Module 9 - Overview

Introduction

In this module, we will look at using classes and objects in a more advanced manner.

Module Learning Outcomes

After successful completion of this module, you will be able to ...

1. Define classes with data members that are initialized from parameter values.
2. Define classes with data members that are not initialized from parameter values.
3. Define and call methods that accept objects as parameters.
4. Create and use objects of one class inside a method of another class.

Key questions:

* How can you pass an object as a parameter?
* How can one object use another object's data members?
* How can one object use another object's methods?
* How can you create objects of one class inside another class?

Explorations

Use the pages within this module to explore the following concepts:

* Exploration: [The init method and self](https://canvas.oregonstate.edu/courses/1928696/pages/exploration-the-init-method-and-self) (CLO 1e, MLOs 1-2)
* Exploration: [Passing objects as arguments to methods](https://canvas.oregonstate.edu/courses/1928696/pages/exploration-passing-objects-as-arguments-to-methods) (CLO 1d-e, MLO 3)
* Exploration: [Creating and using objects of one class inside another class](https://canvas.oregonstate.edu/courses/1928696/pages/exploration-creating-and-using-objects-of-one-class-inside-another-class) (CLO 1d-e, MLO 4)
* [Module 9 Exercise Solutions](https://canvas.oregonstate.edu/courses/1928696/pages/module-9-exercise-solutions)

Task List

Complete the following assignments and other tasks:

