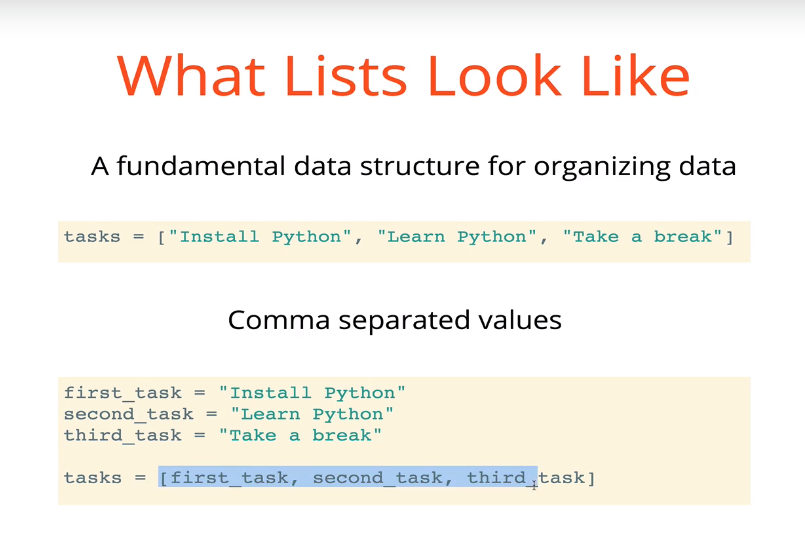
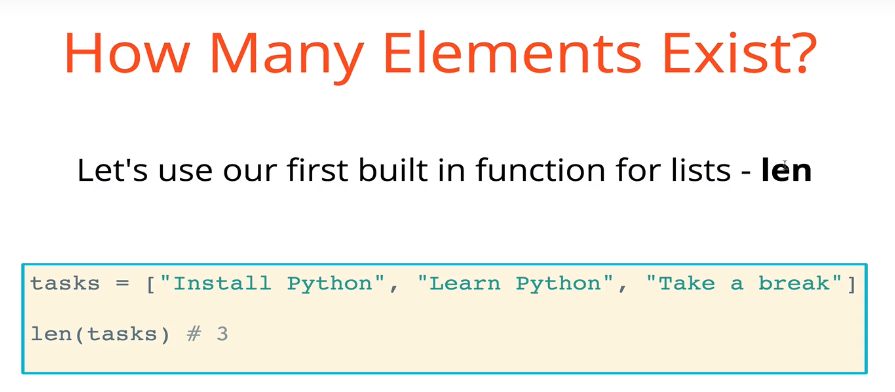
Section 12: Lists

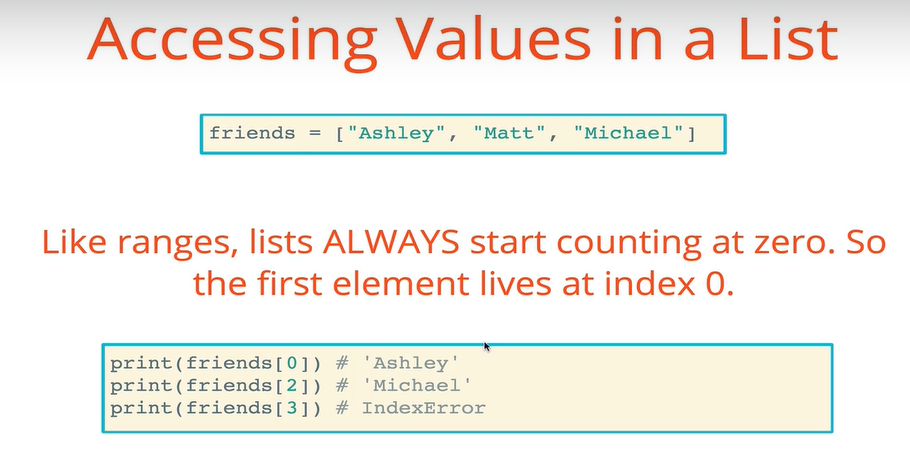
* A list is a mutable collection or grouping of items
  + These items can be anything!
    - Strings, floats, integers, Booleans, other lists, dictionaries, etc.
* It allows us to have as many items as we want in a single structure
  + Can be reordered, split, deleted, and modified
* Lists can be created manually using square brackets, with different entries in the list separated by commas
  + Remember that a single list can contain any number of any type of entry.

