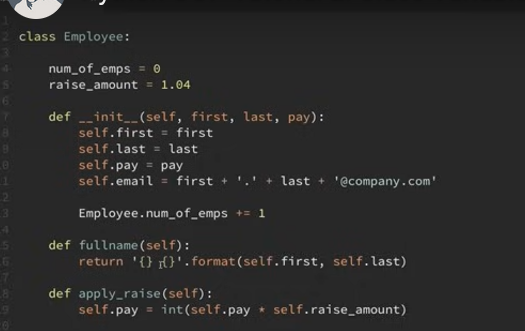
A computer screen shot of a program

Description automatically generated

<https://www.youtube.com/watch?v=ZDa-Z5JzLYM>

This code snippet creates the class “Employee”. This class shares all it’s attributes among all instances of “Employee”. The “init” method takes attributes of the first and last names, the pay, and from the first and last names constructs the employee email. The parameter “self” is telling the “Employee” class that it should be creating an instance of the class with all the attributes. We create emp\_1 and emp\_2 and pass the parameters for the employees into the class “Employee”. In “fullname”, we must pass “self” so that the method knows it is being called on whatever instance of self we’re using “fullname” on. The last two lines of code accomplish the same thing, they’re just showing examples of how to print the full name of the employee we’ve specified



A screen shot of a computer

Description automatically generated

<https://youtu.be/BJ-VvGyQxho>

In the first screenshot of this section there is a line of code under the class “Employee” which sets the “raise\_amount” to 1.04 or 4%. If the “raise\_amount” is not set for an individual instance like emp\_2 or unlike emp\_1, it will fall back to the class defined “raise\_amount”. The way this is coded is that in the method “apply\_raise” the raise is determined from the instance’s “raise\_amount” this is done with – self.raise\_amount we could alternatively set this to be Employee.raise\_amount and the method would fall back to the class defined “raise\_amount” every time instead of the instance’s “raise\_amount”.

A computer screen shot of a program

Description automatically generated

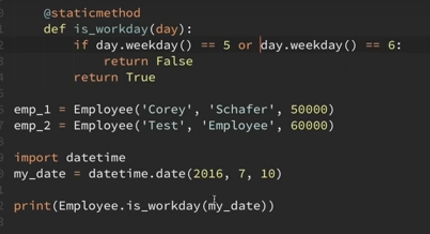
In order to make a class method we use the decorator @classmethod as seen in the code snippet above. To use a class method typically the first argument will be “cls” instead of self like our other methods from this example. We can then use the class method in the general code to set the raise amount for the whole class to whatever we’d like, in this case 5% instead of 4%. We can also set the class amount by using the class method on an instance instead of a class. If we say “Employee.set\_raise\_amt()” and “emp\_1.set\_raise\_amt()” it will accomplish the same thing. In this case emp\_1 is an instance of the class Employee so when we defined the set\_raise\_amt() method and told it to accept a class as the first argument, it is taking the class of the instance “emp\_1” rather than that specific instance which is “self” in the other methods of this example.

<https://youtu.be/rq8cL2XMM5M>

A computer screen shot of a program

Description automatically generated

They are using this class method “from\_string” as an alternative constructor for a new employee. They take the string of employee data, feed it to the class method, parse it, and run the class method and it creates a new employee for us.



<https://youtu.be/RSl87lqOXDE>

Static methods do not pass the instance or class in the arguments i.e. “self” or “cls” either explicitly or implicitly but they typically are logically related to the class itself but don’t need to reference the class. The static method is created with just the arguments we want to work with and the decorator @staticmethod. If your method doesn’t need to use the “self” or “cls” argument it can likely be a static method.



In order to create subclasses, you create your new class with the name of the parent class(es) in parenthesis. This will create your subclasses with all of the functionality and structure from the inherited class info while being tailored to the specific needs of the subclass. To add an additional argument to a subclass, you need to structure your code as follows:

A computer screen shot of a computer code

Description automatically generated

Another example is the manager subclass.

<https://youtu.be/3ohzBxoFHAY>

A computer screen shot of a code

Description automatically generated

A computer screen with text and images

Description automatically generated

When creating the subclass “Manager” there may be temptation to give the argument “employees” as an empty list, but you should not give the option of mutable data types for the entry argument. Creating a new manager should look something like this:



Editing the manager’s list of employees looks like this in the main code:

A computer screen shot of a black background

Description automatically generated

With the code isinstance() or issubclass() we can check the structure of inheritance on different types of employees.

There are magic or special methods that developers often use:

A computer screen with text

Description automatically generated

repr and str. The repr method is typically used for developers while str is often used for the end user of the program. The commented lines and regular lines in the following screengrab print the same thing.

A screen shot of a computer code

Description automatically generated

Now we’ll look at addition of class items. You can specify how the addition is carried out using code similar to the following screengrab

A screen shot of a computer code

Description automatically generated

You can also write code for subtraction, multiplication, division, length, and other special methods. They’re all set up similar to \_\_add\_\_ above which is set up as a “dunder” (double underscore).

There is also the option to use a property decorator which will enable us to treat a method like an attribute in this case the email

<https://youtu.be/jCzT9XFZ5bw>

A computer screen shot of a program

Description automatically generated

We are able to use decorators for setters and deleters as well.

A computer screen with text and numbers

Description automatically generated

