applications Danny Adams developed this course hi guys welcome to this object-oriented programming in Python course now if you struggle to get your head around objectoriented programming in the past then this course is for you as object-oriented programming can be pretty daunting for beginners as there are many Concepts principles and terminologies such as abstract classes polymorphism encapsulation abstraction and the list goes on but by the end of this course you'll fully understand everything that you need to know about these Concepts to go on to develop objectoriented software in Python and the great thing about everything that you'll learn in this course is that all of the concepts that you'll learn also apply to other programming languages that support object-oriented programming such as Java C JavaScript and PHP so it's going to be very easy for you to learn and understand other languages and Frameworks after completing this course so to take this course you should know the basics of python such as variable functions if statements and for Loops uh so you just need to know the very very Basics you definitely don't need to be any expert in Python and if you don't have much python experience but know these Concepts from other programming languages then that is also fine you should be able to take this course and also you don't need to know anything about objectoriented programming to take this course my goal for this course is to teach you as a complete beginner so what will you learn in this course so the first part of the course assumes that you're a complete object-oriented programming beginner and introduces the very fundamentals of objectoriented programming such as creating classes and objects attributes and methods get and set of methods and why they are important properties and why you should favor properties over GS and Setters in Python static attributes and methods public protected and private access modifiers and Abstract classes and as the course progresses we'll start to introduce important objectoriented programming Concepts such as encapsulation abstraction inheritance and polymorphism and this is where object-oriented programming and its advantages will start to make a lot more sense to you and once you've completed this beginner's course if you want to take your skills to an advanced level you can check out my objectoriented programming in Python course on udemy which covers more advanced concepts such as composition versus inheritance coupling The Fragile based class problem and dependency injection it also covers all five solid principles and six of the Gango 4 software design pattern giving you everything that you need to write elegant organized and maintainable objectoriented software and there's also a book version of this course available and I created this book as it can save you time for having to make lots of notes throughout the course and it makes it really easy for you to quickly revise these object-oriented programming Concepts principles and patents so if that sounds like it would save you some time and make it easier to revise then you can check out the Amazon and gumroad links in the description below so I'll be using vs code throughout this course and if you want to set up like me you can follow this link here and here are the VSS code python extensions that I use and all code examples that we go through in this course are included inside of this GitHub repository which you can see right here and if I go to the root of the project you can see that we have uh each section of the course has its own folder so we start off with this folder here and you can see we have different files for the different examples that we go through all right so let's get started so the first thing that I want to show you is that everything that you create in Python is an

object so let's create a couple of variables so we can have a string variable and called name and we'll assist assign that to Dany and we'll say age equals 29 so an integer so let's actually have a look at what um type of objects these are so if we print the type of name and we also print the type of age and then let's run this file uh so we can say python script.py we can see that this uh name variable is assigned to a string object so an Str Str object and objects are made from classes so you can see that this object this name object is made from the class St Str okay and classes are basically the blueprints for objects meaning that they describe what an object should look like so we can also see that this age variable is assigned to an integer so in Python that means that this uh is a int object so it's made from the class int and different objects have uh different behaviors so for example on string objects there are different functions or methods as they're called in objectoriented programming that allow us to manipulate the data contained within this object okay so the data here is just a sequence of characters that spells Danny so if we call name dot okay we can see that vs code now provides us with a load of uh methods or functions that are available on this string object so if we say uh name. upper uh we can see that this will manipulate the data contained within the string object so if we run this we can see uh we need to get rid of this type actually so let's run that again and you can see that the data has been manipulated is now uppercase but if we have a look at this uh the methods or behaviors available on an INT object you can see that they are different from those contained on a string object okay so if we look at the uh name variable here so the name variable is a is assigned to a string object okay which is built into python so the string object is Created from the string class which is built into Python and the St class or the string class defines what each string object looks like and provides useful methods which are functions that operate on certain objects such as name dooper to make all the letters in the string object uppercase um so they so the methods basically or the functions manipulate the string data so the int and the St classes demonstrate some of Python's builtin classes that we can use to create data structures like strings or integers but we can also create our own classes then create objects also known as instances from those classes so let's actually create our own class now so let's create a dog class and on that dog class we can Define what information so what data or attributes and what behaviors or methods or functions every dog should have so first let's create a very simple dog class that has no data and just one Behavior so the way we create a class in Python we say class which is a python keyword and we can call this class whatever we want so let's just call it dog then we have a colon and then within the class we can Define either data or attributes and we can also Define uh methods or functions so a method is basically just a function contained within a class so we can say uh def bar and then in methods we have to put this self parameter now don't worry about this for now we're going to come back to this and let's just print uh wo woof okay so now that we've created a dog class we can use it to create dog objects so let's Define a variable called dog and assign it to an instance of which means an object made from the dog class so we do that by calling the class name and essentially call ing it or instantiating it so here we are creating a dog object and assigning it to a variable called dog so let's call this dog one and now what we can do is um we can access the uh method on this dog uh object so dog one object by saying dog one dob okay so that's how we access uh methods on objects so on this dog one object we have a method

or a function called bark and then way we access that is by calling uh the variable name Dot and then the method name okay so let's run this and see what we got and we got wo woof okay we can also create other objects from this class so we can say dog 2 equals dog and say dog 2. bark and again again we get the same thing okay but this is not very useful because all of the objects that we create from this dog class will be the same they just have this bark method which is I does the same thing and it doesn't matter uh what object we call Bark on it's always going to just print woof woof it's pretty useless really so let's add some attributes or data fields to this dog class so that each dog object or instance can have a name and a breed so we can do this in the dog class by defining an init method okay so init is a special method in Python that is basically allows us to um set up some data fields or data for the for the class okay so let's just actually create this for now and then we'll explain it afterwards so again ignore this self keyw but we want to give each dog a name and a breed okay so the way we can do this is we say self. name equals name and we can say self. breed equals breed okay so these uh the names don't have to match by the way we could call this I don't know first name for example and then ass sign it's a first name however generally um it is you know the normal thing the sort of convention to just call them the same thing basically okay so we can see here now that when we are creating these dog objects vs code is showing us that we have an error so if we hover over this we can see the arguments are missing for the parameters name and breed so whenever we create a dog object now we have to pass a name and a breed so let's call this dog Bruce and this can be a let's say a Scottish terrier and this dog can be called freyer and this dog can be of breed Greyhound okay so now what we can do is we can actually access the data on this on these dog objects so we can print uh dog one. name and we can also print uh dog one. breed okay so these are data fields on the dog object this dog one object has a data field for name and it has a data field for breed and we can access those using the same dot notation that we used to call uh methods okay and we can also do the same for dog two okay so we now have a way of kind of uh making uh different dog dog objects uh different from each other so let's run this script and we can see that we have Bruce Scottish terrier and then we have Freya the Greyhound okay so let's just recap what we've done so far so here we've created a dog class which has an inate method and the inate method with a double underscore on each side is a special method in Python that is ran only once when an object is instantiated so we can see here we are instantiating the dog class or creating a dog object from this dog class and assigning it to dog one and we're doing the same here creating a dog object and assigning it to a variable called Dog 2 and this init method is run only once when an object is created or instantiated so it's run once here and it's ran once here when we create these objects and it's very common to set up an object's data inside of this method so we can see here that when we create this dog one object that we are setting up the data stored inside of this dog one object and it's Unique to this object and we do the same for dog two we pass the data and we set up some data fields inside this init method when we create the object and again this is unique to this dog 2 uh object we can also create different types of objects so so far we've created uh dog objects but let's say that each dog should have an owner and each owner should have a name an address and a phone number so so let's create a class that allows us to create owner objects with those data attributes so let's come down here and define a class called owner and this needs to

have uh so each owner needs a name an address and a contact number so if you can remember we can add data fields to an object by uh defining an init method okay which is a special python method method which allows us to kind of initialize objects so let's create an init method and again we pass the self parameter which we will get to a little bit later in the course and we're going to pass a name parameter and address and a contact number okay so then we can assign uh these sort of parameters that we pass when we create an owner object and assign assign them to data fields within the owner object okay so we can do this by calling self. name so we create a data field on an owner object called name and we assign it to the name that we pass when we create an owner object okay so we can say name we can do the same for address and we can do the same for the contact number and just to show you that these can be different let's call this one phone number so we're calling the data field on the object phone number and the parameter we're going to call contact number okay just to show you that they can be different uh names okay so that's all we're going to do now for this owner class so in our dog class let's add an extra parameter in the init method so that we can pass and owner object when we create a new dog object okay so then we can create an owner attribute with self. owner and assign it to the owner object passed in as an argument during the dog object creation so let's create an owner parameter in this init method and we can then create a data field on each dog class called owner and and assign it to the owner uh object that we pass in okay so now we can see we get a red line again because we need to pass an owner to each of the dogs so what we can do here is we can say owner one equals owner so we create an owner object we can give this owner a name so let's give it a name of Danny and then some address so let's just say one 122 Springfield drive and then we need to give it a contact number so let's just say 888 8999 and we need to pass an owner to the dog so we can pass the owner here okay and that's going to now be a data field in this dog object okay so now this dog dog one has an owner of owner one this owner object that we create here and and we can do the same for dog 2 so we can call this owner 2 and let's just change the name to let's say Sally and we'll just keep this the same and we can pass the owner here owner to just like that and what we can do now is we can actually access each dog's owner data uh using dot notation so let's actually get rid of this for now just so we can see what's going on so we can say dog one okay now if we press dot we can see that each dog now has an owner uh attribute okay or data field so we can say dog one. owner so that's going to give us this owner object but what we actually want to do is print let's say the uh owner's name okay so we can say owner. name you can see here we're getting the owner object uh owner objects have fields for name address and phone number and you can see those are showing up here we have name address and phone number here that we can access on this owner object contained within this dog object okay so if we actually uh print this and copy paste that in here and let's do the same thing for dog two and let's run the script and you can see that the owner of dog one is Danny and the owner of dog two is Sally which is correct because we passed owner 2 into dog 2 when we create this dog 2 object and the owner's name is s okay so just to summarize we've kind of displayed here that we can create different kinds of objects so we can Define different kinds of classes and we can kind of have relationships between different objects so for example dog objects have owners okay so we pass an owner object into uh dog objects when we instantiate them or create them and we can see that dog

objects have a data field of owner which is assigned to the owner object that we pass in during object creation Okay so here is the object creation for dog one and here we create an owner object and we pass that into this dog uh class when we instantiate the object so let's now clarify all of the concepts that we've learned so far and then we'll also create another simple example to help you understand the concepts of classes objects attributes methods the self parameter and instances in Python so first of all let's just clarify uh some of the basic objectoriented programming Concepts that you've learned so far so a class is like a blueprint for creating objects it defines what attributes or data and what methods or functions or behaviors that the objects created from this class will have and an object okay so an object is an instance of a class so think of an object as a specific example created from the class blueprint with its own data attributes are the variables that store information about an object so um for example a dog uh object has a name attribute a breed ATT attribute and an own our attribute so methods are the functions defined inside of a class they Define what actions or behaviors an object can perform so dog objects each have a bark method they have their own bark method um and they can perform this Behavior so let's now take a look at the self parameter which we've been avoiding so far so self refers to the instance or the specific object of the class itself so let me just clarify that a little bit so when I say refers to the uh specific object of the class itself so this uh is a specific object of the dog class it's its own you know thing it's it's its own object and this is a separate object okay so they these are both dog objects but they are different okay we're passing in different data and we're assigning them to different variables so this uh self parameter or self key word is used to access the uh specific objects attributes and methods from inside of the class okay so let's again clarify this so when we instantiate this dog object okay we pass in the variables we pass in a name variable a breed and an owner now within the class uh this self refers to this specific object okay that we are creating here so what we're saying is that self so on this specific dog objects object it should have a name attribute assigned to the name that we pass in uh which is Bruce okay so again for the breed we we pass in the breed when we instantiate the uh object and we can access the current object with self and we say that on the current object there should be a data attribute called breed and that should be assigned the value that is passed in during instantiation so hopefully that makes uh sense to you uh that we'll probably get you will get more familiar with this uh as we go further down the course it will make a lot more sense as we explain some other uh objectoriented programming Concepts such as uh static attributes so when we get to the static attributes part of the course I think this will make a lot more sense if it doesn't already Okay so let's leave that there for now so let's just finally discuss what is an instance or what it means to instantiate so when you create an object uh from a class you are instantiating the class okay and the object is an instance of that class so to be honest you will have probably heard me mixing up uh class and object when I when I say instantiate an object what I actually really mean is instantiate the class to be more precise so if I make that mistake uh I apologize but essentially we are instantiating this dog class to create an instance of the dog class which is a dog object so an instance of it the dog class is a dog object okay so that's just the kind of terminology that is used within objectoriented programming so let's now create a very simple example to clarify further all of the object oriented programming Concepts that we have covered so far so we're going

to create a very simple class called person and we're going to give this person attributes or data for name and age and if you can remember we can do that in the init method so we pass the self parameter and then we can say name and age and we can say self so self refers to the uh current or specific objects uh the current or specific object and then we can give it this uh give this object a uh data field of name and set it to the name that we pass in during the object creation okay and we can do the same for age and let's actually give this person uh let's give each person object a uh Behavior or a method that allows them to greet okay so we can say print and we'll do a format string here and we'll say uh hello my name is and we will say self. name and I am self. AG years old so you can see the reason we pass self as a parameter or the reason we have a self parameter on each kind of method that we Define within a class is because it allows us to access the data that we assigned to that object in the init method okay so it gives us access to the current objects data okay which is very useful as you can see here it allows us to greet someone in a kind of personal way so let's just again explain the code what we've uh explained the code uh that we've created so far so first of all we have a class definition so class person right these two kind of words here uh this defines a new class called person it's like and a class is like a template for creating person objects then we have the special python method the init method uh that automatically runs when we create a new person object then we have self. name equals name and self. AG equals age which initialize the attributes name and age uh for each specific person that we create for each specific person object that we instantiate then uh we have the uh self keyword which allows each person object to have its own values for name and age so the attributes so self. name and self. AG are attributes of the person class they store data about each person object and then we have a method called greet which is a method that displays a greeting message and self uh gives access to the specific person objects name and age okay so now let's actually go ahead and create a person object so we can say uh person one equals person and we'll give it this person person a name of Alice and an age of 30 and let's call person one. greet okay and let's run this so you can see we got hello my name is Alice and I am 30 years old so you can see the um the Behavior now the method on this object is now kind of personalized right each method is personalized because it has access or it is access accessing this current uh person one objects data fields for name and age and we can do the same thing uh again but create uh call this one person two and we will give this uh person a name of Bob and Bob can be 42 and if we run the script again we can see we get a different personalized greeting for each person okay so let's again uh let's go ahead and explain this uh code here so first of all we have instantiation so person one equals person and then we pass in the data so this creates a new person object with the name of Alice and an age of 30 and here we have another person object that we are creating and assigning it to to a variable called person two and this one has a name attribute of Bob and an age attribute of value 42 and person one and person two are both instances of the person class and each instance has its own unique data for name and age okay which we can uh set using self we can then all the methods stored on each object so person one has its own greeting method and person two also has its own unique greet method okay so person 1.g greet calls the Greet method on person one which uses the name and age attributes specific to person one okay and the same is also true for person two okay so each person object displays a unique greeting based on its

