**w06-01-ObjectOrientedProgramming-2024**

0:01  
OK, so today we are introducing another new concept that is objects.

0:08  
So today's lecture is object oriented programming.

0:11  
We are going to talk about it today and next week.

0:15  
And then what we do for the remaining weeks is we'll introduce some functions to read and write data to files.

0:22  
And we'll just keep playing with objects and classes to become a bit more used to it.

0:28  
OK, so today's lecture, let's make sure we are good, right?

0:36  
So we start off with a simple lecture, sorry, simple class.

0:40  
In this case, we have at the top here our class definition.

0:45  
Now in Python, you'll remember that our function definition is just def and the name.

0:51  
So it could be def my fun, def multiply something.

0:55  
The syntax here for define a class is class and then the name.

1:00  
Now in the PEP 8 coding style similar to other languages, the class name should be in Pascal case meaning that you have a capital letter at the beginning and then if you have two words stuck together you have another capital letter.

1:15  
Camel case is where the 1st letter is lower case.

1:17  
So for class definitions we use Pascal case like this.

1:23  
Now we have a strange thing in Python which is not the same as other languages.

1:28  
So the way that you will see a constructor is written in I think every other language I can think of is not the same as Python.

1:38  
So a constructor is a function that is called when you want an instance of a class.

1:44  
A class you can think of as a bit like a type.

1:47  
Maybe you can think of an integer, you create an integer variables.

1:51  
In this case, we have a class and we're going to create an object which is an instance of that type.

1:59  
Now what we have here is a function called\_\_init\_\_Now for whatever reason, the people who thought about the Python programming language decided that that was the name for the constructor.

2:12  
So if you're going to create an object of Indiana, this case my class type, what happens is this function init is called.

2:20  
The other strange thing about Python with respect to every other object oriented language I can think of is this variable self.

2:28  
Now if you have something that belongs to an object, so it's a function that belongs to a specific object, it has to have an input parameter here called self.

2:39  
And this thing self is used to say it belongs to the object.

2:44  
So you can say here we've got self dot name.

2:48  
So here name is a variable that belongs to an object.

2:52  
It is a data member.

2:55  
Now, what we've done in this little function is we've passed in a value for this thing name, and we've assigned the value to a data member name, which is a member of this object.

3:07  
OK, so there's a copying of data between these two.

3:11  
They're not the same variable.

3:13  
This one is a local variable, just like any function that you encounter.

3:18  
And this one belongs to the object.

3:20  
That's why it's got self.

3:22  
You can think of myself this object in other languages.

3:26  
Sometimes you see the word this, it's a, this pointer to this particular object.

3:31  
For whatever reason, Python uses self.

3:33  
Don't ask me why.

3:34  
You have to ask the inventors of the language.

3:37  
Anyway, here we go.

3:38  
We've got the class name, you can see, and then we've passed it a text string.

3:43  
And each time we do this, we're creating an object.

3:46  
So here M2 is an object and M1 is an object.

3:50  
And this will actually call the init constructor.

3:55  
Because you can see we've used the class name as if it's a function.

3:59  
So if you use the class name as if it's a function, you're calling the constructor.

4:03  
Now again, another strange thing about Python is that we can only define one constructor.

4:11  
In other languages, we can have several constructors where we have different input parameters.

4:17  
This is true of Python, sorry, not Python.

4:20  
It's true of Java, it's true of C, it's true of C You can do this in several other languages, but not Python.

4:27  
If you want to have different inputs, then what you can do with your constructor is you can assign default values.

4:34  
If you remember the last series of examples of functions, I had a function there where I assigned a default value.

4:43  
This was the triangle.

4:45  
Well, it was the triangle function that was going to return a value depending on the inputs.

4:50  
So we essentially end up with a constructor where if you don't want to maybe sometimes pass in that variable, you set it as a default value.

5:01  
OK, anyhow, so we create 2 objects, 1-2.

5:06  
Now once you've got an object, we can then work with the data member.

5:10  
So here you can see we've got M1 dot name here, the name that's a data member, it's this one here, the data member that belongs to.

5:21  
Can you see the M1 object?

