14. Lists of Strings

Topics:
- List methods-again
- Setting up a list of strings
- Reading from a textfile
- .csv files
- delimiters and the method split

What You Know About Lists of Numbers Carries Over

Set-Up:
```
s = ['cat','dog','mice']
```
Length:
```
L = len(s)
```
Slicing:
```
t = s[1:3]
```
Methods:
```
append, extend, insert, pop,
count, sort
```

A Note About sort

Before: s: | 'dog' | 'mouse' | 'cat' |

```
s.sort()
```

After: s: | 'cat' | 'dog' | 'mouse' |

When you sort a list of strings and the strings are made up of letters, digits, and blanks, then it alphabetizes the items according to the order in this string:

' 0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz'

Strings vs Lists of Characters

```
LC = 'abcdefghijklmnopqrstuvwxyz'
UC = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ'
Digits = '0123456789'
All = LC + UC + Digits + ' '
s = []                          <-- empty list
for c in All:
    s.append(c)                 <-- repeated appending
s.sort()
lex = ''                        <-- empty string
for x in s:
    lex = lex + x               <-- repeated concatenation
print lex
```

' 0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz'

Note on sum

If
```
s = ['cat','dog','mice']
```
then
```
t = sum(s)
```
produces an error:

```
TypeError: unsupported operand type(s) for +:
           'int' and 'str'
```

not
```
'catdogmice'
```

Visualizing Lists of Strings

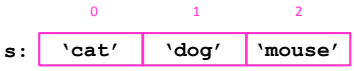
Informal:

	0	1	2
s:	'cat'	'dog'	'mouse'

Formal:

```
s ----> 0 ---> 'cat'
        1 ---> 'dog'
        2 ---> 'mouse'
```

Subscript "Reasoning"



The statement

```
c = s[2][1:3]
```

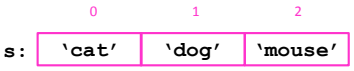
assigns 'ou' to c

The statements

```
t = s[2]
c = t[1:3]
```

assign 'ou' to c

Subscript "Reasoning"



How to decipher `s[2][1:3]`:

`s` names a list

`s[2]` names item 2 in `s`

`s[2][1:3]` names slice `1:3` in item 2 in `s`

Three Examples

1. A function that returns a list of random 3-letter words.
2. A function that reads a text file and returns a list where each item in the list is a line in the file.
3. A script that uses a US Census dataset to examine county population growth rates from 2010 to 2014.

1. A Function that Returns a List of Strings

Let's implement this:

```
def ListOfRandomStrings(n,m):
    """ Returns an alphabetized length-n
    list of random strings each of which
    is made up of m lower case letters.
    The items in the list are distinct.

    PreC: n and m are positive integers
    and n<=26**m. """
```

There are 26**m different possible strings. So n cannot be bigger than that.

ListOfRandomStrings(100,4)

Sample outcome:

```
afei atou atzo auvf bdus bmut bnhk btqp bztw cabs
cdnr chda dayy dhtb dinj drfq ecme eixm ethh evsv
frar gfam gssn gtnx gvmp hfhb hlwe ilsr inxs iolb
itzv izwd jfmc jtph jzai kefo keiy keyo kfft kwnu
kyoi lbgt ldgs ldrc luwn lvtg lynx medj mplc muzs
mvov nawk ngvb nkhp nogc npgc ntjk nwbt oefw oepg
pddo pewe phpp qapi qhal qmod qryd qwhj rmhk rorl
rvhu sauo sebg segl sknu slgk svsf tmry uake vinu
vlvx vygo wtoi wxmj xpcn xuni ypta yqxc yqzq ysny
ywsd yyut zayj zhym zqdn zsqf zvce zwgj zxog zyyp
```

Helper Function RandString

Assume the availability of…

```
def RandString(m):
    """ Returns a random length-m string
    consisting of lower case letters.

    PreC: m is a positive integer.
    """
```

If we allowed Repeats…

```
def ListOfRandomStrings(n,m):
    s = []
    for k in range(n):
        w = RandString(m)
        s.append(t)
    s.sort()
    return s
```

Repeat n times: Generate a random string and append

Check Before Appending…

```
def ListOfRandomStrings(n,m):
    s = []
    k = 0   # k is the length of s.
    while k<n:
        w = RandString(m)
        if w not in s:
            s.append(w)
            k+=1
    s.sort()
    return s
```

Notice how we can use "in" to look for values in a list. And "not in" to confirm the absence of a value in a list

Repeat: Generate a random string and append IF it is not yet in s

2. Reading a Text File Into a List of Strings

Text files can be visualized like this:

MyFile.txt

```
abcd
123 abc d fdd
xyz
3.14159   2.12345
```

This text file has four lines.

Our Plan

We will "read" the file line-by-line and make each line an item in a list of strings.

