Module 10 - Overview

Introduction

In this module, we will look at combining some of the things you have already seen so far. We will look at how you put lists inside dictionaries, dictionaries inside lists, and objects inside dictionaries and lists. And then finally we will look at how to put dictionaries and lists inside objects. Of course we will also take a look at how to actually iterate, add and modify each of these in all the scenarios.

Module Learning Outcomes

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

1. Nest dictionaries inside lists.
2. Nest lists inside dictionaries.
3. Store objects inside lists and dictionaries.
4. Use dictionaries and lists as data members of classes.
5. Iterate over nested collections in all the above scenarios.

Key questions:

* How do you distinguish between dictionaries and lists when one is nested in the other?
* When an object is stored inside a list or a dictionary, how do you access that object's data members?
* When an object is stored inside a list or a dictionary, how do you call that object's methods ?
* When an object has a list as a data member, how do you add and remove elements from that list data member?
* When an object has a dictionary as a data member, how do you add and remove elements from that dictionary data member?

Explorations

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

* Exploration: [Nested data collections](https://canvas.oregonstate.edu/courses/1928696/pages/exploration-nested-data-collections) (CLO 1c, MLOs 1-2)
* Exploration: [Iterating over nested collections](https://canvas.oregonstate.edu/courses/1928696/pages/exploration-iterating-over-nested-collections) (CLO 1c, MLOs 1-2)
* Exploration: [Collections inside objects](https://canvas.oregonstate.edu/courses/1928696/pages/exploration-collections-inside-objects) (CLO 1c-e, MLOs 3-5)
* Exploration: [Objects inside collections](https://canvas.oregonstate.edu/courses/1928696/pages/exploration-objects-inside-collections) (CLO 1c-e, MLOs 3-5)
* [Module 10 Exercise Solutions](https://canvas.oregonstate.edu/courses/1928696/pages/module-10-exercise-solutions)

Task List

Complete the following assignments and other tasks:

* Read the Exploration page and do the exercises on the page (CLOs 1c-e, MLOs 1-5).
* Do [Assignment 10](https://canvas.oregonstate.edu/courses/1928696/assignments/9073284) which gives you practice with using nested data structures (CLOs 1c-e, MLOs 1-5).
* Submit [End-of-Course reflection](https://canvas.oregonstate.edu/courses/1928696/assignments/9073295).

Exploration: Nested data collections

Brief explanation

Now that we have seen lists, dictionaries, tuples, sets, integers, strings and objects of your own classes, let's try nesting them inside one another!

In the first example, where we create *special\_numbers\_list*, you can see it's a list that contains other lists. Try running*print(special\_numbers\_list)* to see how it prints out. What happens if you add more numbers to the odd numbers list and remove some numbers from the evens number list? Is the output of *print(special\_numbers\_list)* affected?

In the second example, we create two separate dictionaries and then put them both inside the *info\_about\_some\_states* list. Try running *print(info\_about\_some\_states)*to see how it prints out. What happens if you add info about "California" to the *state\_capitals* dictionary but do not add it to the *state\_population* dictionary?

Finally, in the third example, instead of creating individual things first and then putting them together, we create a dictionary called *states\_and\_cities\_as\_dictionary* which has state names as the keys and the values are dictionaries. Inside each state's dictionary, we have *popular\_cities*which is a list of popular cities (no offense if we left out your favorite city!) and *capital\_city* which is a string. Now, can you modify that dictionary's assignment to add the following information about Texas?

* Popular cities in Texas: Dallas, Houston, San Antonio, El Paso
* Capital of Texas: Austin

How would states\_and\_cities\_as\_dictionary finally look with the above information added?

**Answer**

Summary and Syntax

To recap, when you define lists, you use [ ] around comma-separated values. For dictionaries, we use { } around comma-separated key-value pairs. For sets, we also use { } but instead of key-value pairs, there are just comma-separated values. To define tuples, you use ( ) around comma-separated values. When you want to put different kinds of collections together or read a nested collection someone else put together, look for those characters to figure out the types of the collections.

Where would you use nested data collections?