* Read the Exploration pages and do the interactive exercises on those pages (CLO 1e, MLOs 1-4).
* Do [Assignment 9](https://canvas.oregonstate.edu/courses/1928696/assignments/9073294) which gives you practice on advanced object-oriented programming concepts (CLO 1e, MLOs 1-4).
* Exploration: The init method and self
* The \_\_init\_\_ method (initializer method)
* As you may recall from previous modules, the init method is automatically called by Python, only once, when you are creating an object of a class. The purpose of the code in the initializer method is to give initial values to the class data members (which can be modified later by other methods of the class). So it's like a launchpad for any object of that class.
* An init method doesn't need to take a parameter for every data member. If a certain data member will always be initialized with the same value, you don't need a parameter to initialize it with. Let's look at some examples.
* Here we have a class **NonCustomizableRobot** with an init method that doesn't take any parameters. As can be seen on line 2 and when we create objects of that class on lines 23 and 24,  we don't pass any parameters when creating objects of the class.
* And we have another class called **CustomizableRobot** with an init method that accepts 3 parameters - *sensors, actuators*and *name.*As can be seen on line 12 and then when we create objects of that class on lines 31 and 36, we pass parameters when creating objects of the class. However, the *state* and *operating\_system* data members are always initialized to the same values (lines 19 and 20), so no parameters are needed to initialize them with.
* Can you change the init method for **CustomizableRobot** to  accept a value for the parameter *operating\_system* and then use that to initialize the*operating\_system* data member?
* To make this change effective, which lines would you need to modify to make the program run successfully and what would those changed lines look like ?
* **Answer:**
* What is *self?*
* *self*is a reference to the object currently being initialized. It's like the pronoun "I" in English which can be used by every person in a conversation but it still refers to the person currently speaking. Using *self* allows us to write code that is executed for all objects of a class without having to figure out the names of the objects that anyone would create.
* *self* also provides anyone reading the code an easy way to distinguish between data members and local variable names (including parameters). Inside the initializers in the above example, we are using the same variable names for the data members and the parameters but Python knows that we are assigning the values from the parameters to the data members and not vice versa because the data members will always have a *self.*prefix while any local variables will not.
* When Python is executing the code to create an object, it automatically creates a *self* reference to point to the object being created. You can see the changing references to *self*in [this visualizationLinks to an external site.](https://pythontutor.com/visualize.html#code=class%20NonCustomizableRobot%3A%0A%20%20def%20__init__%28self%29%3A%20%0A%20%20%20%20%22%22%22%0A%20%20%20%20Initializes%20an%20object%20without%20accepting%20any%20parameters%0A%20%20%20%20%22%22%22%0A%20%20%20%20self._sensors%20%3D%20%5B'gyrometer','lidar','camera'%5D%20%23a%20list%20of%20sensors%20that%20this%20robot%20has%0A%20%20%20%20self._actuators%20%3D%20%5B'wheels','leds'%5D%20%23a%20list%20of%20actuators%20that%20this%20robot%20has%0A%20%20%20%20self._name%20%3D%20%22Data%22%20%23the%20name%20of%20this%20robot%0A%20%20%20%20self._state%20%3D%20%22on%22%20%23whether%20it's%20on%20or%20off%0A%0Aclass%20CustomizableRobot%3A%0A%20%20def%20__init__%28self,%20sensors,%20actuators,%20name%29%3A%0A%20%20%20%20%22%22%22%0A%20%20%20%20Initializes%20an%20object%20while%20accepting%20parameters%20and%20using%20them%20to%20initialize%0A%20%20%20%20%22%22%22%0A%20%20%20%20self._sensors%20%3D%20sensors%20%23a%20list%20of%20sensors%20that%20this%20robot%20has%0A%20%20%20%20self._actuators%20%3D%20actuators%20%20%23a%20list%20of%20actuators%20that%20this%20robot%20has%0A%20%20%20%20self._name%20%3D%20name%20%23the%20name%20of%20