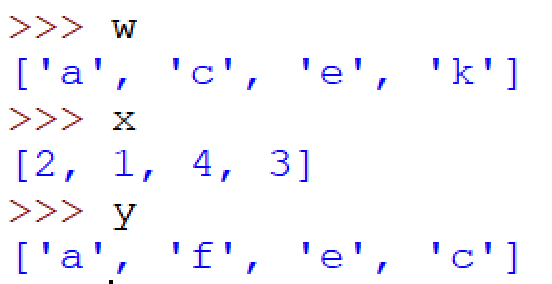
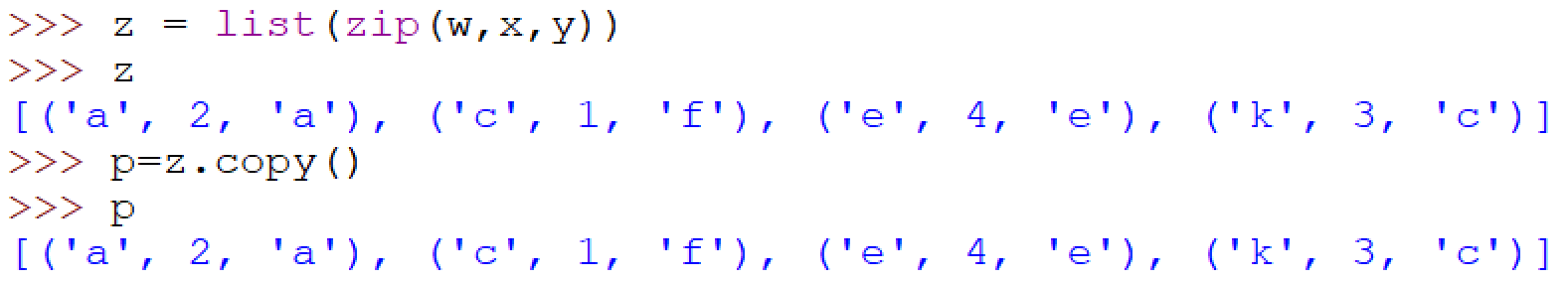
**Anonymous Functions**

Small transient functions are often the basis for data filtering and sorting in Python. We create small functions *on-the-fly* using the “**lambda**” keyword and use the built-in **filter()** and **sorted()** functions. **lambda** is a tool (like **def**) that is used to build functions. **lambda** is generally reserved for small, in-line functions that do not have a formal name, are usually temporary, and may be used in only one place in your code. Revisit **zip()** , **sort()** , and **sorted()** functions with **lambda** in mind as you read the material below.

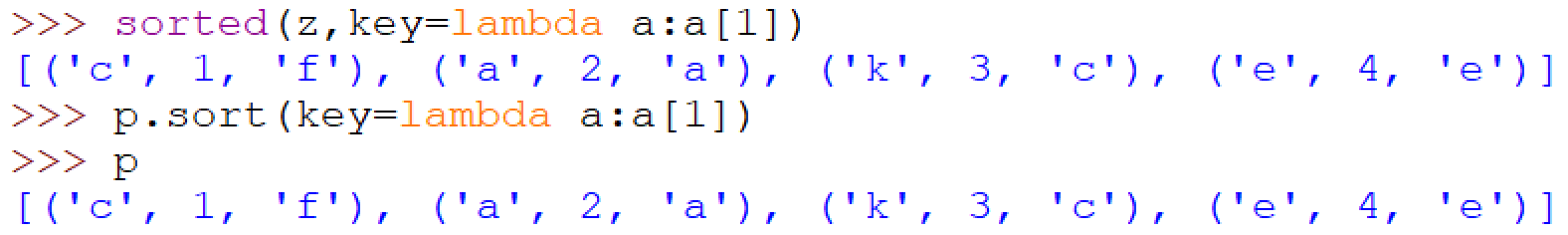
**Zip, sort, list comprehension, and scrambled words.**

Consider the three lists **w**, **x**, **y** shown to the right. List **w** and list **y** contain characters and list **x** contains integers. These lists were created to jumble common English words. List **x** is the key that puts the letters in the correct order when the three lists are combined and sorted.