5:23  
So we've changed the data member value in this object here, and we can, yeah, we can update them with the assignment operator.

5:32  
And we can also just go ahead and use them so we can say the object dot the name.

5:38  
Now this works because this particular data member is a public one, and we'll come to public and private next time, so bear with me.

5:48  
By default, in Python, data members and member functions are public.

5:52  
In other languages, by default, they're private.

5:54  
So don't make that mistake.

5:57  
Anyway, you can see the output of the code down here.

6:00  
We're just printing out the values.

6:02  
So the concept of a class is useful potentially when you want to put several things together.

6:10  
So you might want to put several bits of data together and call it something or several functions that go with those data together.

6:18  
And that's basically the idea.

6:20  
You're trying to group things into a class, right?

6:27  
So now I'm just going to discuss a little bit of how this works inside the computer to try to demystify it.

6:34  
There's nothing complicated here.

6:37  
It is behaving in a similar way to a normal variable.

6:42  
So when we create an object, and in this case my constructor doesn't require any inputs, I've called the class as if it's a function.

6:50  
So I created an object and then I can go ahead and access the data member.

6:55  
Now obviously here I've if I've not initialised it, I'm relying on whatever happens inside the constructor to set the value of X, but whatever.

7:03  
So I can go ahead and use this data member.

7:07  
Now, what's actually happening inside the computer is that for an object, you have a block of memory which is big enough to hold the whole object.

7:15  
And then the dot and the variable name is essentially acting a bit like an offset.

7:22  
If you think of a list with a list, we had zero as being the 1st element of the list and one as being the second element.

7:29  
It's an offset.

7:31  
Whereas here we've got the object which is telling the computer, all right, we're here in the memory.

7:36  
And then the data members or member functions are just an offset with respect to that.

7:41  
So the two things together, the object and its data member or member function, refer to somewhere in memory.

7:50  
So that's the general idea.

7:52  
It's a block of stuff that you have in your computer's memory.

7:56  
OK, now a little bit on functions, and I'll take any questions.

8:02  
If anybody's got any questions about this stuff so far, we can have a constructor.

8:07  
We've already covered that.

8:08  
So in this case, this constructor doesn't have any inputs.

8:11  
It has to have self here because it belongs to an object, but it doesn't have any other input parameters.

8:17  
So we could just call it like that.

8:19  
My class with no inputs.

8:21  
Now you can see in this constructor we've assigned a default value to the data member name.

8:26  
We've given it this string my class, but we've also defined another function.

8:31  
This other function is called full name.

8:34  
Now any function that's sort of an extra 1 you added in.

8:38  
You can give it whatever name you like, as long as it doesn't conflict with any of these standard names.

8:43  
We'll mention a few more in the next lecture, but not today.

8:47  
So anyway, here we have a function called full name.

8:51  
If it's to be a member function of the class, it has to have self here because it belongs to an object.

8:57  
And then again you can do whatever you like inside.

8:59  
Here I've only got one line, but any just like any other function you could have many lines and also you could have a comma and several other input parameters and pass the other things in and do stuff with the things that are inside.

9:12  
Now you'll notice immediately that there is no input to full name here of self dot name.

9:20  
It's not explicitly passed in.

9:21  
It's in fact inside this thing's self.

9:24  
So one of the benefits of a class, if you like, is that when we've created the data values or the we've initialised the data members inside that class inside another function that belongs to the object.

9:38  
We can then just go ahead and change or use those data members.

9:41  
So you can put the data into the class instance, the object, and then you can have these functions operate on the data.

9:48  
An example might be that your your class here's some kind of configuration data, right, Which is read from a file.

9:56  
And then you say, oh, right, I want these values at this configuration data.

10:00  
And so you have member functions that are in the class, which then are going to return particular data which is processing somehow of what was loaded into the class, right?

10:11  
So what?

10:12  
The object instance got the data, then you call the functions.

10:15  
Anyway, that's the general idea.

10:17  
It makes more sense with classes.

10:19  
If it's a thing, like something you can imagine as an entity.

10:24  
If it's just a nebulous concept, it doesn't really help.

10:27  
But you're trying to put functions and data together.