attributes and this shows you how different instances or objects of the same class can hold different data and perform the same behavior with that data so again just to summarize a class is a blueprint for creating objects in this case person is the class an object or an instance is a specific example created from the class so person one and person two are both both objects or instances of the person class we have attributes which are variables okay so we have attributes for name and age that store uh data about an object okay so these are attributes of the person class we have methods which are functions uh defined in a class that perform actions and greet is a method that displays a greting and then we have the self keyword here and also here whicher refers to the specific instance of a class so here this is a specific instance of a class and here person two is also a specific instance of a class and it gives each object access to its own attributes and methods so this example introduces the fundamentals of objectoriented programming in Python providing you with a nice foundation for understanding how classes and objects work and again don't worry too much if you don't fully understand the self keyword for now it will make way more sense once I introduce you to static methods and attributes a little later on in the course so we're now going to have a look at the different ways of accessing and modifying data in objects and we'll also discuss the preferred way of doing this in Python so let's create a user class that defines what data or attributes and what Behavior or methods a user should have so we'll create a class called user and we will um Define some attributes within the init method so a user is going to have a username an email and a password and we can set those attributes on the object using the self keyword so we can say the current object and we want to set a data field or an attribute on that current specific object called username and assign it to the value that we pass in when we create the object okay and we'll do the same for email and we'll do the same for password okay we also want to define a method that uh allows us to say hi to another user okay okay so we'll create a method called uh say hi uh to user we have the self uh parameter which we always pass into these uh methods that are you know specific for each object okay um and we're going to pass in a user that we want to uh that we want this user to say hi to okay so then we're going to print uh a message message and what we'll do is we'll say sending Sending message to user. username uh let's see what the mistake I think that's okay and then we'll say uh colon hi user dot username it's self. username okay so in here this say hi to user method we pass a user which will be a different user object you can see here that user object we are going to access the username on the user object that we pass in and we're also accessing the username of the current user object Okay so let's actually show you uh an example of how to use this uh class now so let's first of all create a couple of users so we'll have user one and let's give this guy a username of Dan the man and we'll give him an email of Dan gmail.com and a password of 123 and then we'll create user two uh so this can be Batman and we can say bat at uh let's say outlook.com and the password can be ABC so let's say that we now want to uh we now want user one or user one wants to say hi to user two so what we can do is we can say user one dot and we want to say hi to user we want to say hi to user to okay so now if we run the script you can see here Sending message to Batman so we are sending a message to Batman uh and the message that we're sending is hi Batman it's danam the man so you can see here again on this say hi to user method we pass the user that we want to say hi to so the current object that we are dealing with is user user

one so self is effectively referring to itself so user one and we are saying hi to user 2 and we are accessing the data fields on user 2 so let's now take a look at how we can access and also modify or update the data within an object so let's just get rid of this user two for now and just have deal with user one for Simplicity so as we've already shown you can access data on an object using dot notation okay that's very simple and but we can also assign uh a new value to this uh email attribute for user one so what we can do is we can say user one. email is equal to uh let's say Danny at uh gmail.com okay instead of Dan okay so now if we actually print the uh email of user one after modifying it we should see a different value so you can see initially it's Dan gmail.com and we've modified it here to Danny gmail.com okay but the problem we have is we can actually modify this to any value that we want for example 1 two 3 or let's say uh Jim for example or just Dan right we can just modify it to a non email address okay so this is not a valid email address but we can just assign it any value that we want so this is not good okay so what we actually need is a way of controlling the way that we can get and set or read and modify the data on an object so what I'm going to do now is I'm going to show you two different ways of doing this of getting and setting data on an object so the first way I'm going to show you is the kind of traditional Java style approach which is was sort of popularized uh popularized in the Java programming language and then I'm going to show you the more modern uh python way of doing things using properties and this is also popular in in the C uh programming language so the first approach that I'm going to show you of uh safely accessing and modifying data is the kind of traditional approach which is where we make the data attribute private and use GAA and Setter methods or Getters and Setters so first of all let's modify our user class to make email protected and I'm just going to get rid of this say hi to user method because we don't really uh need it anymore so the way we make a data attribute uh protected is by prefixing it with an underscore okay so this is the kind of python way of making a data attribute protected okay and when we make an attribute protected it means that we shouldn't uh read this attribute outside of this class or any of its subclasses and don't worry about sub classes for now if you are unfamiliar with them because they will be introduced in The Inheritance section of this course so basically we make an attribute protected in Python by prefixing an underscore in front of the attribute name so I just want to have a quick discussion of kind of Python's take on access modifiers so unlike languages such as Java or C++ which enforce strict uh access control so using keywords such as private or protected python takes a more relaxed approach so in Python a single underscore before the name for example underscore email is a convention in Python that indicates that something is intended for internal use within the class or module and it means that it's um should not be used outside of the class so let me show you what that means so let's create for example a method uh I don't know let's call it uh get email right so this is essentially a getter method and what we do is we just return email and again we need to pass self so we can actually access the data on the object and we return self. email so here we are accessing the email field this protected email field within the class okay within the user class which is no problem we can do that um however um what we shouldn't do now is access this email field outside of the class okay so if we call user one. email we can see that it no longer exist be uh exists we we get this underscore email which does exist however in Python uh python developers will look at this and think hm

why are you accessing a protected attribute you're not supposed to do this okay so that's a kind of python convention really python doesn't really really enforce this restriction right it doesn't you know throw an error here it's python doesn't say email is protected we shouldn't be accessing this let's you know throw an error and you know abort the program like you know programming languages would such as Java or C it's kind of um the attribute or you know sorry the uh the protected attribute is still accessible from outside of the class so you can see here we've accessed this uh private or protected uh attribute but it signals to developers that it's meant to be protected or internal to the class okay so it's meant to be used in the class as we've done here we've used it within a method however we're not supposed to do this outside of the class okay it's just a python convention basically so for example even though we made email protected we can still access this underscore email from outside of the class but in Python you are trusted not to so underscore email is now only supposed to be used within the class so let me just get rid of this uh get email uh method and maybe give a better example so if we have a method called clean email and let's say that what we want to do here is basically make the email lowercase and strip any white space uh at the beginning or end of the email so what we can do here is we can just return self. email and we can say uh lower and strip okay so we uh make it lowercase and we strip the trailing and leading white space so let's now remove this we're not supposed to access uh email outside of the class anymore and let's just remove this so let's now print uh user one. clean email and run the script so what if we actually provide a kind of unclean email with an uppercase letter at the front and some white space and if we actually print user one. email now we're not supposed to do this but I'm just going to show you uh for Dem demonstration purposes what the initial email is and what it's kind of cleaned to okay so we have Whit space at the front and back we have an uppercase letter and then we clean the email within uh the class basically so I just wanted to sort of demonstrate here um why you would want to actually access something within the class but not outside of the class so by prefixing email with an underscore we are effectively communicating to the python developers that this is a protected attribute and it shouldn't be accessed outside of the class okay so if we now do user one uh. email equals uh you know just some random invalid email this is something that you are not supposed to do right because we have pre prefixed this with an underscore so effectively this is uh this is poor kind of uh python etiquette if you like um so what we can actually do is provide some methods within the class for modifying this email field in a controlled way rather than just allowing um uh the the email field to be assigned to any value outside of the class and that's kind of the advantage of making this email feel protected we can now provide methods which we will do shortly for uh controlling how the email is set uh so that it can't just be set to any invalid value outside of the class before we move on with this example I just want to to discuss Python's consenting adults philosophy so gido van rossom the creator of python has this consenting adults philosophy which highlights Python's emphasis on developer responsibility rather than and strict rules so this philosophy suggests that developers are trusted to respect the convention of not accessing underscore prefixed attributes or methods and the access is not strictly prevented as python will assume that developers will act responsibly and won't misuse or access protected members unless it's absolutely necessary however it is still possible to make a attributes private if absolutely necessary so

you can do this by actually putting a double underscore in front of the attribute so for example if we want to make email uh private we can put a double underscore in front of it and now it's actually impossible to access email outside of the class so if we try to run this file you can see that python will now throw an error because um this attribute is protected and effectively it says that this object this user object has no attribute of double uncore email even though we created one with this name so how does this work under the hood so python will actually change the name of this attribute so double double uncore email so that an error will be thrown if you try to access it outside of the class so behind the scenes when you put a double underscore in front of the attribute python will do something called name mangling which basically means to change the name of the attribute so that it can't actually be accessed outside of the class or in other words if we try to access the variable outside of the class we will get an error because under the hood python has actually changed the name of the attribute Okay so kind of makes the attribute effectively private okay so it kind of enforces the uh the rule rather than leaving it up to the developers to you know behave themselves as was the case when we have a protected uh attribute with just this single underscore so I just now want to summarize the differences between protected and private variables so protected with the single underscore and private with the double underscore both of these variable types can be accessed within the class so we could access a protected or private variable within the class that's perfectly acceptable uh however protected variables can be accessed outside of the class but as a good python developer we should respect that rule and not access P protected variables outside of the class so this is actually something that we're not supposed to do here however with private variables also known in python as name mangled variables so let me just actually uh show you how that is spelled just in case you're wondering what I'm saying so it's name mangled and mangled basically kind of just means changed basically under the hood as we discussed earlier so private variables cannot be accessed outside of the class so protected variables can but you're not to supposed to whereas private variables with the double underscore uh cannot be accessed outside of the class because under the hood python changes the name of the attributes so that it can no longer be accessed now the question then arises uh of whether you should use protected or private variables or when should you use each type of variable so in most cases using a protected variable is is enough to Signal internal use within the class and protected variables generally offer better readability and flexibility making them the Preferred Choice in Python so generally you should probably reach for protected variables rather than private variables unless it's absolutely necessary to make the variable private and make it effectively much more harder to read outside of the class so if we now move back to our example uh we can see here that we made email protected okay so we now should not uh read or modify the email outside of the class so you may now be thinking uh how do we actually uh read email or modify email outside of the class and we do this uh using geta and seta methods so I'm just I'm just going to going to get rid of this clean email uh method for now and let's see so let's also remove this so let's create a getter for email so we can see email is protected so we can't read it outside of the class so this is not great because we should be able to read the email outside of the class uh but currently we we're not supposed to right so we can create a method a geta method called get email and this is a convention uh throughout

pretty much all programming languages or all object-oriented programming languages uh for creating geta methods so what we do is we return self. email so the convention is basically to prefix the uh name of the variable with get so in Python we have getor email you know in Java we would probably have uh get email ma like this with the kind of um this style of casing uh so uh in Python we have the sort of uh snake case um and then we just return the uh name of the variable okay so now outside of the class we have a way of getting the emails so we print the email of user one of get email okay so we can see here uh if we're not supposed to do this because it's protected with with an underscore we're supposed to be responsible python developers so we can just call get email the geta method and then if we run this script we get back the email okay so this is all okay so far but what if we actually wanted to set date uh the email to a different value so let's actually just make this email kind of valid again so if we want to set the email we could do this and we could set it to any value however we're not supposed to do this again this is a protected attribute so we should not be reading or uh updating this value outside of the class okay or we shouldn't be doing this directly okay so what we need to do is we need to create a Setter method okay so we do this by convention we call it set email so set and then the attribute name so set email and inside of here what we can do is we can pass the new email okay so this is the email that we want to set it to and we can say self. email is equal to the new email address okay so now instead of doing this which we're not supposed to do we can say user one. set email and we can pass the new email so maybe Danny outlook.com okay so then if we print the email so if we get the email again we should get that updated value so you can see there we have the updated email address so again because we made email protected and we can't or at least shouldn't as python developers access it outside of the class we've had to provide methods to read or modify this email attribute so you might now be thinking but why do we need to do all of this when we could just read or modify the data by accessing the data directly with say user. email as this seems to require a lot more work and the reason that we do this is because it gives us full control over what happens when an email is read or modified for example if we wanted to print the email address and the current time to the terminal every time any uh user objects email is read we can easily add some code to this get email uh method okay so we can easily print uh something to the console so we can say email accessed at uh and let's just import the datetime module so from datetime import date time and we can say date time.now to get the current uh date and time and let's now run the script and we can see that whenever we access an email it's going to log to the terminal email accessed at a certain time now if for example there are multiple Parts in our program where we are accessing email but we still wanted to log this every time an email was accessed then we would have to make changes to multiple parts of our program every single time that this method was called okay so we'd have to make lots of modifications to our existing program whereas if we just have this geta uh method then we can just add it to this uh method in one place and every time that we access an email this will be logged to the console so you can see that by using geta methods um it kind of um provides a kind of single point where all of the logic can be uh placed when we get an email all of the kind of things that we want to do when we get an email can be placed within this uh geta method and it so effectively it's providing a controlled way of accessing email and we could also do other things for

example maybe we would want to check whether a particular user is authorized to read emails you know if they are maybe an admin then we will allow them to read the email otherwise we will not provide them access maybe we could throw an error if the user is not an admin so it provides a controlled way of reading the email address rather than just accessing it directly with say user one. email okay so and the same thing is true for setting an email because currently we can set email to any random value so let's now add just a little bit of say validation logic to uh help to make sure that emails are actually set to email addresses to valid email addresses so what we can do in our set email uh method we can say if there is an at symbol in the new email then we can set the email otherwise we will not set it and maybe we could also throw an error if there's no at symbol but let's just keep things simple for now so let's actually have a look at what we've got here so this is an invalid email address it has no at symbol so let's see what happens so you can see here the email was not updated because it didn't contain an at symbol whereas now if we provide an at symbol and.com you can see that the email is now a valid email and it is updated it's set okay so we can you can see the set a method is providing a controlled way of updating email addresses to ensure that the the uh users of this class or the other parts of the program using this class cannot do anything stupid okay so it really helps to uh improve in this case the Integrity of the data throughout our program and of course this is not complete we would also need to have other validation checks but it's a nice simple example so in summary Getters and Setters allow us to access and modify data in a controlled way so for another example um before we set a username we could actually check that the username is not already taken in the database okay and before setting a password we could check that it's at least 8 kartic long and contains both letters and numbers and again we could do this outside of the class okay by just accessing these fields directly without gets and Setters but then every time set a password we'd have to remember to validate it okay and this leaves it possible for invalid passwords to be set so it's much safer and also less code in the long run and much more maintainable to do the validation in one place within the class however so far we have looked at gets and sets which is kind of the traditional Java style objectoriented program programming approach where we kind of set uh variables as protected or private and then provide gets and Setters for each attribute and this is a pretty verbose or long winded way of doing something very simple and it's often why you'll hear people complain that Java is a verose programming language and luckily python def uh provides a much nicer kind of cleaner and less loated way of effectively doing the same thing using what uh something known as python properties and that's what we're going to take a look at next properties are the recommended approach to controlling access to data in Python so here we have an example a very simple user class with an init method which initializes an objects uh data attributes so we give objects a username uh an email and a pass password and here we are creating a user object passing in some data and you can see here that we are accessing the uh a data attribute directly okay so we're saying user one. email and we are setting it to just some invalid value and if we want run the program you can see we got no problem this is all okay so this is not good uh because we can freely um because any other classes or any other parts of our codebase or our software effectively can uh freely read and modify emails so anywhere within our program anywhere else in our program that is using user objects can freely set emails to