The kinds of data collections (lists, dictionaries, sets and tuples) that you have seen so far are good to represent simple real world data, but real world data is sometimes more complex with more details to capture. This is where the ability to nest different data collection inside one another comes handy. For example, consider a dictionary of words. Each word can have multiple meanings.  You can see an example visualized [hereLinks to an external site.](https://pythontutor.com/visualize.html" \l "code=my_dictionary%20%3D%20%7B%0A'apple'%3A%20%5B%20%23from%20https%3A//www.dictionary.com/browse/apple%0A%20%20%20%20%20%20%20%20%20%20%20%20%22the%20usually%20round,%20red%20or%20yellow,%20edible%20fruit%20of%20a%20small%20tree,%20Malus%20sylvestris,%20of%20the%20rose%20family.%22,%0A%20%20%20%20%20%20%20%20%20%20%20%20%22the%20tree,%20cultivated%20in%20most%20temperate%20regions.%22,%0A%20%20%20%20%20%20%20%20%20%20%20%20%22the%20fruit%20of%20any%20of%20certain%20other%20species%20of%20tree%20of%20the%20same%20genus.%22,%0A%20%20%20%20%20%20%20%20%20%20%20%20%22any%20of%20these%20trees.%22,%0A%20%20%20%20%20%20%20%20%20%20%20%20%22any%20of%20various%20other%20similar%20fruits,%20or%20fruitlike%20products%20or%20plants,%20as%20the%20custard%20apple,%20love%20apple,%20May%20apple,%20or%20oak%20apple.%20%22,%0A%20%20%20%20%20%20%20%20%20%20%20%20%22Informal.%20anything%20resembling%20an%20apple%20in%20size%20and%20shape,%20as%20a%20ball,%20especially%20a%20baseball.%22,%0A%20%20%20%20%20%20%20%20%20%20%20%20%5D,%0A%20%20%20%20%20%20%20%20'English'%3A%20%20%23from%20https%3A//www.dictionary.com/browse/english%0A%20%20%20%20%20%20%20%20%20%20%20%20%5B%0A%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%7B%0A%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%22adjective%22%3A%20%5B%0A%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%22of,%20relating%20to,%20or%20characteristic%20of%20England%20or%20its%20inhabitants,%20institutions,%20etc.%22,%0A%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%22belonging%20or%20relating%20to,%20or%20spoken%20or%20written%20in,%20the%20English%20language%3A%20%22%0A%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%5D,%0A%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%22noun%22%3A%20%5B%0A%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%22%28used%20with%20a%20plural%20verb%29%20the%20people%20of%20England%20collectively,%20especially%20as%20distinguished%20from%20the%20Scots,%20Welsh,%20and%20Irish.%22,%0A%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%22the%20Germanic%20language%20of%20the%20British%20Isles,%20widespread%20and%20standard%20also%20in%20the%20U.S.%20and%20most%20of%20the%20British%20Commonwealth,%20historically%20termed%20Old%20English%20%28c450%E2%80%93c1150%29,%20Middle%20English%20%28c1150%E2%80%93c1475%29,%20and%20Modern%20English%20%28after%20c1475%29.%20Abbreviation%3A%20E%22,%0A%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%22English%20language,%20composition,%20and%20literature%20as%20offered%20as%20a%20course%20of%20study%20in%20school.%22,%0A%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%22a%20specific%20variety%20of%20this%20language,%20as%20that%20of%20a%20particular%20time,%20place,%20or%20person%3A%20American%20English%3B%20Shakespearean%20English.%22,%0A%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%22simple,%20straightforward%20language%3A%20%22,%0A%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%7B%0A%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%22Sports.%28sometimes%20lowercase%29%20%22%3A%20%0A%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%5B'%20a%20spinning%20motion%20imparted%20to%20a%20ball,%20especially%20in%20billiards.',%0A%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20'%20body%20English.%20',%0A%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%5D,%0A%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%7D,%0A%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%22Printing.%20a%2014-point%20type%20of%20a%20size%20between%20pica%20and%20Columbian.%22,%0A%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%22a%20grade%20of%20calendered%20paper%20having%20a%20smooth%20matte%20finish.%22,%0A%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%5D,%0A%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%22verb%20%28used%20with%20object%29%22%3A%20%5B%0A%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%22to%20translate%20into%20English%3A%20%22,%0A%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%22to%20adopt%20%28a%20foreign%20word%29%20into%20English%3B%20Anglicize.%20%22%0A%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%5D%0A%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%7D%0A%20%20%20%20%20%20%20%20%20%20%20%20%5D%0A%20%20%20%20%20%20%20%20%7D&cumulative=false&curInstr=20&heapPrimitives=nevernest&mode=display&origin=opt-frontend.js&py=3&rawInputLstJSON=%5B%5D&textReferences=false" \t "_blank) (use the horizontal and vertical scroll bars to see the entire data collection).

Exercises

(See the module overview for a link to example solutions.)

