1 Administrative Topics

- Stephanie didn’t teach today. These notes are from Spring 2011

2 Lists

Let’s compare lists to strings:

A string is an immutable sequence of characters.

A list is a mutable sequence of values of any type (including lists!).

2.1 Creating Lists

We have seen that the range function returns a list. We can also create a list explicitly, like this:

\[ \text{myList} = [0,1,2,3] \]

2.2 List Operations

The same operators that applied to strings apply to lists, with additional operators which mutate the list (basically, we can have a slice operation on the left hand side of an assignment statement).
There are several operators that shown in Table 1.

Table 1: List Operations

<table>
<thead>
<tr>
<th>Operation</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>(x \in s)</td>
<td>True if an item of (s) is equal to (x), else False</td>
</tr>
<tr>
<td>(x \not\in s)</td>
<td>False if an item of (s) is equal to (x), else True</td>
</tr>
<tr>
<td>(s + t)</td>
<td>the concatenation of (s) and (t)</td>
</tr>
<tr>
<td>(n \times s)</td>
<td>(n) shallow copies of (s) concatenated</td>
</tr>
<tr>
<td>(s[i])</td>
<td>(i)th item of (s), origin 0</td>
</tr>
<tr>
<td>(s[i:j])</td>
<td>slice of (s) from (i) to (j) (but not including the (j)th item)</td>
</tr>
<tr>
<td>(s[i:j:k])</td>
<td>slice of (s) from (i) to (j) with step (k) (but not including the (j)th item)</td>
</tr>
<tr>
<td>(\text{len}(s))</td>
<td>length of (s)</td>
</tr>
<tr>
<td>(s[i] = x)</td>
<td>item (i) of (s) is replaced by (x)</td>
</tr>
<tr>
<td>(s[i:j] = t)</td>
<td>slice of (s) from (i) to (j) is replaced by the contents of the iterable (t)</td>
</tr>
<tr>
<td>(s[i:j:k] = t)</td>
<td>the elements of (s[i:j:k]) are replaced by those of (t)</td>
</tr>
</tbody>
</table>

When indexing into a list, it is acceptable to use negative integers. -1 indicates the last item in the last. -2 indicates the penultimate item in the list, etc.

When using non-negative integers to index, remember that the first item is at index 0, the second item is at index 1, etc.

Here are some examples illustrating the operations (with the command typed at the Python prompt and the result that is displayed) assuming the following line has been executed:

```python
georgeList = ['George', 2, 24.2, ['The Giant Jam Sandwich', 'Cars and Trucks and Things that Go', 'Goodnight Moon']]
```

- \(\text{georgeList[0]} \Rightarrow 'George'\)
- \(\text{georgeList[3][-1]} \Rightarrow 'Goodnight Moon'\)
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.

2.3 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] = "Noah’s Ark"
print georgeList
```

results in the output:

```python
['George', 2, 24.199999999999999, ['Noah’s Ark', 'Cars and Trucks and Things that Go', 'Good night Moon']]
```

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....
2.4 Lists Methods

One of the most useful list methods is `append`, which appends an item to the end of a list. Its syntax, along with that of several other list methods is in Table 2.

<table>
<thead>
<tr>
<th>Operation</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>s.append(x)</code></td>
<td>append the item <code>x</code> to the end of list <code>s</code></td>
</tr>
<tr>
<td><code>s.extend(t)</code></td>
<td>append each item of list <code>t</code> to the end of list <code>s</code></td>
</tr>
</tbody>
</table>

For a complete list of list methods, see http://docs.python.org/library/stdtypes.html#mutable-sequence-types