10:31  
OK, so here you can see we can use that function in the same way we use the data member.

10:36  
We can say the object dot the function, and then that's referring to this function here.

10:41  
And lo and behold, yeah, it works just as normal.

10:45  
OK.

10:46  
Does anybody have any questions or comments before we dive in and look at the code?

10:59  
Yes, good question.

11:00  
So the word attribute is a word that's associated with the UML class diagram design.

11:07  
So it will come back to UML next week.

11:10  
And I think that we picked up in Java as well.

11:13  
So attribute is a data member, right?

11:19  
And the other one, which I've forgotten already, is the member function.

11:23  
So anyway, when we look at UML class diagrams, which we all do next week, we'll have an explicit, here's the UML, here's the class attribute that we sort of, you can see it right one next to the other.

11:34  
If you want to read ahead, you can look at the Python programming examples and look for attributes with the UML discussion.

11:44  
UML is a unified model modelling language, so it's a way of graphically describing what you're going to build.

11:53  
Now there are many UML diagram standards, so one of the ones we use next week and what will be used in the Java is UML class diagrams, where we're just drawing little boxes to say what's inside of our class.

12:09  
Imagine we've written no code and we just want to say it's got these data members, it's got these member functions.

12:16  
So we put that into a little box diagram.

12:19  
You'll see it next week and then everybody knows what we're talking about.

12:23  
And that could be implemented in Python, C#, Java, it doesn't matter.

12:28  
It's a standard way of saying the class data members, member functions, right?

12:34  
UML is wider than that in that there are several other diagram standards which we'll come back to next semester, which we use when we're designing software.

12:43  
So we're trying to express how does a programme behave before we've actually built it.

12:48  
OK, so it's unified modelling language, of which under that umbrella you have several diagram standards.

12:54  
Anybody else?

12:58  
All right, let's have a look at the code.

13:00  
It's always good to look at code because there's nothing complicated here.

13:03  
It's just another way of doing something.

13:06  
So let's just go in and see what's going on.

13:09  
All right, so we're going to debug.

13:11  
So we're going to start here and I'm going to just quickly run and debug this.

13:16  
The purpose of me doing this is to show that the constructor is called.

13:20  
As I have said, nothing strange is happening.

13:24  
Your request appears in the function.

13:27  
Now you can do whatever you like in the constructor.

13:30  
Generally speaking, the constructor should normally be used for just initialising variables.

13:37  
The reason why you do that is because if something crashes inside the constructor, you don't end up with an object.

13:43  
So you sort of like what's gone wrong.

13:46  
It's better that you just initialise stuff in a constructor, and then if you've got anything that's going to do anything that might crash, put it outside the constructor anyway.

13:55  
So here we go.

13:56  
So we've arrived at this line 12.

14:00  
Now we're going to step into the function.

14:02  
Can you see what happens?

14:03  
We're in the constructor, and you can see here that we have passed in the new object and it's going to be assigned here to this thing.

14:13  
And now we've got this object, and you can see that the Visual Studio Code tells us it's got a memory address, right?

14:20  
It's this lovely hexadecimal string that's the block of memory that's occupying.

14:25  
And it says inside it's got a data member called name, which has this value.

14:31  
Now we can do that again.

14:32  
So I'm just going to hop over.

14:34  
So I've now got another object, object called M2, which has also got a data member called name, which is also contains a new object.

14:40  
Wonderful.

14:42  
Now I'm gonna go ahead and update that one.

14:44  
So now you can see that the M1 object now contains updated name.

14:49  
All right, And we can go ahead and print them.

14:51  
Yeah, great.

14:53  
And that's it.

14:55  
Let's go and have a look at the other one.

14:59  
So in this case, we are going to just run debug to there.

15:03  
Never run debug.

15:12  
And all we're doing this time is we're just checking.

15:15  
Yeah, the function works as we expect.

15:18  
Nothing fancies going on here.

15:25  
Get rid of all the gobbledygook.

15:32  
All right, Here we go.

15:33  
So we've now got an object called M You can see that if I scroll this down, M contains a thing called name.

15:40  
That's its data member, which has got my class in it.

15:42  
You can see that was initialised in the constructor here.