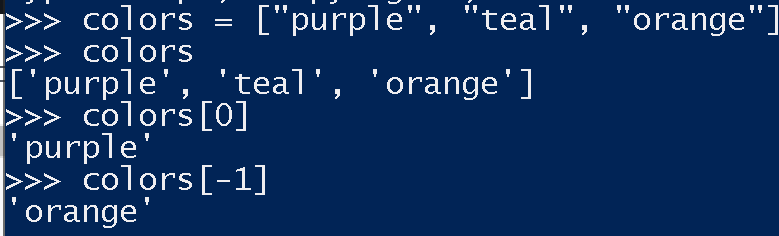


* The len(*list\_name*) function returns the number of items in a list

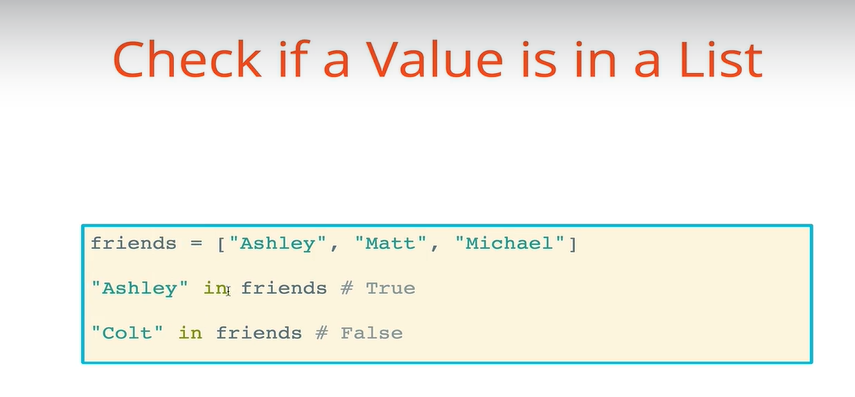


* Lists can also be created using the built-in function list()
* Lists are *ordered*, and indexing for them starts at 0
  + You can access specific items in the list by referencing the indices of those items using square brackets: *list\_name*[*index\_of\_value*]
  + If an item does not exist at that index, you will receive an indexing error
  + You can also use negative indices. It works backwards, where -1 refers to the last element in the list