whatever uh value they want so this is clearly uh not a good solution here so the way that we can fix this and the solution to this problem is by creating getter and seter properties for the email attribute okay so before uh previously we looked at creating GAA and seta methods which is the kind of java way of doing things I guess you could say here we're going to have a look at the more python way of doing things uh which is using a GAA and seta properties so first of all I'm going to show you how you would go about this so this is kind of the standard way of creating properties in Python so first of all what we do is we make the attribute private so we can do that by prefixing with a single underscore we then uh State uh the property decorator so we say property so at property we then create a method uh with the same name as the uh attribute that we want to provide a property for so we say def email because we're creating a get a property for email okay and then we pass self so that we can actually access the uh current objects attributes and then we just return self. email okay so now what we can do is also create a uh set a property so this is a geta property so whenever we read email we can just call user one. email okay so I'm just going to comment this out for now and we'll we'll just focus for now on this GAA property okay so what we've done here is we can still access email by just it looks like we're accessing the attribute directly but actually we are accessing the email property which is then returning the uh kind of protected email uh attribute okay so so now again then if we wanted to uh for example print something to the console every time an email is accessed uh we can just let's just keep things simple and say email accessed and run this you can see that we are uh printing to the console email accessed before we uh print the email okay so it's kind of very similar to using a uh creating a GAA uh method but it kind of makes access to email much more simpler because it looks like we're accessing it directly and the advantage of this is if we actually remove the uh property and go back to you know kind of what we were at initially where we just have this self. email equals email so this is a public attribute you can see that any code that is using this uh using these user objects doesn't have to change um it it if we were originally accessing the rute directly uh then all of a sudden we know we needed to Pro provide some logic later on in our program to you know print email accessed then we can just add that in and all of our classes in our program which could be a lot of you know classes using this these user objects don't have to change it still looks like we're accessing uh the email directly even though we're actually now accessing a property so that's the kind of big advantage of using a property we can keep things simple initially for example if initially uh when we create our program we don't require any logic at all when we get uh email addresses we just get them and we read them then this is fine but then later on you know maybe a few weeks down the line the boss comes in and says that you need to now uh print something to the console uh every time we access an email we can just add in this get a property make the attribute private but all of the code using user objects does not have to change we still just say user one. email whereas with a gets and sets if we um made this uh protected then provided a get email method then all of the code using user objects would then have to call the get email get a method rather than just keeping it the same as it was before so that's kind of the hu huge advantage of property you don't have to worry about creating Getters and Setters uh initially when you create a class you can just keep things very simple initially and then later down the line if we need to you know add some extra functionality add some extra

validation logic then we can add properties in and all of the code using these classes or objects doesn't need to change so that's the big advantage of using properties over geta and seta methods it's much less verose and it's kind of easier from the off if you like because um if I was using getter and Setter methods here then I would have to be thinking somewhere down the line it's very likely that we're going to need to add some logic to uh when we access or modify emails so therefore I'm going to have to make this protected you know right from the get-go and then provide these getter and set of methods whereas with properties I can just keep things simple access things directly and then down the line if we need to provide some logic we can just add the property in and all of the code using the user object doesn't have to change so that's a huge advantage of this method so let's now take a look at how we can create a set of properties so we have a geta property but let's now create a set of property so that we can actually set emails and if we just uncomment this you'll see that we have an error here and this is because we cannot assign to attribute email for class user and this is because we haven't actually created a set of property for this class we've created a geta property but we've provided no way of setting it so if we actually leaved our user uh class like this it kind of uh basically means that we can read email but we cannot update it uh but we want the ability to update emails so we need to create a geta property and the way we do that is we first state a decorator and first of all we start with the attribute name and then we say do setter this is just the way we create a set of properties in Python don't worry too much about it for now uh we create a method with the same name as the as the attribute and here uh within the method we have a parameter for the new email okay that we want to set email to and let's just add some validation logic in here so let's say if at in email so this needs to be uh let's see new email if at in the is in the new email then we can set self. email equal to the new email address okay so let's actually run this to see if it works so here we're trying to set email to this invalid email address we print the email and you can see that it's not been changed because this doesn't have an N symbol okay so we now have a controlled way of getting and setting data um for our user email uh attribute so let's just break down this code uh and summarize what we've done so far so you can see here self. email we've made this protected so this is a protected instance attribute um initialized to the value past um past when we create the object and the single underscore prefix uh means that it is protected meaning that it should only be used within the class like we're doing here and we don't use it outside of the class which is what we are doing so that's all good so far uh and then we have this property decorator so this at symbol uh at sign with property this is just the standard way in Python of creating a g a property so what this property decorator does is it turns the following method so this email method into a geta property and by making it a property we can access this uh underscore email attribute using user. email as we have done here and well as we've done here because we're getting it here we're setting it here so for the GAA property we can see we can just access it access it by calling email okay so that's great so um so by uh naming this geta property email uh without the underscore we are exposing a public email property while keeping this email attribute private so you can see this is a public email property me public meaning we can access it outside of the class and we keep the uh email uh attributes protected and you can see here we just return we are logging something to the console then we return the value of the email attribute okay we also then have a seter

property um so the way we create these is we have a decorator uh starting with the name of the attribute that we want to create a seta for then we say do seta again this is just the standard python way of creating uh set of properties and this set of method is triggered whenever a new value is sign is assigned to user. email allowing us to add uh this validation logic or any other logic during the ass assignment to email so whenever we call user. email equals some new value this set of property method is called giving us a chance to control the assignment so far we've explored instance attributes in Python classes and you now know that each instance or object of a class can have its own set of unique data however python classes can also have static attributes which are shared among all instances of the class so in this section of the course we're going to cover the differences between instance attributes and static attributes when you should use each and how the self keyword fits into the picture so a static attribute which is sometimes called a class attribute is an attribute that belongs to the class itself and not to any specific instance of the class or any specific object created from the class so this means that static attributes are shared by all instances or objects of the class it also means that there is only one copy of a static attribute in uh memory regardless of how many objects or instances you create from the class and static attributes can be accessed directly through the class itself and also through the instances although they are stored at the class level so let's go through an example to make this easy to understand so say that we wanted to keep track of the number of user objects that are created from the user class um so first of all let's create a user class and also create a static attribute that counts all of the instances or objects that are created from this class okay so let's clear everything out here and I'm just going to create a user class and here we're going to create a static attribute so we're going to say user count and we're going to initialize this to zero because initially there's going to be zero user objects uh we're then going to create our uh instance attributes which if you remember are created in the init method so we create the init method we pass self first we then pass a username and an email for the user so each user object is going to have a unique email uh username and email but this user count attribute which is static remains on this user class whereas these are instance attributes which are going to be assigned to each individual object created from this user class so we say self or the current object do username is going to be equal to username and the same for email and these are our instance attributes when but we're all every time we create a new user object we want to increment the user count by one so the way we can do that is we can say user so the class and remember this is a static attribute which exists on the class not on the objects created from the class so we say user do user count is going to be plus equals one so essentially we increment it by one okay so let's uh also provide a method for displaying the uh user so we we can just display the user's details their username and their email in a kind of nicely formatted way so we can say display user and then we're just going to print a format string uh so we'll say username is going to be equal to uh self. username and we can say email is going to be self. email okay so here we can see that the static class attribute is created in the class body okay so in within the class body itself okay it's only in indented uh once effectively whereas uh the instance attribut attributes are created inside of the inet method okay so that's the kind of difference of how we create static attributes and instance attributes in Python so let's actually see if this all works correctly so let's create a couple of users so I'm just going to

paste this in just to save a little bit of time so we'll create uh two user objects so user one and user two and since the static user count attribute is stored inside of the user class itself we can access this directly from the class so let's now count the number of users in the class so we can say user do user count okay so we're counting the number of user objects created because remember every time we create a user object we increment the static attribute by one so let's actually save this and run the script and you can see we get two back we have two user objects created which is what we would expect now we can also access the user count from the two instances that we created here uh even though this static us account attribute is um on the class we can access it via the objects but you should remember that it's still a shared static attribute it's still accessing this one you know uh uh static attributes stored in one place it's not stored you know uniquely on each object okay it's just stored in one place on the class um so let's show you how you can do that so we can print user one do uh user count and we can also do the same thing with user two okay so you can see this is actually accessing user on the class uh user. user count uh but it we can also access it uh from the instances created from the class so let's just check if that works and you can see it logs to each time we access us account so the main things to remember here are that static attributes are created once at the class level and are shared between the class it resides in and the instance objects of that class whereas instance attributes are created every time we create a new instance so here and here and instance attributes are sh um are contained within that instance so we can see that the instance attributes self. username and self. email are contained within each unique object okay so these are kind of recreated and stored inside of each uh object that we create okay so use self uh so this username attribute will be stored uniquely inside of this user one object and also for this user two object it will have its own username attribute assigned to whatever value we pass here okay so those that's the kind of difference between static attributes and instance attributes so you may be thinking when should you use static attributes compared with say instance attributes so static attributes are useful for data that is common to all instances of a class so let me give you some uh example scenarios where static attributes would make sense to use so um it makes sense to use static attributes for uh things like counters and totals like for example tracking the total number of objects created like this user count uh attribute is doing in our examp example here uh so it's static attributes are also useful for shared constants or values that are constant and applicable across all instances uh for example a default value or some sort of configuration setting um and they're also useful for class level configuration so any configuration or setting that should be the same for all instances of a CL of a class such as uh default parameters so just to summarize instant attributes are unique to each object and are accessed using uh well accessed within the class using self Dot and then the attribute name and they are ideal for storing object specific data so you know unique data effectively you know the username is unique the email is unique uh if it's a dog then the breed needs to be unique to that particular object um whereas static attributes are shared among all objects of a class and can be accessed with uh class name do attribute name and these are ideal for storing data that needs to be consistent across all instances a static method in Python is a method that belongs to the class itself rather than to any instance of the class so unlike instance methods static methods do not take self as a parameter meaning that they cannot access

or modify instance specific data instead they are used for functionality relevant to the class but not tied to individual instances and to define a static method we use the static method decorator so now let's go through a realistic but simple example to demonstrate the differences between static methods and instance methods so let's now create an example to show the differences between static and in inst methods so we're going to create a class called bank account and this is going to have first of all a static attribute called Min balance okay and it's kind of a convention in Python to capitalize constants so this is going to remain constant in the class and it makes sense to have this as a static class uh attribute because it's going to be shared between all uh bank account instances it really doesn't need it's not going to change and therefore it makes no sense to have it as an instant attribute um next what we're going to do is we're going to create some uh instance attributes so we can do that in the AIT method and we're going to have each bank account is going to have an owner and also a balance and we're going to initialize the balance to zero so in here we can say the inst we could we create an instant attribute of owner and we assign it to the owner passed in and we can say self. uh balance is going to be equal to balance so here we're creating a protected instance attribute okay and we're going to provide some methods that allow us to uh withdraw and deposit into the bank account in a controlled uh way so next up we're going to create our uh deposit method which is going to be an instance method okay and it's going to be an instance method because it needs access to an instance attribute okay which exists on the object so it needs to be an instance attribute so we're going to call this deposit and we deposit an amount of money and what we're going to do in here we're going to add a little bit of logic so we're going to say uh if amount is greater than zero then we can increment the balance by the amount so self do uh balance plus equals amount and let's just print uh a message to the console so that we can see what's going on so we can say uh let's see so let's see so self dot owner and then let's say owners uh New Balance and then we can just print the balance uh let's put a dollar sign here self dot balance okay uh and if the amount that we're trying to deposit is uh less than zero we can say print uh deposit amount must be positive okay and I'm going to create one more method now and this is going to be a static method okay so we do that uh we can create static methods by using the static method decorator above the method that we want to be a static method okay so now we can create our static method we're going to call it is valid interest rate and we're going to put in the uh we're going to pass the rate here the interest rate and what we can do is we can uh just return true or false if it's a valid interest rate so let's keep things simple and say if the interest rate is greater than or equal to zero um and less than or equal to five then it will return true okay so this is a static method that just exists on the class and not an instance method uh like this one where we pass the self as a parameter which will exist on each of the objects or instances that we create from bank account so let's have a look at how we would use this solution so let's create a bank account and set it equal to bank account and the owner can be Alice and we will set the initial amount to 500 so let's actually uh deposit some money so we can say account. deposit and we can deposit uh $200 and let's actually log this to the console so you can see uh whenever we deposit money we get a nice message of the updated uh balance uh and now let's actually use the static method that we created down here so let's print now we can access static methods uh from the class itself because they exist on the

class and not on each uh instance of the class so we can say bank account do is valid interest rate and we can pass three and we can pass do the same thing again and we can pass 10 and then we run this script and you can see three is a valid interest rate because it falls between zero and five and 10 is an invalid in interest rate so let's just recap what we have done here so we have an instance method called deposit and this method uses self um let's see use self to access the uh balance attribute specific to each bank account instance and calling deposit on account so here and passing in a value uh increases the account. balance allowing each instance to manage its own data independently uh whereas this static method uh called is valid interest rate uh this method does not interact with any specific account or instance instead it performs a check on the rate on the rate parameter which we pass in here uh independent of any bank account object making it suitable as a static method and we call it directly on the class using Bank account.is valid interest rate and passing in an integer argument so you might be thinking where are static and instance methods stored so both static and instance methods are stored in the class itself not in each individual object that we create from the class so this means that we have memory efficiency as only one copy of each method exists in memory no matter how many instances that we create so just to clarify uh for when you should use static methods so static methods are ideal for tasks related to The class's Domain but don't require any specific instance data so for example uh you can use static methods uh for like utility uh functions so on this bank account class we use a static method to check if an interest rate is valid um but you can also use uh static methods uh to help to process data or to format outputs that don't rely on any instance specific data okay so is it uh this method here doesn't rely on any specific object data such as the owner of the account or the balance it just um takes in a rate and returns true or false so it makes sense that this is a static method it's kind of like a helper or a utility method so using static methods can help to give a clear separation between behavior that require instant specific data and behavior that doesn't and this approach also reinforces encapsulation which is an important objectoriented program concept that we'll dive uh into shortly uh so it reinforces encapsulation by keeping related functionality within the relevant class so so far we've discussed how we can control access to attributes by making them either public protected or private but we can do the same thing with methods again using onecore for protected uh to be which means to be used within the class or subclasses but not outside of the class or two underscores for private which means to be used within the class only and again we'll get more into uh using sub uh what subclasses are uh very shortly in the course so let's take a look again at our bank account uh class and let's say that we wanted to create a protected method meaning that it should only be used in the class or it's subclasses um to check uh if the amount being deposited is valid so rather than uh having this here we have a method that does this for us so we can create a protected method with an underscore and we can call this is valid amount and we pass self we pass the amount and we return uh the same thing that we have here so we can say amount greater than zero and we can now call uh self do is uh valid amount okay and then pass the amount okay so this is a protected uh um method we can also create uh private methods so let's for example create a method that uh logs the uh transaction type and the amount being uh withdrawn or deposited currently we only have a deposit method just to keep things simple we didn't provide a