this%20robot%0A%20%20%20%20self._state%20%3D%20%22off%22%20%23whether%20it's%20on%20or%20off%0A%20%20%20%20self._operating_system%20%3D%20%22Linux%22%0A%20%20%20%20%0A%23Creates%20an%20object%20of%20the%20NonCustomizableRobot%20class%20without%20passing%20any%20parameters%0Aoff_the_shelf_robot_object1%20%3D%20NonCustomizableRobot%28%29%20%0Aoff_the_shelf_robot_object2%20%3D%20NonCustomizableRobot%28%29%20%0A%0A%23values%20that%20I%20want%20to%20use%20as%20parameters%20when%20creating%20my%20custom%20robot%20object%0Amy_sensors%20%3D%20%5B'sound%20sensor','compass','light%20sensor'%5D%0Amy_actuators%20%3D%20%5B'robot%20arms','robot%20feet'%5D%0A%0A%23Creates%20an%20object%20of%20the%20CustomizableRobot%20class%20while%20passing%20parameters%0Amy_custom_robot_object%20%3D%20CustomizableRobot%28my_sensors,%20my_actuators,%20%22Terminator%22%29%20%0A%0A%23values%20that%20I%20want%20to%20use%20as%20parameters%20when%20creating%20y) as the objects are being initialized.
* Python requires that all methods of a class have *self*as their first parameter. And whenever we want to write code that uses a data member, Python also requires that we prefix the data member's name with "*self*." (Note the "." after the self).
* Try removing self from Line 2 and running the code. Now put it back and try removing it for the init method of **CustomizableRobot** and running the code. What do you notice? What happens if you remove it from any of the lines where the data members are being initialized?
* You might notice that even though we don't pass any parameters to the init method for **NonCustomizableRobot**, we still write a pair of empty parentheses after the class name when creating objects of that class (lines 23 and 24). What happens if we remove that empty pair of parentheses?
* **Answer:**
* Summary and Syntax
* To summarize, the init method, which is called automatically by Python when creating an object, always needs a *self* parameter in its definition. Apart from *self* a method can take more parameters but doesn't have to.
* When creating an object of a class with an initializer that accepts parameters, the syntax is:
* object\_name = ClassName(value\_for\_parameter\_1, value\_for\_parameter2, value\_for\_parameterN)
* For a class with an initializer that does not accept any parameter, the syntax for creating an object of that class is:
* object\_name = ClassName()
* Real Life Usage
* Almost every class you write should have an init method unless you have a very good reason not to. You can use all or some of the parameters passed to the init method to initialize the data members for that object. Data members can also be initialized without having a corresponding parameter for  them.
* The design of your init method depends on what your customer/team/the assignment text says.
* The *self*reference object allows you to change the data members of an object and have those changes persist across calls to different methods. It also allows us to call one method from inside another.
* Exercises
* (See the module overview for a link to example solutions.)
* 1. Replace the word BLANK with the right  character or word and make this code work successfully.
* 2. Debug and fix this program to make it work successfully.
* 3. Write code for a CookingBot and a Recipe class that are supposed to work as follows.
* The CookingBot class has two data members: *currently\_cooking*which should be initialized to None and *cuisine\_mode*to whatever the user chooses when creating an object of this class.
* The initializer should only accept a parameter to initialize the cuisine\_mode data member. It should print a message saying "Initializing CookingBot object to cuisine\_mode" where you print the actual value of the cuisine\_mode data member.
* The Recipe class has 3 data members: recipe\_id which is an integer, recipe\_name which is a string, and ingredients which is a dictionary where the keys are ingredient names and values are the quantity. When creating a Recipe object, the initializer should accept parameters and initialize all the data members.
* Your program should work with the following test code:
* cb = CookingBot("Canadian") #should create a CookingBot object that cuisine\_mode set to "Italian" and currently\_cooking set to None.  
    