15:45  
Now we're going to go ahead and step forward 1.

15:48  
So we're now on this line.

15:49  
I'm going to step into the function so you can see what happens.

15:52  
So let's step in and yeah, we're here.

15:55  
We're in the function full name.

15:57  
So this was called here.

15:59  
Now you can see the variable name, the data member name that is contains new name.

16:05  
And then we can just use it as we would normally, just sticking 2 strings together.

16:09  
They're returned and it's just printed.

16:11  
That's it.

16:13  
All right.

16:15  
OK.

16:17  
You're either very tired or everything's great, or you're just sort of lost in the woods somewhere going.

16:24  
All right, Well, hopefully you tell me.

16:26  
If you get lost in the woods, you are lost in the woods.

16:30  
Yeah.

16:33  
So it, yeah, Practise, practise.

16:36  
Try it out.

16:38  
Nothing strange is happening.

16:39  
And when you see a new concept like this, it can become a like a difficult thing.

16:45  
What's going on?

16:48  
Run the code and we'll test stuff out in the lab and we can discuss again, right?

16:53  
So when you're building classes, you can have classes to do different things.

17:01  
Now what you normally do is you have classes which are mostly data.

17:07  
So imagine that you have a class that's going to store something to do with, I don't know, the diagnostic data of your car.

17:14  
It's going to record how the engine temperature is running or something like that.

17:18  
It's going to hold mostly data.

17:21  
You might have one or two member functions in there which are going to return so-called transient values.

17:27  
That means that you are just going to create them and return them.

17:31  
An example of this could be for example, if if you store map coordinates, then you've got a couple of positions, but you could you could return an angle.

17:42  
Now that angle would be derived from the coordinates using trigonometry.

17:46  
So you don't store the thing you're going to calculate because it's, you know, the original data are the coordinates.

17:53  
But that would be an example of transient data.

17:56  
So mostly data and some functions that are going to return the data in a slightly different format because that's useful to you for a particular reason.

18:05  
So calculating something simple.

18:08  
So these are data classes.

18:09  
And when you're describing a programme, they become what you call the data model.

18:16  
So if you're saying I've got this programme here, the data model classes.

18:19  
And when you come to writing stuff like web applications, you end up with a Python file called models, which lo and behold, contains the data model classes, which are mostly data.

18:31  
So you'll see this models data model classes, all right.

18:36  
The other type of class you have is an algorithm class.

18:39  
Now these classes are the other extreme.

18:42  
So the idea here is that we have basically a collection of functions that do useful things which all belong together.

18:50  
And they might have some kind of configuration data passed in.

18:53  
So that will change their behaviour.

18:56  
Perhaps.

18:57  
Now you try not to have classes that are a mixture because you end up passing data model classes around your programme.

19:07  
Think about the data flow goes into an object, well, goes into a function or whatever else.

19:11  
Whereas an algorithm class tends to operate on the data.

19:14  
So you try to do this, don't have a sort of, oh, this is kind of both.

19:18  
That's not a good idea because then you'll get lost about where the data are.

19:24  
Now, a warning in red, a class might not be needed.

19:29  
Sadly, some languages such as Java require you to have classes because the designers thought that was a good idea.

19:37  
It's not a good idea because what happens is sometimes you have a collection of functions that don't really belong to anything.

19:44  
You just have something like.

19:45  
I've got a nice little useful function that does something and then you create these strange classes called utils or helpers or something, which has, it's not an object at all that you're ever going to create.

19:56  
You're going to end up with a bunch of static functions that you call.

19:59  
So you'll have to do that in Java.

20:01  
Sorry, in Python, don't do it.

20:04  
If you don't need a class or an object, just don't, you know, go ahead and use a function or a bunch of functions in a module.

20:10  
That's a better design concept.

20:12  
So as you're programming.

20:15  
You need to ask yourself what's the best way of doing this?

20:17  
Is this going to be helpful?

20:18  
Is that going to be easy to understand for somebody else, or is it just going to confuse them?

20:24  
OK, now, so that's the two kinds of classes, data model classes and algorithm classes.

20:33  
Now inheritance.