withdraw method but let's create this uh log uh transaction method and this is going to take the self parameter and the transaction type and the amount and then in here we can print so let's see so we can say logging the transaction type of amount account and we can say new balance is self dot balance okay now rather than logging this uh message inside of the deposit method we can just call this log transaction um method okay so we can say self DOT log transaction and then we can pass the transaction type which is a deposit and we can pass the amount which is amount okay so this should work pretty much the same as before but we now have a few extra methods okay so we have a protected method here with a single underscore so if we actually look um what would happen if we try to access this outside of the class so if we have a scroll down what would we call it again so is valid amount oh is valid amount you can see that we can access uh this outside of the class but we are not supposed to because it's got an it's prefixed with an underscore meaning that it should be used within the class only um whereas the private method um which is called log transaction so if we uh let's see so let's say withdraw and 300 now if we run the program you can see that we get an error because this is a private method and we are not allowed to access it outside of the CL uh the class so you can see here we have the protected method which uh is used in internally to validate the uh deposit amount and is accessible by any subclasses if needed and then we have the private method called log transaction with the double underscore uh which is used to log transactions and is intended only for internal use within this bank account class and this method should not be accessible or overridden in subclasses okay and again we'll get to that uh very shortly in The Inheritance section of the course encapsulation is a fundamental principle of object-oriented programming that involves bundling the data or the attributes or fields and methods or the behaviors that operate on the data into a single unit called a class encapsulation helps in hiding the internal implementation details of a class by only exposing the necessary functionalities to the outside world so let's now create a simple example demonstrating encapsulation in Python so first let's create a bad example with no encapsulation so we're going to create a class called bad bank account this is our sort of bad example and then we're going to refactor this to use en capsulation and solve the issues that we find with this uh bad example so we're going to create an init method and let's get rid of that and we're going to pass in a balance an initial balance which we will set uh to uh a balance instance attribute and let's now have a look at how any users of this class so any uh other part of our program that uses uh this class is a kind of client of this class that's the kind of um terminology that we use so any other parts of our program you know let's say we created a new file you know some I don't know uh account dop and we imported this class you know this new file would be a client uh for this bad bank account class okay so any other parts of our code using this class effectively so uh users of this class or clients uh now have free reign to assign balance to whatever value that they want so first of all let's create an account and pass in an initial balance let's say zero and we can now set uh balance to any value that we want for example a negative number and this is not good because um our our programs logic let's just say uh should not allow the balances to be set to zero uh set to a negative number okay but if we print the account. balance we will see that H has been set without any uh errors being raised okay so this is not ideal so let's now create a better Bank accounter class with encapsulation of the fields and internal logic so let's just come down

here and create a better example that uses encapsulation so we we have our init method um and in here we're going to set the balance equal to zero initially 0.0 so float and we're going to make uh balance a protected attribute so we can't access it directly outside of the the class we are effectively encapsulating this balance attribute inside of the class okay making it inaccessible or in Python at least it means we shouldn't access it outside of the class okay so now what we can do is we can either provide a geta uh method or a uh geta property for getting this balance and in Python generally you want to be create using properties because they are much uh as we discussed earlier much less for house and you can just create them as needed ra rather than GAA methods where we have to create everything up front so let's create a uh a geta property for balance so we say balance def balance self and we're just going to return self dot balance okay so there's our geta property and let's now create uh methods for depositing money into the account so what I'm actually going to do is I'm not going to create a set of property okay we what I'm going to do is I'm going to say that in order to modify the balance we have to call either a deposit method or a withdraw method so I'm going to create a deposit method here and that takes an amount and first of all we're going to check if the amount is uh less than or equal to zero so if we try to deposit zero or less which makes absolutely no sense so we're going to raise a value error in Python with a message of deposit amount must be positive okay otherwise if we pass this test we can set the the balance uh so self do balance and we can just Plus on the uh amount okay so we add the amount to the current balance okay and now let's create a withdraw method so we can say death def withdraw and we want to withdraw a certain amount and again these are instance methods because we are passing the self parameter uh and we need to access instance data so it makes sense that these are instance methods or they need to be instance methods otherwise we can't access the uh balance on each object so in withdraw what we can do again we can check if the amount uh being withdraw is less than or equal to zero then we want to raise a value error so let's just copy paste that and we will say this time withdraw amount must be positive uh and then we also need to check here if um amount is uh the amount that we are trying to withdraw if it's greater than the actual amount in the balance then we need to also raise a value error and we'll provide a message called insufficient insu insufficient ient funds okay and if we pass all of these uh conditional statements then we get to the end and we can just uh subtract the amount from the balance so we can say uh minus equals the amount okay so now let's go ahead and actually create an account from this uh bank account class so we can say account is equal to bank account and let's actually print the account balance okay so because we have provided a property a geta property for balance we can just you know uh access this balance property it looks like we're accessing the balance attribute directly but actually we're calling this method um and if we try to uh assign uh account. balance a value so for example minus one you can see we get an error so we cannot assign to attribute balance for class bank account and this is because we haven't actually provided a set of property okay we're we're kind of forcing the user to use these two um these two uh methods okay for depositing and withdrawing uh cash okay so that's the way we've designed this class to work so let's actually try to deposit some money so we can say account deposit and let's deposit uh one let's see uh 199 and let's print the account. balance and then let let's actually withdraw some money so account do withdraw and let's

withdraw $1 and then print the balance okay so let's run this program and you can see uh let's see so let's just uncomment what we had previously so let's uncomment uh all of this and run the program again so we can see the initial balance is zero which we are setting in the uh init meth method when we create a new account object which we're doing here we deposit 199 print the balance and then we withdraw one and print the balance again and all of this is working correctly and if we try to do something stupid such as uh withdraw uh an amount greater than we have in the account so for example account withdraw uh 100 we should get an error message uh because we are actually uh trying to withdraw more than we have an account so we get this insufficient funds uh method so now we have a nice controlled way of uh it of updating this balance field in the account so let's just recap what we've got here so notice in the bank account class again we we've provided a get property for this protected balance attribute but no set of property and this allows uh users of this uh class to read the balance using account. balance but not set the balance directly with for example account. balance equals minus one you know which would still be bad so to modify balance bank account provides a simple public API these two uh methods deposit and withdraw and this ensures the Integrity of the balance attribute value and that our expected program logic can't be violated so step by step the bank account class encapsulates the account data which is this balance attribute and its related methods deposit and withdraw into a single unit uh of this bank account class the data members so balance are marked as protected encapsulating them within the class and preventing direct access from outside of the class a geta property method called balance um is provided uh is used to provide controlled access to the protected balance data and methods deposit and withdraw are used to manipulate balance ensuring that operations are performed safely and according to the business rule tools of our program so what we've demonstrated here is how to create an instance of the bank account class and interact with its properties and methods without needing to know the internal implementation details so here we have uh a sort of user of this bank account class so this can be you know other developers uh maybe use this class or other classes in our program other files in our in our program or software using this bank account class uh they can't directly access the balance field as it's marked as protected uh so this data is incapsulated within the class and the methods dictate the rules uh for how this data can be accessed and modified ensuring that our programs rules can't be violated by users or consumers of the bank account class for example it's no longer possible to withdraw more money than is in the account and encapsulation of the logic inside of the methods in bank accounts also means that users don't have to worry about the implementation details when interacting with a bank account object for example the user doesn't have to worry about the logic involved in withdrawing money they can just call uh account. withdraw and then Prov pass the amount that they want to withdraw and the implementation details are hidden and encapsulated in the class okay so they're all here they don't have we don't have to repeat this uh this logic or any users of a bank account don't have to worry about the sort of complex details uh within uh withdrawing money from the account okay and if the user tries to do something silly uh like uh deposit a negative amount the program will throw an error and the user will be notified so so encapsulation of logic within methods in the bank account class allows users to interact with a bank account object without needing to know or understand the

internal implementation details of how withdrawals deposits or other operations are carried out uh so users of the bank account class can interact with bank accounts using a very simple uh intuitive API or intuitive methods like withdraw or deposit without needing to to understand the complex logic behind these operations so encapsulation abstracts away the complexity of the implementation details allowing users to focus on the higher level functionality provided by the bank account class users only need to know about the public interface of the bank account class in other words it's public methods or properties to use it effectively while the internal implementation details remain hidden so in summary in capture ation allows for Clear separation between the public interface and the internal implementation details of a class providing users with a simplified and intuitive way to interact with objects while hiding the complexity of how those interactions are handled internally the aim of abstraction is to reduce complexity by hiding unnecessary details so for example when you press a button on a TV remote you don't have to worry about or interact directly with the internal circuit board within the remote those details those complex details are abstracted away so let's go through an example of abstractions so let's create an email service class whose job is to basically send emails so to send an email we have to first connect to the email server then we have to authenticate uh then we send the email and then we disconnect so let's create a few methods for uh sending an email so we'll first create a method called connect and this is going to be a protected method and I'll explain why shortly so let's just print uh connecting to email server and we don't have to actually connect to an email server we're just going to print connecting to email server this is just a for demonstration purposes then we can create a method called authenticate and again we can just print so let's just print authenticating authenticating and finally uh we need a method for sending an email so let's create a method a method called send email mail and this needs to be a public method because uh uh users of this email service class are only really interested in sending emails they're not too interested in connecting to the email server or or authenticating those are implementation details so details that are kind of specific to this class but you know outside of the class no one really cares about those complex details so we create a public method called send email and what this is going to do is it's first going to connect to the email server so it can call first of all self. connect then we can call self. authenticate uh and then we can send the email so let's just print sending email and then we need to disconnect from the email s uh server so let's create a method called disconnect and this can also be a protected method because it's only needs to be used within the class uh other classes or other parts of the program are not concerned about this it's an implementation detail for this class and we can say disconnecting from email server and let's now call that uh method inside of our send email uh method okay so let's actually put this public method at uh actually we'll leave it here so we connect and let's see what's going on here what did we do wrong there we go okay so first we connect authenticate send the email and then dis connect okay that's all good so the user of this class can now send emails without any knowledge of the internal implementation details involved in sending an email so all of these details have been abstracted away and life is much simpler for users of this email service class so for example if we create uh a new email service object and we now say email Dot so let's look at the methods available on this email service object so we can see that we have some underscored methods so as a python

developer I know that I should not be touching these methods I know that they are kind of to be used internally within the class but outside of the class I don't need to worry about them so I can see here I can look down and I see a public method here so no underscore that says send email so I know as a user of this class that I can just call the send email uh method and the email will be sent I don't have to worry about how emails are sent I can just call send email so life is really really simple and without abstraction the user would have more decisions to make as they are exposed to more information and complexity than is necessary to perform a task and uh the user would have to write more complex code so let's just for example change these uh these protected methods to uh public so if we get rid of the underscore okay and let's just comment uh these out for now so send email just sends the email but before the send email method is sent we would have to first uh connect to the server then we would need to authenticate and then after sending the email we would need to disconnect from the email server and you can see life is much more complicated for any users of this class now because we have to understand how emails are sent call all these these methods in the specific order correct specific order and we have to repeat this code every single time that we want to send an email so this is would be an awful uh solution really uh with no obstru abration whereas if we abstract all of those uh details away all of those implementation details uh related to sending an email then life is nice and simple for any users of this class okay so if the methods are protected by being prefixed with an underscore it's going to communicate to the python developer that these methods are for internal use within the class and don't need to be touched with it uh to use the class as intended and importantly by using encapsulation if any of these protected methods are changed for example they take another parameter for example let's say if we authenticate we need to pass say I don't know a uh username and password for example currently we don't but let's just say uh we did uh at some point in the future then only this email service class has to change we just need to pass uh these values into the authenticate method here okay uh whereas if um without abstraction if we had to actually call this authenticate method outside of the class and all of the users of this class would have to change okay so it it could potentially you know we' have to make a lot of changes in our code and it would be very easy to introduce books so um uh by using abstra abstraction we can change the implementation details of email service without it affecting other classes or other parts of our application so it may be difficult to see the differences between encapsulation and abstraction as encapsulation is often used to to support abstraction by hiding implementation details from users so let's just have a look at the differences between the two so encapsulation focuses on bundling data or attributes and methods that operate on that data into a single unit called the class and it restricts access to the internal implementation details so this is achieved by defining attributes and methods as private or protected and exposing only a control interface EG uh you know in other words public methods uh for example by marking connect authenticate and disconnect as protected hides their details from the user of the email class whereas abstraction on the other hand focuses on hiding complexity by providing a simplified highlevel interface to interact with so just by providing the simple send email uh method we are effectively concealing the underlying ing implementation of sending an email so it allows users to focus on what an object does rather than how it does it so for example the

send email method as we mentioned abstracts away those multiple internal steps uh required to send an email providing a nice and simple interface for the user so just to clarify even further encapsulation and emphasizes bundling and restricting access while abstraction focuses more on simplifying usage by hiding unnecessary details so and it's also important to note that encapsulation is a mechanism that enables abstraction but they are still distinct uh Concepts inheritance is a fundamental Concept in objectoriented programming that involves creating new classes based on existing classes so subclasses inherit properties and behaviors from their super classes and can also add new features or override existing ones and inheritance is often described in terms of an is a relationship so for example uh a car is a vehicle okay and a bike is a vehicle okay so this demonstrates uh that inheritance uh kind of uh resembles or um kind of models this Isa relationship between classes okay so let's go through an example to show you uh how inheritance works so first of all we're going to create a base class or a super class representing a vehicle so we can say class vehicle and we can set up some attributes in the anit method for each object so we say self uh let's say each vehicle needs to have a brand a model and the year it was made so we can set those up here so we create a brand attribute assign it to the value passed in same for model and the same for the year and let's just say that every vehicle should have a St method so it should have the ability uh to start the vehicle so let's just print keep things nice and simple and we're just going to say vehicle is starting and we can also provide a stop method so every vehicle should have a stop method and we can just say vehicle is stopping okay now all specific Vehicles such as cars bikes or planes can inherit this common Behavior this common vehicle behavior and also these common vehicle attributes uh from this uh vehicle kind of parent class so now let's create a few subclasses so first of all let's create a core so we're going to say class core and what we can do is we can uh say that car should inherit V vehicle okay so it should inherit all of these attributes and also these uh two methods and the way we do that in Python is we add some parentheses and we specify the class that this uh core class should inherit okay so now what we can do is we can also add some extra um some extra attributes that are specific to a car vehicle but not necessarily uh other vehicles so for example what we can do is in the net method of car we can pass in the uh values that we need for vehicle so we need to pass in brand model and year because car inherits vehicle meaning that it needs to have these brand model and year attributes so if we paste those in but also let's just say a car should also have an attribute for the number of doors that it has and also the number of wheels for example okay so the way that so what we need to do here is we actually need to call the init method of its parent class so this vehicle class we do that by calling the super uh ma uh function so super is essentially refer referring to the super class of car which is vehicle okay and then we can call super. init so we call the init method on vehicle and we can now pass in the values uh received by the core object so we can pass in brand the model and the year okay and then we can uh also create some attributes on the core class the core object itself so we can create a self. number of doors and I've spelled this wrong and we can also have uh self. number of Wheels like so okay okay so let's now create a subass representing a bike which also inherits from vehicle okay so let's create a bike class class bike and this is going to inherit vehicle so it's going to that means it's going to have these attributes and also it's going to inherit uh these methods okay so let's create our init