  r1 = Recipe(42, "Poutine", {"water" : "2tbsp", "beef broth" : "20z", "unsalted butter": "6 tbsp", "Russet potatoes":" 2 pounds", "Frying oil":"Sufficient quantity", "pepper":"to taste"}) #should create a Recipe object with the given values for the parameters recipe\_id, recipe\_name and ingredients respectively.  
    
  r2 = Recipe(43, "Chicken tikka masala", None) #should create a Recipe object with the given values for the parameters recipe\_id, recipe\_name and ingredients respectively.
* Exploration: Passing objects as arguments to methods
* Passing objects as arguments to methods
* Objects can be passed as arguments to a method. We know this because everything in Python is an object - ints, floats, Booleans, strings, lists, dictionaries, etc. However, we can also pass objects of user-defined classes as arguments. Let's look at an example.
* In the code above, we define a Robot class and a Sensor class. And on lines 36 through 40 we create a Robot object and multiple Sensor objects.
* The Robot class has a method *install\_sensor* that can accept a sensor object and then adds it to the sensors data member, defined on line 15 as a list.  Since sensorsis a list, we can call the *append*method on it to add anything to the list. Thus, on line 27 we are adding the sensor\_object parameter to the sensors data member. If the sensors data member were a dictionary instead, adding more sensors would be different, since adding to a dictionary is different than adding to a list.
* And if you look at the output of the program you will notice the output is similar to
* Installing sensor  <\_\_main\_\_.Sensor object at 0x7f906a9dd820>
* As you might recall, that's because for objects of a user-defined class, Python doesn't know what information about the object it should print. Therefore it prints the most basic information i.e. the name of the class and the memory address where the object is.
* Once the *sensor\_object* is received as a parameter inside the *install\_sensor* method, it's a reference to the original *sensor\_object* and not a copy of it. And this reference allows us to access all the methods and data members that are available on the *sensor\_object.*You can look at [this visualizationLinks to an external site.](https://pythontutor.com/visualize.html#code=class%20Sensor%3A%0A%20%20def%20__init__%28self,%20name%29%3A%0A%20%20%20%20%22%22%22%0A%20%20%20%20Initializes%20a%20sensor%20object%20while%20giving%20it%20a%20name%0A%20%20%20%20%22%22%22%0A%20%20%20%20self._name%20%3D%20name%0A%0A%20%20def%20get_name%28self%29%3A%0A%20%20%20%20return%20self._name%0A%0Aclass%20CustomizableRobot%3A%0A%20%20def%20__init__%28self,%20name%29%3A%0A%20%20%20%20%22%22%22%0A%20%20%20%20Initializes%20a%20robot%20while%20giving%20it%20a%20name%0A%20%20%20%20%22%22%22%0A%20%20%20%20self._name%20%3D%20name%20%23the%20name%20of%20this%20robot%0A%20%20%20%20self._sensors%20%3D%20%5B%5D%20%23a%20list%20that%20will%20contain%20the%20sensors%20later%20on%0A%20%20%20%20self._actuators%20%3D%20%5B%5D%20%20%23a%20list%20that%20will%20contain%20the%20actuators%20later%20on%0A%20%20%20%20self._state%20%3D%20%22off%22%20%23whether%20it's%20on%20or%20off%0A%20%20%20%20self._operating_system%20%3D%20%22Linux%22%0A%0A%20%20def%20install_sensor%28self,%20sensor_object%29%3A%0A%20%20%20%20%22%22%22%0A%20%20%20%20Install%20the%20given%20sensor%20object%20by%20adding%20to%20the%20list%20of%20sensors%20for%20the%20robot%0A%20%20%20%20%22%22%22%0A%20%20%20%20print%28%22Installing%20sensor%20%22,%20sensor_object%29%0A%20%20%20%20self._sensors.append%28sensor_object%29%0A%0A%20%20def%20remove_sensor%28self%29%3A%0A%20%20%20%20%22%22%22%0A%20%20%20%20Removes%20the%20given%20sensor%20by%20removing%20it%20from%20the%20list%20of%20sensors%20for%20the%20robot%20%0A%20%20%20%20%22%22%22%0A%20%20%20%20pass%0A%20%20%20%20%23TODO%3A%20This%20is%20for%20you%20to%20implement!%0A%20%20%20%20%23TODO%3A%20To%20test,%20uncomment%20line%2048%0A%0A%0A%23Creates%20an%20object%20of%20the%20CustomizableRobot%20class%20%0Awalle_robot_object%20%3D%20CustomizableRobot%28%22WALL-E%22%29%0A%0A%23Creates%20Sensor%20objects%0Acamera_sensor%20%3D%20Sensor%28%223D%20Camera%22%29%0Amic_sensor%20%3D%20Sensor%28%22Microphone%22%29%0A%0A%23Install%20the%20sensors%0Awalle_robot_object.install_sensor%28camera_sensor%29%0Awalle_robot_object.install_sensor%28mic_sensor%29%0A%0) to see what's happening.
* So if you want to print the name of a Sensor object, you can call the *get\_name* method as on line 45. Now can you modify the line 26 to do the same thing inside the *install\_sensor*method of the CustomizableRobot class, to print the name of the sensor which is being passed to the method?
* **Answer:**
* **Explanation of the answer:**
* Can you implement the *remove\_sensor*method in the above program to remove a passed sensor\_object from the list of sensors and print a message after the object is removed? Uncomment the line 48 to test your code. What would the *remove\_sensor*method look like after the changes?