20:34  
So to add one more complexity to the problem this morning before we look at some more code examples, inheritance is a concept which you can think of as you know, you inherit something from your parents, maybe your good looks or whatever it is.

20:50  
And so we have the same in a class essentially.

20:53  
So you have a class that can inherit from a parent class.

20:57  
Now, often we call the, it could be a parent or it could be a base class.

21:02  
That's the thing we inherit from.

21:04  
And the thing that's inheriting is called the derive class or something like that.

21:09  
OK, so classes inherit or can inherit from other classes.

21:15  
And the idea here is that the, the thing that's being derived, the inheritor is inheriting the data members and member functions.

21:25  
So it's, it can be useful when you don't want to write that member function or data member again.

21:31  
You just, you know, OK, I've got something that's a little bit more specialised than the base one, so I'm gonna inherit it from the base 1, the derived class, yet can directly use the member functions and data if they're public or protected.

21:46  
We'll come back to that next time.

21:48  
So if they are, as we've already looked at, they're public, then it can access them directly.

21:55  
Whereas if we use private, which again we'll come back to next time, it can't.

22:00  
Now a big warning, and this is true for any programming language.

22:05  
Only use inheritance if absolutely necessary.

22:08  
Why?

22:11  
Because if you change what you're inheriting from, imagine some other programmer comes in and goes.

22:17  
The base class is wrong.

22:19  
We need to change it, and they change it, and then you assume that something's in there and all sorts of other things are broken, and then you spend, I don't know, a week or something horrible.

22:30  
Rewriting all of the things that derive from the base class is very nasty.

22:34  
I've seen that happen in other programming languages like C++.

22:39  
People go, oh, this is great, let's have a really complicated inheritance structure, keep everything really nice and tidy.

22:45  
And then somebody else pops up and says, oh, we need to change the base class.

22:48  
Oops, goodbye.

22:50  
Lots of time.

22:52  
So don't have inheritance unless you really need it.

22:56  
In fact, you can do most things in Python without inheritance.

22:59  
In Java, you'll have to use inheritance, but use it sparingly because like I say, if somebody needs to change that base class later, it potentially costs you a lot of time.

23:10  
All right, here we go, An inheritance example, and then I'll talk about some other code examples and take whatever questions we've got left.

23:19  
So here we've got a class, it's called Map Position, and it has two data members, latitude and longitude.

23:26  
And we've given them zero to start with.

23:28  
So they're there, they've got nothing in them, and we have inherited from Map Position.

23:34  
This class here is called Incline Position.

23:36  
So can you see the difference here?

23:38  
We have a colon.

23:39  
Here we have these rounded brackets and a colon and we've inherited from that one.

23:44  
So we're saying incline position is essentially a specialisation of map position.

23:50  
It's a very simple example here, but you can see the map position is a latitude, longitude and incline position has got another data member called elevation.

23:59  
But because it's inherited from the other one, it basically has these two inside it.

24:04  
Apply that inheritance.

24:05  
We didn't have to write it again.

24:07  
We didn't have to have inclined map position dot latitude dot longitude because it receives them from this inheritance anyway.

24:14  
So here you can see in action, we create an object of map position type.

24:19  
We then go ahead and assign the data members.

24:22  
That's completely normal.

24:23  
That's what we're doing with the first class you saw today.

24:26  
And then in the second case, we instantiated the inclined position object and you can see, yeah, we can actually access the latitude and longitude which were defined in this base class that we inherited from.

24:40  
So we can access those and we can also access the elevation.

24:44  
So this is the general idea of inheritance, that you are inheriting the data members or the member functions from the base class.

24:53  
And you can keep doing this.

24:54  
You can inherit from another one, another one.

24:56  
You can build up an entire inheritance tree with the danger warning that if you have to change the base one, nasty things might happen in terms of your code.

25:05  
Anyway, that's inheritance.

25:08  
Does anybody have any questions about inheritance before I go and chat through some more code?

25:17  
No.

25:18  
OK, let's go and have a look at the.

25:22  
Yes, yes, right.

25:32  
Good question, yes.

25:34  
So super is used if you want to call something in the base class from the derive class.