method because we need to pass uh the uh brand model and year into the Super class to the vehicle superclass okay so we need to set those up in that class so in here we can have again the brand model and the year and let's just say a bike only has one kind of uh attribute contained within its class uh called number of Wheels uh so again we can set up the attributes for the super class by calling super so super in this case is going to refer to vehicle and we can call init so we need to call the init method on the vehicle class and pass in the brand model and year for this particular bike uh object that is been created so we can pass in the brand model and year so let's just copy and paste in those and we also need to create a new attribute on the bik class itself called number of wheels so self. number of Wheels equals number of Wheels okay okay so let's actually create a few vehicles from these classes so first of all let's create a car class and in here we have to pass a brand model the year the car was made the uh number of doors so you can see here these are the kind of inherited attributes that we have to pass so these come from the vehicle uh sort of super class or the parent class and now we have the cars classes so the specific the most specific car class uh has its own attributes that we now have to also pass so you can see here we passed those to the parent class and now and we set these uh attributes in the color class itself so in here we need to pass the number of doors so let's say this is a five door and the number of Wheels which is four okay and let's also create a bike so let's create a Honda Scoopy and let's say the year is 2018 and the number of Wheels is two it's kind of a silly attribute I guess but uh I guess it's possible to have different numbers of wheels on different bikes uh anyway let's print uh what this car uh class looks like so a car object looks like so we can do that by saying car do dictionary so you can see this is an attribute on uh objects okay and let's just run this uh program to see what we get so you can see by uh printing uh car uh so double uncore dict this is kind of like dictionary it kind of gives us back a sort of dictionary for all of the uh attributes that this object has so if we call this kind of dict uh attribute which is available on all objects in Python it it gives us back a kind of uh dictionary with all of the attributes on this object and we can do the same for the bike okay and we can see all of the attributes that we have on the uh bike object okay so as you can see uh just to kind of summarize what we've got here we don't have to write the commonly used uh fields and methods uh for every single type of vehicle we can just inherit them in from the vehicle class so we only have to write them in one place if we didn't use inheritance here we would have to create a start and stop method on car and we would also have to you know repeat all of these attributes that are shared by all vehicles in every single type of vehicle uh class um so now also the advantage of this is that if we want to change the uh start method then we only have to change it in one place in this vehicle class Okay so just to show you uh another thing uh so if we call the start method on car we we also have this start method on bike as well so you can see vehicle is starting and vehicle is starting so that is inheritance and inheritance also allows for a very important objectoriented programming concept called polymorphism which we will look at next the word polymorphism is derived from Greek and means to have multiple forms so poly meaning many and morph meaning forms and in programming polymorphism is the ability of an object to take many forms so first of all I'm going to show you an example that has no polymorphism so here we have a car class with an init method where we set up some attributes that a car should have so we have the brand the model the year and the

number of doors and we also have a couple of methods to start a car and stop a car okay and we also have a motorcycle class which also has its own attributes so the brand model and the year and it also has a couple of methods for starting and stopping a bike okay or motorcycle so let's say that we want to first of all create a list of vehicles uh to uh inspect so we're going to Loop through this list of vehicles and inspect them so uh the inspection can involve let's say starting and stopping the vehicle so first of all let's create a sort of list of vehicles so we can create first of all a car so let's create a Ford Focus and also let's add a motorcycle to the list okay and now we need to Loop through these so let me just make a comment here so create list of vehicles to inspect and now we're going to Loop through a list of vehicles and inspect them so we can do this in Python by saying for uh vehicle in the vehicles list now in here we need to actually check uh what the type of object is so the thing what the problem we have here is that this vehicle's list is actually made up of different types of Vehicles we have a car uh we have car objects and we have motorcycle objects okay so uh we don't really have like a common uh class to work with here this we have to kind of deal with separate uh types of objects okay so for example if I look at a vehicle and let's say I want to look at uh I want to start uh start the vehicle there are actually no com methods in this uh these objects in these classes right there's nothing to enforce these uh vehicle objects to actually have a start method for example in car we have a start a method called start and in motorcycle we have a method called start bike so there's there's nothing in our program that kind of gives us any certainty that uh the developer has kind of kept these methods cons consistent right so this is an issue um and that can kind of be solved in this case by checking what um uh what type of uh object that we are currently working with so for example if we want to check if this is a uh car object we can use is instance which means is an instance of the uh of the class so we can see is in is vehicle in this case an instance of the car class and if it is then we know that uh this is a car which means it will have all of the attributes and uh methods available on car okay so we can safely uh access those attributes and methods so let's just create a uh print here so inspecting uh vehicle. brand and then let's also log the type of the vehicle okay which we will know as a uh car object and we just want the name here actually and then we also want to check if we are dealing with a bike so we can say is the current vehicle that we're dealing with in this Loop uh of type or is an instance of the motorcycle uh class and if it is we will do an inspection so so we can print the uh vehicle brand the model and the type and also what I want to do is actually start and stop the vehicle so we can say vehicle do uh start so you can see here that python knows that we are dealing now with a car object because we've checked that it's a car object and python is sort of it's kind of known I think as type scoping where python knows that within this if conditional block we are dealing with a car object because we have checked and ensured that it's a car object here so you can see uh vs code now is providing us uh with all of the uh methods and attributes that are available on core objects so we can call the start method and we can call the uh stop method uh and then let's also do the same for motorcycles so we can say vehicle dot you can see on motorcycle we actually gave the method a kind of different name so start bike and vehicle dot stop bike okay and we could also just to kind of make our program a little bit more uh safe I guess we could throw or raise an exception if there is an object in this list that is an invalid vehicle okay so if it's not a car or a motorcycle we can throw an error so we can say else uh raise

exception uh object is not a valid vehicle okay so let's run this program to see what happens and we can see that we first inspect the Ford Focus which is of type uh which is made of the class name car and then we inspect the Honda the motorcycle which is made from the motorcycle class and we start and stop the motorcycle so you can notice this really ugly code inside of this for Loop and the reason for this is because Vehicles this vehicle's list is a list of any type of objects we can put anything in this list and so we have to figure out when we Loop through this uh list of objects uh what type of object that we are currently dealing with inside of each Loop before we can access any information uh on the object or at least access any information uh safely okay because there's nothing to kind of enforce um there's no kind of consistency enforced between motorcycle and car we can kind of call you know these methods whatever we want we can call these attributes whatever we want and there's nothing kind of enforced or you know shared or consistent between the these two objects so that means we have to check what type of objects they are before we can safely access the attributes and methods on those objects and another issue with this is that that this code will continue to get uglier with more uh if conditional statements as we add more vehicle types so for example if we extended our code base to include a new uh plane class so like an airplane then we'd need to modify this existing uh code that we have here we'd have to add another um conditional check in this for Loop to check if the current vehicle that we are dealing with is an instance of the plane class so we can solve this issue that we have with polymorphism so cars and vehicles sorry cars and motorcycles are both vehicles uh so they both share some common properties and methods so let's create create a parent class that contains these shared properties and methods so we're going to create a kind of parent uh vehicle uh class or a sort of super class for vehicles okay so we'll call this uh class vehicle and it's going to uh have an init method where we initialize some attributes so each vehicle no matter whether it's a car motorcycle plane boat should have a brand a model and a year and we can set those up okay and also each vehicle should have the ability to start or stop so let's provide some methods to start and stop each type of vehicle okay so car and motorcycle can now both inherit these attributes and these methods from the vehicle class so if you can remember from before to inherit we just have some parentheses after the class name and add in the class that we want to inherit from and also if you can remember from before we can access the uh super class by calling super and then we can access the anit method on this super class and pass in the uh values that we need to pass to the superclass so brand model and year are all shared attributes for each vehicle and then you can see that we have number of doors specific only to this car class okay and we can do the same for uh motorcycle so let's go ahead and do that and you can see that motorcycle actually has no uh none of its own kind of specific um um attributes okay so let's see what is the issue here here so we actually need to inherit the vehicle class here okay so let's actually have a look to see if all of this still works so you can see all of this is still working um however uh I think what we'll do actually now is just comment out these methods for now so we're just going to inherit these start and stop methods from the vehicle class for now so let's comment those go out and we're actually going to recreate our inspection logic because this is pretty uh messy with all these if conditional statements so for vehicle uh for vehicle in vehicles so now we know that all vehicles inherit this vehicle class okay so we can actually now treat all specific Vehicles just

as a type of type vehicle okay so we don't have to worry about the specific type of vehicle to perform an inspection on each vehicle so all we now need to do is just check that it's a type uh is an instance of vehicle so we can say if is instance uh the current object if it's a instance of the vehicle super class then we can just perform our inspection so I'm just going to copy and paste in the inspection Logic for now so you can see here we're doing basically what we were doing before so we inspect the brand the model and then we log the type of uh class that this vehicle is made from we then start and stop the vehicle so now if we've run our program we can see that we got vehicle is starting and vehicle is stopping and then we inspect the motorcycle which is of type motorcycle so it's made from the motorcycle class and we got vehicle is starting and vehicle is stopping but let's say that we wanted to provide more kind of specific methods for starting and stopping each vehicle well what we can actually do is we can override the methods from the vehicle superclass within each specific class so in car we inherit these start and stop methods but we can actually override them so kind of provide more specific implementations within the car class itself and we can also do the same thing for a bike okay but we need to actually uh give it the same name as the method that we are inheriting to actually override it so let's run the program now to see what we get and we can see now we get a more specific uh uh log to the console okay so you might be wondering um that about car and motorcycle uh are now both extending this vehicle class as they are both vehicles but you might be wondering what's the point in car and motorcycle both extending vehicle if they are going to implement their own versions of the start and stop method and the reason for that is because we can now uh treat um all of these uh objects as Vehicles whether it's a car or or a motorcycle we can now treat them in the same way we don't have to worry about each specific vehicle we can just treat them as Vehicles okay which means that they all share we know it's kind of guaranteed that each vehicle is going to have these set of attributes and this set of methods okay so despite the vehicles being of different types polymorphism allows us to treat them all as instances of the base vehicle class and the specific implementations of the start and stop methods for each vehicle type are invoked dynamically within this uh loop okay so what we can also do here in Python we can use uh kind of type hinting so we can say that vehicle is a list uh sorry Vehicles it needs to be a list of vehicle objects okay so now this is going to kind of further enforce that uh Vehicles has to be a uh list of vehicles okay so now what we can actually do is remove all of this code and because python knows that this vehicles list can only contain Vehicles we can actually just have our inspection logic so if we uncomment this and okay so you can see py actually knows um that these methods and attributes exist because we are saying that Vehicles is a list of vehicle objects and that therefore it's guaranteed that they have a brand a model and a start and start method so if we now run the code we can see we get everything as before without all of these uh conditional statements um so because this list can now only contain objects that extend the vehicle class we know that every object will share these common fields and methods meaning that we can safely call them without having to worry about whether each specific vehicle has these fields or methods so this demonstrates how polymorphism enables code to be written in a more generic and flexible manner allowing for easy extension and maintenance as new types of vehicles are added to the system so for example if we wanted to add another type of vehicle we don't have to modify the code

used to inspect vehicles or the client code right we can just extend our code base so add another class to our codebase without having to modify the existing inspection logic so let me just show you another uh example here so if we actually add another type of uh vehicle so let's say we add a plane class okay so here I've just pasted in a plane class which extends the vehicle class we then set up the brand model here and planes also like cars have a more specific attribute for the number of doors and then we override these start and stop methods from the vehicle superclass now let's say in our list we add a plane okay and let's see um this can be Boeing uh let's see 747 and the year can be 2015 and the number of doors let's say is 16 okay so you can see we're getting no errors in the program so far and if we run the program we can see inspecting the Boe 747 which is of type plane and then we start and stop the plane you can see that we haven't actually had to edit any of our current inspection logic whereas beforehand we would have to add another uh if statement for if to check if the uh current object current vehicle object that in the list is of type plane so the code to perform the vehicle inspection logic doesn't have to change to account for a plane everything still works without having to modify our inspection logic congratulations on completing this course you now have all of the tools that you need to create readable maintainable flexible top quality software and this will save you lots of time and headaches throughout your life as a programmer allow you to work can collaborate more effectively in a team environment and lend you more interesting better paying jobs on larger projects so I hope that you found this course useful and if you'd like to take your object-oriented programming skills to an advanced level then you can check out my full course on udemy or the book version on Amazon the links are in the description below and if you did find this course useful and you're somehow not sick of me and you'd like to hear more from me then you could subscribe to my YouTube channel at doable Danny so thank you very much for watching and I will see you in the next one

7 | Object-Oriented Programming

In a previous Chapter, I defined programming as “storing data and doing stuff with the data”. This definition is not the most technical and detailed definition you’ll read, but it’s still a very good one. You’ve learned about both parts of this definition. You can store data using various data types, and you can perform actions with the data, primarily by using

functions. Object-oriented programming merges these two into a single item.

I'll get back to this merger a bit later. This topic brings along with it some new terminology, too. You'll soon read about classes, objects, instances, attributes, and a few more too. As I've done previously in this book, I'll make sure to explain the concepts behind these terms as clearly as possible.

Indeed, you've already come across the term class. You've been using classes since your first program in Chapter 1. Have a look at the following variables:

```
>>> number = 10
>>> name = "Stephen"
>>> shopping_list = ["Coffee", "Bread", "Butter", "Tomatoes"]
>>> type(number)
<class 'int'>
>>> type(name)
<class 'str'>
>>> type(shopping_list)
<class 'list'>
```

When you use the type() built-in function to display an object's data type, you're told that this is of class 'int' or class 'str' or whatever the name of the data type may be.

If you reread the previous sentence, you'll also notice the use of the word 'object'. You can see you've also been using objects all the time! All will be clear by the end of this Chapter.

The Object-Oriented Programming Paradigm

First, let's have a quick word about philosophy. Object-oriented programming is one of several programming paradigms or styles of programming. Whereas the tools you have learned about so far are unavoidable in most modern programming, object-oriented programming is a style of programming that you can use if you wish. However, there are some applications where it may be hard to avoid this paradigm.

If you search on the internet–a dangerous place to be most of the time–you'll find a vast range of views about object-oriented programming, or OOP. Some love it. Others hate it. But if you don't get dragged into these philosophical debates about programming, you'll find that object-oriented programming is a great tool to have in your toolbox that you can choose to use on some occasions and avoid in other cases.

Python is inherently an object-oriented programming language. However, you'll often see Python described as a multi-paradigm language. This means that it's a language that supports many different programming styles, including object-oriented programming.

Meet The Market Seller

That's enough about the theory of object-oriented programming for the time being. Let's

work our way towards what object-oriented programming is.

In this Chapter, you'll use the example of a market seller who's been learning Python and who decides to write code to help him deal with his daily operations of selling items at the market stall. You've already met the market seller in Chapter 5 when you read about errors and bugs.

He has four items on sale in this small stall. His first attempt at writing this code uses a number of lists:

```
items = ["Coffee", "Tea", "Chocolate", "Sandwich"]
cost_price = [1.1, 0.5, 0.9, 1.7]
selling_price = [2.5, 1.5, 1.75, 3.5]
stock = [30, 50, 35, 17]
```

His next section prompts the user to enter the item sold and the quantity:

```
items = ["Coffee", "Tea", "Chocolate", "Sandwich"]
cost_price = [1.1, 0.5, 0.9, 1.7]
selling_price = [2.5, 1.5, 1.75, 3.5]
stock = [30, 50, 35, 17]
# Input items and quantity sold
item_sold = input("Enter item sold: ").title()
quantity = int(input("Enter quantity sold: "))
item_index = items.index(item_sold)
```

In this code, you'll note that you used the string method title() directly after the input() function. This is possible because input() returns a string. The title() method converts a string into title case. Try to run "this is a string".title() in the Console to see what's the output.

You've already seen the need for changing the string returned from input() to a numeric format when you get the value for the quantity sold. You've used the built-in function int() with the input() function as its argument.

The list method index() returns the index that matches the item in the list. You can use this method to find the location of a specific item in the list.

The += and -= Operators

Let's take a small detour before returning to the market seller's first attempt at this code.

A common requirement in a program is to increment the value stored within a variable. For example, if you want to add to the score in a game, you can write:

```
>>> score = 0
>>> score = score + 1
>>> score
```

1

There’s a shortcut in Python for this operation which makes the code easier to write and more succinct:

```
>>> score = 0
>>> score += 1
>>> score
1
```

The += operator increments the value of the variable by the amount which follows the operator. The above code snippet is identical to the one before it.

You can also decrease the value in a similar manner:

```
>>> score = 0
>>> score -= 5
>>> score
-5
```

Other arithmetic operators work in the same way. You may want to experiment with *= and /= as well, for example.