In the console session below, the three lists are zipped together. List **z** is a list of tuples. The items in the tuples come from the original **w**, **x** and **y** lists. The original order of the lists is retained. Then I copy **z** to create list **p**. List **p** is just a copy of **z**.

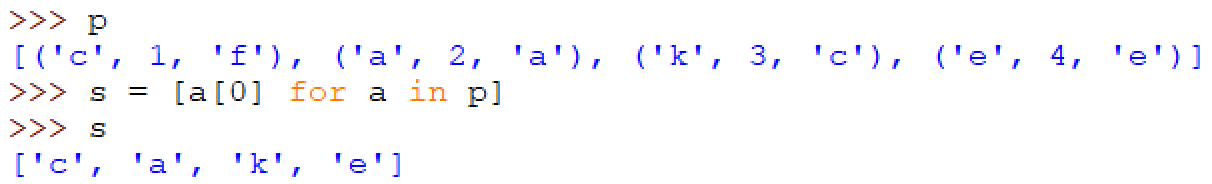


Now we can put the lists in order by sorting the zipped list. Recall that Python provides two ways to sort. Each list has a **sort()** method that changes the order of the original list (called sorting in-place). But Python also has a built-in method called **sorted()** that will assign a sorted version of the list parameter to another variable. By default, these lists perform the sort based on the items at index 0. In the case of our zipped lists, the item represented at index 0 of each element of **z** is a character from list **w**. We, however, want to sort on the integers from list **x**. For each tuple in **z**, the integer is at index 1. We need to provide a key for the sort that returns the value at index 1 for each tuple in the list. Now that we know about **lambda**, we can put the function that provides the key directly in the statement. The code below shows how **lambda** can be used in this situation. The statement, “**lambda a: a[1]**” means: “for item a, return the item at index 1”. When the **sorted()** function is used with list **z**, a new (sorted) list is created and **z** is unchanged. When the **sort()** method is used with list **p**, list **p** is changed to reflect the sort.

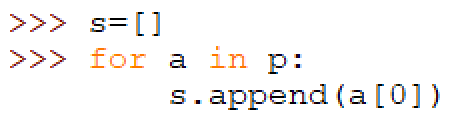


Now list **p** has been sorted and we want to extract the characters in order to create new lists that represent the unscrambled words. Maybe you have accomplished this in the past by initializing a new list and appending items from the old list to the new one in a for loop. But there’s another, briefer, way that accomplishes the same thing using a construct called a “list comprehension”. The simplest form of list comprehension has this structure: **new\_list = [** *expression* **for old\_item in old\_list]**. Here, one item defined by *expression* will be appended to **new\_list** as **old\_item** iterates through **old\_list**.

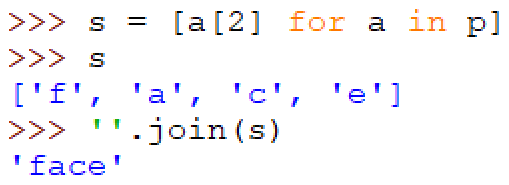
The list comprehension concept may be easier to grasp if you examine an actual example like the one below. Here, list **s** is created by list comprehension. Variable **a** traverses through **p**. Each value of **a** is a tuple. The list comprehension returns **a[0]** which is appended to **s**. In the end, “**c**”, **”a**”, ”**k**” and **“e**” are appended to **s**. In this order, the characters spell the word “**cake**”.



The list comprehension, **s = [a[0] for a in p]** is a more concise way to do the work of this sequence of Python statements:



We can do a similar thing to reveal the other encoded word:



The code above also uses the string method **join()** to create a string from the individual strings in list **s**. In the case above, **join()** is a method associated with a null string.

**ASSIGNMENT:**

**Complete the three scramble challenges (scramble, double scramble, super scramble). Use the Python templates provided. They contain the scrambled words and the code template you must use. Do not hard-code your solutions. Upload your solutions to Moodle. As always, your code must be fully commented and Pythonic to receive full credit.**