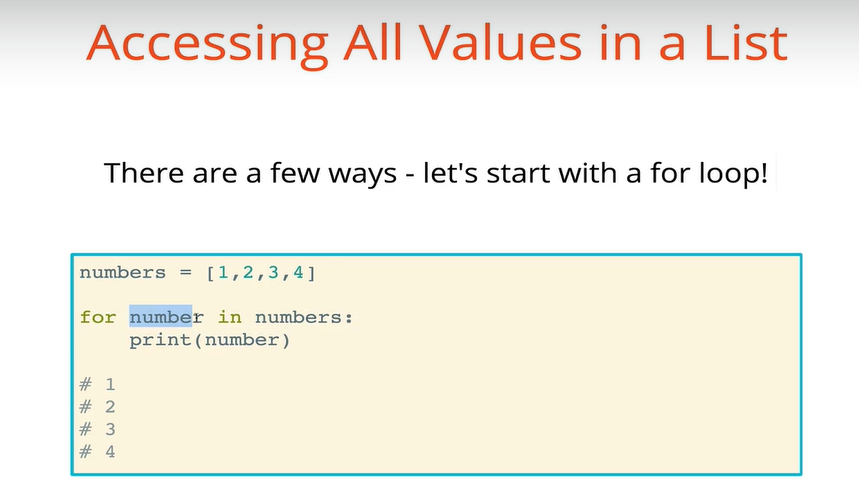




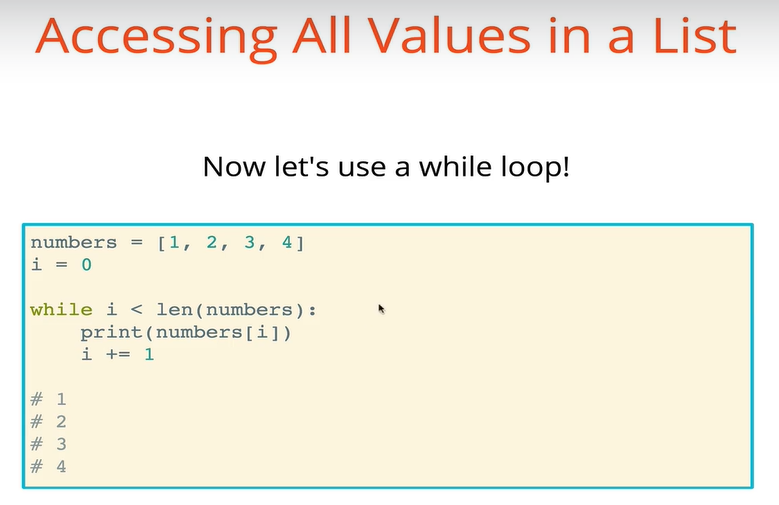
* You can use the “in” keyword to determine whether a particular value is in a particular list. It will return a boolean
  + It must be exactly the element you are looking for (capitalization matters)



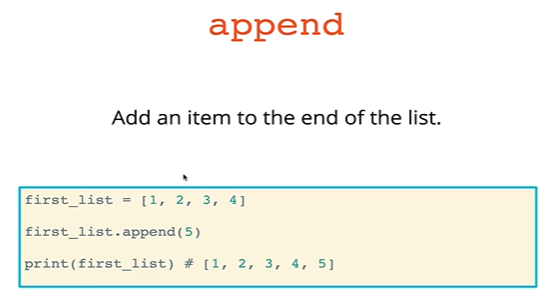
* You can also use the indexing reference to change a value in a list
  + *list*[*index*] = *new\_value* will change the value of the item at the indexed location to the *new*\_*value*
* For loops can be used to iterate over lists



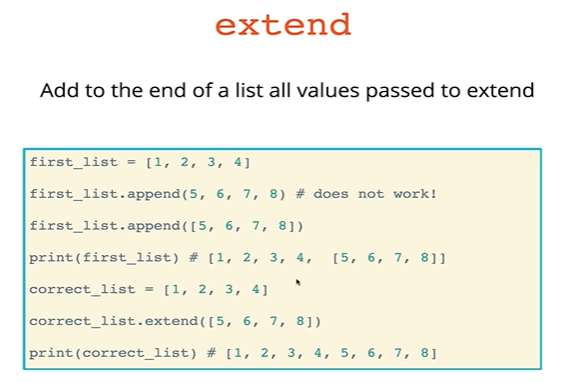
* While loops can also be used to iterate over lists, although it requires a bit more syntax
  + We start a counter at 0 (assuming we want to start iterating at the beginning of the list), and with every pass through the while loop we increment the counter by 1
  + Important: we need to stop the loop when the counter value becomes equal to the length of the list. Remember that because list indexing starts at 0, the length of any list is one higher than the highest index of the list
    - A list of length 10 uses indices 0 through 9. Attempting to access the 10th element of the list will give you an indexing error



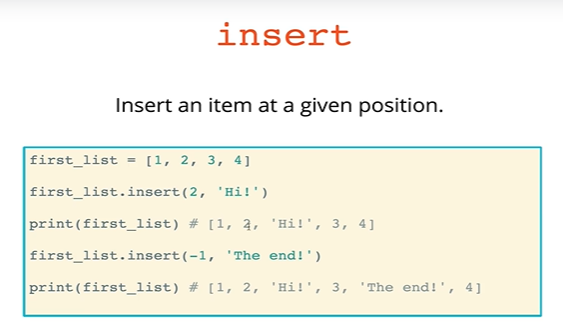
* + One advantage of the while loop is that you can directly use the index. See example in “while\_loop\_lists.py”
* Brief overview of methods vs. functions
  + A method is a type of function, but there is an important distinction between them
  + *print* and *len* are built-in functions, not specific to any particular data type
  + However, methods are specific to particular data types or instances of data types (known as objects). This usually involves using dot notation
    - For example, if we have a list of data, we can utilize list methods on that data
      * data.append()
  + When we have a function attached to a particular object, it is a method. If we have a function that is free-floating, it is simply a function.
* The list.append() method