Market Seller’s First Attempt

Let’s look at the next section in the market seller’s first attempt at writing the code:

```
items = ["Coffee", "Tea", "Chocolate", "Sandwich"]
cost_price = [1.1, 0.5, 0.9, 1.7]
selling_price = [2.5, 1.5, 1.75, 3.5]
stock = [30, 50, 35, 17]
daily_income = 0
# Input items and quantity sold
item_sold = input("Enter item sold: ").title()
quantity = int(input("Enter quantity sold: "))
item_index = items.index(item_sold)
# Work out required values
profit = quantity * (selling_price[item_index] - cost_price[item_index])
daily_income += selling_price[item_index] * quantity
# Update stock
stock[item_index] -= quantity
# TODO Add check to make sure stock does not go below zero
print(profit)
print(daily_income)
print(stock[item_index])
```

This works. The section headed with the comment ‘Work out required values’ works out the profit by subtracting the item’s cost price from its selling price and multiplying that by the

quantity sold. The next line increases the income for the day using the increment operator += which takes the current value of daily_income and adds the income from this sale to it.

The following section, which has the heading ‘Update stock’, updates the number of items in stock by using the decrement operator. The market seller also added a comment to remind him to add a bit more code later to check that the stock doesn’t go below zero. Adding such a comment is a common technique to leave notes in your code for later on. Most IDEs also interpret the # TODO comments differently from other comments, and the IDE will show you a to-do list using these comments.

You can see the output from the test the market seller ran:

```
Enter item sold: chocolate
Enter quantity sold: 7
5.95
12.25
28
```

But the market seller stopped at this point as he realised that this might become quite cumbersome to deal with. Every operation will need to carefully reference the correct item in the correct list using the index. As the market seller adds more products and more information about each product, this method can quickly get out of hand.

Data that belong together

The code above stores the item’s name, cost price, selling price, and stock quantity in separate variables. You know that these separate storage boxes “belong together” and that each item in each list is related to the other items occupying the same position in the other lists.

However, the computer program doesn’t know that these data are related. Therefore, you need to ensure you make all of those links in the code. This style can make the code less readable and more prone to errors and bugs.

The Market Seller’s Second Attempt

The market seller noticed the problem with having many lists which store the various attributes linked to each product. So, he decided to refactor his code and use dictionaries instead. Refactoring is the process of changing the code to make it neater and better without changing the overall behaviour of the code. Here’s the market seller’s refactored second attempt:

```
# Create dictionary with item names as keys and a list of numbers as the values. The
# numbers in the lists refer to the cost price, selling price, and quantity in stock
# respectively
items = {
"Coffee": [1.1, 2.5, 30],
```

```
"Tea": [0.5, 1.5, 50],
"Chocolate": [0.9, 1.75, 35],
"Sandwich": [1.7, 3.5, 17],
}
daily_income = 0
# Input items and quantity sold
item_sold = input("Enter item sold: ").title()
quantity = int(input("Enter quantity sold: "))
# Work out required values
profit = quantity * (items[item_sold][1] - items[item_sold][0])
daily_income += items[item_sold][1] * quantity
# Update stock
items[item_sold][2] -= quantity
# TODO Add check to make sure stock does not go below zero
print(profit)
print(daily_income)
print(items[item_sold][2])
```

You're now storing the data in a single dictionary instead of several lists. The keys in the dictionary are strings with the names of the items. The value for each key is a list. Each list contains the cost price, selling price, and the number of items in stock in that order.

You may have already spotted one drawback. You need to reference the data in the lists using the index. Look at the following line as an example:
`profit = quantity * (items[item_sold][1] - items[item_sold][0])`

You first need to use one of the keys to extract a value from the dictionary items. The variable item_sold is the string that contains the name of the item.

items[item_sold] refers to the value of the item_sold key, which is a list. You're then indexing this to get one of the numerical values in the list. Therefore, items[item_sold][1] refers to the selling price of the item represented by the string item_sold. This makes the code harder to read and, for the same reason, more prone to errors.

Using a single data structure to store all the data has its advantages. Adding a new item or removing one that's no longer sold is easier with the dictionary version than with the list approach. However, as the data structure becomes more complex, accessing items can also become trickier, leading to code that's harder to write, read, and maintain.

Adding Functions To Perform The Tasks

Before looking at yet another way of writing this code, you can add some functions to the code you have so far:

```
# Create dictionary with item names as keys, and a list of numbers as value. The
# numbers in the lists refer to the cost price, selling price, and quantity in stock
```

```
# respectively
items = {
"Coffee": [1.1, 2.5, 30],
"Tea": [0.5, 1.5, 50],
"Chocolate": [0.9, 1.75, 35],
"Sandwich": [1.7, 3.5, 17],
}
daily_income = 0
def get_sale_info():
item_sold = input("Enter item sold: ").title()
quantity = int(input("Enter quantity sold: "))
return item_sold, quantity
def get_profit(item_sold, quantity):
return quantity * (items[item_sold][1] - items[item_sold][0])
def update_income(item_sold, quantity):
return daily_income + items[item_sold][1] * quantity
def update_stock(item_sold, quantity):
items[item_sold][2] -= quantity
# TODO Add check to make sure stock does not go below zero
# Call functions
item_sold, quantity = get_sale_info()
profit = get_profit(item_sold, quantity)
daily_income = update_income(item_sold, quantity)
update_stock(item_sold, quantity)
print(profit)
print(daily_income)
print(items[item_sold][2])
```

This code now puts all the functionality of the code into standalone functions. You'll recall the definition of programming I started the Chapter with. This code now clearly has the data stored in one variable and all the tasks that need to happen are defined as functions.

Let's look at these functions a bit closer. The first one is get_sale_info(). This function asks the user to type in the product sold and how many of it were sold in a particular sale. The function returns a tuple with the values stored in item_sold and quantity. You'll recall that the parentheses are not needed to create a tuple, so the variable names in the return statement are not enclosed in brackets.

The functions get_profit(), update_income(), and update_stock() all have two parameters. When you call these functions, the name of the item sold and the quantity sold need to be passed to the function.

Accessing global variables in functions

These three functions also use the data stored in the variable items. And update_income()

also uses the data in daily_income. Functions have access to the global variables of the scope that's calling them. This means that when Monty, the computer program, goes to a Function Room, he can still use any of the boxes which are on the White Room's shelves.

However, you can only ever use these functions with these variables. So, you may want to define the functions so that all the data needed is passed in as an argument and then returned when the function finishes its tasks. However, I won't make this change to this code as instead, I'll discuss the third option.

Changing the state of an object

You can notice another difference in the behaviour of these functions. The update_stock() function doesn't return anything as its only job is to change a value directly within the dictionary items. Dictionaries are mutable data types, and therefore this is valid. You may hear this referred to as changing the state of the object items.

The other functions do not make any changes to existing variables directly. Instead, they return the new values, which you can then assign to variables when you call the functions, as you do in the section labelled 'Call functions' in the code.

In this Chapter and the later Chapter on Functional Programming, you'll learn more about functions that change existing objects' state and others that do not.

Object-Oriented Programming

We've gone a long way in this Chapter without having written any object-oriented code. Let's redress this now that you're familiar with the problems the market seller has encountered.

When thinking with an object-oriented programming mindset, the starting point is to think of the main objects relevant to the problem you're solving. In a way, you need to think of the problem from a human being's perspective first and design your solution accordingly.

Let's see what this means from the market seller's perspective. What are the objects that are relevant for the market seller? The objects that matter to him are the four products he sells. Although coffee, tea, chocolate, and sandwiches are different products with different prices and so on, they all share similar characteristics. They are all products that the market seller sells.

You can therefore start by creating a class of objects in Python that describes these products.

Creating A Class

To create a class in Python, you can use the class keyword. You can create a new Python file called market_stall.py where you'll create your classes. If you choose a different file name, make sure you don't use any spaces in the file name:

class Product:

The convention is to use a capital letter for the name of a class. More generally, you should use upper camel case, such as MyInterestingClass.

When you create a class, you're creating your own data type, one that's relevant for the application you're writing. The class definition that you'll be writing in the following sections acts as a template or a blueprint to create objects that belong to this class of objects.

You'll see how to create an object of type Product soon, but first, we need to add a function definition within the class definition. A function that's a member of a class is called a method. Methods are functions, and therefore they behave in the same way as functions:

```
class Product:
def __init__(self):
self.name = "Coffee"
self.cost_price = 1.1
self.selling_price = 2.5
self.number_in_stock = 30
```

There's a lot to understand in this code. I'll explain all of the strange new additions in the following paragraphs.

The __init__() method

The __init__() method is a special function, and it's often the first part of a class definition. This method tells the program how to initialise a Product when you create one. When you create an object of type Product, the initial state of this object will be determined based on what happens in the __init__() method. Hint: if you just start typing "init" in your IDE, the underscores and the parameter self, which you'll learn about soon, will autocomplete.

Methods whose names start and end with double underscores have a special status and are often referred to as dunder methods. Dunder is a contraction of double underscore. Sometimes they're also called magic methods. However, there's nothing magical about them, so some prefer not to use that term.

You'll have noticed another new term used in the __init__() method: self. I'll discuss self shortly as you'll see it a lot in class definitions.

Creating An Instance of A Class

When you defined the class in market_stall.py, you've created a template for making products, but you haven't created any products yet. Let's create an instance of the class Product in the Python Console instead of in the script market_stall.py.

When you start a Console session, you're creating a new White Room, which is separate from the one created when you run the market_stall.py script. Therefore, the Console's

White Room doesn’t know about the class Product yet. However, you can import your script in the same way you’ve imported other modules previously:
>>> import market_stall
>>> market_stall
<module 'market_stall' from '/<path>/market_stall.py'>

When you import a module, you’re importing the contents of a Python script. This script could be one written by someone else or one you’ve written yourself. You can see that the name market_stall in the Console session refers to the module, and it references the file. As with any module you import, you can now access anything from within this module. In this case, you can access the class Product.

You can create an instance of the class Product as follows:
>>> market_stall.Product()
<market_stall.Product object at 0x7f9178d61b20>

You can create an instance of a class by using the class name followed by parentheses. The output doesn’t tell you much except that you’ve created an object of type Product which is a part of the market_stall module. The memory address is also displayed, but you won’t need this.

You can assign this instance of the class to a variable name:
>>> my_product = market_stall.Product()
>>> my_product.name
'Coffee'
>>> my_product.selling_price
2.5

The name my_product now refers to the object you have created. When an instance of the class is created, the __init__() method is called. This means that every Product will always have a variable called name attached to it, and another one called selling_price. You can see the values of these instance variables displayed above. You can even access the other instance variables you have defined in the __init__() method in the same way.
Making the class more flexible

The problem with the code you have is that each product you create can only be coffee with the same cost price, selling price, and quantity in stock. You’d like your class to be more general than this so that you can create other products as well.

You can go back to market_stall.py and make some changes:
# market_stall.py
class Product:
def __init__(self, name, cost_price, selling_price, number_in_stock):
self.name = name

```
self.cost_price = cost_price
self.selling_price = selling_price
self.number_in_stock = number_in_stock
```

Like any other function, __init__() can have parameters defined in its signature. The mysterious self was already there, but you've now added more parameters.

The arguments passed to the __init__() method are then attached to the instance. In the next section, we'll walk through what happens when you create an instance again, but this time you'll have to pass arguments when you create the instance.

Creating a new file to test the class definition

Rather than carrying on in the Console, you can now create a second file. Let's call this second script market_seller_testing.py. When dealing with object-oriented programming, it's common practice to define the classes in one file, or module, and then use these classes in other files:

```
# market_seller_testing.py
from market_stall import Product
first_prod = Product("Coffee", 1.1, 2.5, 30)
second_prod = Product("Chocolate", 0.9, 1.75, 35)
print(first_prod.name)
print(second_prod.name)
```

I've included the file name as a comment at the top of each code block since you'll be working on two separate files from now on in this Chapter.

You've also used the import keyword differently from previous times. Instead of importing the whole module, you're now only importing one object from within that module. In this case, you're just bringing in the Product class and not the entire module. You can now use the class name Product without having to add market_stall and a dot before it.

You created two instances of the class Product. The arguments you use within the parentheses are passed to the class's __init__() method. The objects first_prod and second_prod are both of type Product, and therefore they have been created using the same template. Both of them have instance variables called name, cost_price, selling_price, and number_in_stock. However, the values of these instance variables are different, as the output from the print() lines show:

```
Coffee
Chocolate
```

You can also confirm the data type of these two objects you have created:

```
# market_seller_testing.py
from market_stall import Product
```

```
first_prod = Product("Coffee", 1.1, 2.5, 30)
second_prod = Product("Chocolate", 0.9, 1.75, 35)
print(type(first_prod))
print(type(second_prod))
```

The output now shows that both objects are of type Product:

```
<class 'market_stall.Product'>
<class 'market_stall.Product'>
```

When you create a class you're creating a new data type, one that you have designed to suit your specific needs.

Viewing two scripts side-by-side

A small practical note: In most IDEs, you can split your window into multiple parts so that you can view two files side-by-side. Since you're defining your class in one file and testing it in a different file, this is a perfect time to explore some options in your IDE.

If you're using PyCharm, you can look in the Window menu and choose the Editor Tabs option. You'll find the option to split your screen there. You can also make sure that any sidebars on the left-hand side are closed so that your screen is split between the two files.

Split screen setup in PyCharm IDE for object-oriented programming

If you're using other IDEs, you'll be able to find similar functionality, too.

Understanding self

In the definition of the __init__() method, you've used the keyword self several times. Let's start by looking at its use inside the function definition first. Here are the four lines in this definition that all reference self:

```
# ...
self.name = name
self.cost_price = cost_price
self.selling_price = selling_price
self.number_in_stock = number_in_stock
```

The keyword self refers to the instance of the class that you'll create when you use the class. You read earlier that the class definition doesn't create any objects itself but serves as a blueprint for creating objects elsewhere in the code. The name self refers to the future name of the variable that the object will have.

For example, in market_seller_testing.py, you created two variables named first_prod and second_prod. These variables each store a different instance of the class Product. You can think of the term self in the class definition as a placeholder for these names.

Therefore, the above lines provide the instructions for the program to create four instance variables when it creates a new object. These lines are assignment statements, as shown by the = operator. The variables you're creating are attached to the instance. The dot between self and the variable name shows this link.

An instance variable is a variable that is attached to an instance of a class. So every instance of the class you create will have its own instance variables name, cost_price, selling_price, and number_in_stock.
Creating the instance variables in the __init__() method

The __init__() method assigns values to the instance variables based on the arguments passed when creating the object. Recall that the values you place within parentheses when you create the instance of Product are passed to the __init__() method. The parameters name, cost_price, selling_price, and number_in_stock listed in the method's signature store the values passed when creating the instance.

The parameter names and the instance variable names do not need to be the same name. Therefore, the following __init__() method is identical to the one you've written above:

```
# market_stall.py
class Product:
def __init__(self, product_name, cost, sell, n_stock):
self.name = product_name
self.cost_price = cost
self.selling_price = sell
self.number_in_stock = n_stock
```

The parameter names are only used to move information from the method call to the instance variables. Recall that the program calls the __init__() method whenever you create an instance of the class.
self in the method signature

The other place you've used self is as the first parameter in the __init__() signature. You may have noticed that your IDE probably auto-filled this for you when autocompleting __init__(). If you're not making the most of autocompletion in your IDE, then you should do so!

This parameter tells the function that its first argument is the object itself. Therefore, the method has access to the object it is acting on. You'll see this used again in the following section when you create other methods for this class.
Adding Methods To The Class

So far, the Product class assigns instance variables to each instance of the class. Each product you'll create using the class Product will have a 'storage box' attached to it to store the product's name, another one to store the selling price, and so on.

