loops continued

Genome 559: Introduction to Statistical and Computational Genomics

Prof. James H. Thomas

Review

- Pick variable names that are descriptive
- Comment your code if it complex (# sign)

```
for <element> in <object>:
    <statement>
    <statement>
    . . .
    <last statement>
```

Use `for` loop to iterate over elements in a list, numbers, or characters in a string

```
while (conditional test):
      <statement1>
      <statement2>
      . . .
      <last statement>
```

Use `while` loop to run until some condition is met

Review

Increment operator

```
x += y      # adds y to the value of x
x *= y      # multiplies x by the value y
x -= y      # subtracts y from the value of x
```

Explicit program exit

```
sys.exit()   # exit program immediately
```

Use to terminate when something is wrong - best to use `print` to provide user feedback before exit

Smart loop use

Read a file and print the first ten lines

```
import sys
infile = open(sys.argv[1], "r")
lineList = infile.readlines()
counter = 0
for line in lineList:
    counter += 1
    if (counter > 10):
        break
    print line
infile.close()
```

Does this work?

YES

Is it ideal?

NO

What if the file has a million lines? (not uncommon in bioinformatics)

```
import sys
infile = open(sys.argv[1], "r")
lineList = infile.readlines()
counter = 0
for line in lineList:
    counter += 1
    if (counter > 10):
        break
    print line
infile.close()
```

this statement reads all million lines!!

How about this instead?

```
import sys
infile = open(sys.argv[1], "r")
for counter in range(10):
    line = infile.readline()
    print line
infile.close()
```

this version reads only the first ten lines, one at a time

This while loop does the same thing, just as efficiently:

```
import sys
infile = open(sys.argv[1], "r")
counter = 0
while counter < 10:
    line = infile.readline()
    print line
    counter += 1
infile.close()
```

- The original `readlines()` approach not only takes much longer on large files it also has to store ALL the data in memory.

- I ran original version and efficient version on a very large file.

- Original version ran for 45 seconds and crashed when it ran out of memory.

- Improved version ran successfully in the blink of an eye.

What if the file has fewer than ten lines?

```
import sys
infile = open(sys.argv[1], "r")
for counter in range(10):
    line = infile.readline()
    print line
infile.close()
```

hint - when `readline()` reaches the end of a file, it returns `""`

It prints a blank line repeatedly

Improved version:

```
import sys
infile = open(sys.argv[1], "r")
for counter in range(10):
    line = infile.readline()
    if len(line) == 0:
        break
    print line
infile.close()
```

added code, tests for end of file

Sequential splitting of file contents

Many problems in text or sequence parsing can employ this strategy:

- First, chop file content into chunks (lines or fasta sequences etc.)
- Second, extract needed data from each chunk
- This can even be repeated - split each chunk into subchunks, extract needed data from subchunks

```
import sys
lineList = open(sys.argv[1], "r").readlines()
for line in lineList:
        fieldList = line.strip().split("\t")
        for field in fieldList:
                <do something>
```

What does this do?

Sample problem #1

Write a program `read-N-lines.py` that prints the first N lines from a file, where N is the first argument and filename is the second argument. Use a while loop and be sure it handles short and long files.

```
>python read-N-lines.py 7 file.txt
this
file
has
five
lines

>
```

Solution #1

```
import sys
infile = open(sys.argv[2], "r")
max = int(sys.argv[1])
counter = 0
while counter < max:
    line = infile.readline()
    if len(line) == 0:   # we reached end of file
        break
    print line
    counter += 1
```

Sample problem #2

Write a program `count-fasta.py` that counts the number of fasta sequences in a file specified on the command line. Make sure it can run on a huge file (don't read the entire file content at once).

