1 Administrative Topics

- We go over the HW.
- We take the quiz.

2 Lists of Lists

Let’s look at how the operators work on lists, when they contain other lists. Consider this example:

```python
georgeList = [ 'George', 8, 47, 
               [ 'Harry Potter Book 1',
                 'Harry Potter Book 2', 
                 'Harry Potter Book 3']]
```

- `georgeList[0]` ⇒ ‘George’
- `georgeList[3][−1]` ⇒ ‘Harry Potter Book 3’

Note the “double indexing” in the last example above. First we access the book list using `georgeList[3]`, then we index within the resultant sublist with the [−1] statement.

Note that we could repeat everything with value rather than the symbol and we would get the same results. This is true because the operations are performed on the value itself – symbols just let us get to the values.
2.1 Be Careful with Sublists

Remember that lists are objects and that “copying” a list merely results in the copy of the reference (arrow) to the list object.

If we were to execute the line

```python
booklist = georgeList[3]
```

Then, booklist would point to the exact same list as georgeList[3]. In other words, if I were to change an item of booklist, georgeList would be able to “see” that change.

```python
booklist[0] = "Charlotte's Web"
print georgeList
```

results in the output:

```python
[ 'George', 8, 47, ["Charlotte's Web", 'Harry Potter Book 2', 'Harry Potter Book 3']]
```

Likewise, using an operator like this:

```python
gl2 = list * 2
```

(which doubles the list and puts it into a variable named gl2) performs what is called a shallow copy, meaning sublists themselves aren’t copied – just their references (arrows) are copied. In other words, both book sublists in gl2 will refer to the same list. Beyond that, they will both refer to the list referenced by booklist and georgeList[3].

The moral of the story is that when you are writing code with lists of lists, make sure you understand exactly how many lists are in memory.