However, you can add more attributes to the class, and these are not limited just to data that can be stored in each object. You can also give the instances of the class the ability to do things as well. And as you know, “doing things” is done by functions in Python:

```
# market_stall.py
class Product:
def __init__(self, name, cost_price, selling_price, number_in_stock):
self.name = name
self.cost_price = cost_price
self.selling_price = selling_price
self.number_in_stock = number_in_stock

def decrease_stock(self, quantity):
self.number_in_stock -= quantity

def change_cost_price(self, new_price):
self.cost_price = new_price

def change_selling_price(self, new_price):
self.selling_price = new_price
```

In addition to the __init__() method, you’ve also defined three further methods. These functions are called methods as they belong to a class. Only objects of type Product will have access to these methods.

Using methods

Let’s see how you can use these methods by going back to market_seller_testing.py:

```
# market_seller_testing.py
from market_stall import Product
first_prod = Product("Coffee", 1.1, 2.5, 30)
second_prod = Product("Chocolate", 0.9, 1.75, 35)
print(f"{first_prod.name} | Number in stock: {first_prod.number_in_stock}")
print(f"{second_prod.name} | Number in stock: {second_prod.number_in_stock}")
first_prod.decrease_stock(7)
print()
print(f"{first_prod.name} | Number in stock: {first_prod.number_in_stock}")
print(f"{second_prod.name} | Number in stock: {second_prod.number_in_stock}")
```

In this version of the script, you’re printing lines to show how many items there are in stock for each product you’ve created. You then call the decrease_stock() method for first_prod with the argument 7. When you print out the numbers in stock again, you’ll see that the number of items in stock for first_prod, which is coffee, has decreased by 7. However, the number of items in stock for second_prod, which is chocolate, remains unchanged since you didn’t call the method for this object:

```
Coffee | Number in stock: 30
Chocolate | Number in stock: 35

Coffee | Number in stock: 23
Chocolate | Number in stock: 35
```

This simple example shows one of the benefits of object-oriented programming. Once you've defined the methods in the class, it becomes straightforward to keep track of which data belong to which object. If you look back at the first and second attempts the market seller made at the beginning of this Chapter, where he used lists and dictionaries, you'll recall that things weren't so easy there.

Each object of a certain class has access to methods that will only act on that object. These methods will only change the data linked to that object.

Revisiting self

Each method you've created has self as the first parameter in the signature. You'll always call a method preceded by the object's name and a full stop, for example first_prod.decrease_stock(). The object itself is passed as the first unseen argument of the method decrease_stock(). Therefore the method call first_prod.decrease_stock(7) has two arguments and not just one. The first argument is first_prod, and the second is the integer 7. However, you don't explicitly write the first argument as this is implied.

You can confirm this with the following experiment in the Console (note, you should close your previous Console and restart a new one since market_stall.py has changed):

```
>>> from market_stall import Product
>>> test = Product("Coffee", 1.1, 2.5, 30)
>>> test.decrease_stock(4, 5)
Traceback (most recent call last):
File "<input>", line 1, in <module>
TypeError: decrease_stock() takes 2 positional arguments but 3 were given
```

You've tried calling decrease_stock() with the integers 4 and 5. This is incorrect as this method only needs one integer. Therefore, the program raises an error. However, look at the last line of the error message very carefully. The message tells you that decrease_stock() takes 2 positional arguments because the method has two parameters in its signature. These two parameters are self and quantity. The error message also says that you passed 3 arguments, even though you only put two numbers in the parentheses. The three arguments you passed are test, 4, and 5.

One more method

Let's add one more method to this class. In market_seller_testing.py, you've printed out a formatted string to show the user the number of items in stock for each product. Unless you enjoy typing, you had to copy and paste that line and change the variable name.

However, this is something that an object of type Product should be able to do. You can do this by adding a method to the class whose job is to print out this formatted string:

```
# market_stall.py
class Product:
def __init__(self, name, cost_price, selling_price, number_in_stock):
self.name = name
self.cost_price = cost_price
self.selling_price = selling_price
self.number_in_stock = number_in_stock
def decrease_stock(self, quantity):
self.number_in_stock -= quantity
def change_cost_price(self, new_price):
self.cost_price = new_price
def change_selling_price(self, new_price):
self.selling_price = new_price
def show_stock(self):
print(f"{self.name} | Number in stock: {self.number_in_stock}")
```

The method show_stock() doesn't make any changes to the object. It merely prints out the formatted string. You can compare the print() line in this method to the ones you wrote in market_seller_testing.py earlier. In the method, you've now replaced the variable name with the keyword self, which is the placeholder for the name of the object.

You can now also update market_seller_testing.py to use this new method:

```
# market_seller_testing.py
from market_stall import Product
first_prod = Product("Coffee", 1.1, 2.5, 30)
second_prod = Product("Chocolate", 0.9, 1.75, 35)
first_prod.show_stock()
second_prod.show_stock()
first_prod.decrease_stock(7)
print()
first_prod.show_stock()
second_prod.show_stock()
```

You should spend some time experimenting with the methods you've created, and you can even create one or two more!

Storing Data And Doing Stuff With Data

At the beginning of this Chapter, I described object-oriented programming as a way of merging the storage of data with doing stuff with data into one structure. There are two types of attributes that an object of a certain class can have:

instance variables such as self.name. These are also called data attributes
methods such as self.decrease_stock()

The instance variables, like all variables, store data. These are the boxes we use to store information in. The methods do stuff with the data, as all functions do. Therefore an object contains within it both the data and the functions to perform actions on the data.

In object-oriented programming, each object carries along with it all the data and the tools it needs. As you did with variable names and function names previously, it is best practice to name your instance variables using nouns and to start the name of your methods with a verb. And as always, descriptive names are better than obscure ones!

You've Already Used Many Classes

You've been using object-oriented programming for a very long time in your Python journey. Let's look at an example:

```
>>> my_numbers = [5, 7, 3, 20, 1]
>>> type(my_numbers)
<class 'list'>
>>> my_numbers.append(235)
>>> my_numbers
[5, 7, 3, 20, 1, 235]
```

In the first line, you've created an instance of the class list. Since lists are one of the basic data types in Python, the way you create an instance looks different to how you've created an instance of Product, but the process behind the scenes is very similar. An object of type list has access to several methods, such as append(), which act on the object and make changes to the object.

When you define your own class, you're creating your own special data type that you can use in the same way you can use other data types.

Type Hinting

I'll make a slight detour in this section to briefly introduce a new topic. There is more information about this topic in one of the Snippets at the end of this Chapter.

In Python, you don't need to declare what data type you will store in a variable. Python will decide what type of data you're storing depending on what you write in the assignment. This statement may seem obvious. However, there are programming languages that require the programmer to state that a specific variable will be an integer, say, before storing an integer in it. In these languages, once you declare the type of a variable, you cannot store any other data type in that variable.

The same is valid for function definitions. Look at the following simple function:

```
>>> def add_integers(a, b):
```

... return a + b

It’s clear from the function’s name and how you’ve written the function that your intention is that a and b are both numbers. You want a and b to be both of type int. However, your Python program doesn’t know that. Both of these function calls are valid:

```
>>> add_integers(5, 7)
12
>>> add_integers("Hello", "World")
'HelloWorld'
```

Clearly, in the second case, you didn’t add numbers, but Python knows how to ‘add’ two strings using the + operator, so it’s not bothered that the arguments are strings and not integers. Only you know that a and b were meant to be numbers.

A colleague you share this code with would also probably guess that a and b should be integers in this simple example. But this is not always the case.

You can add hints in your code to let colleagues know what data type you’re expecting the arguments to be:

```
>>> def add_integers(a: int, b: int):
... return a + b
```

Note that this does not force you to use integers as arguments. These are merely hints to assist programmers who are using this function. The call add_integers("Hello", "World") will still work as it did earlier.

Other reasons to use type hinting

There are some other benefits of using type hinting other than helping other programmers who use your code. Many tools you use in programming will also understand these type hints and will warn you about transgressions. Have a look at the warning that PyCharm gives you when the you write the example above in a script:

Type hinting warnings in PyCharm IDE

You’ll still be able to run this code, but PyCharm has highlighted the string arguments in yellow, and when you hover on the yellow highlight, you’ll get a pop-up window telling you that an int is expected. Other IDEs will also offer similar functionality.

There’s another advantage of using type hinting. As your IDE is now aware of the data type you want your parameters to represent, you can now use autocompletion when you type the parameter name followed by a dot.

Adding Another Class to Help The Market Seller

The market seller has now learned the philosophy of object-oriented programming. Therefore, he asked himself the question: What are the objects that matter to me to run my

market stall?

The items he sells are important, and the Product class creates a data type that allows him to create products. However, that's not enough. The other object that matters to him is his till or cash register. The cash register is where he keeps his money and where he records his transactions. Let's help him create a new class:

```
# market_stall.py
class Product:
def __init__(self, name, cost_price, selling_price, number_in_stock):
self.name = name
self.cost_price = cost_price
self.selling_price = selling_price
self.number_in_stock = number_in_stock
def decrease_stock(self, quantity):
self.number_in_stock -= quantity
def change_cost_price(self, new_price):
self.cost_price = new_price
def change_selling_price(self, new_price):
self.selling_price = new_price
def show_stock(self):
print(f"{self.name} | Number in stock: {self.number_in_stock}")


class CashRegister:
def __init__(self):
self.income = 0
self.profit = 0
self.cash_available = 100
```

You've created the CashRegister class with its __init__() method. When you create an instance of the class CashRegister, you'll create three instance variables that all have a starting value. The market seller's code will create a CashRegister instance every morning when he opens his stall and runs his program.

The instance variables income and profit have initial values of 0 as the market seller has not made any sales yet at the start of the day. He always starts the day with £100 in the cash register, so the cash_available instance variable is initialised with the value 100.

Registering a sale

Let's look at what functions a cash register needs to perform. The main one is to register a sale whenever a customer comes along and buys an item.

You can create a method for the CashRegister class with the following signature:

```
def register_sale(self, item, quantity):
```

The method has three parameters:

self is the first parameter that represents the object the method is acting on
item identifies what product is purchased, whether it’s a coffee or a sandwich, say
quantity determines how many of the item were purchased in the transaction

Question: What data type would you choose for item?

You could make item a string, for example "Coffee". However, there’s a better option.

Product is a data type and objects of this type can therefore be used as an argument for a function or method. By making item an object of type Product, you’re making the most of all the data and functionality available in the Product class.

To make this clearer, you can use type hinting in this method definition:

```
# market_stall.py
class Product:
def __init__(self, name, cost_price, selling_price, number_in_stock):
self.name = name
self.cost_price = cost_price
self.selling_price = selling_price
self.number_in_stock = number_in_stock
def decrease_stock(self, quantity):
self.number_in_stock -= quantity
def change_cost_price(self, new_price):
self.cost_price = new_price
def change_selling_price(self, new_price):
self.selling_price = new_price
def show_stock(self):
print(f"{self.name} | Number in stock: {self.number_in_stock}")
class CashRegister:
def __init__(self):
self.income = 0
self.profit = 0
self.cash_available = 100
def register_sale(self, item: Product, quantity: int):
sale_amount = quantity * item.selling_price
self.income += sale_amount
self.cash_available += sale_amount
self.profit += quantity * (item.selling_price - item.cost_price)
```

The method signature now uses type hinting showing that item should be of type Product and quantity should be an int. The method then works out the sale amount using quantity

and the selling_price instance variable of the item. Incidentally, type hinting means that we can be lazy (read: efficient) when coding as the IDE will autocomplete item's attributes since the IDE is aware that this variable is of type Product.

The register_sale() method then updates the instance variables of the CashRegister object to increase the daily income and the cash available in the till. To work out the profit, you need to use both the item's cost price and selling price to get the profit from the sale of that item.

Testing the method

You can check that this method works by using it in market_seller_testing.py:

```
# market_seller_testing.py
from market_stall import Product, CashRegister
first_prod = Product("Coffee", 1.1, 2.5, 30)
second_prod = Product("Chocolate", 0.9, 1.75, 35)
till = CashRegister()
print(till.income)
print(till.profit)
print(till.cash_available)
till.register_sale(second_prod, 5)
print()
print(till.income)
print(till.profit)
print(till.cash_available)
```

You're now creating an instance of the class CashRegister and showing the data attributes before and after you call register_sale(). This code gives the following output:

```
0
0
100

8.75
4.25
108.75
```

However, there's a bit more you can do in this method.

Updating the Product when registering a sale

You've passed an object of type Product as an argument for the CashRegister method register_sale(). You can also update the number of items in stock of the product. If the market seller sold 5 chocolates in one transaction, then he has five fewer chocolates in stock.

Let’s add an extra line to the register_sale() method in the CashRegister class:

```
# market_stall.py
class Product:
def __init__(self, name, cost_price, selling_price, number_in_stock):
self.name = name
self.cost_price = cost_price
self.selling_price = selling_price
self.number_in_stock = number_in_stock
def decrease_stock(self, quantity):
self.number_in_stock -= quantity
def change_cost_price(self, new_price):
self.cost_price = new_price
def change_selling_price(self, new_price):
self.selling_price = new_price
def show_stock(self):
print(f"{self.name} | Number in stock: {self.number_in_stock}")
class CashRegister:
def __init__(self):
self.income = 0
self.profit = 0
self.cash_available = 100
def register_sale(self, item: Product, quantity: int):
sale_amount = quantity * item.selling_price
self.income += sale_amount
self.cash_available += sale_amount
self.profit += quantity * (item.selling_price - item.cost_price)
item.decrease_stock(quantity)
```

And you can make a few changes in market_seller_testing.py, too:

```
# market_seller_testing.py
from market_stall import Product, CashRegister
first_prod = Product("Coffee", 1.1, 2.5, 30)
second_prod = Product("Chocolate", 0.9, 1.75, 35)
till = CashRegister()
print(till.income)
print(till.profit)
print(till.cash_available)
second_prod.show_stock()
till.register_sale(second_prod, 5)
print()
print(till.income)
print(till.profit)
print(till.cash_available)
second_prod.show_stock()
```

This gives the following output:

```
0
0
100
Chocolate | Number in stock: 35

8.75
4.25
108.75
Chocolate | Number in stock: 30
```

You can see from the output that the program also decreased the number of chocolates in stock when you called till.register_sale().

Before finishing this Chapter, you should go back to the top and review the first and second attempts that the market seller made. These were the versions of the code that didn't use the object-oriented programming approach. Look at the code you wrote earlier on and compare it with the OOP version. You'll be able to appreciate that, once you've defined the classes, using object-oriented programming leads to neater and more readable code in some projects. This will made developing your code quicker and less likely to lead to errors and bugs that may be hard to find.

Conclusion

In this Chapter, you've learned about the object-oriented programming paradigm and the philosophy behind this topic. There are two reasons why you need to know the basics of object-oriented programming.

Firstly, you may want to write your own classes for specific projects in which the investment you put into writing the class in the first place pays off when you write your application. Any classes you define, you can reuse in other projects, too.

You also need to be familiar with classes and OOP because you'll come across many classes as you use standard and third-party modules in Python. Even though someone else has already written these classes, understanding the concept of classes and OOP will help you understand and use the tools in these modules.

There's a lot more to say about object-oriented programming. The aim of this Chapter is to provide an introduction to the basics. I'll briefly discuss a couple of additional topics in the Snippets section at the end of this Chapter. However, a detailed study of OOP is beyond the scope of this book.