1. Replace the word BLANK with the right character and make this code work.

course\_records = BLANK  
  
      "CS161" : {  
  
          "course\_name": "Introduction to Computer Science I",  
  
          "instructors" : BLANK'Luyao Zhang', 'Tim Alcon', 'Samarendra Hedaoo'BLANK,  
  
         "undergraduate\_learning\_assistants": ['Corbin','Brandon','Darren','Sophia','Michelle','Julia','Lucas','Joseph','Andrew']  
  
    }  
  
BLANK

2. Create a nested data collection to store the following kind of data.

* + CRA is a govt agency which issues recall alerts of various products the market. Recall alerts are issued for these 5 different types of products: Food, Vehicle, Health, CPS and and Other. Each recall product has a recallD which is a number, and a title, a date and a URL which are all strings. The way CRA organizes wants this information organized is Recall alerts -> Type of Products -> List of recalled products -> Detail for each recall.

Exploration: Iterating over nested collections

Iterating over nested collections

Now let's look at how to iterate over nested collections.

Iterating means processing each element of a collection. Processing could mean printing or comparing or any operation that you want to perform on each element. In the context of nested collections, we might need to process each element of the nested collections as well.

Let's start by looking at basics of how we iterate over lists and dictionaries and then look at nested collections.

Summary and Syntax

A rule of thumb we can come up with is that you would need as many levels of nested *for*loops as the levels of nesting in your data collection. This of course doesn't apply if you don't need to access the most deeply nested data. We can also do without multiple levels of nested for loops if we start right at the data collection that we want to process.

There are three questions to ask when you want to iterate over nested data collections:

1. What kinds of data collections do we have? Can you name them? For example: "The outermost data collection is a list whose elements are strings and a dictionary, where the keys of each dictionary are strings and the associated values are lists".
2. What is the nesting level where the element that you want to process is located? For example, you might want be only interested in printing the value of  *popular\_cities*key inside the dictionary for each state.
3. Do you want to do something with each level of nesting as you go or are you interested in only the level where the interesting data is located?

The first question will help you figure out the way to write your *for*loops. The second question will let you know the level of nested loops you will need to write. And the third question will tell you if you can use fewer number of for loops.

Here's an example of iterating over only the innermost level of a nested collection with only one forloop.

Note that this allows us to access only one horizontal level of nesting i.e. this will not work if we want to print the contents of *popular\_cities* for all the states.

Exercise

1. Match the things on the left with the best possible option on the right and explain why that is the best matching pair. (This exercise uses single-letter variable names, but that's only so that you can't make a choice based on clues given by better, more descriptive variable names.)

|  |  |  |
| --- | --- | --- |
| **Left side** | **Answers (hidden)** | **Right side** |
| **1)** A dictionary whose values are lists | Answer | **a)** my\_data = {'a':42, 'b': 56} |
| **2)** A list that contains a list and a dictionary | Answer | **b)** your\_data = [1,2,3,4, {'a':42,'b':56}] |
| **3)** A list that contains some ints and a dictionary | Answer | **c)** their\_data = [[4,8,15,16,23,42],{'Name':'Lost'}] |
| **4)**  Iterating over a list of lists and iterating over each inner list to print each of its elements | Answer | **d)** a\_list = [ {'a':4, 'b':8, 'c':15, 'd':16, 'e':42}, {'course\_1\_name':course\_object1, 'course\_2\_name': course\_object2} ] |
| **5)** Iterating over a list and printing each element | Answer | **e)** for i in some\_data\_collection:            print(i, some\_data\_collection[i]) |
| **6)** Iterating over a dictionary and printing each key-value pair | Answer | **f)**for i in some\_data\_collection:           print(i) |
| **7)**Iterating over a list of dictionaries and iterating over each of those dictionaries to print the values stored in it | Answer | **g)** for i in a\_data\_collection:          for j in a\_data\_collection[i]:                print(j) |
| **8)** Iterating over a dictionary that has lists for its values, and iterating over each of those lists to print its elements | Answer | **h)** a\_collection = {'a':[42], 'b':[4,8,15,16,23,42]} |
| 9) A list of dictionaries | Answer | **i)** for i in a\_data\_collection:           for j in i:                 print(i[j]) |
|  |  | **j)**  for i in a\_data\_collection:           for j in i:                 print(j) |

Exploration: Collections inside objects

Brief explanation

So far we have stored various kind of collections as variables either inside or outside functions. But we can also put them inside *objects*! Look at this example:

Here, the *ingredients*data member is a list. (Recall that variables inside a class with the prefix *self* are called data members.) Inside theinit method, which runs only when an object of the class is created, we initialize the *ingredients* data member as an empty list.

