
And this time we're asking for a value.

40:05  
And this time it's not a list, it's two.

40:09  
You hop over.

40:10  
And we print it out here and hop over.

40:14  
It's not a list.

40:16  
We print it out, we'll hop over and we've run out of values and we leave.

40:22  
And we're now at the other function call the top level 1.

40:28  
So this time we're going to put 4-5 into the other variable.

40:33  
Yeah.

40:34  
And we've finished because I pressed the other button anyway.

40:38  
Does anybody have any questions or comments about recursion?

40:43  
Yes, yeah.

40:54  
So yeah, effectively, as I said earlier, the reason why you have a limit on recursion is because it's essentially holding a version of the function that's running in memory.

41:04  
And it's like running another one in memory and another one.

41:07  
And each time you're running a function, it's it's creating a little bit of memory.

41:11  
And there are variables in the function.

41:13  
So you are essentially creating a bunch of bits of allocated memory.

41:17  
And if you keep doing this, yeah, it becomes bigger and bigger in terms of the programme.

41:24  
And when you finish the function, it also has to return back to where it came from.

41:28  
So it's OK, I'm returning to this function now to this, to this, to this.

41:32  
But like I said, it is very useful for when you're treating nodes in a hierarchy in the same way.

41:40  
So in a physics problem, I would say have modelled objects.

41:46  
And I want to ask the speed of objects and they're in a system and they're all connected to each other, like with springs or something.

41:52  
So then I have a reference to the other thing, but I want to treat it in the same way.

41:56  
And that's when recursion helps.

42:00  
So it's got a particular use case.

42:01  
Don't use it too much because it's harder to debug potentially.

42:06  
You know, you have to realise, yeah, it's recursion.

42:09  
And don't hide recursion.

42:10  
Like I said earlier, you can have a function that calls a function which calls the original function, right?

42:16  
Which is sneaky.

42:17  
It's better just to be honest.

42:20  
Anybody else?

42:26  
No.

42:27  
All right.

42:29  
Well, we're a little bit early.

42:30  
So does anybody have any questions or comments in general about functions or anything else to do with the module so far?

42:41  
Yeah, yes.

42:43  
So I'm just thinking that the exercise for this week is going to make from what may be a better way to say we do that, We do that an exam where you answer in a kind of more cumbersome way.

43:00  
Yeah.

43:02  
So if you, let's go back to the earlier example, right?

43:04  
So if it's clear that what you're doing is like basically extra stuff for no reason, like if we put an else in or, you know, we've got an if else and we've got lots of nesting and like, why are we doing this?

43:17  
So we've got a loop where we don't need a loop or, you know, it's clearly superfluous code, then yeah, you might lose a mark or something and I might give you like, oh, OK, you really don't need this.

43:27  
I mean, the style of solution that I'm after is essentially similar to the exercises or the solutions I'm giving you.

43:35  
So if you give me back what I'm giving you and you've thought about it, fine, we.

43:41  
OK, so you've probably already seen the let's go to the where is it called?

43:48  
You've probably already seen the exam paper.

43:50  
Has everybody found the exam paper yet?

43:53  
Yes.

43:54  
No, some people have.

43:55  
So there's last year's exam paper up here.

43:58  
Now, I wouldn't scare yourself too much because it's what people can do at the end when they've, you know, spent a few more weeks doing object oriented programming.

44:09  
So there are ideas in here which we haven't covered yet.

44:13  
But nevertheless, the structure of the exam is quite similar to this previous exam.

44:19  
I tell you fairly clearly what to do.

44:24  
Like I'll ask you, please make a something dot PY file.

44:28  
And then we need in this case a class, but whatever.

44:31  
So I'm saying make this in this file and we need to give it some input parameters.

44:37  
So these are arguments or things we're passing into the constructor.

44:42  
If there's any kind of language that we haven't really discussed in depth.

44:47  
I've in this case, I gave the students this because, all right, it wasn't a later example, but I just decided I'm just going to give them that piece because we didn't spend loads of time thinking about it.

44:59  
Other than that, everything you'll find is in either the Python programme examples or in the lecture notes or, you know, it's some base of code that I've already given you.

45:09  
So it's more like a, a Lego building experiment, right?

45:13  
You're going to look in your bits box and go, Oh yeah, I'll use that bit, stick it together.

45:18  
That will solve my problem if I go further down here.

45:24  
Yeah.

45:25  
So this is another example where I didn't really spend loads of time doing this, so I gave the people the piece of code.

45:33  
So here you can see I say I'd like a function, I'd like you to call it find by modified, and it's going to have two input arguments.

45:41  
And then I tell you what it needs to do.

45:43  
It needs to return 01.

45:45  
So each one of these functions is relatively simple inside.

45:50  
So the chance of you sort of badly screwing it up is not that big.

45:55  
But yeah, if you're if you're using a loop where you don't need a loop or doing something really weird, then yeah, you potentially you might lose a mark or two, but none of the problems are particularly hard.

46:05  
So if you have been practising, you'll probably find the exam relatively easy.

46:11  
Anybody else?

46:14  
Yeah, it's in the last example that you showed there.

46:17  
Is there any particular this one for the sorry, the example code, the example code, Yeah, sure.

46:24  
This is there any signal reason why you would use is this is instance over the type function in this case or any basically interchange.

46:33  
So the is instance is nice because you you're missing out a step effectively.

46:38  
So here what I'm doing is I'm passing something in and Python being a duck type language, we don't know what type value is.

46:47  
So I can then give it the type and it will just return true or false.