25:40  
So if you say you're in here and you explicitly want to run a function that's in here, you can use super to do that.

25:48  
Now, unlike other languages, super will find the base class automatically.

25:56  
Yeah, yeah, you won't need unless you want to override something.

26:08  
Yeah.

26:08  
So super, I've only ever seen it used for member functions where you want to access the one in the base class.

26:15  
So you might see an explicit where you've said I'm going to call a function of the same name, because you can have the function of the same name.

26:22  
Here you can see there are two constructors, same name.

26:25  
So which one do I want to call?

26:26  
Do I want to call this one or that one?

26:28  
Right.

26:29  
And therefore you can call this one.

26:30  
And then inside it, you can use super to call that one explicitly.

26:34  
Yeah, here.

26:40  
OK, here.

26:41  
What's happening is that's been called by default anyway, so these are still 0.

26:46  
But yeah, if you want to do something more complicated, then go ahead and use super.

26:51  
In terms of the curriculum of this course, and for people who are deeply lost in the woods, I'm not going to do loads with inheritance.

26:57  
We essentially stop here and just say OK, this is inheritance.

27:02  
If you're key, you can go to the Python programme examples and read about polymorphism, which is another concept which I'm not going to talk about, and static data members, which again is another concept I'm not going to talk about.

27:14  
They are used to potentially later on, but I don't want to overcomplicate things.

27:17  
But yeah, super is used for calling things in the base class and you know the languages, you'd have to explicitly say I'm going to call this one, but super just finds the base class.

27:28  
All right, OK anybody else?

27:34  
Yes, go for it.

27:36  
Far away.

27:37  
We need questions and comments.

27:41  
Yes, same way.

27:49  
OK, so some.

27:52  
Yeah, yeah, yeah.

27:53  
So you can decide like here, you've got 2 as I was saying to the other student, you've got 2 functions called in it.

28:01  
So by default what will happen is if you say I want a function called, whatever name it is, it will use the one in the class that's derived.

28:12  
You won't use the base 1.

28:13  
So imagine you've got 2 functions, my funk and my funk, and one is in the class you inherited from, 1 is in the derived class you want to call the base Class 1, you use super.

28:23  
Otherwise it by default will use the derived class one.

28:27  
All right, so you read about this.

28:29  
This is sort of extra stuff that's not going to be assessed.

28:32  
Interesting.

28:33  
But you know I won't assess you on it.

28:36  
All right, fine, let's let's have a go at some code.

28:42  
So here we go.

28:43  
Here's inheritance and we'll go ahead and just show what's happening.

28:50  
Put some break points in here, and let's put a break point in here, and we'll run this thing debug.

28:58  
Actually, we probably don't need the other ones.

29:07  
Yeah, let's take that one.

29:10  
OK, so here we go.

29:12  
We're gonna create an object.

29:13  
What happens?

29:14  
Let's see.

29:15  
We go into the constructor for that class.

29:18  
There we go.

29:19  
We've got those two data members.

29:20  
They both have zero, and we can now see we've assigned 2 values.

29:25  
So now we've got this object called M and it contains two data members which have got the two values in.

29:31  
Now this time we're gonna step in and you can see we've gone into this constructor and we've set the elevation to be 0.

29:40  
And let's see, Yep, you can see it doesn't actually call that one, right?

29:45  
Cuz I didn't use super to do it.

29:47  
Now we're gonna have inclined position as an object.

29:51  
And if we look at P, you can see it's currently got those variables.

29:55  
I'm allowed to access the other ones, and there we go.

30:01  
Anyway, let's put a side inheritance and we'll think about when you will use classes.

30:12  
So as I said earlier, classes are normally used when you have a thing.

30:16  
It's something that's obviously an entity.

30:21  
All right, so I've got a very simple one here called cube.

30:24  
The idea is that we have a cube, and we want to have a few functions that might deal with the cube, and that returns some data.

30:32  
Now, of course, a cube has one data value associated with it, which is the length of a side.

30:38  
You might have a tiny cube like a die that sat on the table, or you might have an enormous cube, right?

30:44  
You're just varying that length.

30:45  
So here you can see my cube.

