Lab 5: Lists in Python
Due Thursday, 15 October, 8 a.m., via Git

Details: Most of today’s lab is a tutorial on Python and lists. **What’s the big deal about lists?** Lists are perhaps the most commonly-occurring data structure in computer programming. Languages such as Python, Perl, Ruby, R, and many others use lists extensively, and the Lisp-like languages (including Scheme, Racket, Clojure, etc.) are almost completely based on the notion of a list. Nearly every other data structure can be encoded using a list.

Running Python

To run Python in interactive mode, just type “python”. (NOTE: we are using version 2.7.6. Python 3 is also installed (to run it, you must type “python3”). However, there are differences in things like the “print” statement, so programs written in Python 2 may not work in Python 3 and vice-versa.

To exit, just type “quit()” or press CTRL-D.

I suggest you try all of the examples below.

Basic Indexing

In Python a list is a comma-separated sequence of items enclosed in square brackets. Items can be of any type (including other lists). For instance,

```python
>>> list1 = []
>>> list2 = [1, 2, 3, 'hello, world', True, False, 3.14159, (3+2j)]
>>> list3 = [[1, 2, [3, 4], [[[5]]]]]
>>> list4 = [[[[]], [[]], []], []]
```

We can access the elements of a list using the usual array indexing, e.g.,

```python
>>> list2[2]
3
>>> list2[6]
3.14159
>>> list3[0]
[1, 2, [3, 4], [[[5]]]]
```

If a list item is itself another list, we can index that, too. For instance, in the above examples, `list3[0]` is a list, so we can do things like:
>>> list3[0][2]
[3, 4]
>>> list3[0][2][1]
4

Lists can be indexed from either the left or the right; when indexing from the right the rightmost position is -1; the next-to-last element is -2; etc.

>>> list2 = [1, 2, 3, 'hello, world', True, False, 3.14159, (3+2j)]
>>> list2[-1]
(3+2j)
>>> list2[-2]
3.14159
>>> list2[-3]
False

Slices

We can index “slices” of a list using the “start:end” notation to access element in positions start, start+1, start+2, ..., end−1. **NOTE:** the last index in the slice is end−1, not end. You can use either positive or negative indices; either start or end (or both) may be omitted, in which case the slice extends to the left or right end of the list.

```python
>>> list2
[1, 2, 3, 'hello, world', True, False, 3.14159, (3+2j)]
>>> list2[1:5]
[2, 3, 'hello, world', True]
>>> list2[1:-1]
[2, 3, 'hello, world', True, False, 3.14159]
>>> list2[-6:-3]
[3, 'hello, world', True]
>>> list2[:6]
[1, 2, 3, 'hello, world', True, False]
>>> list2[-4:]
[True, False, 3.14159, (3+2j)]
>>> list2[:]
[1, 2, 3, 'hello, world', True, False, 3.14159, (3+2j)]
```

You can also specify a “step size” as part of a slice by adding another colon and an integer. For example,

```python
>>> list2[1:6:2]
[2, 'hello, world', False]
>>> list2[:,:3]
[1, 'hello, world', 3.14159]
```
List Operators, Functions, and Methods

The “+” operator concatenates two lists into one:

```python
>>> a = [1,2,3]
>>> b = ['x','y','z']
>>> c = a+b
>>> c
[1, 2, 3, 'x', 'y', 'z']
```

The “*” operator takes a list and an integer and performs multiple concatenations of the list with itself:

```python
>>> 3*a
[1, 2, 3, 1, 2, 3, 1, 2, 3]
>>> b*2
['x', 'y', 'z', 'x', 'y', 'z']
>>> 0*a
[]
>>> b*-3
[]
```

The “in” operator takes an item and a list and returns `True` if the item is in the list, `False` otherwise. (NOTE: it does not look recursively at sub-lists.)

```python
>>> d = [10,20,30,[40,50]]
>>> 10 in d
True
>>> 40 in d
False
>>> 40 in d[3]
True
>>> [40,50] in d
True
```

Here are some of the functions in Python that can operate on lists:

```python
>>> len(a)  # length of a list
3
>>> len(list2)
8
>>> len([])
0
>>> x = [1,3,4,8,-1]
>>> sum(x)  # sums element; only for numeric lists!
15
```
>>> max(x)  # max and min--only for items that can be ordered
8
>>> min(x)
-1
>>> y = ['hello', 'apple', 'microsoft', 'bye']
>>> max(y)
'microsoft'
>>> min(y)
'apple'

And here are some methods that belong to the class of lists. These change the value of the list as described in the comments.

