Lists
(and the end of File I/O)

CS303E: Elements of Computers and Programming

July 25, 2012

Files are Objects

- *Objects* are a grouping of data and operations that can be performed on that data
  - The operations are essentially functions, but in this context they are called *methods*
- To call a method on an object:

```
objectName.methodName()
```

Examples

```
fileVar.read()
fileVar.open()
stringVar.upper()
stringVar.find(sub)
```

iClicker Question

When you finish using a file, it is important to close it.

A. True

B. False

Files as Parameters

- Recall that the file object maintains state for you, so that each time you call `readline()` you receive the next line
- This state is maintained even as the file is passed into functions as a parameter
- AND changes to that state are reflected in the calling function after the call

Lists

- A *list* is an ordered collection of elements
  - Numbers, strings, objects (such as files, other lists, …)
  - Elements may appear more than once
- Lists are *mutable*
  - Their elements can be modified
  - Remember that strings are immutable
- Lists are objects

Example List

```
favs=[“Charlie”,3,”lemon pie”,
 math.pi]
```

Then, if you say:

```
>>>print favs
```

Output is:

```
[“Charlie”, 3, “lemon pie”,
 3.14159265]
```

Lists vs. Arrays

- Many other languages have arrays
- Lists are Python’s version of the array
- Arrays differ from lists in two important ways:
  - They are ordered collections of objects *of the same type*
  - Their size is chosen when they are created and *cannot be changed*

Creating a List

- Create a list by enumerating the objects in the list

Example:

```
numbers = [2,3,5]
words = [“hi”,”hello”,”hey”]
```

Modifying a List: Appending

- Adds an item to the end

Example:

```
myList = []        #create
                   #empty list
myList.append(7)  #[7]
myList.append(11) #[7,11]
```

Modifying a List: Inserting

- Can insert items into specific locations

```
myList.insert(<location index>, <item>)
```

Example:

```
myList = [9,2,1]
myList.insert(1,3)    #[9,3,2,1]
```

Modifying a List: Concatenating Existing Lists

- Guess!

Modifying a List: Concatenating Existing Lists

- The + operator!

Example:

```
a=[1,2]
b=[3,4]
c=a+b      #c=[1,2,3,4]
```

Modifying a List: Repetition

- Guess!

Modifying a List: Repetition

- The `*` operator!
- Repeats the contents of the list the specified number of times

Example:

```
a=[0,1]
a=a*3      #[0,1,0,1,0,1]
```

iClicker Question

How do lists differ from strings?

A. Strings are mutable, lists are immutable
B. Lists are mutable, strings are immutable
C. Both are mutable
D. Both are immutable

2-Dimensional Lists

- Lists can contain lists, so you can create a 2-dimensional list like this:

```
list_2D=[[1,2,3],[4,5,6],[7,8,9]]
```

Picture it like this:

```
 [[1,2,3],
   [4,5,6],
   [7,8,9]]
```

2-Dimensional Lists: Another Way

```
a=[1,2,3]
b=[4,5,6]
c=[7,8,9]
list_2D = [a,b,c]
```

Picture it like this:

```
 [[1,2,3],
   [4,5,6],
   [7,8,9]]
```

Basic List Operations

- Length, indexing, slicing, and traversing
- Performed on lists same way as on strings
- But lists and strings differ! How?
  - Strings are *immutable*
  - Lists are *mutable*

List Operations: Length

The `len()` function

Example:

```
myList = ["a","b","c"]
print len(myList)
```

Output:

```
3
```

List Operations: Indexing

- Again, indices are from 0 to length-1.
- Negative indices also valid

```
a=[4,2,7]
x=a[0]      #x=4
y=a[-1]     #x=7
a[1]=10     #a=[4,10,7]---not
            #valid for strings!
```

List Operations: Indexing a 2D list

- Specify first the row, then the column

Example:

```
list_2D=[[1,2,3],[4,5,6],[7,8,9]]
x=list_2D[0][0] #x is 1
y=list_2D[1][2] #y is 6
```

iClicker Question

- What does it mean for an object to be mutable?

A. It can be changed

B. It cannot be changed

List Operations: Slicing

- Gets a sub-list
  - Recall that with strings it gets a substring

`myList[start:end]`

- Gets values of my list from *start* up to, but not including, *end*
- As with strings, can omit *start* or *end*

Slicing Examples

```
a = [4,10,7,3]
b = a[1:3]      #b = [10,7]
c = a[1:]       #c = [10,7,3]
d = a[:-2]      #d = [4,10]

Also
a[2:4] = [8,5] #a = [4,10,8,5]
```

List Operations: Traversing a List

- Visit every element in order

```
list1 = [2,3,6,0]
for i in list1:
  print i

for i in range(len(list1)):
  print list1[i]
```

Example

Write a segment of code that creates a list with the elements 1, 13, 2, and 6. It should then use a loop to double each element.

List Operations: Membership

- Checks if a value is in the list, returns a Boolean indication the answer

Example:	Output:
`a=[1,2,3]`	`yes!`
`if 2 in a:`	
`  print "yes!"`	

More List Operations

Operation	Description
`list.sort()`	Sorts the entries in ascending order
`list.reverse()`	Reverses the order of the list items
`list.index(x)`	Returns the index of the first occurrence of x in the list. *Error if x does not occur in the list.*
`list.count(x)`	Returns the number of occurrences of x in list
`list.pop(i)`	Deletes and returns the item at index i in the list
`list.remove(x)`	Removes first occurrence of x from the list
`del list[i:j]`	Removes items from index i to index j-1 (Same as list[i:j] = [])
`min(list)`	Returns the smallest list item
`max(list)`	Returns the largest list item

Exercise

Using the IMDB list of top movies, "TopMoviesIMDB.txt", find the number of titles that have "cat" or "dog" in them. Print each title to the file "petMovies.txt".