46:51  
I could use type.

46:53  
So if I wanted to use type, what I'd have to do is I'd have to say something like value.

46:59  
Well it depends on the implementation.

47:00  
I think it's value dot type.

47:02  
And then I'd have to have an equals.

47:07  
So I've got like 2 steps.

47:08  
So is instance is just a little bit more, well, a little bit more simple.

47:14  
In some cases you will check the type maybe and you might say give me the type here.

47:20  
You can actually use the type as a variable itself.

47:23  
So you can say type of this thing and then I'm going to give that to a function.

47:28  
And then the function is going to receive that type and it's going to do something different on that.

47:33  
You know, if it's an int, it will do one thing.

47:35  
If it's a float, it will do another.

47:37  
So type is sometimes useful, but it's a lot more common that these instances actually what you want because you're just saying is this type that great, do something.

47:45  
Do do you see what I mean?

47:47  
So I use that basically for transparency and it's simpler.

47:53  
Sometimes they do use type, but it's for a different reason.

47:57  
Anybody else?

48:03  
All right, well, I'm going to hang around for a few minutes in case anybody is shy wants to come and talk to me.

48:08  
Otherwise, I look forward to seeing you in the lab tomorrow.

48:11  
And keep trying, keep playing with the language.

48:14  
It's all about how confident you feel.

48:17  
None of this stuff is wizardry or magic or it's just simple logic.

48:22  
A computer runs and the more you practise, the more you become confident you know what it's doing.

48:28  
And then when I give you a problem or to do, it'll be easy.

48:32  
You'll come to the exam and go ah, yeah, that's fine.

48:34  
Seen that before you gave us that in that lab.

48:36  
We'll just change it a bit and we're fine.

48:39  
Yes.

48:43  
I'm just thinking in my head that there could be something wrong where I like that exact situation where I've always been through that before.

48:50  
Yeah, from the examples and the solutions that we've been through, yes, I'm, I'm being even kinder than that.

48:59  
Everything will be on my place, but the home directory where you are going to take the exam will have all of those files pre downloaded in a file structure.

49:09  
So they're really easy to find.

49:10  
So if you, because I didn't want you to spend any time downloading anything, you can, of course, do download stuff if you want.

49:21  
So if you go to, let's see down here somewhere, I've got the exam.

49:28  
Maybe I've hidden it.

49:29  
I can't remember.

49:33  
Programming exam, You go down once more.

49:42  
Yeah.

49:42  
All right.

49:43  
Have I hidden it?

49:44  
I've hidden it, yes.

49:47  
OK, so it's currently hidden.

49:49  
I will unhide this at some point when I know the exam date.

49:53  
So that's the reset.

49:54  
That is the programming exam, which I currently don't know the date, which is why it's hidden.

50:00  
But what you see in here is you've got the links to the reference material if you want to download them.

50:07  
However, you have a zip file called resources dot zip.

50:14  
So before the exam, you can have a look at the zip file and see what's inside it.

50:19  
So you remember everything is what's in that zip file will be on the home directory of the PC.

50:24  
So if we open this thing, what have we got inside?

50:33  
Inside here we've got a folder called resources.

50:36  
And inside here we have a folder called labs, one called lectures, one called practise exam, Python programme examples.

50:44  
You can look inside the labs folder and you have all the examples from all the different labs.

50:50  
So yeah, so these are different things, classes, variables, whatever else with the solutions.

50:58  
And then the lectures, we've got all the lectures here, the PDF's and all the code, so it should be relatively easy to find stuff.

51:09  
Like I say, that zip file will be made available a few weeks before the exam so you can download it, familiarise yourself where everything is and it will be pre loaded on the machine.

51:17  
So you better get out.

51:19  
I had a question about files in the exam which I did post the answer to at the top here.

51:26  
So if you have questions about assessments, just ask right?

51:28  
And the best way to ask them if you're shy, you can send me them by e-mail, but other than that just stick it on the forum.

51:36  
So here we've got a question.

51:38  
Somebody said bringing notes to the exam.

51:41  
And my answer is, yeah, you can bring your notes as in your paper notes.

51:45  
You can bring your textbook.

51:47  
You can bring anything apart from your laptop or your mobile phone, basically.

51:52  
Why?

51:52  
Because, Yeah.

51:53  
All right.

51:53  
There's a very small chance somebody has something naughty.

51:55  
And Jeremy, you AI on here?

51:58  
Yeah, I'll just get that to solve my problem.

52:00  
So that that's why I'm not allowing the personal laptops.

52:04  
You can preload files if you want into my place.

52:07  
So if you don't know how to do this, you go back to my place and then you see where you've logged in.

52:14  
You see this little Chevron you can click on there and you can say private files, and then you can put together a folder of stuff, right?

52:23  
Other stuff you got from somewhere else.

52:25  
Don't care what, as long as it's not generative to AI.

52:29  
You can upload it there and then you can download it during the beginning of the exam if you want.

52:34  
All right, So you've got reasonable amount of stuff you can pull in if you wish to.

52:40  
OK, great.

52:44  
All right.

52:45  
Well, you got a question if you were like, practising on your own.

52:55  
Yeah, Yeah, you can put those in there.

53:02  
That's fine.

53:03  
I don't mind.

53:04  
I mean, the constraint around the lab is just because of generative AI.

53:08  
So.

53:08  
Yeah.

53:09  
All right then.

53:11  
Fine.

53:11  
So I think I'll stop there because we've run over slightly and I'll see you in the lab.