Lists in Python

In general, we can define a list as an object that contains multiple data items (elements). The contents of a list can be changed during program execution. The size of a list can also change during execution, as elements are added or removed from it. We have seen several examples of lists in Python. Sounds are lists of sample objects. Pictures are lists containing lists of pixel objects. When we use a for loop, we iterate over a list of items. We have typically used lists of pixels, lists of samples, or lists of integers within a certain range in our for loops.

We will now look at how we can create lists and how we can work directly with lists. Most of our previous experience with lists has been using functions to return lists to us.

Examples of lists:

```python
codes = [1, 2, 3, 4, 5]
names = ["Pam", "Sue", "Dan", "Sandi"]
info = ["Jordan", 123876, "Jr"]
```

Notice that square brackets are used to enclose the elements in a list, and the elements themselves are separated by commas.

Lists can be created using the repetition operator, *. We are accustomed to using the * symbol to represent multiplication, but when the operand on the left side of the * is a list, it becomes the repetition operator. The repetition operator makes multiple copies of a list and joins them all together. The general format is:

```
list * n,
```

where list is a list, and n is the number of copies to make.

Examples (you could try these on the command line in JES)

```python
>>> numbers = [0] * 5
>>> print numbers
[0, 0, 0, 0, 0]
```

In this example, [0] is a list with one element, 0. The repetition operator makes 5 copies of this list and joins them all together into a single list.

```python
>>> moreNumbers = [1, 2, 3] * 3
>>> print moreNumbers
[1, 2, 3, 1, 2, 3, 1, 2, 3]
```

Aside: Lists vs. Arrays

Many programming languages allow you to create arrays, which are objects similar to lists. Since lists serve the same purpose as arrays and have many more built-in capabilities, traditional arrays cannot be created in Python.
In this example, \([1, 2, 3]\) is a list with three elements, 1, 2, and 3. The repetition operator makes 3 copies of this list and joins them together into a single list. When we used sound functions such as `getSampleValueAt` and `setSampleValueAt` we were using *indices* to get and set the value of particular samples (elements) from our sound. (Remember, a sound is a list of sample objects.) Each element of a list has an index that specifies its position in the list. The index of the first element of a list is 0, the index of the second element is 1, and so on. The index of the last element of the list is 1 less than the number of elements in the list. If, for example, we have 5 elements in our list, the indices would be 0, 1, 2, 3, 4.

We can directly modify our list by using the index of the element we want to change. For example, suppose we have the list `names = [“Pam”, “Sue”, “Dan”, “Sandi”]` and we want to change the third element of the list to be “John” instead of “Dan”. We can do this with the following statement:

```python
```

The list then becomes `[“Pam”, “Sue”, “John”, “Sandi”]`. The general format for accessing individual elements of a list is:

```python
list[index],
```

where `list` is the name of the list and `index` is the position of the element in the list to be accessed.

**Example:** A function that will create a list to store the sums of the values in two lists of equal length.

```python
def sumLists(list1, list2):
    newList = [0] * len(list1) # get the values from the two lists
    # store the sum in the new list
    for index in range(len(list1)):
        newList[index] = list1[index] + list2[index]
    return newList
```

Notice that in lines (1) and (2) we use the function `len`, a built-in Python function that returns the number of elements in the list.

**Exercise:** Write a function, `reverseList`, that takes a list as input and returns a new list that is the reverse of the original list.

**Exercise:** Write a function, `concatenateLists`, that takes two lists as input and returns a new list that contains all of the elements of the first list followed by all of the elements of the second list. For example,
```python
>>> list1 = [1, 2, 3, 4, 5]
>>> list2 = [6, 7, 8, 9]
>>> list3 = concatenateLists(list1, list2)
>>> print list3
[1, 2, 3, 4, 5, 6, 7, 8, 9]
```

**Aside: Concatenating Lists**

In Python we can actually use the `+` operator to concatenate lists. In the example given in the exercise above, we could use the statement

```python
list3 = list1 + list2
```

to get list3 as the list `[1, 2, 3, 4, 5, 6, 7, 8, 9]`. For good practice working with lists, the function you write in the previous exercise should NOT use the `+` operator to concatenate the lists.