Then, the add\_ingredient() method takes the *ingredient*passed as a parameter and adds it to the list using the *append*() method that can be used on any list. Similarly, the *remove\_ingredient()* method takes the given *ingredient\_name* and removes it from the list using the *remove()* method that can be used on any list.

Finally, the *print\_ingredients()* method prints the list of ingredients whenever it is called.

Try changing the order of lines 20 through 28 in the above code and then run the program. Notice the difference in the output. What happens if you move line 19 to line 29 ?

Try this out and reflect

Suppose we wanted to change the *ingredients* data member in the above class to be a dictionary that would look something like {'ingredient\_name' : 'quantity'}. How would we do that?

What would you need to do to make *ingredients*a dictionary instead of a list ? You can change the *add\_ingredient*method definition to accept ingredient quantity along with the name.

**Answer**

Summary and Syntax

To summarize, if we want to use a collection inside an object we would need to do the following:

* Figure out what kind of collection do we want to use ? Dictionaries? Lists? Sets? Tuples? The answer will depend on what kind of data to store. It is always a good idea at this point to actually write down how you want your stored data to look. For example, if it's a dictionary, what will be the key and what will be the value?
* In the initializer method, change the data member to use an empty collection as the value. An exception to this is if you want your object to start off with some data.
* Then, define methods to add and remove items from that collection data member. The way they will work depends on the type of collection. To be able to write this method, you will want to find out how an element is added or removed from a specific collection type.

Where would you use collections as data members?

Whenever you need to store an unknown number of values inside an object, it's a good idea to use collections as data members. For example, the number of ingredients for a peanut butter sandwich would be different from those for a peanut coconut stew or an enchilada recipe. So if we want a single Recipe class to be usable for various kind of recipes with varying number of ingredients, then it's good to have the ingredients as a list or a dictionary or set. Note that if you use a tuple as a data member, you cannot add or remove values from it, since tuples are immutable.

Exercises

(See the module overview for a link to example solutions.)

1. Replace all occurrences of BLANK in the code below with the correct code and and make it work!

Exploration: Objects inside collections

Brief explanation

Just like you can store integers, strings, and all kind of collections inside each other, we can also store objects inside collections! So this is essentially the opposite of what you just did above. (Again, this should be no surprise, since all values in Python are objects, but this exploration specifically looks at storing objects of user-defined classes in collections.)

Look at this example:

Here we have an Ingredient class with an init method that accepts name as a parameter and initializes the *name*data member. The class also has a *get\_name()* method that returns the value of the *name* data member.

On lines 13 through 21 we create various Ingredient objects. On line 27 we create a list where we store spice Ingredient objects.  All that we had to do was include the object references (aka the variable names) in the list. This is very similar to creating a list with integers, strings or any other kind of elements. And if we forget to add an element when the list was created, we can see on line 30 how an object can be added later on. Can you try adding the *cinnamon* ingredient object to the *list\_of\_spices?*

Similarly, on line 33-36 we create a dictionary of objects where we have the objects as the values. The difference here is we need to provide a key for each value. In this case, the keys are strings. How would you add the *broccoli*Ingredient object to the*raw\_ingredients\_dictionary*?  Try doing that and running the program.

Remember that when you print a user-defined object, it looks something like <\_\_main\_\_.Ingredient object at 0x7f93d1b3c070>. Python doesn't understand anything about Ingredients (or Recipes or Students or Instructors or any other classes that we can create), so it doesn't know that we might want to see the *name*attribute when printing an Ingredient object. Instead, its default approach is to print the name of the class and the memory address where the object is stored.

So how do we get the name of an Ingredient and print it?  Try un-commenting the line 47.

How would we print the name of *each* ingredient for *each* Ingredient object in a list? Try un-commenting the lines 49 and 50 to see how that's done.

Putting objects inside collections that are data members

Now, let's look at how to put objects of a different class inside a collection that is a data member.

In the program above, inside the Recipe class we have the *ingredients* data member, which is a dictionary. (Recall that variables inside a class with the prefix *self.*are called data members while the *\_*(underscore) part is the convention to indicate that a data member is private.)  We start off with the *ingredients*being initialized as an empty dictionary in theinit method which will run automatically and only once when an object of Recipe class is created.

We also have a class called Ingredient which allows us to create an Ingredient object using a name. And that class also has a method called *get\_name()* that can be used to get the name for a given Ingredient object.