>>> p = [1,2,3]
>>> q = ['a','b','c']
>>> p.reverse()  # reverses the list
>>> p
[3, 2, 1]
>>> p.sort()  # sorts the list
>>> p
[1, 2, 3]
>>> p.extend(q)  # concatenates a list onto p
>>> p
[1, 2, 3, 'a', 'b', 'c']
>>> p.append([10,20])  # adds an element to the end
>>> p
[1, 2, 3, 'a', 'b', 'c', [10, 20]]
>>> p.pop(4)  # removes and returns element at index 4
'b'
>>> p
[1, 2, 3, 'a', 'c', [10, 20]]
>>> p.remove('c')  # removes an element
>>> p
[1, 2, 3, 'a', [10, 20]]
>>> p.pop()  # "pop" with no argument pops the last item
[10, 20]
>>> p
[1, 2, 3, 'a']
>>> p.insert(0,'hello')  # insert an element at a given index
>>> p.insert(2,'goodbye')
>>> p
['hello', 1, 'goodbye', 2, 3, 'a']

Lists are reference objects; assignments are references to a list. Thus, aliasing can occur where two variables point to the same list:
>>> a = [10,8,3,2]
>>> b = a
>>> b.sort()
>>> a
[2, 3, 8, 10]

Miscellaneous Remarks

Here are two more ways to create lists:

>>> r1 = range(10) # list with values 0 through 9
>>> r1
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> r2 = range(5,16) # list with values 5 through 15
>>> r2
[5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15]
>>> r3 = range(10,30,3) # values 10 through 29 in steps of 3
>>> r3
[10, 13, 16, 19, 22, 25, 28]
>>> c1 = [i*i for i in range(1,11)] # parameterized expression
>>> c1
[1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
>>> c2 = [2*i/5.0 for i in range(-5,6)]
>>> c2
[-2.0, -1.6, -1.2, -0.8, -0.4, 0.0, 0.4, 0.8, 1.2, 1.6, 2.0]

Here's how to loop over the elements in a list:

for x in c2:
    print x,"/10 =",x/10.0

Essential Python

This is intended to be “just enough” Python to enable you to write functions that manipulate lists.

Instead of using curly braces “{...}” to denote blocks, Python uses indentation:
There are `for` and `while` loops:

```python
>>> for j in range(5): # note the colon
...   for k in range(j): # note the indentation
...       print j,"","",k
...
1 , 0
2 , 0
2 , 1
3 , 0
3 , 1
3 , 2
4 , 0
4 , 1
4 , 2
4 , 3
>>> a = [3,2,1,4,4,7,4,2,2,4]
>>> while 4 in a:
...     a.remove(4)
...     print a
...
[3, 2, 1, 4, 7, 4, 2, 2, 4]
[3, 2, 1, 7, 4, 2, 2, 4]
[3, 2, 1, 7, 2, 2, 4]
[3, 2, 1, 7, 2, 2]
```

To define a function:

```python
>>> def add1(x): # add 1 to every element of x
...   for i in range(len(x)):
...       x[i]+=1
...
>>> r = 4*[1,2,3]
>>> r
[1, 2, 3, 1, 2, 3, 1, 2, 3, 1, 2, 3]
>>> add1(r)
>>> r
[2, 3, 4, 2, 3, 4, 2, 3, 4, 2, 3, 4]
>>> def echo(x): # return a new list with each x element repeated twice
...   y = []
...   for i in x:
...       y.extend([i,i])
...   return y
...
>>> z = echo(['a','b',True,3.14,[]])
```
>>> z
['a', 'a', 'b', 'b', True, True, 3.14, 3.14, [[]], [[]]]

To print:

>>> print 'z =', z
z = ['a', 'a', 'b', 'b', True, True, 3.14, 3.14, [[]], [[]]]

>>> print 'first half of z:', z[0:len(z)/2]
first half of z: ['a', 'a', 'b', 'b', True]

1. Write a Python function that takes a list and an item and returns the number of occurrences of that item in the list (possibly zero). Example:

>>> a = [1,2,4,3,2,4,5,3,2,2,2,2,2]
>>> print count(a,2)
6
>>> print count(a,1)
1
>>> print count(a,4)
2
>>> print count(a,6)
0

2. Write a Python function that takes two lists and returns a list containing all elements that are in both lists (don’t worry about duplicates). Example:

>>> a = [3,2,5,4,6,7,4]
>>> b = [7,2,8,9,4]
>>> print common(a,b)
[2, 4, 7, 4]

3. Write a function that takes two lists and returns a list containing all elements that are in one or the other list, but NOT both.

4. Write a function that takes a list and an integer k and rotates the list k positions to the left.
   (If k is negative, rotate to the right by -k positions.)

Save your functions in a single file, comment them, and upload the file to your repository.