# Module 8 – Lists

Instructions

1) If not already open, from the Anaconda Navigator Home screen, launch Sypder

2) In Spyder, go to “File”, then “Open”, then navigate to the location where you saved: “8_Lists.py”, and select it to open the script file.

**3) Introduction to Lists**

In a couple of previous modules, we have seen a couple examples of lists, but they’ve been used as parts of examples illustrating other topics. Now let’s focus on them, as they’re a pretty common data type that you will encounter as you continue to work in Python. They’re flexible, powerful, and knowing the basics of what you can and cannot do with them is the focus on this module.

In contrast to string data, list data are not immutable, so you can change them without needing to set new variables. As you pass data into and out of them as needed, this becomes helpful. Lists are collections of items. They also can contain *different* data types as individual objects within a single list. As in, elements in a list can include integers, float data, strings, or even other lists.

**4) Working with lists**

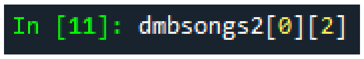
**Run line 8**, which initiates a simple list of integers. Note a couple things about the syntax. Lists are bounded by brackets [item 1, item 2, item 3, item n….], where each item is referenced by its index position. So in testlist (Line 8), the integer 11 is at index value 1, and 13 is at index value 2. **Run lines 8 and 9 to confirm** and see the whole list contents.

**Running Line 10**, though, will identify the element at testlist’s index position 2 (which is the integer 13), and then set it to 14. When you run the line that prints the list contents (**Run Line 11**), this change should be reflected.

Next, **run line 14** to store this list of string elements in a list variable. Then, run the simple definite loop (**Run lines 16-17**) to print out a statement that begins with “Band member:” and then includes the element value (referenced by accessing the iterator variable). Now **run lines 19-22** to do something similar, but also include a counter to go alongside the element value.

*Activity – Can you adjust the code in this definite loop to have the counter start at 1 instead of 0?*

**Run line 26**, which now enters different data types into one list. There are four string objects and one integer object in this list. Now **run line 27**, which also has four elements stored in the variable initiated (**confirm by running line 29**): 2 lists, one string object, and one tuple object (more on that soon). You can access elements of lists, and of lists within lists using index position values, too. What do you think will be returned if you enter the following in the console?



```
In [11]: dmbsongs2[0][2]
```

Lines 30 and 31 may help offer some clues.

## 5) Some List Operations

**Run Lines 33 and 34**, which stores two list variables, each of which contain 6 integer values.

Using the '+' operator between a and b will **concatenate** them (**run Lines 37-38**), so be aware of this when using that operation. That may or may not be what you want to do with these list elements.

To see a full list of all allowable list operations, **run line 41**. You will note this list is a bit shorter than the one you saw for strings.

You can also initiate an empty list, and store it as a variable. This list can act as a container to hold the output of operations (like functions), for example. Lines 45 and 47 both show you examples of how to create an empty list, so **run both**

To add elements to a list, make use of the **append** operation. **Running Lines 50 and 51** will add two elements to the list initiated on Line 47, *in the order in which they were entered*. Confirm by entering **myotherlist** in the console. Lines 53 and 54 offer some language that can be used to identify whether or not an items is in the list.

Lines 57-60 are all allowable operations that can be used to report summary statistics about a list. Returning to our list c (initiated on Line 37), we can use these lines to accomplish such a report. But, alas mean(c) is still not available – Line 61 provides it by leveraging two allowable operations, though. **Run lines 57-61 together**

## 6) An Applied Example

**Run Lines 64-68**. Much of this looks familiar from previous modules, save for the now empty list initiated on Line 65. The rest is as before:

- A list of 17 unique population values – as integers – is created on line 64
- On line 66, a string of all 17 names is created
- Lines 67 and 68 split the elements of the string anytime the character "," is found, and stores them in another list (NVList, on Line 67). Line 68 sorts the elements of the list alphabetically, storing them accordingly at new index positions.

**Run lines 71-73**, which, iterate through each the items in the NVList variable and do the following: 1) removes any blank spaces using the .strip method, then 2) appends these formatted string elements to the previously empty list initiated on Line 65. You can check by **running Line 74.**

Pay close attention to Line 77, which offers another way to do everything accomplished on Lines 65 and 71-73 in a different, and more compact and elegant way. **Run line 77** and confirm by entering NVCounties2 into the console. Note the brackets on the right hand side of the variable assignment expression in the code. This signals that NVCounties2 will be a list object, and by referencing the already-populated list on Line 67, this executes a definite loop that makes use of the iterator variable and the .strip method *before* the "for i in NVList…" syntax is encountered.

*Activity – can you modify the code on Lines 71-74 to ensure that the word "County" in included for each of the list elements appended to the NVCounties list, except for Carson City, which should be left as is?*

This completes the structured part of Module 8. Feel free to experiment some more with setting variables and values, working with print statements, or anything else you are curious about.