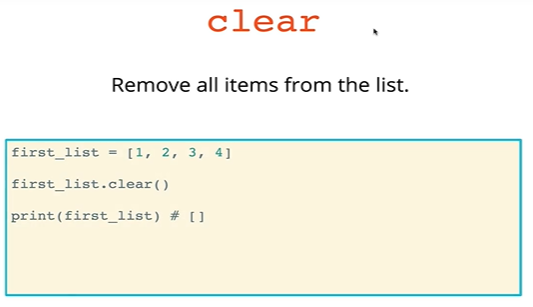
* + Append can only append one item at a time. If you try to append multiple items, you will get an error. If you try to append a single list of multiple items, the whole list (not the items inside) will be added as a single item
* The list.extend() method
  + This can be used to add multiple items to the end of a list



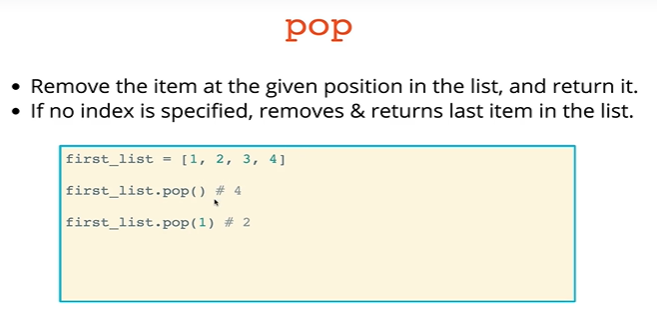
* The list.insert(index, item) method
  + The insert method is used to add an item to a specific index in the list



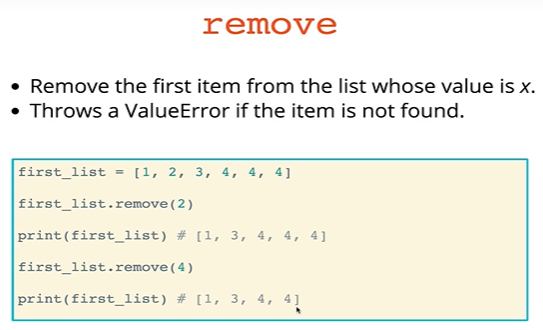
* The list.clear() method removes ALL items from the list at once. It is among the least-frequently used list methods



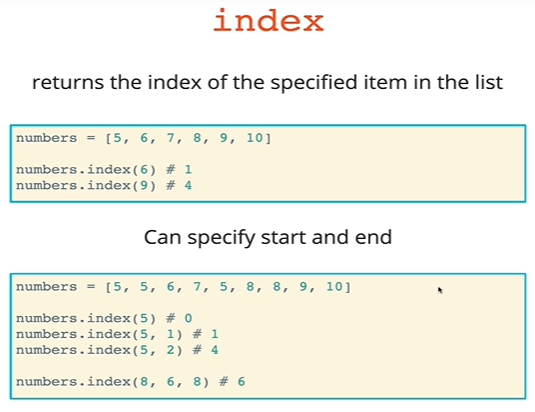
* + Like any other function, the parentheses are required, even if they are empty. We’ll see why later
* The list.pop(*index*) method takes as an argument the index of the item that we want to remove from the list, then **returns the removed item**. If an index is not given, the last item is removed from the list by default
  + The removed item is returned and can be captured, e.g. by saving it as a variable



