

## CS 161 Lab Activity: Lists

Enter these statements in Python Interactive (Idle) and observe what happens. Try to make sense of what you see based on the material covered in class.

```
>>> aList = []*10
>>> print aList

>>> aList = [0] * 10
>>> print aList

>>> for i in range (10):
        aList [ i ] = 2 * i

>>> print aList

>>> aList [ 1 ] = 100
>>> print aList

>>> aList [10] = 200

>>> for i in aList:
        print i
```

Notice the different format of the **for** header above

```
>>> for i in aList:
        print i,          # note the comma at the end

>>> anotherList = aList
>>> print anotherList

>>> anotherList [0] = 99
>>> print anotherList

>>> print aList
```

Uh-oh! Two names for the same space in memory!

```
>>> bList = [ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 ]
>>> for i in range (10):
        bList [ i ] = aList [ i ]

>>> print bList

>>> bList [ 0 ] = 1
>>> bList [ 1 ] = 1
>>> bList [ 2 ] = 1
>>> bList [ 3 ] = 1
>>> print bList
>>> print aList
```

```
>>> listLength = len (aList)
>>> print listLength
```

```
>>> aList.append (22)
>>> print aList
```

**append** is a python function that adds a value to the end of the list—it is not standard in most languages

```
>>> print len (aList)
```

```
>>> cList = aList + bList # what do you expect will happen here?
>>> print cList
```

```
>>> def someFunction ( a ):
    for i in a:
        i = 0
```

```
>>> someFunction (bList)
>>> print bList
```

```
>>> def theFunction ( a ):
    for i in range (len(a)):
        a [i] = 0
```

```
>>> theFunction (bList)
>>> print bList
```

Why do we see different result with these two function calls?  
Try this:

```
>>> for i in aList:
    i = i * 10
    print i
```

```
>>> print aList
```

The **for** structure isn't using the values in the array—it's just copying them, one by one, into the variable **i**.