30:47  
I have one input value into the constructor length and then what's happening is that I can then can use this self length in the member function.

30:58  
So here I've got 1 member function which is just going to return the volume of the cube, which of course is just the length cubed length times the length times the length is the volume of it.

31:11  
We could also have another function here which does you know, calculates the area using the data member.

31:16  
Anyway, so here we go.

31:18  
We've got a cube and we'll go ahead and run this.

31:29  
And what will happen is, again, we'll go into the constructor, you'll see the value 2 being assigned to the data member length, and then here you go.

31:46  
It will run to the end here.

31:51  
Great.

31:51  
So we've now got a object.

31:54  
It's got inside the object a length of one, and we can call the function volume.

32:00  
And you can see here that length of two there.

32:03  
So I put in two.

32:04  
I've got a length of two, and two cubed is just gonna be 8.

32:09  
And so once we step over this, you can see that we've got a value of eight.

32:14  
So this is an example of where having an object might make sense, right?

32:19  
Cube.

32:19  
Everybody knows what a cube is.

32:21  
It's got a function called volume.

32:23  
We could have another function called surface area, which would return the surface area of the cube.

32:29  
And there you go.

32:31  
Anybody got any questions about that?

32:33  
Does that seem reasonable?

32:37  
Yeah.

32:37  
Now I noticed the standard of commenting again.

32:40  
Commenting is assessed at the end.

32:43  
For each class we should have a multi line comment like this with double quotes where we just say what it's for.

32:50  
What is it there for?

32:53  
We do the same with functions you can see here.

32:55  
We don't normally bother to say this is a constructor because everybody knows that's a constructor.

33:01  
You could have a function here that says what the functions for.

33:04  
I know volume is fairly self-evident, but I've put the the comment in anyway and there you go.

33:12  
Right, Let's have a look at another one.

33:16  
So this is a bit more complicated, but it's just showing you what you could do.

33:21  
In this example, we have a class called an Account.

33:26  
Now here the Account is storing a list of transactions.

33:31  
And the idea is that we are going to either spend or receive money.

33:35  
And so if we are spending it, the transaction is negative and if we're receiving it, it's a positive transaction.

33:41  
It's going to increase our account.

33:43  
Now, just like a normal data member here you can see we've got self dot transactions.

33:51  
It's a list, but it's just a data member list.

33:54  
Great.

33:55  
And then we're going to go down here.

33:58  
Let's go down, down, down.

34:00  
We're going to create an instance of the Account class.

34:04  
So normally, if you want just a generic object, you may have the same name as the class, but look, it's lower case.

34:14  
So this Pascal case class names the variables same as ever.

34:21  
They're in snake case.

34:22  
All right, this Python standard snake case for variable.

34:25  
So anyway, this thing account is an object.

34:29  
We can then use its list.

34:31  
So you can see this is just a list, OK, it belongs to the object, but it's a list, so we can just go ahead and append.

34:38  
So we're pending 1.

34:39  
So we're saying, yeah, we put 1 LB or whatever it is, $1.00 into the account.

34:44  
And here we're pending minus 0.5.

34:46  
We're saying we spent minus 0.5, whatever the currency is.

34:51  
And once we've done that, we can then call these member functions.

34:54  
So you can see that this is kind of logic here to the design.

34:58  
We've got a class that contains the transactions and then we've got other functions.

35:01  
So this one balance, it's going to return the total money we've got left.

35:07  
Expenditure here is going to return how much money we've spent and income is going to return how much money we've received.

35:14  
How do these things work?

35:16  
So if you look at the first one, income, well, what happens is that we're using this list transaction.

35:25  
So we're reliant on somebody having appended to the list outside of the object.

35:30  
So they've done that down here so you can see account.

35:33  
So they've appended the transactions here.

35:35  
So now if we go back to our function, sorry, go back to our function, you can see that we're looping over that data member the same way you'd loop over a list in any function.

35:47  
The only strange thing here is that you have got this self because it belongs to an object and you have to pass the self in there.

35:55  
But that's the only strange thing.

35:56  
So you can see that we've got a list just as normal.

35:59  
We're taking data values from it, the card dealing analogy, and we're saying if that value is greater than 0, then it's an income.