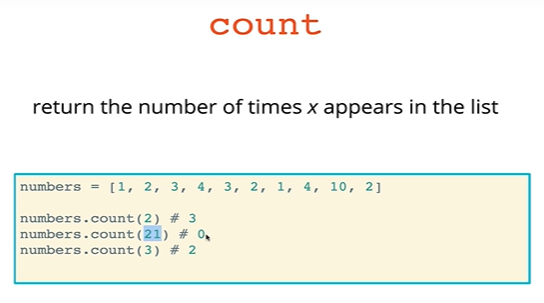
* The list.remove(*item*) method takes an argument the value of the item you want to remove from the list. It will then find the **first** instance of that item in the list and remove it
  + A ValueError will be thrown if an item with the value is not in the list
  + Important: it only removes the FIRST instance of that item
  + The remove() method does NOT return the removed item because you already know what the value of that item is



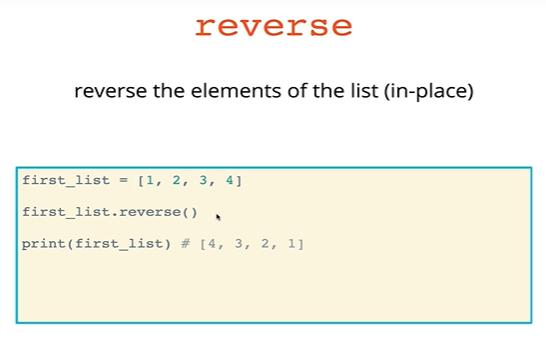
* The list.index() method allows us to find the index of a particular item on the list
  + If an item is present multiple times in the list, by default the index() method will return the index of the first instance of that item
  + But you can also provide arguments for the list.index(*item, start point index*, *end point index*) to define where in the list you want it to look



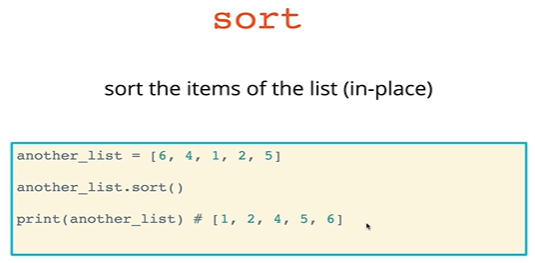
* The list.count(item) method counts the number of times the item appears in the list
  + If the item is not in the list, it will return 0



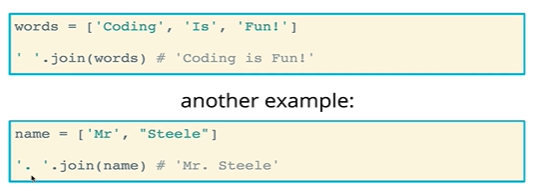
* The list.reverse() method reverses the entire list in place
  + The original list is completely replaced by the reversed version. It does not create a copy



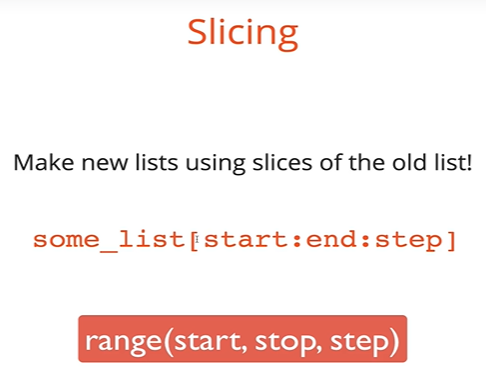
* The list.sort() method, without any arguments, sorts a list in ascending order
  + This may be numerical or alphabetical order
    - Remember that capital letters are higher than lower-case letters
  + Later we will learn how to modify sort to perform custom sorting duties



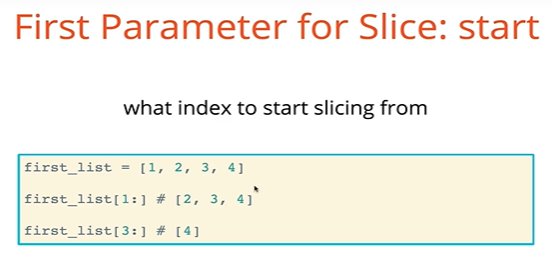
* The string.join([list of strings]) method is actually not a list method, but a string method. It takes as an input a list of strings, then joins (concatenates) them together while inserting the string that the join() method is called on. The new string is then returned.