MyFile.txt

```
abcd
123 abc d fdd
xyz
3.14159   2.12345
```

```
L --> 0 ---> 'abcd'
      1 ---> '123 abc d fdd'
      2 ---> 'xyz'
      3 ---> '3.14159   2.12345'
```

Opening a File

```
L = []
with open('MyFile.txt','r') as F:
    for s in F:
        L.append(s)
```

The name of the file is passed as a string.

The file must be in the same working directory as the file-reading code.

'r' means "read"

The Reading

```
L = []
with open('MyFile.txt','r') as F:
    for s in F:
        L.append(s)
```

F is a "file object".

It can be read line-by-line with a for-loop.

As the loop executes, s takes on the value of each file line in succession.

Problem: Special Characters

New line characters and carriage return characters mess up this process:

MyFile.txt

```
abcd
123 abc d fdd
xyz
3.14159    2.12345
```

Typically these characters are irrelevant once the data is read into the list. So delete them…

Removing Newline and Carriage Return Characters

```
L = []
with open('MyText.txt','r') as F:
    for s in F:
        s1 = s.rstrip('\n')
        s2 = s1.rstrip('\r')
        L.append(s2)
```

```
>>> s = 'abc    '
>>> s.rstrip(' ')
'abc'
```

Putting It All Together

```
def fileToStringList(FileName):
    L = []
    with open(FileName,'r') as F:
        for s in F:
            s1=s.rstrip('\n')
            s2 = s1.rstrip('\r')
            L.append(s2)
    return L
```

Using `fileToStringList`

```
L=fileToStringvList('EnglishWords.txt')
for s in L:
    if len(s)>=5 and Palindrome(s):
        print s
```

`EnglishWords.txt` is a file with about 100000 lines, each containing a single English word.

`Palindrome` is a boolean valued function that is True if and Only if the input string is a palindrome

Using `fileToStringList`

```
L=fileToStringList('EnglishWords.txt')
for s in L:
    if len(s)>=5 and Palindrome(s):
        print s
```

Output:

```
civic      deified   deled    denned   dewed
hallah     kaiak     kayak    level    madam
malayalam  minim     radar    redder   refer
reifier    reviver   rotator  rotor    sagas
seres      sexes     shahs    solos    stats
stets      tenet
```

3. A More Complicated Example

Extracting words from **`EnglishWords.txt`** was easy because there was one data item of interest per line:

EnglishWords.txt

```
aarvaark
baby
cat
  :
```

Multiple Data Items Per Line

In more complicated set-ups, there can be multiple data items per row.

BigStates.txt

```
California  38.8
Florida  19.9
NewYork  19.8
Texas  27.0
```

Here, blanks are being used to separate items of interest.

The `split` Method is Handy for Extracting Data

```
>>> s = 'California  38.8'
>>> v = s.split()
>>> v
['California', '38.8']
```

A list is made up of substrings that are separated by blanks.

The blank acts as a delimiter.

The Comma-Separated Value Format

A more common strategy is to use commas as delimeters.

BigStates.csv

```
California,  38.8
Florida,  19.9
NewYork,  19.8
Texas,  27.0
```

The .csv suffix is used to signal this format in a text file.

Reading a .csv File

BigStates.csv

```
California,  38.8
Florida,  19.9
NewYork,  19.8
Texas,  27.0
```

File

```
L = FileToList('BigState.csv')
for c in L:
   v = c.split(',')
   print v[1],v[0]
```

Code

```
38.8 California
19.9 Florida
19.8 NewYork
27.0 Texas
```

Output

ReadMe Files

Whoever puts together a .csv file is obliged to tell you how the data is laid out.

BigStates.csv

```
California,  38.8
Florida,  19.9
NewYork,  19.8
Texas,  27.0
```

Field 1: State Name
Field 2: Population (millions)

This information is typically placed in a text file that is named ReadMe

An Example

Suppose we have a file **`CensusData.csv`** in which each line houses US Census data on a county. Assume that…

```
Field 6   State Name
Field 7   County Name
Field 8   2010 county population
Field 11  2011 county population
Field 12  2012 county population
Field 13  2013 county population
Field 14  2014 county population
```

Question: Which county that has a 2010 population greater than 100000 grew the most between 2010 and 2014?

An Example

```
Field 6   State Name
Field 7   County Name
Field 8   2010 county population
Field 11  2011 county population
Field 12  2012 county population
Field 13  2013 county population
Field 14  2014 county population
```

If c is a line in the file then

```
v = c.split(',')
growth = float(v[13])/float(v[8])
```

is what we want.

Solution Code

```
TheCounties =
     fileToStringList('CensusData.csv')
gMax = 0
for c in TheCounties:
    v = c.split(',')
    g = float(v[13])/float(v[7])
    if int(v[7])>=100000 and g>gMax:
        gMax = g
        vMax = v
print vMax[6],vMax[5],int(gMax*100),'percent'
```

Hays County Texas 117 percent