In this Chapter, you've learned:

What is the philosophy behind object-oriented programming
How to define a class
How to create an instance of a class
What attributes, instance variables, and methods are
How to define methods

You also learned about:

The increment and decrement operators += and -=
Type hinting

In the next Chapter, you'll learn about using NumPy, which is one of the fundamental modules that you'll use for quantitative applications.
Additional Reading

You can read more about instance variables and an alternative way of picturing them in the blog post about Python Instance Variables.
And you can try out another object-oriented project in which you'll simulate bouncing balls
And here's another project using object-oriented programming to simulate a tennis match in Python

Next Chapter
Browse Zeroth Edition

Snippets
1 | Dynamic Typing and Type Hinting

Python uses dynamic typing. What does this mean? Let's look at the following assignments:

```
>>> my_var = 5
>>> my_var = "hello"
```

In the first line, the Python interpreter creates a label that points towards an integer. You didn't have to let your program know that my_var should be an integer. When the interpreter executed the first line, it looked at the object on the right-hand side of the equals sign. It determined that this is an integer based on the fact that it's a digit, without any quotation marks or brackets, and without a decimal point.

On the second line, the Python interpreter has no problems switching the data type that the variable my_var stores. The interpreter determines the data type of a variable when the line of code runs, and it can change later on in the same program.

This type of behaviour is not universal among programming languages. In statically-typed languages, the programmer needs to state what data type a variable will contain when the

variable is first defined.

As with most things, there are advantages and disadvantages for both systems that I won't get into. Dynamic typing certainly suits Python's style of programming very well.

Duck typing

You'll also hear the term duck typing used to refer to a related concept. The term comes from the phrase "if it walks like a duck and it quacks like a duck, then it is a duck". This idea points to the fact that in many instances, what matters is not what the actual data type is, but what properties the object has.

An example of this concept is indexing:

```
>>> name = "hello"
>>> numbers = [4, 5, 6, 5, 6]
>>> more_numbers = 23, 34, 45, 56, 3
>>> is_raining = True
>>> name[3]
'l'
>>> numbers[3]
5
>>> more_numbers[3]
56
>>> is_raining[3]
Traceback (most recent call last):
File "<input>", line 1, in <module>
TypeError: 'bool' object is not subscriptable
```

The same square brackets notation performs the same task on several data types. In the example above, lists, tuples and strings can all be indexed. Booleans cannot, though.

Type hinting

Python 3.5 introduced type hinting. Type hints do not change Python from a dynamically to a statically-typed language. Instead, as the name suggests, they serve only as hints. Have a look at the following example, which uses type hinting:

```
>>> my_list: list = [4, 5, 6, 7]
>>> my_list: list = "hello"
>>> my_list
'hello'
```

You're adding a type hint when creating the variable my_list. However, in the second line you assign a string to this variable, and there are no complaints from Python's interpreter.

You've already seen an example of type hinting when used with parameters in function definitions and how tools such as IDEs make use of type hints to assist you with your

coding.

You may be working on projects as part of a team where type hinting is used as standard. Type hinting has been used more and more in recent years, especially in the context of production-level code.

For most other applications, it's up to you on how and when to use type hinting. There are times when the extra information can make a significant contribution to making your code more readable. In other instances, you may use it to make the most of the IDEs functionality or other third-party tools that rely on type hinting for performing checks on your code.

2 | Dunder Methods

You've already come across one of the dunder methods you'll see when you define a class in object-oriented programming. These methods whose names start and end with a double underscore have a special status, and they perform specific tasks.

Let's look at a few more in this Snippet. You'll use the Product class you defined earlier in the Chapter:

```
# market_stall.py
class Product:
def __init__(self, name, cost_price, selling_price, number_in_stock):
self.name = name
self.cost_price = cost_price
self.selling_price = selling_price
self.number_in_stock = number_in_stock
def decrease_stock(self, quantity):
self.number_in_stock -= quantity
def change_cost_price(self, new_price):
self.cost_price = new_price
def change_selling_price(self, new_price):
self.selling_price = new_price
def show_stock(self):
print(f"{self.name} | Number in stock: {self.number_in_stock}")
```

Let's experiment with this in a new script testing_dunder_methods.py:

```
# testing_dunder_methods.py
from market_stall import Product
a_product = Product("Coffee", 1.1, 2.5, 30)
print(a_product)
```

The output from this shows the following:

```
<market_stall.Product object at 0x7fec10a6e9d0>
```

This output is not very useful in most instances.
The __str__() method

You may want to customise what happens when you print an object of type Product. To do this, you need to define a dunder method:

```
# market_stall.py
class Product:
def __init__(self, name, cost_price, selling_price, number_in_stock):
self.name = name
self.cost_price = cost_price
self.selling_price = selling_price
self.number_in_stock = number_in_stock
def decrease_stock(self, quantity):
self.number_in_stock -= quantity
def change_cost_price(self, new_price):
self.cost_price = new_price
def change_selling_price(self, new_price):
self.selling_price = new_price
def show_stock(self):
print(f"{self.name} | Number in stock: {self.number_in_stock}")
def __str__(self):
return f"{self.name} | Selling Price: £{self.selling_price}"
```

The __str__() dunder method has the self parameter and returns a string. If you rerun testing_dunder_methods.py now you'll get a different output when you print the object:

```
Coffee | Selling Price: £2.5
```

You can also format the price a bit further:

```
# market_stall.py
class Product:
def __init__(self, name, cost_price, selling_price, number_in_stock):
self.name = name
self.cost_price = cost_price
self.selling_price = selling_price
self.number_in_stock = number_in_stock
def decrease_stock(self, quantity):
self.number_in_stock -= quantity
def change_cost_price(self, new_price):
self.cost_price = new_price
def change_selling_price(self, new_price):
self.selling_price = new_price
def show_stock(self):
```

```
print(f"{self.name} | Number in stock: {self.number_in_stock}")
def __str__(self):
return f"{self.name} | Selling Price: £{self.selling_price:.2f}"
```

Following the selling_price instance variable in the curly brackets of the f-string, you've added a colon to format the output further. The code following the colon formats the float and displays it with two decimal places:

```
Coffee | Selling Price: £2.50
```

It's up to you as the programmer to decide how you'd like the object to be displayed when you need to print it out.

Comparison operators

Let's try the following operation on two objects of type Product:

```
# testing_dunder_methods.py
from market_stall import Product
a_product = Product("Coffee", 1.1, 2.5, 30)
another_product = Product("Chocolate", 0.9, 1.75, 35)
print(a_product > another_product)
```

You're using one of the comparison operators to check whether one object is greater than the other. But what does this mean in the context of objects of type Product? Let's see whether the Python interpreter can figure this out:

```
Traceback (most recent call last):
File "<path>/testing_dunder_methods.py", line 8, in <module>
print(a_product > another_product)
TypeError: '>' not supported between instances of 'Product' and 'Product'
```

No, it cannot. You can see the TypeError stating that the > operator is not supported between two objects of type Product. However, you can fix this with the __gt__() dunder method, which defines the behaviour for the greater than operator:

```
# market_stall.py
class Product:
def __init__(self, name, cost_price, selling_price, number_in_stock):
self.name = name
self.cost_price = cost_price
self.selling_price = selling_price
self.number_in_stock = number_in_stock
def decrease_stock(self, quantity):
self.number_in_stock -= quantity
def change_cost_price(self, new_price):
self.cost_price = new_price
```

```
def change_selling_price(self, new_price):
self.selling_price = new_price
def show_stock(self):
print(f"{self.name} | Number in stock: {self.number_in_stock}")
def __str__(self):
return f"{self.name} | Selling Price: £{self.selling_price:.2f}"
def __gt__(self, other):
return self.selling_price > other.selling_price
```

The __gt__() dunder method has two parameters. You'll see that the IDE autofills both of these. Since this operator compares two objects, there's self and other to represent the two objects. The self parameter represents the object to the left of the > sign, and other represents the object on the right.

In this case, we're assuming that in the context of objects of type Product, the result of the > operator should be determined based on the selling price of both products. The return statement in this dunder method should return a Boolean data type.

Here are some other related operators you can also define:

less than operator < using __lt__()
less than or equal operator <= using __le__()
greater than or equal operator >= using __ge__()
equality operator == using __eq__()

Arithmetic operators

Let's finish with one last dunder method. What happens if you try to add two objects of type Product together:

```
# testing_dunder_methods.py
from market_stall import Product
a_product = Product("Coffee", 1.1, 2.5, 30)
another_product = Product("Chocolate", 0.9, 1.75, 35)
print(a_product + another_product)
```

You may have guessed it's not obvious what adding two products means. The program raises another TypeError:

```
Traceback (most recent call last):
File "<path>/testing_dunder_methods.py", line 8, in <module>
print(a_product + another_product)
TypeError: unsupported operand type(s) for +: 'Product' and 'Product'
```

Another dunder method comes to the rescue. The __add__() dunder method defines how

the + operator works for these objects:

```
# market_stall.py
class Product:
def __init__(self, name, cost_price, selling_price, number_in_stock):
self.name = name
self.cost_price = cost_price
self.selling_price = selling_price
self.number_in_stock = number_in_stock
def decrease_stock(self, quantity):
self.number_in_stock -= quantity
def change_cost_price(self, new_price):
self.cost_price = new_price
def change_selling_price(self, new_price):
self.selling_price = new_price
def show_stock(self):
print(f"{self.name} | Number in stock: {self.number_in_stock}")
def __str__(self):
return f"{self.name} | Selling Price: £{self.selling_price:.2f}"
def __gt__(self, other):
return self.selling_price > other.selling_price
def __add__(self, other):
return self.selling_price + other.selling_price
```

Again, we've decided that the total selling price is what we'd like in this case. How you define these behaviours will depend on the class you're defining and how you want objects of this class to behave. You can try out __sub__() and __mul__() too!

There are many other dunder methods that allow you to customise your class and how it behaves. For example, there is a dunder method to make a data type an iterable and another to make it indexable.

3 | Inheritance

In this Snippet you'll look at a brief introduction to the concept of inheritance in object-oriented programming. When you define a class, you're defining a template to create objects that have similar properties.

You may need objects of groups which are similar to each other but different enough that they cannot use exactly the same class.

Consider the market seller you met earlier in the Chapter. After using his code for a while, he noticed he has a problem. His code treats all sandwiches in the same way. However, he sells different types of sandwiches, and he wants to keep track of them separately.

He decides to write a new class called Sandwich. However, this class has a lot in common

with Product. Therefore, he wants the new class to inherit its properties from the Product class. He'll then make some additional changes.

Inheritance is a key area of object-oriented programming. To create a class that inherits from another class, you can add the parent class in the parentheses when you create the class:

```
class Sandwich(Product):
```

The child class Sandwich inherits from the parent class Product. The child class still needs an __init__() method:

```
# market_stall.py
# Definition of Product not shown
# ...
class Sandwich(Product):
def __init__(self, filling, cost_price, selling_price, number_in_stock):
super().__init__("Sandwich", cost_price, selling_price, number_in_stock)
self.filling = filling
```

The __init__() method parameters are similar to those of the parent class. There is one difference. The second parameter represents the filling of the sandwich and not the name of the product. The name of the product must be "Sandwich" for all objects of this type.

The super() function

The first line of the __init__() method has a new function you've not seen so far. This is the super() function. This function allows you to access the properties of the parent class. You're calling the __init__() method of the parent class in the first line of Sandwich's __init__() method. This means that when you initialise an object of type Sandwich, you first initialise the parent class and then go on with specific tasks for the child class.

The call to super().__init__() doesn't use the filling parameter an object of type Product does not need this. The first argument in super().__init__() is the string "Sandwich", and this will be assigned to the instance variable name.

The last line then creates a new instance variable filling which is specific only to this child class.

Let's try this out in a new script called testing_inheritance.py:

```
# testing_inheritance.py
from market_stall import Sandwich
type_1 = Sandwich("Cheese", 1.7, 3.5, 10)
type_2 = Sandwich("Ham", 1.9, 4, 10)
type_3 = Sandwich("Tuna", 1.8, 4, 10)
print(type_1.name)
print(type_2.name)
```

```
print(type_3.name)
print(type_1.filling)
print(type_2.filling)
print(type_3.filling)
```

You're creating three instances of the class Sandwich with different arguments. An object of type Sandwich has all the properties of an object of type Product. You can see for the first three lines printed out that all objects have the same value for the instance variable name:

```
Sandwich
Sandwich
Sandwich
Cheese
Ham
Tuna
```

However, they all have different values for filling. Let's see what happens when you print the object directly:

```
# testing_inheritance.py
from market_stall import Sandwich
type_1 = Sandwich("Cheese", 1.7, 3.5, 10)
type_2 = Sandwich("Ham", 1.9, 4, 10)
type_3 = Sandwich("Tuna", 1.8, 4, 10)
print(type_1)
print(type_2)
print(type_3)
```

This gives the output defined by the __str__() dunder method of the parent class Product:

```
Sandwich | Selling Price: £3.50
Sandwich | Selling Price: £4.00
Sandwich | Selling Price: £4.00
```

Overriding methods

However, you can define a __str__() dunder method specifically for the Sandwich class:

```
# market_stall.py
# Definition of Product not shown
# ...
class Sandwich(Product):
def __init__(self, filling, cost_price, selling_price, number_in_stock):
super().__init__("Sandwich", cost_price, selling_price, number_in_stock)
self.filling = filling
def __str__(self):
```

return f"{self.filling} sandwich | Selling Price: £{self.selling_price:.2f}"

Sandwich's __str__() method overrides the same method in the parent class Product. The output from testing_inheritance.py now looks as follows:

Cheese sandwich | Selling Price: £3.50
Ham sandwich | Selling Price: £4.00
Tuna sandwich | Selling Price: £4.00

You should also override the show_stock() method similarly.

In the same way that you've added a data attribute to the child class, in this case filling, which doesn't exist for the parent class, you can also create methods specifically for the child class.

Next Chapter
Browse Zeroth Edition

Multiple Choice Questions (MCQs)

What is the main principle of OOP (Object-Oriented Programming)?

a) Inheritance

b) Polymorphism

c) Encapsulation

d) All of the above

Answer: d) All of the above

Which of the following is the correct way to create a class in Python?

a) class MyClass:

b) class MyClass[]:

c) class MyClass(){}

d) MyClass()

Answer: a) class MyClass:

What is used to create an object in Python?

a) object()

b) new MyClass()

c) MyClass()

d) MyClass.new()

Answer: c) MyClass()

Which method is called when an object is created in Python?

a) __initialize__()

b) __start__()

c) __new__()

d) __init__()

Answer: d) __init__()

What does the self keyword refer to in a class method?

a) The class itself

b) The function’s parameters

c) The current instance of the class

d) The parent class

Answer: c) The current instance of the class

Which of the following is true about inheritance in Python?

a) A class can only inherit from one parent class.

b) Inheritance allows the child class to access methods and attributes of the parent class.

c) Python doesn’t support inheritance.

d) Inheritance is only supported for built-in types.

Answer: b) Inheritance allows the child class to access methods and attributes of the parent class.

What does polymorphism mean in Python?

a) Multiple classes can share a name.

b) A class can have multiple methods with the same name.

c) An object can take on many forms.

d) An object can inherit from multiple classes.

Answer: c) An object can take on many forms.

Which method is used to initialize an object’s attributes in Python?

a) __init__()

b) __start__()

c) __new__()

d) __object__()

Answer: a) __init__()

What does super() do in Python?

a) Creates a new class.

b) Calls methods from the parent class.

c) Stops the execution of a method.