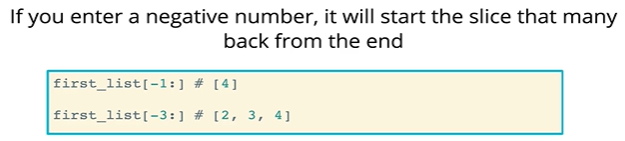


* List slicing allows us to make **copies** of entire lists, or continuous portions of lists
  + Slicing is not a method, but rather is done with square brackets on the list itself

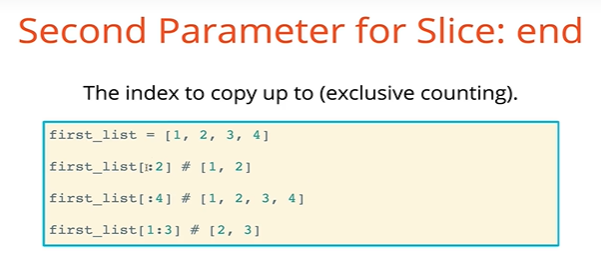


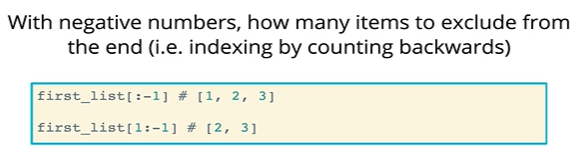
* + The first parameter provided in the slice is the **start** index.
    - Strictly speaking, you can just pass in a colon (:) into the square brackets and it will return a copy the entire list. It is the same as entering list[0:]
    - You can also slice with negative indices.



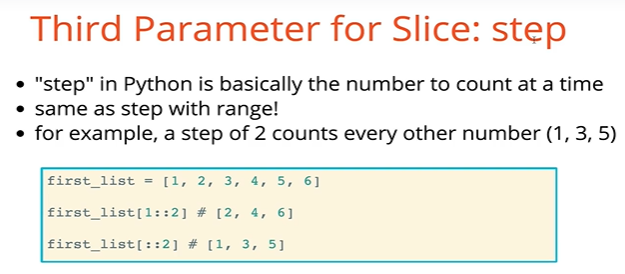


* + The second parameter of interest for slicing is the **end** parameter. This parameter instructs where to STOP the slice. However, it is exclusive, meaning that the **end** index will not be included in the slice
    - Negative numbers can also be used for the end parameter. Entering a negative **end** value will tell you up to how many values to exclude from the end
      * list[:-1] will exclude the last item in the list
      * list[:-3] will exclude the last three items in the list

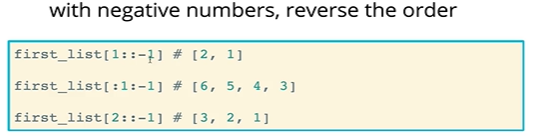




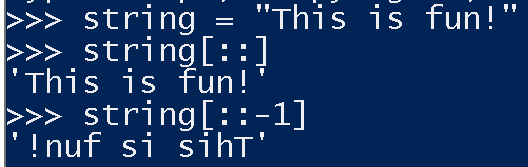
* The last parameter in slicing is the **step** parameter. This works the same way that the **step** parameter works in the range() function.



* + Steps can also be negative! Using a negative step value will count from the **start** position to the **end** position in the *reverse direction*. Therefore, it usually makes sense to have your start position as a higher index than your end position
    - If you do not provide a start position, then by default the count will start at the last position of the list and work in reverse toward the end position (exclusive of the actual end position)
    - If you do not provide an end position, the count will begin at the start position and move in reverse toward the first position of the list



* Neat tricks with slicing
  + Reverse an entire string



* + Modifying specific portions of lists