* Answer:
* Summary and Syntax
* To summarize, an object can be passed to a method like any other kind of variable by having a parameter for it in the method definition:
* def method\_name(self, object):
* And it can be passed to a method in a function call like any other kind of variable:
* object\_1.method\_name(object\_2)
* When a method receives an object as a parameter, the parameter is a reference to the original object and using the parameter, you can access all the data members and call the methods of the object that was passed.
* def method\_of\_object1(self, object2):       
   object2.method\_of\_object2()
* Note that any changes you make to the object inside the method will persist outside the method as well.
* Another important thing to note is that although you can access data members of an object passed as a parameter, you should not access private data members (i.e. those with an underscore as a prefix) of a passed object. Instead call the corresponding *get\_* method like how we called *get\_name*above to access the private data member \_*name,*in the above code examples.
* Why pass the whole object?
* You might remember that information hiding is an important part of OOP (object-oriented programming). If the code that is calling the function doesn't itself need access to the object's data members, then it shouldn't get them from the object just in order to pass them to a function. That code should instead just pass the object and let the function ask the object for whatever it needs from it. This doesn't require any additional memory because you're not making a copy of the object for the function - you're just sharing with the function the location in memory where the object is stored (which is how all argument passing in Python works).
* Exercises
* (See the module overview for a link to example solutions.)
* 1. Change the code below to implement the following changes:
* Add an Actuator class with an *\_\_init\_\_* method that accepts a parameter called name and initializes a data member called name. Add a method called *install\_actuator*to the CustomizableRobot class that accepts a parameter called actuator\_object and adds it to actuator data member in the CustomizableRobot class.
* 2.  Replace the word BLANK with the right character/word and make this code work.
* 3.  Implement the following from scratch
* Write code for a CookingBot and a Recipe class that should work as follows.
* The CookingBot class has one data member: *currently\_cooking*which should be initialized to an empty dictionary inside the *\_\_init\_\_*method. It would later on contain data which looks like {recipe\_id: Recipe\_object}
* The CookingBot class has a method called set\_current\_cooking which accepts a Recipe object as a parameter and adds it to the currently\_cooking dictionary such that the key is the recipe\_id and the value is the Recipe object and it prints a message which looks like this:
* "Currently cooking <recipe\_name>"
* The Recipe class has 2 data members: recipe\_id which is an integer and recipe\_name which is a string.  When creating a Recipe object, the initializer should accept 2 parameters and initialize all the data members. The Recipe class should also have corresponding get\_name and get\_id methods.
* Exploration: Creating and using objects of one class inside another class
* Creating objects of one class inside a method of another class
* So far we've defined a lot of classes and created a lot of objects of those classes. You may have noticed that we can create a new object wherever that class is available, even inside a method of another class.
* Let's look at an example:
* Here, inside the ClubAssociation class, the method *add\_new\_club()* creates an object of the Club class every time it is called. This Club object is available as new\_club\_object and then it's added to the data member recognized\_clubs of the ClubAssociation object. Thus the *recognized\_clubs* data member will then contains multiple Club objects.
* What would happen if we commented out line 22, which adds the new\_club\_object to the recognized\_clubs list?
* The variable new\_club\_object is a local variable and its lifetime is only as long as the *add\_new\_club()* method is being executed. So every time the *add\_new\_club()*method is called, the object is created and then as soon as the method ends it disappears, never to be found again. So if we don't want to lose our Club objects, we need to make sure they are stored inside a data member, because data members exist as long as the object they belong to exists.
* So as long as an ClubAssociation object exists, its recognized\_clubs data member will exist even if no method call is being executed on the ClubAssociation class object. And thus all the contents of the data member will continue to exist.
* Can you create another ClubAssociation object (perhaps for your local city) and add a club to it by calling the add\_club method on this ClubAssociation object? What would that look like?
* **Answer:**