36:10  
And so we just going to count up the total and return it.

36:13  
So notice in this function you have a mixture of a data member.

36:19  
This is your data member, and these other things are just normal variables, OK, total value.

36:25  
They're just normal variables just like any other function.

36:29  
And you can have as many normal variables in there as you like.

36:32  
We're using the data member inside.

36:35  
Now.

36:36  
We could pass in some transactions here, but then you wouldn't need a class anymore.

36:40  
You could just have a standalone function which receives a list and then returns a value.

36:46  
The expenditure 1 is the same idea.

36:48  
So we've got a list and we're looping over it, and we're just appending to that total, and then we're returning the total and that's it.

37:00  
And then the balance, Well, I'm just being super lazy and I'm using the sum function.

37:04  
So I give it the list, and that just returns the sum of all the transactions.

37:08  
Because the expenditure is negative and the income is positive.

37:13  
Then that gives you the balance.

37:15  
Anyway, you can see again the idea we've got a class that's somewhat meaningful, right?

37:20  
An account.

37:21  
Yeah, Somebody knows what an account is.

37:24  
It's got data members that somewhat make sense.

37:27  
It has a transaction data member, which is a list of transactions.

37:31  
And then, yeah, we've got some functions which seem to belong to that class, which have reasonable names, and they do something that seems useful.

37:39  
And so we've used the concept of a class to try to bring things together into one thing rather than having separate functions and data that we pass around.

37:53  
So that's really it.

37:54  
The class idea is a way of gathering things together so that you can think of that as one conceptual entity rather than having loads of functions scattered around or data separate from them.

38:10  
OK, so that's all I have to say.

38:12  
So I'm ready for lots of questions, hopefully.

38:15  
Does anybody have any questions or comments about classes?

38:22  
Yes, I haven't yes.

38:30  
So here, well, I've got here a I've called the constructor, right.

38:35  
So what that will do is it will create an object and you can see inside the constructor, the constructor has made a list, right?

38:45  
And it's assigned it to this thing transactions.

38:48  
The reference is here.

38:49  
So if we go, let's just step forward, let's go to here, one of the first example that you showed, you just give the parameter name.

39:06  
Yeah, you could pass the list in into the constructor.

39:10  
That you could do.

39:11  
You have to be careful when you're passing in things that are mutable because you remember if it's mutable, we're not passing in a copy of it, we're passing in a reference to the original.

39:22  
So you can cause yourself a bit of confusion because let's say you pass a list into the object, right?

39:28  
You say, OK, constructor here have a list.

39:31  
You then say, oh, the everything's inside this object, right?

39:35  
So we pass around the object, but then some sneaky person uses the original reference to the list and changes it.

39:42  
And then that's changed what's inside the object.

39:44  
So there's a concept which we'll talk about next time, encapsulation, meaning we want the object to sort of own it's data.

39:53  
So if you are going to pass in a list here, you should consider into the constructor.

39:58  
That is, you should consider making a copy of it.

40:01  
So the object actually owns it.

40:04  
Otherwise somebody could do something nasty like take the original reference to the list and change it.

40:09  
And then somebody else is like, oh, why is my object changed?

40:12  
Anyway, here you can see that we've got a an account object and it has a data member list inside.

40:21  
And then we just use it as normal.

40:23  
So we append and you can see that what's happened is, yeah, the data member list now has got one.

40:28  
And we append again and the data member list, when it finally, as me, has now got the two transactions in.

40:35  
So it's behaving as if it's just a normal list.

40:38  
It's just referenced via the object name.

40:43  
Anybody else?

40:50  
All right, Nothing at all.

40:53  
It's all fine, right?

40:56  
Rabbit in headlights, Maybe have a go try out the code.

41:01  
Debug is really helpful.

41:03  
There's nothing strange that's going on.

41:05  
All that we've got is this prefix which is referring to the object, and then what's inside is working in the same way as we were working before.

41:16  
All right, so I'm going to stop there.

41:17  
If anybody is shy, you can come and ask me a question.

41:21  
I'm going to hang around for the next 10 minutes.

41:23  
Otherwise, I'll see you in the lab.