Looking back at the Recipe class, the *add\_ingredient()* method accepts an *ingredient\_object* as a parameter. As the name of the parameter suggests, this parameter is supposed to be an object of the Ingredient class. (Note that good variable/parameter names make your code readable for yourself and others; they don't have any impact on how Python treats them).

The *add\_ingredient* method takes the name of the passed ingredient object and assigns it to a variable called *ingredient\_name*on line 13. Then on line 14, you see a new key-value pair being added to the *ingredients* dictionary, where *name* is used as the key and the Ingredient object that was passed as a parameter is used as the associated value!

Similarly, the *remove\_ingredient()* method takes the given ingredient name and removes it from the dictionary using the *pop()* method that can be used on any dictionary. The *pop()*method accepts a key as the parameter and looks inside the dictionary for a matching key and removes the key-value pair. Since keys are always unique in a dictionary, you can assume that the only instance of that key-value pair will be removed correctly.

Finally, the*print\_ingredients()* method prints the entire dictionary but you would notice the same kind of output for printing objects that you saw in the previous section.

Summary and Syntax

An object of a user-defined class can be added to a list or a dictionary just like an integer object or a string object can. The only thing that is required is that the object be created already. Note that if we are adding an object to a dictionary, we need to provide a key if we intend to use the object as the value.

When removing the objects, we would use the list or dictionary's built-in methods to remove any kind of element.

When printing an object of a user-defined class, we will get a default output from Python that will not include any data members of the object. To print a more useful output we could call the relevant *get\_* method of the objects.

Where would you use this?

In the object-oriented programming paradigm, a technique called composition is used to model scenarios where an object can contain objects of a different class. For example, a Car class could have a *sensors* data member which would be a collection containing multiple objects of *Sensor* class.

This would allow the methods of Car class to call the methods of various sensor objects to get data from the relevant sensors. It would be highly difficult to do this if we associate the Car class with the Sensor class in any other way.

A rule of thumb to recognize scenarios for composition is to look for requirements that describe some kind of "*has"*relationship between two classes or objects like *"*A Car has Sensors".

Exercises

(See the module overview for a link to example solutions.)

1. Replace all occurrences of BLANK in the code below with the correct code and and make it work!

2. Debug and fix this piece of code.

3. Write this program

* + Take the code for Recipe class and Ingredients class [from the example code in this section.](https://canvas.oregonstate.edu/courses/1928696/pages/exploration-objects-inside-collections?module_item_id=22739876#code_for_exercise)
  + Modify the *add\_ingredient* method in the Recipe class to take two parameters, *ingredient\_object*and *quantity.*It should store them in the the *ingredients* data member such that the *ingredient\_object*would be the key, and the *quantity*would be the value. Modify the calls to the *add\_ingredient*method with random values for quantity parameters so that you can test the method.
  + Make appropriate changes to the*remove\_ingredient*method as well, without changing the parameter it receives.

4. Match the things on the left with the best possible option on the right and explain why that is the best matching pair.

|  |  |  |
| --- | --- | --- |
| **Left side** | **Answers (hidden)** | **Right side** |
| **1)** A dictionary of objects | Answer | **a)** self.\_dictionary\_of\_courses.pop(key) |
| **2)** A data member that is a list of objects | Answer | **b)** list\_of\_courses.append(course\_object) |
| **3)** A list of objects | Answer | **c)** self.\_list\_of\_courses.append(course\_object) |
| **4)** A list containing dictionaries | Answer | **d)** a\_list = [ {'a':4, 'b':8, 'c':15, 'd':16, 'e':42}, {'course\_1\_name':course\_object1, 'course\_2\_name': course\_object2} ] |
| **5)** Adding an object to a data member that is a list | Answer | **e)** course\_completed\_by\_student = {'course\_1\_name':course\_object1, 'course\_2\_name': course\_object2} |
| **6)** A data member which has an object as a value | Answer | **f)**self.\_program\_enrolled\_for = ProgramObject |
| **7)**Adding an object to a list | Answer | **g)** self.\_courses\_completed\_by\_student = [course\_object1, course\_object2, course\_object3] |
| **8)** Removing an object from a dictionary that is a data member | Answer | **h)** courses\_completed\_by\_student = [course\_object1, course\_object2, course\_object3] |
|  | Answer | **i)** self.\_dictionary\_of\_courses.remove(key) |
|  | Answer | **j)** dictionary\_of\_courses.pop(key) |