* Summary and Syntax
* If we consider two classes ClassA and ClassB,  you can create objects of ClassA inside a method of ClassB using the same syntax as you would when creating an object of ClassA outside. The reverse can also be done.
* But if we create objects of ClassA inside ClassB and don't store them in a data member of ClassB, they will be lost after the method of ClassB finishes executing. You can see this in [this step-by-step visualizationLinks to an external site.](https://pythontutor.com/visualize.html#code=class%20Book%3A%0A%20%20def%20__init__%28self,%20title,%20author%29%3A%0A%20%20%20%20self._title%20%3D%20title%0A%20%20%20%20self._author%20%3D%20author%0A%0A%20%20def%20get_title%28self%29%3A%0A%20%20%20%20return%20self._title%0A%0A%20%20def%20get_author%28self%29%3A%0A%20%20%20%20return%20self._author%0A%20%20%20%20%0Aclass%20Library%3A%0A%20%20def%20__init__%28self%29%3A%0A%20%20%20%20self._holdings%20%3D%20%5B%5D%20%23list%20containing%20all%20Book%20objects%20that%20are%20in%20this%20library%0A%20%0A%20%20def%20add_new_book%28self,%20title,%20author%29%3A%0A%20%20%20%20%22%22%22Creates%20a%20new%20book%20object%20using%20the%20given%20info%20and%20adds%20it%20to%20the%20holdings%20list%22%22%22%0A%20%20%20%20print%28%22Adding%20the%20book%20%22,%20title,%20%22by%22,%20author%29%0A%20%20%20%20%23create%20a%20new%20book%20object%20using%20the%20parameters%0A%20%20%20%20new_book_object%20%3D%20Book%28title,%20author%29%0A%20%20%20%20%0A%20%20%20%20%23add%20it%20to%20the%20holdings%20list%0A%20%20%20%20%23self._holdings.append%28new_book_object%29%0A%0A%20%20def%20list_available_books%28self%29%3A%0A%20%20%20%20print%28self._holdings%29%0A%0A%0Amy_public_library%20%3D%20Library%28%29%20%23create%20the%20library%20object%0A%23now%20add%20books%20to%20the%20library%0Amy_public_library.add_new_book%28%22To%20Kill%20a%20Mockingbird%22,%20%22Harper%20Lee%22%29%0Amy_public_library.add_new_book%28%22One%20Thousand%20and%20One%20Nights%22,%20%22Unknown%22%29%0Amy_public_library.add_new_book%28%22Romance%20of%20the%20Three%20Kingdoms%22,%20%22Luo%20Guanzhong%22%29%0Amy_public_library.list_available_books%28%29%0A&cumulative=false&heapPrimitives=nevernest&mode=edit&origin=opt-frontend.js&py=3&rawInputLstJSON=%5B%5D&textReferences=false).
* To summarize, you can create objects of one class inside a method of another class using the same syntax as you would when creating an object outside a class.
* But if we don't store those created objects in a data member, they will be lost after the method finishes executing as the local variables referencing them go out of scope (the method they were defined in ends).
* Where to use in real life?
* It's useful if you write a library with multiple classes and want to control the creation of objects of some classes. For example, imagine a Library class that will assign an internal barcode to each book when creating a Book object and only the Library class is aware of how to generate the barcode.
* Using objects of one class inside a method of another class
* Once an object has been created, you can access its methods. Let's look at an example of that when a method has created an object of another class.
* Here, we have a slightly modified version of the Club and ClubAssociation classes where now the Club class has a new data member called virtual\_meetings that has the value True if a club holds virtual meetings and False otherwise.
* Take a careful look at lines 42-44, where the add\_new\_club() method is called, and then at lines 20-26, where that method is defined. That method creates a new Club object and then appends it to the list of recognized clubs.
* Inside the *list\_available\_clubs()*method,we iterate through the *recognized\_clubs* data member using a for loop and call the *get\_title*and*get\_theme*method available on the stored Club class objects to get the the title and theme of each Club object and then print it.
* Then on lines 35 and 36, we call another method called *meets\_virtually() of* the Club object to ensure a message is printed if that club has virtual meetings. Taking a careful look at line 35, you will see that not only do we call a method of the Club object, we also use the value returned by that method in an if conditional to decide whether to print the message.
* What would happen if we removed the pair of parentheses on line 35 after the method name in the if conditional? And what happens if we remove the pair of parentheses on line 34?
* **Answer:**
* Using objects of a class inside a method of the same class
* A class can also have methods that take objects of that same class as parameters.
* Let's look at an example using the same Club class with a new method :
* In the method *meet\_with\_other\_club* on line 16, we see that the method accepts one parameter, another\_club. (Of course, we also have the self parameter that is automatically passed by Python when the method is called.) Do you remember what self refers to? It's the object on which the method was called. So when line 34 is executed and Python starts executing the method, self will refer to club1 and for the call on line 35, self will refer to club2.