Fasta format:

```
>identifier1 comment comment comment
AAOSIUBOASIUETOAISOBUAOSIDUGOAIBUOABOIUAS
AOSIUDTOAISUETOIGLKBJLZXCOITLJLBIULEIJLIJ
>identifier2 comment comment
TXDIGSIDJOIJEOITJOSIJOIGJSOIEJTSOE
>identifier3
Etc.
```

sequence on any number of lines until next ">"

Two files are linked in News on the course web page – run your program on both: small.txt and large.txt

Solution #2

```
import sys
```

Not required, but a good habit to get into

```
# Make sure we got an argument on the command line.
if (len(sys.argv) < 2):
    print("USAGE: count-fasta.py file argument required")
    sys.exit()
```

```
# Open the file for reading.
fasta_file = open(sys.argv[1], "r")
lineList = fastaFile.readlines()
num_seqs = 0
for line in lineList:
    # Increment if this is the start of a sequence.
    if (line[0] == ">"):
        num_seqs += 1

print num_seqs
fasta_file.close()
```

Not so good - will run out of memory if file is huge

Alternative solution #2

```
import sys

# Make sure we got an argument on the command line.
if (len(sys.argv) < 2):
    print("USAGE: count-fasta.py file argument required")
    sys.exit()

# Open the file for reading.
fasta_file = open(sys.argv[1], "r")
wholeText = fastaFile.read()
print wholeText.count(">")
fasta_file.close()
```

Not so good - will run out of memory if file is huge

Improved solution #2

```
import sys

# Make sure we got an argument on the command line.
if (len(sys.argv) < 2):
    print "USAGE: count-fasta.py file argument required"
    sys.exit()

# Open the file for reading.
fasta_file = open(sys.argv[1], "r")
num_seqs = 0
line = fasta_file.readline()  # read first line
while len(line) > 0:
    # Increment if this is the start of a sequence.
    if line[0] == ">":  # or if line.startswith(">"):
        num_seqs += 1
    line = fasta_file.readline()  # read next line
print num_seqs
fasta_file.close()
```

Note - when `readline()` encounters the end-of-file (EOF) it returns `""` (empty string)

Challenge problem

Write a program `seq-len.py` that reads a file of fasta sequences and prints the name and length of each sequence and their total length.

```
>seq-len.py seqs.fasta
seq1 432
seq2 237
seq3 231
Total length 900
```

Here's what fasta sequences look like:

```
>foo
gatactgactacagttt
ggatatcg
>bar
agctcacggtatcttag
agctcacaataccatcc
ggatac
>etc...
```

('>' followed by name, newline, sequence on any number of lines until next '>')

One solution

this version may have problems with large files

```
import sys

filename = sys.argv[1]
myFile = open(filename, "r")
myLines = myFile.readlines()
myFile.close()                  # we read the file, now close it

cur_name = myLines[0]           # initialize required variables
cur_len = 0
total_len = 0
for index in range(1, len(myLines)):
   line = myLines[index]
   if (line.startswith(">")):    # we reached a new fasta sequence
       print cur_name, cur_len   # write values for previous sequence
       total_len += cur_len      # increment total_len
       cur_name = line.strip()   # record the name of the new sequence
       cur_len = 0               # reset cur_len
   else:                    # still in the current sequence, increment length
      cur_len += len(line.strip())
   index += 1
print cur_name, cur_len            # we need to write the last values

print "Total length", total_len
```

Lea came up with a far more elegant solution. Here is my version using Lea's method:

```
import sys
filename = sys.argv[1]
myFile = open(filename, "r")
whole_string = myFile.read()
myFile.close()
seqList = whole_string.split(">")
total_len = 0
for seq in seqList:
    lineList = seq.split("\n")
    length = len("".join(lineList[1:]))
    total_len += length
    print lineList[0], length
print "Total length", total_len
```

this version may have problems with large files

What this does is split the text of the entire file on ">", which gives a list of strings (each containing the sequence with its name). Each of these strings is split at "\n" characters, which gives a list of lines. The 0^{th} line in this list is the name, and the rest of the lines are sequence. The funky looking join statement just merges all the sequence lines into one long string and gets its length.

A solution that will handle large files without running out of memory

```
import sys

filename = sys.argv[1]
myFile = open(filename, "r")

cur_name = ""
cur_len = 0
total_len = 0
for line in myFile:
    if (total_len > 0 and line.startswith(">")): # we reached a new fasta sequence
        print cur_name, cur_len  # write values for previous sequence
        cur_name = line.strip()  # record the name of the new sequence
        cur_len = 0              # reset cur_len
    elif line.startswith(">"):    # this is the first fasta name, record it
        cur_name = line.strip()
    else:
        cur_len += len(line.strip())
        total_len += cur_len
print cur_name, cur_len           # we need to write the last values

print "Total length", total_len

myFile.close()
```

One of the arts of programming is seeing how to write elegant loops that do complex things.

It takes time and practice.