* We can see how this method is called on lines 34 and 35 where another club object is passed as a parameter. This is the normal syntax for calling a method. The only difference is the parameter is a user-defined object instead of a string, integer, etc.
* Inside the meet\_with\_other\_club method, we use both parameters, self and another\_club. Both are objects of the Club class, so when using these objects we can call methods of the Club class.
* In the if/elif statements on lines 20 through 25, we call the meets\_virtually method and use the value returned by the method to determine whether the given club (self or another\_club) meets virtually or not. How does Python know which Club object to call the meets\_virtually method on?  Again, because of the syntax of calling a method.
* For example, after the call from line 34, when Python encounters line 20, the meets\_virtually method is first called on club1 and then on club3 because inside the if conditional on line 20, we have the call *self.meets\_virtually*first and then *another\_club.meets\_virtually.*
* You can see a [visualization of the entire program hereLinks to an external site.](https://pythontutor.com/visualize.html#code=class%20Club%3A%0A%20%20%20%20def%20__init__%28self,%20title,%20theme,%20virtual_meetings%29%3A%0A%20%20%20%20%20%20%20%20self._title%20%3D%20title%0A%20%20%20%20%20%20%20%20self._theme%20%3D%20theme%0A%20%20%20%20%20%20%20%20self._virtual_meetings%20%3D%20virtual_meetings%20%20%23%20The%20club%20has%20virtual%20meetings,%20True%20or%20False%0A%0A%20%20%20%20def%20get_title%28self%29%3A%0A%20%20%20%20%20%20%20%20return%20self._title%0A%0A%20%20%20%20def%20get_theme%28self%29%3A%0A%20%20%20%20%20%20%20%20return%20self._theme%0A%0A%20%20%20%20def%20meets_virtually%28self%29%3A%0A%20%20%20%20%20%20%20%20return%20self._virtual_meetings%0A%0A%20%20%20%20def%20meet_with_other_club%28self,%20another_club%29%3A%0A%20%20%20%20%20%20%20%20%22%22%22%0A%20%20%20%20%20%20Host%20a%20joint%20meeting%20with%20another%20club%0A%20%20%20%20%20%20%22%22%22%0A%20%20%20%20%20%20%20%20if%20self.meets_virtually%28%29%20is%20True%20and%20another_club.meets_virtually%28%29%20is%20True%3A%0A%20%20%20%20%20%20%20%20%20%20%20%20print%28%22Club%22,%20self.get_title%28%29,%20%22and%20Club%22,%20%20%20%20%20%20%20%20another_club.get_title%28%29,%20%22can%20host%20a%20joint%20meeting%20as%20they%20both%20meet%20virtually%22%29%0A%20%20%20%20%20%20%20%20elif%20another_club.meets_virtually%28%29%20is%20not%20True%20and%20self.meets_virtually%28%29%20is%20not%20True%3A%0A%20%20%20%20%20%20%20%20%20%20%20%20print%28%22Club%22,%20self.get_title%28%29,%20%22and%20Club%22,%20another_club.get_title%28%29,%20%22can%20host%20a%20joint%20meeting%20as%20they%20both%20don't%20meet%20virtually%22%29%0A%20%20%20%20%20%20%20%20else%3A%0A%20%20%20%20%20%20%20%20%20%20%20%20print%28%22Club%22,%20self.get_title%28%29,%20%22and%20Club%22,%20another_club.get_title%28%29,%20%22cannot%20meet%20together%20as%20they%20both%20do%20not%20have%20the%20same%20meeting%20style%22%29%0A%0A%0A%23Create%20Club%20class%20objects%0Aclub1%20%3D%20Club%28%22Strength%20In%20Numbers%22,%20%22recreational%20math%22,%20True%29%0Aclub2%20%3D%20Club%28%22The%20Baker%20Street%20Irregulars%22,%20%22mystery%20fiction%22,). Click "Next" to see the step-by-step visualization. Look carefully at the blue frame in the right hand side and where the arrows are pointing. Note that the arrows may look like they are pointing at a data member but they are actually pointing at the entire Club object.
* Summary and Syntax
* To summarize, you can use objects of the same class or another class inside a method. A method can accept an object of any class as a parameter in the same way it would it accept integer or string parameters. As you will recall from the previous exploration on passing objects as parameters, when an object is received as a parameter we can call methods or access data members of that object. By default a method will always have a *self* parameter and *self* will reference the object on which the method was called. When a method takes an object as a parameter, you can call methods on it that are defined for that class.
* Exercises
* (See the module overview for a link to example solutions.)
* 1. Debug and fix this piece of code so that it can print each book's title, author and a message if an ebook is available.
* 2. Debug and fix this code so that ingredients can be added and removed from the ingredients data member, which is a dictionary.
* 3. Debug and fix the code so that line 88 prints ['Bhel','Salsa'] and line 89 prints ['Salsa']
* 4. Debug and fix the code so that Line 41 works correctly as described in the comment.
* 5. Debug and fix this code so that the line 25 works correctly as described in the comment.
* 6. Complete all the TODO items described in the comments in the code.